AMT 211 2.0 Design and Analysis of Algorithms

Dr. G.H.J. Lanel

Semester 1 - 2017

Outline

1. Mathematics for the Analysis of Algorithms
   - Binomial Identities
   - Recurrence Relations
   - Asymptotic Analysis
2. Introduction to Algorithm Design, Validation and Analysis
3. Analysis of Algorithms
   - Best, Average, and Worst Case Running Times
   - Big O, Little o, Ω (Omega), and Θ (Theta) Notations
   - Complexity classes
4. Time Complexity of Searching and Shorting Algorithms
   - Sequential and Binary Search Algorithms
   - Sorting Algorithms
   - Breadth First and Depth First Search Algorithms
5. Algorithm Design Strategies
   - Divide and Conquer Techniques
   - Dynamic Programming
   - Greedy Algorithms

Outline

1. Mathematics for the Analysis of Algorithms
   - Binomial Identities
   - Recurrence Relations
   - Asymptotic Analysis
2. Introduction to Algorithm Design, Validation and Analysis
3. Analysis of Algorithms
   - Best, Average, and Worst Case Running Times
   - Big O, Little o, Ω (Omega), and Θ (Theta) Notations
   - Complexity classes
4. Time Complexity of Searching and Shorting Algorithms
   - Sequential and Binary Search Algorithms
   - Sorting Algorithms
   - Breadth First and Depth First Search Algorithms
5. Algorithm Design Strategies
   - Divide and Conquer Techniques
   - Dynamic Programming
   - Greedy Algorithms

What is the Binomial Identity?

In general, a binomial identity is a formula expressing products of factors as a sum over terms, each including a binomial coefficient $\binom{n}{k}$.

The prototypical example is the binomial theorem;

For $n > 0$

$$(x + a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

What is the Binomial Identity?

In general, a binomial identity is a formula expressing products of factors as a sum over terms, each including a binomial coefficient $\binom{n}{k}$.

The prototypical example is the binomial theorem;

For $n > 0$

$$(x + a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

Many algorithms, particularly divide and conquer algorithms, have time complexities which are naturally modeled by recurrence relations.

A recurrence relation is an equation which is defined in terms of itself.

Why are recurrences good things?

1. Many natural functions are easily expressed as recurrences:

   $a_n = a_{n-1} + 1, a_1 = 1 \rightarrow a_n = n$ (polynomial)

   $a_n = 2 * a_{n-1}, a_1 = 1 \rightarrow a_n = 2^{n-1}$ (exponential)

   $a_n = n * a_{n-1}, a_1 = 1 \rightarrow a_n = n!$ (weird function)
2. It is often easy to find a recurrence as the solution of a counting problem. Solving the recurrence can be done for many special cases as we will see, although it is somewhat of an art.

Many algorithms, particularly divide and conquer algorithms, have time complexities which are naturally modeled by recurrence relations.

A recurrence relation is an equation which is defined in terms of itself.

Why are recurrences good things?

1. Many natural functions are easily expressed as recurrences:

   $a_n = a_{n-1} + 1, a_1 = 1 \rightarrow a_n = n$ (polynomial)

   $a_n = 2 * a_{n-1}, a_1 = 1 \rightarrow a_n = 2^{n-1}$ (exponential)

   $a_n = n * a_{n-1}, a_1 = 1 \rightarrow a_n = n!$ (weird function)
2. It is often easy to find a recurrence as the solution of a counting problem. Solving the recurrence can be done for many special cases as we will see, although it is somewhat of an art.

Recursion is Mathematical Induction!

In both we have general and boundary conditions, with the general condition breaking the problem into smaller and smaller pieces.

The initial or boundary condition terminate the recursion.

As we will see, induction provides a useful tool to solve recurrences- guess a solution and prove it by induction.

Example:

$T_n = 2 * T_{n-1} + 1, T_0 = 0$

n	0	1	2	3	4	5	6	7
T_n	0	1	3	7	15	31	63	127

Guess what the solution is?

Prove $T_n = 2^n - 1$ by induction:

Solution:

1. Show that the basis is true: $T_0 = 2^0 - 1 = 0$.
2. Now assume true for T_{n-1}.
3. Using this assumption show:

$$T_n = 2 * T_{n-1} + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$$

Solving Recurrences

No general procedure for solving recurrence relations is known which is why it is an art.

Asymptotic Analysis

1. The asymptotic analysis of an algorithm determines the running time in big-Oh notation.
2. To perform the asymptotic analysis
    1. We find the worst-case number of primitive operations executed as a function of the input size.
    2. We express this function with big-Oh notation.

Example of Asymptotic Analysis

An algorithm for computing prefix averages

Algorithm prefixAverages1(X):

Input: An n-element array X of numbers.

Output: An n -element array A of numbers such that A[i] is the average of elements X[0], ... , X[i].

```
Let A be an array of n numbers.
for i ← 0 to n - 1 do
    a ← 0
    for j ← 0 to i do
      a ← a + X[j]
    A[i] ← a/(i+ 1)
return array A
```

1 step

i iterations with i=0,1,2...n-1

n iterations

Algorithm prefixAverages1(X) "runs in $O(n)$ time"

Asymptotic Notation(Terminology)

1. Special classes of algorithms:

   logarithmic: $O(logn)$
   linear: $O(n)$
   quadratic: $O(n^2)$
   polynomial: $O(n^k), k \geq 1$
   exponential: $O(a^n), n > 1$

2. "Relatives" of the Big-Oh
   1. $\Omega(f(n))$: Big Omega–asymptotic lower bound
   2. $\Theta(f(n))$: Big Theta–asymptotic tight bound

Asymptotic Analysis of The Running Time

1. Use the Big-Oh notation to express the number of primitive operations executed as a function of the input size.
2. For example, we say that the **prefixAverages** algorithm runs in $O(n)$ time.
3. Comparing the asymptotic running time
   1. an algorithm that runs in $O(n)$ time is better than one that runs in $O(n^2)$ time.
   2. similarly, $O(logn)$ is better than $O(n)$.
   3. hierarchy of functions: $logn \leq n \leq n^2 \leq n^3 \leq 2n$.

Caution! Beware of very large constant factors.

An algorithm running in time $1,000,000$ n is still $O(n)$ but might be less efficient on your data set than one running in time $2n^2$, which is $O\left(n^2\right)$.

Outline

1. Mathematics for the Analysis of Algorithms
   - Binomial Identities
   - Recurrence Relations
   - Asymptotic Analysis
2. Introduction to Algorithm Design, Validation and Analysis
3. Analysis of Algorithms
   - Best, Average, and Worst Case Running Times
   - Big O, Little o, Ω (Omega), and Θ (Theta) Notations
   - Complexity classes
4. Time Complexity of Searching and Shorting Algorithms
   - Sequential and Binary Search Algorithms
   - Sorting Algorithms
   - Breadth First and Depth First Search Algorithms
5. Algorithm Design Strategies
   - Divide and Conquer Techniques
   - Dynamic Programming
   - Greedy Algorithms

Algorithm

- An algorithm is a process to solve a problem manually in sequence with finite number of steps.
- It is a set of rules that must be followed when solving a specific problem.
- It is a well-defined computational procedure which takes some value or set of values as input and generates some set of values as output.
- So,an algorithm is defined as a finite sequence of computational steps, that transforms to given input into the output for a given problem.

- An algorithm is considered to be correct, if for every input instance, it generates the correct output and gets terminated.
- So, a correct algorithm solves a given computational problem and gives the desired output.
- The main objectives of algorithm are
  - To solve a problem manually in sequence with finite number of steps.
  - For designing an algorithm, we need to construct an efficient solution for a problem.

Algorithm Paradigm

It includes four steps.That is:

1. Design of Algorithm
2. Algorithm validation
3. Analysis of algorithms
4. Algorithm testing

1. Design of algorithm:

- Various designing techniques are available which yield good and useful algorithm.
- These techniques are not only applicable to only computer science, but also to other areas, such as operation research and electrical engineering.
- The techniques are: divide and conquer, incremental approach, dynamic programming...etc. By studying this we can formulate good algorithm.

2. Algorithm validation:

- Algorithm validation checks the algorithm result for all legal set of input.
- After designing, it is necessary to check the algorithm, whether it computes the correct and desired result or not for all possible legal set of input.
- Here the algorithm is not converted into the program.But after showing the validity of the method, a program is written.This is known as ”program providing” or ” program verification”.
- Here we check the program output for all possible set of input.
- It requires that, each statement should be precisely defined an all basic operations can be correctly provided.

3. Analysis of algorithms:

- The analysis of algorithm focuses on time complexity or space complexity.
- The amount of memory needed by program to run to completion is referred to as space complexity.
- The amount of time needed by an algorithm to run to completion is referred to as time complexity.
- For an algorithm time complexity depends upon the size of the input, thus is a function of input size ’n’.

- Usually, we deal with the best case time, average case time and worst case time for an algorithm.
- The minimum amount of time that an algorithm requires for an input size 'n', is referred to as Best Case Time Complexity.
- Average Case Time Complexity is the execution of an algorithm having typical input data of size 'n'.
- The maximum amount of time needed by an algorithm for an input size 'n' is referred to as Worst Case Time Complexity.

4. Algorithm testing:

- This phase involves testing of a program.It consists of two phases.That is: Debugging and Performance Measurement.
- Debugging is the process of finding and correcting the cause at variance with the desired and observed behaviors.
- Debugging can only point to the presence of errors, but not their absence.
- The performance measurement or profiling precise by described the correct program execution for all possible data sets and it takes time and space to compute results.

NOTES

- While designing and analyzing an algorithm, two fundamental issue to be considered.That is:
  1. Correctness of the algorithm
  2. Efficiency of the algorithm
- While designing the algorithm, it should be clear, simple and should be unambiguous.
- The characteristics of algorithm is: finiteness, definiteness, efficiency, input and output.

Outline

1. Mathematics for the Analysis of Algorithms
   - Binomial Identities
   - Recurrence Relations
   - Asymptotic Analysis
2. Introduction to Algorithm Design, Validation and Analysis
3. Analysis of Algorithms
   - Best, Average, and Worst Case Running Times
   - Big O, Little o, Ω (Omega), and Θ (Theta) Notations
   - Complexity classes
4. Time Complexity of Searching and Shorting Algorithms
   - Sequential and Binary Search Algorithms
   - Sorting Algorithms
   - Breadth First and Depth First Search Algorithms
5. Algorithm Design Strategies
   - Divide and Conquer Techniques
   - Dynamic Programming
   - Greedy Algorithms

We can have three cases to analyze an algorithm:

1. Worst Case
2. Average Case
3. Best Case

Let us consider the following implementation of Linear Search.

```
// Linearly search x in arr[].  If x is present then return the index,
// otherwise return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i=0; i<n; i++)
    {
       if (arr[i] == x)
         return i;
    }
    return -1;
}

/* Driver program to test above functions*/
int main()
{
    int arr[] = {1, 10, 30, 15};
    int x = 30;
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("%d is present at index %d", x, search(arr, n, x));

    getchar();
    return 0;
}
```

Worst Case Analysis (Usually Done)

- In the worst case analysis, we calculate upper bound on running time of an algorithm.
- We must know the case that causes maximum number of operations to be executed.
- For Linear Search, the worst case happens when the element to be searched (x in the previous code) is not present in the array.
- When x is not present, the search() functions compares it with all the elements of arr[] one by one.
- Therefore, the worst case time complexity of linear search would be $\theta(n)$.

Average Case Analysis (Sometimes done)

- In average case analysis, we take all possible inputs and calculate computing time for all of the inputs.
- Sum all the calculated values and divide the sum by total number of inputs.
- We must know distribution of cases.
- For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array).
- So we sum all the cases and divide the sum by (n+1).

- Following is the value of average case time complexity.

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{\theta((n+1)*(n+2)/2)}{(n+1)}$$

$$= \Theta(n)$$

Best Case Analysis (Bogus)

- In the best case analysis, we calculate lower bound on running time of an algorithm.
- We must know the case that causes minimum number of operations to be executed.
- In the linear search problem, the best case occurs when x is present at the first location.
- The number of operations in the best case is constant (not dependent on n).
- So time complexity in the best case would be $\theta(1)$

- Most of the times, we do **worst case analysis** to analyze algorithms. In the worst case analysis, we guarantee an upper bound on the running time of an algorithm which is good information.
- The **average case analysis** is not easy to do in most of the practical cases and it is rarely done. In the average case analysis, we must know the mathematical distribution of all possible inputs.
- The **best case analysis** is bogus. Guaranteeing a lower bound on an algorithm doesnt provide any information as in the worst case, an algorithm may take years to run.

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

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

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

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

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

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 $f \in O(g)$

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

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

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

$$= 1 > 0$$

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. 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 $x \geq 3$

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

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 $x \geq 3$

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

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 $x \geq 3$

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

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 $x \geq 3$

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

$$\geq \left|x^2 + 10\right|$$

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. 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

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

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

$$= |log_a b|$$

$$> 0$$

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, $log_a x \in O(log_b x)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\lim_{x\to\infty} \frac{x^n}{a^x} = \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x}$$

$$= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x}$$

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

$$= 0$$

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

$$\lim_{x\to\infty} \frac{x^n}{a^x} = \frac{n(n-1)(n-2)}{ln(a)^3} \lim_{x\to\infty} \frac{x^{n-3}}{a^x}$$

$$= \frac{n(n-1)(n-2)...(1)}{ln(a)^n} \lim_{x\to\infty} \frac{x^0}{a^x}$$

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

$$= 0$$

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

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

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,

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

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

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

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,

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

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 $\frac{x^4}{x^3+1} \in O(x)$

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

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

- $Big - \Omega$ 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

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

- $Big - \Omega$ 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

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

- $Big - \Omega$ 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

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

- $Big - \Omega$ 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

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

- $Big - \Omega$ 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,

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

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

$$= 0$$

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. 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

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

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

$$\left|2x^3 + x^2 - 3x + 2\right| = 2x^3 + x^2 - 3x + 2$$

$$\geq 2x^3$$

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

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

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

$$\left|2x^3 + x^2 - 3x + 2\right| = 2x^3 + x^2 - 3x + 2$$

$$\geq 2x^3$$

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

$$\left|2x^3 + x^2 - 3x + 2\right| = 2x^3 + x^2 - 3x + 2$$

$$\geq 2x^3$$

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,

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{(2x^2-3)x^2}{3x^4+x^3-2x^2-1}$$

$$= \lim_{x\to\infty} \frac{2-\frac{3}{x^2}}{3+\frac{1}{x}-\frac{2}{x^2}-\frac{1}{x^4}}$$

$$= \frac{2}{3}$$

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 $f \in \Theta(g)$

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, $O(f)$).

What is a Complexity Class

- A complexity class contains a set of problems that take a similar range of space and time to solve.

  For example ”all problems solvable in polynomial time with respect to input size,” ”all problems solvable with exponential space with respect to input size,” and so on.

- Problems are usually proven to be in a particular complexity class by running the problem on an abstract computational model, usually a Turing machine.

- Complexity classes help computer scientists groups problems based on how much time and space they require to solve problems and verify solutions.

  For example, complexity can help describe how many steps it would take a Turing machine to decide a problem A?

- The **time complexity** of an algorithm is usually used when describing the number of steps it needs to take to solve a problem, but it can also be used to describe how long it takes to verify an answer.
- The **space complexity** of an algorithm describes how much memory the algorithm needs in order to operate. In terms of Turing machines, the space needed to solve a problem relates to the number of spaces on the Turing machines tape it needs to do the problem.

Common complexity classes

- ALL is the class of all decision problems. Many important complexity classes can be defined by bounding the time or space used by the algorithm.
- Some important complexity classes of decision problems defined in this manner are the following:

1. **Time-complexity classes** (execution time)
2. **Space-complexity classes** (amount of memory required)

Time-complexity classes

Model of computation	Time constraint $f(n)$	Time constraint poly(n)	Time constraint $2^{\text{poly}(n)}$
Deterministic Turing machine	DTIME	P	EXPTIME
Non-deterministic Turing machine	NTIME	NP	NEXPTIME

Space-complexity classes

Model of computation	Space constraint $f(n)$	Space constraint O(log n)	Space constraint poly(n)	Space constraint $2^{\text{poly}(n)}$
Deterministic Turing machine	DSPACE	L	PSPACE	EXPSPACE
Non-deterministic Turing machine	NSPACE	NL	NPSPACE	NEXPSPACE

Which measure is more important?

Answer often depends on the limitations of the technology available at time of analysis.

For most of the algorithms associated with this course, time complexity comparisons are more interesting than space complexity comparisons

Time Complexity

- Factors that should notaffect time complexity analysis:
  - The programming language chosen to implement the algorithm
  - The quality of the compiler
  - The speed of the computer on which the algorithm is to be executed
- Time complexity analysis for an algorithm is independentof programming language,machine used.
- Objectivesof time complexity analysis:
  - To determine the feasibility of an algorithm by estimating an upper boundon the amount of work performed
  - To compare different algorithms before deciding on which one to implement

Standard Complexity Classes

Tractable vs Intractable Problems:

- We can distinguish problems between two distinct classes.
- Problems which can be solved by a polynomial time algorithm and problems for which no polynomial time algorithm is known.
- An algorithm for a given problem is said to be a polynomial time algorithm if its worst case time complexity is $O(n^k)$, where k is a fixed integer and n is size of a problem.

For example:

- Sequential search: $O(n)$

- The set of all problems that can be solved in polynomial amount of time are called "**Tractable Problems**". These problems can be solved in a reasonable amount of time for even very large amount of input data. Their worst case time complexity is $O(n^k)$.
- The set of all problems that cannot be solved in polynomial amount of time are called "**Intractable Problems**". Their worst case time complexity is $O(k^n)$. These problems require huge amount of time for even modest input sizes.

  For example:

  - 0-1 Knapsack Problem: $O(2^n)$
  - Traveling Salesperson Problem: $O(n^2 2^n)$

Deterministic vs Non-deterministic Algorithms:

- **Deterministic Machines:** Conventional Digital machines are deterministic in nature. Serialization of resource access or sequential execution is the basic concept used in these machines (Von Neumann Architecture).
- **Non-deterministic Machines:** These are hypothetical machines which can do the jobs in paralel fashion i.e. more than one job can be done in one unit of time.

- **Deterministic Algorithms:** Algorithms in which the result of any operation is uniquely defined are termed as Deterministic Algorithms. All algorithms studied so far are deterministic algorithms. Such algorithms agree with the way programs are executed on a digital computer i.e. a deterministic machine.
- **Non-deterministic Algorithms:** If we remove the restriction on the outcome of every operation, then outcomes are not uniquely defined but they are limited to specified set of possibilities. There is a termination condition in such algorithms. Such algorithms are called as non-deterministic algorithms.

Decision Problem Optimization Problem Algorithm:

- **Decision Problem:** Any problem for which answer is either 0 or 1 is called a decision problem and the corresponding algorithm is referred as a decision algorithm. For example: To search a given number
- **Optimization Problem:** Any problem that involves the identification of an optimal (either min. or max.) value of a given cost function is known as an optimization problem and the corresponding algorithm is referred as an optimization algorithm. For example: Knapsack problem, Minimum cost

P vs NP class problems:

- **P class:** The class of decision problems that can be solved in polynomial time using deterministic algorithms is called the P class or Polynomial problems.
- **NP class:** The class of decision problems that can be solved in polynomial time using non-deterministic algorithms is called NP class or Non-deterministic Polynomial problems.
- Any P class problem can be solved using NP class algorithm. Therefore P is contained in NP class.
- Whether Np is contained in NP is unknown.

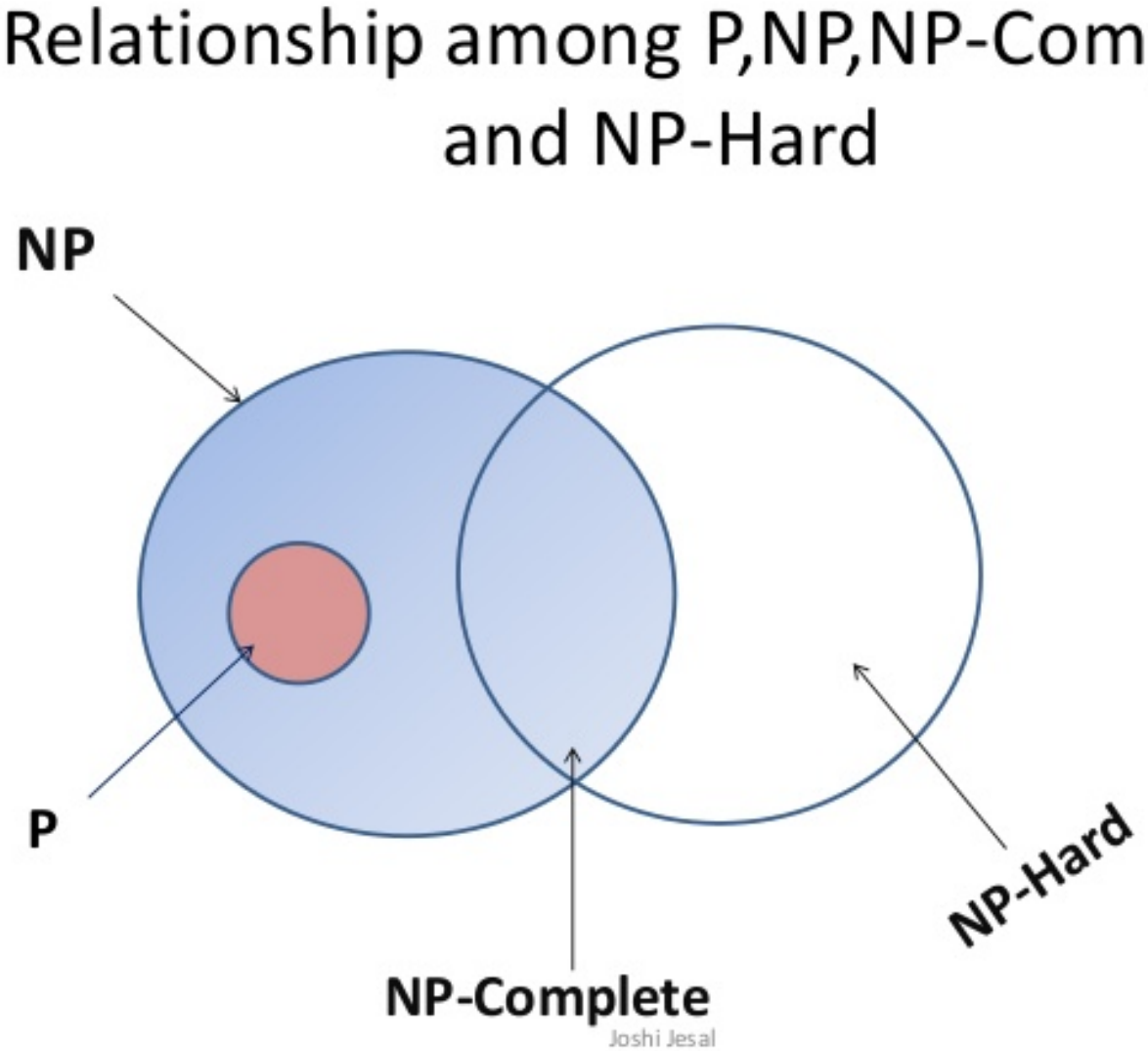
NP-Complete problems:

- A decision problem D is said to be NP-Complete if
  1. It belongs to NP class
  2. Every problem in NP class is polynomially reducible to D
- If one instance of such problem can be solved using a polynomial algorithm, the complete class of problems can be solved using a polynomial algorithm.
- Examples:
  - Traveling Salesperson Problem: optimal tour
  - Printed circuit board problem
  - Bin packing problem
  - 0-1 Knapsack problem
  - Vertex (node) cover problem

NP-Hard problems:

- A problem L is said to be NP Hard problem if and only if satisfiability $\propto L$.
- NP-Hard problems are basically the optimization versions of the problems in NP complete class.
- NP-Hard problems are not mere yes/ no problems. They are p[roblems wherein we need to find the optimal solution.
- A problem L is NP complete if and only if L is NP-Hard and $L \in NP$.

- Commonly believed relationship among P, NP, NP-Complete and NP-Hard



- An example of NP-Hard problem not in NP-Complete

  Halting Problem: To determine for an arbitrary deterministic algorithm A and an input I whether algorithm A with input I ever terminates or enters an infinite loop. This problem is undecidable. No algorithms exists to solve this problem.

Outline

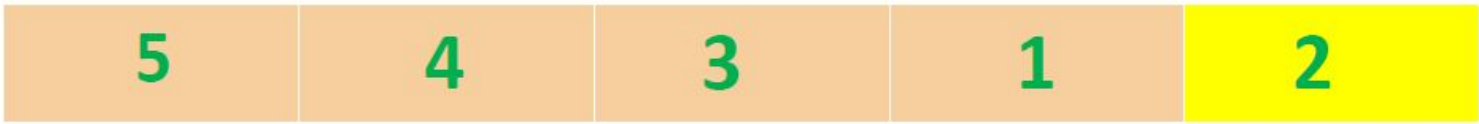
1. Mathematics for the Analysis of Algorithms
   - Binomial Identities
   - Recurrence Relations
   - Asymptotic Analysis
2. Introduction to Algorithm Design, Validation and Analysis
3. Analysis of Algorithms
   - Best, Average, and Worst Case Running Times
   - Big O, Little o, Ω (Omega), and Θ (Theta) Notations
   - Complexity classes
4. Time Complexity of Searching and Shorting Algorithms
   - Sequential and Binary Search Algorithms
   - Sorting Algorithms
   - Breadth First and Depth First Search Algorithms
5. Algorithm Design Strategies
   - Divide and Conquer Techniques
   - Dynamic Programming
   - Greedy Algorithms

Sequential Search/Linear Search

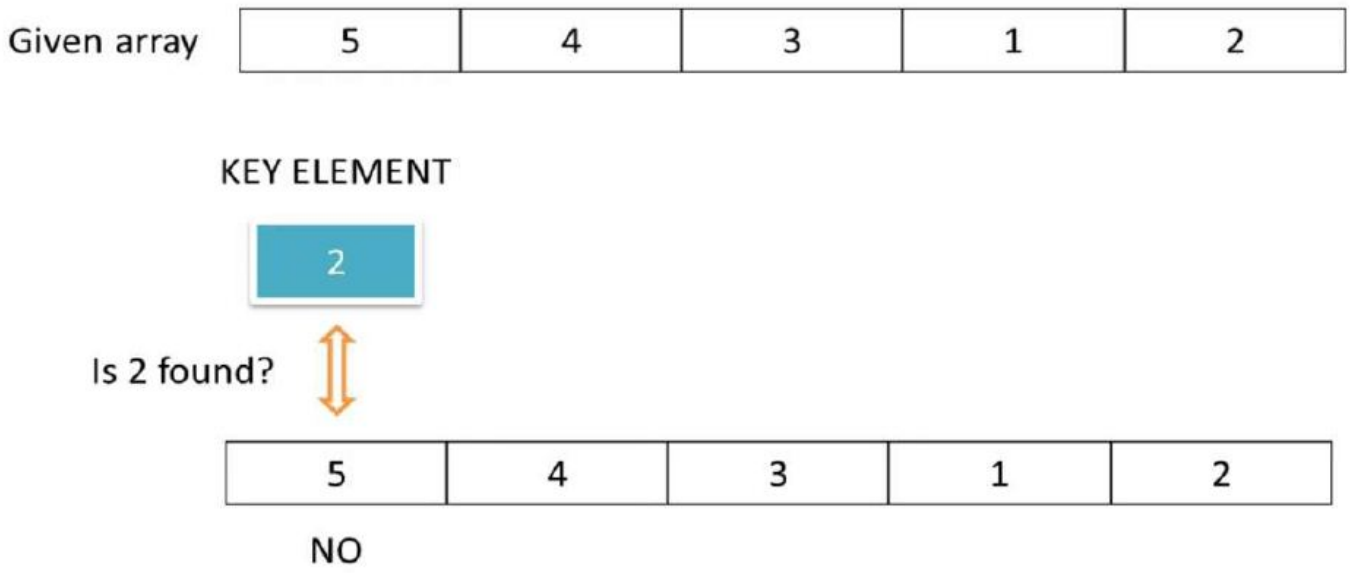
How Sequential Search works?

Linear search or sequential search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

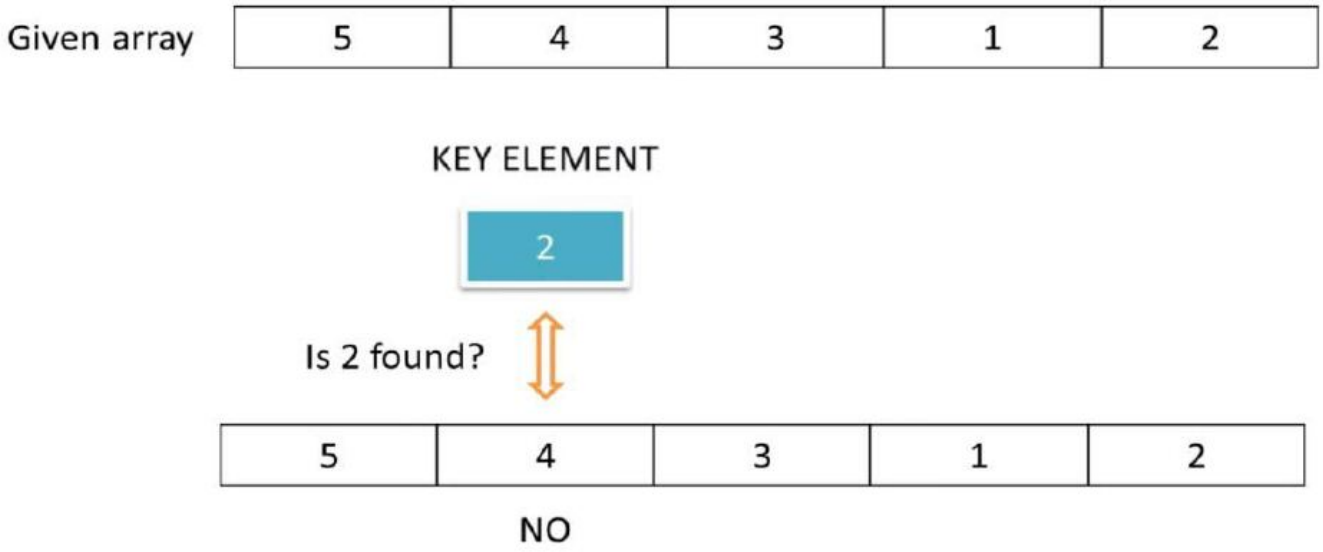
Find Element 2 in this array



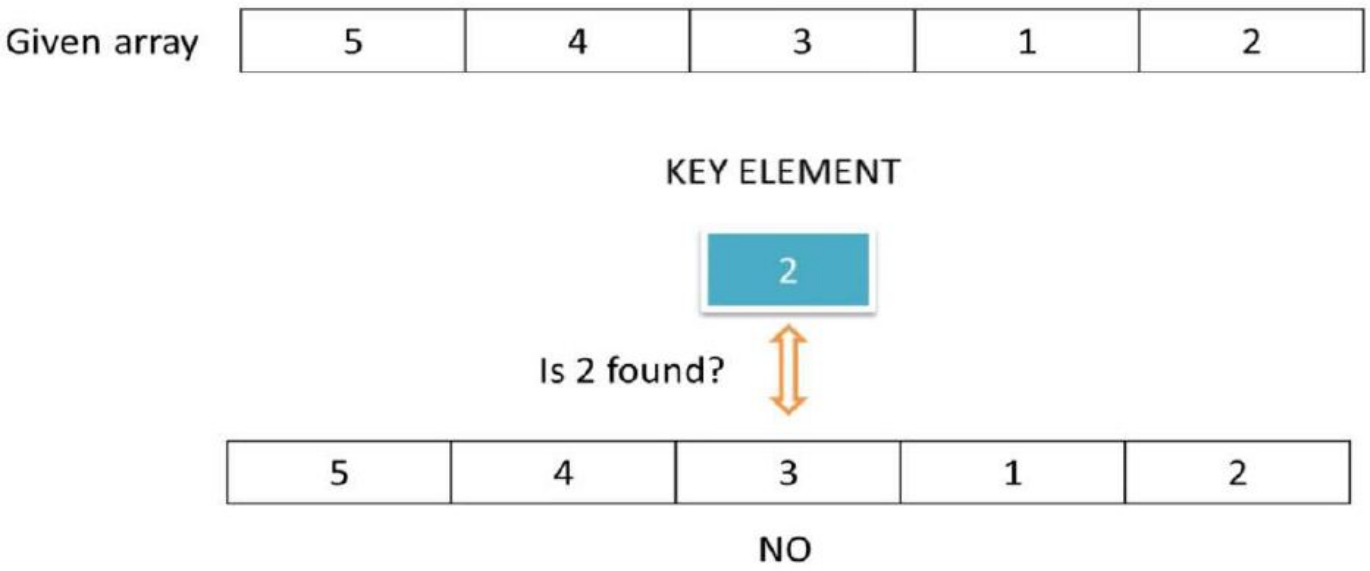
Searching for 2



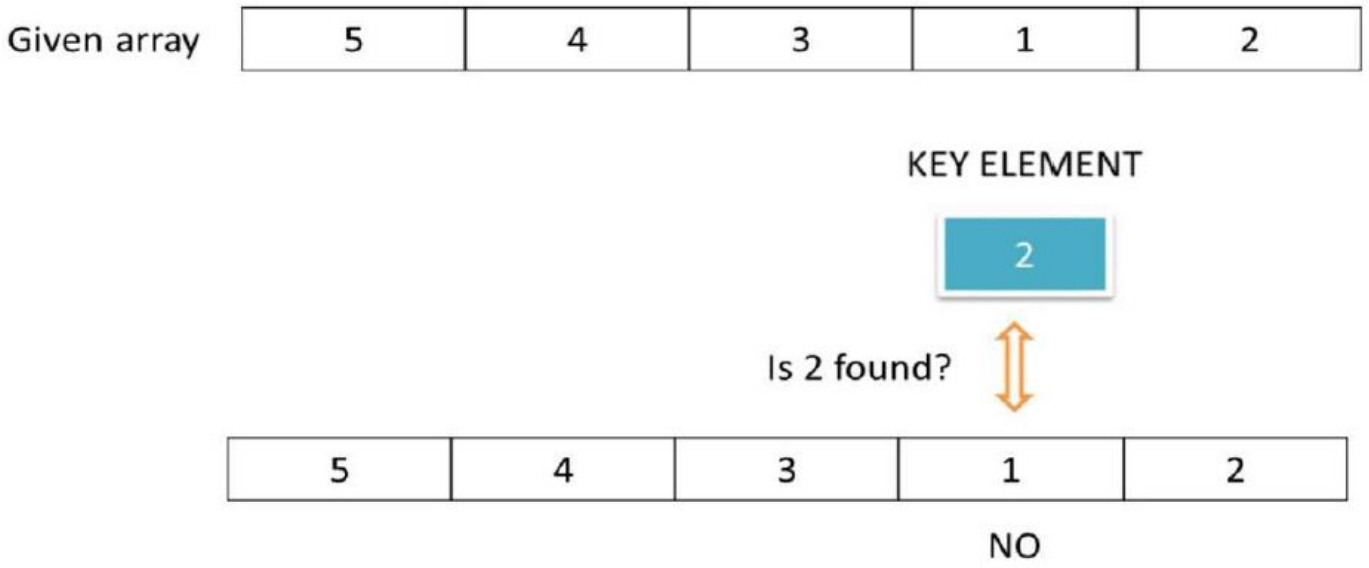
Searching for 2



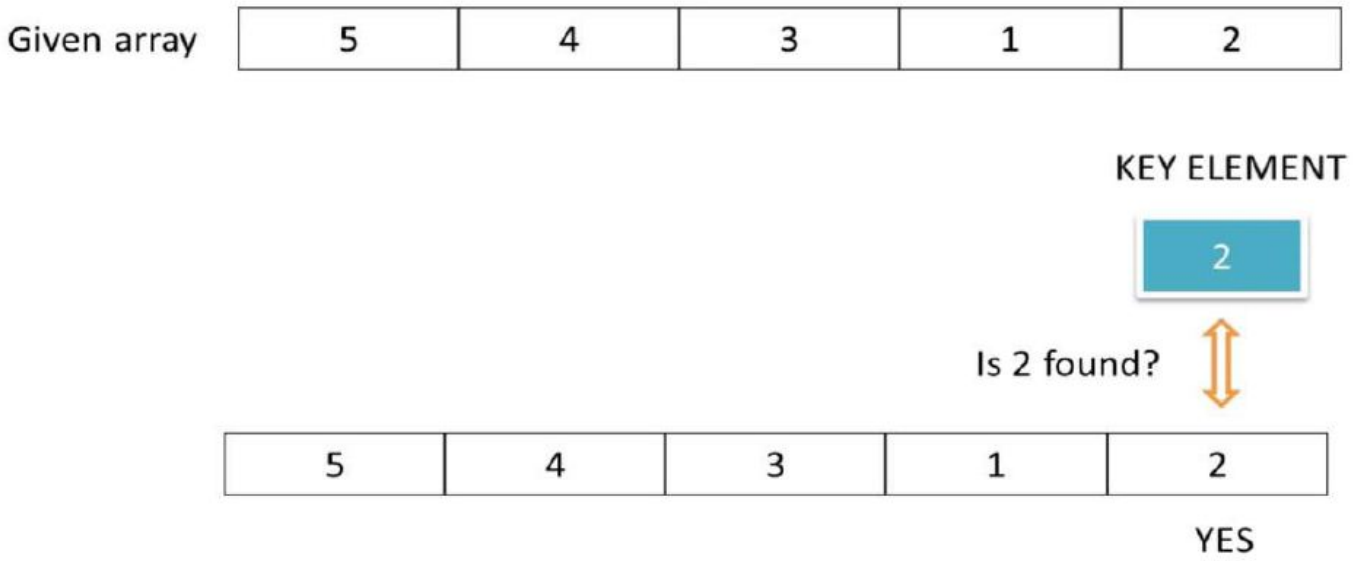
Searching for 2



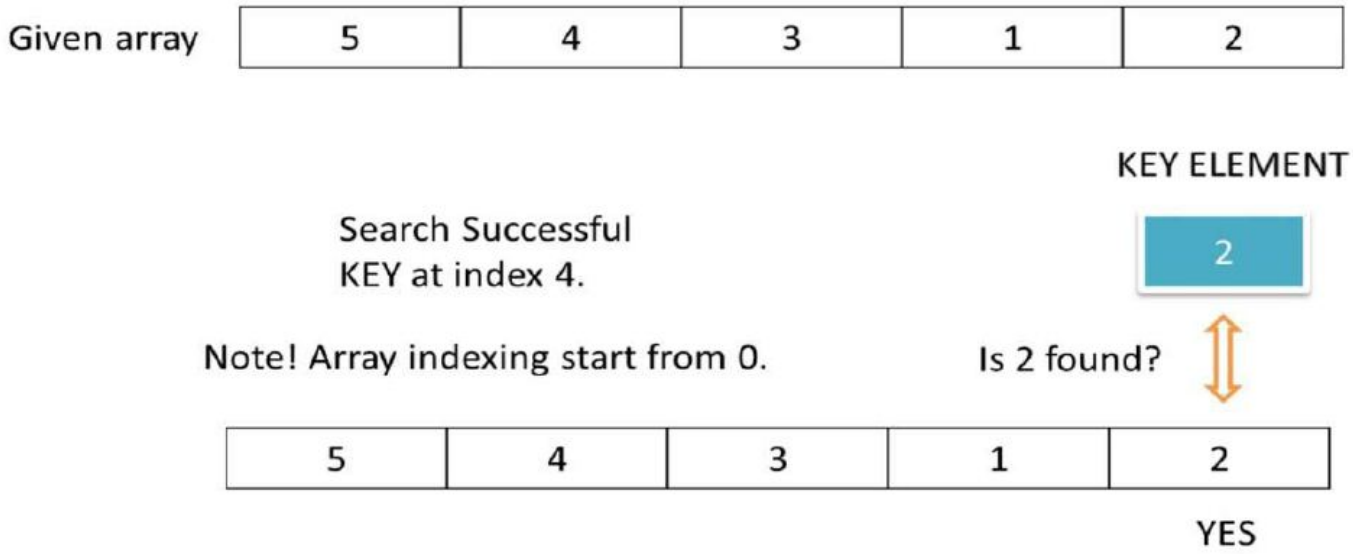
Searching for 2



Searching for 2



Searching for 2

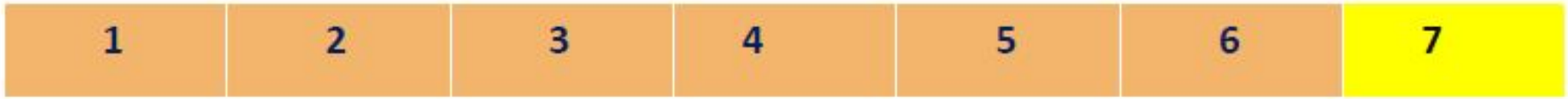


Binary Search

How Binary Search works?

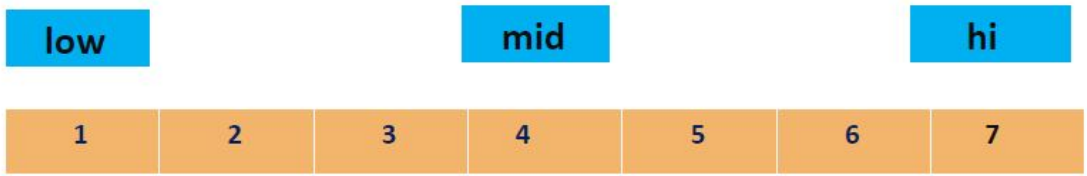
Binary search compares the target value to the middle element of the array; if they are unequal, half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful or the remaining half is empty.

Find Element 7 in this array



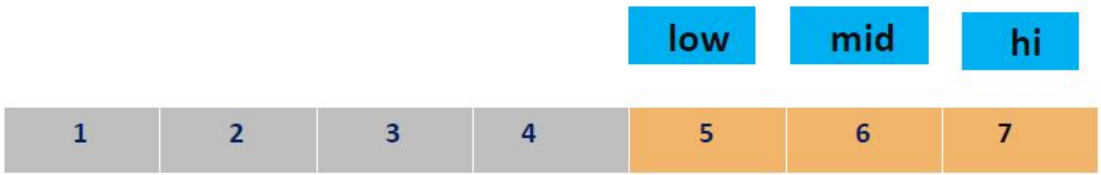
Searching For 7

mid =(low+hi)/2



Key > mid

Searching For 7



Key > mid

Searching For 7

Key = mid

Searching For 7

Key = mid

Sorting Algorithms

- Sorting Algorithm is an algorithm made up of a series of instructions that takes an array as input, and outputs a sorted array.
- There are many sorting algorithms, such as:
  - Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, heap Sort,

    Quick sort, Radix Sort, Counting Sort, Bucket Sort, Shell Sort,

    Comb Sort, Pigeonhole Sort, Cycle Sort.

Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

- Algorithm:
  - **Step 1:** Compare each pair of adjacent elements in the list
  - **Step 2:** Swap two element if necessary
  - **Step 3:** Repeat this process for all the elements until the entire array is sorted.

- Example 1 Assume the following Array:

5	1	4	2

- First Iteration:
- Compare

- First Iteration:
- Swap

- First Iteration:
- Compare

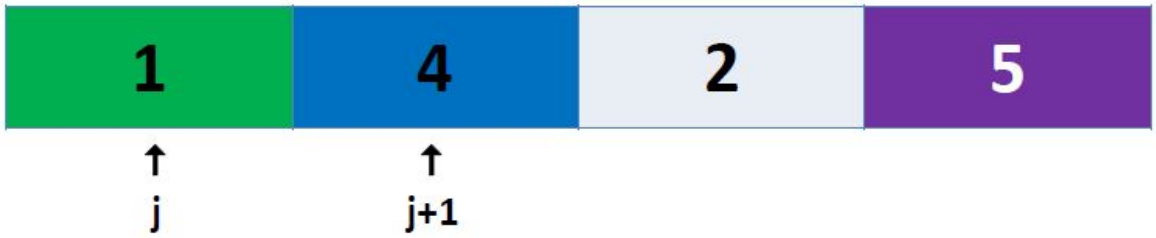
- First Iteration:
- Swap

- First Iteration:
- Compare

- First Iteration:
- Swap

1
4
2
5

- Second Iteration:
- Compare



- Second Iteration:
- Compare

- Second Iteration:
- Swap

1
2
4
5

- Third Iteration:
- Compare

1
2
4
5

- Array is now sorted

Selection Sort

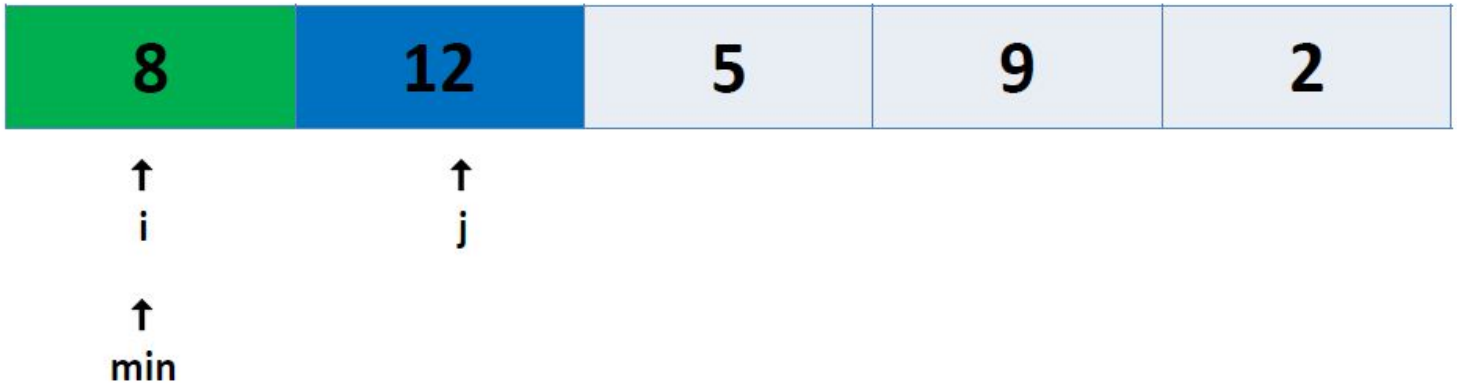
- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

- Algorithm:
  - **Step 1:** Find the minimum value in the list
  - **Step 2:** Swap it with the value in the current position
  - **Step 3:** Repeat this process for all the elements until the entire array is sorted.

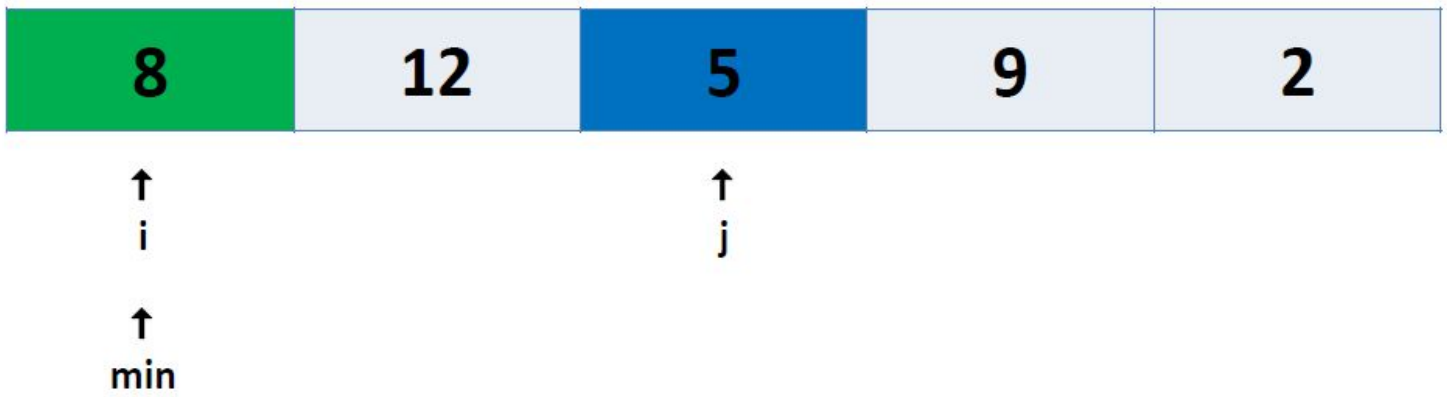
- Example 1 Assume the following Array:

8	12	5	9	2

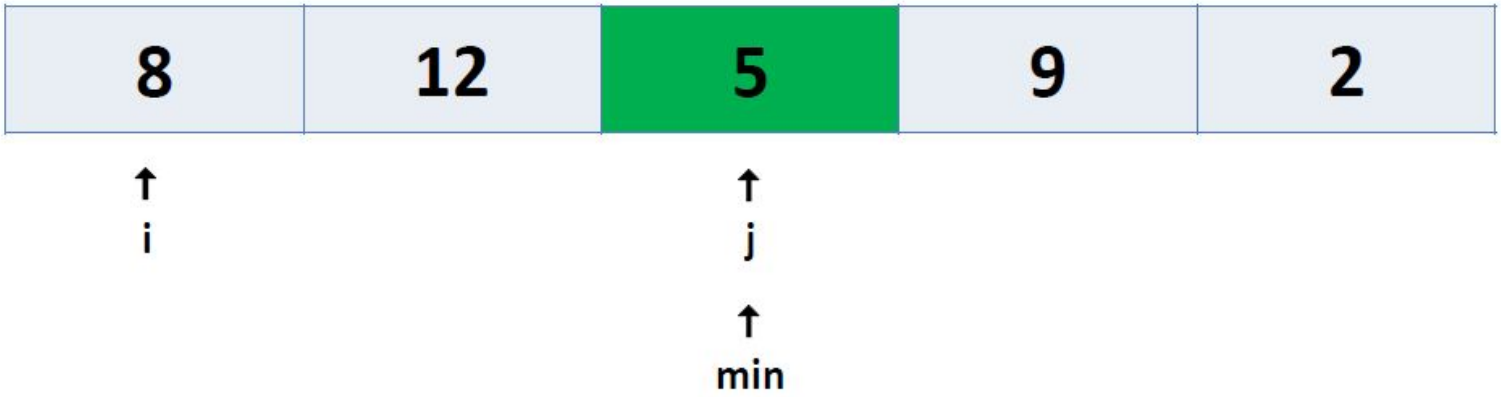
- Compare



- Compare



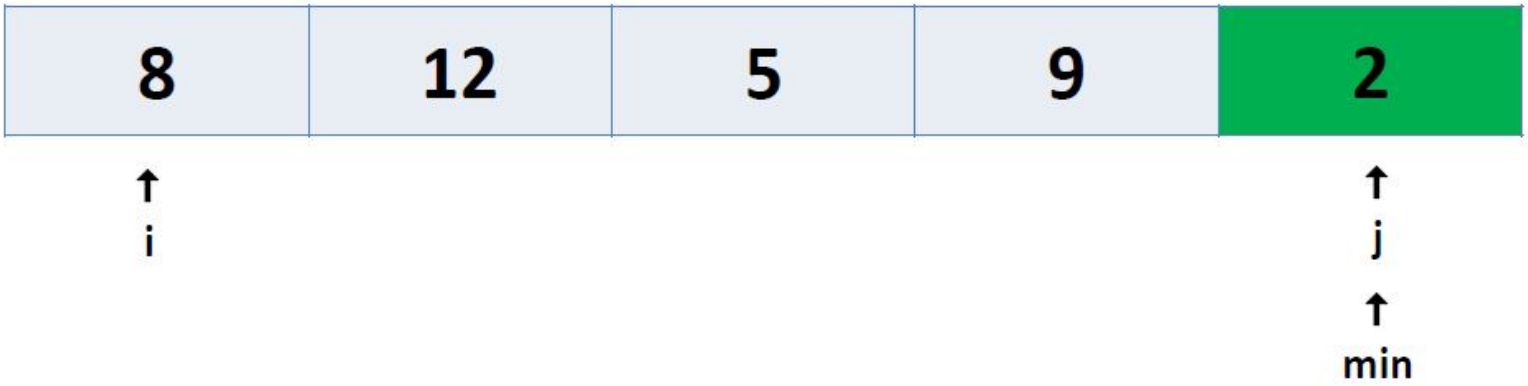
- Move



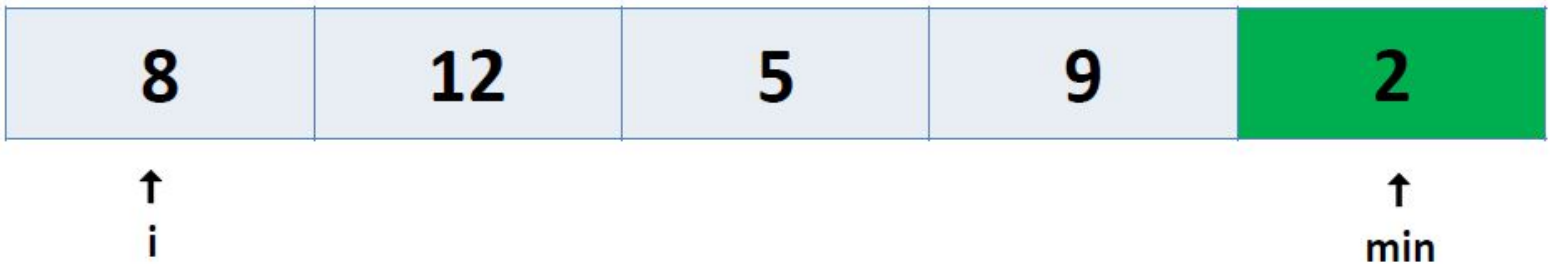
- Compare

- Compare

- Move



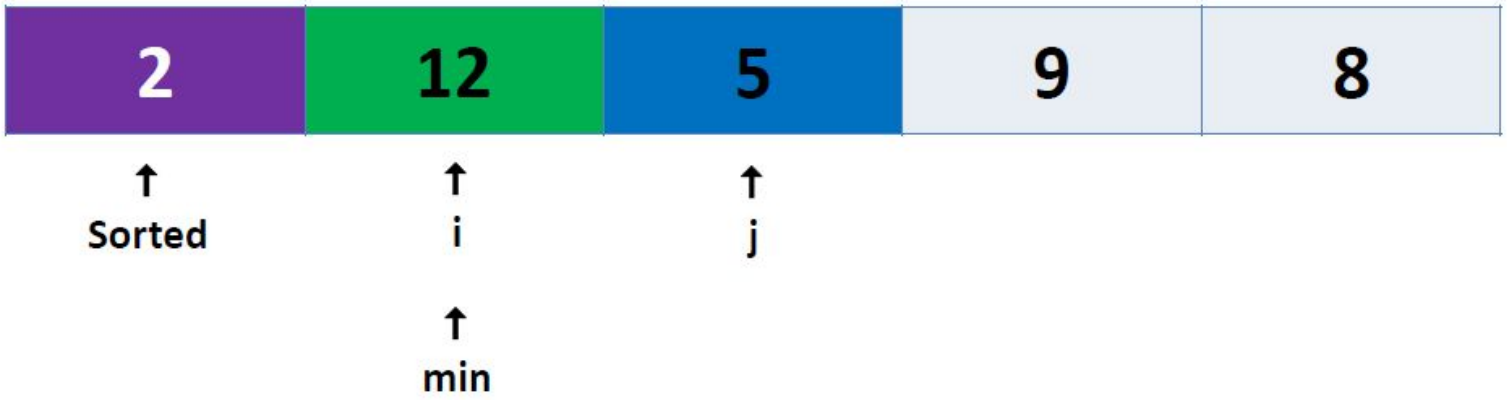
- Smallest



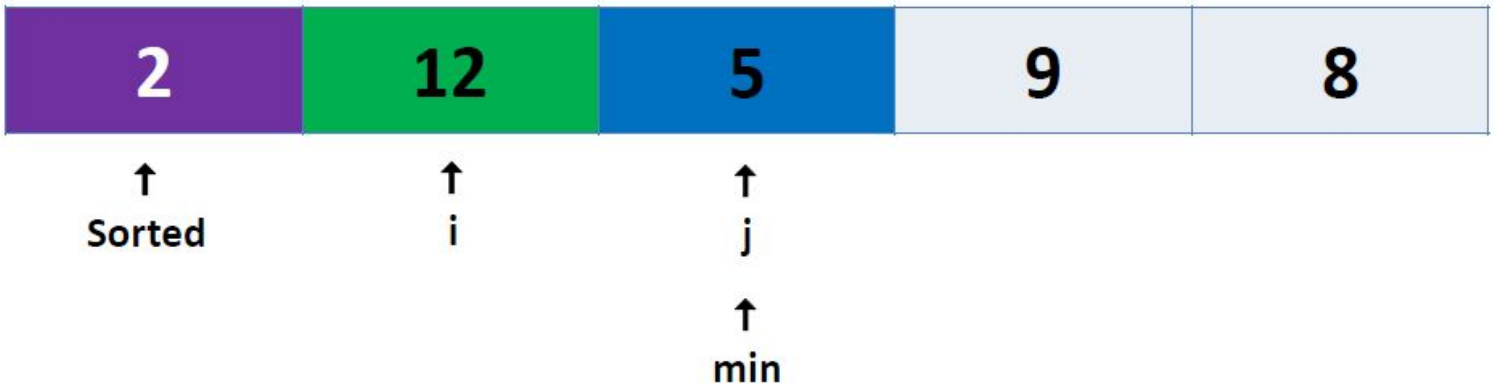
- Swap

- Sorted
- Un Sorted

- Compare



- Move



- Compare

- Compare

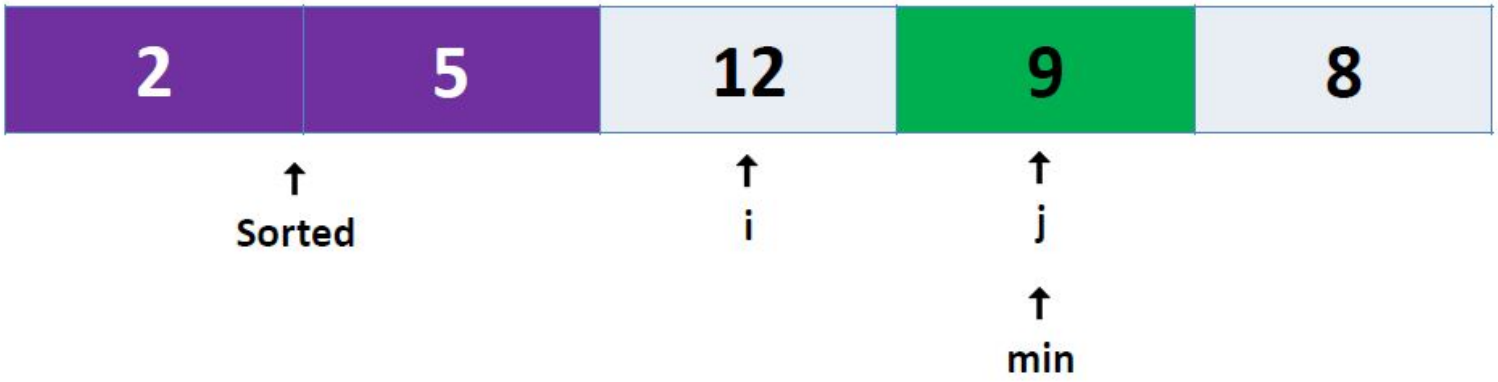
- Smallest

- Swap

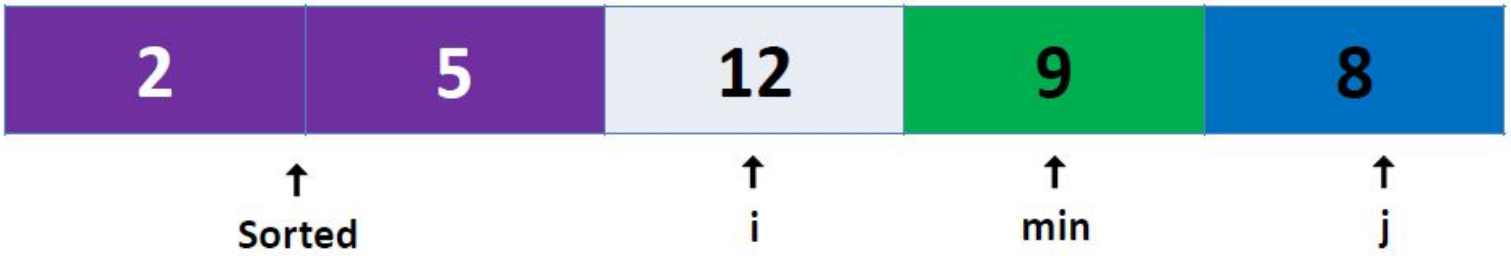
- Sorted
- Un Sorted

- Compare

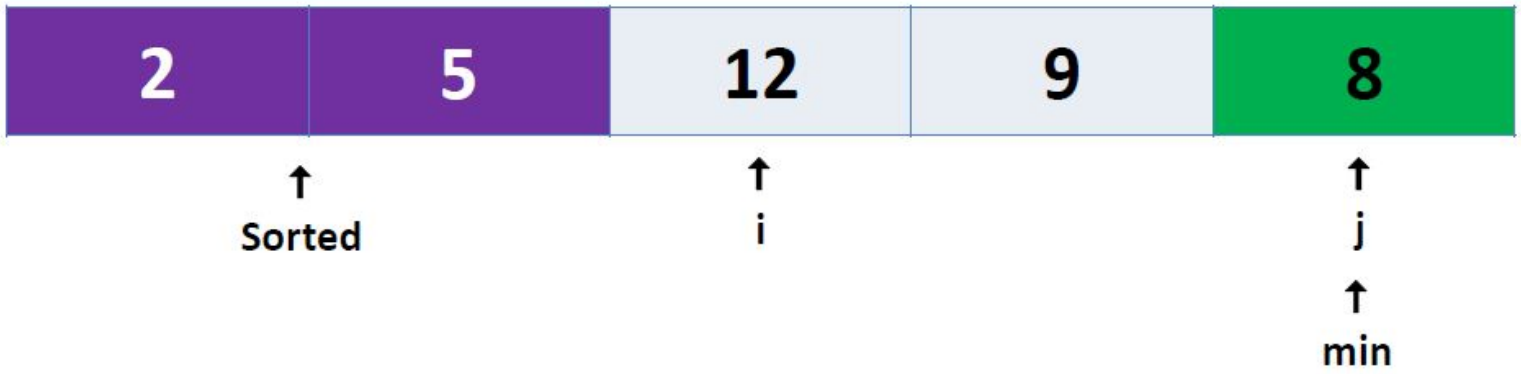
- Move



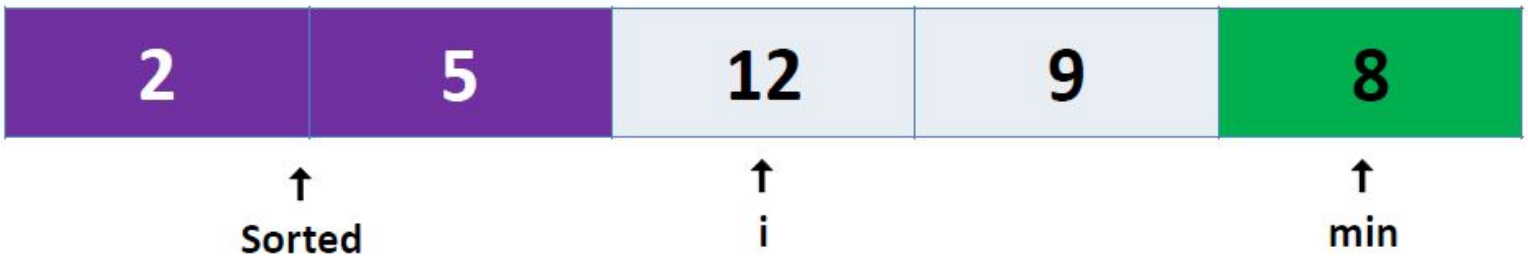
- Compare



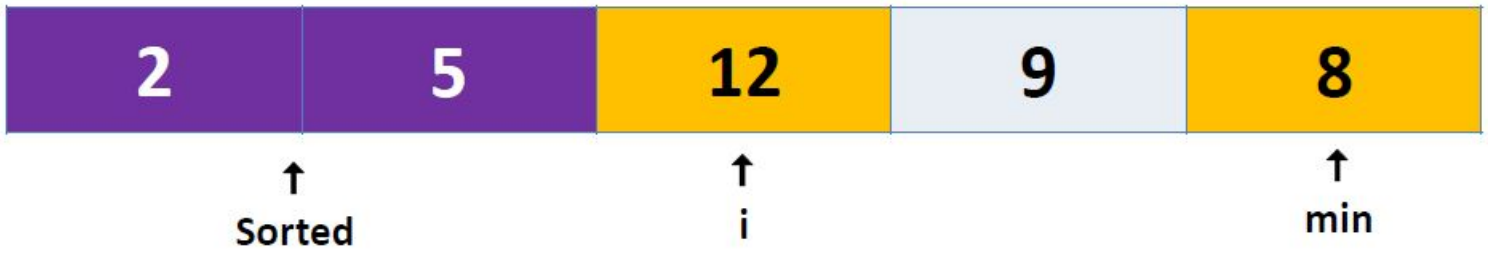
- Move



- Smallest

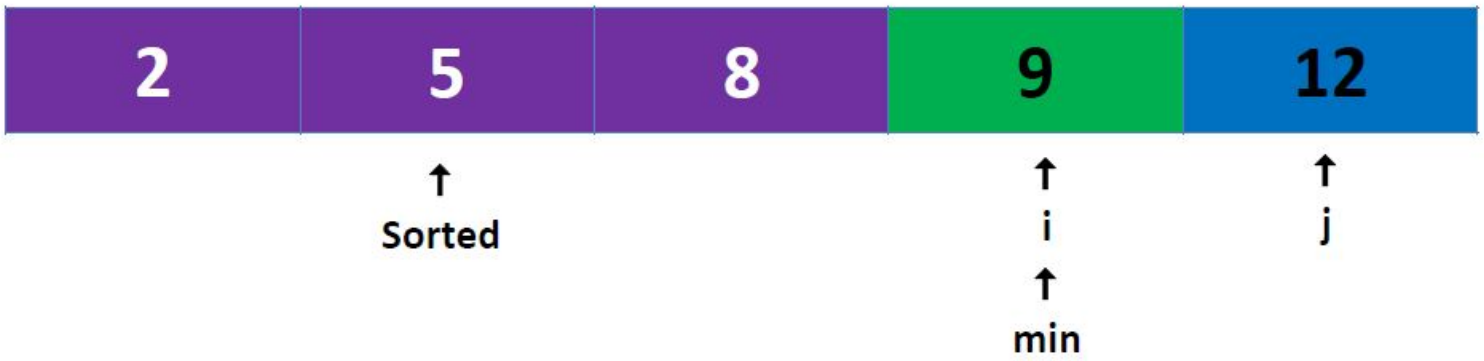


- Swap



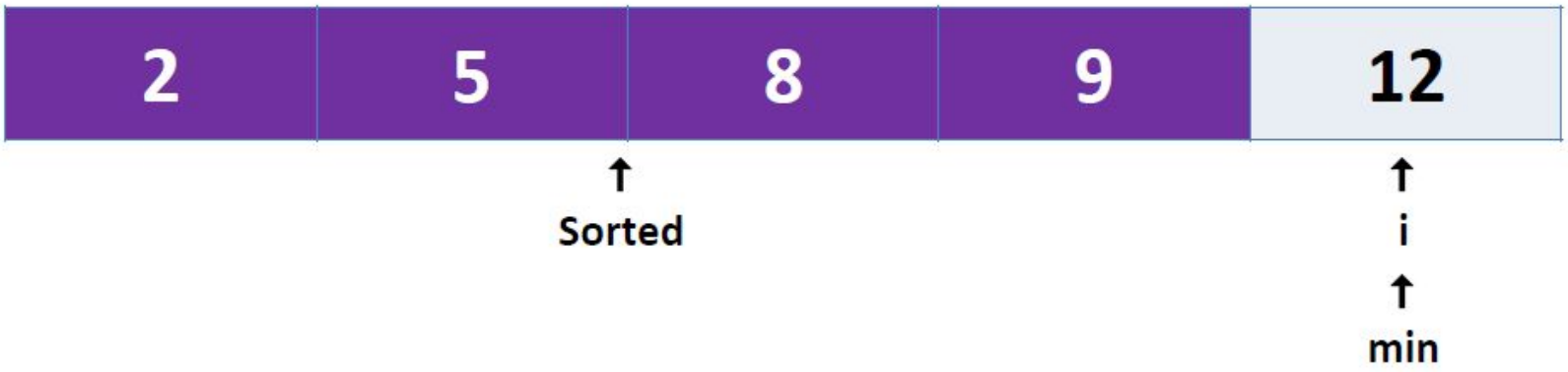
- Sorted
- Un Sorted

- Compare

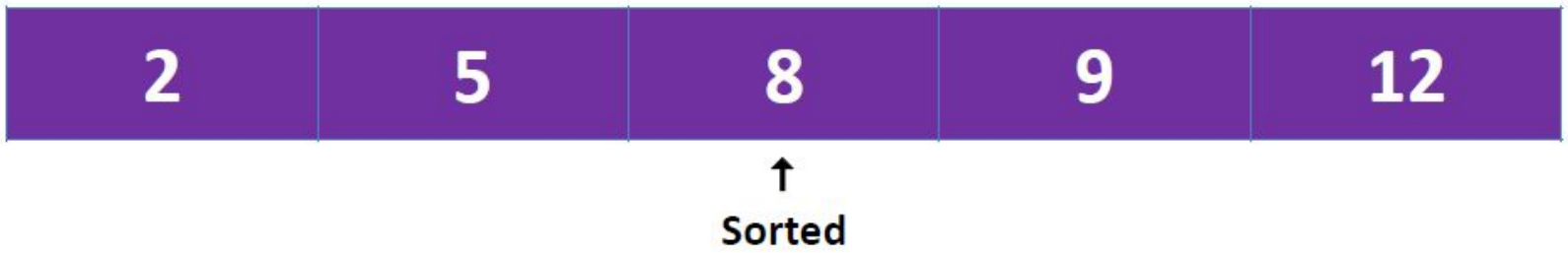


- Sorted
- Un Sorted

- Sorted
- Un Sorted



- Array is now sorted



Spanning trees

There are two different efficient algorithms for finding spanning trees.

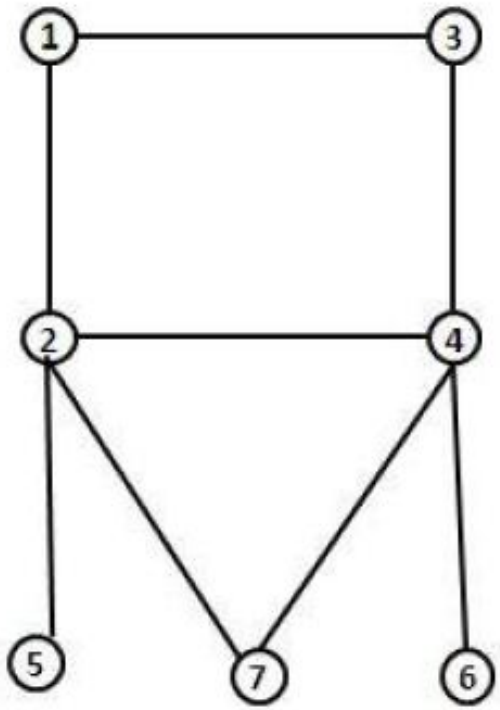
1. Depth first search Algorithm (DFS)
2. Breadth first search Algorithm (BFS)

Depth first search Algorithm (DFS)

Depth first search is a recursive algorithm for visiting all the vertices of a connected graph G.

Depth first search Algorithm (DFS)

Example:



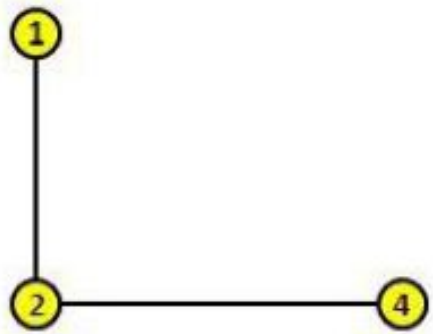
Step 1:

Start at vertex 1.

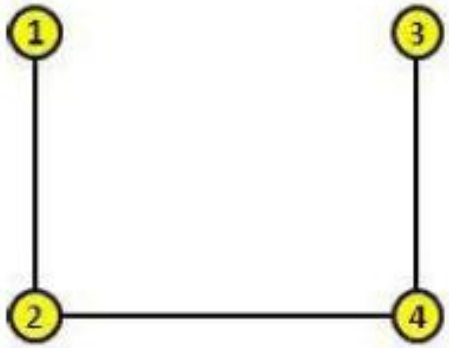
Step 2:

Look at 1's first neighbor nearly vertex 2 has not get been visited we visit it.

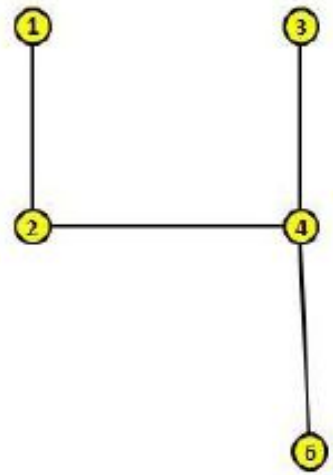
Step 3:



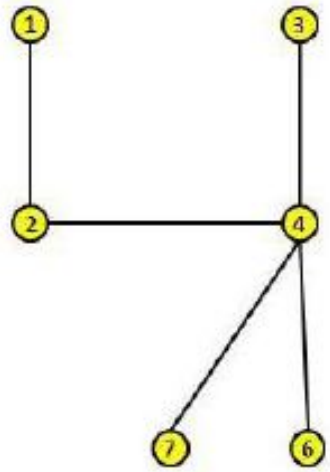
Step 4:



Step 5:

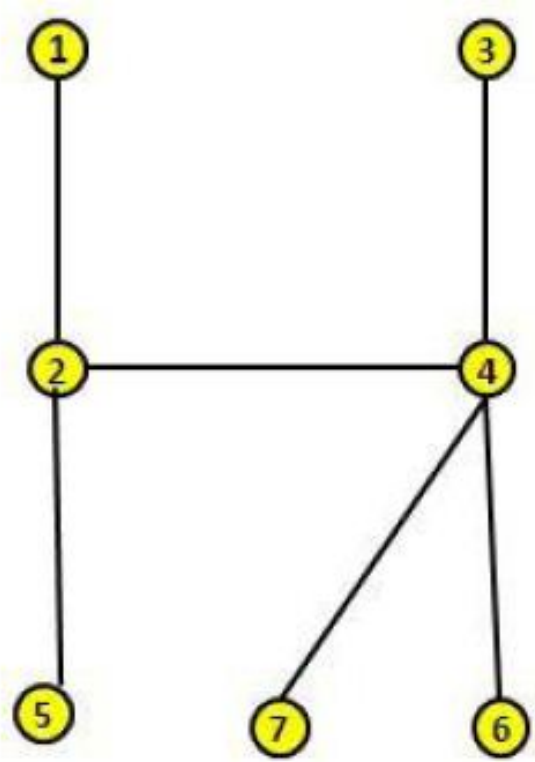


Step 6:



Step 7:

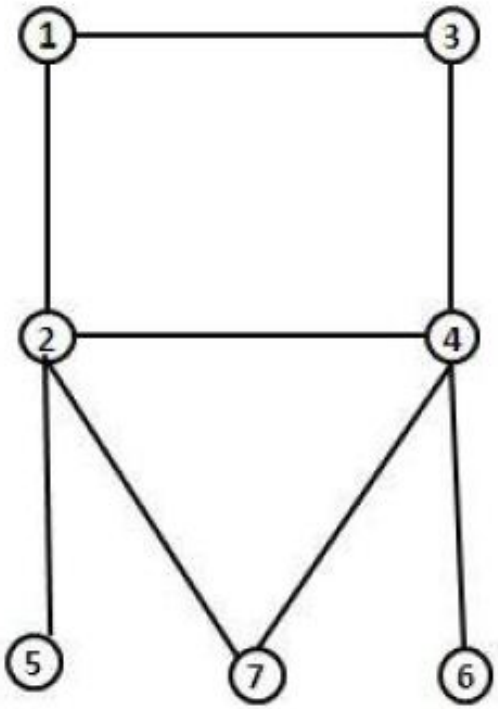
Spanning tree of *G*.



Breadth first search Algorithm (BFS)

Breadth first search Algorithm (BFS)

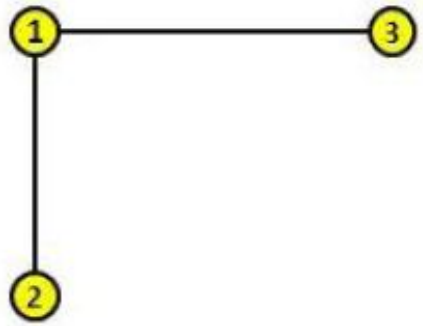
In breadth first search, when are first encounter a vertex, we do not proceed to search further from that vertex immediately. Example:



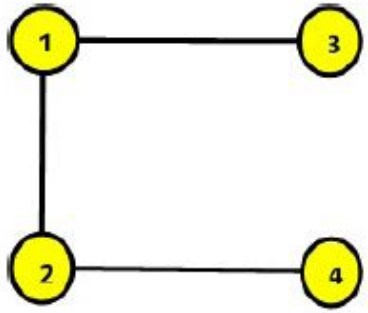
Step 1:

Step 2:

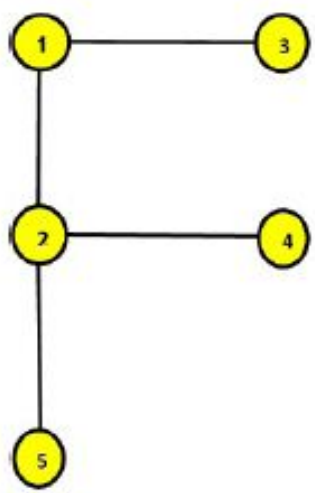
Step 3:



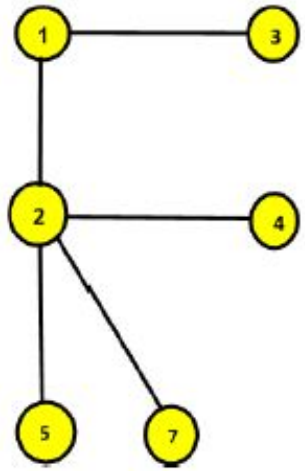
Step 4:



Step 5:

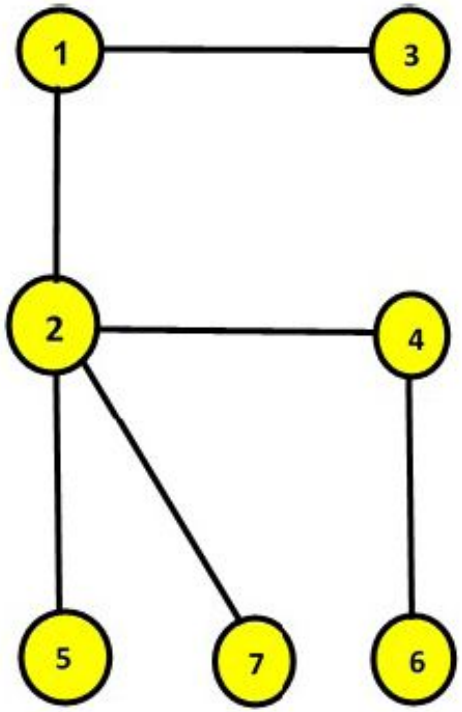


Step 6:



Step 7:

Spanning tree of *G*.



Outline

1. Mathematics for the Analysis of Algorithms
   - Binomial Identities
   - Recurrence Relations
   - Asymptotic Analysis
2. Introduction to Algorithm Design, Validation and Analysis
3. Analysis of Algorithms
   - Best, Average, and Worst Case Running Times
   - Big O, Little o, Ω (Omega), and Θ (Theta) Notations
   - Complexity classes
4. Time Complexity of Searching and Shorting Algorithms
   - Sequential and Binary Search Algorithms
   - Sorting Algorithms
   - Breadth First and Depth First Search Algorithms
5. Algorithm Design Strategies
   - Divide and Conquer Techniques
   - Dynamic Programming
   - Greedy Algorithms

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

Applications

- Finding the maximum and minimum of a sequence of numbers
- Integer multiplication
- Matrix multiplication

Applications

- Finding the maximum and minimum of a sequence of numbers
- Integer multiplication
- Matrix multiplication

Applications

- 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,

$$\left\lfloor \frac{x+y}{2} \right\rfloor - x + 1 = \frac{x+y-1}{2} - x + 1$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

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,

$$\left\lfloor \frac{x+y}{2} \right\rfloor - x + 1 = \frac{x+y-1}{2} - x + 1$$

$$= \frac{y-x+1}{2}$$

$$= n/2.$$

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,

$$y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 = y - \frac{x+y-1}{2}$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

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,

$$y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 = y - \frac{x+y-1}{2}$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

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,

$$y - \left(\left\lfloor \frac{x+y]}{2} \right\rfloor + 1 \right) + 1 = y - \frac{x+y-1}{2}$$
$$= \frac{y-x+1}{2}$$
$$= n/2.$$

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,

$$T(n) = \begin{cases} 1, & \text{if } n = 2 \\ 2T(n/2) + 2, & \text{otherwise} \end{cases}$$

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:

$$u := (a + b)(c + d)$$
$$v := ac$$
$$w := bd$$
$$x := v2^n + (u - v - w)2^{n/2} + w$$

Thus to multiply n bits numbers we need 3 multiplications of $n/2$ bit numbers and a constant number of additions and shifts.

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.

Strassen's Algorithm for Matrix Multiplication

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.

$$
\begin{aligned}
T(n) &= 7T(n/2) + dn^2 \\
&= 7(7T(n/4) + d(n/2)^2) + dn^2 \\
&= 7^2 T(n/4) + 7dn^2/4 + dn^2 \\
&= 7^3 T(n/8) + 7^2 dn^2/4^2 + 7dn^2/4 + dn^2 \\
&= 7^i T(n/2^i) + dn^2 \sum_{j=0}^{i-1} (7/4)^j \\
&= 7^{\log n} T(1) + dn^2 \sum_{j=0}^{\log n - 1} (7/4)^j \\
&= cn^{\log 7} + dn^2 \frac{(7/4)^{\log n} - 1}{(7/4) - 1} \\
&= cn^{\log 7} + \frac{4}{3} dn^2 \left(\frac{n^{\log 7}}{n^2} - 1\right) \\
&= O(n^{\log 7}) \approx O(n^{2.8}).
\end{aligned}
$$

Improved Algorithms

- 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.

Introduction to 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.

Introduction to 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.

Introduction to 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.

Dynamic Programming Technique

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

Dynamic Programming Technique

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

Dynamic Programming Technique

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

Dynamic Programming Technique

To design a dynamic programming algorithm:

Identification:

- Devise divide and conquer algorithm
- Analyze: running time is exponential
- Same sub problems solved many times

Dynamic Programming Technique

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”

Applications

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Applications

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Applications

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Applications

Dynamic programming: divide and conquer with a table.

Application to:

- Counting combinations
- Knapsack problem

Counting Combinations

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,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Counting Combinations

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,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Counting Combinations

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,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Counting Combinations

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,

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Counting Combinations

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!

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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])
```

A Better Algorithm: 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

Analysis

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 $O(2^n)$.

Analysis

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 $O(2^n)$.

Analysis

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 $O(2^n)$.

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 correctness.

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 correctness.

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 correctness.

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 correctness.

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 correctness.

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 correctness.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Analysis

$O(n \log n)$ for sorting

$O(n)$ for the rest

Analysis

Analysis

$O(n \log n)$ for sorting

$O(n)$ for the rest

Analysis

Analysis

$O(n \log n)$ for sorting

$O(n)$ for the rest

Thank You!