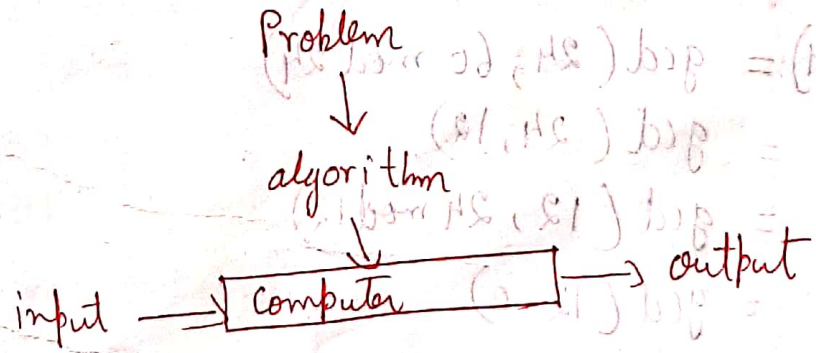
# The Design & Analysis of Algorithms

## UNIT - 1

1.1) Introduction: This subject is all about algorithms, what is an algorithm? Why we need algorithms? How it is going to help in problem solving?

Notion of algorithm; An algorithm is defined as the sequence of unambigous procedure for solving a problem. A procedure is a finite sequence of well defined steps or operations, each of which requires only a finite amount of time / memory to complete.




Notion of algorithm.

While designing an algorithm, few points has to be considered

1) The nonambiguity requirement for each step of an algorithm cannot be compromised.

Scanned by CamScanner

2) The range of inputs for which algorithm works has to be specified very carefully. Eg: gcd → i/p non zero & non -ve integers

3) Same algorithm can be represented in different ways. Eg: Prime no:- using functions, without functions

4) Several algorithms for solving the same problem may exist. Eg: Sorting - bubble sort, selection sort
Searching - linear, binary

5) Different algorithms works with different speed. (i.e for same problem)

Example: As an example, consider 3 methods of finding gcd of two integers.

(1) Euclid's algorithm: It is based on applying repeatedly the equality $gcd(m,n) = gcd(n, m \bmod n)$

$$gcd(60,24) = gcd(24, 60 \bmod 24)$$
$$= gcd(24, 12)$$
$$= gcd(12, 24 \bmod 12)$$
$$= gcd(12, 0)$$
$$gcd(60,24) = 12$$

$$\begin{array}{r} 2 \\ 24\overline{)\,60} \\ \underline{48} \\ 12 \end{array}$$

Euclid's Algorithm for computing gcd(m,n)

step1 : If n = 0, return m as answer & stop.
otherwise goto step2

step2 : Divide m by n & assign value of remainder to r

step3 : Assign m = n, n = r, goto step1

Scanned by CamScanner

Algorithm: Euclid(m,n)

// Computes gcd(m,n)

// Input: Two non-negative, Non zero integers

// output: GCD of 'm' and 'n'

```
while (n ≠ 0) do
    r = m mod n
    m = n
    n = r
return m
```

## ② Consecutive integer checking algorithm

Step$_1$: Assign the value of min{m,n} to t.

Step$_2$: Divide m by t, If the remainder of this division is 0, goto step3 otherwise goto step4

Step$_3$: Divide n by t. If the remainder of this division is 0, return the value of 't' as answer and stop. otherwise proceed step4

Step$_4$: Decrease the value of t by 1, goto step2

m, n

$t = \min\{60, 24\}$

$t = 24$

① $x = m \bmod t \Rightarrow 60 \bmod 24$

$x = 12$

$x \neq 0 \therefore t = t - 1$

$t = 23$

② $x = 60 \bmod 23$

$x = 14$ $t = 23 - 1$

$x \neq 0 \therefore t = 22$

③ $x = 60 \bmod 22$ (22)60(2, 44, 16)

$x = 16$ $t = t - 1 = 22 - 1$

$x \neq 0 \therefore t = 21$

④ $x = 60 \bmod 21$ (21)60(2, 42, 18)

$x \neq 0 \therefore t = t - 1 = 20$

⑤ Now divide n by t Now $60 \% 20 = 0$

$t = 12$ i.e $24 \% 20$ step2

$r \neq 0$ $60 \% 19$

~~$x = 60 \bmod 12$~~ $t - 1 = 19$ $r \neq 0$

~~$x = 0$~~

return $t = 12$

remainder is 0 $60 \% 12 = 0$, $24 \% 12 = 0$

$\therefore n \bmod t = 24 \bmod 12 = 0$ Stop

Scanned by CamScanner

(3) Middle - School procedure for finding gcd (m,n)

Step 1 : Find the prime factors of m.

Step 2 : Find the prime factors of n.

Step 3 : Identify all common factors in two prime expansions found in Step 1 & 2.

Step 4 : compute the product of all common factor & return it as gcd

gcd (60, 24)

| | |
|---|---|
| 2 | 60 |
| 2 | 30 |
| 3 | 15 |
| | 5 |

$60 = 2 \times 2 \times 3 \times 5$

| | |
|---|---|
| 2 | 24 |
| 2 | 12 |
| 2 | 6 |
| | 3 |

$24 = 2 \times 2 \times 2 \times 3$

$\therefore gcd(60, 24) = 2.2.3 = 12$

(4) Sieve of Eratosthenes Method to generate prime nos not exceeding given integer 'n'. if n = 16

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

First consider 2, and remove all multiples of 2

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~

Take 3, remove all multiples of 3

2 3 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~

Take it, List of Primes till 16 Stop

Scanned by CamScanner

## Algorithm

If p is a number whose multiples are eliminated, then first multiple is $p*p$, because all its smaller multiples would be eliminated earlier.

$p*p$ should not be greater than $n$

$\therefore$ p cannot exceed $\sqrt{n}$

## Algorithm

// Implements Sieve of Eratosthenes
// Input : A positive integer $n \geq 2$
// O/P : Array L of all prime nos less than or equal to 'n'.

```
for p ← 2 to n do A[p] ← p
for p ← 2 to √n do
    if A[p] ≠ 0
        j ← p*p
    while j ≤ n do
        A[j] ← 0
        j ← j+p
// copy the remaining elements of Array A to L
i ← 0
for p ← 2 to n do
    if A[p] ≠ 0
        L[i] ← A[p]
        i ← i+1
return L
```

Scanned by CamScanner

| A = | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| | | | 2 | 3 | 0 | 5 | 0 |

(p=2, p*p = 4 i.e 4 < 6, p=3 => p*p = 9 > 6 No need because p*p can't exceed n)

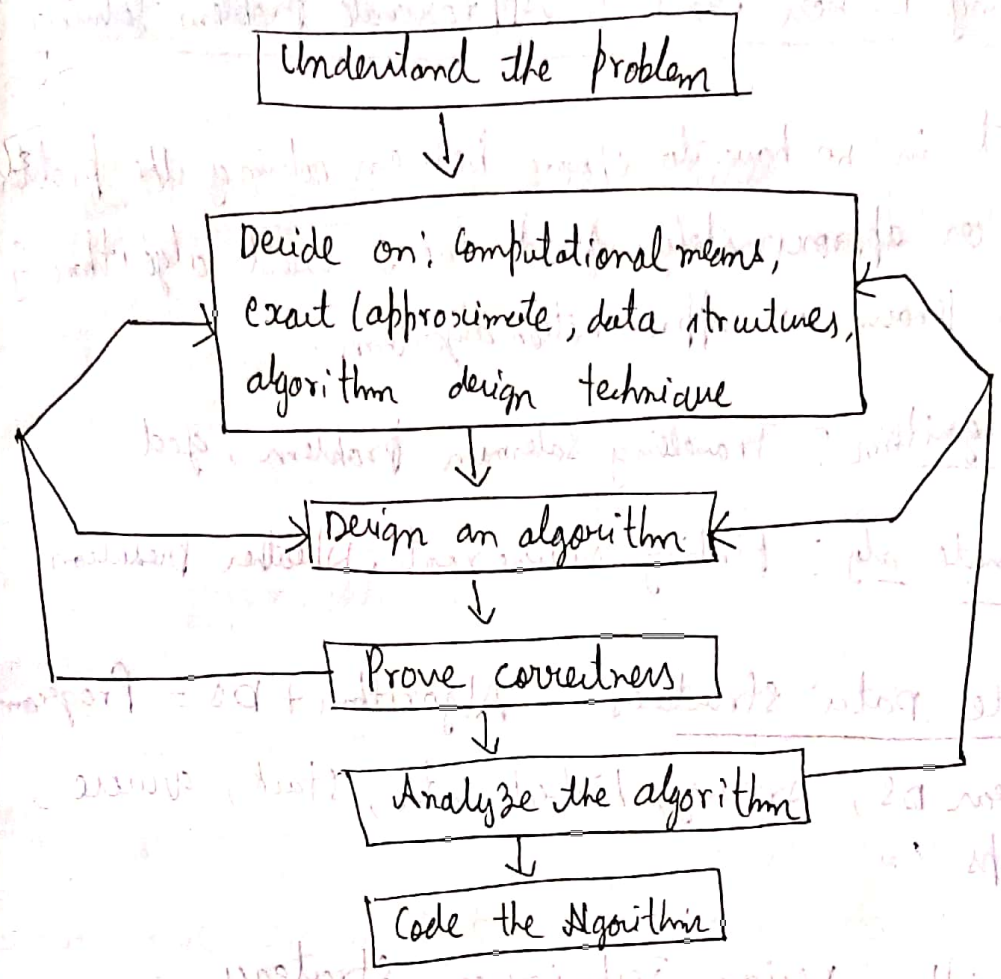
n = 6

```
for (p=2; p<=16; p++) // This is for generating array 2,3,4,5,6....16
{                                                    index (2) (3) (4)
  A[p] = p;                                          √16 = 4
}
for (p=2; p <= √16; p++)
{
   if (A[p] != 0)
      j = p * p;
   while j <= n do
   {
      A[j] = 0;  // mark an element as eliminated
      j = j + p;
   }
}
// copy remaining elements of A array to L
i = 0;
for (p=2; p <= n; p++)
{
  if (A[p] != 0)
     L[i] = A[p];
     i = i + 1;
}
return L
```

## 1.2) Fundamentals of Algorithmic Problem solving

The sequence of steps one typically goes through in designing and analyzing an algorithm is as shown in figure.

Scanned by CamScanner




) Understanding the Problem: First thing you need to do before designing an algorithm is to understand completely the problem given. Read the problem's description carefully. Input to the algorithm is very important, it specifies an instance of the problem the algorithm solves. Its important to specify exactly the range of instances algorithm need to handle.

) Ascertaining the capabilities of computational device:

Once we understand the problem, we need to ascertain the capabilities of computational device, i.e configuration memory.

Now technology is improved, so we need not worry about memory.

Scanned by CamScanner

3) Choosing between exact & Approximate Problem Solving.

Next is we have to choose between solving the problem exactly or approximately. first one is exact algorithm & later is known as approximation algorithm.

Exact algorithm: Travelling Salesman problem, gcd

Approximate alg: finding square root, Whether Prediction

4) Decide Data structures: Algorithm + DS = Program

Linear DS, Array, Linked lists, Stack, queue, graphs

5) Algorithm Design Techniques: Strategy

6) Methods of Specifying an algorithm: 1) Using Natural lang, i.e steps

2) Pseudocode

3) Flowchart

7) Proving Correctness: P.T algorithm yields a required result for every input in a finite amount of time.

8) Analyzing the Algorithm: Analyze for its efficiency

1) Time Efficiency
2) Space Efficiency
3) Simplicity
4) generality

Scanned by CamScanner

Problem Solving.

solving the problem
exact algorithm &

blem, gcd
Weather Prediction

m + DS = Program
stack, queue,

rategy

; 1) Using Natural
lang, i.e step
Pseudocode
lowchart
ithm yields a
t in a finite amount

e for its efficiency
ty

9) Coding

## 1.3) Fundamentals Data Structures

1) What is Data Structure?

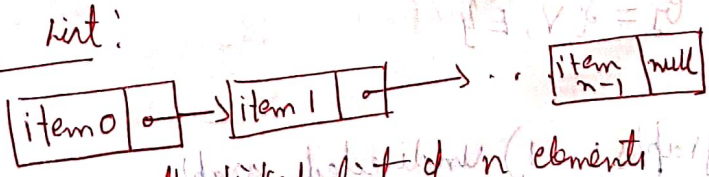
A data structure can be defined as a particular scheme of organizing related data items. Data items can be of type integers or characters.

### Linear Data Structures

Array

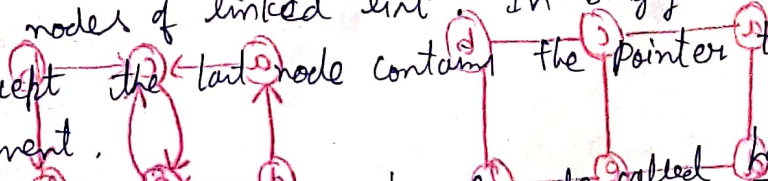
The two important elementary data structure are the arrays and linked list. A 1D array is a sequence of n items of same data type that are stored in continous memory location, accesible by specifying value of array's index.

Linked List:




Singlly linked list of n elements

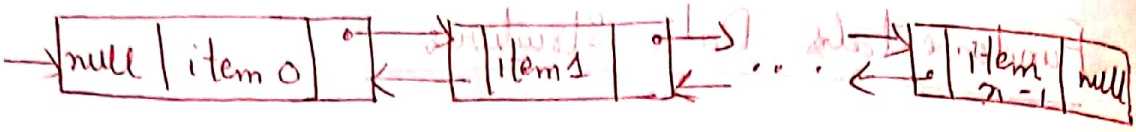
A linked list is a sequence of zero or more elements called nodes, each node containing data and pointer to other nodes of linked list. In Singly LL each node except the last node contains the pointer to next element.



Start a linked list with a special node called header, This contains information about linked list, such as its length, pointer to first element, last element

Scanned by CamScanner

# Doubly linked list




Extension to Singly LL is called Doubly LL, in which every node, except the first and last, contains pointer to both its successor and predecessor.

Stack -> LIFO

queue -> FIFO

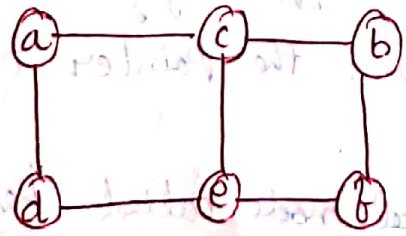
Priority queue

## Graphs

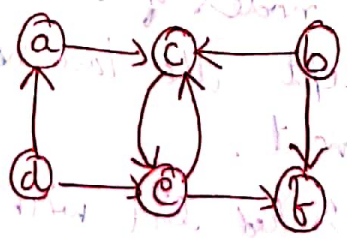
Graph is defined as a collection of points in a plane called "vertices" or "nodes", some of them connected by a line segment called as 'edges' or arcs.

graph => $G = \{V, E\}$

Types of graphs: 1) Undirected graph
2) Directed graph (Digraph)
3) Weighted graph




a) undirected graph




b) Digraph

Scanned by CamScanner

If the pair of vertices of graph G are unordered, it is undirected graph. i.e pair of vertices (u,v) is same as (v,u).

If the edge (u,v) is directed from vertex u to vertex v, then it is directed graph.

For undirected graph

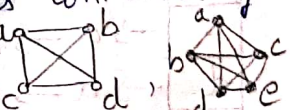
$V = \{a, b, c, d, e, f\}$

$E = \{(a,c), (a,d), (b,c), (b,f), (c,e), (d,e), (b,e)\}$

For Digraph

$V = \{a, b, c, d, e, f\}$

$E = \{(a,c), (b,c), (b,f), (c,e), (d,a), (d,e), (e,c), (e,f)\}$

A graph with all its pair of vertices connected by a edge is called complete graph. Eg: 


A graph with few possible edges missing is called dense.

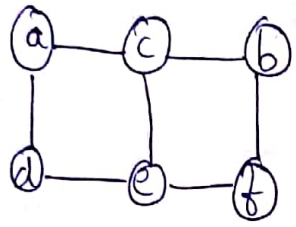
A graph with few edges relative to the no of vertices is called sparse.

Scanned by CamScanner

# Graph Representations :

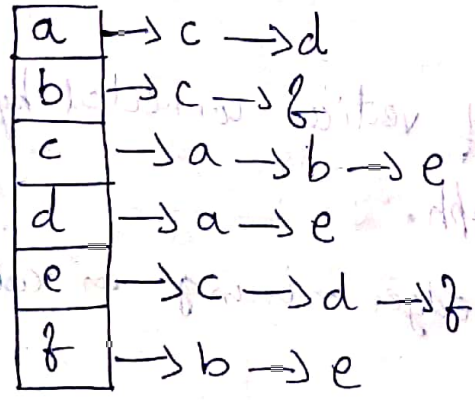
Graph can be represented in two ways

1) Adjacency matrix
2) Adjacency linked list




| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a | 0 | 0 | 1 | 1 | 0 | 0 |
| b | 0 | 0 | 1 | 0 | 0 | 1 |
| c | 1 | 1 | 0 | 0 | 1 | 0 |
| d | 1 | 0 | 0 | 0 | 1 | 0 |
| e | 0 | 0 | 1 | 1 | 0 | 1 |
| f | 0 | 1 | 0 | 0 | 1 | 0 |

Adjacency matrix



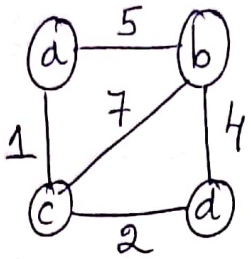

Adjacency Link list : In this for each vertex 'a', we write down all other vertex that are connected to a.

Scanned by CamScanner

# Weighted graph

A weighted graph is a graph with numbers assigned to its edges. These numbers are called weights or cost.

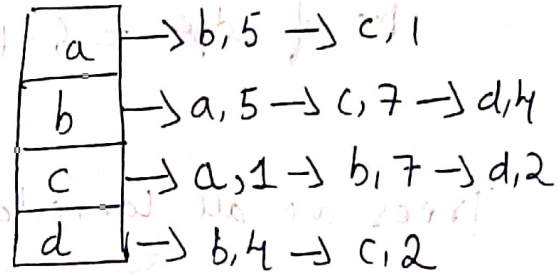
Example:




Representation

| | a | b | c | d |
|---|---|---|---|---|
| a | ∞ | 5 | 1 | ∞ |
| b | 5 | ∞ | 7 | 4 |
| c | 1 | 7 | ∞ | 2 |
| d | ∞ | 4 | 2 | ∞ |

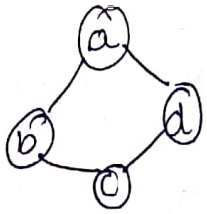
Adjacency matrix



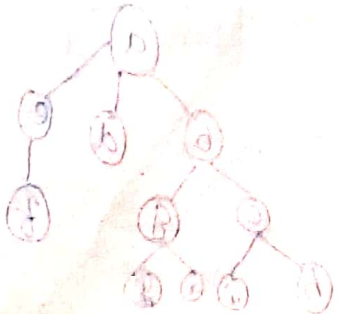

Path: A path from vertex u to v of graph G is defined as sequence of adjacent vertices that starts from u & ends at v.

Length of path is no of vertices in path − 1

Cycle is simple path of +ve length that starts and ends at same vertex.




Cycle

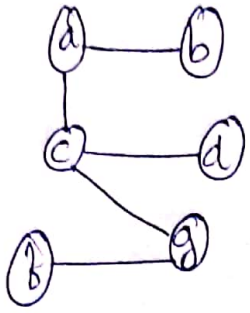


Scanned by CamScanner

# Trees

A tree in a connected acyclic graph. The no of edges in tree is one less than no of vertices

Eg:



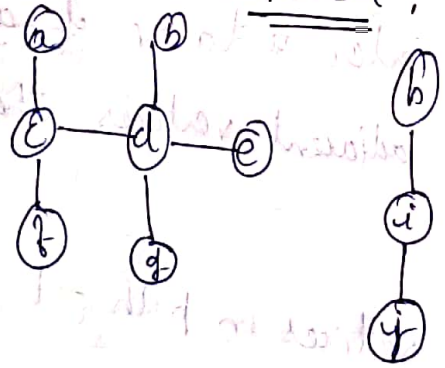

No of vertices = 6

No of edges = 6 - 1 = 5

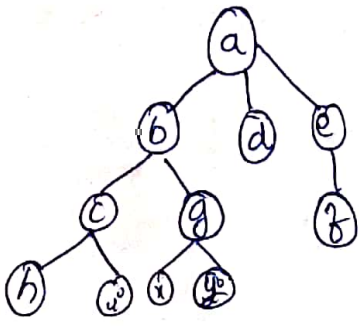
Trees are all connected graph, No cycle

A graph that has no cycles but not necessarily connected is called Forest.

Eg:

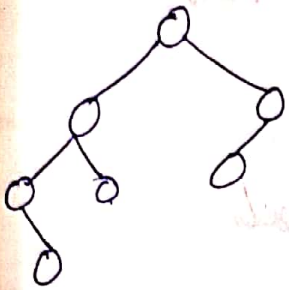



Rooted tree: Rooted tree is a tree where we select an arbitrary vertex as root, Root is placed on top.



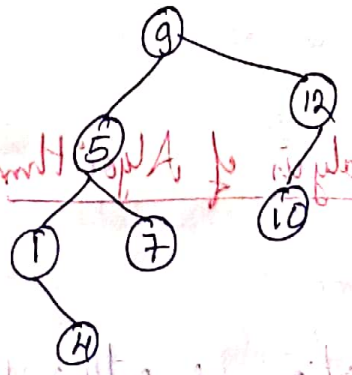

Scanned by CamScanner

**Binary Tree** : Binary Tree is a tree where each parent can have atmost 2 child, not more than 2.

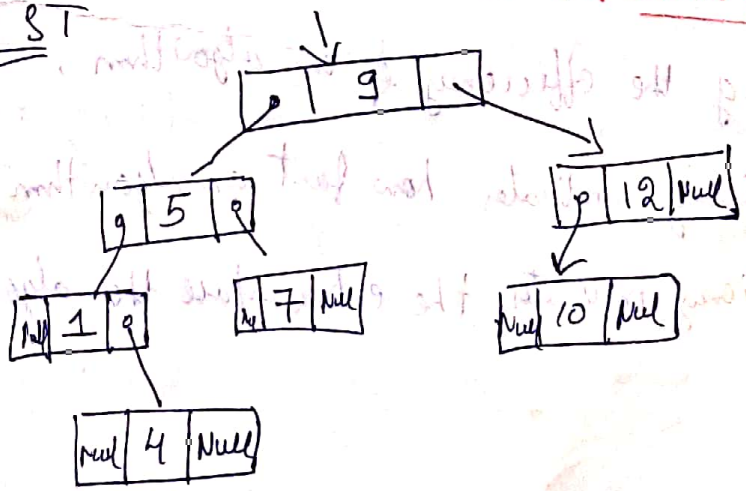


Binary Tree

**Binary Search Tree** : Its a tree where the left child has value less than root & right child has value greater than root value. Its an ordered tree




BST




Scanned by CamScanner

## Sets and Dictionaries

Sets can be described as an unordered collection of distinct items called elements of sets.

$S = \{2, 3, 5, 7\}$

Union & Intersection operations on sets

Union means combining the elements of two sets.
Intersection means taking only the common elements from two sets.

operations we can perform on sets is like searching for an element, adding an element, deleting an element. A data structure which implements these operations is dictionary.

## 1.4) Fundamentals of the Analysis of Algorithm Efficiency

Analysis framework: In this section, we outline the framework for analyzing the efficiency of an algorithm.

Time Efficiency indicates how fast an algorithm runs.

Space Efficiency indicates the extra space the algorithm requires.

1) Measuring Inputs si

All Algorithms

For Eg: it to

multiples larger

∴ Its logical to

based on some pa

'n' indicating th

2) Units for Mea

As an unit for

we can measure

it depends on

compilers. It

So, the possib

each algorithm's

to identify th

called as bas

Eg: 1) Sorting A

2) Matrix m

Cop = Time

cn = no of ti

T(n) = Esti

T(n) ≈

Scanned by CamScanner

## Measuring Input's size

All Algorithms runs longer on larger inputs.

For Eg: it takes longer to sort larger arrays, multiply larger matrices so on.

∴ Its logical to investigate the algorithm's efficiency based on some parameter. A function of some parameter 'n' indicating the algorithm's input size.

## Units for Measuring Running Time

As an unit for measuring the algorithm's Running time we can measure in terms of milliseconds. but it depends on Computer speed, quality of program, Compilers! It depends on these extraneous factors.

So, the possible approach is to count the no of times each algorithm's operation is executed. The thing to do is to identify the most important operation of algorithm, called as basic operation

Eg: 1) Sorting Algorithm works by comparing elements.
2) Matrix multiplication :— Multiplication, Addition.

$C_{op}$ = Time of Execution of algorithm's basic operation.
$C_n$ = no of times basic operation is executed
$T(n)$ = Estimated Running time of Program

$$T(n) \approx C_{op} * C(n)$$

Scanned by CamScanner

$C(n) = \frac{1}{2}n(n-1)$

$= \frac{1}{2}n^2 - \frac{1}{2}n$

$C(n) \approx \frac{1}{2}n^2$

If we run algorithm on machine which is double its i/p size

Assume $C_n = \frac{1}{2}n(n-1)$

$\therefore \frac{T(2n)}{T(n)} \approx \frac{C_{op}C(2n)}{C_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4$

So, if we double the i/p size, the algorithm runs 4 times longer

## 3) Orders of Growth

We have to analyze the order of growth for analysis of algorithms. Table shows values of some function for analysis of growth

| $n$ | $\log_2 n$ | $n$ | $n \log_2 n$ | $n^2$ | $n^3$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|---|
| 10 | 3.3 | $10^1$ | $3.3 \cdot 10^1$ | $10^2$ | $10^3$ | $10^3$ | $3.6 \cdot 10^6$ |
| $10^2$ | 6.6 | $10^2$ | $6.6 \cdot 10^2$ | $10^4$ | $10^6$ | | |

## 4) Worst-case, Best-case, Average case efficiencies

Example: // Sequential Search

// Searches for given value in a given array

// Input: A[0..n-1] & key

// O/p: Returns index of element A that matches key or -1

n=4 key:20

10 0

20 1

30 2

40 3

```
i ← 0
while (i < n) and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return -1
```

Worst-case efficiency:

Worst case efficiency

Worst case input

the longest.

$C_{wo}$

Best-case efficiency:

Best-case input;

$C_{best}(n) = 1$

Avg case efficiency

in neighter wor

$C_{avg}(n) = [$

$= \frac{p}{n}$

$= \frac{p}{n}$

$= \frac{p}{n}$

$C_{avg}(n) = p$

we do comp

the ini

Scanned by CamScanner

## Worst-case efficiency:

Worst case efficiency of algorithm is its efficiency for worst case input of size n, for which algorithm runs the longest. $C_{worst}(n) = n \quad O(n)$

Best-case efficiency: Is the efficiency of algorithm for Best-case input, for which algorithm runs fastest.

$C_{best}(n) = 1 \quad O(1)$

Avg case efficiency: Is the efficiency of alg which is neither worst nor best

$$C_{avg}(n) = \left[1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \ldots + i \cdot \frac{p}{n} \ldots n \cdot \frac{p}{n}\right] + n \cdot (1-p)$$

$$= \frac{p}{n}\left[1 + 2 + \ldots + i + \ldots + n\right] + n(1-p)$$

$\rightarrow$ std formula for Arithmetic Progression

$$= \frac{p}{n} \frac{n(n+1)}{2} + n(1-p)$$

$$= \frac{p}{n} \frac{n(n+1)}{2} + n(1-p)$$

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p)$$

$O(n)$ because the no of comparisons we do cannot exceed n, it can be less than or equal to n. So we use big oh

Scanned by CamScanner

In average case efficiency for sequential search,

$p$ = The probability of a successful search $(0 \le p \le 1)$

$C_{avg}(n)$ = The average number of key comparisons.

In case of successful search, probability of first match occuring at $i^{th}$ position is $\frac{p}{n}$ for every $i$.

In case of unsuccessful search, no of comparisons is $n$, Probability of such search is $(1-p)$.

$$\therefore C_{avg}(n) = \left[1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \dots i \frac{p}{n} + \dots n \cdot \frac{p}{n}\right] + n \cdot (1-p)$$

$$= \frac{p}{n}\left[1 + 2 + 3 + \dots + i + \dots + n\right] + n \cdot (1-p)$$

↓

Arithmatic Progenium formula $= \frac{n(n+1)}{2}$

$$= \frac{p}{n} \frac{n(n+1)}{2} + n(1-p)$$

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p)$$

## 3) Order of Growth of 'n'

We all expect algorithms must execute faster, but what will be the value of 'n'? If algorithms works faster for smaller values of 'n' and slower for large value of 'n', it's not good algorithm.

Scanned by CamScanner

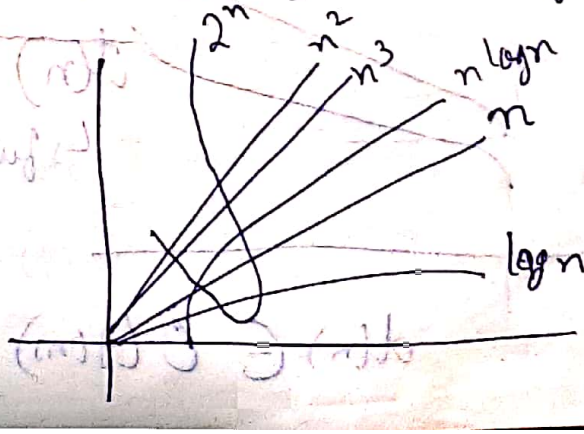
| Example: | Function | Name |
|---|---|---|
| 1. | 1 | Constant |
| 2. | $\log n$ | Logarithmic |
| 3. | $n$ | Linear |
| 4. | $n \log n$ | $n \log n$ |
| 5. | $n^2$ | quadratic |
| 6. | $n^3$ | cubic |
| 7. | $2^n$ | Exponential |
| 8. | $n!$ | Factorial |

Table 3.1

binary log: $\log_2 n$ by what value 2 should be raised to get n:

| | $n$ | $\log n$ | $n \log n$ | $n^2$ | $n^3$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|---|
| 1) | 1 | 0 $(2^0)$ | 0 | 1 | 1 | 2 | 1 |
| 2) | 2 | 1 $(2^1)$ | 2 | 4 | 8 | 4 | 2 |
| 3) | 4 | 2 $(2^2)$ | 8 | 16 | 64 | 16 | 24 |
| 4) | 8 | 3 $(2^3)$ | 24 | 64 | 512 | 256 | 40320 |
| 5) | 16 | 4 $(2^4)$ | 64 | 256 | 4096 | 65536 | 20922789888000 |

We can compare the results for different values of n.
The function $2^n$ & $n!$ has the faster growth.




Scanned by CamScanner

## 1.5) Asymptotic Notations

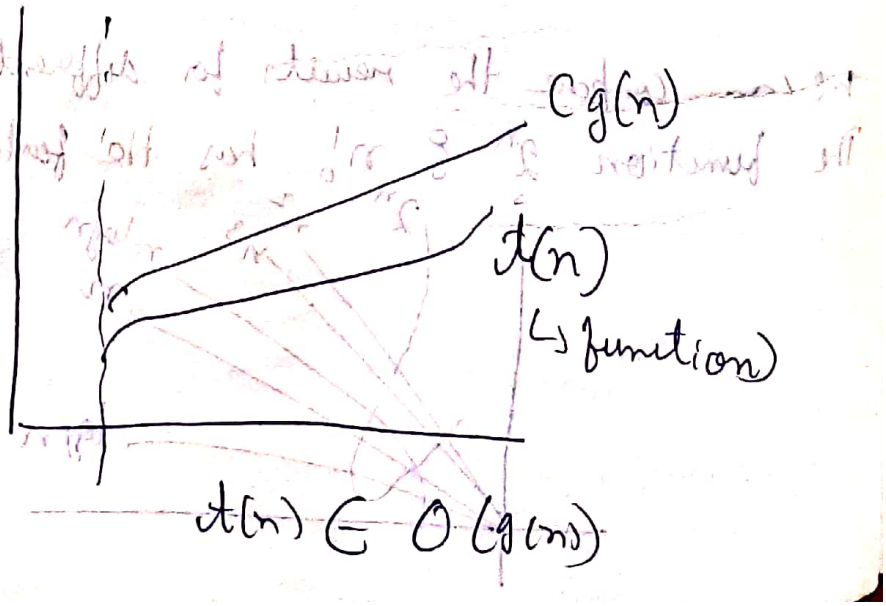
The word asymptotic means the study of functions of a parameter 'n', as 'n' grows larger and larger without bounds. In other words, we are concerned with how the running time of an algorithm increases with the size of the input. An algorithm that is asymptotically more efficient will be the best choice for all inputs.

### Big oh Notation (O)

We use O-notation to give an upper bound on a function $f(n)$ within a constant factor. The upper bound on $f(n)$ indicates that the function $f(n)$ will be the worst-case that it doesn't consume more than this computing time.

i.e $f(n)$ is of order $g(n)$ such that $f(n)$ grows no more than $g(n)$

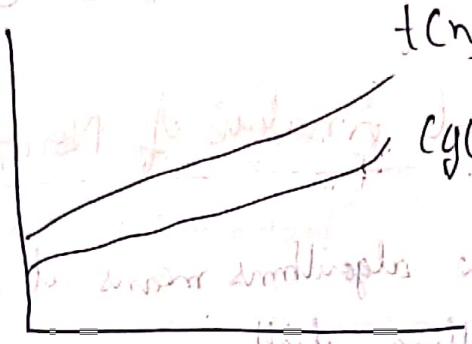
Eg: Linear Search
No of comparisons (cmd) be more than 'n'
it can be $\leq n$
$O(n)$




Scanned by CamScanner

## Big Omega Notation ($\Omega$)

We use $\Omega$ notation to give a lower bound on function $t(n)$ within a constant factor. The lower bound on function $t(n)$ indicates that the function $t(n)$ has to take atleast this amount of execution time. It cannot be less than the lower bound.




Efficient Alg

$n = 3$

Subset $= 2^3$

i.e $2^n$

$t(n) \in \Omega(g(n))$

## Big Theta Notation ($\Theta$)

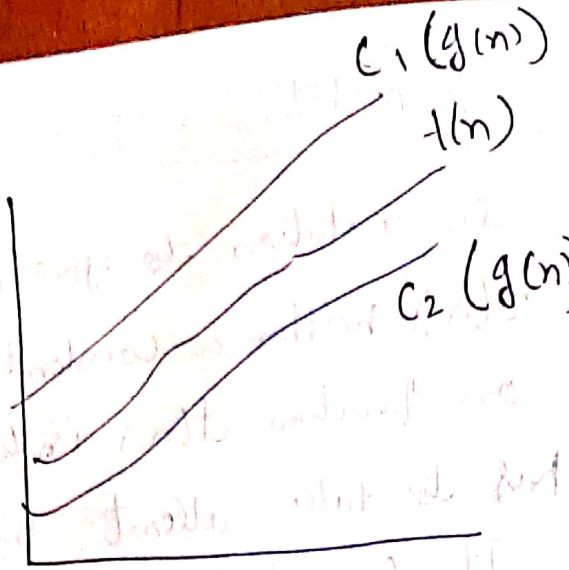
For some problem the lower bound & upper bound may yield the same order of growth. i.e $O, \Omega$ will have same growth $g(n)$.

Eg: finding minimum element & maximum element in array of size takes same computing time

In such case we use theta

The function $t(n)$ is said to be in $\Theta(g(n))$, $t(n) \in \Theta(g(n))$, $t(n)$ is bounded both above & below i.e lower bound & upper bound

Scanned by CamScanner




$t(n) \in \Theta(g(n))$

## 1.6) Mathematical Analysis of Non-recursive Algorithm

Non Recursive algorithms means it do not include any function calling itself.

Example 1: Algorithm to find the largest element in a array.

Algorithm

// Determines the value of the largest element in array
// Input: An array A[0...n-1] of real nos
// output: The value of the largest element in A.

maxval ← A[0]
for i ← 1 to ≤ n-1 do
if A[i] > maxval
maxval ← A[i]
return maxval

Pseudocode:

```
maxval = a[0];
for (i=1; i<=(n-1); i++)
{
    if (a[i] > maxval)
    {
        maxval = a[i];
    }
}
return maxval
```

| | |
|---|---|
| 10 | 0 |
| 12 | 1 |
| 35 | 2 |
| 75 | 3 |
| 2 | 4 |

Scanned by CamScanner

In this algorithm, the basic operation is comparison optn bcoz comparison optn is repeated for each iteration

C(n) = no of times comparison is executed

$$C(n) = \sum_{u=1}^{n-1} 1$$

$\sum_{i=l}^{u} 1 = u - l + 1$

$n - 1 - 1 + 1$

$\sum_{i=1}^{n-1} 1$ can be expanded as below

$$\sum_{i=1}^{n-1} 1 = 1 + 1 + 1 + \ldots\ldots + (n-1)$$

(n-1) times 1 is added

$$= 1(n-1)$$

$$\sum_{u=1}^{n-1} 1 = n - 1 \in \Theta(n)$$

Example 2: Algorithm to check whether all elements of array are distinct.

Algorithm: // checks whether all the elements are distinct
// Input: Array a[0..n-1]
// O/P: Returns "true" if distinct else false

for i ← 0 to n-2 do
  for j ← i+1 to n-1 do
    if a[i] == a[j] return false
return true

Scanned by CamScanner

Pseudocode:

| | |
|---|---|
| 10 | 0 |
| 20 | 1 |
| 20 | 2 |
| 30 | 3 |

$n = 4$,

```
for (i=0; i ≤ n-2; i++)
    for (j=i+1; j ≤ n-1; j++)
        if (a[i]==a[j])
            return false
return true
```

| | |
|---|---|
| 20 | 3 |
| 30 | 2 |
| 20 | 1 |
| 10 | 0 |

Note: How this alg works means, a[0] is compared with a[1], a[2], a[3]. Then a[1] is compared a[2], a[3] and then a[2] is compared with a[3]. Stop.

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

Note: 2 important formulas required

1) $F_1$: $\sum_{i=l}^{u} 1 = u - l + 1$, here 1 is constant function $f(i) = 1$, $f(0) = 1$, $f(1) = 1$, $f(l) = 1$

2) $F_2$: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, here $f(i) = i$, $f(0) = 0$, $f(1) = 1$, $f(n) = n$

$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$

$$C(n) = \sum_{i=0}^{n-2} \left( \sum_{j=i+1}^{n-1} 1 \right)$$ → Apply $F_1$

$$C(n) = \sum_{i=0}^{n-2} (n-1) - (i+1) + 1$$

$$= \sum_{i=0}^{n-2} n - 1 + i^0 - 1 + 1$$

$$= \sum_{i=0}^{n-2} n - 1 - \sum_{i=0}^{n-2} i^0$$

Apply $F_1$ Apply $F_2$

$$= n-1 \left( \sum_{i=0}^{n-2} 1 \right) + \left( \sum_{i=0}^{n-2} i^0 \right)$$

$$= n-1 \left[ n-2-0+1 \right] - \left[ \frac{(n-2)(n-2+1)}{2} \right]$$

$$= (n-1)(n-1) - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1) \left[ (n-1) - \frac{(n-2)}{2} \right]$$

$$= (n-1) \left[ \frac{2n-2-n+2}{2} \right]$$

$$= (n-1) \left( \frac{n}{2} \right) \simeq \frac{n^2}{2} \in \Theta(n^2)$$

Example 3: Algorithm to multiply two matrices A & B.

Scanned by CamScanner

**Algorithm**: // Multiplies two square matrices of order n
// Input: Two matrices A & B
// O/P: Matrix C = A×B

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        C[i,j] ← 0
    for k ← 0 to n-1 do
        C[i,j] ← C[i,j] + A[i,k] * B[k,j]
```

Total number of multiplications M(n) is expressed by following triple sum.

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left( \sum_{k=0}^{n-1} 1 \right) \rightarrow \text{Apply } F_1$$

$$F_1: \sum_{i=l}^{u} 1 = u - l + 1$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n - k - 0 + 1$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} n \left( \sum_{j=0}^{n-1} 1 \right) \rightarrow \text{Apply } F_1$$

$$= \sum_{i=0}^{n-1} n \cdot n$$

$$= n^2 \left( \sum_{i=0}^{n-1} 1 \right) \text{ Apply } F_1 = n^3$$

$$\sum_{k=0}^{n-1} 1 = (n-1) - 0 + 1$$

$$= (n-1) + 1$$

$$= n - 1 + 1$$

$$= n$$

$$\sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} n \sum_{j=0}^{u} \rightarrow n$$

$$= \sum_{i=0}^{n-1} n \sum_{j=0}^{u-l}$$

$$n^2$$

$$n^3$$

Scanned by CamScanner

Running time of algorithm on a particular machine,

$$T(n) \simeq C_m M(n) = C_m n^3$$

$T_n \rightarrow$ Total Run time of alg

$M(n) \rightarrow$ No of multiplication

$C_m \rightarrow$ Time of one Multiplication

$$T(n) \approx C_m M(n) + C_a A(n)$$

$$= C_m n^3 + C_a n^3$$

$$= (C_m + C_a) n^3$$

$A(n) =$ time on addition operation

Example 4: Algorithm to find the number of binary digits in the binary representation of a positive decimal integer.

Algorithm: // Input: a positive decimal integer n
// output: The number of binary digits

count ← 1
while n > 1 do
count ← count + 1
n ← ⌊n/2⌋
return count

Pseudocode:
count = 1
while (n > 1) do {
count = count + 1;
n = n/2
}
return count

Eg: n = 4, Binary Rep 1 2 4
0 0 1

① cnt = 1
while (4 > 1) do
cnt = 2;
n = 4/2 = 2

② while (2 > 1) do
cnt = 3;
n = 2/2 = 1

③ while (1 ≯ 1)
terminate
return cnt = 3

Scanned by CamScanner

here, rather than comparison, the number of times the loop executed in considered as basic operation.

Since, in each iteration the value of n is halved, the answer should be $\log_2 n$. The no of ~~times~~ bits in binary representation is $\log_2 n + 1$.

$\log_a^x = \log x / \log a$ ; ie; $\log_3^{17} = (\log 7)/(\log 3)$

## 1.7) Mathematical Analysis of Recursive Algorithms

Example 1: Computing the factorial function $F(n) = n!$ for an arbitrary non-integer n.

Since $n! = 1 \cdot \ldots \cdot (n-1) \cdot n$

$n! = (n-1)! \cdot n$ for $n > 1$

$0! = 1$, $F(n) = F(n-1) \cdot n$ with ~~full~~ recursive algorithm.

Algorithm: // Computes $n!$ recursively
// Input: A nonnegative integer n
// O/P: The value of $n!$
if $n = 0$ return 1
else return $F(n-1) * n$

The basic operation of this algorithm is multiplication.

$M(n)$ = number of multiplications

It is computed from the formula $F(n)$

$F(n) = F(n-1) \cdot n$ for $n > 0$

The number of multiplications $M(n)$ needed to compute factorial must satisfy the equality

Scanned by CamScanner

$M(n) = M(n-1) + 1$ for $n > 0$

$M(n-1)$: $(n-1)$ multiplication; $1$: to multiply $n$

Note: $M(n)$ = number of multiplications

$M(n) = M(n-1) + 1$ for $n > 0$

$M(0) = 0$ call Mofs when $n=0$, no multiplication when $n=0$

$F(n) = F(n-1) \cdot n$ for $n > 0$

$F(0) = 1$ as $0! = 1$

Note: $F(n)$ = function to compute factorial

## Method of backward Substitution for solving Recurrence relations

$M(n) = M(n-1) + 1 \rightarrow F_1$

Substitute $F_1$ to calculate $M(n-1)$

$M(n-1) = M(n-1-1) + 1 = \boxed{M(n-2) + 1}$ → Substitute in $F_1$

$M(n) = [M(n-2) + 1] + 1 = M(n-2) + 2$

↳ calculate $M(n-2)$ using $F_1$

$M(n-2) = M(n-2-1) + 1 = \boxed{M(n-3) + 1}$

Substitute $M(n-2)$

$M(n) = [(M(n-3) + 1) + 1] + 1$

$M(n) = M(n-3) + 3$ if we generalize $M(n-i) + i$

$\vdots$

$M(n) = M(n-1) + 1$

$\vdots$

$M(n) = M(n-i) + i$

$M(n) = M(n-n) + n = n$

Scanned by CamScanner

## General Plan for Analyzing Efficiency of Recursive Algorithms

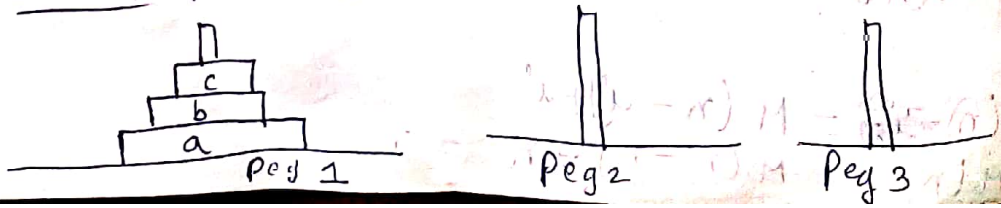
1) Decide a parameter indicating an input's size
2) Identify the algorithm's basic operation
3) check whether the no of time basic operation executed can vary for input size. if so investigate best case, worst case, avg case efficiency.
4) Set up a recurrence relation, with appropriate initial condition, for no of times basic operation is executed.
5) Solve the recurrence or atleast ascertain the order of growth of its solution.

### Example 2: Tower of Hanoi Puzzle

In this puzzle, we have n disks of different sizes, and 3 Pegs. initially, all the disks are on first peg in order of size, largest on bottom and smallest on top. The goal is to move all disks to the third peg, using the second as auxillary.

Conditions: 1) we can move only one disk at a time.
2) we can not place larger disk on smaller disk

for Example: $n = 3$




Scanned by CamScanner

1) move disk 'c' from $Peg_1$ to $Peg_3$
2) Move disk 'b' from $Peg_1$ to $Peg_2$
3) move disk 'c' from $Peg_3$ to $Peg_2$
4) move disk 'a' from $Peg_1$ to $Peg_3$ directly
5) move disk 'c' from $Peg_2$ to $Peg_1$
6) move disk 'b' from $Peg_2$ to $Peg_3$
7) move disk 'c' directly from $Peg_1$ to $Peg_3$

Move n-1 disks from $Peg_1$ to $P_3$ using $P_2$ as aux
Move largest disk directly from $Peg_1$ to $Peg_3$
Move n-1 disks from $Peg_2$ to $Peg_3$ using $P_1$ as aux

$$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1$$

$M(n)$ = number of moves

$$M(n) = 2M(n-1) + 1$$

if only one disk $M(1) = 1$

Backward Substitution method

$$M(n) = 2M(n-1) + 1 \rightarrow (F_1)$$

$$2[2M(n-2)+1]+1$$

$$= 2^2 M(n-2) + 2 + 1 \rightarrow (F_2)$$

$$2^2[2M(n-3)+1]+2+1$$

$$= 2^3 M(n-3) + 2^2 + 2 + 1$$

$$\vdots$$

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \ldots 2 + 1$$

$$= 2^i M(n-i) + 2^i - 1$$

$M(n) = 2M(n-1) + 1$

$M(n-1) = 2M(n-1-1) + 1$

$M(n-1) = 2M(n-2) + 1$

Substitute in $F_1$

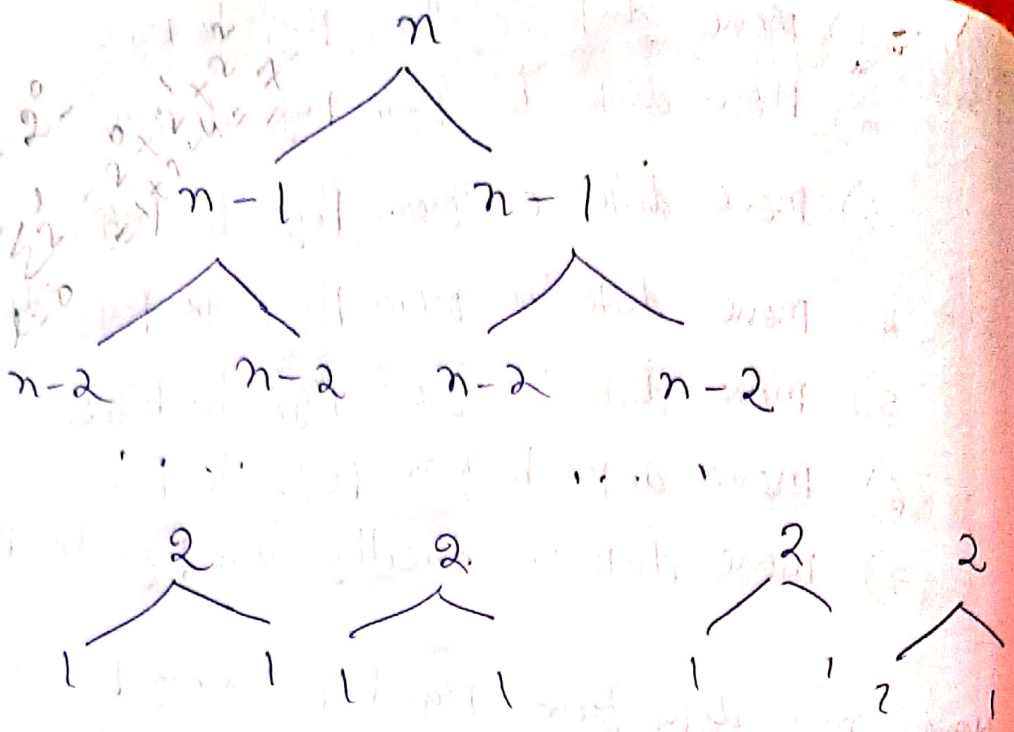
calculate $M(n-2)$

$M(n-2) = 2M(n-2-1) + 1$

$= 2M(n-3) + 1$

Substitute in $F_2$

Scanned by CamScanner




Since initial condition is specified for $n = 1$, which is achieved for $n = n - 1$, we get formula for solution of recurrence

$$M(n) = 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1$$

$$C(n) = \sum_{l=0}^{n-1} 2^l \quad (l = \text{level of the tree})$$

Note: $C(n)$ = no. of calls made by algo

$$C(n) = 2^n - 1$$

$$= 2^{n-1} M(n-n+1) + 2^{n-1} - 1$$

$$= 2^{n-1} M(1) + 2^{n-1} - 1$$

$$= 2^{n-1} + 2^{n-1} - 1$$

$$= 2 \times 2^{n-1} - 1$$

$$= 2^{n-1+1} - 1 = 2^n - 1$$

Example 3: Binary Reunion

// Input: $n$

// O/P: no of digits in $n$'s binary representation

if $n = 1$ return 1

else return BinRec($\lfloor n/2 \rfloor$) + 1

Scanned by CamScanner

Eg: $n=8$, Binary → 1 2 4 8 : 0 0 0 1 ; 8 4 2 1 : 1 0 0 0

(1) $n \neq 1$

BinRec(8/2) + 1 = Bin(4) + 1

↳ Bin(4/2) + 1 + 1

↳ [Bin(2) + 1] + 1

↳ [Bin(2/2) + 1 + 1] + 1

↳ [Bin(1) + 1 + 1] + 1

↳ 1 + 1 + 1 + 1 = 4

The number of additions made in computing BinRec($\lfloor n/2 \rfloor$) is $A(\lfloor n/2 \rfloor)$, plus one more addition to increase the returned value by 1.

$$A(n) = A(n/2) + 1 \quad \text{for } n > 1$$

When $n = 1$, Recursive calls end, no additions are made

$$A(1) = 0$$

Because of $\lfloor n/2 \rfloor$ in function BinRec, the backward substitution method may fail if value of n is not power of 2. ∴ std approach to solve is for $n = 2^k$.

Scanned by CamScanner

$$A(2^k) = A(2^{k-1}) + 1 \text{ for } k > 0$$

$$A(2^0) = 0 \Rightarrow A(1) = 0$$

Substitution

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

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

$$A(2^k) = [A(2^{k-3}) + 1] + 2$$

$$A(2^k) = A(2^{k-3}) + 3$$

$$\vdots = [A(2^{k+1}) + 1] + 3$$

$$\vdots$$

$$A(2^k) = A(2^{k-i}) + i$$

$$\vdots$$

$$A(2^k) = A(2^{k-k}) + k = A(2^0) + k$$

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

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

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

$$= A(2^{k-3}) + 1$$

$$A(2^{k-3}) = A(2^{k-3-1}) + 1$$

we end up $A(2^k) = A(1) + k$

$$A(2^k) = k$$

$$n = 2^k, \quad k = \log_2 n$$

$$A(n) = \log_2 n \in \Theta(\log n)$$

Scanned by CamScanner

Example 4: Fibonacci numbers

Consider 0, 1, 1, 2, 3, 5, 8 . . . .

can be defined by simple recurrence

$f(n) = f(n-1) + f(n-2)$ for $n \geq 1$ with 2 initial conditions, $f(0) = 0$, $f(1) = 1$

We can make use of homogenous 2nd order linear eqtn

$$ax(n) + bx(n-1) + cx(n-2) = 0$$

a, b, c are some fixed real numbers $a \neq 0$,

$x(n)$ = sequence to be found.

$$f(n) - f(n-1) - f(n-2) = 0$$

Characteristic equation for this is

$$r^n - r^{n-1} - r^{n-2} = 0$$

$$r - 1 - r^{-1} = 0$$

$$r - 1 - \frac{1}{r} = 0$$

$$\frac{r^2 - r - 1}{r} = 0$$

$\therefore r^2 - r - 1 = 0$ $\left( ax^2 + bx + c = 0, x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \right.$

$a = 1, b = -1, c = -1$

roots, $r_1 = \frac{1 + \sqrt{5}}{2}$

$r_2 = \frac{1 - \sqrt{5}}{2}$

$r = \frac{1 \pm \sqrt{1 - 4 \times 1 \times -1}}{2}$

$r = \frac{1 \pm \sqrt{5}}{2}$

Scanned by CamScanner

$$F(n) = C_1\left(\frac{1+\sqrt{5}}{2}\right)^n + C_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

When roots are real & distinct,

Find $C_1$ and $C_2$ using $F(0)=0$, $F(1)=1$

$$F(0) = C_1\left(\frac{1+\sqrt{5}}{2}\right)^0 + C_2\left(\frac{1-\sqrt{5}}{2}\right)^0$$

$$F(0) = C_1(1) + C_2(1)$$

$$F(0) = C_1 + C_2 \longrightarrow (1) \quad \text{i.e } 0 = C_1 + C_2$$

$$F(1) = C_1\left(\frac{1+\sqrt{5}}{2}\right)^1 + C_2\left(\frac{1-\sqrt{5}}{2}\right)^1$$

$$1 = C_1\left(\frac{1+\sqrt{5}}{2}\right) + C_2\left(\frac{1-\sqrt{5}}{2}\right) \longrightarrow (2)$$

as $C_1 + C_2 = 0$, $C_1 = -C_2$

Substitute for $C_1$ in (2)

$$1 = -C_2\left(\frac{1+\sqrt{5}}{2}\right) + C_2\left(\frac{1-\sqrt{5}}{2}\right)$$

$$1 = \frac{C_2}{2}\left[-1-\sqrt{5}+1-\sqrt{5}\right]$$

$$1 = C_2(-2\sqrt{5})$$

$$C_2 = \frac{2}{-2\sqrt{5}} = -\frac{1}{\sqrt{5}} \quad \therefore \boxed{C_2 = -\frac{1}{\sqrt{5}}}$$

Scanned by CamScanner

$C_1 = -C_2 \Rightarrow C_1 = -\left(-\frac{1}{\sqrt{5}}\right)$

$$\boxed{C_1 = \frac{1}{\sqrt{5}}}$$

$$F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

Algorithm: // I/p: +ve integer n
// O/p: Fibonacci sequence:
if $n \leq 1$ return n
else return $F(n-1) + F(n-2)$

Number of additions required for computing F(n) is A(n) and one more addition to compute their sum.

$$A(n) = A(n-1) + A(n-2) + 1$$

$$A(0) = 0, \quad A(1) = 0$$

$$A(n) - A(n-1) - A(n-2) = 1$$

This is non-homogenous eqtn, to convert into homogenous, rewrite

$$[A(n)+1] - [A(n-1)+1] - [A(n-2)+1] = 0$$

Substitute $B(n) = A(n)+1$

$$B(n) - B(n-1) - B(n-2) = 0$$

$$B(0) = 1, \quad B(1) = 1$$

$B(n) = A(n)+1$

$B(n-1) = A(n-1)+1$

$B(n-2) = A(n-2)+1$

$\therefore A(n) = B(n) - 1$

Because $F(0), F(1)$ but $B(0)$ [illegible]

$A(n) = F(n+1) - 1 = \frac{1}{\sqrt{5}}\left(\phi^{n+1} - \hat{\phi}^{n+1}\right) - 1$

Scanned by CamScanner

# Theorem

## Useful Property involving the Asymptotic Notations

**Theorem Statement:** If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$ then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

It states that function $t_1$ belongs to order of growth $g_1(n)$ & $t_2$ belongs to order of growth $g_2(n)$ then if we add both of them i.e $t_1 + t_2$, then the result of addition's order will be the order of function which is having higher order.

Eg: $t_1(n) \in O(n)$

$t_2(n) \in O(n^2)$

then $t_1(n) + t_2(n) \in O(n^2)$

**Proof:** $a_1, b_1, a_2$ & $b_2$ be arbitrary real nos, if $a_1 \le b_1$ & $a_2 \le b_2$ then $a_1 + a_2 \le 2 \max\{b_1, b_2\}$

Since $t_1(n) \in O(g_1(n))$ there exist some constant $c_1$ & +ve integer $n_1$ such that

$$\frac{t_1(n)}{a_1} \le \frac{c_1 g_1(n)}{b_1} \text{ for all } n \ge n_1$$

similarly $\frac{t_2(n)}{b_2} \le \frac{c_2 g_2(n)}{b_2}$ $n \ge n_2$

Let us denote $c_3 = \max\{c_1, c_2\}$ eg:

$$t_1(n) + t_2(n) \le c_1 g_1(n) + c_2 g_2(n)$$

$$\le c_3 g_1(n) + c_3 g_2(n) \quad (\text{Substitute } c_1, c_2 \text{ by } (3))$$

Scanned by CamScanner

$\leq c_3 [g_1(n) + g_2(n)]$ Apply (1)

$\leq c_3 \, 2 \max\{g_1(n), g_2(n)\}$

$b_1 \quad b_2$

hence $\boxed{t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})}$

why $2\max\{g_1(n), g_2(n)\}$, here 2 for all values it wont hold good. for Eg: if $t_1 = a_1 + b_1$, $t_2 = a_2 + b_2$

where $a_1 = 2, b_1 = 3$

$a_2 = 3, b_2 = 5$

Then $t_1 + t_2 = [a_1 + b_1] + [a_2 + b_2]$

$a_1 + b_1 = 5$, $a_2 + b_2 = 8$

$t_1 + t_2 = 13 \leq 8$ (NO, 8 wont hold good)

## Using limits for comparing order of growth

O, $\Omega$, $\Theta$ are notations used for order of growth, they are rarely used. Better method to compare order of growth is based on computing limit of the ratio of two functions. 3 principal cases arise

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 \Rightarrow t(n) \text{ has smaller order of growth than } g(n) \in O \\ c \text{ i.e constant}, t(n) \text{ has same order of growth as } g(n) \in \Theta \\ \infty \; t(n) \text{ has larger order of growth than } g(n) \in \Omega \end{cases}$$

L. Hopitali Rule

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \lim_{n\to\infty} \frac{t'(n)}{g'(n)}$$

Eg: $\lim_{x\to 0} \frac{e^x - 1}{x} = \frac{e^0 - 1}{0} = \frac{1-1}{0}$

$= \frac{0}{0}$ for $\frac{0}{0}$ form

$\lim_{x\to 0} \frac{\frac{d}{dx}(e^x) - \frac{d}{dx}(1)}{\frac{d}{dx}(x)} = \lim_{x\to 0} \frac{e^x - 0}{1} = e^0 = 1$

Scanned by CamScanner

L. Hopital's Rule

$$= \lim_{n \to \infty} \frac{6n}{5}$$

$$= \frac{6}{5} \times \infty = \infty$$

Stirling's formula

$$n! \simeq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

for large value of n

## 2) Brute Force

Brute force is a straightforward approach to solving a problem, directly based on problem's statement and definitions of the concepts involved.

Eg: Consider exponential problem, compute $a^n$ for a given number 'a' and +ve integer 'n'.

By definition of exponentiation,

$a^n =$

Scanned by CamScanner