MAT 331 1.0 Algorithms and Complexity

Dr. G.H.J. Lanel

Semester 2 - 2016

Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

What is Algorithm Analysis?

- **Algorithm analysis** is the analysis of resource usage of given algorithms
- Exponential resource usage is bad
- It is best to
  - make resource usage a polynomial
  - make the polynomial small as possible

What is Algorithm Analysis?

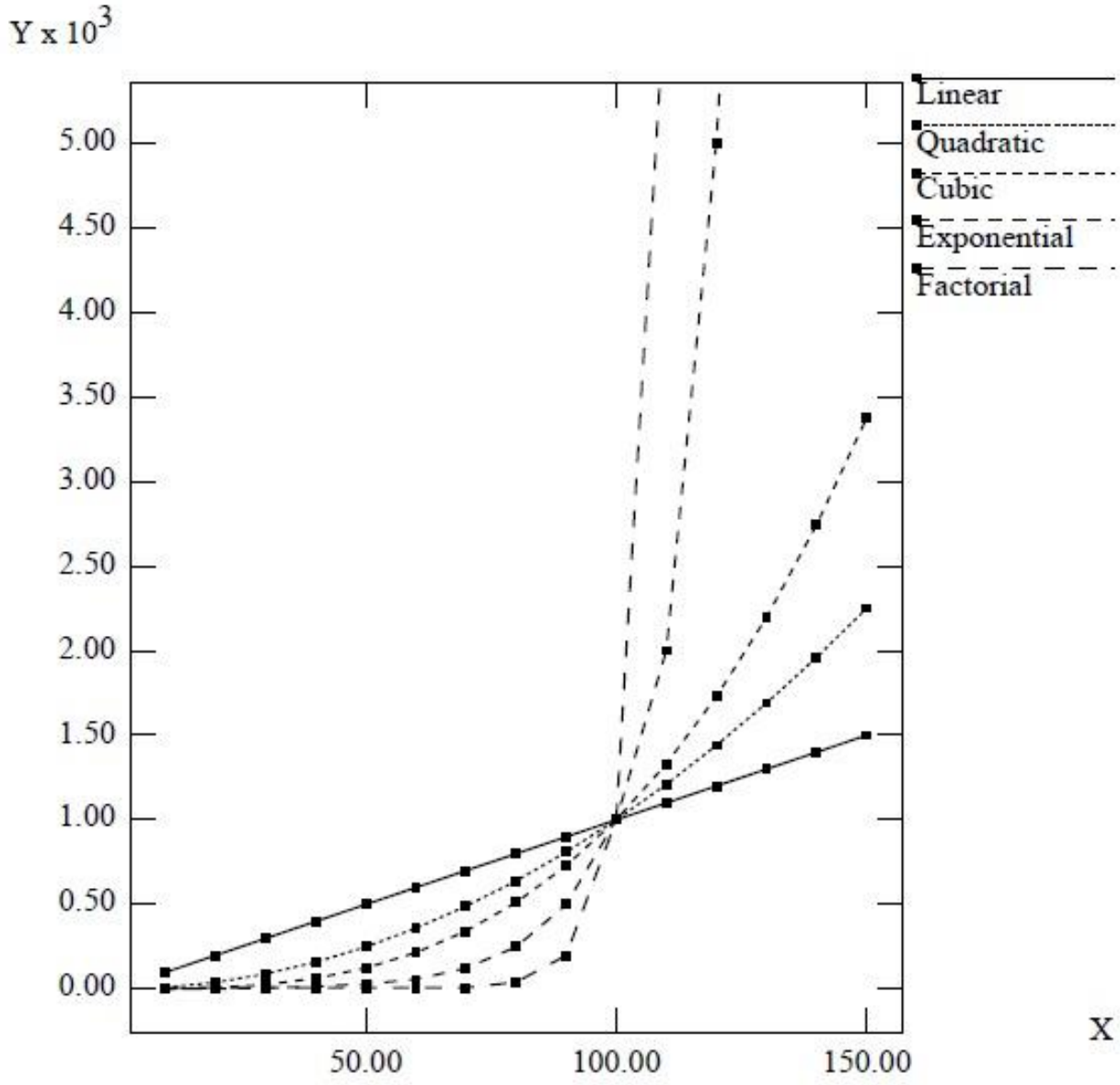
- **Algorithm analysis** is the analysis of resource usage of given algorithms
- Exponential resource usage is bad
- It is best to
  - make resource usage a polynomial
  - make the polynomial small as possible

What is Algorithm Analysis?

- **Algorithm analysis** is the analysis of resource usage of given algorithms
- Exponential resource usage is bad
- It is best to
  - make resource usage a polynomial
  - make the polynomial small as possible

What is Algorithm Analysis?

- **Algorithm analysis** is the analysis of resource usage of given algorithms
- Exponential resource usage is bad
- It is best to
  - make resource usage a polynomial
  - make the polynomial small as possible


Y x 10^3
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
50.00
100.00
150.00
X
Linear
Quadratic
Cubic
Exponential
Factorial

What is Computational Complexity Theory?

- **Computational Complexity Theory** is the study of the cost of solving problems, measure the amount of resource needed.
  - Time
  - Space
- Two aspects:
  - Upper bounds - give fast algorithms
  - Lower bounds - no algorithm is faster

What is Computational Complexity Theory?

- **Computational Complexity Theory** is the study of the cost of solving problems, measure the amount of resource needed.
  - Time
  - Space
- Two aspects:
  - Upper bounds - give fast algorithms
  - Lower bounds - no algorithm is faster

What is Computational Complexity Theory?

- **Computational Complexity Theory** is the study of the cost of solving problems, measure the amount of resource needed.
  - Time
  - Space
- Two aspects:
  - Upper bounds - give fast algorithms
  - Lower bounds - no algorithm is faster

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- ...

How large was his reward?

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?

The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?

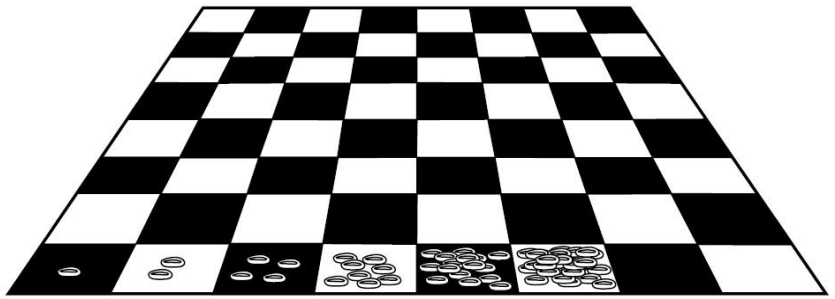
The Game of Chess

According to the legend, when Grand Vizier Sissa Ben Dahir invented chess, King Shirham of India was so taken with the game that he asked him to name his reward.

The Vizier asked for

- One grain of wheat on the first square of the chess board
- Two grains of wheat on the second square
- Four grains of wheat on the third square
- Eight grains of wheat on the fourth square
- etc

How large was his reward?



Solution

- Amount of wheat need to fill all squares of the chess board:

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1 = 1.8 \times 10^{19}$$

- A bushel of wheat contains 5×10^6 grains
- Therefore he asked for 3.6×10^{12} bushels
- The price of wheat is around \$2.50 per bushel
- Therefore he asked for $\$9 \times 10^{12} = \$90$ trillion at current prices
- **Cost for $n \times n$ chess board grows proportionally to 2^{n^2}**

Solution

- Amount of wheat need to fill all squares of the chess board:

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1 = 1.8 \times 10^{19}$$

- A bushel of wheat contains 5×10^6 grains
- Therefore he asked for 3.6×10^{12} bushels
- The price of wheat is around \$2.50 per bushel
- Therefore he asked for $\$9 \times 10^{12} = \$90$ trillion at current prices
- **Cost for $n \times n$ chess board grows proportionally to 2^{n^2}**

Solution

- Amount of wheat need to fill all squares of the chess board:

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1 = 1.8 \times 10^{19}$$

- A bushel of wheat contains 5×10^6 grains
- Therefore he asked for 3.6×10^{12} bushels
- The price of wheat is around \$2.50 per bushel
- Therefore he asked for $\$9 \times 10^{12} = \$90$ trillion at current prices
- **Cost for $n \times n$ chess board grows proportionally to 2^{n^2}**

Solution

- Amount of wheat need to fill all squares of the chess board:

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1 = 1.8 \times 10^{19}$$

- A bushel of wheat contains 5×10^6 grains
- Therefore he asked for 3.6×10^{12} bushels
- The price of wheat is around \$2.50 per bushel
- Therefore he asked for $\$9 \times 10^{12} = \$90$ trillion at current prices
- **Cost for $n \times n$ chess board grows proportionally to 2^{n^2}**

Solution

- Amount of wheat need to fill all squares of the chess board:

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1 = 1.8 \times 10^{19}$$

- A bushel of wheat contains 5×10^6 grains
- Therefore he asked for 3.6×10^{12} bushels
- The price of wheat is around \$2.50 per bushel
- Therefore he asked for $\$9 \times 10^{12} = \$90$ trillion at current prices
- Cost for $n \times n$ chess board grows proportionally to 2^{n^2}

Solution

- Amount of wheat need to fill all squares of the chess board:

$$\sum_{i=0}^{63} 2^i = 2^{64} - 1 = 1.8 \times 10^{19}$$

- A bushel of wheat contains 5×10^6 grains
- Therefore he asked for 3.6×10^{12} bushels
- The price of wheat is around \$2.50 per bushel
- Therefore he asked for $\$9 \times 10^{12} = \$90$ trillion at current prices
- **Cost for $n \times n$ chess board grows proportionally to 2^{n^2}**

Time Traveling Investor

- A time traveler invests \$1000 at 8% interest compounded annually. How much money does he/she have if he/she travels 100 years into future? 200 years? 1000 years?
- **Return for n years of time travel is proportional to 1000×1.08^n**

Time Traveling Investor

- A time traveler invests \$1000 at 8% interest compounded annually. How much money does he/she have if he/she travels 100 years into future? 200 years? 1000 years?

Years	*Amount*
100	$\$2.9 \times 10^6$
200	$\$4.8 \times 10^9$
300	$\$1.1 \times 10^{13}$
400	$\$2.3 \times 10^{16}$
500	$\$5.1 \times 10^{19}$
1000	$\$2.6 \times 10^{36}$

- **Return for n years of time travel is proportional to** 1000×1.08^n

Factors Influencing Program Efficiency

- Programming language
- Compiler
- Computer hardware
- Programmer's ability
- Programmer's effectiveness
- Algorithm

Factors Influencing Program Efficiency

- Programming language
- Compiler
- Computer hardware
- Programmer's ability
- Programmer's effectiveness
- Algorithm

Factors Influencing Program Efficiency

- Programming language
- Compiler
- Computer hardware
- Programmer's ability
- Programmer's effectiveness
- Algorithm

Factors Influencing Program Efficiency

- Programming language
- Compiler
- Computer hardware
- Programmer's ability
- Programmer's effectiveness
- Algorithm

Factors Influencing Program Efficiency

- Programming language
- Compiler
- Computer hardware
- Programmer's ability
- Programmer's effectiveness
- Algorithm

Factors Influencing Program Efficiency

- Programming language
- Compiler
- Computer hardware
- Programmer's ability
- Programmer's effectiveness
- Algorithm

Why study this subject?

- Efficient algorithms lead to efficient programs
- Efficient programs sell better
- Efficient programs make better use of hardware
- Programmers who write efficient algorithms are more marketable than those who don't

Why study this subject?

- Efficient algorithms lead to efficient programs
- Efficient programs sell better
- Efficient programs make better use of hardware
- Programmers who write efficient algorithms are more marketable than those who don't

Why study this subject?

- Efficient algorithms lead to efficient programs
- Efficient programs sell better
- Efficient programs make better use of hardware
- Programmers who write efficient algorithms are more marketable than those who don't

Why study this subject?

- Efficient algorithms lead to efficient programs
- Efficient programs sell better
- Efficient programs make better use of hardware
- Programmers who write efficient algorithms are more marketable than those who don't

Induction with people

Scenario

- Fact 1 : The first person is Greek
- Fact 2 : Pick any person in the line. If all the people up to that point are Greek, then the next person is Greek too

Question: Are they all Greek?

Induction with people

Scenario

- Fact 1 : The first person is Greek
- Fact 2 : Pick any person in the line. If all the people up to that point are Greek, then the next person is Greek too

Question: Are they all Greek?

Induction with people

Scenario

- Fact 1 : The first person is Greek
- Fact 2 : Pick any person in the line. If all the people up to that point are Greek, then the next person is Greek too

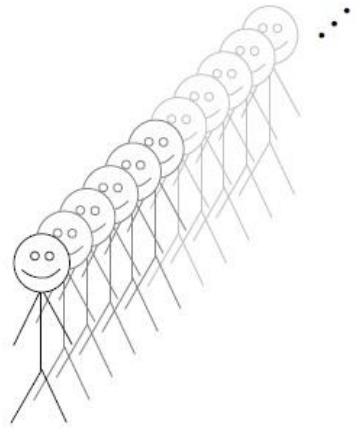
Question: Are they all Greek?

Induction with people

Scenario

- Fact 1 : The first person is Greek
- Fact 2 : Pick any person in the line. If all the people up to that point are Greek, then the next person is Greek too

Question: Are they all Greek?



Mathematical Induction

To prove that a property holds for all $n \in \mathbb{N}$ prove:

1. The property holds for $n = 1$
2. For all $n \geq 1$, if the property holds for all $1 \leq m \leq n$, then it holds for $n + 1$

Mathematical Induction

To prove that a property holds for all $n \in \mathbb{N}$ prove:

1. The property holds for $n = 1$
2. For all $n \geq 1$, if the property holds for all $1 \leq m \leq n$, then it holds for $n + 1$

Mathematical Induction

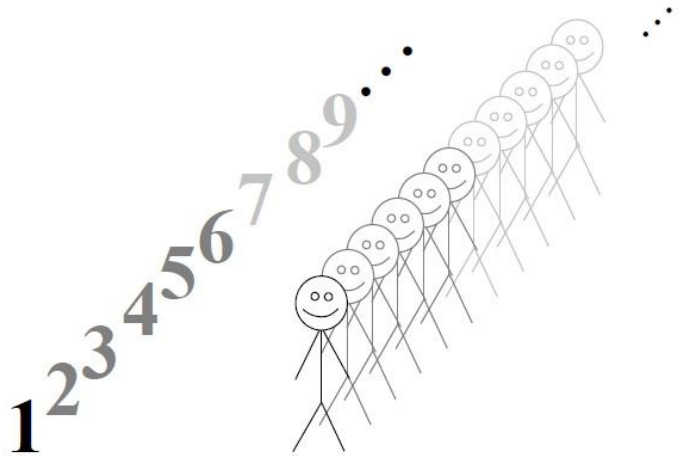
To prove that a property holds for all $n \in \mathbb{N}$ prove:

1. The property holds for $n = 1$
2. For all $n \geq 1$, if the property holds for all $1 \leq m \leq n$, then it holds for $n + 1$

Mathematical Induction

To prove that a property holds for all $n \in \mathbb{N}$ prove:

1. The property holds for $n = 1$
2. For all $n \geq 1$, if the property holds for all $1 \leq m \leq n$, then it holds for $n + 1$



Example 01

- **Claim**: For all $n \in \mathbb{N}$ if $1 + x > 0$, then

$$(1 + x)^n \geq 1 + nx$$

- **First**: Prove that property holds for $n = 1$. Both sides of the equation equal to $1 + x$
- **Second**: Prove that if the property holds for n, then the property holds for $n + 1$
- **Assume**: $(1 + x)^n \geq 1 + nx$, Required to prove

$$(1 + x)^{n+1} \geq 1 + (n + 1)x$$

Example 01

- **Claim**: For all $n \in \mathbb{N}$ if $1 + x > 0$, then

$$(1 + x)^n \geq 1 + nx$$

- **First**: Prove that property holds for $n = 1$. Both sides of the equation equal to $1 + x$
- **Second**: Prove that if the property holds for n, then the property holds for $n + 1$
- **Assume**: $(1 + x)^n \geq 1 + nx$, Required to prove

$$(1 + x)^{n+1} \geq 1 + (n + 1)x$$

Example 01

- **Claim**: For all $n \in \mathbb{N}$ if $1 + x > 0$, then

$$(1 + x)^n \geq 1 + nx$$

- **First**: Prove that property holds for $n = 1$. Both sides of the equation equal to $1 + x$
- **Second**: Prove that if the property holds for n, then the property holds for $n + 1$
- **Assume**: $(1 + x)^n \geq 1 + nx$, Required to prove

$$(1 + x)^{n+1} \geq 1 + (n + 1)x$$

Example 01

- **Claim**: For all $n \in \mathbb{N}$ if $1 + x > 0$, then

$$(1 + x)^n \geq 1 + nx$$

- **First**: Prove that property holds for $n = 1$. Both sides of the equation equal to $1 + x$
- **Second**: Prove that if the property holds for n, then the property holds for $n + 1$
- **Assume**: $(1 + x)^n \geq 1 + nx$, Required to prove

$$(1 + x)^{n+1} \geq 1 + (n + 1)x$$

Example ctd

Take

$$(1+x)^{n+1} = (1+x)(1+x)^n$$

$$\geq (1+x)(1+nx) \quad \text{(by induction hypothesis)}$$

$$= 1 + (n+1)x + nx^2$$

$$\geq 1 + (n+1)x \qquad (\text{since } nx^2 \geq 0)$$

Example ctd

Take

$$(1+x)^{n+1} = (1+x)(1+x)^n$$

$$\geq (1+x)(1+nx) \quad \text{(by induction hypothesis)}$$

$$= 1 + (n+1)x + nx^2$$

$$\geq 1 + (n+1)x \qquad (\text{since } nx^2 \geq 0)$$

Example ctd

Take

$$(1+x)^{n+1} = (1+x)(1+x)^n$$

$$\geq (1+x)(1+nx) \quad \text{(by induction hypothesis)}$$

$$= 1 + (n+1)x + nx^2$$

$$\geq 1 + (n+1)x \qquad (\text{since } nx^2 \geq 0)$$

Example ctd

Take

$$
\begin{aligned}
(1+x)^{n+1} &= (1+x)(1+x)^n \\
&\geq (1+x)(1+nx) \quad \text{(by induction hypothesis)} \\
&= 1+(n+1)x+nx^2 \\
&\geq 1+(n+1)x \qquad (\text{since } nx^2 \geq 0)
\end{aligned}
$$

Example ctd

Take

$$
\begin{aligned}
(1+x)^{n+1} &= (1+x)(1+x)^n \\
&\geq (1+x)(1+nx) \quad \text{(by induction hypothesis)} \\
&= 1+(n+1)x+nx^2 \\
&\geq 1+(n+1)x \qquad \text{(since } nx^2 \geq 0\text{)}
\end{aligned}
$$

Example 02

Find $S(n)$ where

$$S(n) = \sum_{i=1}^{n} (5i + 3)$$

This can be solved analytically, but it illustrates the concept.

Guess: $S(n) = an^2 + bn + c$ for some $a, b, c \in \mathbb{R}$

Base : $S(1) = 8$, hence guess is true provided $a + b + c = 8$

Inductive Step : Assume $S(n) = an^2 + bn + c$

Required to prove : $S(n+1) = a(n+1)^2 + b(n+1) + c$

Example 02

Find $S(n)$ where

$$S(n) = \sum_{i=1}^{n} (5i + 3)$$

This can be solved analytically, but it illustrates the concept.

Guess: $S(n) = an^2 + bn + c$ for some $a, b, c \in \mathbb{R}$

Base : $S(1) = 8$, hence guess is true provided $a + b + c = 8$

Inductive Step : Assume $S(n) = an^2 + bn + c$

Required to prove : $S(n+1) = a(n+1)^2 + b(n+1) + c$

Example 02

Find $S(n)$ where

$$S(n) = \sum_{i=1}^{n} (5i + 3)$$

This can be solved analytically, but it illustrates the concept.

Guess: $S(n) = an^2 + bn + c$ for some $a, b, c \in \mathbb{R}$

Base : $S(1) = 8$, hence guess is true provided $a + b + c = 8$

Inductive Step : Assume $S(n) = an^2 + bn + c$

Required to prove : $S(n+1) = a(n+1)^2 + b(n+1) + c$

Example 02

Find $S(n)$ where

$$S(n) = \sum_{i=1}^{n} (5i + 3)$$

This can be solved analytically, but it illustrates the concept.

Guess: $S(n) = an^2 + bn + c$ for some $a, b, c \in \mathbb{R}$

Base : $S(1) = 8$, hence guess is true provided $a + b + c = 8$

Inductive Step : Assume $S(n) = an^2 + bn + c$

Required to prove : $S(n+1) = a(n+1)^2 + b(n+1) + c$

Example 02

Find $S(n)$ where

$$S(n) = \sum_{i=1}^{n} (5i + 3)$$

This can be solved analytically, but it illustrates the concept.

Guess: $S(n) = an^2 + bn + c$ for some $a, b, c \in \mathbb{R}$

Base : $S(1) = 8$, hence guess is true provided $a + b + c = 8$

Inductive Step : Assume $S(n) = an^2 + bn + c$

Required to prove : $S(n+1) = a(n+1)^2 + b(n+1) + c$

Example 02

Find $S(n)$ where

$$S(n) = \sum_{i=1}^{n} (5i + 3)$$

This can be solved analytically, but it illustrates the concept.

Guess: $S(n) = an^2 + bn + c$ for some $a, b, c \in \mathbb{R}$

Base : $S(1) = 8$, hence guess is true provided $a + b + c = 8$

Inductive Step : Assume $S(n) = an^2 + bn + c$

Required to prove : $S(n+1) = a(n+1)^2 + b(n+1) + c$

Example ctd

Now,

$$\begin{aligned} S(n+1) &= S(n) + 5(n+1) + 3 \\ &= an^2 + bn + c + 5(n+1) + 3 \\ &= an^2 + (b+5)n + c + 8 \end{aligned}$$

We want

$$\begin{aligned} an^2 + (b+5)n + c + 8 &= a(n+1)^2 + b(n+1) + c \\ &= an^2 + (2a+b)n + (a+b+c) \end{aligned}$$

Each pair of coefficients has to be the same.

$$an^2 + (b+5)n + c + 8 = an^2 + (2a+b)n + (a+b+c)$$

Example ctd

Now,

$$\begin{aligned} S(n+1) &= S(n) + 5(n+1) + 3 \\ &= an^2 + bn + c + 5(n+1) + 3 \\ &= an^2 + (b+5)n + c + 8 \end{aligned}$$

We want

$$\begin{aligned} an^2 + (b+5)n + c + 8 &= a(n+1)^2 + b(n+1) + c \\ &= an^2 + (2a+b)n + (a+b+c) \end{aligned}$$

Each pair of coefficients has to be the same.

$$an^2 + (b+5)n + c + 8 = an^2 + (2a+b)n + (a+b+c)$$

Example ctd

Now,

$$\begin{aligned} S(n+1) &= S(n) + 5(n+1) + 3 \\ &= an^2 + bn + c + 5(n+1) + 3 \\ &= an^2 + (b+5)n + c + 8 \end{aligned}$$

We want

$$an^2 + (b+5)n + c + 8 = a(n+1)^2 + b(n+1) + c$$
$$= an^2 + (2a+b)n + (a+b+c)$$

Each pair of coefficients has to be the same.

$$an^2 + (b+5)n + c + 8 = an^2 + (2a+b)n + (a+b+c)$$

Example ctd

Now,

$$\begin{aligned} S(n+1) &= S(n) + 5(n+1) + 3 \\ &= an^2 + bn + c + 5(n+1) + 3 \\ &= an^2 + (b+5)n + c + 8 \end{aligned}$$

We want

$$\begin{aligned} an^2 + (b+5)n + c + 8 &= a(n+1)^2 + b(n+1) + c \\ &= an^2 + (2a+b)n + (a+b+c) \end{aligned}$$

Each pair of coefficients has to be the same.

$$an^2 + (b+5)n + c + 8 = an^2 + (2a+b)n + (a+b+c)$$

Example ctd

Now,

$$\begin{aligned} S(n+1) &= S(n) + 5(n+1) + 3 \\ &= an^2 + bn + c + 5(n+1) + 3 \\ &= an^2 + (b+5)n + c + 8 \end{aligned}$$

We want

$$\begin{aligned} an^2 + (b+5)n + c + 8 &= a(n+1)^2 + b(n+1) + c \\ &= an^2 + (2a+b)n + (a+b+c) \end{aligned}$$

Each pair of coefficients has to be the same.

$$an^2 + (b+5)n + c + 8 = an^2 + (2a+b)n + (a+b+c)$$

Example ctd

Now,

$$\begin{aligned} S(n+1) &= S(n) + 5(n+1) + 3 \\ &= an^2 + bn + c + 5(n+1) + 3 \\ &= an^2 + (b+5)n + c + 8 \end{aligned}$$

We want

$$\begin{aligned} an^2 + (b+5)n + c + 8 &= a(n+1)^2 + b(n+1) + c \\ &= an^2 + (2a+b)n + (a+b+c) \end{aligned}$$

Each pair of coefficients has to be the same.

$$an^2 + (b+5)n + c + 8 = an^2 + (2a+b)n + (a+b+c)$$

Example ctd

- The first coefficient tells us nothing
- The second coefficient tells us $b + 5 = 2a + b$, therefore $a = 2.5$
- We know $a + b + c = 8$(from the Base), therefore $c = 0$(looking at the third coefficient)
- Since we now know $a = 2.5$, $c = 0$ and $a + b + c = 8$, we can deduce that $b = 5.5$
- Therefore $S(n) = 2.5n^2 + 5.5n = n(5n + 11)/2$

Example ctd

- The first coefficient tells us nothing
- The second coefficient tells us $b + 5 = 2a + b$, therefore $a = 2.5$
- We know $a + b + c = 8$(from the Base), therefore $c = 0$(looking at the third coefficient)
- Since we now know $a = 2.5, c = 0$ and $a + b + c = 8$, we can deduce that $b = 5.5$
- Therefore $S(n) = 2.5n^2 + 5.5n = n(5n + 11)/2$

Example ctd

- The first coefficient tells us nothing
- The second coefficient tells us $b + 5 = 2a + b$, therefore $a = 2.5$
- We know $a + b + c = 8$(from the Base), therefore $c = 0$(looking at the third coefficient)
- Since we now know $a = 2.5, c = 0$ and $a + b + c = 8$, we can deduce that $b = 5.5$
- Therefore $S(n) = 2.5n^2 + 5.5n = n(5n + 11)/2$

Example ctd

- The first coefficient tells us nothing
- The second coefficient tells us $b + 5 = 2a + b$, therefore $a = 2.5$
- We know $a + b + c = 8$(from the Base), therefore $c = 0$(looking at the third coefficient)
- Since we now know $a = 2.5, c = 0$ and $a + b + c = 8$, we can deduce that $b = 5.5$
- Therefore $S(n) = 2.5n^2 + 5.5n = n(5n + 11)/2$

Example ctd

- The first coefficient tells us nothing
- The second coefficient tells us $b + 5 = 2a + b$, therefore $a = 2.5$
- We know $a + b + c = 8$(from the Base), therefore $c = 0$(looking at the third coefficient)
- Since we now know $a = 2.5, c = 0$ and $a + b + c = 8$, we can deduce that $b = 5.5$
- Therefore $S(n) = 2.5n^2 + 5.5n = n(5n + 11)/2$

Example 03

Prove that for all $n \geq 1$

$$\sum_{i=1}^{n} \frac{1}{2^i} < 1$$

- The claim is clearly true for $n = 1$
- Assume that the claim is true for n

$$\sum_{i=1}^{n+1} \frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{n+1}}$$

Example 03

Prove that for all $n \geq 1$

$$\sum_{i=1}^{n} \frac{1}{2^i} < 1$$

- The claim is clearly true for $n = 1$
- Assume that the claim is true for n

$$\sum_{i=1}^{n+1} \frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{n+1}}$$

Example 03

Prove that for all $n \geq 1$

$$\sum_{i=1}^{n} \frac{1}{2^i} < 1$$

- The claim is clearly true for $n = 1$
- Assume that the claim is true for n

$$\sum_{i=1}^{n+1} \frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{n+1}}$$

Example 03

Prove that for all $n \geq 1$

$$\sum_{i=1}^{n} \frac{1}{2^i} < 1$$

- The claim is clearly true for $n = 1$
- Assume that the claim is true for n

$$\sum_{i=1}^{n+1} \frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{n+1}}$$

Example 03

Prove that for all $n \geq 1$

$$\sum_{i=1}^{n} \frac{1}{2^i} < 1$$

- The claim is clearly true for $n = 1$
- Assume that the claim is true for n

$$\sum_{i=1}^{n+1} \frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{n+1}}$$

Example ctd

$$
\begin{aligned}
&= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \ldots + \frac{1}{2^n}\right) \\
&= \frac{1}{2} + \frac{1}{2}\sum_{i=1}^{n}\frac{1}{2^i} \\
&< \frac{1}{2} + \frac{1}{2}.1 \qquad \text{(by induction hypothesis)} \\
&= 1
\end{aligned}
$$

Example ctd

$$
\begin{aligned}
&= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \ldots + \frac{1}{2^n}\right) \\
&= \frac{1}{2} + \frac{1}{2}\sum_{i=1}^{n}\frac{1}{2^i} \\
&< \frac{1}{2} + \frac{1}{2}.1 \qquad \text{(by induction hypothesis)} \\
&= 1
\end{aligned}
$$

Example ctd

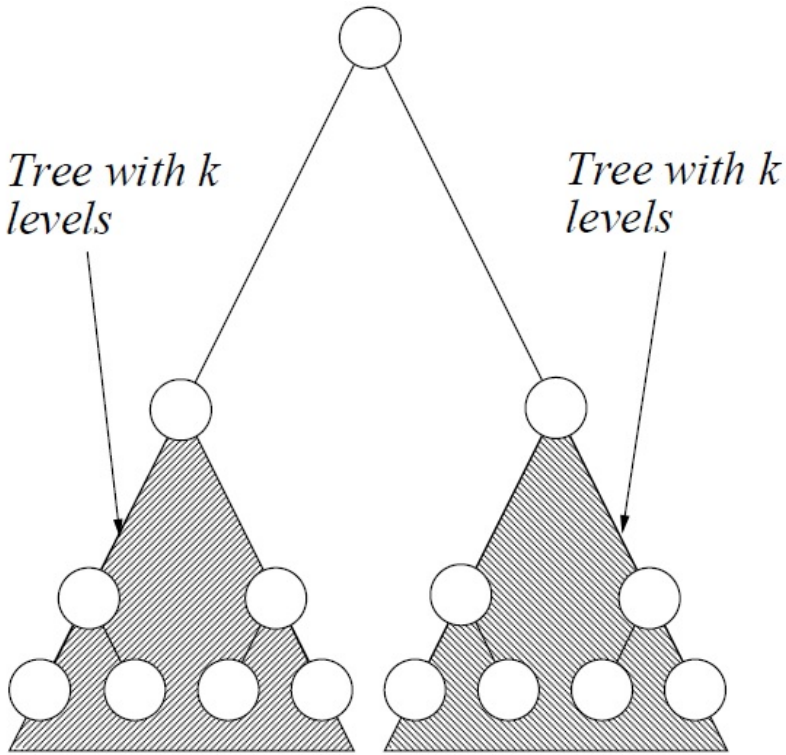
$$
\begin{aligned}
&= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \ldots + \frac{1}{2^n}\right) \\
&= \frac{1}{2} + \frac{1}{2}\sum_{i=1}^{n}\frac{1}{2^i} \\
&< \frac{1}{2} + \frac{1}{2}.1 \qquad \text{(by induction hypothesis)} \\
&= 1
\end{aligned}
$$

Example ctd

$$
\begin{aligned}
&= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \ldots + \frac{1}{2^n}\right) \\
&= \frac{1}{2} + \frac{1}{2}\sum_{i=1}^{n}\frac{1}{2^i} \\
&< \frac{1}{2} + \frac{1}{2}.1 \qquad \text{(by induction hypothesis)} \\
&= 1
\end{aligned}
$$

Complete Binary Trees

Claim: A complete binary tree with k levels has exactly $2^k - 1$ nodes



Proof :

- Proof by induction on number of levels. The claim is true for $k = 1$, since a complete binary tree with one level consists of a single node
- Suppose a complete binary tree with k levels has $2^k - 1$ nodes. We are required to prove that a complete binary tree with $k + 1$ levels has $2^{k+1} - 1$ nodes
- A complete binary tree with $k + 1$ levels consists of a root plus two trees with k levels. Therefore, by the induction hypothesis the total number of nodes is

$$1 + 2(2^k - 1) = 2^{k+1} - 1$$

Proof :

- Proof by induction on number of levels. The claim is true for $k = 1$, since a complete binary tree with one level consists of a single node

- Suppose a complete binary tree with k levels has $2^k - 1$ nodes. We are required to prove that a complete binary tree with $k + 1$ levels has $2^{k+1} - 1$ nodes

- A complete binary tree with $k + 1$ levels consists of a root plus two trees with k levels. Therefore, by the induction hypothesis the total number of nodes is

$$1 + 2(2^k - 1) = 2^{k+1} - 1$$

Proof :

- Proof by induction on number of levels. The claim is true for $k = 1$, since a complete binary tree with one level consists of a single node
- Suppose a complete binary tree with k levels has $2^k - 1$ nodes. We are required to prove that a complete binary tree with $k + 1$ levels has $2^{k+1} - 1$ nodes
- A complete binary tree with $k + 1$ levels consists of a root plus two trees with k levels. Therefore, by the induction hypothesis the total number of nodes is

$$1 + 2(2^k - 1) = 2^{k+1} - 1$$

Proof :

- Proof by induction on number of levels. The claim is true for $k = 1$, since a complete binary tree with one level consists of a single node
- Suppose a complete binary tree with k levels has $2^k - 1$ nodes. We are required to prove that a complete binary tree with $k + 1$ levels has $2^{k+1} - 1$ nodes
- A complete binary tree with $k + 1$ levels consists of a root plus two trees with k levels. Therefore, by the induction hypothesis the total number of nodes is

$$1 + 2(2^k - 1) = 2^{k+1} - 1$$

Proof :

- Proof by induction on number of levels. The claim is true for $k = 1$, since a complete binary tree with one level consists of a single node

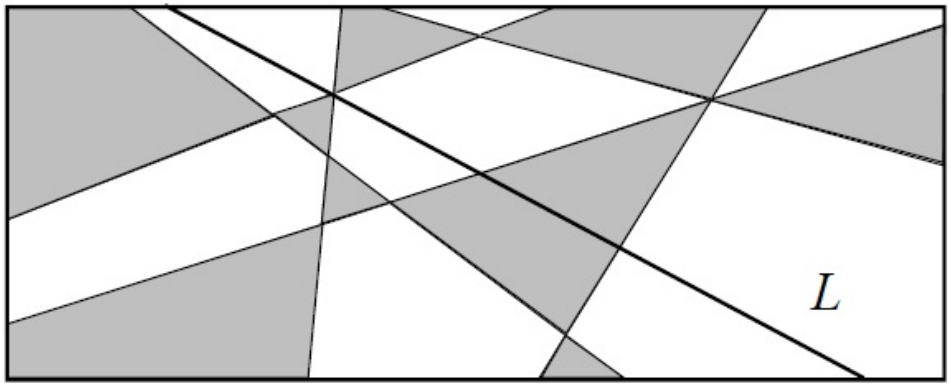
- Suppose a complete binary tree with k levels has $2^k - 1$ nodes. We are required to prove that a complete binary tree with $k + 1$ levels has $2^{k+1} - 1$ nodes

- A complete binary tree with $k + 1$ levels consists of a root plus two trees with k levels. Therefore, by the induction hypothesis the total number of nodes is

$$1 + 2(2^k - 1) = 2^{k+1} - 1$$

Geometric Example

Prove that any set of regions defined by n lines in the plane can be colored with only two colors so that no two regions that share an edge have the same color



Example ctd

- Proof by induction on n. True for $n = 1$(color one side light, the other side dark). Now suppose that the hypothesis is true for n lines
- Suppose we are given $n + 1$ lines in the plane. Remove one from the lines, say L, and color the remaining regions with two colors(which can be done by induction hypothesis). Replace L. Reverse all of the colors on one side of the line.

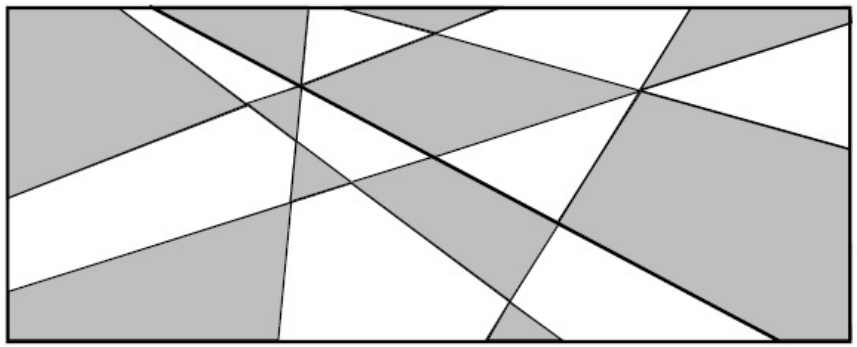
Example ctd

- Proof by induction on n. True for $n = 1$(color one side light, the other side dark). Now suppose that the hypothesis is true for n lines
- Suppose we are given $n + 1$ lines in the plane. Remove one from the lines, say L, and color the remaining regions with two colors(which can be done by induction hypothesis). Replace L. Reverse all of the colors on one side of the line.

Example ctd

- Proof by induction on n. True for $n = 1$(color one side light, the other side dark). Now suppose that the hypothesis is true for n lines

- Suppose we are given $n + 1$ lines in the plane. Remove one from the lines, say L, and color the remaining regions with two colors(which can be done by induction hypothesis). Replace L. Reverse all of the colors on one side of the line.



Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

Introduction

- When studying the complexity of an algorithm, significance consideration has to be given for the growth in the number of operations required by the algorithm as the size of the problem increases.
- In order to handle on its complexity, we first look for the function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n, to the values of some known function (polynomial, exponential function, log function, etc.)
- Growth of functions directly related to complexity of algorithms.
- For given functions f and g we wish to show how to quantify the statement

"g grows as fast as f".

Introduction

- When studying the complexity of an algorithm, significance consideration has to be given for the growth in the number of operations required by the algorithm as the size of the problem increases.
- In order to handle on its complexity, we first look for the function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n, to the values of some known function (polynomial, exponential function, log function, etc.)
- Growth of functions directly related to complexity of algorithms.
- For given functions f and g we wish to show how to quantify the statement

"g grows as fast as f".

Introduction

- When studying the complexity of an algorithm, significance consideration has to be given for the growth in the number of operations required by the algorithm as the size of the problem increases.
- In order to handle on its complexity, we first look for the function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n, to the values of some known function (polynomial, exponential function, log function, etc.)
- Growth of functions directly related to complexity of algorithms.
- For given functions f and g we wish to show how to quantify the statement

"g grows as fast as f".

Introduction

- When studying the complexity of an algorithm, significance consideration has to be given for the growth in the number of operations required by the algorithm as the size of the problem increases.
- In order to handle on its complexity, we first look for the function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n, to the values of some known function (polynomial, exponential function, log function, etc.)
- Growth of functions directly related to complexity of algorithms.
- For given functions f and g we wish to show how to quantify the statement

"g grows as fast as f".

Big-O

- The most basic concept concerning the growth functions is **Big-O**.
- The statement that f is big-O of g express the fact that for large enough x, f will be bounded above by some constant multiple of g.

Definition

Let f and g be functions from the natural numbers to the real numbers. Then ***g* asymptotically dominates *f*** or ***f* is big-O of *g***, if there exists a positive constants c and k s.t.

$$|f(x)| \leq c\,|g(x)|\,, \quad \text{for } \; x \geq k$$

If f is big-O of g then we write $f(x)$ is $O(g(x))$ or $f \in O(g)$

Big-O

- The most basic concept concerning the growth functions is **Big-O**.
- The statement that f is big-O of g express the fact that for large enough x, f will be bounded above by some constant multiple of g.

Definition

Let f and g be functions from the natural numbers to the real numbers. Then ***g* asymptotically dominates *f*** or ***f* is big-O of *g***, if there exists a positive constants c and k s.t.

$$|f(x)| \leq c\,|g(x)|\,, \quad \text{for } \; x \geq k$$

If f is big-O of g then we write $f(x)$ is $O(g(x))$ or $f \in O(g)$

Big-O

- The most basic concept concerning the growth functions is **Big-O**.
- The statement that f is big-O of g express the fact that for large enough x, f will be bounded above by some constant multiple of g.

Definition

Let f and g be functions from the natural numbers to the real numbers. Then ***g* asymptotically dominates *f*** or ***f* is big-O of *g***, if there exists a positive constants c and k s.t.

$$|f(x)| \leq c\,|g(x)|\,, \quad \text{for } \; x \geq k$$

If f is big-O of g then we write $f(x)$ is $O(g(x))$ or $f \in O(g)$

Following theorems gives a necessary condition for f to be big-O of g in terms of limits.

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, *where* $L \in \mathbb{R}$ *and* $L \geq 0$ *then* $f \in O(g)$

Proof:

Suppose $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, where $L \in \mathbb{R}$ and $L \geq 0$.

Then by the definition of limit, we can find $\frac{|f(x)|}{|g(x)|}$ as close to L by choosing x large enough.

Following theorems gives a necessary condition for f to be big-O of g in terms of limits.

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, *where* $L \in \mathbb{R}$ *and* $L \geq 0$ *then* $f \in O(g)$

Proof:

Suppose $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, where $L \in \mathbb{R}$ and $L \geq 0$.

Then by the definition of limit, we can find $\frac{|f(x)|}{|g(x)|}$ as close to L by choosing x large enough.

Following theorems gives a necessary condition for f to be big-O of g in terms of limits.

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, *where* $L \in \mathbb{R}$ *and* $L \geq 0$ *then* $f \in O(g)$

Proof:

Suppose $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, where $L \in \mathbb{R}$ and $L \geq 0$.

Then by the definition of limit, we can find $\frac{|f(x)|}{|g(x)|}$ as close to L by choosing x large enough.

Following theorems gives a necessary condition for f to be big-O of g in terms of limits.

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, *where* $L \in \mathbb{R}$ *and* $L \geq 0$ *then* $f \in O(g)$

Proof:

Suppose $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, where $L \in \mathbb{R}$ and $L \geq 0$.

Then by the definition of limit, we can find $\frac{|f(x)|}{|g(x)|}$ as close to L by choosing x large enough.

Following theorems gives a necessary condition for f to be big-O of g in terms of limits.

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, *where* $L \in \mathbb{R}$ *and* $L \geq 0$ *then* $f \in O(g)$

Proof:

Suppose $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = L$, where $L \in \mathbb{R}$ and $L \geq 0$.

Then by the definition of limit, we can find $\frac{|f(x)|}{|g(x)|}$ as close to L by choosing x large enough.

Proof Contd...

In particular, we can ensure that $\frac{|f(x)|}{|g(x)|}$ is within a distance 1 of L by choosing $x \geq k$ for some $k > 0$, (i.e. $\exists\ k(>0)$ s.t if $x \geq k$), then

$$\left| \frac{|f(x)|}{|g(x)|} - L \right| \leq 1$$

$$\text{In particular} \quad \frac{|f(x)|}{|g(x)|} - L \leq 1$$

$$\frac{|f(x)|}{|g(x)|} \leq L + 1$$

$$\text{Therefore} \quad |f(x)| \leq (L+1)g(x)$$

So we can choose $c = L + 1$, thus $f \in O(g)$

Proof Contd...

In particular, we can ensure that $\frac{|f(x)|}{|g(x)|}$ is within a distance 1 of L by choosing $x \geq k$ for some $k > 0$, (i.e. $\exists\ k(>0)$ s.t if $x \geq k$), then

$$\left| \frac{|f(x)|}{|g(x)|} - L \right| \leq 1$$

In particular $\frac{|f(x)|}{|g(x)|} - L \leq 1$

$$\frac{|f(x)|}{|g(x)|} \leq L + 1$$

Therefore $|f(x)| \leq (L+1)g(x)$

So we can choose $c = L + 1$, thus $f \in O(g)$

Proof Contd...

In particular, we can ensure that $\frac{|f(x)|}{|g(x)|}$ is within a distance 1 of L by choosing $x \geq k$ for some $k > 0$, (i.e. $\exists\ k(>0)$ s.t if $x \geq k$), then

$$\left| \frac{|f(x)|}{|g(x)|} - L \right| \leq 1$$

$$\text{In particular} \quad \frac{|f(x)|}{|g(x)|} - L \leq 1$$

$$\frac{|f(x)|}{|g(x)|} \leq L + 1$$

$$\text{Therefore} \quad |f(x)| \leq (L+1)g(x)$$

So we can choose $c = L + 1$, thus $f \in O(g)$

Proof Contd...

In particular, we can ensure that $\frac{|f(x)|}{|g(x)|}$ is within a distance 1 of L by choosing $x \geq k$ for some $k > 0$, (i.e. $\exists\ k(>0)$ s.t if $x \geq k$), then

$$\left| \frac{|f(x)|}{|g(x)|} - L \right| \leq 1$$

$$\text{In particular} \quad \frac{|f(x)|}{|g(x)|} - L \leq 1$$

$$\frac{|f(x)|}{|g(x)|} \leq L + 1$$

$$\text{Therefore} \quad |f(x)| \leq (L+1)g(x)$$

So we can choose $c = L + 1$, thus $f \in O(g)$

Proof Contd...

In particular, we can ensure that $\frac{|f(x)|}{|g(x)|}$ is within a distance 1 of L by choosing $x \geq k$ for some $k > 0$, (i.e. $\exists\ k(>0)$ s.t if $x \geq k$), then

$$\left| \frac{|f(x)|}{|g(x)|} - L \right| \leq 1$$

$$\text{In particular} \quad \frac{|f(x)|}{|g(x)|} - L \leq 1$$

$$\frac{|f(x)|}{|g(x)|} \leq L + 1$$

$$\text{Therefore} \quad |f(x)| \leq (L+1)g(x)$$

So we can choose $c = L + 1$, thus $f \in O(g)$

Proof Contd...

In particular, we can ensure that $\frac{|f(x)|}{|g(x)|}$ is within a distance 1 of L by choosing $x \geq k$ for some $k > 0$, (i.e. $\exists\ k(>0)$ s.t if $x \geq k$), then

$$\left| \frac{|f(x)|}{|g(x)|} - L \right| \leq 1$$

$$\text{In particular} \quad \frac{|f(x)|}{|g(x)|} - L \leq 1$$

$$\frac{|f(x)|}{|g(x)|} \leq L + 1$$

$$\text{Therefore} \quad |f(x)| \leq (L+1)g(x)$$

So we can choose $c = L + 1$, thus $f \in O(g)$

E.g. 01 Show that $x^2 + 10$ *is* $O(x^2)$

Sol: Taking the limit

$$\lim_{x\to\infty} \frac{|x^2+10|}{|x^2|} = \lim_{x\to\infty} \left|\frac{x^2+10}{x^2}\right|$$

$$= \lim_{x\to\infty} \left|1 + \frac{10}{x^2}\right|$$

$$= 1 > 0$$

Therefore by theorem, $x^2 + 10 \in O(x^2)$

E.g. 01 Show that $x^2 + 10$ *is* $O(x^2)$

Sol: Taking the limit

$$\lim_{x\to\infty} \frac{|x^2+10|}{|x^2|} = \lim_{x\to\infty} \left|\frac{x^2+10}{x^2}\right|$$

$$= \lim_{x\to\infty} \left|1 + \frac{10}{x^2}\right|$$

$$= 1 > 0$$

Therefore by theorem, $x^2 + 10 \in O(x^2)$

E.g. 01 Show that $x^2 + 10$ *is* $O(x^2)$

Sol: Taking the limit

$$\begin{aligned}\lim_{x\to\infty} \frac{|x^2+10|}{|x^2|} &= \lim_{x\to\infty} \left|\frac{x^2+10}{x^2}\right| \\ &= \lim_{x\to\infty} \left|1+\frac{10}{x^2}\right| \\ &= 1 > 0\end{aligned}$$

Therefore by theorem, $x^2 + 10 \in O(x^2)$

E.g. 01 Show that $x^2 + 10$ *is* $O(x^2)$

Sol: Taking the limit

$$\begin{aligned}\lim_{x\to\infty} \frac{|x^2+10|}{|x^2|} &= \lim_{x\to\infty} \left|\frac{x^2+10}{x^2}\right| \\ &= \lim_{x\to\infty} \left|1+\frac{10}{x^2}\right| \\ &= 1 > 0\end{aligned}$$

Therefore by theorem, $x^2 + 10 \in O(x^2)$

E.g. 01 Show that $x^2 + 10$ *is* $O(x^2)$

Sol: Taking the limit

$$\begin{aligned}\lim_{x\to\infty} \frac{|x^2+10|}{|x^2|} &= \lim_{x\to\infty} \left|\frac{x^2+10}{x^2}\right| \\ &= \lim_{x\to\infty} \left|1+\frac{10}{x^2}\right| \\ &= 1 > 0\end{aligned}$$

Therefore by theorem, $x^2 + 10 \in O(x^2)$

E.g. 01 Show that $x^2 + 10$ *is* $O(x^2)$

Sol: Taking the limit

$$\begin{aligned}\lim_{x\to\infty} \frac{|x^2+10|}{|x^2|} &= \lim_{x\to\infty} \left|\frac{x^2+10}{x^2}\right| \\ &= \lim_{x\to\infty} \left|1+\frac{10}{x^2}\right| \\ &= 1 > 0\end{aligned}$$

Therefore by theorem, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\left|3x^2\right| = \left|x^2 + 2x^2\right| \geq \left|x^2 + 2 \times 3^2\right|$$
$$\geq \left|x^2 + 10\right|$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 02 Show that $x^2 + 10$ is $O(x^2)$ using the definition of $Big - O$

Sol:

Let $c = 3$ and $k = 3$, Then if $x \geq 3$

$$\begin{aligned} \left|3x^2\right| = \left|x^2 + 2x^2\right| &\geq \left|x^2 + 2 \times 3^2\right| \\ &\geq \left|x^2 + 10\right| \end{aligned}$$

Then, $\left|x^2 + 10\right| \leq 3\left|x^2\right|$ for $x \geq 3$

Hence, $x^2 + 10 \in O(x^2)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} = \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right|$$

$$= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right|$$

$$= |log_a b|$$

$$> 0$$

Therefore by theorem, $log_a x \in O(log_b x)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} = \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right|$$

$$= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right|$$

$$= |log_a b|$$

$$> 0$$

Therefore by theorem, $log_a x \in O(log_b x)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\begin{aligned}
\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} &= \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right| \\
&= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right| \\
&= |log_a b| \\
&> 0
\end{aligned}$$

Therefore by theorem, $log_a x \in O(log_b x)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\begin{aligned}
\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} &= \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right| \\
&= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right| \\
&= |log_a b| \\
&> 0
\end{aligned}$$

Therefore by theorem, $log_a x \in O(log_b x)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\begin{aligned}
\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} &= \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right| \\
&= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right| \\
&= |log_a b| \\
&> 0
\end{aligned}$$

Therefore by theorem, $log_a x \in O(log_b x)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\begin{aligned}
\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} &= \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right| \\
&= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right| \\
&= |log_a b| \\
&> 0
\end{aligned}$$

Therefore by theorem, $log_a x \in O(log_b x)$

E.g. 03 Let $a, b \in \mathbb{R}^+ - \{1\}$, Prove $log_a x$ is $O(log_b x)$

Sol: Taking the limit

$$\begin{aligned}
\lim_{x\to\infty} \frac{|log_a x|}{|log_b x|} &= \lim_{x\to\infty} \left| \frac{log_a b \times log_b x}{log_b x} \right| \\
&= |log_a b| \lim_{x\to\infty} \left| \frac{log_b x}{log_b x} \right| \\
&= |log_a b| \\
&> 0
\end{aligned}$$

Therefore by theorem, $log_a x \in O(log_b x)$

How do you interpret the statement $f \notin O(g)$?

That is how you negate the definition.

$f \in O(g)$ iff there exist constants c and k such that for all $x \geq k$ then $|f(x)| \leq c|g(x)|$

The negation would the be $f \notin O(g)$ iff for all constants c and k, $\exists\ x$ s.t. $x \geq k$ and $|f(x)| > c|g(x)|$

How do you interpret the statement $f \notin O(g)$?

That is how you negate the definition.

$f \in O(g)$ iff there exist constants c and k such that for all $x \geq k$ then $|f(x)| \leq c|g(x)|$

The negation would the be $f \notin O(g)$ iff for all constants c and k, $\exists\ x$ s.t. $x \geq k$ and $|f(x)| > c|g(x)|$

How do you interpret the statement $f \notin O(g)$?

That is how you negate the definition.

$f \in O(g)$ iff there exist constants c and k such that for all $x \geq k$ then $|f(x)| \leq c|g(x)|$

The negation would the be $f \notin O(g)$ iff for all constants c and k, $\exists\ x$ s.t. $x \geq k$ and $|f(x)| > c|g(x)|$

How do you interpret the statement $f \notin O(g)$?

That is how you negate the definition.

$f \in O(g)$ iff there exist constants c and k such that for all $x \geq k$ then $|f(x)| \leq c|g(x)|$

The negation would the be $f \notin O(g)$ iff for all constants c and k, $\exists\ x$ s.t. $x \geq k$ and $|f(x)| > c|g(x)|$

How do you interpret the statement $f \notin O(g)$?

That is how you negate the definition.

$f \in O(g)$ iff there exist constants c and k such that for all $x \geq k$ then $|f(x)| \leq c|g(x)|$

The negation would the be $f \notin O(g)$ iff for all constants c and k, $\exists\ x$ s.t. $x \geq k$ and $|f(x)| > c|g(x)|$

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$, *then* f *is* ***not*** $O(g)$ $\quad (f \notin O(g))$

Proof: Suppose

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$$

Then for every $c > 0, \exists\ N\ (> 0)$ $s.t.$ $\frac{|f(x)|}{|g(x)|} > c$, if $x \geq N$

Thus, for all $c, k \geq 0$, $\exists\ x\ (\geq k)$ (take x greater than the larger of k and N) s.t.

$$\frac{|f(x)|}{|g(x)|} > c$$

$$|f(x)| > c|g(x)|$$

Thus $f \notin O(g)$

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$, *then* f *is* ***not*** $O(g)$ $\quad (f \notin O(g))$

Proof: Suppose

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$$

Then for every $c > 0, \exists\ N\ (> 0)$ *s.t.* $\frac{|f(x)|}{|g(x)|} > c$, if $x \geq N$

Thus, for all $c, k \geq 0$, $\exists\ x\ (\geq k)$ (take x greater than the larger of k and N) s.t.

$$\frac{|f(x)|}{|g(x)|} > c$$

$$|f(x)| > c|g(x)|$$

Thus $f \notin O(g)$

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$, *then* f *is* ***not*** $O(g)$ $\quad (f \notin O(g))$

Proof: Suppose

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$$

Then for every $c > 0, \exists\ N\ (> 0)$ *s.t.* $\frac{|f(x)|}{|g(x)|} > c$, if $x \geq N$

Thus, for all $c, k \geq 0$, $\exists\ x\ (\geq k)$ (take x greater than the larger of k and N) s.t.

$$\frac{|f(x)|}{|g(x)|} > c$$

$$|f(x)| > c|g(x)|$$

Thus $f \notin O(g)$

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$, *then* f *is* ***not*** $O(g)$ $\quad (f \notin O(g))$

Proof: Suppose

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$$

Then for every $c > 0, \exists\ N\ (> 0)$ *s.t.* $\frac{|f(x)|}{|g(x)|} > c$, if $x \geq N$

Thus, for all $c, k \geq 0,\ \exists\ x\ (\geq k)$ (take x greater than the larger of k and N) s.t.

$$\frac{|f(x)|}{|g(x)|} > c$$

$$|f(x)| > c|g(x)|$$

Thus $f \notin O(g)$

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$, *then* f *is* ***not*** $O(g)$ $\quad (f \notin O(g))$

Proof: Suppose

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$$

Then for every $c > 0, \exists\ N\ (> 0)$ *s.t.* $\frac{|f(x)|}{|g(x)|} > c$, if $x \geq N$

Thus, for all $c, k \geq 0,\ \exists\ x\ (\geq k)$ (take x greater than the larger of k and N) s.t.

$$\frac{|f(x)|}{|g(x)|} > c$$

$$|f(x)| > c|g(x)|$$

Thus $f \notin O(g)$

Theorem

If $\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$, *then* f *is* ***not*** $O(g)$ $\quad (f \notin O(g))$

Proof: Suppose

$$\lim_{x\to\infty} \frac{|f(x)|}{|g(x)|} = \infty$$

Then for every $c > 0, \exists\ N\ (> 0)$ *s.t.* $\frac{|f(x)|}{|g(x)|} > c$, if $x \geq N$

Thus, for all $c, k \geq 0,\ \exists\ x\ (\geq k)$ (take x greater than the larger of k and N) s.t.

$$\frac{|f(x)|}{|g(x)|} > c$$

$$|f(x)| > c|g(x)|$$

Thus $f \notin O(g)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

E.g. 01 Show that x^2 is not O(x).

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^2}{x}\right| = \lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^2 \notin O(x)$

E.g. 02 Show that $x^5 \notin O(100x^4)$.

Sol: Take the limit

$$\lim_{x\to\infty}\left|\frac{x^5}{100x^4}\right| = \frac{1}{100}\lim_{x\to\infty}|x| = \infty$$

Thus by Theorem, $x^5 \notin O(100x^4)$

Little-o

Definition

If f and g are such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

then we say that ***f* is little-o of *g***, written as $f \in o(g)$

L'Hôpital's Rule would be more useful for some calculations,

L'Hôpital's Rule

If $\lim_{x\to\infty} f(x) = \infty$ and $\lim_{x\to\infty} g(x) = \infty$,

and if $\lim_{x\to\infty} \frac{f'(x)}{g'(x)} = L$ then $\lim_{x\to\infty} \frac{f(x)}{g(x)} = L$

Little-o

Definition

If f and g are such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

then we say that ***f* is little-o of *g***, written as $f \in o(g)$

L'Hôpital's Rule would be more useful for some calculations,

L'Hôpital's Rule

$$\text{If } \lim_{x\to\infty} f(x) = \infty \text{ and } \lim_{x\to\infty} g(x) = \infty,$$

$$\text{and if } \lim_{x\to\infty} \frac{f'(x)}{g'(x)} = L \text{ then } \lim_{x\to\infty} \frac{f(x)}{g(x)} = L$$

Little-o

Definition

If f and g are such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

then we say that ***f* is little-o of *g***, written as $f \in o(g)$

L'Hôpital's Rule would be more useful for some calculations,

L'Hôpital's Rule

$$\text{If } \lim_{x\to\infty} f(x) = \infty \text{ and } \lim_{x\to\infty} g(x) = \infty,$$

$$\text{and if } \lim_{x\to\infty} \frac{f'(x)}{g'(x)} = L \text{ then } \lim_{x\to\infty} \frac{f(x)}{g(x)} = L$$

Little-o

Definition

If f and g are such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

then we say that ***f* is little-o of *g***, written as $f \in o(g)$

L'Hôpital's Rule would be more useful for some calculations,

L'Hôpital's Rule

$$\text{If } \lim_{x\to\infty} f(x) = \infty \text{ and } \lim_{x\to\infty} g(x) = \infty,$$

$$\text{and if } \lim_{x\to\infty} \frac{f'(x)}{g'(x)} = L \text{ then } \lim_{x\to\infty} \frac{f(x)}{g(x)} = L$$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

Theorem

If f is $o(g)$, then f is $O(g)$

E.g. 01 Show that $log_a x \in O(x)$, where a is a positive number different from 1.

Sol: Observe that $\lim_{x\to\infty} log_a x = \infty$ and $\lim_{x\to\infty} x = \infty$, therefore

$$\lim_{x\to\infty} \frac{log_a x}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log_a x)}{\frac{d}{dx}(x)} \quad \text{(L'Hôpital's rule)}$$

$$= \lim_{x\to\infty} \frac{1}{x ln(a)} = 0$$

Therefore $log_a x \in o(x)$, and by the Theorem $log_a x \in O(x)$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\lim_{x\to\infty} \frac{x^n}{a^x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)}$$

$$= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)}$$

$$= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\lim_{x\to\infty} \frac{x^n}{a^x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)}$$

$$= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)}$$

$$= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)} \\
&= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)} \\
&= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}
\end{aligned}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)} \\
&= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)} \\
&= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}
\end{aligned}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)} \\
&= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)} \\
&= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}
\end{aligned}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\begin{aligned}\lim_{x\to\infty} \frac{x^n}{a^x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)} \\ &= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)} \\ &= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}\end{aligned}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

E.g. 02 Show that if $a > 1$ then $x^n \in O(a^x)$

Sol: Observe that $\lim_{x\to\infty} x^n = \infty$ and $\lim_{x\to\infty} a^x = \infty$, then applying L'Hôpital's rule

$$\begin{aligned}\lim_{x\to\infty} \frac{x^n}{a^x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(x^n)}{\frac{d}{dx}(a^x)} \\ &= \lim_{x\to\infty} \frac{nx^{n-1}}{a^x ln(a)} \\ &= \frac{n}{ln(a)} \lim_{x\to\infty} \frac{x^{n-1}}{a^x}\end{aligned}$$

Apply the same rule again

$$= \frac{n(n-1)}{ln(a)^2} \lim_{x\to\infty} \frac{x^{n-2}}{a^x}$$

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x} \\
&= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x} \\
&= \frac{n!}{ln(a)^n} \lim_{x\to\infty} \frac{1}{a^x} \\
&= 0
\end{aligned}$$

Then $x^n \in o(a^x)$, Therefore by the theorem $x^n \in O(a^x)$

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x} \\
&= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x} \\
&= \frac{n!}{ln(a)^n} \lim_{x\to\infty} \frac{1}{a^x} \\
&= 0
\end{aligned}$$

Then $x^n \in o(a^x)$, Therefore by the theorem $x^n \in O(a^x)$

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x} \\
&= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x} \\
&= \frac{n!}{ln(a)^n} \lim_{x\to\infty} \frac{1}{a^x} \\
&= 0
\end{aligned}$$

Then $x^n \in o(a^x)$, Therefore by the theorem $x^n \in O(a^x)$

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x} \\
&= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x} \\
&= \frac{n!}{ln(a)^n} \lim_{x\to\infty} \frac{1}{a^x} \\
&= 0
\end{aligned}$$

Then $x^n \in o(a^x)$, Therefore by the theorem $x^n \in O(a^x)$

$$\begin{aligned}
\lim_{x\to\infty} \frac{x^n}{a^x} &= \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x} \\
&= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x} \\
&= \frac{n!}{ln(a)^n} \lim_{x\to\infty} \frac{1}{a^x} \\
&= 0
\end{aligned}$$

Then $x^n \in o(a^x)$, Therefore by the theorem $x^n \in O(a^x)$

Basic Properties of Big-O

The following theorems and facts will be helpful in determining big-O

Theorem

A polynomial of degree n is $O(x^n)$

Proof : (Exercise)

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(max\{|g_1|, |g_2|\})$

Proof : If $f_1 \in O(g_1)$, then $|f_1(x)| \leq c_1 |g_1(x)|$ for $c_1 > 0$ and $x \geq k_1$

If $f_2 \in O(g_2)$, then $|f_2(x)| \leq c_2 |g_2(x)|$ for $c_2 > 0$ and $x \geq k_2$

Basic Properties of Big-O

The following theorems and facts will be helpful in determining big-O

Theorem

A polynomial of degree n is $O(x^n)$

Proof : (Exercise)

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(max\{|g_1|, |g_2|\})$

Proof : If $f_1 \in O(g_1)$, then $|f_1(x)| \leq c_1 |g_1(x)|$ for $c_1 > 0$ and $x \geq k_1$

If $f_2 \in O(g_2)$, then $|f_2(x)| \leq c_2 |g_2(x)|$ for $c_2 > 0$ and $x \geq k_2$

Basic Properties of Big-O

The following theorems and facts will be helpful in determining big-O

Theorem

A polynomial of degree n is $O(x^n)$

Proof : (Exercise)

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(max\{|g_1|, |g_2|\})$

Proof : If $f_1 \in O(g_1)$, then $|f_1(x)| \leq c_1 |g_1(x)|$ for $c_1 > 0$ and $x \geq k_1$

If $f_2 \in O(g_2)$, then $|f_2(x)| \leq c_2 |g_2(x)|$ for $c_2 > 0$ and $x \geq k_2$

Basic Properties of Big-O

The following theorems and facts will be helpful in determining big-O

Theorem

A polynomial of degree n is $O(x^n)$

Proof : (Exercise)

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(max\{|g_1|, |g_2|\})$

Proof : If $f_1 \in O(g_1)$, then $|f_1(x)| \leq c_1 |g_1(x)|$ for $c_1 > 0$ and $x \geq k_1$

If $f_2 \in O(g_2)$, then $|f_2(x)| \leq c_2 |g_2(x)|$ for $c_2 > 0$ and $x \geq k_2$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Taking $h(x) = max(|g_1(x)|, |g_2(x)|)$ for $x \geq max(k_1, k_2)$

Then, $|f_1(x)| \leq c_1 h(x)$ and $|f_2(x)| \leq c_2 h(x)$

Therefore, $|f_1(x)| + |f_2(x)| \leq (c_1 + c_2)h(x)$

By triangular inequality, $|f_1(x) + f_2(x)| \leq |f_1(x)| + |f_2(x)|$

Then, $|f_1(x) + f_2(x)| \leq (c_1 + c_2)h(x)$

Then, $|f_1(x) + f_2(x)| \leq ch(x)$ where $c = c_1 + c_2$

Therefore, $f_1 + f_2 \in O(max|g_1|, |g_2|)$

Corollary

If f_1 and f_2 are both $O(g)$ then $(f_1 + f_2)$ is $O(g)$

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$

Proof : (Exercise)

Theorem

If f_1 is $O(f_2)$ and f_2 is $O(f_3)$, then f_1 is $O(f_3)$

Proof : (Exercise)

Theorem

If f is $O(g)$, then (af) is $O(g)$ for any constant a

Proof : (Exercise)

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$

Proof : (Exercise)

Theorem

If f_1 is $O(f_2)$ and f_2 is $O(f_3)$, then f_1 is $O(f_3)$

Proof : (Exercise)

Theorem

If f is $O(g)$, then (af) is $O(g)$ for any constant a

Proof : (Exercise)

Theorem

If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$

Proof : (Exercise)

Theorem

If f_1 is $O(f_2)$ and f_2 is $O(f_3)$, then f_1 is $O(f_3)$

Proof : (Exercise)

Theorem

If f is $O(g)$, then (af) is $O(g)$ for any constant a

Proof : (Exercise)

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} = \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}}$$

$$= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}}$$

$$= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} = \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}}$$

$$= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}}$$

$$= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} = \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}}$$

$$= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}}$$

$$= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\begin{aligned}\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} &= \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}\end{aligned}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\begin{aligned}\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} &= \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}\end{aligned}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\begin{aligned}\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} &= \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}\end{aligned}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\begin{aligned}\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} &= \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}\end{aligned}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\begin{aligned}\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} &= \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}\end{aligned}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

E.g. 01 Find the least integer n such that $\frac{(x^4+5log(x))}{(x^3+1)}$ is $O(x^n)$

Sol: Take $\frac{(x^4+5log(x))}{(x^3+1)} = \frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1}$

First consider $\frac{x^4}{x^3+1}$. And taking the limit,

$$\begin{aligned}\lim_{x\to\infty} \frac{x^4/(x^3+1)}{x^n} &= \lim_{x\to\infty} \frac{1/(x^3+1)}{x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1}{x^{n-1}+x^{n-4}} \\ &= \lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3}\end{aligned}$$

Observe that when $n < 1$, $\lim_{x\to\infty} \frac{1/x^{n-1}}{1+1/x^3} = \infty$

Thus when $n < 1$, $\frac{x^4}{x^3+1} \notin O(x^n)$

Therefore when $n = 1$ the least n, where limit exist. Hence $\frac{x^4}{x^3+1} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$

And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Now consider $\frac{5log(x)}{x^3+1}$
And take the limit, when $n = 1$,

$$\lim_{x\to\infty} \frac{5log(x)/(x^3+1)}{x^1} = 5 \lim_{x\to\infty} \frac{log(x)}{x^4+x}$$

Since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x^4 + x = \infty$, Apply L'Hôpital's rule

$$5 \lim_{x\to\infty} \frac{log(x)}{x^4+x} = 5 \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x^4+x)} = 5 \lim_{x\to\infty} \frac{1/(xln(10))}{4x^3+1}$$

$$= 5ln(10) \lim_{x\to\infty} \frac{1/x^4}{4+1/x^3}$$

$$= 0$$

Therefore $\frac{5log(x)}{x^3+1} \in O(x)$, Hence $\frac{x^4}{x^3+1} + \frac{5log(x)}{x^3+1} = \frac{(x^4+5log(x))}{(x^3+1)} \in O(x)$

Big-Omega

Definition

f is Big-Omega of g, denoted by $f \in \Omega(g)$ if there are positive constants c and k such that

$$|f(x)| \geq c\,|g(x)| \quad \textit{for } x > k$$

- $Big - \Omega$ is very similar to $Big - O$
- $Big - \Omega$ is used to indicate a lower bound on functions for large values of the independent variable
- Notice that f is $\Omega(g)$ if and only if g is $O(f)$

Big-Omega

Definition

f is Big-Omega of g, denoted by $f \in \Omega(g)$ if there are positive constants c and k such that

$$|f(x)| \geq c\,|g(x)| \quad \textit{for } x > k$$

- $Big - \Omega$ is very similar to $Big - O$
- $Big - \Omega$ is used to indicate a lower bound on functions for large values of the independent variable
- Notice that f is $\Omega(g)$ if and only if g is $O(f)$

Big-Omega

Definition

f is Big-Omega of g, denoted by $f \in \Omega(g)$ if there are positive constants c and k such that

$$|f(x)| \geq c\,|g(x)| \quad \textit{for } x > k$$

- $Big - \Omega$ is very similar to $Big - O$
- $Big - \Omega$ is used to indicate a lower bound on functions for large values of the independent variable
- Notice that f is $\Omega(g)$ if and only if g is $O(f)$

Big-Omega

Definition

f is Big-Omega of g, denoted by $f \in \Omega(g)$ if there are positive constants c and k such that

$$|f(x)| \geq c\,|g(x)| \quad \textit{for } x > k$$

- $Big - \Omega$ is very similar to $Big - O$
- $Big - \Omega$ is used to indicate a lower bound on functions for large values of the independent variable
- Notice that f is $\Omega(g)$ if and only if g is $O(f)$

Big-Omega

Definition

f is Big-Omega of g, denoted by $f \in \Omega(g)$ if there are positive constants c and k such that

$$|f(x)| \geq c\,|g(x)| \quad \textit{for } x > k$$

- $Big - \Omega$ is very similar to $Big - O$
- $Big - \Omega$ is used to indicate a lower bound on functions for large values of the independent variable
- Notice that f is $\Omega(g)$ if and only if g is $O(f)$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\lim_{x\to\infty} \frac{log(x)}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)}$$

$$= \lim_{x\to\infty} log(e)\frac{1}{x}$$

$$= 0$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\lim_{x\to\infty} \frac{log(x)}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)}$$

$$= \lim_{x\to\infty} log(e)\frac{1}{x}$$

$$= 0$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\lim_{x\to\infty} \frac{log(x)}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)}$$

$$= \lim_{x\to\infty} log(e)\frac{1}{x}$$

$$= 0$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\lim_{x\to\infty} \frac{log(x)}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)}$$

$$= \lim_{x\to\infty} log(e)\frac{1}{x}$$

$$= 0$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\lim_{x\to\infty} \frac{log(x)}{x} = \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)}$$

$$= \lim_{x\to\infty} log(e)\frac{1}{x}$$

$$= 0$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\begin{aligned}\lim_{x\to\infty} \frac{log(x)}{x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)} \\ &= \lim_{x\to\infty} log(e)\frac{1}{x} \\ &= 0\end{aligned}$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\begin{aligned}\lim_{x\to\infty} \frac{log(x)}{x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)} \\ &= \lim_{x\to\infty} log(e)\frac{1}{x} \\ &= 0\end{aligned}$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\begin{aligned}\lim_{x\to\infty} \frac{log(x)}{x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)} \\ &= \lim_{x\to\infty} log(e)\frac{1}{x} \\ &= 0\end{aligned}$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\begin{aligned}\lim_{x\to\infty} \frac{log(x)}{x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)} \\ &= \lim_{x\to\infty} log(e)\frac{1}{x} \\ &= 0\end{aligned}$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 01 Show that x is $\Omega(log(x))$

Sol:

Take $\lim_{x\to\infty} \frac{log(x)}{x}$

But since $\lim_{x\to\infty} log(x) = \infty$ and $\lim_{x\to\infty} x = \infty$

Apply L'Hôpital's rule,

$$\begin{aligned}\lim_{x\to\infty} \frac{log(x)}{x} &= \lim_{x\to\infty} \frac{\frac{d}{dx}(log(x))}{\frac{d}{dx}(x)} \\ &= \lim_{x\to\infty} log(e)\frac{1}{x} \\ &= 0\end{aligned}$$

Therefore $log(x) \in o(x)$, then $log(x) \in O(x)$

Hence $x \in \Omega(log(x))$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$
Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

E.g. 02 Show that $2x^3 + x^2 - 3x + 2$ is $\Omega(x^3)$ using definition of $Big - \Omega$

Sol:

Let, $x \geq 3$, then $x^2 - 3 \geq 0$

Then $x^2 - 3x + 2 \geq 0$, Thus

$$\begin{aligned} \left|2x^3 + x^2 - 3x + 2\right| &= 2x^3 + x^2 - 3x + 2 \\ &\geq 2x^3 \end{aligned}$$

Choosing $c = 2$ and $k = 3$

$\left|2x^3 + x^2 - 3x + 2\right| \geq 2\left|x^3\right|$ for $x \geq 3$

Therefore,

$$2x^3 + x^2 - 3x + 2 \in \Omega(x^3)$$

Big-Theta

Definition

f is $big-\Theta$ of g, written $f \in \Theta(g)$, if f both $O(g)$ and $\Omega(g)$.

The definition given for $Big-\Theta$ is equivalent to the following.

Theorem

f is $\Theta(g)$ if and only if f is $O(g)$ and g is $O(f)$

Proof :

" $\Leftarrow$ "

Since $f \in O(g)$, want to show that $f \in \Omega(g)$

Big-Theta

Definition

f is $big - \Theta$ of g, written $f \in \Theta(g)$, if f both $O(g)$ and $\Omega(g)$.

The definition given for $Big - \Theta$ is equivalent to the following.

Theorem

f is $\Theta(g)$ if and only if f is $O(g)$ and g is $O(f)$

Proof :

" $\Leftarrow$ "

Since $f \in O(g)$, want to show that $f \in \Omega(g)$

Big-Theta

Definition

f is $big-\Theta$ of g, written $f \in \Theta(g)$, if f both $O(g)$ and $\Omega(g)$.

The definition given for $Big-\Theta$ is equivalent to the following.

Theorem

f is $\Theta(g)$ if and only if f is $O(g)$ and g is $O(f)$

Proof :

" $\Leftarrow$ "

Since $f \in O(g)$, want to show that $f \in \Omega(g)$

Big-Theta

Definition

f is $big-\Theta$ of g, written $f \in \Theta(g)$, if f both $O(g)$ and $\Omega(g)$.

The definition given for $Big-\Theta$ is equivalent to the following.

Theorem

f is $\Theta(g)$ if and only if f is $O(g)$ and g is $O(f)$

Proof :

" $\Leftarrow$ "

Since $f \in O(g)$, want to show that $f \in \Omega(g)$

Big-Theta

Definition

f is $big-\Theta$ of g, written $f \in \Theta(g)$, if f both $O(g)$ and $\Omega(g)$.

The definition given for $Big-\Theta$ is equivalent to the following.

Theorem

f is $\Theta(g)$ if and only if f is $O(g)$ and g is $O(f)$

Proof :

" $\Leftarrow$ "

Since $f \in O(g)$, want to show that $f \in \Omega(g)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$

$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$

$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$

$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$
$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$
$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$
$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$
$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$
$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$
$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

Since $g \in O(f)$

$$g(x) \leq cf(x) \text{ for } x > k$$
$$f(x) \geq \frac{1}{c} g(x) \text{ where } c' = \frac{1}{c}$$
$$= c'g(x)$$

Therefore $f \in \Omega(g)$

" $\Rightarrow$ "

$f \in \Theta(g) \Rightarrow f \in O(g)$ and $f \in \Omega(g)$

Since $f \in \Omega(g)$

$$f(x) \geq cg(x) \text{ for } x > k$$
$$g(x) \leq \frac{1}{c} f(x)$$

Therefore $g \in O(f)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1}$$

$$= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}}$$

$$= \frac{2}{3}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1}$$

$$= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}}$$

$$= \frac{2}{3}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1}$$

$$= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}}$$

$$= \frac{2}{3}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\begin{aligned}\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1} \\ &= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}} \\ &= \frac{2}{3}\end{aligned}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\begin{aligned}\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1} \\ &= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}} \\ &= \frac{2}{3}\end{aligned}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\begin{aligned}\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1} \\ &= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}} \\ &= \frac{2}{3}\end{aligned}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\begin{aligned}\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1} \\ &= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}} \\ &= \frac{2}{3}\end{aligned}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\begin{aligned}\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1} \\ &= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}} \\ &= \frac{2}{3}\end{aligned}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

E.g. 01 Show that $\frac{2x^2-3}{3x^4+x^3-2x^2-1}$ is $\Theta(x^{-2})$

Sol: Take $f(x) = \frac{2x^2-3}{3x^4+x^3-2x^2-1}$ and $g(x) = x^{-2}$

Let,

$$\begin{aligned}\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1} \\ &= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}} \\ &= \frac{2}{3}\end{aligned}$$

Therefore $f \in O(g)$

By taking $\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{3}{2}$, Therefore $g \in O(f)$

Hence $f \in \Theta(g)$

Exercise

If a and b are positive real numbers different from 1, show that $log_a x \in \Theta(log_b x)$

Thus logarithmic functions have the same growth rate, hence it doesn't matter what(acceptable) base is used.

Exercise

If a and b are positive real numbers different from 1, show that $log_a x \in \Theta(log_b x)$

Thus logarithmic functions have the same growth rate, hence it doesn't matter what(acceptable) base is used.

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Determine whether following statements are true or false

- $3x^5 - 16x + 2 \in O(x^5)$?
- $3x^5 - 16x + 2 \in O(x)$?
- $3x^5 - 16x + 2 \in O(x^{17})$?
- $3x^5 - 16x + 2 \in \Omega(x^5)$?
- $3x^5 - 16x + 2 \in \Omega(x)$?
- $3x^5 - 16x + 2 \in \Omega(x^{17})$?
- $3x^5 - 16x + 2 \in \Theta(x^5)$?
- $3x^5 - 16x + 2 \in \Theta(x)$?
- $3x^5 - 16x + 2 \in \Theta(x^{17})$?

Summary

Suppose f and g are functions such that

$$\lim_{x\to\infty}\left|\frac{f(x)}{g(x)}\right| = L, \text{ where } 0 \leq L \leq \infty$$

1. If $L = 0$, f is $o(g)$ [hence $O(g)$), and g is $\Omega(f)$ (hence, not $O(f)$].
2. If $L = \infty$, then f is $\Omega(g)$ [hence, not $O(g)$ and g is $o(f)$ (hence, $O(f)$)].
3. If $0 < L < \infty$, then f is $\Theta(g)$ (hence, $O(g)$), and g is $\Theta(f)$(hence, $O(f)$).

Summary

Suppose f and g are functions such that

$$\lim_{x\to\infty}\left|\frac{f(x)}{g(x)}\right| = L, \text{ where } 0 \leq L \leq \infty$$

1. If $L = 0$, f is $o(g)$ [hence $O(g)$), and g is $\Omega(f)$ (hence, not $O(f)$].
2. If $L = \infty$, then f is $\Omega(g)$ [hence, not $O(g)$ and g is $o(f)$ (hence, $O(f)$)].
3. If $0 < L < \infty$, then f is $\Theta(g)$ (hence, $O(g)$), and g is $\Theta(f)$(hence, $O(f)$).

Summary

Suppose f and g are functions such that

$$\lim_{x\to\infty}\left|\frac{f(x)}{g(x)}\right| = L, \text{ where } 0 \leq L \leq \infty$$

1. If $L = 0$, f is $o(g)$ [hence $O(g)$), and g is $\Omega(f)$ (hence, not $O(f)$].
2. If $L = \infty$, then f is $\Omega(g)$ [hence, not $O(g)$ and g is $o(f)$ (hence, $O(f)$)].
3. If $0 < L < \infty$, then f is $\Theta(g)$ (hence, $O(g)$), and g is $\Theta(f)$(hence, $O(f)$).

Summary

Suppose f and g are functions such that

$$\lim_{x\to\infty}\left|\frac{f(x)}{g(x)}\right| = L, \text{ where } 0 \leq L \leq \infty$$

1. If $L = 0$, f is $o(g)$ [hence $O(g)$), and g is $\Omega(f)$ (hence, not $O(f)$].
2. If $L = \infty$, then f is $\Omega(g)$ [hence, not $O(g)$ and g is $o(f)$ (hence, $O(f)$)].
3. If $0 < L < \infty$, then f is $\Theta(g)$ (hence, $O(g)$), and g is $\Theta(f)$(hence, $O(f)$).

Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

What is Divide and Conquer?

- Divide it into smaller problems
- Solve the smaller problems
- Combine their solutions into a solution for the big problem

What is Divide and Conquer?

- Divide it into smaller problems
- Solve the smaller problems
- Combine their solutions into a solution for the big problem

What is Divide and Conquer?

- Divide it into smaller problems
- Solve the smaller problems
- Combine their solutions into a solution for the big problem

Example : Merge sorting

- Divide the numbers into halves
- Sort each half separately
- Merge the two sorted halves

Example : Merge sorting

- Divide the numbers into halves
- Sort each half separately
- Merge the two sorted halves

Example : Merge sorting

- Divide the numbers into halves
- Sort each half separately
- Merge the two sorted halves

- Finding the maximum and minimum of a sequence of numbers
- Integer multiplication
- Matrix multiplication

- Finding the maximum and minimum of a sequence of numbers
- Integer multiplication
- Matrix multiplication

- Finding the maximum and minimum of a sequence of numbers
- Integer multiplication
- Matrix multiplication

Finding Max and Min: In general

Find the maximum and minimum elements in an array $S[1...n]$. How many comparisons between elements of S are needed?

To find the max:

```
max := S[1];
for i := 2 to n do
    if S[i] > max then max:= S[i]
```

(Note that the min can be found similarly).

Finding Max and Min: In general

Find the maximum and minimum elements in an array $S[1...n]$. How many comparisons between elements of S are needed?

To find the max:

```
max := S[1];
for i := 2 to n do
    if S[i] > max then max:= S[i]
```

(Note that the min can be found similarly).

Finding Max and Min: In general

Find the maximum and minimum elements in an array $S[1...n]$. How many comparisons between elements of S are needed?

To find the max:

max := $S[1]$;

for i := 2 to n do

    if $S[i] >$ max then max:= $S[i]$

(Note that the min can be found similarly).

Finding Max and Min: In general

Find the maximum and minimum elements in an array $S[1...n]$. How many comparisons between elements of S are needed?

To find the max:

max := $S[1]$;

for i := 2 to n do

    if $S[i] >$ max then max:= $S[i]$

(Note that the min can be found similarly).

Finding Max and Min: In general

Find the maximum and minimum elements in an array $S[1...n]$. How many comparisons between elements of S are needed?

To find the max:

max := $S[1]$;

for i := 2 to n do

    if $S[i] >$ max then max:= $S[i]$

(Note that the min can be found similarly).

Finding Max and Min: In general

Find the maximum and minimum elements in an array $S[1...n]$. How many comparisons between elements of S are needed?

To find the max:

max := $S[1]$;

for i := 2 to n do

    if $S[i] >$ max then max:= $S[i]$

(Note that the min can be found similarly).

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Divide and Conquer Approach

Divide the array into halves. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

Function maxmin(x, y) {return max and min $S[x..y]$}

If $y - x \leq 1$ then

return (max($S[x]$,$S[y]$), min($S[x]$,$S[y]$))

else

(max1,min1):= maxmin($x, \lfloor (x+y)/2 \rfloor$)

(max2,min2):= maxmin($\lfloor (x+y)/2 \rfloor + 1, y$)

return(max(max1,max2),min(min1,min2))

Finding Max and Min: Analysis

Let $T(n)$ be the number of comparisons made by maxmin(x, y) when $n = y - x + 1$. Suppose n is a power of 2.

What is the size of the sub problems?

The first sub problem has size $\lfloor (x+y)/2 \rfloor - x + 1$. If y-x+1 is a power of 2, then $y - x$ is odd, and hence $x + y$ is odd.

Therefore,

$$\left\lfloor \frac{x+y}{2} \right\rfloor - x + 1 = \frac{x+y-1}{2} - x + 1$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

Finding Max and Min: Analysis

Let $T(n)$ be the number of comparisons made by maxmin(x, y) when $n = y - x + 1$. Suppose n is a power of 2.

What is the size of the sub problems?

The first sub problem has size $\lfloor (x+y)/2 \rfloor - x + 1$. If y-x+1 is a power of 2, then $y - x$ is odd, and hence $x + y$ is odd.

Therefore,

$$\begin{aligned} \left\lfloor \frac{x+y}{2} \right\rfloor - x + 1 &= \frac{x+y-1}{2} - x + 1 \\ &= \frac{y-x+1}{2} \\ &= n/2. \end{aligned}$$

Finding Max and Min: Analysis

Let $T(n)$ be the number of comparisons made by maxmin(x, y) when $n = y - x + 1$. Suppose n is a power of 2.

What is the size of the sub problems?

The first sub problem has size $\lfloor (x + y)/2 \rfloor - x + 1$. If y-x+1 is a power of 2, then $y - x$ is odd, and hence $x + y$ is odd.

Therefore,

$$\left\lfloor \frac{x+y}{2} \right\rfloor - x + 1 = \frac{x+y-1}{2} - x + 1$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

Finding Max and Min: Analysis

Let $T(n)$ be the number of comparisons made by maxmin(x, y) when $n = y - x + 1$. Suppose n is a power of 2.

What is the size of the sub problems?

The first sub problem has size $\lfloor (x + y)/2 \rfloor - x + 1$. If y-x+1 is a power of 2, then $y - x$ is odd, and hence $x + y$ is odd.

Therefore,

$$\begin{aligned} \left\lfloor \frac{x+y}{2} \right\rfloor - x + 1 &= \frac{x+y-1}{2} - x + 1 \\ &= \frac{y-x+1}{2} \\ &= n/2. \end{aligned}$$

The second sub problem has size $y - (\lfloor (x+y)/2 \rfloor + 1) + 1$.

Therefore,

$$y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 = y - \frac{x+y-1}{2}$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

So when n is a power of 2, procedure maxmin on an array chunks of size n calls itself twice on array chunks of size $n/2$.

Therefore,

$$T(n) = \begin{cases} 1, & \text{if } n = 2 \\ 2T(n/2) + 2, & \text{otherwise} \end{cases}$$

The second sub problem has size $y - (\lfloor (x+y)/2 \rfloor + 1) + 1$.

Therefore,

$$\begin{aligned} y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 &= y - \frac{x+y-1}{2} \\ &= \frac{y-x+1}{2} \\ &= n/2. \end{aligned}$$

So when n is a power of 2, procedure maxmin on an array chunks of size n calls itself twice on array chunks of size $n/2$.

Therefore,

$$T(n) = \begin{cases} 1, & \text{if } n = 2 \\ 2T(n/2) + 2, & \text{otherwise} \end{cases}$$

The second sub problem has size $y - (\lfloor (x+y)/2 \rfloor + 1) + 1$.

Therefore,

$$\begin{aligned} y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 &= y - \frac{x+y-1}{2} \\ &= \frac{y-x+1}{2} \\ &= n/2. \end{aligned}$$

So when n is a power of 2, procedure maxmin on an array chunks of size n calls itself twice on array chunks of size $n/2$.

Therefore,

$$T(n) = \begin{cases} 1, & \text{if } n = 2 \\ 2T(n/2) + 2, & \text{otherwise} \end{cases}$$

The second sub problem has size $y - (\lfloor (x+y)/2 \rfloor + 1) + 1$.

Therefore,

$$\begin{aligned} y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 &= y - \frac{x+y-1}{2} \\ &= \frac{y-x+1}{2} \\ &= n/2. \end{aligned}$$

So when n is a power of 2, procedure maxmin on an array chunks of size n calls itself twice on array chunks of size $n/2$.

Therefore,

$$T(n) = \begin{cases} 1, & \text{if } n = 2 \\ 2T(n/2) + 2, & \text{otherwise} \end{cases}$$

Since the maximum and minimum are finding recursively in each half,

$$
\begin{aligned}
T(n) &= 2T(n/2) + 2 \\
&= 2(2T(n/4) + 2) + 2 \\
&= 4T(n/4) + 4 + 2 \\
&= 8T(n/8) + 8 + 4 + 2 \\
&= 2^i T(n/2^i) + \sum_{j=1}^{i} 2^j \\
&= 2^{logn-1} T(2) + \sum_{j=1}^{logn-1} 2^j \\
&= n/2 + (2^{logn} - 2) \\
&= 1.5n - 2
\end{aligned}
$$

Therefore function maxmim uses only 75% as many comparisons as the naive algorithm.

Since the maximum and minimum are finding recursively in each half,

$$\begin{aligned}
T(n) &= 2T(n/2) + 2 \\
&= 2(2T(n/4) + 2) + 2 \\
&= 4T(n/4) + 4 + 2 \\
&= 8T(n/8) + 8 + 4 + 2 \\
&= 2^i T(n/2^i) + \sum_{j=1}^{i} 2^j \\
&= 2^{logn-1} T(2) + \sum_{j=1}^{logn-1} 2^j \\
&= n/2 + (2^{logn} - 2) \\
&= 1.5n - 2
\end{aligned}$$

Therefore function maxmim uses only 75% as many comparisons as the naive algorithm.

Since the maximum and minimum are finding recursively in each half,

$$\begin{aligned}
T(n) &= 2T(n/2) + 2 \\
&= 2(2T(n/4) + 2) + 2 \\
&= 4T(n/4) + 4 + 2 \\
&= 8T(n/8) + 8 + 4 + 2 \\
&= 2^i T(n/2^i) + \sum_{j=1}^{i} 2^j \\
&= 2^{logn-1} T(2) + \sum_{j=1}^{logn-1} 2^j \\
&= n/2 + (2^{logn} - 2) \\
&= 1.5n - 2
\end{aligned}$$

Therefore function maxmim uses only 75% as many comparisons as the naive algorithm.

Multiplication

Given positive integers y,z, compute $x = yz$.

In the naive multiplication algorithm addition takes $O(n)$ bit operations and multiplication takes $O(n)$, $n-$bit operations, where n is the number of bits in y and z

Therefore, the naive multiplication algorithm takes $O(n^2)$ bit operations

Can we multiply using fewer bit operation?

Multiplication

Given positive integers y,z, compute $x = yz$.

In the naive multiplication algorithm addition takes $O(n)$ bit operations and multiplication takes $O(n)$, $n-$bit operations, where n is the number of bits in y and z

Therefore, the naive multiplication algorithm takes $O(n^2)$ bit operations

Can we multiply using fewer bit operation?

Multiplication

Given positive integers y,z, compute $x = yz$.

In the naive multiplication algorithm addition takes $O(n)$ bit operations and multiplication takes $O(n)$, $n-$bit operations, where n is the number of bits in y and z

Therefore, the naive multiplication algorithm takes $O(n^2)$ bit operations

Can we multiply using fewer bit operation?

Multiplication

Given positive integers y,z, compute $x = yz$.

In the naive multiplication algorithm addition takes $O(n)$ bit operations and multiplication takes $O(n)$, $n-$bit operations, where n is the number of bits in y and z

Therefore, the naive multiplication algorithm takes $O(n^2)$ bit operations

Can we multiply using fewer bit operation?

Multiplication: Divide and Conquer Approach

Suppose n is a power of 2. Divide y and z into two halves, each with $n/2$ bits.

y	a	b
z	c	d

Then

$$y = a2^{n/2} + b$$
$$z = c2^{n/2} + d$$

And so

$$yz = (a2^{n/2} + b)(c2^{n/2} + d)$$
$$= ac2^n + (ad + bc)2^{n/2} + bd$$

Multiplication: Divide and Conquer Approach

Suppose n is a power of 2. Divide y and z into two halves, each with $n/2$ bits.

y	a	b
z	c	d

Then

$$y = a2^{n/2} + b$$
$$z = c2^{n/2} + d$$

And so

$$yz = (a2^{n/2} + b)(c2^{n/2} + d)$$
$$= ac2^n + (ad + bc)2^{n/2} + bd$$

Multiplication: Divide and Conquer Approach

Suppose n is a power of 2. Divide y and z into two halves, each with $n/2$ bits.

y	a	b
z	c	d

Then

$$y = a2^{n/2} + b$$
$$z = c2^{n/2} + d$$

And so

$$yz = (a2^{n/2} + b)(c2^{n/2} + d)$$
$$= ac2^n + (ad + bc)2^{n/2} + bd$$

Multiplication: Divide and Conquer Approach

Suppose n is a power of 2. Divide y and z into two halves, each with $n/2$ bits.

y	a	b
z	c	d

Then

$$y = a2^{n/2} + b$$
$$z = c2^{n/2} + d$$

And so

$$\begin{aligned} yz &= (a2^{n/2} + b)(c2^{n/2} + d) \\ &= ac2^n + (ad + bc)2^{n/2} + bd \end{aligned}$$

This computes yz with 4 multiplication of $n/2$ bit numbers, and some additions and shifts.

Running time given by $T(1) = \alpha$, $T(n) = 4T(n/2) + \beta n$, where α and β are constants, which has solution $O(n^2)$ by the general theorem. No gain over naive algorithm!

But $x = yz$ can also be computed as follows:

$$u := (a+b)(c+d)$$
$$v := ac$$
$$w := bd$$
$$x := v2^n + (u - v - w)2^{n/2} + w$$

Thus to multiply n bits numbers we need 3 multiplications of $n/2$ bit numbers and a constant number of additions and shifts.

This computes yz with 4 multiplication of $n/2$ bit numbers, and some additions and shifts.

Running time given by $T(1) = \alpha$, $T(n) = 4T(n/2) + \beta n$, where α and β are constants, which has solution $O(n^2)$ by the general theorem. No gain over naive algorithm!

But $x = yz$ can also be computed as follows:

$$u := (a+b)(c+d)$$
$$v := ac$$
$$w := bd$$
$$x := v2^n + (u - v - w)2^{n/2} + w$$

Thus to multiply n bits numbers we need 3 multiplications of $n/2$ bit numbers and a constant number of additions and shifts.

This computes yz with 4 multiplication of $n/2$ bit numbers, and some additions and shifts.

Running time given by $T(1) = \alpha$, $T(n) = 4T(n/2) + \beta n$, where α and β are constants, which has solution $O(n^2)$ by the general theorem. No gain over naive algorithm!

But $x = yz$ can also be computed as follows:

$$u := (a+b)(c+d)$$
$$v := ac$$
$$w := bd$$
$$x := v2^n + (u - v - w)2^{n/2} + w$$

Thus to multiply n bits numbers we need 3 multiplications of $n/2$ bit numbers and a constant number of additions and shifts.

This computes yz with 4 multiplication of $n/2$ bit numbers, and some additions and shifts.

Running time given by $T(1) = \alpha$, $T(n) = 4T(n/2) + \beta n$, where α and β are constants, which has solution $O(n^2)$ by the general theorem. No gain over naive algorithm!

But $x = yz$ can also be computed as follows:

$$u := (a+b)(c+d)$$
$$v := ac$$
$$w := bd$$
$$x := v2^n + (u - v - w)2^{n/2} + w$$

Thus to multiply n bits numbers we need 3 multiplications of $n/2$ bit numbers and a constant number of additions and shifts.

This computes yz with 4 multiplication of $n/2$ bit numbers, and some additions and shifts.

Running time given by $T(1) = \alpha$, $T(n) = 4T(n/2) + \beta n$, where α and β are constants, which has solution $O(n^2)$ by the general theorem. No gain over naive algorithm!

But $x = yz$ can also be computed as follows:

$$\begin{aligned} u &:= (a+b)(c+d) \\ v &:= ac \\ w &:= bd \\ x &:= v2^n + (u-v-w)2^{n/2} + w \end{aligned}$$

Thus to multiply n bits numbers we need 3 multiplications of $n/2$ bit numbers and a constant number of additions and shifts.

Multiplication: Analysis

Therefore,

$$T(n) = \begin{cases} \alpha, & \text{if } n = 1 \\ 3T(n/2) + \beta n, & \text{otherwise} \end{cases}$$

Assignment 2: Prove that the divide and conquer multiplication algorithm uses

$$T(n) = O(n^{log3}) = O(n^{1.59}),$$

bit operations.

Multiplication: Analysis

Therefore,

$$T(n) = \begin{cases} \alpha, & \text{if } n = 1 \\ 3T(n/2) + \beta n, & \text{otherwise} \end{cases}$$

Assignment 2: Prove that the divide and conquer multiplication algorithm uses

$$T(n) = O(n^{log3}) = O(n^{1.59}),$$

bit operations.

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}
    for $i := 1$ to n do
        for $j := 1$ to n do
            $X[i, j] := 0$
            for $k := 1$ to n do
                $X[i, j] := X[i, j] + Y[i, k] * Z[k, j]$;

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}
    for $i := 1$ to n do
        for $j := 1$ to n do
            $X[i, j] := 0$
            for $k := 1$ to n do
                $X[i, j] := X[i, j] + Y[i, k] * Z[k, j]$;

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}
    for $i := 1$ to n do
        for $j := 1$ to n do
            $X[i, j] := 0$
            for $k := 1$ to n do
                $X[i, j] := X[i, j] + Y[i, k] * Z[k, j]$;

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}

```
for i := 1 to n do
    for j := 1 to n do
        X[i, j] := 0
        for k := 1 to n do
            X[i, j] := X[i, j] + Y[i, k] * Z[k, j];
```

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}

```
for i := 1 to n do
    for j := 1 to n do
        X[i, j] := 0
        for k := 1 to n do
            X[i, j] := X[i, j] + Y[i, k] * Z[k, j];
```

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}

```
for i := 1 to n do
    for j := 1 to n do
        X[i, j] := 0
        for k := 1 to n do
            X[i, j] := X[i, j] + Y[i, k] * Z[k, j];
```

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication

The naive matrix multiplication algorithm:

Procedure matmultiply(X, Y, Z, n) {multiplies $n * n$ matrices $X := YZ$}

```
for i := 1 to n do
    for j := 1 to n do
        X[i, j] := 0
        for k := 1 to n do
            X[i, j] := X[i, j] + Y[i, k] * Z[k, j];
```

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Matrix Multiplication: Divide and Conquer Approach

Divide X, Y, Z each into four $(n/2) * (n/2)$ matrices.

$$X = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

$$Y = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$Z = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Then

$$I = AE + BG$$
$$J = AF + BH$$
$$K = CE + DG$$
$$L = CF + DH$$

Matrix Multiplication: Divide and Conquer Approach

Divide X, Y, Z each into four $(n/2) * (n/2)$ matrices.

$$X = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

$$Y = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$Z = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Then

$$I = AE + BG$$

$$J = AF + BH$$

$$K = CE + DG$$

$$L = CF + DH$$

Matrix Multiplication: Divide and Conquer Approach

Divide X, Y, Z each into four $(n/2) * (n/2)$ matrices.

$$X = \begin{bmatrix} I & J \\ K & L \end{bmatrix}$$

$$Y = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$Z = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Then

$$I = AE + BG$$

$$J = AF + BH$$

$$K = CE + DG$$

$$L = CF + DH$$

Matrix Multiplication: Analysis

Let $T(n)$ be the time to multiply two $n \times n$ matrices.

Thus,

$$T(n) = \begin{cases} c, & \text{if } n = 1 \\ 8T(n/2) + dn^2, & \text{otherwise} \end{cases}$$

where c,d are constants.

Matrix Multiplication: Analysis

Let $T(n)$ be the time to multiply two $n \times n$ matrices.

Thus,

$$T(n) = \begin{cases} c, & \text{if } n = 1 \\ 8T(n/2) + dn^2, & \text{otherwise} \end{cases}$$

where c,d are constants.

Therefore,

$$\begin{aligned}
T(n) &= 8T(n/2) + dn^2 \\
&= 8(8T(n/4) + d(n/2)^2) + dn^2 \\
&= 8^2 T(n/4) + 2dn^2 + dn^2 \\
&= 8^3(T(n/8) + 4dn^2 + 2dn^2 + dn^2 \\
&= 8^i T(n/2^i) + dn^2 \sum_{j=0}^{i} 2^j \\
&= 8^{log\ n} T(1) + dn^2 \sum_{j=0}^{logn-1} 2^j \\
&= cn^3 + dn^2(n-1) \\
&= O(n^3).
\end{aligned}$$

Hence the approach gain us nothing.

Therefore,

$$\begin{aligned}
T(n) &= 8T(n/2) + dn^2 \\
&= 8(8T(n/4) + d(n/2)^2) + dn^2 \\
&= 8^2 T(n/4) + 2dn^2 + dn^2 \\
&= 8^3(T(n/8) + 4dn^2 + 2dn^2 + dn^2 \\
&= 8^i T(n/2^i) + dn^2 \sum_{j=0}^{i} 2^j \\
&= 8^{log\ n} T(1) + dn^2 \sum_{j=0}^{logn-1} 2^j \\
&= cn^3 + dn^2(n-1) \\
&= O(n^3).
\end{aligned}$$

Hence the approach gain us nothing.

Compute

$$
\begin{aligned}
M_1 &:= (A + C)(E + F) \\
M_2 &:= (B + D)(G + H) \\
M_3 &:= (A - D)(E + H) \\
M_4 &:= A(F - H) \\
M_5 &:= (C + D)E \\
M_6 &:= (A + B)H \\
M_7 &:= D(G - E)
\end{aligned}
$$

Then

$$
\begin{aligned}
I &:= M_2 + M_3 - M_6 - M_7 \\
J &:= M_4 + M_6 \\
K &:= M_5 + M_7 \\
L &:= M_1 - M_3 - M_4 - M_5
\end{aligned}
$$

Analysis of Strassen's Algorithm

Therefore,

$$T(n) = \begin{cases} c, & \text{if } n = 1 \\ 7T(n/2) + dn^2, & \text{otherwise} \end{cases}$$

where c,d are constants.

$$
\begin{aligned}
T(n) &= 7T(n/2) + dn^2 \\
&= 7(7T(n/4) + d(n/2)^2) + dn^2 \\
&= 7^2 T(n/4) + 7dn^2/4 + dn^2 \\
&= 7^3 T(n/8) + 7^2 dn^2/4^2 + 7dn^2/4 + dn^2 \\
&= 7^i T(n/2^i) + dn^2 \sum_{j=0}^{i-1} (7/4)^j \\
&= 7^{\log n} T(1) + dn^2 \sum_{j=0}^{\log n - 1} (7/4)^j \\
&= cn^{\log 7} + dn^2 \frac{(7/4)^{\log n} - 1}{(7/4) - 1} \\
&= cn^{\log 7} + \frac{4}{3} dn^2 \left(\frac{n^{\log 7}}{n^2} - 1\right) \\
&= O(n^{\log 7}) \approx O(n^{2.8}).
\end{aligned}
$$

- Integer multiplication:$O(nlognloglogn)$.

  Schonhage and Strassen, ”Schnelle multiplication grosser zahlen”, Computing , Vol. 7, 281-292, 1971.

- Matrix multiplication: $O(n^{2.376})$.

  Copper smith and Winograd , ”Matrix multiplication via arithmetic progressions”, Journal of Symbolic Computation Vol. 9, pp. 251-280, 1990.

Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

When divide and conquer generates a large number of identical sub problems, recursion is too expensive.

Instead, store solutions to sub problems in a table.

This technique is called dynamic programming.

When divide and conquer generates a large number of identical sub problems, recursion is too expensive.

Instead, store solutions to sub problems in a table.

This technique is called dynamic programming.

When divide and conquer generates a large number of identical sub problems, recursion is too expensive.

Instead, store solutions to sub problems in a table.

This technique is called dynamic programming.

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

Construction:

- Take part of divide and conquer algorithm that does the ”conquer” part and replace recursive calls with table look-ups
- Instead of returning a value, record it in a table entry
- Use base of divide and conquer to fill in start of table
- Devise ”look-up template ”
- Devise for-loops that fill the table using ”look-up template”

Construction:

- Take part of divide and conquer algorithm that does the ”conquer” part and replace recursive calls with table look-ups
- Instead of returning a value, record it in a table entry
- Use base of divide and conquer to fill in start of table
- Devise ”look-up template ”
- Devise for-loops that fill the table using ”look-up template”

Construction:

- Take part of divide and conquer algorithm that does the ”conquer” part and replace recursive calls with table look-ups
- Instead of returning a value, record it in a table entry
- Use base of divide and conquer to fill in start of table
- Devise ”look-up template ”
- Devise for-loops that fill the table using ”look-up template”

Construction:

- Take part of divide and conquer algorithm that does the ”conquer” part and replace recursive calls with table look-ups
- Instead of returning a value, record it in a table entry
- Use base of divide and conquer to fill in start of table
- Devise ”look-up template ”
- Devise for-loops that fill the table using ”look-up template”

Construction:

- Take part of divide and conquer algorithm that does the ”conquer” part and replace recursive calls with table look-ups
- Instead of returning a value, record it in a table entry
- Use base of divide and conquer to fill in start of table
- Devise ”look-up template ”
- Devise for-loops that fill the table using ”look-up template”

Construction:

- Take part of divide and conquer algorithm that does the ”conquer” part and replace recursive calls with table look-ups
- Instead of returning a value, record it in a table entry
- Use base of divide and conquer to fill in start of table
- Devise ”look-up template ”
- Devise for-loops that fill the table using ”look-up template”

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

To choose r things out of n, either

- Choose the first item. Then we must choose the remaining $r-1$ items from the other $n-1$ items.

or

- Don't choose the first item. Then we must choose the r items from the other $n-1$ items.

Therefore,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

To choose r things out of n, either

- Choose the first item. Then we must choose the remaining $r-1$ items from the other $n-1$ items.

or

- Don't choose the first item. Then we must choose the r items from the other $n-1$ items.

Therefore,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

To choose r things out of n, either

- Choose the first item. Then we must choose the remaining $r-1$ items from the other $n-1$ items.

  or

- Don't choose the first item. Then we must choose the r items from the other $n-1$ items.

Therefore,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

To choose r things out of n, either

- Choose the first item. Then we must choose the remaining $r-1$ items from the other $n-1$ items.

  or

- Don't choose the first item. Then we must choose the r items from the other $n-1$ items.

Therefore,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

To choose r things out of n, either

- Choose the first item. Then we must choose the remaining $r-1$ items from the other $n-1$ items.

  or

- Don't choose the first item. Then we must choose the r items from the other $n-1$ items.

Therefore,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Divide and Conquer

This gives a simple divide and conquer algorithm for finding the number of combinations of n things chosen r at a time.

function choose (n, r)

If $r = 0$ or $n = r$ then return 1

else return(choose $(n-1, r-1)$+choose$(n-1, r)$)

Divide and Conquer

This gives a simple divide and conquer algorithm for finding the number of combinations of n things chosen r at a time.

function choose (n, r)

If $r = 0$ or $n = r$ then return 1

else return(choose $(n-1, r-1)$+choose$(n-1, r)$)

Divide and Conquer

This gives a simple divide and conquer algorithm for finding the number of combinations of n things chosen r at a time.

function choose (n, r)

If $r = 0$ or $n = r$ then return 1

else return(choose $(n-1, r-1)$+choose$(n-1, r)$)

Divide and Conquer

This gives a simple divide and conquer algorithm for finding the number of combinations of n things chosen r at a time.

function choose (n, r)

If $r = 0$ or $n = r$ then return 1

else return(choose $(n-1, r-1)$+choose$(n-1, r)$)

Analysis : Let $T(n)$ be the worst case running time of choose(n, r) over all possible values of r.

Then,

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{otherwise} \end{cases}$$

for some constants c, d.

Analysis : Let $T(n)$ be the worst case running time of choose(n, r) over all possible values of r.

Then,

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{otherwise} \end{cases}$$

for some constants c, d.

Analysis : Let $T(n)$ be the worst case running time of choose(n, r) over all possible values of r.

Then,

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{otherwise} \end{cases}$$

for some constants c, d.

Hence,

$$\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \\
&= 4(2T(n-3) + d) + 2 + d \\
&= 8T(n-3) + 4d + 2d + d \\
&= 2^i T(n-i) + d\sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d\sum_{j=0}^{n-2} 2^j \\
&= (c+d)2^{n-1} - d
\end{aligned}$$

Hence, $T(n) = O(2^{n-1})$.

The problem is, the algorithm solves the same sub problem over and over again!

Figure

The problem is, the algorithm solves the same sub problem over and over again!

Figure

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$T[i,j]$ holds $\binom{i}{j}$.

```
function choose (n, r)
    for i := 0 to n − r do T[i, 0] := 1;
    for i := 0 to r do T[i, i] := 1;
    for j := 1 to r do
        for i := j + 1 to n − r + j do
    T[i, j] := T[i − 1, j − 1] + T[i − 1, j]
return(T[n, r])
```

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$T[i,j]$ holds $\binom{i}{j}$.

```
function choose (n, r)
    for i := 0 to n − r do T[i, 0] := 1;
    for i := 0 to r do T[i, i] := 1;
    for j := 1 to r do
        for i := j + 1 to n − r + j do
    T[i, j] := T[i − 1, j − 1] + T[i − 1, j]
return(T[n, r])
```

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$T[i,j]$ holds $\binom{i}{j}$.

```
function choose (n, r)
    for i := 0 to n − r do T[i, 0] := 1;
    for i := 0 to r do T[i, i] := 1;
    for j := 1 to r do
        for i := j + 1 to n − r + j do
    T[i, j] := T[i − 1, j − 1] + T[i − 1, j]
return(T[n, r])
```

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$$T[i,j] \text{ holds } \binom{i}{j}.$$

function choose (n, r)

for $i := 0$ to $n - r$ do $T[i, 0] := 1$;

for $i := 0$ to r do $T[i, i] := 1$;

for $j := 1$ to r do

for $i := j + 1$ to $n - r + j$ do

$T[i, j] := T[i - 1, j - 1] + T[i - 1, j]$

return$(T[n, r])$

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$$T[i,j] \text{ holds } \binom{i}{j}.$$

function choose (n, r)

```
for i := 0 to n − r do T[i, 0] := 1;
for i := 0 to r do T[i, i] := 1;
for j := 1 to r do
    for i := j + 1 to n − r + j do
T[i, j] := T[i − 1, j − 1] + T[i − 1, j]
return(T[n, r])
```

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$$T[i,j] \text{ holds } \binom{i}{j}.$$

function choose (n, r)

for $i := 0$ to $n - r$ do $T[i, 0] := 1$;

for $i := 0$ to r do $T[i, i] := 1$;

for $j := 1$ to r do

for $i := j + 1$ to $n - r + j$ do

$T[i, j] := T[i - 1, j - 1] + T[i - 1, j]$

return$(T[n, r])$

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$$T[i,j] \text{ holds } \binom{i}{j}.$$

function choose (n, r)

```
    for i := 0 to n − r do T[i, 0] := 1;
    for i := 0 to r do T[i, i] := 1;
    for j := 1 to r do
        for i := j + 1 to n − r + j do
    T[i, j] := T[i − 1, j − 1] + T[i − 1, j]
return(T[n, r])
```

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$$T[i,j] \text{ holds } \binom{i}{j}.$$

function choose (n, r)

```
    for i := 0 to n − r do T[i, 0] := 1;
    for i := 0 to r do T[i, i] := 1;
    for j := 1 to r do
        for i := j + 1 to n − r + j do
    T[i, j] := T[i − 1, j − 1] + T[i − 1, j]
return(T[n, r])
```

Dynamic Programming

Pascal's Triangle. Use a table $T[0...n, 0...r]$.

$$T[i,j] \text{ holds } \binom{i}{j}.$$

function choose (n, r)

for $i := 0$ to $n - r$ do $T[i, 0] := 1$;

for $i := 0$ to r do $T[i, i] := 1$;

for $j := 1$ to r do

for $i := j + 1$ to $n - r + j$ do

$T[i, j] := T[i - 1, j - 1] + T[i - 1, j]$

return$(T[n, r])$

Initialization

Figure

General Rule

To fill in $T[i, j]$, we need $T[i-1, j-1]$ and $T[i-1, j]$ to be already filled in.

Figure

Fill in the rows from left to right. Fill in each of the columns from top to bottom.

General Rule

To fill in $T[i, j]$, we need $T[i-1, j-1]$ and $T[i-1, j]$ to be already filled in.

Figure

Fill in the rows from left to right. Fill in each of the columns from top to bottom.

General Rule

To fill in $T[i, j]$, we need $T[i-1, j-1]$ and $T[i-1, j]$ to be already filled in.

Figure

Fill in the rows from left to right. Fill in each of the columns from top to bottom.

Example

Figure

How many table entries are filled in?

$(n - r + 1)(r + 1) = nr + n - r^2 + 1 \leq n(r + 1) + 1$

Each entry takes time $O(1)$, so total time required is $O(n^2)$.

This is much better than $O(2^n)$.

How many table entries are filled in?

$$(n - r + 1)(r + 1) = nr + n - r^2 + 1 \leq n(r + 1) + 1$$

Each entry takes time $O(1)$, so total time required is $O(n^2)$.

This is much better than $O(2^n)$.

How many table entries are filled in?

$(n - r + 1)(r + 1) = nr + n - r^2 + 1 \leq n(r + 1) + 1$

Each entry takes time $O(1)$, so total time required is $O(n^2)$.

This is much better than $O(2^n)$.

Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

Given n items of length $s_1, s_2, ..., s_n$ is there a subset of these items with total length exactly S?

Figure

Want knapsack (i, j) to return true if there is a subset of the first i items that has total length exactly j.

Figure

When can knapsack (i, j) return true? Either the ith item is used, or it is not.

- If the ith item is not used, and knapsack $(i-1, j)$ returns true.

  Figure

- If the ith item is used, and knapsack $(i-1, j-s_i)$ returns true.

  Figure

When can knapsack (i, j) return true? Either the ith item is used, or it is not.

- If the ith item is not used, and knapsack $(i-1, j)$ returns true.

  Figure

- If the ith item is used, and knapsack $(i-1, j-s_i)$ returns true.

  Figure

When can knapsack (i, j) return true? Either the ith item is used, or it is not.

- If the ith item is not used, and knapsack $(i-1, j)$ returns true.

  Figure

- If the ith item is used, and knapsack $(i-1, j-s_i)$ returns true.

  Figure

The Code

Call knapsack (n, S)

function knapsack (i, j) % returns true if $s_1, ..., s_i$ can fill j

if $i = 0$ then return $(j = 0)$

else if knapsack $(i - 1, j)$ then return(true)

else if $s_i \leq j$ then

return (knapsack $(i - 1, j - s_i)$)

The Code

Call knapsack (n, S)

function knapsack (i, j) % returns true if $s_1, ..., s_i$ can fill j

if $i = 0$ then return $(j = 0)$

else if knapsack $(i - 1, j)$ then return(true)

else if $s_i \leq j$ then

return (knapsack $(i - 1, j - s_i)$)

The Code

Call knapsack (n, S)

function knapsack (i, j) % returns true if $s_1, ..., s_i$ can fill j

if $i = 0$ then return $(j = 0)$

else if knapsack $(i - 1, j)$ then return(true)

else if $s_i \leq j$ then

return (knapsack $(i - 1, j - s_i)$)

The Code

Call knapsack (n, S)

function knapsack (i, j) % returns true if $s_1, ..., s_i$ can fill j

if $i = 0$ then return $(j = 0)$

else if knapsack $(i - 1, j)$ then return(true)

else if $s_i \leq j$ then

return (knapsack $(i - 1, j - s_i)$)

The Code

Call knapsack (n, S)

function knapsack (i, j) % returns true if $s_1, ..., s_i$ can fill j

if $i = 0$ then return $(j = 0)$

else if knapsack $(i - 1, j)$ then return(true)

else if $s_i \leq j$ then

return (knapsack $(i - 1, j - s_i)$)

The Code

Call knapsack (n, S)

function knapsack (i, j) % returns true if $s_1, ..., s_i$ can fill j

if $i = 0$ then return $(j = 0)$

else if knapsack $(i - 1, j)$ then return(true)

else if $s_i \leq j$ then

return (knapsack $(i - 1, j - s_i)$)

Time Analysis

Let T(n) be the running time of knapsack (n,s).

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{otherwise} \end{cases}$$

Hence, by standard techniques, $T(n) = \Theta(2^n)$.

Time Analysis

Let T(n) be the running time of knapsack (n,s).

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{otherwise} \end{cases}$$

Hence, by standard techniques, $T(n) = \Theta(2^n)$.

Time Analysis

Let T(n) be the running time of knapsack (n,s).

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{otherwise} \end{cases}$$

Hence, by standard techniques, $T(n) = \Theta(2^n)$.

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

$t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

  $t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

    $t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

    $t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

    $t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

    $t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

$t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Store knapsack (i, j) in table $t[i, j]$.

$t[i, j]$ is set to turn iff either:

- $t[i-1, j]$ is true, or
- $t[i-1, j-s_i]$ makes sense and is true.

This is done with the following code:

$t[i, j] := t[i-1, j]$

if $j - s_i \geq 0$ then

    $t[i, j] := t[i, j]$ or $t[i-1, j-s_i]$

Figure

Filling in the Table

Figure

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

The Algorithm

function knapsack $(s_1, s_2, ..., s_n, S)$

1. $t[0,0] := true$
2. for $j := 1$ to S do $t[0,j] := false$
3. for $i := 1$ to n do
4. for $j := 0$ to S do
5. t[i,j]:= t$[i-1, j]$
6. if $j - s_i \geq 0$ then
7. $t[i,j] := t[i,j]$ or $t[i-1, j-s_i]$
8. return$(t[n,S])$

Time Analysis

- Lines 1 and 8 cost $O(1)$.
- The for-loop on line 2 cost $O(S)$.
- Lines 5-7 cost $O(1)$.
- The for-loop on lines 4-7 costs $O(S)$.
- The for-loop on lines 3-7 costs $O(nS)$.

Therefore, the algorithm runs in time $O(nS)$. This is usable if S is small.

Time Analysis

- Lines 1 and 8 cost $O(1)$.
- The for-loop on line 2 cost $O(S)$.
- Lines 5-7 cost $O(1)$.
- The for-loop on lines 4-7 costs $O(S)$.
- The for-loop on lines 3-7 costs $O(nS)$.

Therefore, the algorithm runs in time $O(nS)$. This is usable if S is small.

Time Analysis

- Lines 1 and 8 cost $O(1)$.
- The for-loop on line 2 cost $O(S)$.
- Lines 5-7 cost $O(1)$.
- The for-loop on lines 4-7 costs $O(S)$.
- The for-loop on lines 3-7 costs $O(nS)$.

Therefore, the algorithm runs in time $O(nS)$. This is usable if S is small.

Time Analysis

- Lines 1 and 8 cost $O(1)$.
- The for-loop on line 2 cost $O(S)$.
- Lines 5-7 cost $O(1)$.
- The for-loop on lines 4-7 costs $O(S)$.
- The for-loop on lines 3-7 costs $O(nS)$.

Therefore, the algorithm runs in time $O(nS)$. This is usable if S is small.

Time Analysis

- Lines 1 and 8 cost $O(1)$.
- The for-loop on line 2 cost $O(S)$.
- Lines 5-7 cost $O(1)$.
- The for-loop on lines 4-7 costs $O(S)$.
- The for-loop on lines 3-7 costs $O(nS)$.

Therefore, the algorithm runs in time $O(nS)$. This is usable if S is small.

Time Analysis

- Lines 1 and 8 cost $O(1)$.
- The for-loop on line 2 cost $O(S)$.
- Lines 5-7 cost $O(1)$.
- The for-loop on lines 4-7 costs $O(S)$.
- The for-loop on lines 3-7 costs $O(nS)$.

Therefore, the algorithm runs in time $O(nS)$. This is usable if S is small.

$s_1 = 1, s_2 = 2, s_3 = 2, s_4 = 4, s_5 = 5, s_6 = 2, s_7 = 4,$

$S = 15.$

$t[i, j] := t[i-1, j]$ or $t[i-1, j-s_i]$

$t[3, 3] := t[2, 3]$ or $t[2, 3-s_3]$

$t[3, 3] := t[2, 3]$ or $t[2, 1]$

Figure

$s_1 = 1, s_2 = 2, s_3 = 2, s_4 = 4, s_5 = 5, s_6 = 2, s_7 = 4,$

$S = 15.$

$t[i, j] := t[i-1, j]$ or $t[i-1, j-s_i]$

$t[3, 3] := t[2, 3]$ or $t[2, 3-s_3]$

$t[3, 3] := t[2, 3]$ or $t[2, 1]$

Figure

$s_1 = 1, s_2 = 2, s_3 = 2, s_4 = 4, s_5 = 5, s_6 = 2, s_7 = 4,$

$S = 15.$

$t[i, j] := t[i-1, j]$ or $t[i-1, j-s_i]$

$t[3, 3] := t[2, 3]$ or $t[2, 3-s_3]$

$t[3, 3] := t[2, 3]$ or $t[2, 1]$

Figure

$s_1 = 1, s_2 = 2, s_3 = 2, s_4 = 4, s_5 = 5, s_6 = 2, s_7 = 4,$

$S = 15.$

$t[i, j] := t[i-1, j]$ or $t[i-1, j-s_i]$

$t[3, 3] := t[2, 3]$ or $t[2, 3-s_3]$

$t[3, 3] := t[2, 3]$ or $t[2, 1]$

Figure

$s_1 = 1, s_2 = 2, s_3 = 2, s_4 = 4, s_5 = 5, s_6 = 2, s_7 = 4,$

$S = 15.$

$t[i, j] := t[i - 1, j]$ or $t[i - 1, j - s_i]$

$t[3, 3] := t[2, 3]$ or $t[2, 3 - s_3]$

$t[3, 3] := t[2, 3]$ or $t[2, 1]$

Figure

$t[i,j] := t[i-1,j]$ or $t[i-1,j-s_i]$

$t[3,4] := t[2,4]$ or $t[2,4-s_3]$

$t[3,4] := t[2,4]$ or $t[2,2]$

Figure

$t[i,j] := t[i-1,j]$ or $t[i-1,j-s_i]$

$t[3,4] := t[2,4]$ or $t[2,4-s_3]$

$t[3,4] := t[2,4]$ or $t[2,2]$

Figure

$t[i,j] := t[i-1,j]$ or $t[i-1,j-s_i]$

$t[3,4] := t[2,4]$ or $t[2,4-s_3]$

$t[3,4] := t[2,4]$ or $t[2,2]$

Figure

Outline

1. Introduction
   - Algorithm Analysis
   - Computational Complexity Theory
   - Factors Influencing Program Efficiency
   - Motivation
   - Mathematical Induction
2. Growth of Functions
   - Introduction
   - The Big-O Notation
   - Little-o
   - Basic Properties of Big-O
   - Big - Omega
   - Big - Theta
   - Summary
3. Divide and Conquer
   - Introduction
   - Applications

Greedy Algorithms

- Start with a solution to a small sub problem
- Build up to a solution to the whole problem
- Make choices that look good in the short term

Disadvantage: Greedy algorithms don't always work. (Short term solutions can be disastrous in the long term.) Hard to prove correct.

Advantage: Greedy algorithms work fast when they work. Simple algorithms, easy to implement.

Greedy Algorithms

- Start with a solution to a small sub problem
- Build up to a solution to the whole problem
- Make choices that look good in the short term

Disadvantage: Greedy algorithms don't always work. (Short term solutions can be disastrous in the long term.) Hard to prove correct.

Advantage: Greedy algorithms work fast when they work. Simple algorithms, easy to implement.

Greedy Algorithms

- Start with a solution to a small sub problem
- Build up to a solution to the whole problem
- Make choices that look good in the short term

  Disadvantage: Greedy algorithms don't always work. (Short term solutions can be disastrous in the long term.) Hard to prove correct.

  Advantage: Greedy algorithms work fast when they work. Simple algorithms, easy to implement.

Greedy Algorithms

- Start with a solution to a small sub problem
- Build up to a solution to the whole problem
- Make choices that look good in the short term

Disadvantage: Greedy algorithms don't always work. (Short term solutions can be disastrous in the long term.) Hard to prove correct.

Advantage: Greedy algorithms work fast when they work. Simple algorithms, easy to implement.

Greedy Algorithms

- Start with a solution to a small sub problem
- Build up to a solution to the whole problem
- Make choices that look good in the short term

Disadvantage: Greedy algorithms don't always work. (Short term solutions can be disastrous in the long term.) Hard to prove correct.

Advantage: Greedy algorithms work fast when they work. Simple algorithms, easy to implement.

Greedy Algorithms

- Start with a solution to a small sub problem
- Build up to a solution to the whole problem
- Make choices that look good in the short term

  Disadvantage: Greedy algorithms don't always work. (Short term solutions can be disastrous in the long term.) Hard to prove correct.

  Advantage: Greedy algorithms work fast when they work. Simple algorithms, easy to implement.

Greedy algorithms for

- Optimal tape storage
- Continuous knapsack problem

Greedy algorithms for

- Optimal tape storage
- Continuous knapsack problem

Greedy algorithms for

- Optimal tape storage
- Continuous knapsack problem

Optimal Tape Storage Given n files of length $m_1, m_2, \ldots, m_n$,

find which order is the best to store them on a tape, assuming

- each retrieval starts with the tape rewound.
- each retrieval takes time equal to the length of the preceding files in the tape plus the length of the retrieved file.
- all files are to be retrieved.

Optimal Tape Storage Given n files of length m_1, m_2,..., m_n,

find which order is the best to store them on a tape, assuming

- each retrieval starts with the tape rewound.
- each retrieval takes time equal to the length of the preceding files in the tape plus the length of the retrieved file.
- all files are to be retrieved.

Optimal Tape Storage Given n files of length m_1, m_2,..., m_n,

find which order is the best to store them on a tape, assuming

- each retrieval starts with the tape rewound.
- each retrieval takes time equal to the length of the preceding files in the tape plus the length of the retrieved file.
- all files are to be retrieved.

Optimal Tape Storage Given n files of length m_1, m_2,..., m_n,

find which order is the best to store them on a tape, assuming

- each retrieval starts with the tape rewound.
- each retrieval takes time equal to the length of the preceding files in the tape plus the length of the retrieved file.
- all files are to be retrieved.

Optimal Tape Storage Given n files of length m_1, m_2,..., m_n,

find which order is the best to store them on a tape, assuming

- each retrieval starts with the tape rewound.
- each retrieval takes time equal to the length of the preceding files in the tape plus the length of the retrieved file.
- all files are to be retrieved.

Optimal Tape Storage Given n files of length m_1, m_2,..., m_n,

find which order is the best to store them on a tape, assuming

- each retrieval starts with the tape rewound.
- each retrieval takes time equal to the length of the preceding files in the tape plus the length of the retrieved file.
- all files are to be retrieved.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

Example $n = 3$; $m_1 = 5$, $m_2 = 10$, $m_3 = 3$.

There are $3! = 6$ possible orders.

1,2,3: $(5 + 10 + 3) + (5 + 10) + 5 = 38$

1,3,2: $(5 + 3 + 10) + (5 + 3) + 5 = 31$

2,1,3: $(10 + 5 + 3) + (10 + 5) + 10 = 43$

2,3,1: $(10 + 3 + 5) + (10 + 3) + 10 = 41$

3,1,2: $(3 + 5 + 10) + (3 + 5) + 3 = 29$

3,2,1: $(3 + 10 + 5) + (3 + 10) + 3 = 34$

The best order is 3,1,2.

The Greedy Solution

make tape empty

for $i := 1$ to n do

grab the next shortest file

put it next on tape.

The algorithm takes the best short term choice without checking to see whether it is the best long term decision.

The Greedy Solution

make tape empty

for $i := 1$ to n do

grab the next shortest file

put it next on tape.

The algorithm takes the best short term choice without checking to see whether it is the best long term decision.

The Greedy Solution

make tape empty

for $i := 1$ to n do

grab the next shortest file

put it next on tape.

The algorithm takes the best short term choice without checking to see whether it is the best long term decision.

The Greedy Solution

make tape empty

for $i := 1$ to n do

grab the next shortest file

put it next on tape.

The algorithm takes the best short term choice without checking to see whether it is the best long term decision.

The Greedy Solution

make tape empty

for $i := 1$ to n do

grab the next shortest file

put it next on tape.

The algorithm takes the best short term choice without checking to see whether it is the best long term decision.

Analysis

$O(n \log n)$ for sorting

$O(n)$ for the rest

Analysis

$O(n \log n)$ for sorting

$O(n)$ for the rest

Analysis

$O(n \log n)$ for sorting

$O(n)$ for the rest

This is similar to the knapsack problem met earlier.

- given n objects $A_1, A_2, ..., A_n$
- given a knapsack of length S
- A_i has length s_i
- A_i has weight w_i
- an x_i −fraction of A_i, where $0 \leq x_i \leq 1$, has length $x_i s_i$ and weight $x_i w_i$

This is similar to the knapsack problem met earlier.

- given n objects $A_1, A_2, ..., A_n$
- given a knapsack of length S
- A_i has length s_i
- A_i has weight w_i
- an x_i −fraction of A_i, where $0 \leq x_i \leq 1$, has length $x_i s_i$ and weight $x_i w_i$

This is similar to the knapsack problem met earlier.

- given n objects $A_1, A_2, ..., A_n$
- given a knapsack of length S
- A_i has length s_i
- A_i has weight w_i
- an x_i −fraction of A_i, where $0 \leq x_i \leq 1$, has length $x_i s_i$ and weight $x_i w_i$

This is similar to the knapsack problem met earlier.

- given n objects $A_1, A_2, ..., A_n$
- given a knapsack of length S
- A_i has length s_i
- A_i has weight w_i
- an x_i −fraction of A_i, where $0 \leq x_i \leq 1$, has length $x_i s_i$ and weight $x_i w_i$

This is similar to the knapsack problem met earlier.

- given n objects $A_1, A_2, ..., A_n$
- given a knapsack of length S
- A_i has length s_i
- A_i has weight w_i
- an x_i −fraction of A_i, where $0 \leq x_i \leq 1$, has length $x_i s_i$ and weight $x_i w_i$

This is similar to the knapsack problem met earlier.

- given n objects $A_1, A_2, ..., A_n$
- given a knapsack of length S
- A_i has length s_i
- A_i has weight w_i
- an x_i −fraction of A_i, where $0 \leq x_i \leq 1$, has length $x_i s_i$ and weight $x_i w_i$

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

$S = 20$.

$s_1 = 18, s_2 = 10, s_3 = 15$.

$w_1 = 25, w_2 = 15, w_3 = 24$.

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

$S = 20.$

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

Fill the knapsack as full as possible using fractional parts of the objects, so that the weight is minimized.

Example

S = 20.

$s_1 = 18, s_2 = 10, s_3 = 15.$

$w_1 = 25, w_2 = 15, w_3 = 24.$

(x_1, x_2, x_3)	$\sum_{i=1}^{3} x_i s_i$	$\sum_{i=1}^{3} x_i w_i$
(1,1/5,0)	20	28
(1,0,2/15)	20	28.2
(0,1,2/3)	20	31
(0,1/2,1)	20	31.5

The first one is the best so far. But is it the best overall?

The Greedy Solution

Define the density of object A_i to be w_i/s_i. Use as much of low density objects as possible (i.e. process each in increasing order of density).

If the whole thing fits, use all of it.

If not, fill the remaining space with a fraction of the current object, and discard the rest.

The Greedy Solution

Define the density of object A_i to be w_i/s_i. Use as much of low density objects as possible (i.e. process each in increasing order of density).

If the whole thing fits, use all of it.

If not, fill the remaining space with a fraction of the current object, and discard the rest.

The Greedy Solution

Define the density of object A_i to be w_i/s_i. Use as much of low density objects as possible (i.e. process each in increasing order of density).

If the whole thing fits, use all of it.

If not, fill the remaining space with a fraction of the current object, and discard the rest.

The Greedy Solution

Define the density of object A_i to be w_i/s_i. Use as much of low density objects as possible (i.e. process each in increasing order of density).

If the whole thing fits, use all of it.

If not, fill the remaining space with a fraction of the current object, and discard the rest.

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

First, sort the objects in non decreasing density,so that $w_i/s_i \leq w_{i+1}/s_{i+1}$ for $1 \leq i < n$.

Then, do the following:

$s := S; i := 1$

while $s_i \leq s$ *do*

$x_i := 1$

$s := s - s_i$

$i := i + 1$

$x_i := s/s_i$

for $j := i + 1$ to n do

$x_j := 0$

$O(n \log n)$ for sorting

$O(n)$ for the rest

$O(n \log n)$ for sorting

$O(n)$ for the rest

All the best for your exam!