UPCP

Date
Page

Analysis & Designing Of Algorithm

What is an Algorithm ?

An algorithm is a finite set of instructions i.e followed to accomplish task. All algorithm not satisfied the following

- Input - Zero or more quantity externally
- Output - At least one quantity is produced
- Finiteness - If it trace out the instruction algorithm then for all cases of the algo terminate after a finite no. of steps.
- Effectiveness - Every instruction must be very basic so that it can be carried out in principle by using pencil and paper

There are four area to distinguish algorithm

2) How to validate algorithm
   ↳ correct answer as your input
1) How to device algorithm
3) How to analyse algorithm
   — Space and Time complexity

4) How to test program

Time Complexity and Space Complexity

Performance / evaluation / analysis.

1) Space complexity - Space needed by each algo is sum of following components.

$$S(p) = C + S_p$$

1. Fixed part - Independent on I/P and O/P
2. Variable part - Dependent on I/P and O/P

Pseudo code for sum of n numbers

Total Set	Freq	S/e	
0	0	0	algo sum(a, n)
0	0	0	{
1	1	1	s = 0; → Fixed part
n+1	n+1	1	for i = 1 to n → { Variable part /
n	n	1	s = s + a[i] → { Instance variable / Instance characteristics
1	1	1	return (s) → fixed part.
0	0	0	}
2n+3	Total steps		

Time Complexity — T_p

Time t_p taken by program p is the sum of compile time and run time.

Compile time does not depend on instance characteristics. Compile program will run without re-compilation.

Run time — We concern ourself with just the runtime of our program. This run time is denoted by t_p.

$$t_p(n) = C_a \cdot ADD(n) + C_s \cdot Sub(n) + C_m \cdot MUL(n)$$

C_a: Step per execution; $ADD(n)$: Frequency

UPCP

Date
Page

Performance Analysis

Best Case

Worst Case

Average Case

There types of Notations (Asymptotic Notation)

Big O → gives upper bound → worst case (O)

Omega Notation → gives lower bound → best case (Ω)

Theta Notation → average case (Θ)

Big O :- $f(n) = O(g(n))$ such that

iff there exists positive constants C and n_0 and $f(n) \leq C.g(n)$

$\forall n$ where $n \geq n_0$

$3n+2 = O(n)$

if $3n+2 \leq 4n$, $n \geq n_0$

Here $c = 4$, $n_0 = 2$

Omega Notation - The function $f(n) = \Omega(g(n))$

iff there exists positive constants c and n_0 such that

$f(n) \geq cg(n)$ $\forall n$ where $n \geq n_0$

$3n+2 = \Omega(n)$

$3n+2 \geq 3n$ $\forall n \geq n_0$

$n_0 = 1$

Theta Notation - The function $f(n) = \Theta(g(n))$

iff there exists positive constants c_1, c_2 and n_0.

such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\forall n$ where $n \geq n_0$

$3n+2 = \theta(n)$

where $3n+2 \geq 3n \quad \forall n \geq 2$

and $3n+2 \leq 4n$ where $n \geq 2$

$c_1 = 3, \quad c_2 = 4, \quad n_0 = 2$

Little O and Little o - The function $f(n)$ is equal to $g(n)$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Little Omega - The function $f(n) = \omega g(n)$ iff $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$

Complexity of Recurrence Eqn

1) Substitution Method
2) Recursive Tree
3) Master Theorem

1) $T(n) = T(n-1) + 1 \qquad T(1) = \theta(1)$

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

$T(n) = T(n-2) + 2$
$= T(n-3) + 3$

$T(n) = T(n-i) + i \quad , i \geq 1$

$T(1) = 1$

$n - i = 1$
$i = n-1$

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

2) $$T(n) = \begin{cases} a & n = 1 \\ 2T(n/2) + Cn & n > 1 \end{cases}$$

$$T(n) = 2T(n/2) + Cn$$

$$= 2\left[2T\left(\frac{n}{4}\right) + \frac{Cn}{2}\right] + Cn$$

$$= 4T\left(\frac{n}{4}\right) + 2Cn$$

$$= 4\left\{2\left[T\left(\frac{n}{8}\right)\right] + \frac{Cn}{4}\right\} + 2Cn$$

$$= 8T\left(\frac{n}{8}\right) + 3Cn$$

$$= 2^i T\left(\frac{n}{2^i}\right) + iCn \quad , i \geq 1$$

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

$$n = 2^i$$

$$\log n = i \log 2$$

$$i = \log n$$

$$= 2^{\log n} T(1) + \log n \, Cn$$

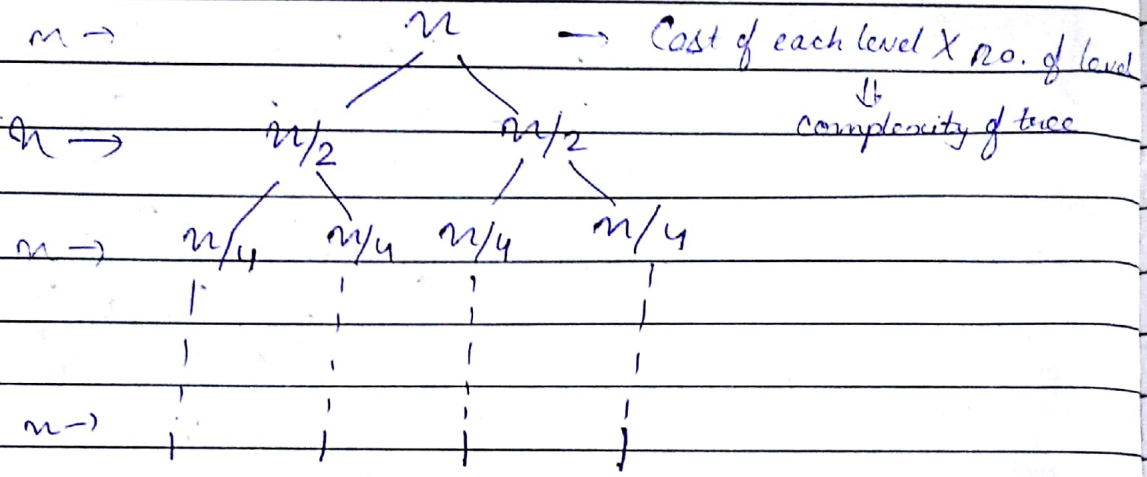
$$= a 2^{\log n} + Cn \log n$$

$$= an + Cn \log n$$

$$= O(n \log n)$$

$$\log n < n < n \log n < n^2$$

Recurrence Tree - It is a pictorial representation of iteration method.



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

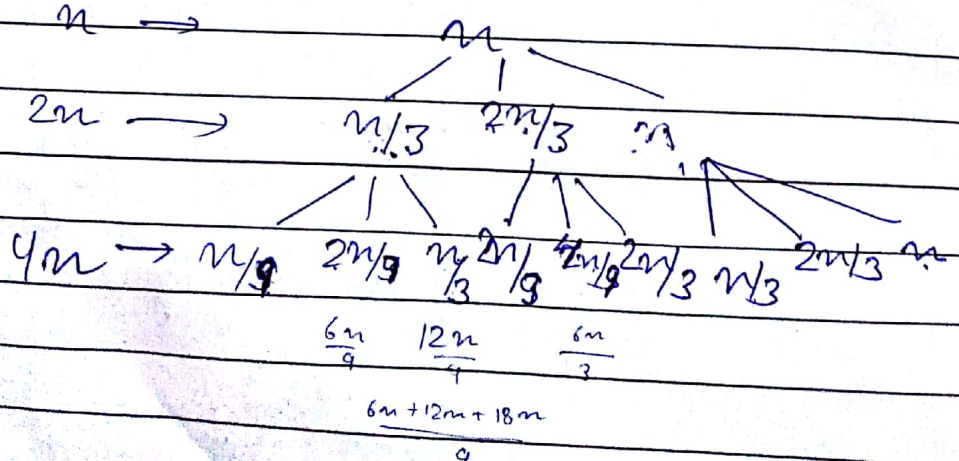
$$n = 2^i$$

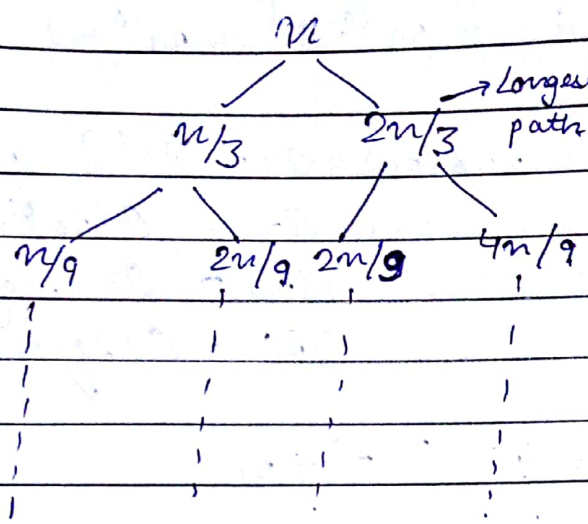
$$\log n = i \log 2$$

$$i = \log_2 n$$

Complexity = $n \log n$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$$





$\left(\frac{2}{3}\right)^k n = 1$

$n = \left(\frac{3}{2}\right)^k$

$\log n = k \log \frac{3}{2}$

$k = \log_{3/2} n$

No. of level

Complexity = No. of level × Cost of each level

$= n \log_{3/2} n$

Master Method :-

$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad a \geq 1, b \geq 1$

1) If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$

2) If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$

3) If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ $\varepsilon > 0$ then $T(n) = \Theta(f(n))$ if $af(n/b) \leq c f(n)$ $\{c < 1\}$

eg - $T(n) = T\left(\frac{3n}{4}\right) + 1$ and $T(1) = \Theta(1)$

$a = 1, \quad b = \frac{4}{3}, \quad f(n) = 1$

$n^{\log_b a} = n^{\log_{\frac{4}{3}} 1} = n^0 = 1$

$\Rightarrow f(n) = O(n^{\log_b a})$ acc to 2nd cond^n

then $T(n) = \Theta(n^{\log_b a} \log n)$

$T(n) = \Theta(1 \cdot \log n)$

$T(n) = \Theta(\log n)$

2) $T(n) = 4T(n/2) + n$

$a = 4, \; b = 2$

$f(n) = n$

$n^{\log_b a} = n^{\log_2 4} = n^{2 \log_2 2} = n^2$

We will subtract constant from n^2 to get n

$\therefore$ I case

$T(n) = \Theta(n^{\log \frac{a}{b}})$

$T(n) = \Theta(n^2)$

3) $T(n) = 4T(n/2) + n^3$

$a = 4, \; b = 2, \; f(n) = n^3$

$n^{\log_b a} = n^{\log_2 4} = n^2$

$n^{k+\epsilon} = n^3$

III case

$T(n) = \Theta(n^{\log \frac{a}{b}})$

$= \Theta(n^2)$

$T(n) = \Theta(f(n))$

$= \Theta(n^3)$

Sets and Dijoint Sets :-

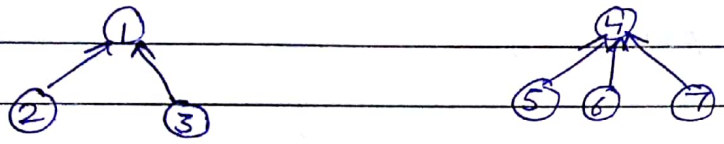
Operations :-

makeset (x) → Create new set whose only member is x.

find (x) → finds the representative of the element x mentioned in find(x)

union (S_1, S_2)

{1, 2, 3} {4, 5, 6, 7}



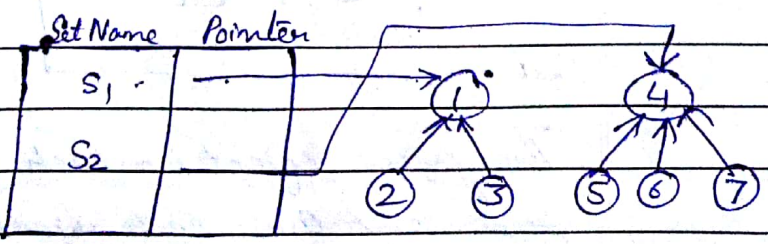
i	[1]	[2]	[3]	[4]	[5]	[6]	[7]
p	-1	1	1	-1	4	4	4

A dijoint set data structure maintains a collection of dijoint dynamic sets. Each set is identified by a representative which is some member of a set.

makeset → Since the sets are dijoint we require that x is not already be in some other set.

union (x, y) → union of two sets choose either S(x) or S_y (representative of set x or y) as the new representative.

findset(x) → This returns a pointer to the representative of the set containing x.



For union and find set (x) we ignore set name and identify set by the root of the tree representing the set.

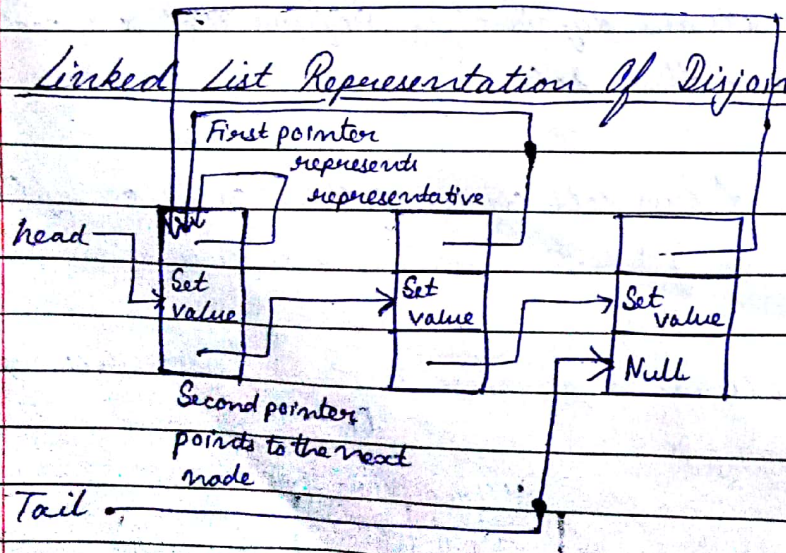
Algo Of Union :-

```
Simple Union (i, j)
{
    p[i] = j;
}
```

Algo Of Find :-

```
Algo find (i)
{
    while (p[i] >= 0) do i = P[i];
    return i;
}
```

Linked List Representation Of Disjoint Set :-



The first object in each linked list serves as its set representative. Each object in the linked list contains a

UPCP

Date
Page

set member a pointer to the object containing the next set member and a pointer back to the representative. Each list maintains pointers head to the representative and tail to the last object in the list

Implementation Of Union Operation Using Linked List

Set representation takes significantly more time than make set or find set.

Union (x, y)

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

Union (x, y) by appending x list on to the end of y list. We must update the pointer to the representative for each object originally on x list. which takes time.

Union $(x_1, x_2) = 1$

$(x_1, x_2, x_3) = 2$

$(x_1, x_2, x_3, x_4) = 3$

$n-1$

$(1+2+3+\cdots+n-1)$

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

Complexity is n^2

$O(n^2)$

Q Solve the recurrence relations :-

1) $T(n) = T(n-1) + 1$, $T(n) = \theta(1)$

$$T(n) = T(n-1) + 1 \quad -①$$
$$T(n-1) = T(n-1-1) + 1$$
$$T(n-1) = T(n-2) + 1$$

$$T(n) = T(n-2) + 1 + 1$$
$$T(n) = T(n-2) + 2 \quad -②$$

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

$$T(n) = T(n-3) + 1 + 2$$
$$T(n) = T(n-3) + 3 \quad -③$$

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

$\because T(1) = 1 \Rightarrow n - i = 1$

$$i = n - 1$$

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

Complexity $\boxed{T(n) = O(n)}$

2) $T(n) = 2T\left(\frac{n}{3}\right) + n$, $T(1) = O(n)$

UPCP

Date
Page

$$T(n) = 2T\left(\frac{n}{3}\right) + n \quad -①$$

$$T\left(\frac{n}{3}\right) = 2T\left(\frac{n}{9}\right) + \frac{n}{3}$$

$$T(n) = 2\left[2T\left(\frac{n}{9}\right) + \frac{n}{3}\right] + n$$

$$T(n) = 4T\left(\frac{n}{9}\right) + \frac{2n}{3} + n$$

$$T(n) = 4T\left(\frac{n}{9}\right) + \frac{5n}{3} \quad -②$$

$$T\left(\frac{n}{9}\right) = 2T\left(\frac{n}{27}\right) + \frac{n}{9}$$

$$T(n) = 4\left[2T\left(\frac{n}{27}\right) + \frac{n}{9}\right] + \frac{5n}{3}$$

$$T(n) = 8T\left(\frac{n}{27}\right) + \frac{4n}{9} + \frac{5n}{3}$$

$$T(n) = 8T\left(\frac{n}{27}\right) + \frac{19n}{9} \quad -③$$

$n \; \frac{2n}{3} \; \frac{4n}{9}$

$$T(n) = 2^i T\left(\frac{n}{3^i}\right) + \left[\frac{\left(\frac{2n}{3}\right)^{\frac{\log n}{\log 3}} - 1}{\frac{2n-3}{3}}\right] n$$

$$T(n) = 2^{\log_3 n} O(n) + \frac{\left(\frac{2n}{3}\right)^{\log_3 n} - 1}{\frac{2n-3}{3}}$$

Complexity = $O(n)$

5) $T(n) = 2T\left(\frac{n}{2}\right) + n - 1$

$a = 2 \quad b = 2 \quad f(n) = n - 1$

$n^{\log_b a} = n^{\log_2 2} = n$

Second case

$T(n) = \Theta(n^{\log_b a} \log n)$

$T(n) = \Theta(n \log n)$

3) $T(n) = 2T\left(\frac{n}{2}\right) + n$

$T(n) = 2T\left(\frac{n}{2}\right) + n \quad -①$

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}$

$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$

$T(n) = 2.2.T\left(\frac{n}{4}\right) + 2n \quad -②$

$T(n) = 2.2.2.T\left(\frac{n}{8}\right) + 3n \quad -③$

$T(n) = 2^i T\left(\frac{n}{2^i}\right) + in$

$T\left(\frac{n}{2^i}\right)$

UPCP

Date
Page

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

$$n = 2^i$$

$$i = \log_2 n$$

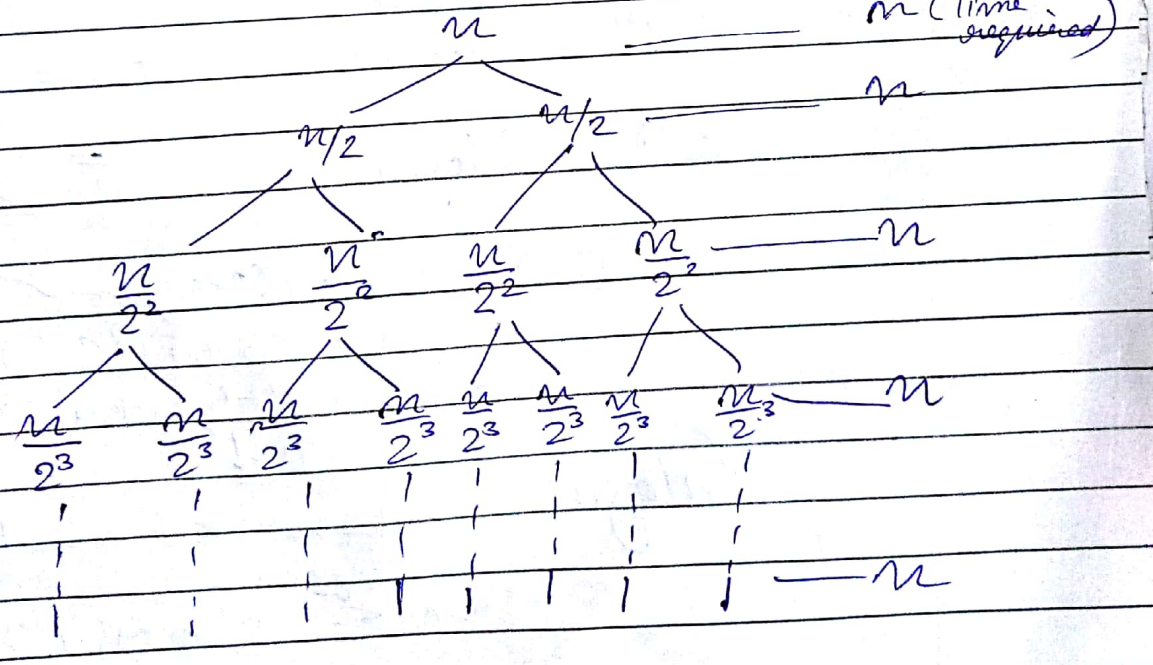
$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n\log_2 n \quad \{2^{\log_2 n} = 1\}$$

$$T(n) = 2^{\log_2 n} + n\log_2 n = n + n\log_2 n$$

$$O = n\log_2 n \qquad O(n\log n)$$

4) $T(n) = 4T\left(\frac{n}{2}\right) + n$ (Recursive tree)

3) $$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$



k no. of levels $\frac{n}{2^k} = 1$

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

Time $= nk = n\log_2 n \quad O(n\log n)$

Date
Page

Matrix Multiplication –

$$\underset{2\times3}{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}} \quad \underset{3\times5}{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}} = \underset{2\times5}{\begin{bmatrix} & \\ & \end{bmatrix}}$$

$$C[i,j] = \sum_{k=0}^{n} A[ik]\,B[kj]$$

$$C_{12} = A_{11}\times B_{11} + A_{12}\times B_{21}$$

$O(n^3)$ → 3 loops for i, j and k.

Divide and Conquer

$$8T\left(\frac{n}{2}\right) + n^2 \Rightarrow O(n^3)$$

$$\begin{bmatrix} [A_{11}] & [A_{12}] \\ [A_{21}] & [A_{22}] \end{bmatrix} \begin{bmatrix} [B_{11}] & [B_{12}] \\ [B_{21}] & [B_{22}] \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Strassen Matrix Multiplication :-

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{11} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + V$$

$$T(n/2) + n^2$$

$$O(n^{2.81})$$

$n^{\log_2 7}$

$n^{2.81}$

$$A = \begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} 6 & 7 \\ 3 & 8 \end{bmatrix}$$

$P = (1+5)(6+8) = 6 \times 14 = 84$

$Q = (7+5) \times 6 = 12 \times 6 = 72$

$R = 1(7-8) = -1$

$S = 5(3-6) = -15$

$T = (1+3)8 = 32$

$U = (7-1)(6+7) = 78$

$V = (3-5)(3+8) = -22$

$C_{11} = 84 - 15 - 32 - 22 = 84 - 69 = 15$

$C_{12} = R + T = -1 + 32 = 31$

$C_{21} = Q + S = 72 - 15 = 57$

$C_{22} = P + R - Q + V = 84 - 1 - 72 + 78 = 89$

01 Sunday January'12
New Year's Day

Unit - I

Why Study Algorithm Design and Analysis ?

- It is basically an analytical and conceptual subject
- Plays key role in modern technological innovation.

Algorithm and Function :-

An algorithm can be converted into a function.

Problem :- Find smallest element in an array.

Algorithm :-

	No. of steps
1. Set min = A[0]	1
2. minIndex = 0	1
3. for (i = 1 to n-1)	n-1
if (A[i] < A(minIndex)	
Set minIndex = i	1
4. return minIndex.	3+n-1
	= n+2

$f(n) = n+2 \rightarrow$ linear function

Complexity of an algorithm is a function describing the efficiency of an algorithm in terms of amount of data the algorithm must process

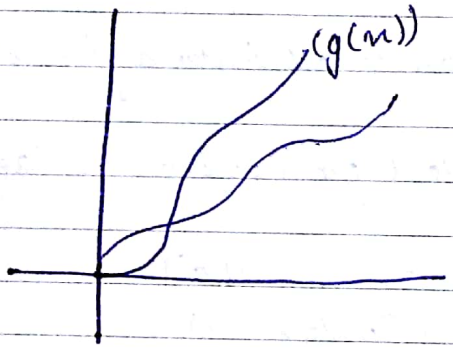
Time Complexity → no. of machine instruction executed
Space Complexity → memory used.

Time Complexity is a function describing the amount of time an algorithm takes in terms of amount of input to the algorithm.

2012

Asymptotic Notation - formal way to speak about functions and clarify them. We will use it to describe running time of algorithm.

O (Big-O) Upper bound Value (Worst Case)



O(g(n)) is the set of all functions with a smaller or the same order of growth as g(n)

$O(n^2) = \{2n^2, 2n+1, \log n \ldots\}$

$O(g(n)) = \{f(n): 0 \le f(n) \le g(n)\}$ where $c > 0, n_0 \ge 1$

for eg:- $f(n) = n+2$

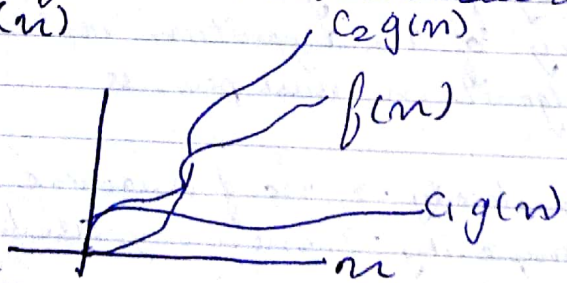
$n+2 \le 3n$ — $g(n)$

$c = 3, n \ge n_0$

$n \ge 1$ $\boxed{n_0 = 1}$

It will satisfy for $g(n) = n^2, n^3, \ldots$ also but we will take closest value.

O(n)



Evening

O (Theta)
(Average Case)

$O(g(n)) : [f(n): c_1 g(n) \le f(n) \le c_2 g(n)]$
$c_1, c_2 > 0, n \ge n_0$

$f(n) = n+2$

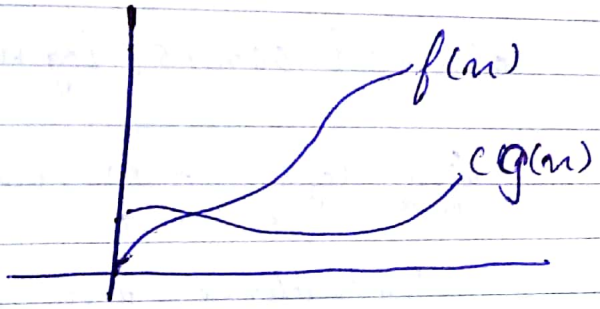
$c_1 = 1 \quad g(n) = n$

$1n \le n+2 \le 3n \quad , n \ge 1$

$\Rightarrow n_0 = 1, \; c_1 = 1, \; c_2 = 3$

$\Theta(n)$

Ω (Big omega) – Lower bound value (Best Case)



$\Omega(g(n)) = [f(n) : 0 \le cg(n) \le f(n)]$
$c > 0, \; n \ge n_0$

$f(n) = n+2$

$c = 1$

$1, n \le n+2 \quad , n \ge 1$

$\Omega(n)$

Little - Oh Notation –

$o(g(n)) = \{ f(n)$: for any +ve constant $c > 0, n_0 > 0$
such that $0 \le f(n) < cg(n) \; \forall n \ge n_0$
Or
$o(n)$ is the set of all f^n with a small rate of growth than $g(n)$

2012

$2n \simeq O(n^2)$ but $2n^2 \neq o(n^2)$

$f(n) \simeq g(n) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$f(n) = 2n$

$g(n) = n^2$

$$\lim_{n\to\infty} \frac{2n}{n^2} = \lim_{n\to\infty} \frac{2}{n} = 0$$

$o(n^2) = \{100n+5, \log n, \ldots\}$

$$\lim_{n\to\infty} \frac{\log n}{n^2} = \frac{\infty}{\infty} \text{ indefinite}$$

For indefinite $\lim_{n\to\infty} \frac{f'(n)}{g'(n)}$

$$= \lim_{n\to\infty} \frac{1/n}{2n} = \lim_{n\to\infty} \frac{1}{2n^2} = 0$$

Little Omega (ω)

$\omega(g(n)) = [f(n)$: for any +ve constant $c > 0$, $n_0 > 0$ such that $0 \leq cg(n) < f(n) \ \forall\ n > n_0]$

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

$\frac{n^2}{2} = \omega(n)$ $\quad \frac{n^2}{2} \neq \omega(n^2)$

$f(n) = 3^n$ $\quad g(n) = 2^n$

$$\lim_{n\to\infty} \frac{3^n}{2^n} = \lim_{n\to\infty} \left(\frac{3}{2}\right)^n = \infty$$

$3^n \simeq \omega(2^n)$

$f(n) = n^2 \simeq \omega(\log n)$

$g(n) = \log n$

$$\lim_{n\to\infty} \frac{n^2}{\log n} = \frac{\infty}{\infty} \text{ (Indefinite)}$$

$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{2n}{1/n} = \lim_{n\to\infty} 2n^2 = \infty$$

$n^2 \simeq \omega(\log n)$

Recurrence Relation :- A recurrence relation is an equation that recursively define a sequence where next term is a function of previous term

1) $a_1 = a_0 + 2$
$a_2 = a_1 + 2$
$a_3 = a_2 + 2$
- - - - - -
- - - - - - -
$a_n = a_{n-1} + 2$

2) $a_n = 3.2^n$
$a_n = a_{n-1}.3$

3) Fibonacci Series
$F_n = F_{n-1} + F_{n-2}$
$F_2 = F_1 + F_0$

```
int Fib (int n)
{ if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return (Fib(n-1) + Fib(n-2));
```

eg → $a_k = a_{k-1} + 2$ with $a_0 = 1$

$k = 1, \ldots, n$

$a_1 = a_0 + 2 = 1 + 2$

$a_2 = a_1 + 2 = 1 + 2 + 2 = 1 + 2.2$

$a_3 = a_2 + 2 = 1 + 2 + 2 + 2 = 1 + 3.2$

$a_n = a_{n-1} + 2 = 1 + 2 + 2 + 2 + \ldots = 1 + n.2$

$a_n = 1 + 2n$

$O(n)$ [By Brute Force Method or Method Of Iteration]

Master Method (in class notes)

Subtract and Conquer Theorem:

Let $T(n)$ be a f^n defined on +ve n

$$T(n) \leq \begin{cases} c & \text{if } n \leq 1 \\ aT(n-b) + f(n) & \text{if } n > 1 \end{cases}$$

for some constants $c, a > 0, b > 0, d \geq 0$ and $f^n f(n)$

if $f(n)$ is in $O(n^d)$ then

$$T(n) \simeq \begin{cases} O(n^d) & \text{if } a < 1 \\ O(n^{d+1}) & \text{if } a = 1 \\ O(n^d a^{n/b}) & \text{if } a > 1 \end{cases}$$

eg- $T(n) = T(n-1) + n$ if $n > 1$ and $T(1) = 1$

$d = 1$

$a = 1 \quad c = 1, b = 1$

$T(n) \simeq O(n^2)$

Proof :-

$$T(n) = \begin{cases} c & \text{if } n \le 1 \\ aT(n-b)+f(n) & \text{if } n > 1 \end{cases}$$

$$T(n) \le a[aT(n-2b)+f(n-b)]+f(n)$$
$$= a^2 T(n-2b) + af(n-b) + f(n)$$
$$= a^3 T(n-3b) + a^2 f(n-2b) + af(n-b) + f(n)$$
$$T(n) \le \sum_{i=0}^{n/b} a^i f(n-ib) + \text{constant}$$

$f(n-ib) \to O(n-ib)$

$$T(n) \simeq O\left\{(n^d)\sum_{i=0}^{n/b} a^i\right\}$$

$$\sum_{i=0}^{n/b} a^i \to \begin{cases} O(1) & \text{if } a<1 \\ O(n) & \text{if } a=1 \\ O(a^{n/b}) & \text{if } a>1 \end{cases}$$

Iteration Method :

Q. $T(n) = 8T(n/2) + n^2$ and $T(1)=1$

Sol $T(n/2) = 8T\left(\frac{n/2}{2}\right) + (n/2)^2$

$$T(n) = 8\left[8T\left(\frac{n}{2^2}\right) + \frac{n^2}{2^2}\right] + n^2$$

$$= n^2 + 8^2 T\left(\frac{n}{2}\right)^2 + 8\left(\frac{n}{2}\right)^2$$

$$= 3n^2 + 8^2 T\left(\frac{n}{2}\right)^2 = n^2 + 2n^2 + 8^2 T\left(\frac{n}{2}\right)^2$$

$$= n^2 + 2n^2 + 8^2\left(8T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right)$$

$$= n^2 + 2n^2 + 2^2 n^2 + 8^3 T\left(\frac{n}{2^3}\right)$$

January'12 09 Monday

Week 02
Day (009-357)

Jan 01

S	M	T	W	T	F
01	02	03	04	05	06
08	09	10	11	12	13
15	16	17	18	19	20
22	23	24	25	26	27
29	30	31			

$T(n) = n^2 + 2n^2 + 2^2 n^2 + \cdots + 2^{\log n - 1} n^2 + 8^{\log n}$ { i^{th} terms; $\frac{n}{2^i} = 1 \Rightarrow i = \log n$

$= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}$ [$8^i T(1) = 1 \cdot 8^{\log n}$; $2^{\log n - 1} - 1$

$\simeq n^2 \Theta(2^{\log n - 1}) + (2^{\log n})^3$

$\simeq n^2 (2^{\log n - 1}) + n^3$

$T(n) \simeq \Theta(n^3)$

Recursive Tree Method (In Class Notes)

Substitution Method :-

- Guess a particular solution in assymptotic form for a given recurrence relation.
- Verify it using mathematical induction.

Q. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n > 1 \end{cases}$

$T(n) = T(n-1) + n$ and $T(1) = 1$ if $(n > 1)$

Evening Sol. $T(n) = O(n^2)$

$T(n) \le C(n^2)$

$g(n) = n^2$

$C > 0, \; n > n_0$

$T(\alpha) \le C\alpha^2$ for $1 \le \alpha \le n-1$

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

$\le C(n-1)^2 + n$

$\le C(n^2 - 2n + 1) + n$

2012

$= cn^2 - 2cn + c + n$

$T(n) \leq cn^2$

$n \geq 1, c = 1$

$n^2 - 2n + 1 + n$

$1 - 2 + 1 + 1$

$= 1$

$n = 2$

$4 - 4 + 1 + 2$

$= 3$

$cn^2 = 4$

$\Rightarrow T(n) \leq cn^2$ (Verified)

$T(n) \simeq O(n^2)$

$T(n) = \Omega n^2$

$T(n) \geq cn^2$ can be proved by induction method

$T(n) = a\,T(n/b) + f(n)$

① if $a = b = 2$

$T(n) = O(f(n) \log n)$

② If $a = 1$, $b =$ any value

$T(n) = O(\log n)$

③ If $a \neq 1$, $b \neq a$

$T(n) = O(n \cdot f(n))$

January'12

11 Wednesday

Week 02
Day (011-355)

Jan

S	M	T	W	T	F	S
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Insertion Method :- (Implace Sorting)

```
Insertion Sort (a)
{
  for (k=1; k<=n; k++)
  {
    temp = a[k];
    j = k-1;
    while (temp < a[j] && (j>=0))
    {
      a[j+1] = a[j];
      j = j-1;
    }
    a[j+1] = temp;
  }
}
```

9	5	6	0	10
0	1	2	3	4

j = 0
temp < a[0]
5 < 9

5	9	6	0	10

temp = a[2] = 6
j = 2-1 = 1
temp < a[1]
6 < 9

5	6	9	0	10

temp = 0

0	5	6	9	10

k = 4
temp = 10

0	5	6	9	10

Worst Case (Comparison → Swapping)

k = 1 ⇒ 1+1 = 2.1
k = 2 ⇒ 2+2 = 2.2
k = 3 ⇒ 3+3 = 2.3
;
k = n-1 ⇒

$$2(1+2+3+\dots+n-1)$$

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

Best Case

6 7 8 9
1 1 1 1

$$1+1+1+1+\dots+n-1 = \Omega(n)$$

↓ Best Case

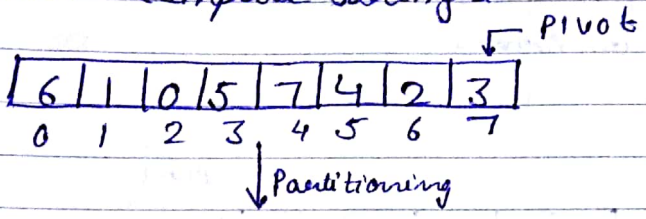
Average → $O(n^2)$

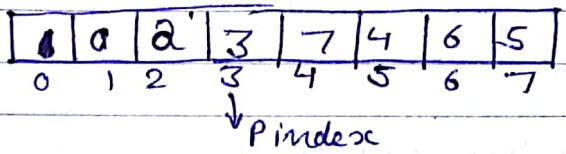
Space Complexity $O(1)$

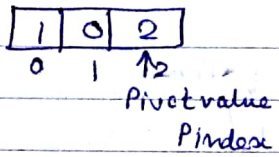
Evening

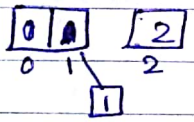
2012

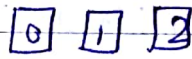
Quick Sort :- (Implace Sorting)









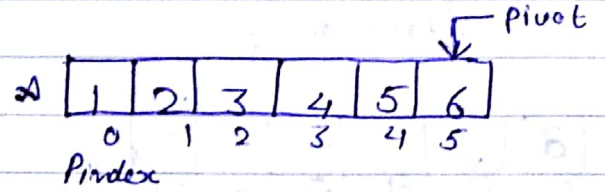



```
Quick Sort (A, start, end)
{ if (start < end) {
Pindex ← Partition ( A, start, end)
   Quick Sort ( A, start, Pindex - 1)
   Quick Sort (A Pindex + 1, end)
} }

Partition (A, start, end)
{
   Pivot ← A[end]
   Pindex ← Start
   for (i = Start to end - 1)
     { if (A[i] < = Pivot)
        { swap (A[i], A[Pindex])
     } } Pindex ← Pindex + 1
```

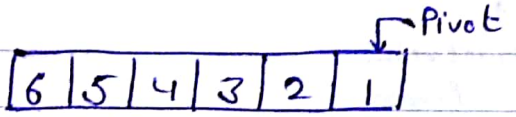
Swap [A[Pindex], A[end])
return Pindex
}

Analysis



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

Worst Case $= O(n^2)$



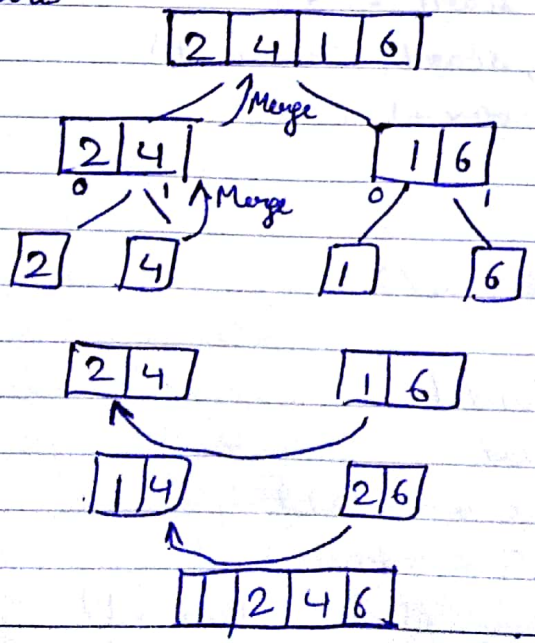
$T(n) = T(n-1) + O(n)$

$= O(n^2)$

Best Case → $T(n) = 2T(n/2) + O(n)$

$\simeq O(n \log n)$

Merge Sort :-



```
Merge Sort (A)
{  n = length (A)
   If [n < 2] return                                  } C1
      mid = n/2
      left = array of size (mid)
      right = array of size (mid)
      for (i = 0 to mid - 1)                          } C2.n
          left [i] = A[i]
      for (i = mid to n-1)
          right [i - mid] = A[i]
      mergesort (left)      → recursively T(n/2)
      mergesort (right)       T(n/2)
      mergesort (left, right, A)   → C3n + C4
}

Merge (L, R, A)
{  nL = length (L)
   nR = length (R)
   i = 0, j = 0, k = 0
   while (i < nL && j < nR)
   {  if (L[i] <= R[j])
         {  A[k] = L[i]
            i = i + 1; }
      else
         {  A[K] = R[j]
            j = j + 1; }
      K = K + 1
   }
   While (i < nL)
   {  A[k] = L[i]; i++; k++; }
   While (j < nR)
   {  A[K] = R[j]; j++; K++; }
}
```

January'12

16 Monday

Week 03
Day (016-350)

Jan

S	M	T	W	T	F	S
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Analysis :-

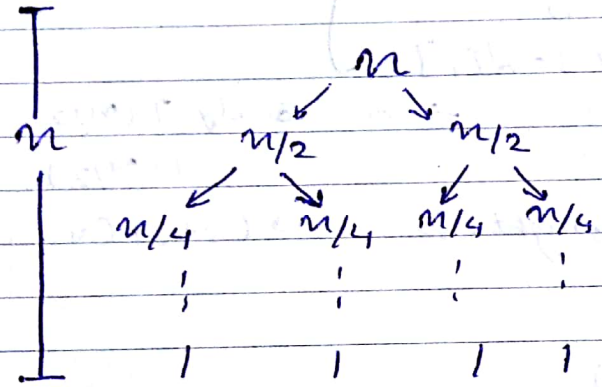
$$T(n) = C_1 + C_2 n + 2T(n/2) + C_3 n + C_4$$

$$T(1) = 1$$

$$C_1 + C_2 n + C_3 n + C_4 = O(n)$$

$$T(n) = 2T(n/2) + O(n)$$

$$\simeq O(n \log n)$$



Space Complexity

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

$$k = \log_2 n$$

If space is created for all arrays

$$n \times \log n = O(n \log n)$$

If space is created and then erased while merging and utilize for other half.

$$n + \frac{n}{2} + \frac{n}{4} + \dots$$

$$n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right)$$

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

$$\simeq O(n)$$

Evening

2012

Strassen's Algorithm for Matrix Multiplication

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \quad Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}$$

Mmult(X, Y, n)
If n = 1 Output X * Y
else compute A, B, ----, H by $m = n/2$
X_1 = Mmult (A, E, n/2)
X_2 = Mmult (B, G, n/2)
X_3 = Mmult (A, F, n/2)
X_4 = Mmult (B, H, n/2) $8T(\frac{n}{2})$
X_5 = Mmult (C, E, n/2)
X_6 = Mmult (D, G, n/2)
X_7 = Mmult (C, F, n/2)
X_8 = Mmult (D, H, n/2)
$Z^{11} = X_1 + X_2$
$Z^{12} = X_3 + X_4$ $\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4}$ $\theta(n^2)$
$Z^{21} = X_5 + X_6$
$Z^{22} = X_7 + X_8$
Output Z
End if

$$T(n) = 8T\left(\frac{n}{2}\right) + \theta(n^2)$$
$$= \theta(n^3)$$

Strassen
If n = 1 output X * Y
else compute A, B, C, ---, H by n/2
P_1 = Strassen (A, F − H)
P_2 = Strassen (H, A + B)
P_3 = Strassen (E, C + D)
P_4 = Strassen (D, G − E)
P_5 = Strassen (A + D, E + F)
P_6 = Strassen (B − D, G + H)
P_7 = Strassen (A − C, E + F)
$Z^{11} = P_6 + P_5 + P_4 - P_2$
$Z^{12} = P_1 + P_2$
$Z^{21} = P_3 + P_4$
$Z^{22} = P_1 + P_5 - P_3 - P_7$
Output z
End if

$$T(n) = 7T\left(\frac{n}{2}\right) + \theta(n^2)$$
$$= \theta(n^{2.32})$$
$$= \theta(n^2)$$

P_1 = A(F − H)
P_2 = H(A + B)
⋮

2012

January'12 18 Wednesday

Week 03
Day (018-348)

Jan
01
S M T W T F S
01 02 03 04 05 06 07
08 09 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

Memorize

AHED

1. Diagonals for multiplication
   X4
   Last CR
   First CR

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

Row + Column −

$P_1 = A*(F-H)$
$P_2 = H*(A+B)$
$P_3 = E*(C+D)$
$P_4 = D*(G-E)$
$P_5 = (A+D)*(E+H)$
$P_6 = (B-D)*(G+H)$
$P_7 = (A-C)*(E+F)$

$$Z = \begin{bmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{bmatrix}$$

$$X = \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} \quad Y = \begin{pmatrix} 1 & 3 \\ 4 & 7 \end{pmatrix}$$

Evening

$P_1 = 2(3-7) = -8$
$P_2 = 7(2+4) = 42$
$P_3 = 1(3+5) = 8$
$P_4 = 5(4-1) = 15$
$P_5 = 56$
$P_6 = -11$
$P_7 = -4$

$Z = \begin{pmatrix} -11+56+15-42 & 42-8 \\ 8+25 & 56-8-15 \end{pmatrix}$

$$Z = \begin{pmatrix} +18 & 34 \\ 23 & 44 \end{pmatrix}$$

2012

19 Thursday January'12

Disjoint Set – also called union find or merge find set data structure
→ representing a dynamic collection of set $S = \{S_1, S_2, \ldots, S_n\}$

$S = \{a, b, c, d, e\}$
(a, b, c similar)

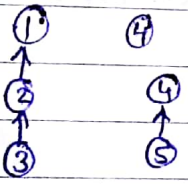
$S = \{S_1, S_2\}$

$S_1 = \{a, b, c\}$ $S_2 = \{d, e\}$

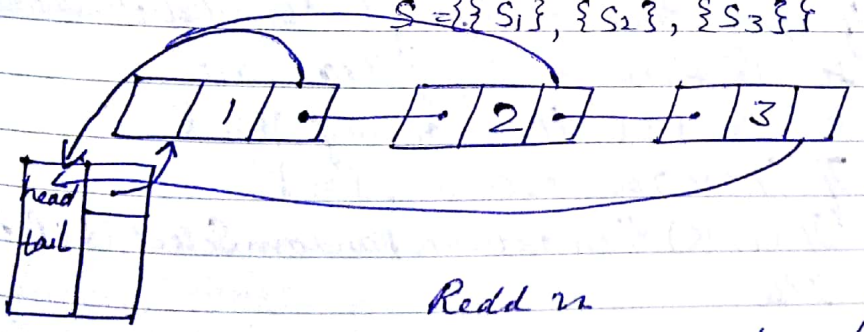
Make-Set (u)
Find-Set (u) → finds elements
Union (u, v)

① ② ③ ④ ⑤ ⑥

1R2
2R3



$S_1 = \{1, 2, 3\}$ $S_2 = \{4, 5\}$, $S_3 = \{6\}$
$S = \{\{S_1\}, \{S_2\}, \{S_3\}\}$



Redd n
For each object x from 1 to n
Make set (x).
Do until all related object are in a set
for each pair (x, y)
if (x ≅ y) (related to)
& (Find set(x) != Find set(y))
Union (x, y)

2012

January '12 20 Friday

Week 03
Day (020-346)

Jan

S	M	T	W	T	F	S
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Median and Order Statistics

K^{th} order statistics of a set of n elements is the K^{th} smallest element

$K = 1$ minimum

$K = n$ maximum

Median - $\lfloor \frac{n+1}{2} \rfloor$ (lower median, ceiling) or $\lceil \frac{n+1}{2} \rceil$ (upper median, floor) → half way point

Finding Order Statistics of K^{th} element

1) Randomized algorithm
2) Worst Case Linear time

1) Randomized Algorithm

start index, end index

```
Randomized Select (A, p, q, i)
if (p=q) then return A[p]          // base case
r = RandomPartition(A, p, q)
K = r - p + 1                      // K = rank of A[r]
If (i=K) then return A[r]
If (i<K) then return RandomSelect(A, p, r-1, i)
else
    return RandomizedSelect(A, r+1, q, i-K)
```

i^{th}

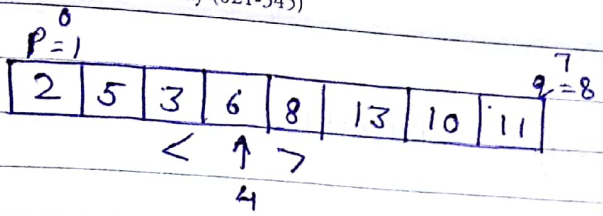
1	4	2	3

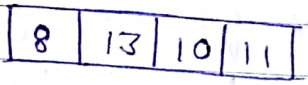
$i = 2$

Q. Find 7^{th} order statistics

6	10	13	5	8	3	2	11

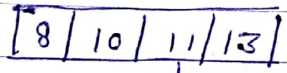
↑ r, Pivot





$k = 4 - 1 + 1$

$k = 4$



Unit-2
Dynamic Programming

Dynamic programming solves problems by combining the solutions to subproblem.
It applies when subproblem overlap.
It solves each subproblem just once and then saves its answer in table

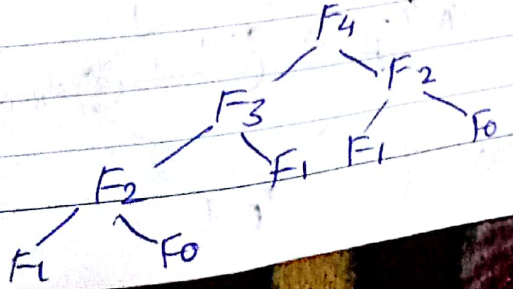
Steps :- 1) Characterize the structure of an optimal solution
2) Recursively define the value of an optimal solution
3) Compute the value of an optimal solution in bottom up fashion
4) Construct optimal solution from computed information

Fibonacci Series

0, 1, 1, 2, 3, ----

$F_n = F_{n-1} + F_{n-2}$

$F_0 = 0$, $F_1 = 1$



Repeative values

2012