[Ravi Kumar] ALGORITHMS [12-15] marks 22/08/17

60-65 hours

Syllubus :-

1) Asymptotic Analysis . Recursive / Non Recursive Algo
2) Asymptotic Notations.
3) Recurrence Relations [Methods to solve Recurrence Rel]
4) Algorithm design Techniques.
   1) Divide and Conquer.
   2) Greedy Method
   3) Dynamic Programming.
5) Trees & Graph Representation & Traversal Priority Queues
6) Sorting algo.

- Notes
- WB / GATE } Must
- Text book Ex problem

atleast one.

1) Introduction to Algo
   Coreman.
2) Fundamentals of Algo
   Sawtaj Sahani
3) DS & Algo Using "C"
   Mark Allen Weiss

Email = ravi.82peddapu@gmail.com
facebook.com

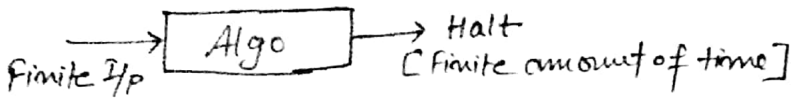
Contact no - 08074172708

Introduction :-

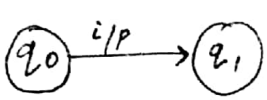
Algorithm :- Step by step representation of computer Program.

Criterias :-

1) **Finiteness :-** Algo must terminate in finite amount of time.

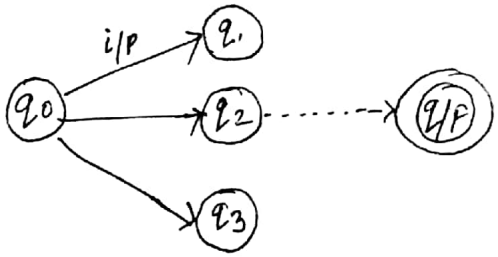


2) **Definiteness :-** Each step of algo must have only unique solution. [Deterministic algo].



[Possible to implement in computer]

Non Deterministic Algo :- Each step of algo consist finite #f solution & algo should **choose correct solution in first attempt.** [Not possible to implement in Computer]



a[1...n] search 'x' present in array.

Deterministic algo:

```
for(i=1; i<=n; i++)
{  if (x==a[i])
      return(i)
}
```

} n comparisons

Non Deterministic algo:

```
i = choose (1...n)
if (x==a[i])
   return (i)
else return (-1)
```

} 1 comparison.

(3) effectiveness :- Each step of algo should be very basic.

Steps to design Algo

1) Device algo :- Design algo for given problem using best design technique.
   - ✓Recursive algo
   - ✓Divide & Conquer
   - ✓Greedy method
   - ✓Dynamic Programming
   - Branch & Bond
   - Back tracking (etc).

2) Validation of algo :-
   Test algorithm correctness.

3) Analysis of algo :- Estimation of CPU execution time / Main memory space required to complete execution of algo.

4) Testing of program :

Decidable Problem :-

Problem which can be solved in Polynomial Time using Deterministic algo. [efficient algo]

n I/p size:

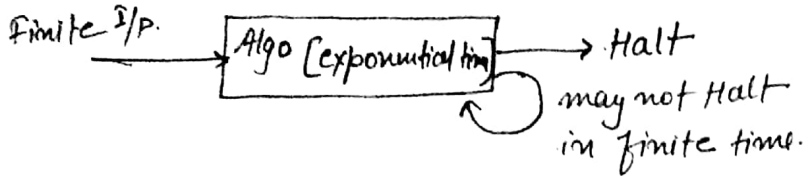
#f computations required : n^2, $n \log n$, n^3, n^k } [Polynomial Time]

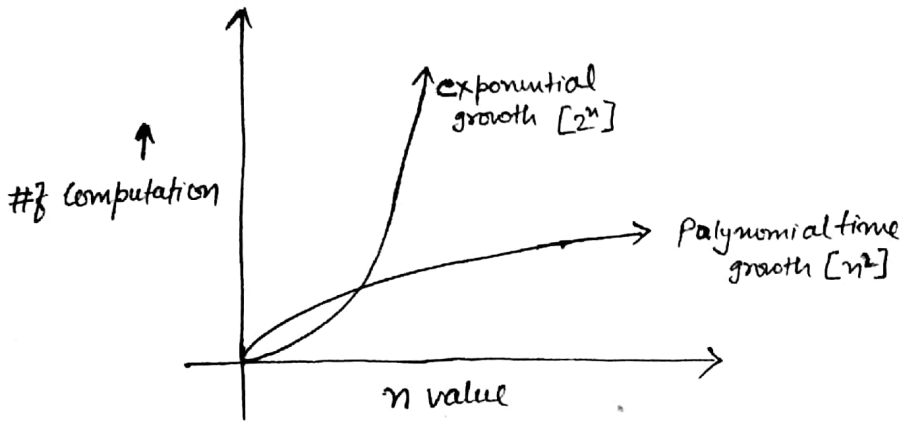
Finite I/p → Algo (Decidable problem) → Halt [Finite Time]

Undecidable problem :- Problems which takes exponential time to solve using Deterministic Algo.

n I/p size :

# of computations required — 2^n, 3^n, $n!$, n^n } exponential time.





n value.	[Tractable] n^2 [Polynomial]	[Untractable] 2^n [exponential]
10	100	1024
11	121	2048
12	144	4096
⋮		
49	$(49)^2$ Halt	2^{49} Halt
50	$(50)^2$ Halt	2^{50} may not Halt.

Algorithm analysis :-

a] Time complexity : Estimation of CPU time [CPU computations] required to terminate algo.

TC of algo = [Sum of frequency count of each instruction of algo]

```
Algo Sum(a,n)                          frequency count
{
    Sum = 0 ---------- 1
    for(i=1; i<n; i++) ---- 2(n+1)
    {
        Sum = Sum + a[i]; --- n
    }
    return (Sum) ---- 1
}
```

i=1	i≤n	i++
	1≤n	2
	2≤n	3
	⋮	⋮
	n≤n	n
	n+1≤n	n+1
1 Initialization	n+1 Comp	n Increm

2(n+1)

$T(n) = 3n+4$.

$T(n) = \theta(n)$

---

b] Space complexity :- Estimation of main memory space required to execute algo.

Sum(a,n) ⇒

i	1
n	1
Sum	1
a[]	n

Space compl : $S(n) = n+3$ memory units.

$S(n) = \theta(n)$

1] TC/SC of loops :-

i) for(i=1; i≤n; i++) ------ 2(n+1)

printf("Made Easy") --- n

TC : $T(n) = 3n+2 = \theta(n)$

SC : Constant = $\theta(1)$

ii) for (i=n; i≥1; i--)

printf ("Made Easy"); —— n times

$\boxed{TC = \theta(n)}$

iii) for (i=1; i≤n; i=i+5)

printf ("Made Easy"); —— $\left[\frac{n-1}{5}\right]+1$ times.

$i = \underbrace{1, 1+5, 1+2*5, 1+3*5, \ldots 1+K*5 \le n}_{K+1 \text{ prints}}$

for K value $1+K\cdot 5 \le n$

$K\cdot 5 \le n-1$

$K = \left[\frac{n-1}{5}\right]+1$ (K+1 prints)

$\boxed{TC = \theta(n)}$

(iv). for (i=n; i≥1; i=i-5)

printf ("Made easy"); ----- $\left[\frac{n-1}{5}\right]+1$ times.

$\boxed{TC = \theta(n)}$

(v). for (i=1; i≤n; i=i*2)

printf ("Made Easy"); ----- $[\log_2 n]+1$ times.

$i \Rightarrow \underbrace{1, 2, 2^2, 2^3, \ldots 2^K \le n}_{K+1 \text{ times printing}}$

$2^K \le n$

$\log_2(2^K) \le \log_2 n$

$K\cdot \log_2 2 \le \log_2 n$

$K = [\log_2 n]$

$\boxed{TC = \theta(\log_2 n)}$

(vi) for (i=n; i≥1; i=i/2)
printf ("Made Easy");

$i = n, \frac{n}{2}, \frac{n}{2^2} \cdots \frac{n}{2^K} \geq 1$

K+1 times

$\frac{n}{2^K} \geq 1$

$n \geq 2^K$

$K = \lfloor \log_2 n \rfloor$

$TC = \theta(\lfloor \log_2 n \rfloor)$

(vii) for (i=1; i≤n; i=i*5)
printf ("Made Easy"); ---- $\lfloor \log_5 n \rfloor + 1$ times.

$TC = \theta(\log_5 n)$

(viii) for (i=2; i≤n; i=i^2)
printf ("Made Easy");

$i = 2, 2^{(2^1)}, 2^{(2^2)}, 2^{(2^3)}, 2^{(2^4)} \cdots 2^{(2^K)} \leq n$

K+1 times printing.

$2^{2^K} \leq n$

$\log(2^{2^K}) \leq \log_2 n$

$2^K \log_2 2 \leq \log_2 n$

$2^K \leq \log_2 n$

$K \log_2 2 = \log_2(\log_2 n)$

$K = \lfloor \log_2(\log_2 n) \rfloor$

$TC = \theta(\log_2 \log_2 n)$

(ix). for (i=n; i≥2; i=√i)
printf ("Made Easy");

$i = n, n^{1/2}, n^{1/2^2}, \ldots, n^{1/2^K} \geq 2$

$n^{1/2^k} \geq 2$

$\frac{1}{2^k} \log_2 n \geq 1$

$\log_2 n \geq 2^k \Rightarrow k \lfloor \log_2 \log_2 n \rfloor$

$TC = \theta(\log_2 \log_2 n)$

Generalization

I
```
for(i=1; i<=n; i=i+c)
   (or)
for(i=n; i>=1; i=i-c)
```
$TC = \theta(n)$

II
```
for(i=1; i<=n, i=i*c)
   (or)
for(i=n; i>=1; i=i/c)
```
$TC: \theta(\log_c n)$

III.
```
for(i=c; i<=n; i=i^c)
   (or)
for(i=n; i>=c; i=√i)
```
$TC: \theta(\log_c \log_c n)$

(X)
```
for(i=2; i<=2^n; i=i^2)
    printf("Made Easy");
```

$i = 2, 2^2, 2^{2^2}, 2^{2^3} \ldots 2^{2^k} \leq 2^n$

$2^k \leq n$

$k \log_2 2 \leq \log_2 n$

$k = \log_2 n$

$TC = \theta(\log_2 n)$

(X).
```
for(i=n/2; i<=n; i=i*2)
    printf("Made Easy");
```

$i = \underbrace{n/2, n}_{\text{2 times.}}, \frac{2n \leq n}{\text{false}}$

$TC = \text{const}$

$TC = \theta(1)$

[xii] for (i=1; $i \le 2^n$; i = i*2)
printf ("Made Easy");

$i = 1, 2, 2^2, \ldots 2^k \le 2^n$

$k = n$.

$TC = \theta(n)$

Nested loops

1] Independent nested loop.

2] Depending Nested Nested loop.

1) Independent nested loop.

Inner loop variable not depends on outer loop variable

Eg.

```
for(i=1, i<=n; i++)                          n times
    for (j=1; j<=n^2; j=j*2)                 * 2log2n
        for (k=n, k>=2; k=√k)                * loglogn
            printf("Made Easy");
```

$TC: n * 2\log n * \log\log n$

$TC = \theta(n \log n \cdot \log\log n)$

Eg = for (i=1; i≤n²; i= i*2) —— $\log n^2 = 2\log_2 n$

for (j=1; j≤n²; j++) —— n^2

for (k=n²; k≥1; k= k/2) —— $\log n^2 = 2\log_2 n$

printf("Made Easy")

$TC = 2\log n * n^2 * 2\log_2 n$

$\boxed{TC = \theta(n^2 \cdot \log^2 n)}$

Eg = main().

{ $n = 2^{2^K}$;

for (i=1; i≤n; i++) — n

{

for (j=2; j≤n; j= j²) — loglogn

}

}

$TC = n * \log\log n$

$n = 2^{2K}$

$K = \log\log n$

$\boxed{TC = \theta(n * K)}$

Eg. for (i=n, j=1; i≥1; i= i/2, j= j+i)

what is value of j after termination of loop.

$i = n, n/2, n/2^2, n/2^3 \ldots n/2^K = 1$

$J = 1 + n/2 + n/2^2 + n/2^3 \ldots + n/2^K$

$= 1 + n\left[\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots + \frac{1}{2^K}\right]$

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

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

Val(j) = θ(n).

+ termination cond^n

$n/2^K = 1$.

$\therefore n = 2^K$

Eg:- ● `i=1; s=1;` time complexity of loop?

```
while (s<=n)
{   i++;
    s = s+i
}
```

	1	2	3		K
i =>	1	[1+1]	[1+2]		[1+2+ ... K]
s =>	1	[1+2]	[1+2+3]		

$[1+2+\dots+K] \le n$

$\frac{K(K+1)}{2} \le n \quad K^2 \le n$

$K = \sqrt{n}$

2] Depending loops

Inner loop variable depending on variable of outer loop.

```
x = 0
for (i=1; i<=n; i++)
{   for (j=1; j<=i; j++)
        for (k=1; k<=j; k++)
            x = x+1
}
```

What is frequency count of loop?

If n = 10 what is final val of x?

Expansion of loops:

i =>	1	2		3			...	n	
j =>	1	1, 2		1, 2, 3				1, 2, 3 ... n	
k =>	1	1	1 to 2	1 to 2		1 to 3		1 to 2	1 to 3 ... 1 to n
x+1 =>	1	1+(1+2)		1+(1+2+3)				1+(1+2+3+...+n)	

$(1+2+3+\ldots+n)$

$S_n = \sum t_n = \sum \frac{n(n+1)}{2}$ $\qquad t_n = \frac{n(n+1)}{2}$

$= \frac{1}{2}\left[\sum n^2 + \sum n\right]$

$= \frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}\right]$

$= \frac{1}{2} \cdot \frac{n(n+1)}{2}\left[\frac{2n+1}{3} + 1\right] = \frac{n(n+1)(n+2)}{6} = \theta(n^3)$

frequency count of loop $= \frac{n(n+1)(n+2)}{6} = \theta(n^3)$

Value of x if $n = 10$

$= \frac{10(10+1)(10+2)}{6} = \frac{10 \cdot 11 \cdot 12^2}{6} = \underline{220}$.

2017 Q.

```
for(i=1; i<=n; i++)
{  for(j=1; j<=n; j=j+i)
       x = x+1
}
```

What is TC?

expansion:

$i \Rightarrow$	1	2	3	...	n
$j \Rightarrow$	1, 2, ... n	$n/2$ times	$n/3$ times	...	n/n times.
$x = x+1 \Rightarrow$	n	$+ \frac{n}{2}$	$+ \frac{n}{3}$		1 times

$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \ldots\ldots + \frac{1}{n}\right]$

$= \underline{\theta(n \log n)}$

$$H_n = \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \log_e n$$

$$\int_{x=1}^{n} \frac{1}{x}\,dx \approx \log_e n$$

Recursive algorithms :—

23/08/17

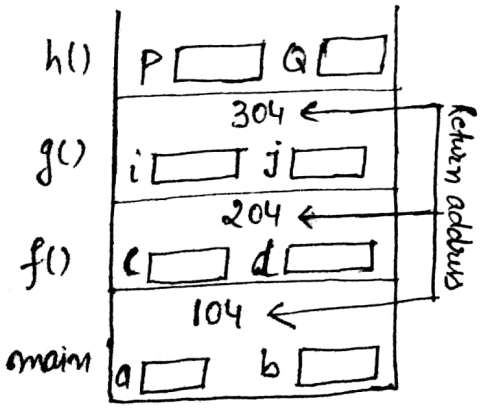
Stack DS required to run Recursive algo & Subroutine calls to store return Address.

```
Algo main()
{   int a,b;
    101 St1
    102 St2
    103 if (cond) f();   (True)
    104 St3
    105 St4
}

f()
{   int c,d;
    201 St11
    202 St12
    203 if (cond) g();   (True)
    204 St13
    205 St14
}

g()
{   int, i,j;
    301 St21
    302 St22
    303 if (cond) h();   (True)
    304 St23
    305 St24
}

h()
{   int p,q;
    401 St31
    402 St32
}
```



Stack

main
f()
g()
h()

4 level fun calls [4 stack entries]

Overhead of fun call :— [Micro Program for fun call]

1] TOP[stack] = PC
2] PUSH [stack]
3] PC ← New Addr
   // f() executes
4] POP[stack]
5] PC ← TOP[stack]

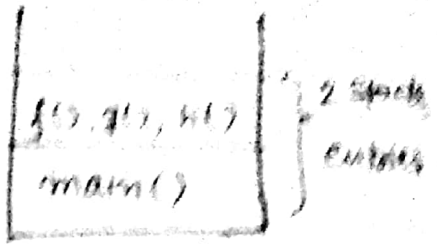
Eg: How much stack space required to run given program.

```
main()
{
  int p, q;
  f();
  g();
  h();
}
```

```
f()
{
}
```

```
g()
{
}
```

```
h()
{
}
```



functions f(), g(), h() are all called by the main() only
So any function f(), g(), h() is called only when any previous function terminates.
So it is a 2 level of function calls which requires 2 stack entries.

Eg

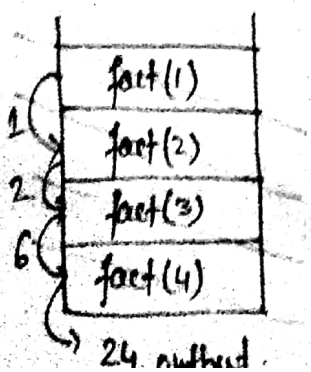
```
Algo fact(n)
{
    if (n ≤ 1)
        return(1);
    else
        return(n * fact(n-1))
}
```

```
fact(4)
{
  4 * fact(3)
}
```

```
fact(3)
{
  3 * fact(2)
}
```

```
fact(2)
{
  2 * fact(1)
}
```

```
fact(1)
{
  return(1);
}
```



fact(n)
↓
fact(n-1)
↓
fact(n-2)
⋮
fact(1)

depth of recursion
n levels
[n stack entries to run]

Space complexity of fact(n): n stack entries.

$\underline{O(n)}$

Q. Algo Rec(n)

```
{
    if(n<=1)
        return (1);
    else
        return (rec(n-1)+rec(n-1)+n);
}
```

```
rec(n) - T(n)
{
    x = rec(n-1)   }
    y = rec(n-2)   } 2T(n-1)
    return (x+y+n)] C
}
```

a] TC of rec(n):- T(n) is TC of rec(n)

$$T(n) = \begin{cases} a = O(1) & n \le 1 \\ 2T(n-1)+C & n > 1 \end{cases}$$

[Substitution method to solve ~~recurrence~~ recurrence relation]

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

$= 2[2T(n-2)+C]+C \quad [\because T(n-1) = 2T(n-2)+C]$

$= 2^2T(n-2)+2C+C$

$= 2^2[2T(n-3)+C]+2C+C \quad [\because T(n-2) = 2T(n-3)+C]$

$= 2^3T(n-3)+C[2^2+2+1]$

$\vdots$ k times

$= 2^kT(n-k)+C\underbrace{[2^{k-1}+\ldots+2+1]}_{\text{Recurrence Relation}}$

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

$\left[S_n = \frac{a[r^n-1]}{r-1}, \text{ if } r>1\right]$

$= 2^kT(n-k)+C[2^k-1]$

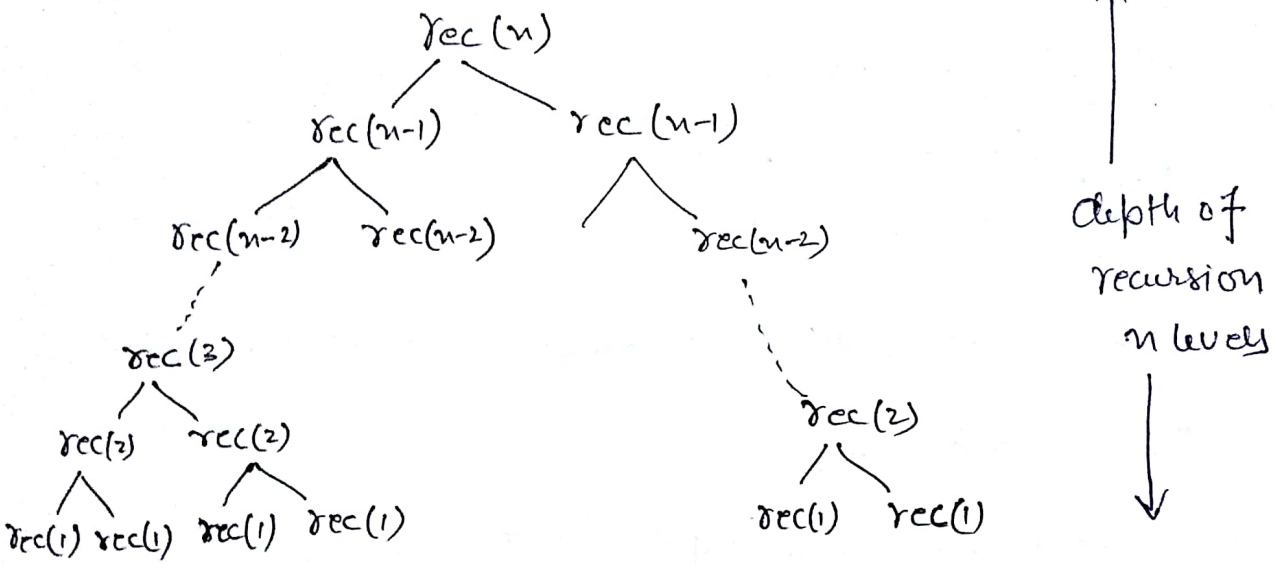
$= 2^{n-1} T(1) + c\left[2^{n-1} - 1\right]$ [Terminating cond^n if $n-k=1$. $\therefore k = n-1$.]

$$T(n) = a \cdot 2^{n-1} + c \cdot 2^{n-1} - c$$

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

---

b] SC of # of fun calls of Rec (n)



So, space complexity of rec(n): $\theta(n)$

$$\text{SC of rec}(n) = \theta(n)$$

#f function call :-

$$= 1 + 2 + 2^2 + 2^3 + \cdots + 2^{n-1}$$

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

* For recursive algorithm if each function call required constant time except time for sub routine call, then

TC of algo = θ(#f function calls)

c] Value return by Algo :-

$$rec(n) = \begin{cases} 1 & , \text{if } n \le 1 \\ 2 \cdot rec(n-1) + n & , \text{if } n > 1 \end{cases}$$

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

$$\equiv \theta(2^n)$$

Q. Algo rec(n)

```
{ if (n<=1)
    return (1);
  else
    return (2 * rec(n-1) + n)
}
```

`rec(n)` $\rightarrow T(n)$

`rec(n-1)` $\rightarrow T(n-1)$

`return (2*x+n)` $\rightarrow c$

a] TC of rec(n):-

$$T_n = \begin{cases} a & , n \le 1 \\ T(n-1) + c & , n > 1 \end{cases}$$

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

$$= T(n-2) + 2C \quad [\because T(n-1) = T(n-2) + C]$$

$$= T(n-3) + 3C \quad [\because T(n-2) = T(n-3) + C]$$

$$\vdots \quad k \text{ times}$$

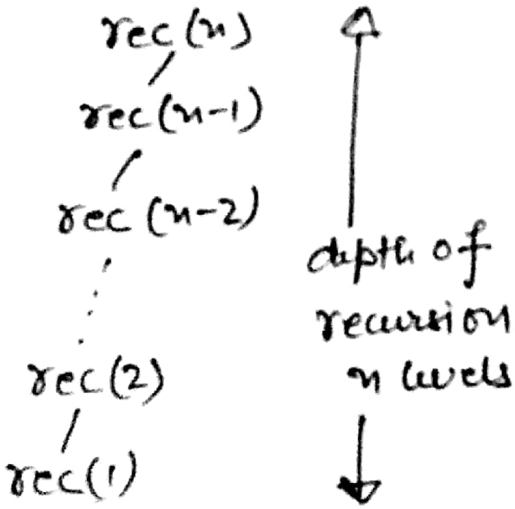
$$= T(n-k) + k \cdot C \quad \left[\begin{array}{l}\text{Terminating condn} \\ \text{if } n-k = 1 \;\therefore\; k = n-1\end{array}\right]$$

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

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

$$\boxed{T(n) = \theta(n)}$$

b] SC & #f func calls



$\boxed{\text{Sc of } rec(n) : \theta(n)}$

#f funcalls : n

c] value return by algo.

$$rec(n) = \begin{cases} 1 & , n \le 1 \\ 2 \cdot rec(n-1) + n & , n > 1 \end{cases}$$

Q3] Algo Rec(n)

```
{
  if (n ≤ 1)
     return(1);
  else
  {  rec(n-1)
     for(i=1; i≤n; i++)
      printf("Made Easy")
  }
}
```

a] TC of rec(n)

b] SC of rec(n)

c] # of fun calls of Rec(n)

a] TC of rec(n)

$$T(n) = \begin{cases} 0 & , n \leq 1 \\ T(n-1)+n & , n > 1 \end{cases}$$

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

$= T(n-2)+n+(n-1)$ $[\because T(n-1) = T(n-2)+(n-1)]$

$= T(n-3)+(n-2)+(n-1)+n$ $[\because T(n-2) = T(n-3)+(n-2)]$

⋮ k times

$= T(n-K)+(n-(K-1))+\ldots+(n-1)+n$

[Terminating condn: $n-K=1$, $\therefore K=n-1$]

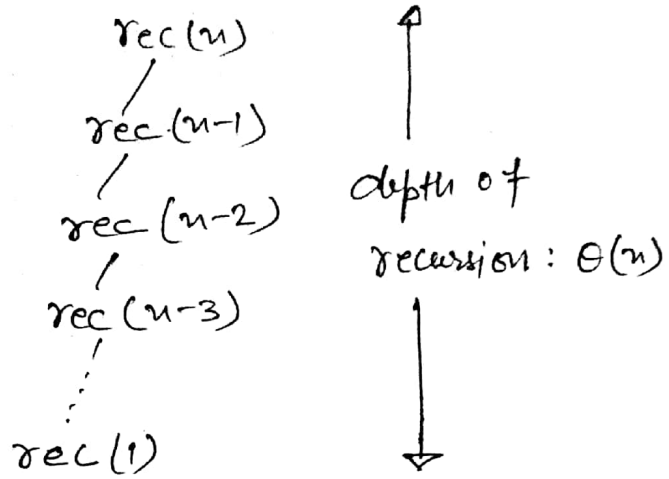
$T(n) = T(1)+2+3+\ldots+(n-1)+n$

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

$T(n) = \frac{n(n+1)}{2} = \boxed{\theta(n^2)} = TC$

b] SC

b] SC & # fun calls :-



SC of rec(n) = Θ(n)

fun calls = Θ(n)

Q3] Algo rec(n)

```
{ if (n≤1)
      return(1)
  else
   { rec(n-1);
     rec(n-1);
      for (i=1; i≤n; i++)
          printf("Made Easy");

   }
}.
```

a] TC of rec(n) :-

$$T(n) = \begin{cases} 1 & , n \le 1 \\ 2T(n-1)+n & , n > 1 \end{cases}$$

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

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

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

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

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

⋮ K times.

$= 2^K T(n-K) + 2^{K-1}(n-(K-1)) + 2^{K-2}(n-(K-2)) + \ldots + 2(n-1) + n$

$n-K$ = 1 terminating cond^n

$n - K = 1, \quad K = n-1.$

$T(n) = 2^{n-1} \cdot 1 + 2^{n-2} \cdot 2 + 2^{n-3} \cdot 3 + \ldots + 2(n-1) + n$ — ①

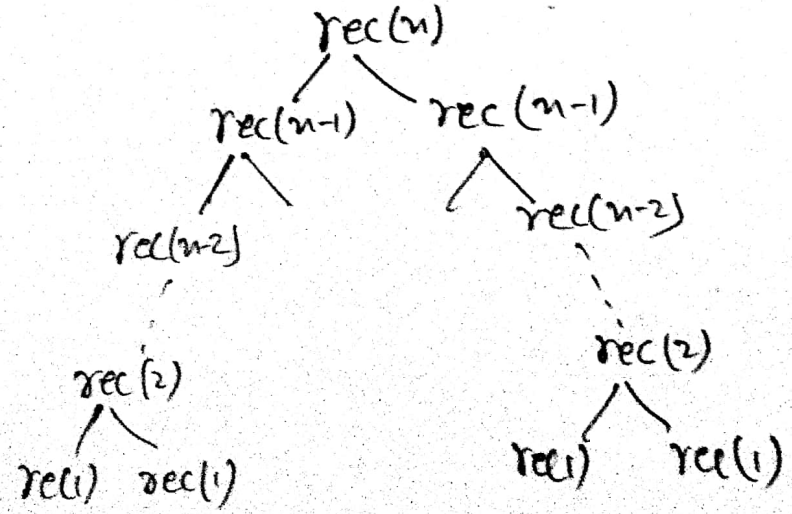
$2T(n) = 2^n \cdot 1 + 2^{n-1} \cdot 2 + 2^{n-2} \cdot 3 + \ldots + 2^2(n-1) + 2 \cdot n.$ ① × 2 — ②

$2T(n) - T(n) \equiv [2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2^1] - n$ — ② − ①

$T(n) \equiv \left[\frac{2(2^n - 1)}{2-1}\right] - n$

$\boxed{T(n) = 2^{n+1} - 2 - n} = \Theta(2^n)$

b] SC & # fun calls :-



SC of rec(n) : $\Theta(n)$

fun calls $1 + 2 + 2^2 + \ldots + 2^{n-1}$

$= 2^n - 1$

Simplification of AP + GP Series :-

1) $T(n) = 5^n \cdot n + 5^{n-1}(n-1) + \dots + 5^2 \cdot 2 + 5^1 \cdot 1$

$5T(n) = 5^{n+1} \cdot n + 5^n (n-1) + \dots + 5^3 \cdot 2 + 5^2 \cdot 1$

$-$

$5T(n) - T(n) \equiv 5^{n+1} \cdot n + 5^n(-1) + 5^{n-1}(-1) + \dots + 5^3(-1) + 5^2(-1) - 5$

$4T(n) \equiv 5^{n+1} \cdot n - [5^n + 5^{n-1} + \dots + 5^3 + 5^2 + 5]$

$\frac{5[5^n - 1]}{5-1}$

$4T(n) \equiv 5^{n+1} \cdot n - \frac{5}{4}(5^n - 1)$

$T(n) \equiv \frac{1}{4}\left\{5^{n+1} \cdot n - \frac{5}{4}(5^n - 1)\right\} \equiv \theta(n \cdot 5^n)$

---

2) $T(n) = \frac{n}{4^n} + \frac{n-1}{4^{n-1}} + \frac{n-2}{4^{n-2}} + \dots + \frac{2}{4^2} + \frac{1}{4}$

$\frac{T(n)}{4} = \frac{n}{4^{n+1}} + \frac{n-1}{4^n} + \frac{n-2}{4^{n-1}} + \dots + \frac{2}{4^3} + \frac{1}{4^2}$

$-$

Q3] $T(n) = \frac{1}{3^n} + \frac{2}{3^{n-1}} + \frac{3}{3^{n-2}} + \cdots + \frac{n-1}{3^2} + \frac{n}{3}$

TOWERS of HONAI :-

Problem Statement :-

Given 3 towers [X, Y, Z] and n plates in tower X with descending order of diameters from bottom to top.

Compute #f plate movements required to transfer n plates from X to Y:

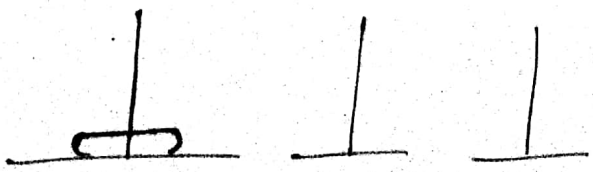
With constraints :-

1) Only top plate allowed to move from tower

2) At any time lower diameter plate should not be below to upper diameter.

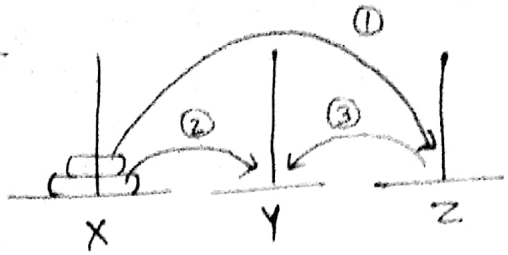
---

for n = 1



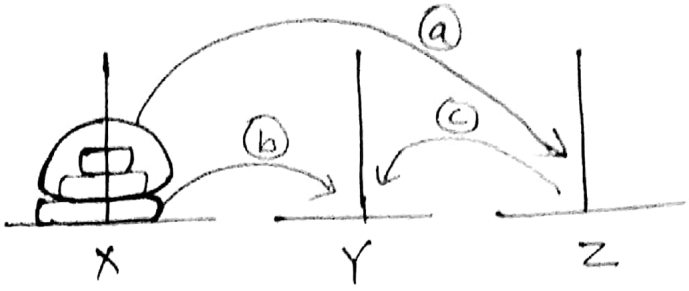
#f Moves : 1 (X → Y)

for n=2



#f Moves : 3 [X→Z, X→Y, Z→Y]

for n=3.



ⓐ : X→Y, X→Z, Y→Z
ⓑ : X→Y
ⓒ : Z→X, Z→Y, X→Y

} 7 moves

Procedure :-

To transfer n plates from X to Y using Z as temporary.

a] Trans Top(n-1) plates from X to Z using Y as temp.

b] move one plate from X to Y

c] Trans (n-1) plates from Z to Y using X as temp.

Algo TOH (X, Y, Z, n)

```
{   // TOH (Source, distination, temp, #f plates)
    if (n==1)
        return (X→Y)
    else
    {   TOH (X, Z, Y, n-1);
        move X→Y
        TOH (Z, Y, X, n-1);
```

$T(n)$: #f moves to trans n plates from X to Y

$$T(n) = \begin{cases} 1 & , n=1 \\ 2T(n-1)+1 & , n>1. \end{cases}$$

$= 2^n - 1$ (moves) Undecidable problem

SC of TOH (X, Y, Z, n) : $\Theta(n)$

24/08/17

Q] Algo Rec(n)

```
{ if (n<=1)
    return(1)
  else
    return (rec(n/2) + rec(n/2) + n)
}
```

a] TC of rec(n) :-

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

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

$= 2[2T(n/2^2)+c]+c \quad [\because T(n/2) = 2T(n/2^2)+c]$

$= 2^2 T[n/2^2) + 2c + c$

$= 2^2[2T(n/2^3)+c] + 2c + c$

$= 2^3 T(n/2^3) + c[2^2+2+1]$

: k times.

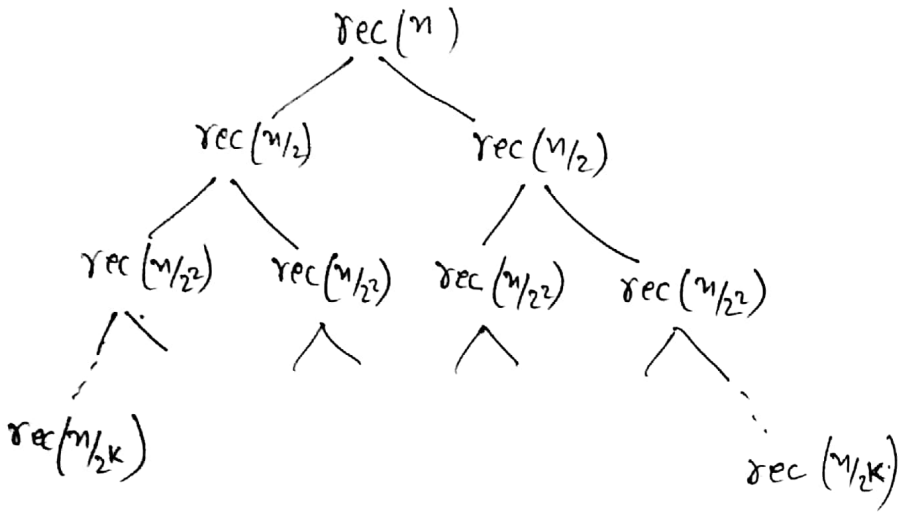
$2^k T(n/2^k) + c\overbrace{[2^{k-1} + \ldots 2^2+2+1]}^{2^k-1}$

$T(n) = n \cdot T(1) + c[n-1]$ $\left[\because \frac{n}{2^k} = 1 \Rightarrow n = 2^k, \; k = \log_2 n\right]$

$\boxed{T(n) = a \cdot n + c[n-1]}$

$\boxed{T(n) = \theta(n)}$

b] SC ff #f fun calls of rec(n)



depth of recursion $k+1$ levels.

$\lfloor \log_2 n \rfloor + 1$ levels

SC of Algo rec(n) : $\boxed{\theta(\log_2 n)}$

# of fun call : $1 + 2 + 2^2 + \cdots + 2^k = 2^{k+1} - 1$

$= 2n - 1 = \boxed{\theta(n)}$

c] Val return by rec(n)

$$rec(n) = \begin{cases} 1 & , n \le 1 \\ 2 \cdot rec(n/2) + n & , n > 1 \end{cases}$$

$rec(n) = 2rec(n/2) + n$

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

$= 2^2 rec(n/2^2) + 2n$

$= 2^2\left[2rec(n/2^3) + \frac{n}{2^2}\right] + 2n$

$= 2^3 rec(n/2^3) + 3n$

⋮ k times

$= 2^k rec(n/2^k) + kn$ $\left[\because \frac{n}{2^k} = 1 \therefore n = 2^k \Rightarrow k = \log_2 n\right]$

$rec(n) = n + n.\log_2 n$

$= \underline{\theta(n.\log n)}$

Q. Algo rec(n)

```
{  if (n ≤ 2)
       return;
   else
       return (rec(√n) + rec(√n) + n)
}
```

Sol. a) TC of rec(n) :-

$$T(n) = \begin{cases} a & , n \le 2 \\ 2T(\sqrt{n}) + c, & n > 2 \end{cases}$$

$T(n) = 2T(n^{1/2}) + c$

$= 2[2T(n^{1/2^2}) + c] + c$

$= 2^2 T(n^{1/2^2}) + 2c + c$

$= 2^3 T(n^{1/2^3}) + c[2^2 + 2 + 1]$

⋮ k times.

$2^k T(n^{1/2^k}) + c[\overbrace{2^{k-1} + \dots + 2 + 1}^{2^k - 1}]$

$n^{1/2^k} = 2$

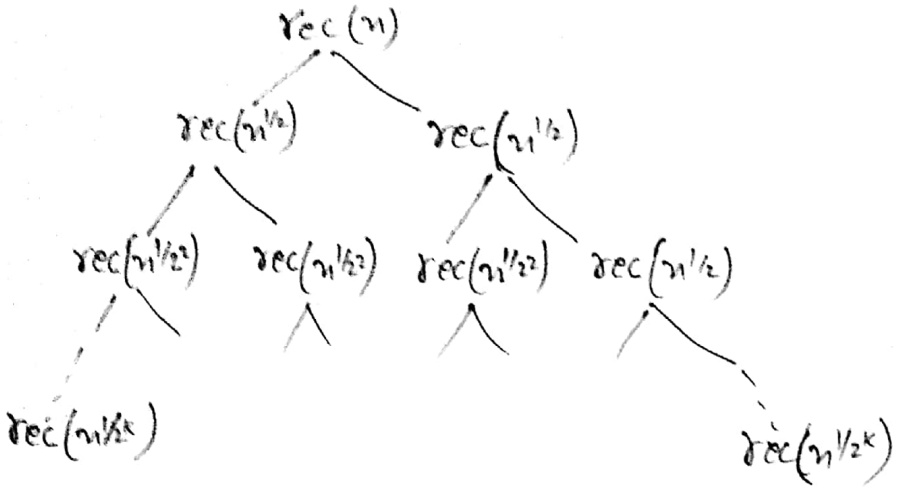
$n^{1/2^k}$ → 2

$\Rightarrow (2^k = \log n)$

$T(n) = \log n \cdot a + c[\log n - 1]$

$\boxed{T(n) = \theta(\log n)}$

6] SC + #f fun calls



$n^{1/2^k} = 2$

$K = \log_2 \log_2 n$

SC of algo rec(n): $\theta(\log_2 \log_2 n)$

#f fun calls of rec(n): $\theta(\log n)$

Generalize

rec function	SC of rec(n) [stack space]
$rec(n) = rec(n-c) + n$	$\approx \frac{n}{c} = \theta(n)$
$rec(n) = rec(n/c) + n$	$\approx \log_c n = \theta(\log_2 n)$
$rec(n) = rec(\sqrt{n}) + n$	$\approx \log_k \log_c n = \theta(\log \log n)$

exponential time complexity rec Algo :- [Sample]

```
rec(n)
{ rec(n-1)
  rec(n-1)
}
```

```
rec(n)
 { rec(n-1)
   rec(n-2)
 }
```

```
rec(n)
 { rec(n-1)
   rec(n-1)
   for(i=1; i<=n; i++)
       x=x+1
 }
```

```
rec(n)
{
   rec(n-1)
   rec(n-1)
   rec(n-1)
}
```

```
rec(n)
{ rec(n-c1)
  rec(n-c2)
}.
```

⇒ Fast exponential algo :-

Compute : x^n

To compute x^n ⇒

a] $y = x^{n/2}$

b] if n is even
return(y·y)
else
return (x·y·y)

eg $x^{64} = x^{32} * x^{32}$

$x^{16} * x^{16}$

$x^8 * x^8$

$x^4 * x^4$

$x^2 * x^2$

$x * x$

6 Multiplication required
$\approx \log_2 64$

eg $x^{100} = x^{50} * x^{50}$

$x^{25} * x^{25}$

$x * x^{12} * x^{12}$

$x^6 * x^6$

$x^3 * x^3$

$x * x^2 * x^2$

8 Multiplication required
$\approx \log_2 100$

```
Algo  Fast Exp (x,n)
   {   // Compute x^n
       if (n==0) return(1);
       else if (n==1) return (x);
          else
          {  y = Fast Exp (x, n/2);
             if (n%.2==0)
                 return(y * y);
             else
                 return (x * y * y);
          }
   }
```

⇒ x^5 ∈ Fast Exp (x, 10)

$x^5 = x \cdot x^2$ Fast Exp (x, 5)

x Fast Exp (x, 2)

x Fast Exp (x, 1)

TC of Fast Exp Algo :-

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

$$= \Theta(\log_2 n)$$

SC of Fast Exp Algo [Recursive]

$$= \Theta(\log_2 n)$$

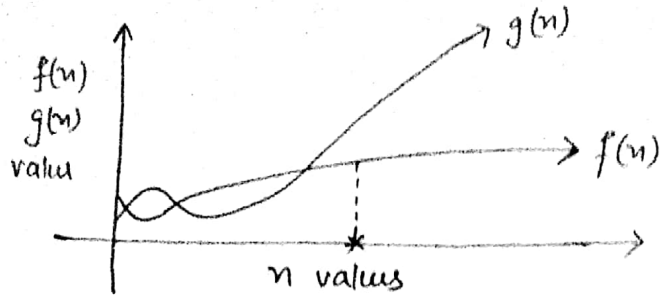
Asymptotic Notations :-

⇒ Asymptotic Comparison of non negative fun :-

Growth rate Comparison for non negative functions for large n values.

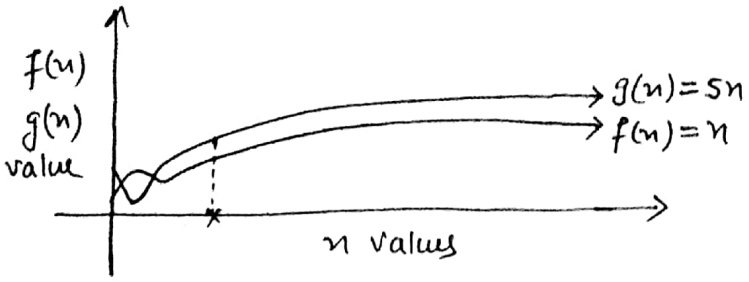
f(n) g(n) Asymptotic Comp :-

Growth rate comp of f(n) & g(n) for large n values.



$g(n)$ Asymptotically bigger than $f(n)$

[$g(n)$ growth rate more than growth rate of $f(n)$ for large n values]



$f(n)$ & $g(n)$ are Asymptotically equal

[Asymptotically equal means their values are in common ratio, but not the same].

$\Rightarrow$ $f(n)$ & $g(n)$ Asymptotically equal

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

Eg: $f(n) = 10n^2 + 20n$

$g(n) = 2n^2 + 100n + 100$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2[10 + 20/n]}{n^2[2 + 100/n + 100/n^2]} = 5 \text{ [const]}$$

$\therefore$ $f(n)$ & $g(n)$ Asymptotically equal.

⇒ $f(n)$ Asymptotically bigger than $g(n)$

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

$f(n) = 5n^2 + 10$

$g(n) = 10n + 20$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n[5n + 10/n]}{n[10 + 20/n]} = \frac{\infty}{10} = \infty$$

∴ [$f(n)$ Asymptotically bigger than $g(n)$]

⇒ $f(n)$ Asymptotically smaller than $g(n)$

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

$f(n) = 2n^2$

$g(n) = n^3 + 10$

$$\lim_{n\to\infty} \frac{2n^2}{n^2\left(n + \frac{10}{n^2}\right)} = \frac{2}{\infty} = 0$$

∴ [$f(n)$ Asym smaller than $g(n)$]

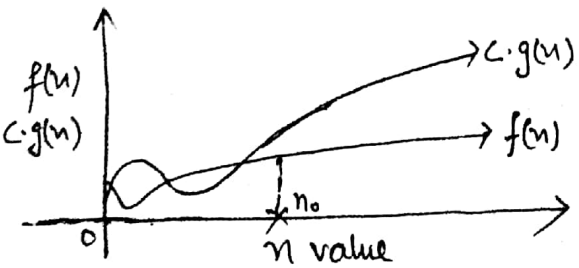
Big oh Notation :-

$f(n)$ $g(n)$ non negative fun

$f(n) = O(g(n))$ iff

$f(n) \le c \cdot g(n)$ for all n values.

where $n \ge n_0$.



⇒ $f(n) = O(g(n))$ iff

$g(n)$ Asymptotically bigger or equal to $f(n)$.

Eg:- $2n+3 = O(n)$

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

$[2n+3] \leq c[n]$ for all value $n \geq n_0$

$\underbrace{[2n+3]}_{f(n)} \leq 3\underbrace{[n]}_{c \cdot g(n)} \quad \forall n \; n \geq 3 \Rightarrow \underline{2n+3 = O(n)}$ ✓

Eg:- $2n+3 = O(n^2)$

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

$2n+3 \leq c \cdot n^2 \quad \forall n \quad n \geq n_0.$

$\underline{2n+3 = O(n^2)}$ ✓

Eg:- $2n+3 = O(\log_{10} n)$

	10	100	10^3	10^4	10^5
$f(n) = 2n+3$	23	203	2003	20003	200003 ⟶
$c.g(n) = 10000 * \log_{10} n$	10,000	20,000	30,000	40,000	50,000

Here $f(n)$ is growing faster.

So $2n+3 \leq c \cdot \log_{10} n$ [not true for all value]

$\underline{2n+3 \neq O(\log_{10} n)}$ ✗

Asymptotic Comparisions

⇒ Decrement function :-

$f_1(n) = \frac{c}{n}, \frac{1}{n^2}, \frac{n^2}{2^n}$

$\left[\frac{n^2}{2^n} < \frac{1}{n^2} < \frac{c}{n}\right]$

⇒ Constant function

$f_2(n)$ = Constant

⇒ Log functions :-

$f_3(n) = \log n,\ (\log n)^k,\ (\log\log n),\ (\log\log n)^k$

$[\log\log n < (\log\log n)^k < \log n < (\log n)^k]$

⇒ Polynomial function $[n^k]$

$f_4(n) = n^{0.1}, n^2, n^{0.5}, n\log n, n^3$

$(n^{0.1} < n^{0.5} < n\log n < n^2 < n^3)$

⇒ Exponential function :-

$f_5(n) = 2^n, 3^n, n!, n^n, (1.01)^n$

$[(1.01)^n < 2^n < 3^n < n! < n^n]$

Comparision

$[\underset{\text{fun}}{\text{dec}} < \underset{\text{fun}}{\text{const}} < \underset{\text{fun}}{\log} < \underset{\text{fun}}{\text{poly}} < \underset{\text{fun}}{\exp}]$

Problems :-

① ⇒ Which is false ?

a) $100\log n = \frac{\log n}{100} = O(\log n)$

b) if $x < y$ then $n^x = O(n^y)$

c) $n^a = O(a^n)$ $(a > 1$ const$)$

d) $\sqrt{\log n} = O(\log_2 \log_2 n)$

② $f(n) = \log_y n^x$ (x, y const)

$g(n) = \log_2 n$

which is true?

a) $f(n) = O(g(n))$

b) $g(n) = O(f(n))$

c) a & b

d) None.

$f(n) = n \cdot \log_y n = \frac{n}{\log_2 y} \cdot \log_2 n$

$g(n) = \log_2 n$

[$f(n)$ & $g(n)$ Asy equal]

④ $f(n) = \begin{cases} 2^n & \text{for even } n \\ n & \text{otherwise} \end{cases}$

$g(n) = \begin{cases} 2^n & \text{for odd } n \\ n & \text{otherwise} \end{cases}$

which is true?

a) $f(n) = O(g(n))$ b) $g(n) = O(f(n))$

c) a & b ~~d)~~ None.

$f(n)$ $g(n)$ Asy non comparable

$f(n) \neq O(g(n))$

$g(n) \neq O(f(n))$

⑥ $f(n) = \sqrt{n}$ $g(n) = n^{1+\sin n}$.

which is true?

a) $f(n) = O(g(n))$

b) $g(n) = O(f(n))$

c) a & b

d) None.

⇒

n	$f(n) = n^{0.5}$		$g(n) = n^{1+\sin n}$
90	$(90)^{0.5}$	<	$(90)^2$
180	$(180)^{0.5}$	<	$(180)^1$
270	$(270)^{0.5}$	>	$(270)^0 = 1$.
360	$(360)^{0.5}$	<	$(360)^1$

③ $f(n) = \begin{cases} n^3 & 0 \le n \le 10,000 \\ n & n > 10000 \end{cases}$ $f(n) = n$

$g(n) = \begin{cases} n & 0 \le n \le 100 \\ n^2 & n > 100 \end{cases}$ $g(n) = n^2$ for large n

which is true

a) $f(n) = O(g(n))$ b) $g(n) = O(f(n))$

c) a & b d) None.

⑤ $f(n) = n^{1+\sin n}$

$g(n) = n^2$

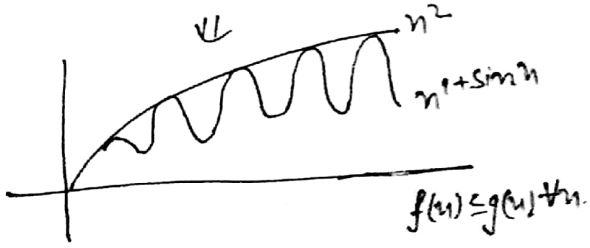
which is true?

a) $f(n) = O(g(n))$

b) $g(n) = O(f(n))$

c) a & b

d) None.



⇒ $f(n) = n!$ $g(n) = n^n$

$f(n) = n(n-1)(n-2) \cdots 2 \times 1$

$g(n) = n \cdot n \cdot n \cdots n \cdot n$

[n^n Asymptotically bigger than $n!$]

$n! = O(n^n)$

$n^n \neq O(n!)$

⇒ $f(n) = n!$ $g(n) = (n-1)!$ $h(n) = (n-1)! + n$.

Write Asymptotic increasing order.

$(n-1)! + n = (n-1)! < n! \Rightarrow$ $g(n) = h(n) < f(n)$

⇒ $f(n) = 2^n$ $g(n) = n^{\sqrt{n}}$

Soln $\log(f(n))$ $\log g(n)$

$= \log 2^n$ $= \sqrt{n} \log n$

$= n \log 2$

$= \sqrt{n} \cdot \sqrt{n}$

[if $\log f(n)$ Asymptotically bigger than $\log g(n)$ then $f(n)$ Asymptotically bigger than $g(n)$]

$n^{\sqrt{n}} = O(2^n)$

$2^n \neq O(n^{\sqrt{n}})$

⇒ If $\log f(n)$ Asy bigger than $\log g(n)$ then $f(n)$ Asym. bigger than $g(n)$.

⇒ If $f(n)$ Asy bigger than $g(n)$ then

[$\log f(n)$ Asy bigger than $\log g(n)$] (or) —

[$\log f(n)$ Asy equal to $g(n)$]

$f(n) > g(n)$

$2^n > n^2 \Rightarrow \log(2^n) > \log(n^2)$

$n^3 > n^2 \Rightarrow \log(n^3) = \log(n^2)$

$\Rightarrow$ if $\log f(n)$ Asy equal to $\log g(n)$

then $f(n) > g(n)$ (Asy)

or $f(n) < g(n)$ (Asy)

$f(n) = g(n)$

$\Rightarrow$ if $f(n) + g(n)$ Asy equal

then $\log f(n)$ $\log g(n)$ also Asy equal

***

$f(n) = \log_2(n!)$

$g(n) = \log_2(n^n) = n\log_2 n$

Sterling formula:-

$$n! \underset{\text{Asy}}{\approx} \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$\Downarrow$ equal

$$\log(n!) \underset{\text{Asy equal}}{\approx} \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$$

$$\underset{\text{Asy equal}}{\approx} \left[\frac{1}{2}\log 2\pi e + \frac{1}{2}\log n + n\log n + - n\log e\right]$$

$(\log n!) \underset{\text{Asy equal}}{\approx} (n\log n)$ / $\log n! = O(n\log n$, $n\log n = O(\log n!)$

25/08/17

Testing of $f(n)$

$f(n)$ is polynomially bounded or Exponentially bounded :-

⇒ if $\log f(n) = O(\log n)$

$\log f(n) \leq \log n$ [Asymptotically]

then $f(n)$ polynomially bounded.

⇒ if $\log f(n) \neq O(\log n)$

$\log f(n) > \log n$ [Asymptotically]

then $f(n)$ exponentially bounded.

⇒ $f(n) = n^k$ [k const]

// poly fun.

$\log f(n) = k \cdot \log n = O(\log n)$

⇒ $f(n) = 2^n$ [Exponential fun]

$\log f(n) = n \cdot \log 2 \neq O(\log n)$

⇒ Test whether the given fun is polynomially or Exponentially bounded ?

1] $f(n) = n!$

$\log f(n) = \log n!$

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

$f(n) = n!$ // Exponential fun.

2] $f(n) = (\log n)!$

$\log f(n) = \log (\log n)!$

$= \log n \cdot \log\log n \neq O(\log n)$ $f(n) = (\log n)!$ exponential fun.

3] $f(n) = (\log\log n)!$

$\log f(n) = \log(\log\log n)!$

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

$f(n) = (\log\log n)!$ Polynomially bounded.

4] $f(n) = \log(n!) = n\log n$

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

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

$(f(n) = \log n!$ poly. bounded$)$

5] $f(n) = \log((\log n)!) = [\log n \cdot \log\log n]$

$\log f(n) = \log(\log n \cdot \log\log n)$

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

$f(n) = \log((\log n)!$ polynomially bounded.

*** $n!$ } Exp fun

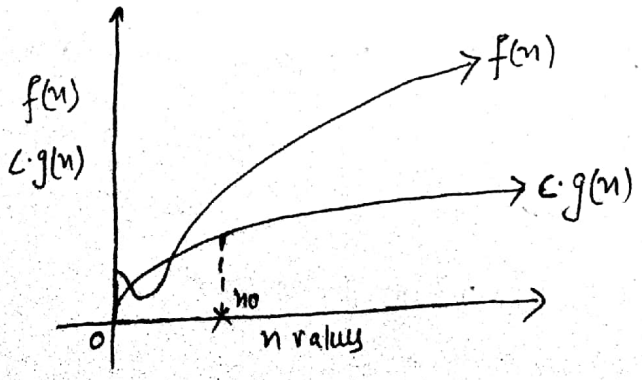
$(\log n)!$ }

$(\log\log n)!$ } Poly fun

Omega Notation :- (Ω)

⇒ $f(n)$ & $g(n)$ non negative fun

$f(n) = \Omega(g(n))$ iff $f(n) \geq c \cdot g(n)$ for all n val where $(n \geq n_0)$.



$f(n) \geq c \cdot g(n)$ $\forall n$ $n \geq n_0$

$f(n) = \Omega(g(n))$

⇒ $f(n) = \Omega(g(n))$ iff $g(n)$ Asymptotically Smaller or equal to $f(n)$.

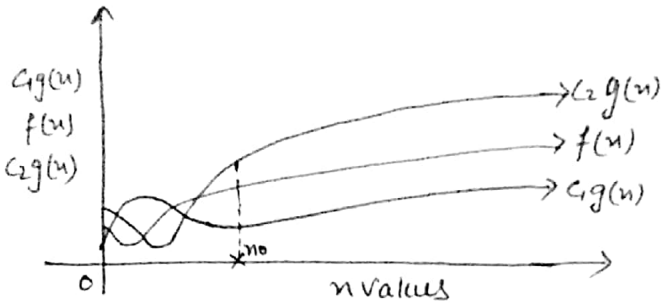
⇒ $f(n) = \Omega(g(n))$ iff $g(n) = O(f(n))$.

Theta Notation :- (θ)

⇒ $f(n)$ $g(n)$ non negative function

$f(n) = \theta(g(n))$ iff $c_1 g(n) \le f(n) \le c_2 g(n)$ $\forall n$ where $[n \ge n_0]$

[c_1, c_2, n_0 : constants]



$f(n) = \theta(g(n))$

⇒ $f(n) = \theta(g(n))$ iff $g(n)$ Asymptotically equal to $f(n)$

⇒ $f(n) = \theta(g(n))$ iff $[f(n) = O(g(n)) \text{ && } f(n) = \Omega(g(n))]$

Little o'h Notation :- (o)

⇒ $f(n)$ & $g(n)$ non negative fun⇒ $f(n) = o(g(n))$ iff

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

⇒ $f(n) = o(g(n))$ iff $g(n)$ Asymptotically bigger than $f(n)$.

Q Test given stmt True/false

1) $n! = o(n^n)$
$n! < n^n$
(True)

2) $2^n = o(n!)$
$2^n < n!$
(True)

3) $\log n! = o(n \log n)$
$\log n! = n \log n$
(False.)

4) $\log n! = o((\log n)!)$
$\log n! < (\log n)!$
(True)

5) $n^2 \log n = o(n^{2.01})$
$n^2 \log n < n^{2.01}$
(True)

Little omega :- (ω)

$f(n)$ $g(n)$ non negative fun

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

$\Rightarrow$ $f(n) = \omega(g(n))$ iff $g(n)$ Asymptotically smaller than $f(n)$.

Q. Find given stmt true or false?

1] $2^{n+1} = \theta(2^n)$ $\Rightarrow$ True

2] $2^{2n+1} = \theta(2^n)$ $\Rightarrow$ false

3] $x^{n+1} = \theta(x^n)$ (x: const) $\Rightarrow$ True

4] $x^{2n+1} = \theta(x^n)$ (x: const) $\Rightarrow$ False

5] $n! = \theta(n^n)$ $\Rightarrow$ False

6] $\log n! = \theta(n \log n)$ $\Rightarrow$ True.

Q. Write given fun in asy ascending (Increasing) order.

$\log^k \log\log n$, $\log\log^k \log n$, $\log\log\log^k n$, $\log\log\log n^k$ [k: const]

$\log^k \log\log n = \underline{(\log\log\log n)^k}$, $\log\log^k \log n = \log(\log\log n)^k$

$= \underline{k \cdot \log\log\log n}$

$\log\log\log^k n = \log \cdot \log(\log n)^k$

$= \log[k \cdot \log\log n]$

$= \log k + \log\log\log n$

$\approx \underline{\log\log\log n}$

$\log\log\log n^k = \log\log\log(n^k)$

$= \log\log(k \cdot \log n)$

$= \log[\log k + \log\log n]$

$\approx \underline{\log\log\log n}$

Ans: $\log\log^k \log n = \log\log\log^k n = \log\log\log n^k < \log^k \log\log n$

Q Write given fns in Asy Increasing Order.

⇒ $\log n!$, $\sqrt{n}$, $(\log\log n)!$, $(\log n)!$, 2^n, $2^{n/2}$, 2^{2n}, 2^{n+1}

$(\log n)^{\log n}$, $(\log n)^n$, $n^{\log n}$, $n\log n$.

Soln

Poly: $\log n!$, $\sqrt{n}$, $(\log\log n)!$, $n\log n$

Exp: $(\log n)!$, 2^n, $2^{n/2}$, 2^{2n}, 2^{n+1}, $(\log n)^{\log n}$, $(\log n)^n$, $n^{\log n}$

$(\log\log n)! < \sqrt{n} < \log n! = n\log n$

$(\log\log n)! < \sqrt{n} < \log n! = n\log n$ $< (\log n)! < (\log n)^{\log n} < n^{\log n} < 2^{n/2} < 2^n = 2^{n+1}$

$< 2^{2n} < (\log n)^n$.

Q PSU $O(n) + \Omega(n) + \Theta(n) =$

a) $O(n)$ b) $\Omega(n)$ c) $\Theta(n)$ d) none.

$O(n) + \Omega(n) + \Theta(n)$

$[\leq n] + [\geq n] + [= n]$

Case 1. $[\sqrt{n} + n^2 + n] = \Omega(n)$

Case 2. $[n + n + n] = \Omega(n)$

⇒ Armstrong rule ove Notations $[O, \Omega, \Theta, o, \omega]$

a] Reflexivity :- [O, Ω, Θ allowed reflexivity].

$f(n) = O(f(n))$

$f(n) = \Omega(f(n))$

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

b] Transitivity :-

if $f(n) = O(g(n))$ & $g(n) = O(h(n))$ then $f(n) = O(h(n))$

Similarly Ω, Θ, o, ω also represent Transitivity.

c] Assymetric :-

1] O, Ω

$f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

2] o, ω

$f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$

⇒ $f(n) * c = \frac{f(n)}{c} = \Theta(f(n))$

$f(n) * c = \frac{f(n)}{c} = O(f(n))$

$f(n) * c = \frac{f(n)}{c} = \Omega(f(n))$

⇒ Find T/F :-

$f(n) + g(n) = O(\max(f(n), g(n)))$ } True

$f(n) + g(n) = \Omega(\max(f(n), g(n)))$ } True

$f(n) + g(n) = \Theta(\max(f(n), g(n)))$ } True

$f(n) + g(n) = o(\max(f(n), g(n)))$ } False.

$f(n) + g(n) = \omega(\max(f(n), g(n)))$ } False.

⇒ Test T/F

① $f(n) = O((f(n))^2)$

⇒ $f(n) = 1/n$

$1/n = O(1/n^2)$ is false.

② if $f(n) = O(g(n))$ then

$2^{f(n)} = O(2^{g(n)})$ is also false.

⇒ $f(n) = 2n$ $g(n) = n$

$2n = O(n) \Rightarrow 2^{2n} = O(2^n)$

False

***

$\Rightarrow (x+a)^n$ (a is constant) x,n is not const. [Binomial fun]

$(x+a)^n = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0 = \Theta(x^n)$

$c_n \cdot x^n \le [x+a]^n \le [c_n + c_{n-1} + \ldots c_0] \cdot x^n$

$\Rightarrow \underbrace{c_n \cdot x^n} \le \underbrace{c_n x^n + c_{n-1} x^{n-1} + \ldots c_0} \le \underbrace{[c_n + c_{n-1} + \ldots c_0]} x^n$

$\text{Const} \cdot g(n) \le f(n) \le \text{Const} \cdot g(n)$

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

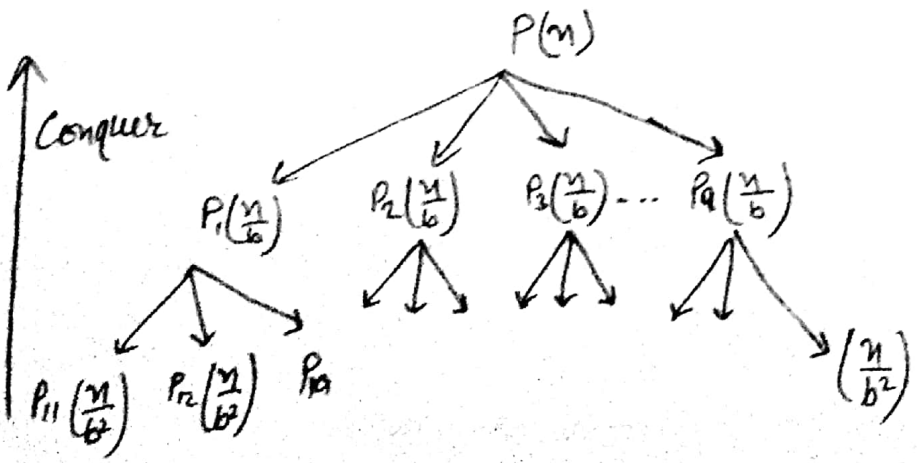
$(n+a)^n = \Theta(x^n)$

DIVIDE and CONQUER DESIGN TECHNIQUES:—

Problem P with n i/p's if not small problem

Divide P into "a" #f subproblems $[P_1 P_2 \ldots P_n]$ each with n/b i/p's && if sub problem not small, divide into subsubproblem until subproblem becomes small.

Return sol of small problem && conquer solution in bottom-up approch untill returns solution (P).



untill Sub problem becomes small
&&
return sol of smallproblem
&&
using conquer combine solutions in bottom up approach.

Control Abstraction of D and C:-

Algo DAndC(n) (Tn)

g(n) [if (n==1)
return (Solution (P))
else

aT(n/b)+f(n) [{ // Divide
Divide Problem P with n I/p's into 'a' #f subproblems with n/b inputs each
f(n)
Conquer (DAndC(n/b), DAndC(n/b), ---- , DAndC(n/b))
T(n/b) T(n/b) T(n/b)
}

Recurrence Rel. of DAndC Algo:-

$$T(n) = \begin{cases} g(n) = \theta(1) & , \text{if } n=1 \\ aT(n/b)+f(n) & , \text{if } n>1 \end{cases}$$

Where a & b constants
T(n): TC of DAndC(n)
g(n): TC to solve small problem
f(n): TC to divide & conquer.

Solving of Recurrence Rel

Imp
1) Substitution Method
2) Recursive tree Method
3) Master Methods $\Rightarrow T(n) = aT(n/b) + f(n)$

Opt 4) change of variable method $\Rightarrow T(n) = aT(\sqrt[b]{n}) + f(n)$

DM 5) Generating fun Method.

Substitution Method

i) $T(n) = \begin{cases} 7T(n/2) + n^2 & n > 1 \\ c & n = 1 \end{cases}$

$T(n) = 7T(n/2) + n^2$

$= 7\left[7T(n/2^2) + \left(\frac{n}{2}\right)^2\right] + n^2 \quad \because \left[T(n/2) = 7T(n/2^2) + \left(\frac{n}{2}\right)^2\right]$

$= 7^2 T\left(n/2^2\right) + \frac{7}{4} \cdot n^2 + n^2$

$= 7^2\left[7T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right] + \frac{7}{4} n^2 + n^2$

$= 7^3 T\left(n/2^3\right) + n^2\left[\left(\frac{7}{4}\right)^2 + \left(\frac{7}{4}\right) + 1\right]$

$\vdots$ k times

$= 7^k T\left(n/2^k\right) + n^2\left[\left(\frac{7}{4}\right)^{k-1} + \cdots\cdots + \left(\frac{7}{4}\right) + 1\right]$

$\frac{\left(\frac{7}{4}\right)^k - 1}{\frac{7}{4} - 1}$

$= 7^k T\left(n/2^k\right) + \frac{4}{3} \cdot n^2\left[\left(\frac{7}{4}\right)^k - 1\right]$

$\left[\because \frac{n}{2^k} = 1 \quad \therefore k = \log_2 n\right]$ ⟶ Terminating condn

$= 7^{\log_2 n} T(1) + \frac{4}{3} n^2\left[\left(\frac{7}{4}\right)^{\log_2 n} - 1\right]$

$= c \cdot n^{\log_2 7} + \frac{4}{3}\left[n^{\log_2 (7/4) + 2} - n^2\right] \quad \left[a^{\log_c b} = b^{\log_c a}\right]$

$T(n) = c \cdot n^{\log_2 7} + \frac{4}{3} \cdot n^{\log_2 7} - \frac{4}{3} n^2$

$= \theta\left(n^{\log_2 7}\right) \approx \underline{\theta\left(n^{2.81}\right)}$

2) $T(n) = \begin{cases} 2T(n/2) + n \cdot \log_2 n & n > 1 \\ c & n = 1 \end{cases}$

$T(n) = 2T(n/2) + n \cdot \log_2 n$

Substituting value of $T(n/2^k)$, k times

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

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

$\log \frac{n}{2^{k-1}} = \log \frac{2n}{2^k} = 1.$

$\log \frac{n}{2^{k-2}} = \log \frac{2^2 n}{2^k} = 2.$

$T(n) = n \cdot c + n[1 + 2 + 3 + \dots + \log n]$

$\boxed{T(n) = cn + n \cdot \frac{(\log n)(\log n + 1)}{2}}$

$\boxed{T(n) = \theta(n \cdot \log^2 n)}$

---

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

Ans:- $\theta(n \cdot \log\log n)$

---

4) Loeman Ex

$T(n) = \begin{cases} \sqrt{n}\,T(\sqrt{n}) + n, & n > 2 \\ c, & n \le 2 \end{cases}$

$T(n) = n^{1/2} T(n^{1/2}) + n$

$= n^{1/2}\left[n^{1/2^2} T(n^{1/2^2}) + n^{1/2}\right] + n \quad \left[\because T(n^{1/2}) = n^{\frac{1}{2^2}} T(n^{1/2^2}) + n^{1/2}\right]$

$= n^{3/2} T(n^{1/2^2}) + 2n$

$= n^{3/2}\left[n^{1/2^3} T(n^{1/2^3}) + n^{1/2^2}\right] + 2n \quad \left[\because T(n^{1/2^2}) = n^{1/2^3} T(n^{1/2^3}) + n^{1/2^2}\right]$

$= n^{7/2^3} T(n^{1/2^3}) + 3n$

after k times.

$n^{\left[\frac{2^k-1}{k}\right]} \cdot T(n^{1/2^k}) + k \cdot n$

$\left[n^{\frac{1}{2^k}} = 2 \quad \therefore \quad 2^k = \log_2 n \;,\; k = \log\log_2 n\right]$ Terminating cond^n

$$T(n) = \frac{n}{n^{1/2^k}} T\left(n^{1/2^k}\right) + k \cdot n$$

$$T(n) = \frac{n}{2} \cdot T(2) + (\log\log n) \cdot n$$

$$\boxed{T(n) = c \cdot \frac{n}{2} + n \log\log n} = \boxed{\theta(n \log\log n)}$$

Master's Theorems:—

$$T(n) = aT(n/b) + f(n) \quad [a \& b \text{ const}]$$

Case 1⟩ :- If $f(n) = O(n^{\log_b^a - \epsilon})$ $[\epsilon > 0$ real number$]$

then $\boxed{T(n) = \theta(n^{\log_b^a})}$

Case 2⟩ :- If $f(n) = \theta(n^{\log_b^a})$

then $\boxed{T(n) = \theta(n^{\log_b^a} \cdot \log_b^n)}$

Case 3⟩ :- If $f(n) = \Omega(n^{\log_b^a + \epsilon})$ $[\epsilon > 0$ real number$]$

then $\boxed{T(n) = \theta(f(n))}$

Eg $T(n) = 7T(n/2) + n^2$

Case 1 → $n^2 = O(n^{\log_2^7 - \epsilon})$

$n^2 \leq n^{2.81 - \epsilon}$ [Here ϵ value can be $0 - 0.8$ So it satisfies the condition $n^2 \leq n^{2.81}$]

$$\boxed{T(n) = \theta(n^{\log_2^7})}$$

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

Case i → $n^2 = O(n^{\log_2^4 - \epsilon})$

$n^2 \leq n^{2-\epsilon}$ × [Not satisfying the condition]

Case 2 → $n^2 = \theta(n^{\log_2^4})$ ✓

$$\boxed{T(n) = \theta(n^2 \cdot \log_2^n)}$$

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

Case 1 ⇒ $n^2 = O(n^{\log_2^3 - \epsilon})$

$n^2 \leq n^{1.69 - \epsilon}$ (Not satisfying)

Case 2 ⇒ $n^2 = \theta(n^{\log_2^3})$

$n^2 = n^{1.69}$ × (Not Satisfying)

Case 3 ⇒ $n^2 = \Omega(n^{\log_2^3 + \epsilon})$

$n^2 \geq n^{1.69 + \epsilon}$

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

1) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \text{const} \iff f(n)$ & $g(n)$ Asy equal.

2) $\frac{f(n)}{g(n)} = \text{const or logfun} \iff f(n)$ & $g(n)$ polynomially equal.

	$f(n)$	$g(n)$	Asy comp	Poly comp.
(1)	$2n^2$	n^2	$f(n) = g(n)$	$f(n) = g(n)$
(2)	n^2	$n^2 \cdot \log n$	$f(n) < g(n)$	$f(n) = g(n)$
(3)	n^2	$\frac{n^2}{\log n}$	$f(n) > g(n)$	$f(n) = g(n)$
(4)	n^2	$n^{2.1}$	$f(n) < g(n)$	$f(n) < g(n)$
(5)	$n^2(\log n)^{10}$	$n^{2.5}$	$f(n) < g(n)$	$f(n) < g(n)$
(6)	n^2	2^n	$f(n) < g(n)$	$f(n) < g(n)$

Different way of using Master's theorem :-

$$T(n) = aT(n/b) + f(n)$$

case 1) if $n^{\log_b^a}$ polynomially bigger than $f(n)$

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

Case 2) If $n^{\log_b^a}$ & $f(n)$ Asymptotically equal

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

Case 3) If $f(n)$ polynomially bigger than $n^{\log_b^a}$

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

$T(n) = aT(n/b) + f(n)$	$n^{\log_b^a}$	$f(n)$	
1) $T(n) = 2T(n/2) + \sqrt{n}$	$n^{\log_2^2} = n$	$\sqrt{n} = n^{0.5}$	$= \theta(n)$
2) $T(n) = T(n/2) + \sqrt{n}$	$n^{\log_2^1} = n^0$	$\sqrt{n} = n^{0.5}$	$= \theta(\sqrt{n})$
3) $T(n) = 2T(n/3) + n^3$	$n^{\log_3^2} \approx n^{2.9}$	n^3	$= \theta(n^3)$
4) $T(n) = 27T(n/3) + n^3$	$n^{\log_3^{27}} = n^3$	n^3	$= \theta(n^3 \cdot \log_2^n)$
5) $T(n) = T(n/3) + c$	$n^{\log_3^1} = n^0 = 1$	c	$= \theta(\log_3^n)$
6) $T(n) = 4T(n/2) + n\log n$	$n^{\log_2^4} = n^2$	$n\log n$	$= \theta(n^2)$
7) $T(n) = 4T(n/2) + n^3\log n$	$n^{\log_2^4} = n^2$	$n^3\log n$	$= \theta(n^3\log n)$

Q $T(n) = 2T(n/2) + n\log n$.

$f(n) = n\log n$, $n^{\log_b^a} = n$ } $f(n)$ & $n^{\log_b^a}$ polynomially equal but not Asymptotically equal

So Here Master theorem fails.

$\Rightarrow T(n) = 2T(n/2) + \frac{n}{\log n}$

$\Rightarrow T(n) = 4T(n/2) + n^2 \log\log n$

$\Rightarrow T(n) = 4T(n/2) + \frac{n^2}{(\log n)^{10}}$

$\Rightarrow T(n) = 16T(n/2) + n^4(\log n)^{10}$

Modified Case 2

if $f(n) = \theta(n^{\log_b^a} \cdot (\log n)^k)$ $k \geq 0$ constant

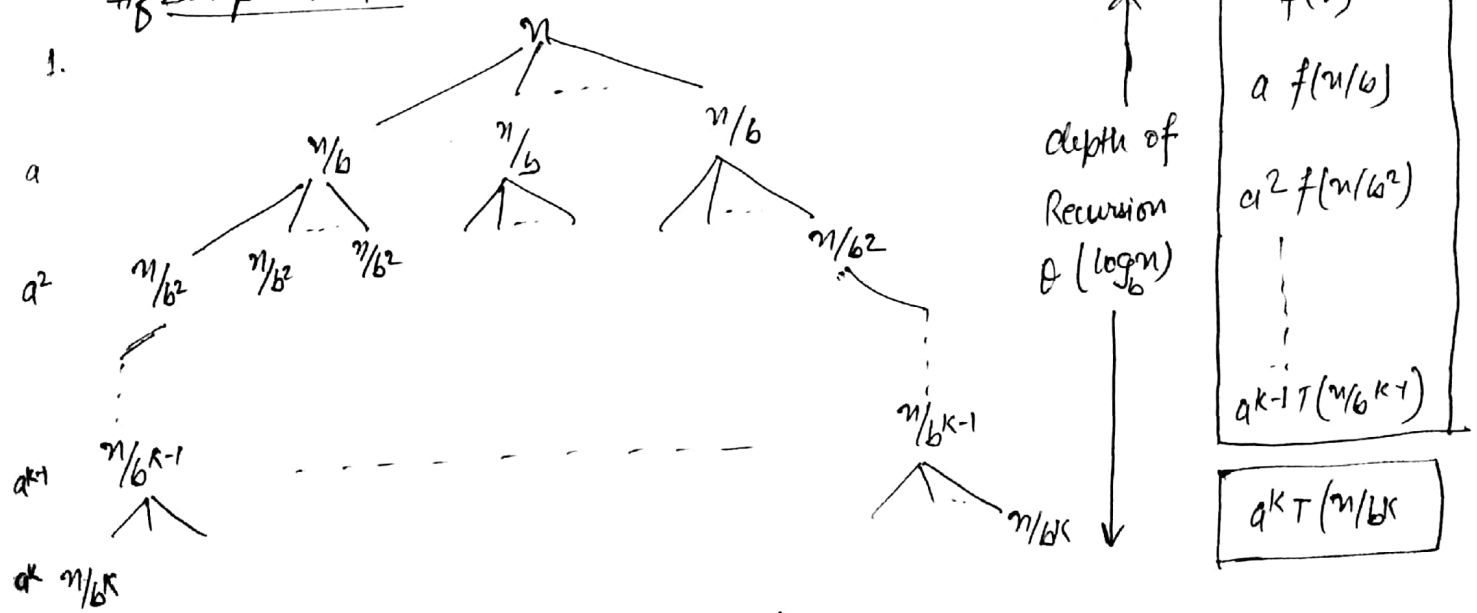
then $\boxed{T(n) = \theta(n^{\log_b^a} \cdot (\log n)^{k+1})}$

26/08/17

Recursive Tree Method

$$T(n) = aT(n/b) + f(n)$$

#f Sub problem



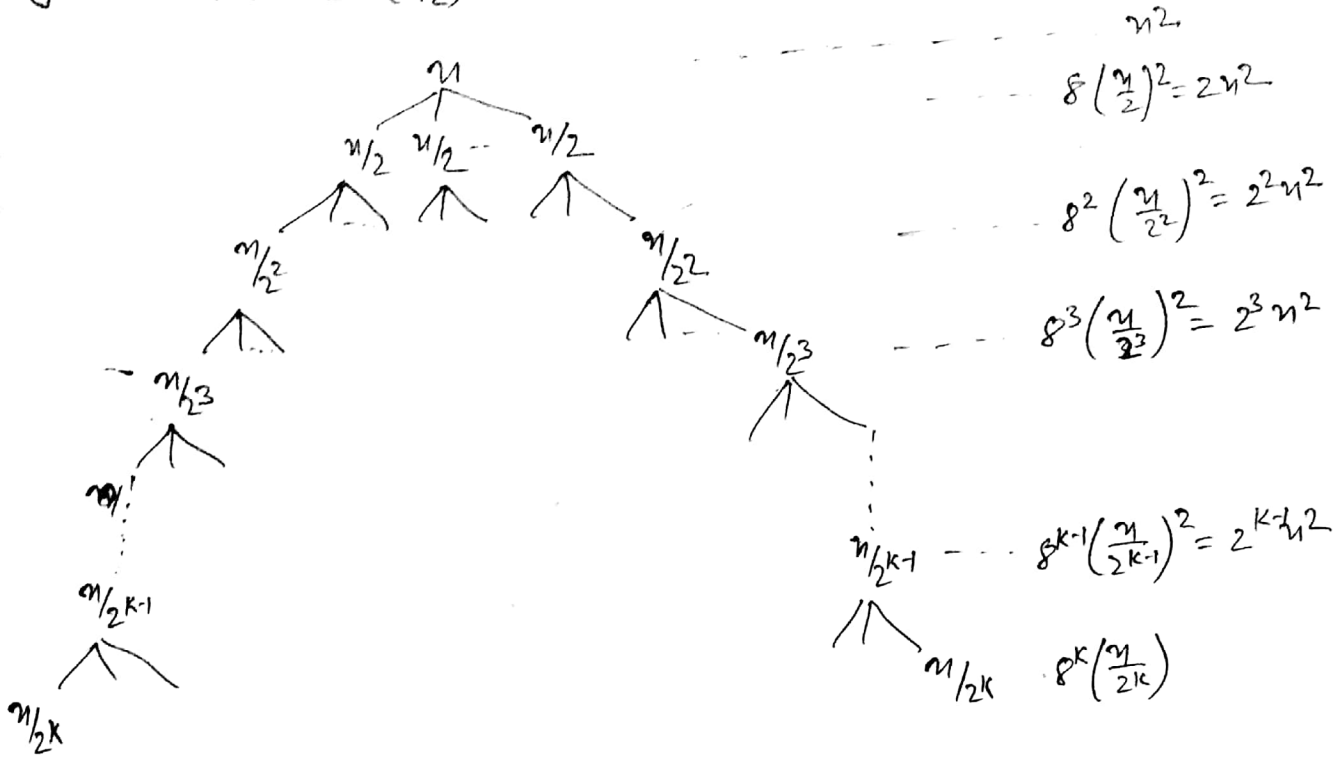
$$T(n) = a^k T(n/b^k) + \sum_{i=0}^{k-1} a^i f(n/b^i)$$

$$\left[\because \frac{n}{b^k} = 1 \quad n = b^k \quad k = \log_b^n\right]$$

$$T(n) = a^{\log_b^n} T(1) + \sum_{i=0}^{\log_b^n} a^i f(n/b^i)$$

$$\boxed{T(n) = n^{\log_b^a} \cdot c + \sum_{i=0}^{\log_b^n - 1} a^i f(n/b^i)}$$

Eg → $T(n) = 8T(n/2) + n^2$



$$T(n) = 8^k T\left(\frac{n}{2^k}\right) + n^2\left[1 + 2 + 2^2 + \ldots 2^{k-1}\right]$$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \quad k = \log_2 n$$

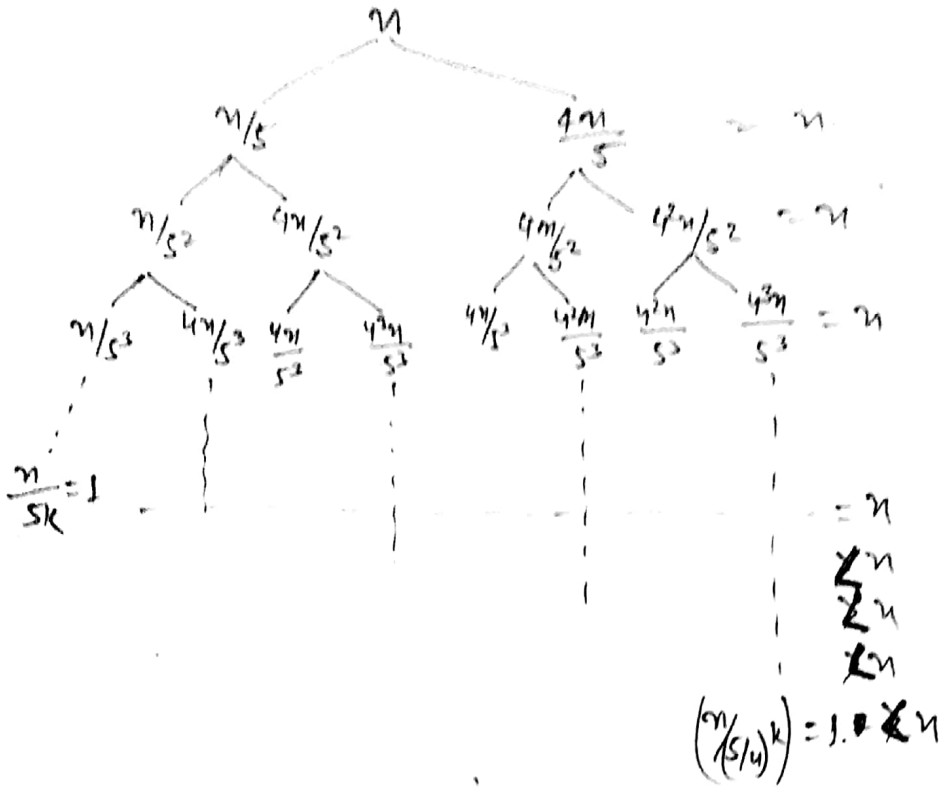
$$T(n) = 8^{\log_2 n}\, T(1) + n^2[n-1]$$

$$\boxed{T(n) = c \cdot n^3 + n^3 - n^2} = \theta(n^3).$$

Q $T(n) = \begin{cases} T(n/5) + T(4n/5) + n, & n > 1 \\ a, & n = 1 \end{cases}$

Too complex to solve using substitution

Recursive tree method can be used to solve above recurrence Rel.

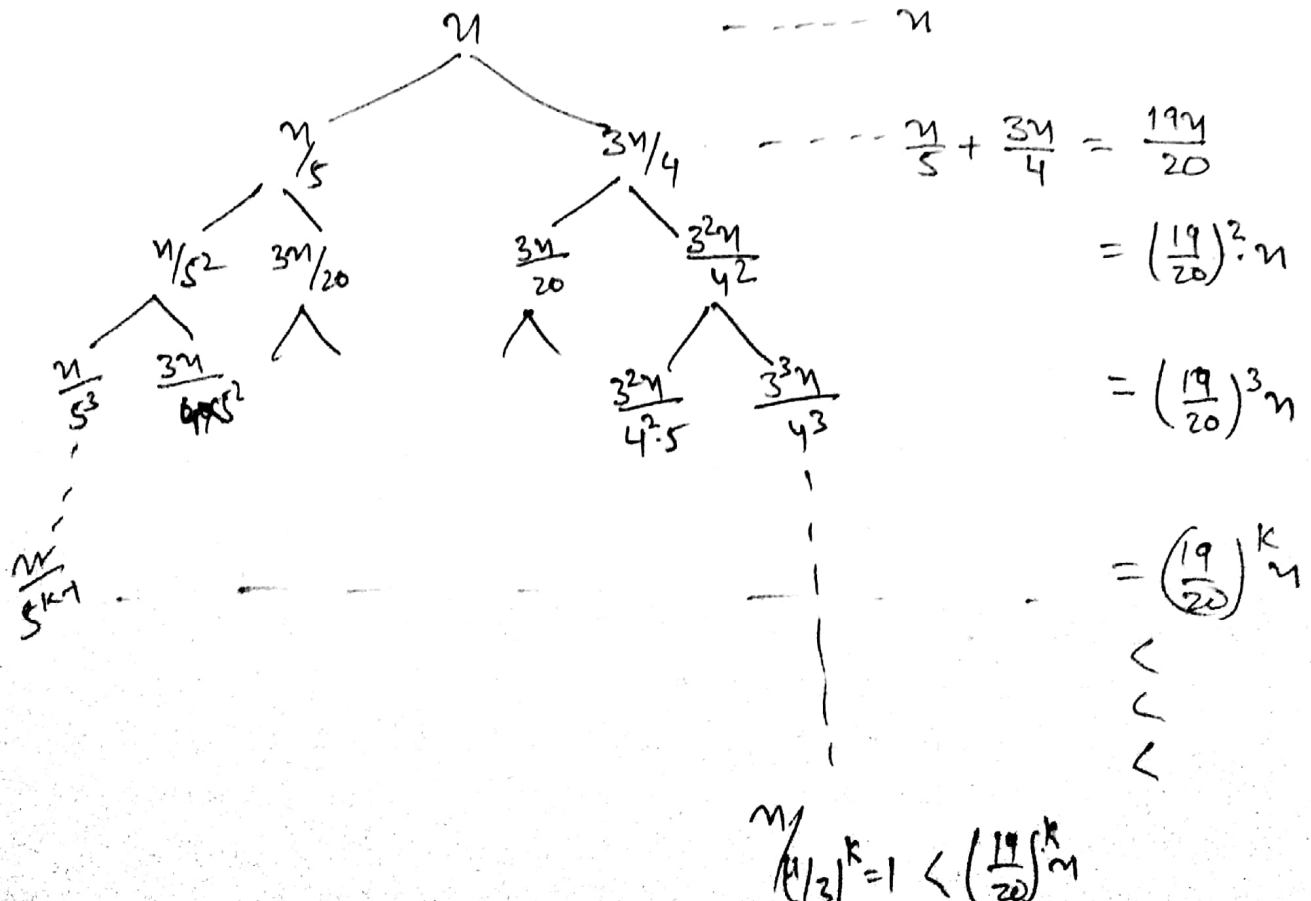


$T(n) = \left(\log_{5/4} n\right) \cdot n$

$= O(n \log n)$

GATE

$$T(n) = \begin{cases} T(n/5) + T(3n/4) + n & , n > 1 \\ 1 & n \le 1 \end{cases}$$



$$T(n) = n\left[1 + \frac{19}{20} + \left(\frac{19}{20}\right)^2 + \dots + \left(\frac{19}{20}\right)^k\right]$$

$$= n\left[\frac{1-\left(\frac{19}{20}\right)^{k+1}}{1-\frac{19}{20}}\right]$$

$$= 20n\left[1 - \left(\frac{19}{20}\right)^{\log_{4/3} n + 1}\right]$$

$$= 20\left[n - n\cdot\left(\frac{19}{20}\right)^{\log_{4/3} n + 1}\right] = \underline{\Theta(n)}$$

Chang of variable method :—

If recurance relation

$$T(n) = aT(\sqrt[b]{n}) + f(n)$$

Convert into Master theorem format of recurence relation.

Assume: $n = b^m \Rightarrow \sqrt[b]{n} = b^{m/b}$

$$T(b^m) = aT(b^{m/b}) + g(m)$$

Assume: $T(b^m) = S(m)$

$$\boxed{S(m) = a\,S(m/b) + g(m)}$$

Now apply master theorem

Eg→ $T(n) = T(\sqrt{n}) + C$

[Assume $n = 2^m$ $\sqrt{n} = 2^{m/2}$]

$\Downarrow$ $m = \log_2 n$

$$T(2^m) = T(2^{m/2}) + C$$

[Assume: $T(2^m) = S(m)$

$T(2^{m/2}) = S(m/2)$]

$S(m) = S(m/2) + C$ Apply Master thm

$S(m) = \Theta(\log_2 m)$

$S(m) = T(2^m) = T(n)$

$= \underline{\Theta(\log\log n)}$

Eg → $T(n) = 2T(\sqrt{n}) + c$

$T(n) = 2T(n^{1/2}) + c$

$n = 2^m \Rightarrow n^{1/2} = 2^{m/2}$

$T(2^m) = 2T(2^{m/2}) + c$

$T(2^m) = S(m) \Rightarrow T(2^{m/2}) = S(m/2)$

$S(m) = 2S(m/2) + c$ [Apply master theorem]

$S(m) = \theta(m)$

$S(m) = \boxed{T(n) = \theta(\log_2 n)}$

Eg → $T(n) = 5T(5\sqrt{n}) + \log_5^n$

$n = 5^m$

$T(5^m) = 5T(5^{m/5}) + m$

$T(5^m) = S(m)$

$S(m) = 5S(m/5) + m$

$S(m) = \theta(m \log m)$ Applying Master Method

$S(m) = T(n) = \theta(\log_5 n \cdot \log_5 \log_5 n)$

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

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

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

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

} Should use generating function method [Descreate maths]

n^{th} Fib Num

$$Fib(n) = \begin{cases} 1 & , n = 0 \\ 1 & , n = 1 \\ fib(n-1) + fib(n-2) & , n > 1 \end{cases}$$

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

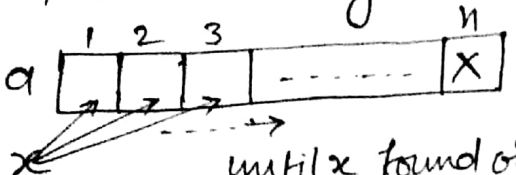
$T(n) = 2T(n-2) + n, \quad T(0) = 1 \; T(n) = 2$
$T(1) = 1$

Binary Search :- [D and C Example]

Linear Search Algo:-

$a[1 \ldots n]$ n elmnts in array

x is searching element



[Comp]

```
Algo Linear Search (a, n, x)
{   for (i=1; i≤n; i++)
    {  if (x == a[i])
           return (i)
    }
   return (-1)
}
```

Best case TC:-
if x present at a[1]
#f Comp: 1
$TC = \theta(1)$

Worst case TC:-
if x is not presnt in array or
present at a[n]
#f Comp: n
$TC = \theta(n)$

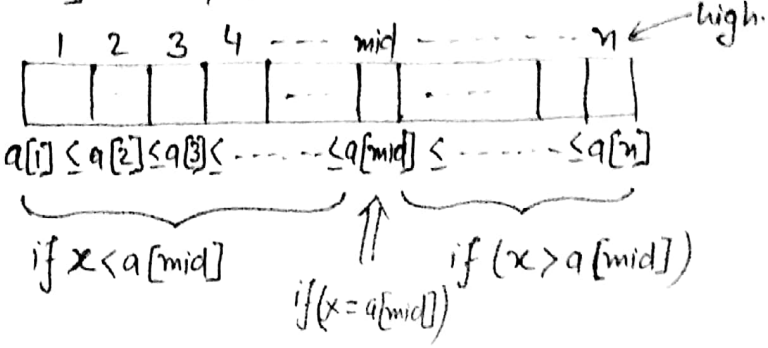
Avg Case TC:-
Comp: $\frac{1+2+3+\ldots+n}{n}$

$\therefore \frac{n(n+1)}{2n} = \frac{n+1}{2}$ Comp

$TC = \theta(n)$

Binary Search :-

a[1...n] must be in Sorted order



Algo BinSearch (a, n, x)

```
{   // a[1...n] are n sorted elements
    low = 1; high = n
    while (low ≤ high)
    {
        mid = (low + high)/2;
        if (x < a[mid])
            high = mid - 1;
        else if (x > a[mid])
            low = mid + 1
        else
            return (mid); // element x present
    }
    return (-1); // element x is not present.
}
```

TC of straight Binary Search :-

Best case : Tc : $\boxed{\theta(1)}$

When x is present at middle of array

Worst case TC :-

Maxm no of encounter of while loop is the worst case for the algo.

low = 1 high = n [n elmnts for search]

low = 1 high = $\frac{n}{2}-1$ [$\frac{n}{2}$ elmnts for search]

[$\frac{n}{2^2}$ elmnts for search]

[$\frac{n}{2^K}$ elmnts for search]

K

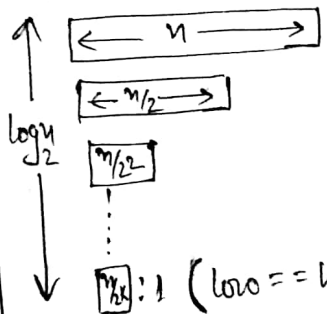
Max ($\log_2 n$) while loops to terminate algo.

⇒ Avg case TC of Binary Search : $\theta(\log n)$

Recursive Binary Search :-

```
Algo RBSearch(l, h)
{  // a[l...h] array elmnts
   if (l == h) // Small(P)
   { if (x == a[l])
        return (l)
     else return (-1)
   }
   else
   {   m = (l+h)/2;
       if (x < a[m])
          RBSearch(l, m-1);
       else if (x > a[m])
          RBSearch (m+1, h);
       else return(m)
   }.
```

Worstcase TC:



WC TC Recurrence Rel of Bin Search

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

$= \theta(\log n)$

Best case TC of Bin Search:-

$\theta(1)$

when x present middle of array.

SC of recursive BinSearch :-

$\theta(\log n)$ // Stack space.

Excluded I/P array space

Because TC of recursive and non recursive algo is same. So we preffer non recursive algo to use. Because. If array space is excluded.

Problems :-

1) If array of 10 elemnts in sorted order. Avg # of elemnts comp to search elemnts using Binary search.

a) 2.9 b) 3 c) 4 d) 2.

# of comparison for the possibility of element present at every position individually.

at first position. ⇒ 3 comp.
2nd " ⇒ 2 Comp.
3rd " ⇒ 3 Comp.
4th " ⇒ 4 comp
5th " ⇒ 1 Comp.
6th " ⇒ 3 Com
7th position ⇒ 4 Comp
8th " ⇒ 2 Comp
9th " ⇒ 3 Comp
10th " ⇒ 4 Comp.

$$\therefore \text{Avg \# of comparisons} = \frac{3+2+3+4+1+3+4+2+3+4}{10}$$

$$= \frac{29}{10} = \underline{2.9}$$

3] Modify Binary Search Algo such that algo should return position of first occurance of searching elemnt whose worst case TC $\theta(\log n)$

1	2	3	4	5	6	7	8	9
12	12	14	14	14	14	15	15	20

[x = 14] should return (3);

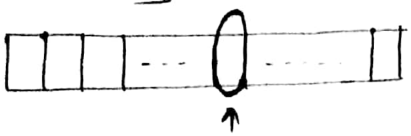
⇒
```
{  low = 1  high = n
   while(low ≤ high)
   {  mid = (low+high)/2
      if (x < a[mid])
          high = mid - 1;
      else if (x > a[mid])
          low = mid + 1;
      else if (a[mid] == a[mid-1])
          high = mid - 1.
   }
   else
       return(mid)
   return(-1);
}
```

3) What is TC to find leading element exists or not in array of n elements.

a) $\theta(\log n)$ b) $\theta(n)$ c) $\theta(n \log n)$ d) $\theta(n^2)$

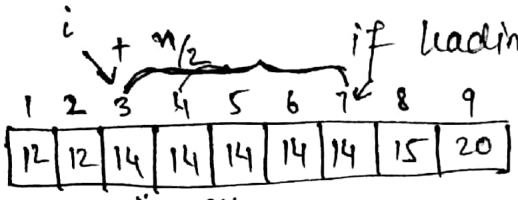
---

for 1) Sorted array

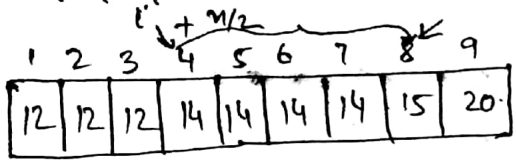


// Sorted array.

Middle element must contain leading element if leading element exists.



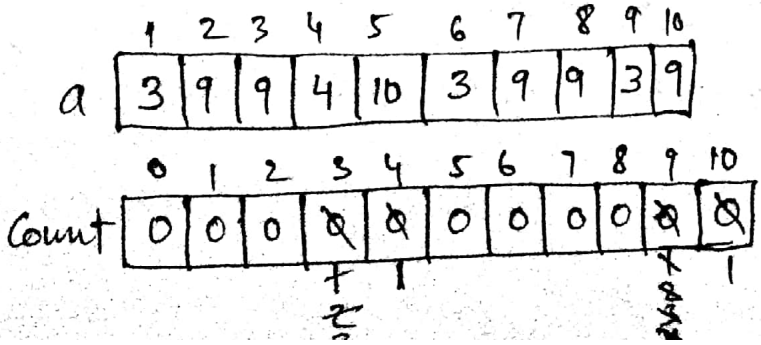
⇒ leading elements exists



⇒ leading element no exists.

```
X = a[mid]
i = call FirstBinSearch(low, high, X)
if (a[i + n/2] == X)
        Then leading element present
else
        leading element not present.
```

for 2) Unsorted array of n elements range of elements [0:...n]



```
for(i=0; i≤n; i++)
{  count[i] = 0 }
for(i=1; i≤n; i++)
{ count[a[i]] = count[a[i]]+1; }
for(i=0; i≤n;i++)
{ if (count[i] ≥ n/2)
        then return (true)
}
```

TC: $\theta(n)$

---

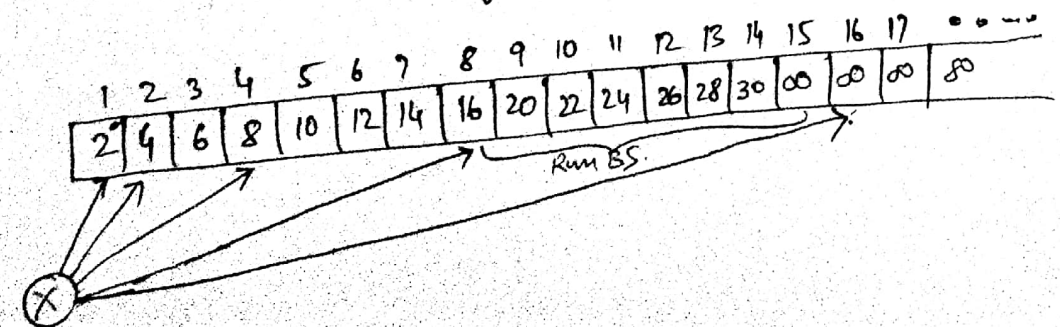
for (iii) array of n elements

i) Sort array using best sorting algo } $\theta(n\log n)$

ii) Find leading element in sorted array } $\theta(\log n)$

$\theta(n\log n)$

Q4] Assume size of array unknown [Infinite]
First few elements of array is sorted order
remaining elements "∞"
Write Binary search To search "x" array or
To find last position of defined number.



```
i = 1
while (a[i] ≠ ∞)
{   i = 2i
}
low = i/2 ; high = i
call BS (low, high)
    // for n elmnts
```

$\Theta(\log n)$

$\Theta(\log n)$

TC : $\Theta(\log n)$

// Assume last defined num at n^{th} position.

Q5] Array n distinct sorted elments which can be -ve or +ve. Find any elmnt "x" whose index "x" present in array or not ? ⇒ Worst case TC : $\Theta(\log n)$.

	1	2	3	4	5	6	7	8	9
x	-5	-3	-2	0	2	3	5	8	10

	1	2	3	4	5
	-5	-3	-2	0	2

x

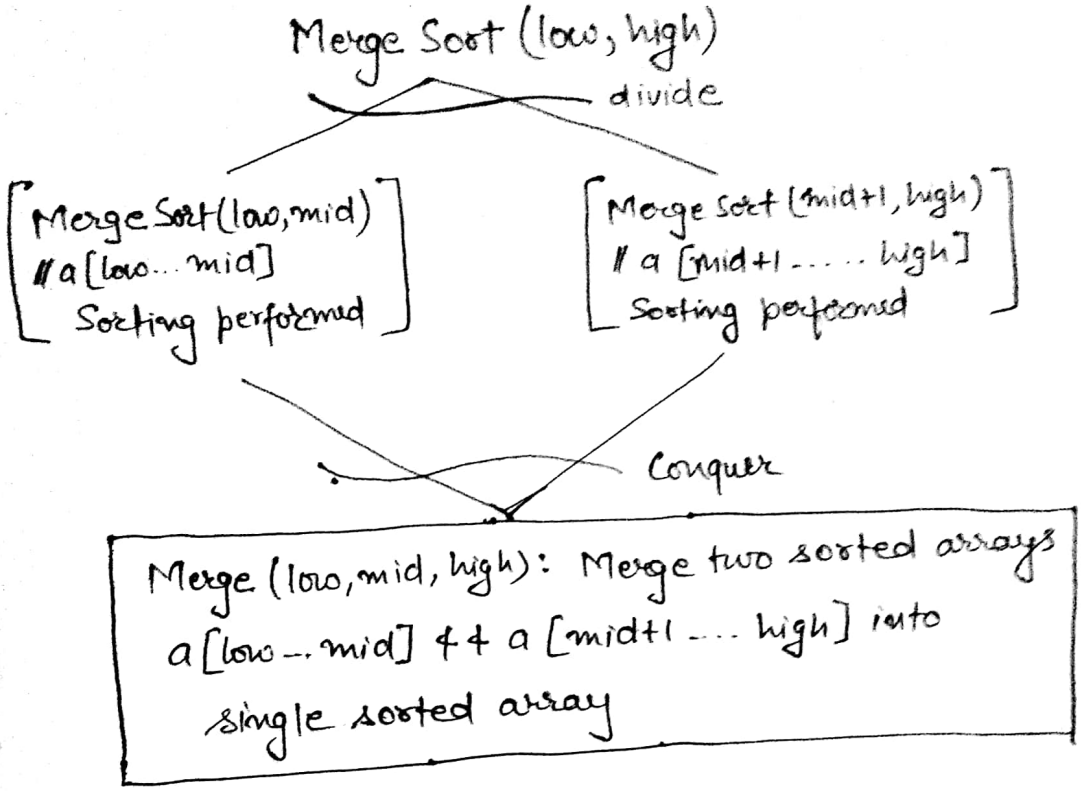
	5	6	7	8	9
	8	10	15	20	22

x

```
low = 1  high = n
while (low ≤ high)
{   mid = (low + high)
    if (mid < a[mid])
        high = mid - 1
    else if (mid > a[mid])
        low = mid + 1;
    else return (mid);
}
return (-1);
```

Merge Sort Algo [Fully D and C Approach]

a [low .... high] one array of n elements.




```
Algo Merge Sort (low, high)
{
    // a[low... high] are 'n' elements
    if (low < high)
    {   mid = (low+high)/2;
        Merge Sort (low, mid);
        Merge Sort (mid+1, high);
        // Conquer
        Merge (low, mid, high);
    }
}
```

Algo Merge (low, high, mid) ⇒ TC : $\theta(n)$

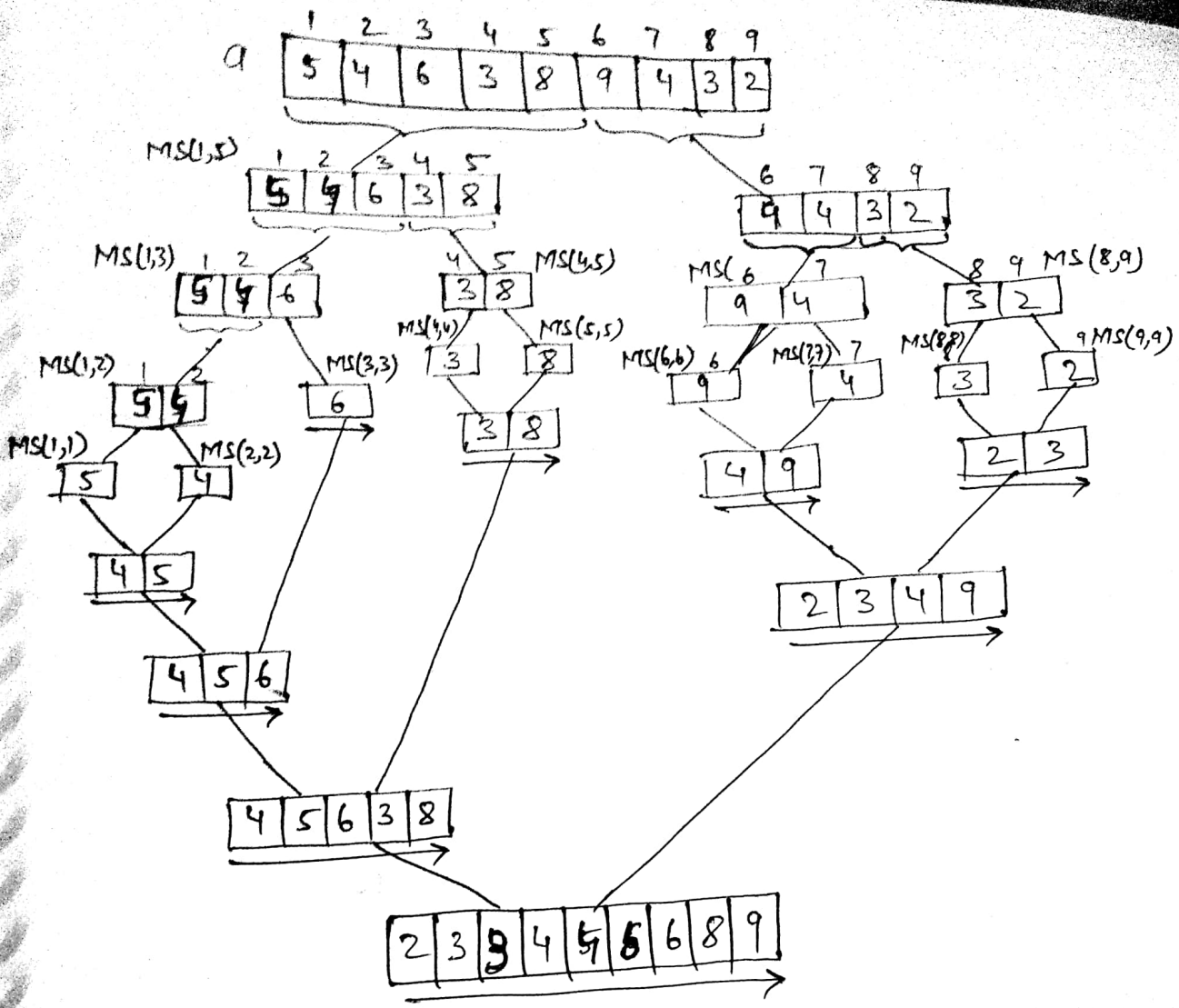
```
{  // To merge two sorted arrays a[low ... mid] &&
   // a[mid+1 ... high] into single sorted array
   i = low ; j = mid+1 ; k = low
   while (i ≤ mid && j ≤ high)                          } n-1
   { if (a[i] ≤ a[j])
     { b[k] = a[i] ; i++; }
     else
     { b[k] = a[i] ; j++; }
     k++ ;
   }
   if (i > mid)                                         } 1
     for (x = j ; x ≤ high ; x++)
     { b[k] = a[x] ; k++; }
   else
     for (x = i ; x ≤ mid ; x++)
     { b[k] = a[x] ; k++; }
   for ( x = low ; x ≤ high ; x++)                      } n
   { a[x] = b[x] ; }
}
```

⇒ TC to merge two sorted array of $n/2$ & $n/2$ elmnts each into single sorted array : $\theta(n)$

⇒ TC of Recurrence of Merge Sort :-

$$T(n) = \begin{cases} 1 & , n=1 \\ 2T(n/2) + cn & , n>1 \end{cases} \Rightarrow \underline{\theta(n \cdot \log_2 n)} \text{ [All cases]}$$

⇒ Depth of recursion of Merge sort : $\underline{\theta(\log_2 n)}$



Stable Sorting algo :-

Identical elements of unsorted array should occupy same order in sorted array.

1	2	3	4	5	6
3	2a	0	2b	1	2c

Sorting algo $\Rightarrow$

1	2	3	4	5	6
0	1	2a	2b	2c	3

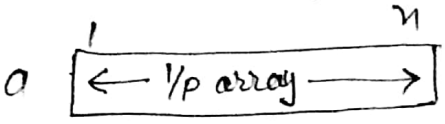
$\Rightarrow$ Stable sorting algo.

Sorting algo $\Rightarrow$ 0 | 1 | 2b | 2c | 2a | 3 $\Rightarrow$ Unstable sorting algo.

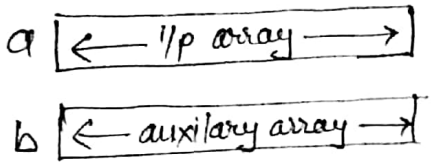
⇒ Inplace sorting algo :-

To sort array of n elements {a[1...n]} if sorting algo uses only i/p array to sort elements. and may use stack space for recursion.

then Inplace sorting.



⇒ Non Inplace sorting :-



Sorting algo uses extra array to sort elements along with I/p array.

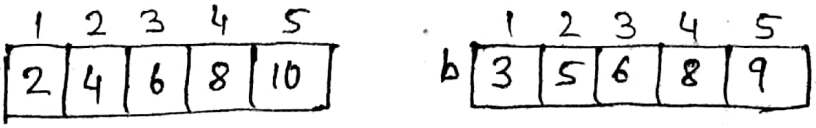
Merge sort is stable sorting algo but not Inplace sorting algo.

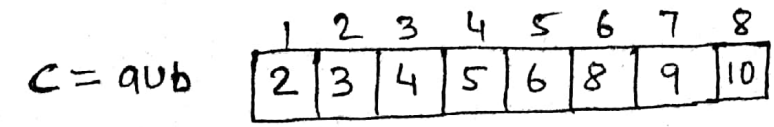
Q ⇒ What is TC to perform

i) Union ii) Intersection iii) Minus

of two sorted arrays n + m distinct elements each ?

Soln





```
i=1  j=1,  k=1
While ( i≤n && j ≤m)
{  if (a[i] < b[j]) {c[k] = a[i]; i++}
   else if(a[i] > b[j]) {c[k]=b[j]; j++}
      else
          {c[k] = a[i]; i++; j++}
      k++;
}
if (i>n) for (x=j; x≤m; x++)
          { c[k] = b[x]; k++}
   else  for(x=i; x≤n; x++)
          { c[k] = a[x]; k++}
```

For intersection algo remove green coloured stmt in above algo, and only "while" loop is sufficient. Discard the last two "for" loops.

for Minus

```
i=1, j=1, k=1.
While ( i≤n && j ≤m)
{ if(a[i] < b[j]) {          c[k]=a[i]; i++}
  else if (a[i]>b[j]) {j++;}.
   else  i++; j++; k++;
}
if (i>n) for (x=j; x≤m; x++)
          { c[k]=b[x]; k++}

}.
```

Q ⇒ How many max ~~com~~ elemnt comp required to Merge two sorted arrays n+m elemnts each into single sorted array: _____

a] n+m b] n+m+1 c] n+m-1 d] n+m

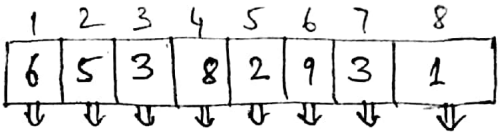
Q. Given a[1...n] array of n elemnts

Inverse defined as two elemnts of array such that

i<j && a[i]>a[j].

What is TC to find #f Inverses of array of n elemnts?

a) $\theta(n^2)$ b) $\theta(n \log n)$ c) $\theta(\log n)$ d) $\theta(n)$



Inv ⇒ 5+4+2+3+1+2+1+0 = 18 Inv

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

TC: $\theta(n^2)$

Counting Inv using Merge Sort

Changing portion only

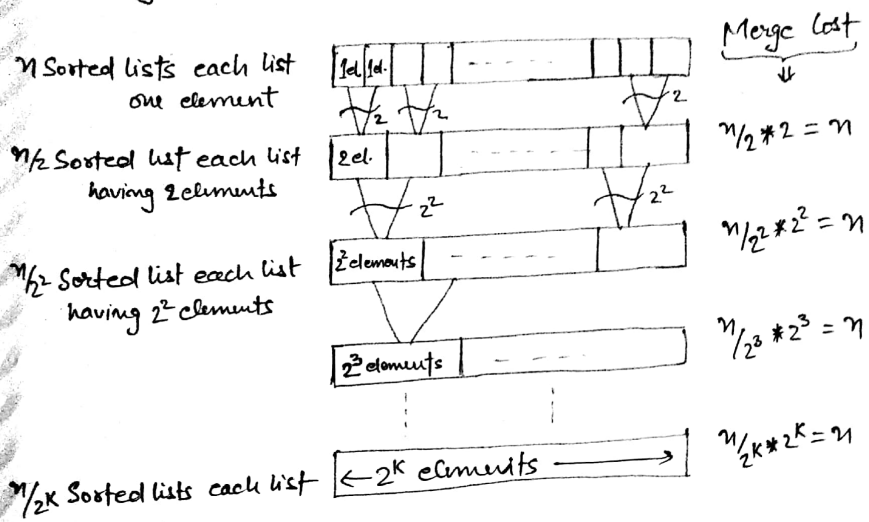
```
while(i<=mid && j<=high)
{ if(a[i]<=a[j])
   { b[k]=a[i]; i++;}
  else
   { b[k]=a[j]; j++;
     Inv=Inv+(mid-i+1)
   }
  k++
}
```

Remaining also will be same.

27/08/17

2 Way Merge Sort:- [Straight Merge Sort]

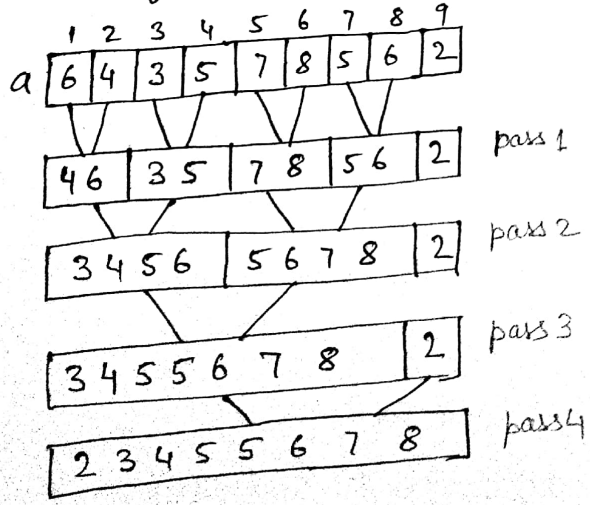
Initially a[1...n] consider as n sorted lists each list one element. Merge two lists untill 'n' sorted lists becomes single sorted list.



$\frac{n}{2^K} = 1 \Rightarrow K = \log_2 n$ [# of pass]

TC of 2-way Merge Sort: $\theta(n \cdot \log_2 n)$

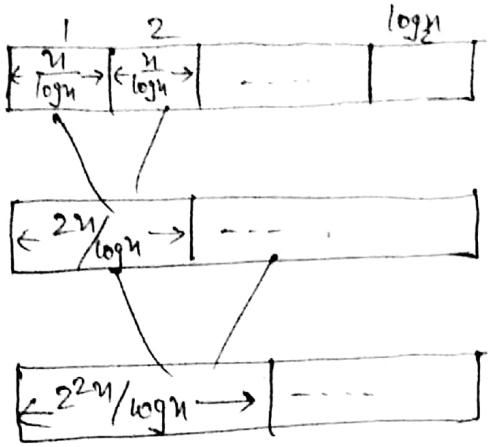
Eg>



Q. Array consists $\log_2 n$ sorted lists, each list $\frac{n}{\log n}$ elements. What is TC to merge array into single sorted list?

Soln

$\log n$ sorted lists each $n/\log n$ elements



$\frac{\log n}{2}$ sorted list each list $\frac{2n}{\log n}$ elements

$\frac{\log n}{2} * \frac{2n}{\log n} = n$

$\frac{\log n}{2^2}$ sorted list each list $\frac{2^2 n}{\log n}$ elements

$\frac{\log n}{2^2} * \frac{2^2 n}{\log n} = n.$

$\frac{\log n}{2^K}$ sorted list

$\frac{\log n}{2^K} = 1$

$K = \log \log n$ [# of pass]

TC: $\theta(n \cdot \log \log n)$

Q. Array consist n strings each string of n characters. What is TC to sort n strings in lexiographic order?

Ans TC: $\theta(n^2 \log n)$

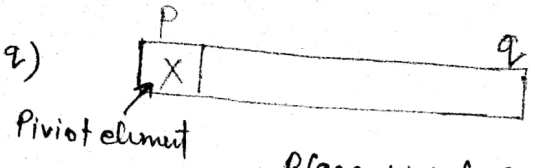
Quick Sort Algo:— [Also partially D And C]

* Widely used Sorting Algo.
* Inplace ++ Not Stable Sorting Algo
* Tony Hoarge [1970] {Turing Award 1985}

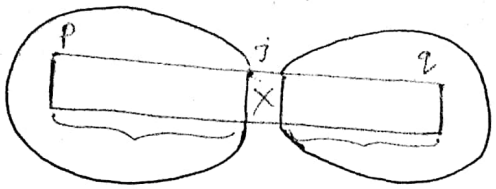
Basic idea of Quick Sort:—

Choose Pivot element and place in correct position.

QS(P,q)



Place piviot elemnt in correct position.



$a[P..j-1] \le x \le a[j+1...q]$

QS(P,j-1) ⇓ ⇓ QS(j+1,q)

Algo Quick Sort

```
{   // a[P-q] array of n elements
    if(P<q)
    {
        J= partition(a,P,q);
        Quick sort(P, j-1);
        Quick sort(j+1, q);
    }.
```

---

partition (a, P, q); // Procedure.

1) Increment i untill a[i] >x

2) decrement j untill a[j] <x

3) if i<j Swap (a[i], a[j])

4) Repeat ①②③ untill i≥j

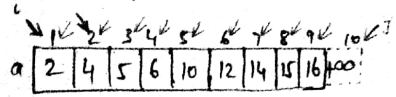
5) Swap (a[P], a[j])

6) return (j).

```
// Algo Partition (a, P, q)
{   a[q+1] = +∞
    i=P ; j= q+1 ; x = a[P]
    do
    {   do
        { i= i+1;
        }while(a[i] <x)
        do
        { j= j-1;
        }while(a[j]>x)
        if (i<j) {swap(a[i],a[j])}
    }while (i<j)
    Swap (a[P],a[j])
    return(j);
}.
```

Partion algo Time complexity:-

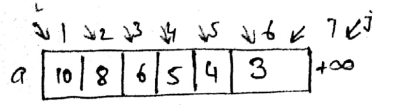
Case 1.



(a[i] < x) : 1 comp

(a[j] > x) : n comp.

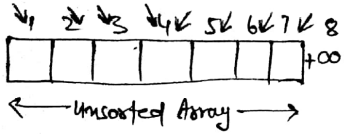
n+1 comp.



(a[i] < x) : n comp.

(a[j] > x) : 1 comp

n+1 comp.



(a[i] < x) : n/2

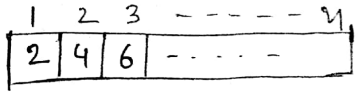
(a[j] > x) : n/2 + 1

n+1 comp.

Hence Time complexity of partion Algo $\boxed{\theta(n)}$ All Cases.

▷ Worst Case TC of Quick sort :-

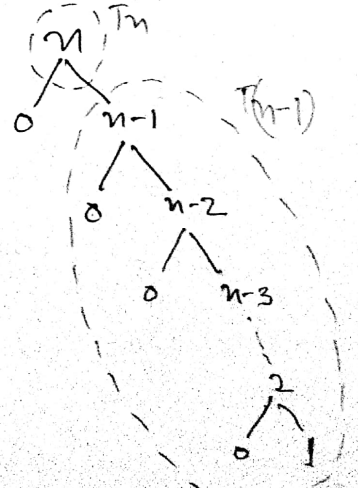
Quick sort behaves as worst case if Given I/p array to algo is in sorted order.



$$T(n) = \begin{cases} T(n-1) + n, & n > 1 \\ 1, & n \le 1 \end{cases}$$

$\boxed{T(n) = \theta(n^2)}$

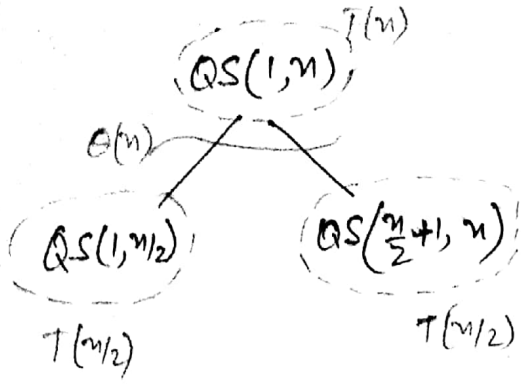
depth of recursion n levels

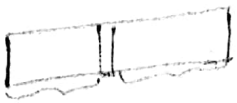


2] Best Case TC of Quick Sort

if pivot element places middle of list in every fun call.

Then QS behaves as best case.





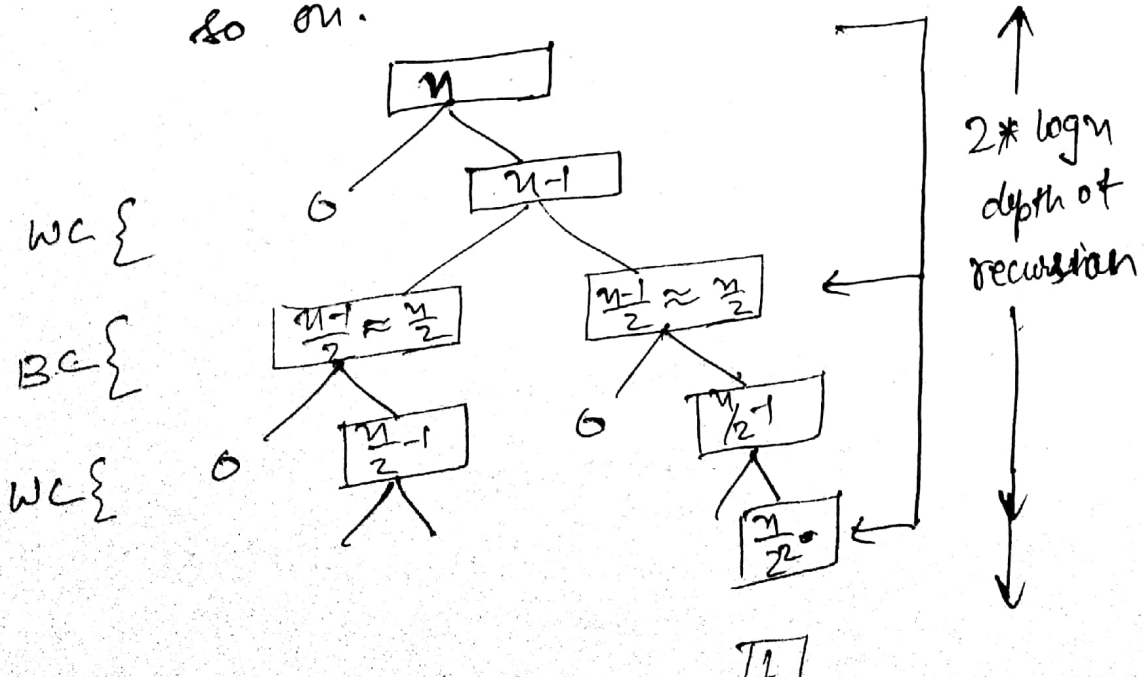
Best case TC recurrence rel of Quick sort

$$T(n) = \begin{cases} 2T(n/2)+n & n \geq 1 \\ 1. & n \leq 1 \end{cases}$$

$$\boxed{T(n) = \theta(n \log_2 n)}$$

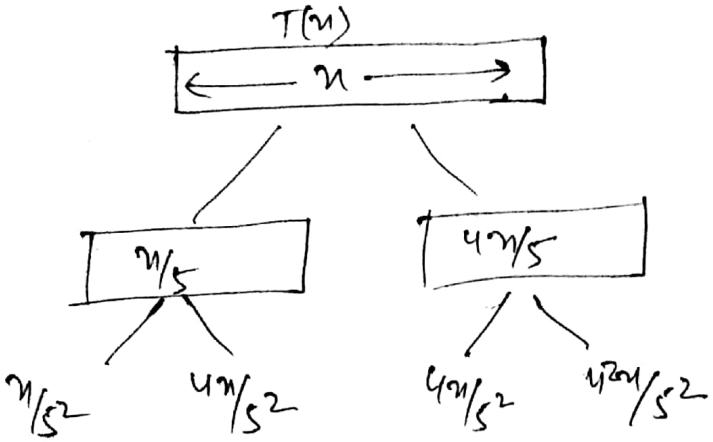
3] Avg Case TC of Quick Sort.

Case 1] Partion algo divides list as Best case + worst case and



Avg Case TC of QS: $\theta(n \log n)$

Case 2] Every time partion divides list into $n/5$ elmnts & $4n/5$ elmnts for given n elmnts.



Recurr Rel'

$$T(n) = \begin{cases} T(n/5) + T(4n/5) + n & n < 1 \\ 1 & n \leq 1 \end{cases}$$

$$\theta(n \log_{5/4} n) \simeq \theta(n \log n)$$

Pivot Selection Method :-

1] First / last element of array is Pivot :-

⇒ Worst case TC of Quick Sort : $O(n^2)$ if I/p array in sorted order.

2] Pivot is Avg of min & Max of array

worst case TC of Quick sort ⇒

2	4	6	(8)	10	12	14

Pivot Selection ⇒ Min = 2, Max = 14, Avg = 8 } $\theta(n)$ time.

```
Algo QS(P,q)
{   if (P<q)
    {   k = Pivot Set(a,P,q) ⇒ θ(n)
        Swap(a[k], a[P])
        j = partition(a,P,q) ⇒ θ(n)
        QS(P, j-1);          ⇒ T(n-2)
        QS(j+1, q)           ⇒ T(1)
    }
}.
```

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

$= \underline{\underline{\theta(n^2)}}$

3] Median element as piviot.

Array of n elements

Median element of array "x" such that

$$\left[\frac{n}{2} \text{ elements}\right] \leq x \leq \left[\frac{n}{2} \text{ elements}\right]$$

if median element of array is Piviot element then worst case TC of Quick sort:

```
Algo QS(P,q)
{   if (P<q)
    {   k = Median(a,P,q) ⇒ Ω(n)
        Swap(a[k], a[P])
        j = Partition(a,P,q) ⇒ θ(n)
        QS(P, j-1)           ⇒ T(n/2)
        QS(j+1, q)           ⇒ T(n/2)
    }
}
```

$TC = \Omega(n \log n)$ (All Cases) which depends on TC of to find median].

Q WC TC of Quicle Sort using Median Piviot Selection.

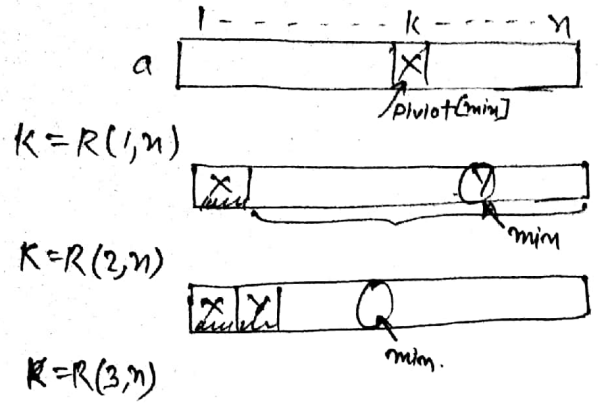
a) $O(n^2)$

b) $O(n)$

c) $\theta(n \log n)$

d) $O(n \log n)$

Random Piviot Selection




```
Algo QS(P,q)
{ if (P<q)
  { k = Random (P,q)
    swap (a[k], a[P])
    j = partition(a,P,q)
    QS(P,j-1)
    QS(j+1,q)
  }
}.
```

WC TC of QS: $\theta(n^2)$

Every Time Randomly chooses piviot which unknowonly min elemnt.

[Probability of Min/Max element selected by Random Piviot Selection for every partition is almost "0"].

Q. what is TC of QuickSort using Random Piviot Sel.

a) $O(n^2)$

b) $O(n\log n)$

c) $\theta(n\log n)$

d) $\theta(n)$

Q.) Piviot Sel Algo $\theta(n)$ time which is $n/5^{th}$ biggest elemnt of array.

WC TC of QS?

a) $\theta(n^2)$ b) $\theta(n\log n)$

c) $O(n)$ d) $\theta(n)$

Q. If all elemnts of array is identical.

what is TC of QS?

a) $\theta(n\log n)$ b) $\theta(n^2)$

c) $\theta(n)$ d) $O(n)$

$\Rightarrow$

	1	2	3	4	5	6	7
a	9	9	9	9	9	9	9

Piviot element is moved to the mid in first partion pass.

// $\theta(n\log n)$ or $\theta(n^2)$ depends on implementation of partition

[from our algo: $\theta(n\log n)$]

Q. $list_1$: 1, 2, 3, ... n

$list_2$: n, n-1, n-2, ... 1.

if C_1 & C_2 comp required to sort $list_1$ & $list_2$ in Ascending order using Quick sort Algo.

which is true ?.

a) $C_1 < C_2$ b) $C_1 > C_2$

✓c) $C_1 = C_2$ d) $C_1 = C_2 + n\log n$.

*** Q. Write algo to find k^{th} smallest element in array of n distinct elements.

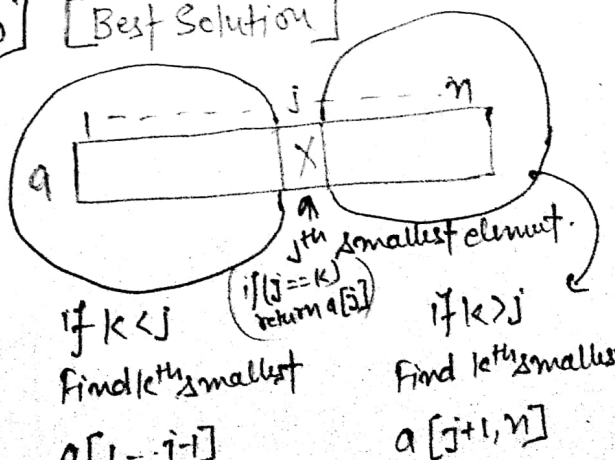
⇒ Method1 - 1) Sort array using sorting algo
2) Return a[k];

} $\theta(n\log n)$ All Cases

Method 2: - 1) Find Position of min from a[1] to a[n] ++
Swap with a[1]
2) Find position of min from a[2] to a[n]. ++
Swap with a[2]
⋮
k times.

TC
$\theta(k \cdot n)$
BC ⇒ $\theta(n)$
WC ⇒ $\theta(n^2)$

Method 3: - [Using Partition Algo] [Best Solution]



```
Algo Kth smallest ( P, q, K)
{
    if (P == q)
    { if (K == P)
        return(a[K])
      else
        return(false)
    }
    else
    {   j = Partition (a, P, q)
        if (K < j)
            Kth smallest (P, j-1, K)
        else if (K > j)
            Kth smallest (j+1, q, K)
        else return (a[j])
    }
}.
```

Time Complexity :—

Best case $\theta(n)$

One fun call to terminate algo if $K == j$ after one partition

Avg case :—

$$\left.\begin{aligned} T(n) &= T(n/5) + cn \\ T(n) &= T(4n/5) + cn \\ T(n) &= T(n/2) + cn \end{aligned}\right\} \theta(n)$$

Worst Case :—

$T(n) = T(n-c) + n = \theta(n^2)$.

every time partition algo divides c & $n-c$ for i/p array of network.

Max-Min Algo :—

$a[1 \ldots n]$ array of n elements find Max & Min element of array.

Straight Max Min Algo

```
Algo MaxMin(a, n)
{
    Max = Min = a[1];
    for(i=2; i<=n; i++)
    { if (a[i] > max)
        Max = a[i]
      else if (a[i] < min)
        Min = a[i]
    }
    return(Max, Min);
}
```

n-1 times

⇒ Min Comp require to find Max & Min :—

(n-1) Comp.

⇒ Max Comp required to Find Max & Min

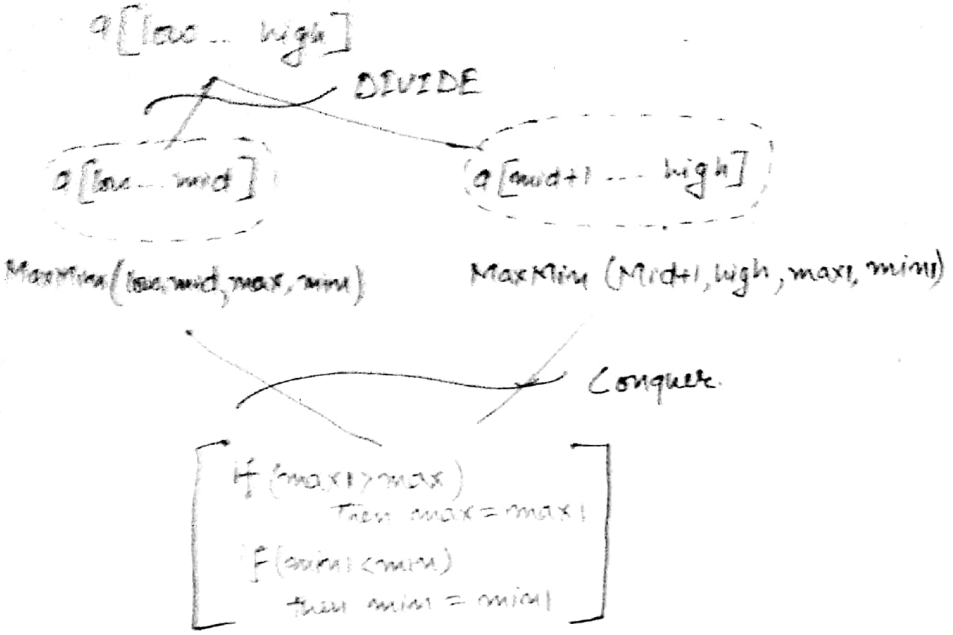
$2(n-1) = 2n-2$ Comp.

⇒ Avg Comp required to find Max & Min

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

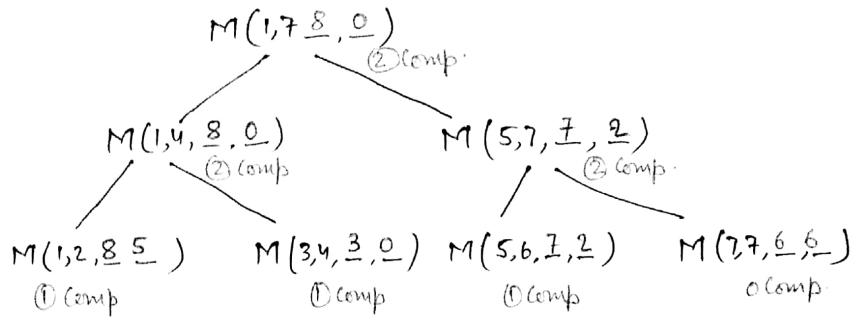
$$= \frac{3n}{2} - 1.5 \text{ Comp.}$$

'D and C' Max Min Algo:—




```
Algo MaxMin (low, high, max, min)
{  if (low == high)
      max = min = a[low];
   else if (low == high-1)
      {  if (a[low] ≤ a[high])
          {  max = a[high];
             min = a[low];
          }
         else { Max = a[low];
                Min = a[high]; }
      } else
      {  mid = (low+ high) /2
         Max-Min (low, mid, max, min)
         Max_Min (mid+1, high, max1, min1)
         if(max1 > max) max = max1;
         if (min1 < min) min = min1;
      }
      return(max, min)
}.
```

	1	2	3	4	5	6	7
a	8	5	3	0	2	7	6



'9' elements comp to find Max Min a[1...7] elements.

T(n): #f elemnits comp to find Max Min using D and C.

$$T(n) = \begin{cases} 0 & , n=1 \\ 1 & , n=2 \\ 2T(n/2)+2 & ; n>2 \end{cases}$$

$$T(n) = 2T(n/2)+2$$
$$= 2[2T(n/2^2)+2]+2$$
$$= 2^2T(n/2)+2^2+2$$
$$= 2^3T(n/2^3)+2^3+2^2+2$$
$$= 2^kT\left(\frac{n}{2^k}\right)+[2^k+\dots 2^2+2]$$

if $\frac{n}{2^k}=2$ (eliminating comp'n)

$[\because \frac{n}{2^k}=2 \;,\; 2^k=n/2]$

$$= \frac{n}{2}\cdot 1+2\left(\frac{n}{2}-1\right)$$
$$= \frac{3n}{2}-2 \text{ Comparisom}$$

Min Max Algo	Min Comp	Avg Comp	Max Comp
Straight Algo	$(n-1)$	$\frac{3n}{2}-1.5$	$2n-2$
D and C.	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$

Q Min element comp to find maximum of n array of element.

a) $n-1$ b) $2n-2$ c) $\frac{3n}{2}-2$ d) None.

Q Max element comp to find Max + Min of array of n elements.

a) $n-1$ b) $2n-2$ c) $\frac{3n}{2}-2$ d) None.

Q What is Time complexity required to return element from array of n distinct element which is neither max or not min element?

a) $\theta(\log n)$ b) $\theta(n)$ c) $\theta(n^2)$ d) $\theta(1)$

Integer Addition :-

X, Y are two n bit integers.

TC required to add X + Y: $\theta(n)$ [# of bit operation]

$$\begin{array}{rl} X \Rightarrow & X_{n-1}\ X_{n-2} \cdots X_1\ X_0 \\ + \quad Y \Rightarrow & Y_{n-1}\ Y_{n-2} \cdots Y_1\ Y_0 \\ \hline Z \Rightarrow & Z_n\ Z_{n-1}\ Z_{n-2} \quad Z_1\ Z_0 \end{array}$$

Integer Multiplication Algo :-

a) Straight Int Mul

b) D and C Int Mul

c) Strassen's D and C Int Mul

Straight Int Mul:—

X & Y are two n bit integers.

$X = 25 \quad Y = 5 \quad Z = 0$

1) If x is odd then $Z = Z + Y = 5$

$x = \lfloor \frac{X}{2} \rfloor = 12 \quad Y = 2Y = 10$

2) If x is even then No Add

$x = \lfloor \frac{X}{2} \rfloor = 6 \quad Y = 2Y = 20$

3) If x is even then No Add

$x = \lfloor \frac{X}{2} \rfloor = 3 \quad Y = 2Y = 40$

4) If x is odd then $Z = Z + Y = 45$

$x = \lfloor \frac{X}{2} \rfloor = 1 \quad y = 2Y = 80$

5) If x is odd then $Z = Z + y = 125$

$x = \lfloor \frac{X}{2} \rfloor = 0 \quad Y = 2Y = 160$

```
Algo Int Mul (X, Y)
{  //X, Y two n bit integers.
   Z = 0
   while (X > 0)                          O(1)
   {  if (X % 2 == 1)                     O(n)
      {  Z = Z + Y;  }
      X = ⌊X/2⌋ ; //Right Shift O(1)
      Y = 2Y; // Left Shift O(1)
   }  return (Z);
}
```

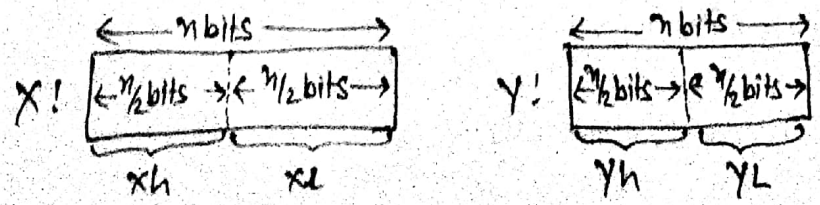
log x base 2 iterations

TC: $\Theta(n^2)$

#f bit required to represent int x is $\lceil \log_2 x \rceil$ bits.

#D and C Int Mul :—

X, Y are n bit num if $n > 1$ divide X & Y into two sub Int each with $n/2$ bits.



$X = X_h * 2^{n/2} + X_L \qquad Y = Y_h * 2^{n/2} + Y_L$

$X \cdot Y = (X_h \cdot 2^{n/2} + X_L)(Y_h \cdot 2^{n/2} + Y_L)$

$X \cdot Y = (X_h * Y_h \cdot 2^n + (X_h * Y_L + X_L * Y_h) \cdot 2^{n/2} + Y_L * Y_L$

2^n: n left shift; $2^{n/2}$: $n/2$ left shifts.

To multiply two n bit Int x & y

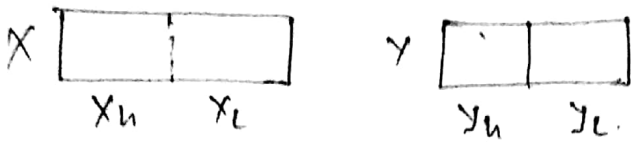
- 4 Sub Int Mul of $n/2$ bits.
- $3n/2$ left shift op.
- 3 Int Additions.

```
Algo IntMul(X, Y, n) ⇒ T(n)
{
  if (n ≤ 1) // Small(P)          } a
      return (X * Y)
  else
  { // Divide.
    Xh Xl are two sub int of X
    Yh Yl are two sub int of Y
    P = IntMul(Xh, Yh, n/2);      }
    Q = IntMul(Xh, Yl, n/2);      } 4T(n/2)
    R = IntMul(Xl, Yh, n/2);      }
    S = IntMul(Xl, Yl, n/2);      }
    // Conquer
    return(P·2^n + (Q+R)2^{n/2} + S) ⇒ Cn
  }
}
```

D and C Rec Rel for Int Mul

$$T(n) = \begin{cases} 4T(n/2) + Cn, & n > 1 \\ a, & n \le 1 \end{cases} \Rightarrow \theta(n^2)$$

c) Stresson's D and C :-



$X = X_h \cdot 2^{n/2} + X_L \qquad Y = Y_h \cdot 2^{n/2} + Y_L$

$$\begin{cases} P = X_h * Y_h \\ Q = X_L * Y_L \\ R = (X_h + X_L) * (Y_h + Y_L) \\ X \cdot Y = P \cdot 2^n + [R - P - Q]\, 2^{n/2} + Q \end{cases}$$

To multiply X & Y of n bits each

- 3 sub int Mul of $n/2$ bit each : $T(n)$
- $3n/2$ left shifts : $\theta(n)$
- 6 Int Add/Sub. : $\theta(n)$

```
Algo IntMut (X, Y, n)
{  if (n≤1) // Small (P)
       return (x * y)
   else
   { // Divide.
       Xh XL are two sub int of X
       Yh . YL are two sub int of Y
       P = IntMul (Xh, Yh, n/2);
       Q = IntMul (XL, YL, n/2);
       R = IntMul (Xh+XL, Yh+YL, n/2)
     // Conquer.
     return (P · 2^n + [R - P - Q] 2^(n/2) + Q)
   }
}
```

Stressen's D and C Int Mul

Recurrence Rel :-

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

$$= \theta(n^{\log_2^3})$$

$$\approx \theta(n^{1.69})$$

Matrix Multiplication Algo:—

a) Straigh Matrix Mul b) Dand C Matrix Mul c) Stressens Dand C Matrix Mul

⇒ Matrix Addition :-

$C_{n\times n} = A_{n\times n} + B_{n\times n}$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots \\ c_{21} & & \\ \vdots & & \end{bmatrix} = \downarrow i \begin{bmatrix} \overrightarrow{j} & & \\ a_{11} & a_{12} & \cdots \\ a_{21} & & \\ \vdots & & \end{bmatrix}_{n\times n} + \downarrow i \begin{bmatrix} \overrightarrow{j} & & \\ b_{11} & b_{12} & \cdots \\ b_{21} & & \\ \vdots & & \end{bmatrix}_{n\times n}$$

```
for(i=1; i≤n; i++)
{  for(j=1; j≤n; j++)
     c[i,j] = A[i,j] + B[i,j]
}
```

a) Straight Matrix Multiplication :-

$c_{11} = \underbrace{a_{11} * b_{11} + a_{12} * b_{21} + \cdots + a_{1n} * b_{n1}}$

n elemnts Multiplications for each elemnt of result Matrix.

TC of Straight Matrix Mul : $O(n^3)$

```
for(i=1; i≤n; i++)
{ for(j=1; j≤n; j++)
  { c[i,j]=0;
    for(k=1; k≤n; k++)
      cij = cij + aik * bkj
} }
```

Q. Apxq Bqxr are two matrices. How many elemnt Mul required to Multiply Matrices A&B? ⇒ P·Q·R [#f elemnt Mul / cost of Matrix Mul]

c) Stressen's Matrix Mul :—

Divide each Sub Matrix into 4 Sub Matrices.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

To multiply matrices of $n \times n$

{ 7 sub matrix mul of $n/2 \times n/2$
{ 18 matrix mul

Stressen's Matrix Mul Recurrence Relation

$$T(n) = \begin{cases} 7T(n/2) + cn^2 & , n > 2 \\ a & , n \le 2 \end{cases}$$

$TC = \theta(n^3)$

$P = (A_{11} + B_{12}) * (B_{11} + B_{22})$

$Q = (A_{21} + B_{22}) * B_{11}$

$R = A_{11} * (A_{12} - A_{22})$

$S = A_{22} * (B_{21} - B_{11})$

$T = (A_{11} + A_{12}) * B_{22}$

$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$V = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P + S - T + V$ | $C_{21} = Q + S$

$C_{12} = R + T$ | $C_{22} = P + R - Q + U$

28/08/17

⇒ Graph & Tree representation

⇒ Priority Queues [Mean heap / Max heap] *** } DS Topic

⇒ Set Algorithms [Union / Find Algo]

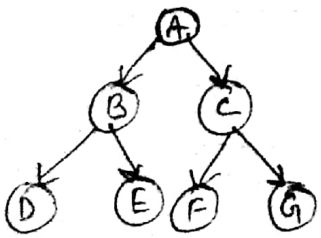
Tree

⇒ Single rooted [Starting point]
[Every tree operation starts from Root]

⇒ Always Connected

⇒ Always Acyclic

⇒ Always Directed
[Tree with n-vertices must be exactly (n-1) edge]



Graph

⇒ Any vertex of graph can be used as starting point.

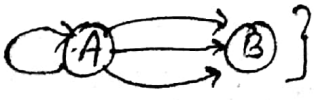
⇒ May / May not be connected.

⇒ May / May not Acyclic

⇒ May / May not Directed
[Simple Graph with n vertices can be atleast 0 edge [null graph]]
at most $\frac{n(n-1)}{2}$ Edges [Complete Connected graph.]

Simple Graph

No Self loop, No parallel Edges.



} Not Simple Graph

Q. How many simple undirected Graph possible with n vertices?

$= 2^{\frac{n(n-1)}{2}}$ # of graphs. Where $\frac{n(n-1)}{2}$ is max no of edge can be present within a graph of n vertices.

Tree representation :-

a] Using Array b] Using Linked List

a] Array Representation to store Binary tree :-

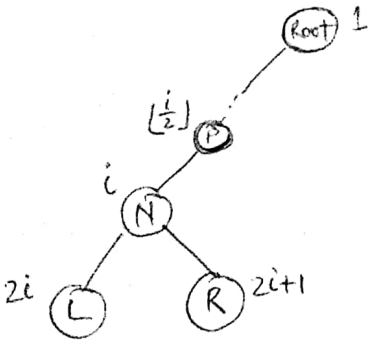
Root Node Index : 1.

If Node N array index i then,

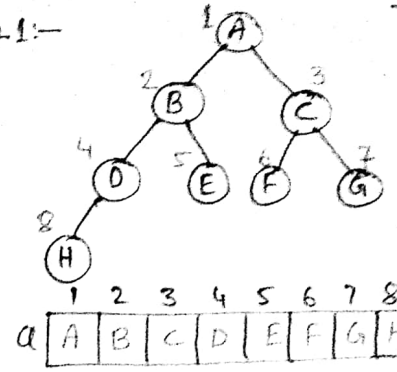
Left child of N is 2i index

Right child of N is 2i+1 index

Parent of N is $[i/2]$ index.

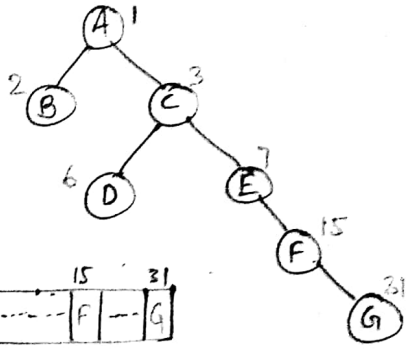


Eg 1:-



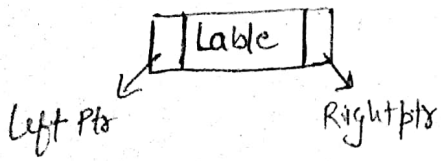
If nodes indexes contains integers from 1 to n, then array representation Prefered.
Coz no overhead of ptrs.

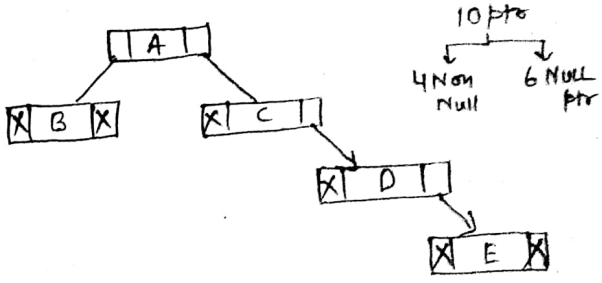
Eg: 2→



Note → Array representation not preferred because of wastage of Mem. Space.

b] Linked List Representation to store Binary Tree :-



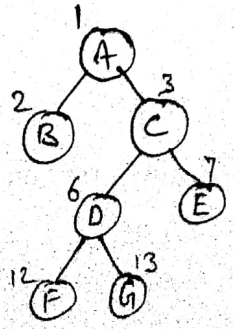


To store Binary Tree of n nodes : 2n ptrs required.

[N+1 Null ptrs] [N-1 Non Null ptrs]

1] Strict Binary Tree → Every node of tree must be either 0 or 2 childs.

Linked list representation Preffered to store Binary Tree.



Q Complete k-ary, every internal node exactly k childs. How many "#f leaf nodes" with "n internal nodes?

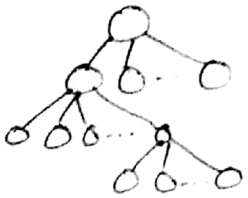
a) nk b) (n-1)k+1 c) n(k-1)+1 d) n(k-1)

Soln

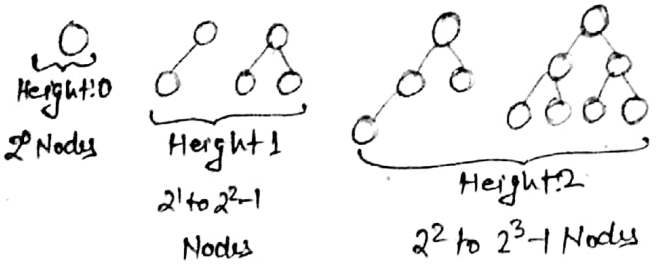
Given: n: #f internal nodes

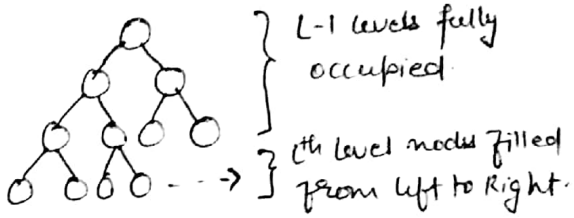
Find: #f leaf nodes

n	#f leaf nodes
0	1
1	K
2	2k-1
3	3k-2
⋮	
n	n(k-1)+1.



2] Complete Binary Tree :— Complete Binary Tree of L levels must be (L-1) levels fully occupied L^{th} level nodes can be add left to Right.





⇒ Complete Binary Tree of Height h $[2^h$ to $2^{h+1}-1]$ Nodes.

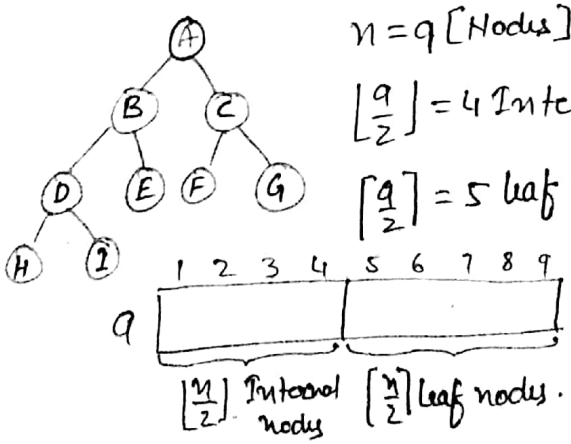
⇒ Array Representation preffered to store complete Binary Tree

⇒ CBT of n nodes have $\lfloor \frac{n}{2} \rfloor$ #f internal nodes, $\lceil \frac{n}{2} \rceil$ #f leaf nodes.

Height of CBT or FBT for n nodes: $\theta(\log_2 n)$

$n = 2^h \Rightarrow \boxed{h = \log_2 n}$ when n is min.

$n = 2^{h+1} - 1 \Rightarrow h = \boxed{\log_2(n+1) - 1}$ when n is max.

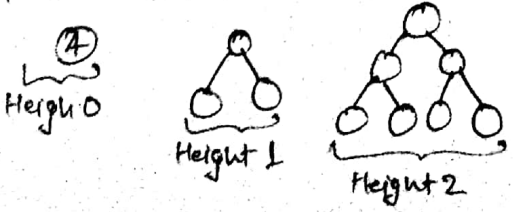


n = 9 [Nodes]

$\lfloor \frac{9}{2} \rfloor$ = 4 Internal Nodes

$\lceil \frac{9}{2} \rceil$ = 5 leaf Nodes

3] Full Binary Tree :— Every node 0 or 2 childs 44 all leaf nodes must be same level.



#f nodes in FBT of Height h $(2^{h+1}-1)$ nodes.

***

⇒ Priority Queue DS :—

Data Structure which should support efficient way to perform

- Repeated min/max element deletion
- Repeated Insertion's.
- May required to store duplicate elements.

Max Val : High Priority

⇒ Repeated Max Deletion
⇒ Repeated Insertion
⇒ Should Able to store duplicate values.

Min Val : High Priority

⇒ Repeated Min deletion
⇒ Repeated Insertion
⇒ Should support to store duplicate val.

Algo	Sorted array	Unsorted array	Sorted Linked list	Unsorted Linked list	Balanced Binary Search Tree	Min heap
1. Build list for n elements [Assume elements should Insert one after other ++ May not distinct elements]	1+2+3+...n-1 comp++ shift $= \theta(n^2)$	$n * \theta(1) = \theta(1)$	1+2+3+... ..n-1 comp $= \theta(n^2)$	$n * \theta(1)$ ↑ Insertion $= \theta(n)$	$\theta(n \log n)$	$n * \theta(\log n) = \theta(n \log n)$
2. Repeat Min del over n times.	$n * \theta(1) = \theta(n)$	$(n-1)+(n-2)$ ----+2+1 $= \theta(n^2)$	$n * \theta(1) = \theta(1)$	(comp) $n+n-1+\dots-1$ $= \theta(n^2)$	$\theta(n \log n)$	$n * \theta(\log n) = \theta(n \log n)$
	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$	$\theta(n \log n)$	$\theta(n \log n)$

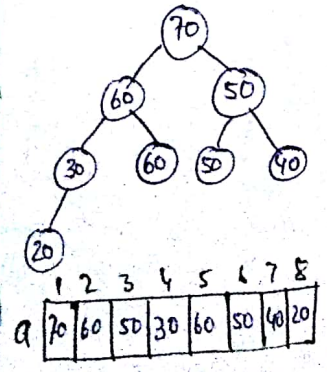
⇒ Balanced Binary Search tree : not suitable for duplicate elements and uses linked list representation.

⇒ Min heap used to maintain duplicate elements
No overhead of ptrs (array representation is used.

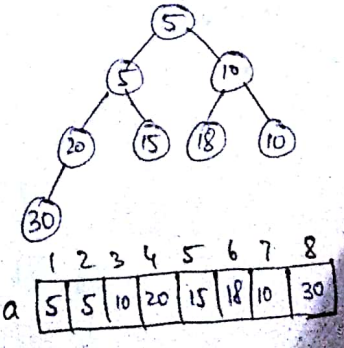
Note :— Balanced Binary Search tree best DS for
1) Search element 2) Insert / Delete 3) Discard duplicate while insertion.

⇒ **Max (Min) Heap :—** Complete Binary tree with every parent must be max/min than all left + Right Sub tree Nodes.

Max Heap

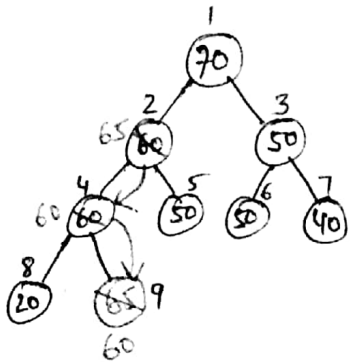


Min Heap



Insertion into Max heap:—

a[1...n+1] array elements already in Max heap ↑↑ Insert element into Max heap.




```
Algo InsertMaxheap (a, n, x)
{ i=n;
  item=x;
  while (i≥1 && a[i/2]<item)
  { a[i] = a[i/2];
    i = i/2;
  }
  a[i]= item;
}
```

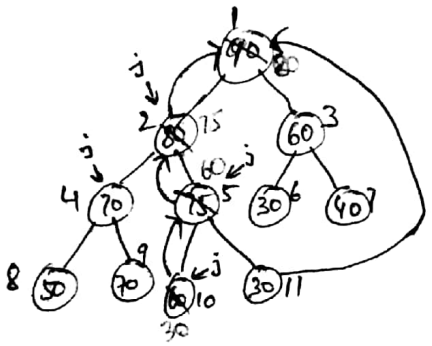
Comp ↑↑ shift op: Insert element into max heap: $\theta(\log n)$

($\log n$: Height of max heap)

TC to insert into Max heap: $\theta(\log n)$ in Worst & Avg Case

$\theta(1)$ in Best Case.

Delete Max from Max heap:—



item = 30

j = ~~2~~ 4 ~~8~~ 16 20

n = ~~21~~ 20

∵ max ele = a[1]

a[1] = a[n]

Adjust (a, i, n-1)

[Assumption to use adjust fun]

```
Algo DelMaxheap(a,n)
{ MaxEle = a[1];
  a[1] = a[n];
  Adjust (a,1,n-1);
  return (MaxEle);
}
Algo Adjust(a, i, n)
{ j=2i ; item = a[i]
  while (j≤n)
  { if(j<n && a[j] < a[j+1])
    { j = j+1; }
    if(a[j] ≤ item) break;
    a[j/2] = a[j];
    j = 2j
  }
  a[j/2] =item;
}
```

TC to adjust $\theta(\log n)$ in WC/AC

$\theta(1)$ in BC

TC to delete max from heap of n elements:

$\theta(\log n)$ AC/WC

$\theta(1)$ BC

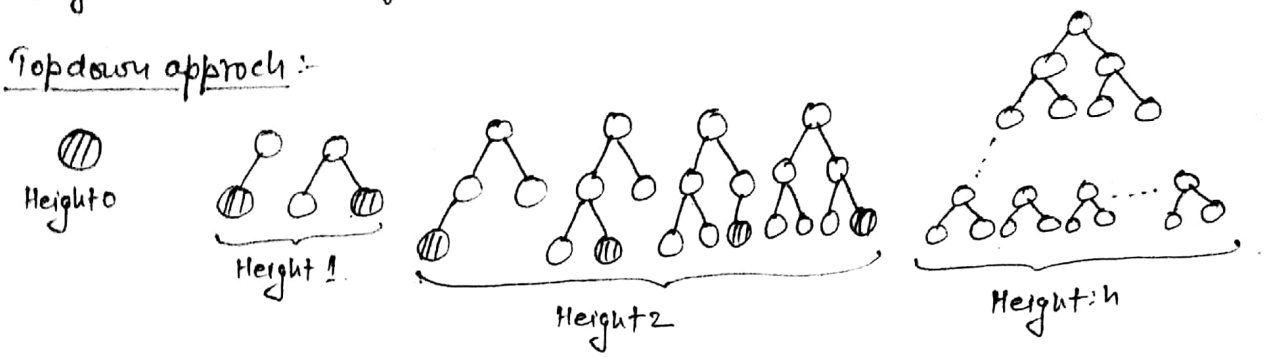
29/08/17

⇒ Building of Max Heap:-

1] Top down approch : TC: $\theta(n \log n)$
To build max heap if elements [keys] should enter one after other

2] Bottom up approch : TC: $\theta(n)$
To build max heap if all elements are available

1) Top down approch :-



Cost to Build Max heap in Top down Approch :-

$$T(h) = 2*1 + 2^2*2 + 2^3*3 + \ldots + 2^h * h$$

$$2T(h) = 2^2 \cdot 1 + 2^3 \cdot 2 + 2^4 \cdot 3 + \ldots + 2^{h+1} \cdot h$$

$$2T(h) - T(h) \equiv -[2 + 2^2 + 2^3 + 2^4 + \ldots + 2^h] + 2^{h+1} \cdot h$$

$$T(h) = 2^{h+1} \cdot h - [2(2^h - 1)]$$

$$\boxed{T(h) \equiv 2^{h+1} \cdot h - [2^{h+1} - 1] + 1}$$

$$T(n) \equiv (n+1)\left[\log_2(n+1) - 1\right] - n + 1$$

$$\boxed{T(n) = \theta(n \cdot \log_2 n)}$$

Height	# Nodes @ given height	Cost per Insertion [Comp + Swap]
0	1	0
1	2	1
2	2^2	2
3	2^3	3
h	2^h	h.

$n = 2^{h+1} - 1$ // n elements

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

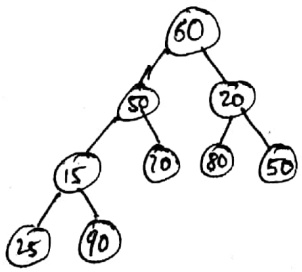
$h = \log_2(n+1) - 1$

2) Bottom up approch :-

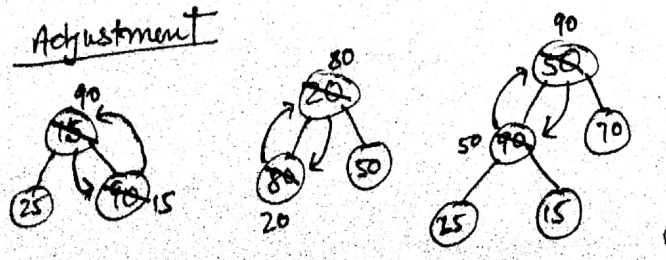
Eg → a

1	2	3	4	5	6	7	8	9
60	50	20	15	70	80	50	25	90

Internal nodes | Leaf Nodes.



Adjustment



1	2	3	4	5	6	7	8	9
90	70	80	50	60	20	50	25	15

a → Max heap Array

$T(h)$ = TC to build Max heap in bottom up approch of height (h) [Sum of heights of all nodes in CBT of height h]

$$T(h) = 1*h + 2*(h-1) + 2^2*(h-2) + \ldots + 2^{h-1}*1.$$

$$2T(h) = 2\cdot h + 2^2(h-1) + 2^3(h-2) + \ldots + 2^h \cdot 1.$$

$$2T(h) - T(h) = -h + [2 + 2^2 + 2^3 + \ldots + 2^{h-1} + 2^h]$$

$$T(h) \equiv 2(2^h - 1) - h$$

$$\boxed{T(h) \equiv (2^{h+1} - 1) - (h+1)}$$

$$\boxed{T(h) \equiv n - \log_2(n+1)} \equiv \boxed{\theta(n)}$$

Height	#of Nodes of given height	Cost for a node to adjust [at most]
h	1	h
$h-1$	2	$h-1$
$h-2$	2^2	$h-2$
⋮		
1	2^{h-1}	1
0	2^h	0

$n = (2^{h+1} - 1)$ // #of nodes in Max heap

$h+1 = \log_2(n+1)$

Algo to build Max heap in Top-down Apr.

```
Algo Max_heapTD (a,n)
{
  for (i=2; i<=n; i++)
  {
    Insert Maxheap (a, i, a[i]);
  }
}
```

TC of algo : $O(n \log n)$

Algo to build Max heap in Bottom up Apr.

```
Algo Max_heap BU (a,n)
{
  for(i=⌊n/2⌋; i>=1; i--)
  {
    Adjust(a, i, n);
  }
}
```

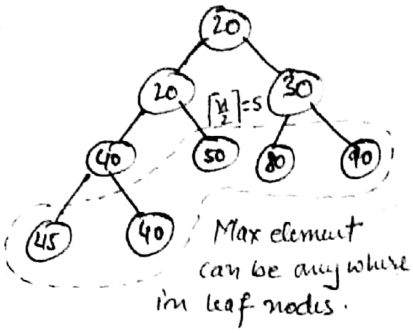
TC of algo : $\theta(n)$

Heap operations :—

1] TC to find Min ele in Min heap of n elements : $\theta(1)$

2] TC to delete Min ele from Min heap of n elements : $\theta(\log n)$

3] TC to find Max element in Min heap of n element : $\theta(n)$



Algo

```
Max = a[⌈n/2⌉]
for (i = ⌈n/2⌉ + 1; i ≤ n; i++)
{ if (a[i] > max)
    max = a[i]
}.
```

4] Del Max from minheap of n elements: $\theta(n)$

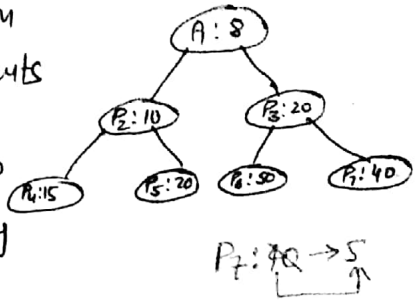
⇒ Find Max pos in Min heap } $\theta(n)$

⇒ Del Max } $\theta(\log n)$

$\theta(n)$

5] Key decrement operation from Min heap of n elements

[Key decrement of min heap uses for increase in priority of some process]



6] Key increment operation TC in min heap of n elements : $\theta(\log n)$

⇒ Use Adjust fun.

7] Find K^{th} smallest element in mean heap of n distinct elements: ——

Method 1 :- 1) Del Min from Min heap (K−1) times.

2) Root is K^{th} Min

Method 2 :- 1^{st} Min a[1] 1

2^{nd} Min {a[2], a[3]} 2

3^{rd} Min {a[4], a[5], a[3]} 2

4^{th} Min {a[8], a[9], a[5], a[3]} 3

⋮

K^{th} Min {K elements } K-1

Find K^{th} min of Min Heap	if K constant	if K in terms of n {$K = O(n)$}.
Method 1 $\theta(K \cdot \log n)$	⇒ $\theta(\log n)$	$\boxed{\theta(n \log n)}$
Method 2 $\theta(K^2)$	⇒ $\boxed{\theta(1)}$	$\theta(n^2)$

Q. What is TC to find 7^{th} min in Min heap of n ele

a) $\theta(\log n)$ (b) $\theta(n \log n)$

c) $\theta(1)$ d) $\theta(n^2)$

Q. What is TC to find $\frac{n^{th}}{10}$ min in min heap of n distinct elements?

a) $\theta(\log n)$ b) $\theta(n \log n)$

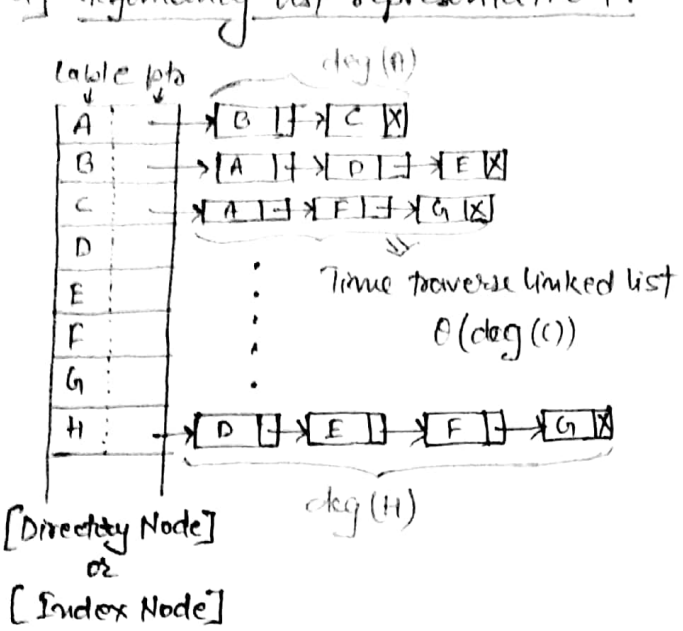
c) $\theta(1)$ d) $\theta(n^2)$

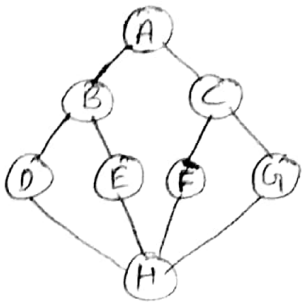
Graph Representation

a] Adjacency list representation [Using linked lists]

b] Adjacency Matrix Representation [Using 2D Array]

a] Adjacency list representation :-





Using Adjacency list Representation of Graph :-

1) Tc to find Adj of given vertex (v)

$Adj(v) = \theta(deg(v))$

{SL(1) to O(n)}.

2) Tc to find Adj of all vertices of Graph.

$Adj(v_1) + Adj(v_2) + \ldots + Adj(v_n) =$

$deg(v_1) + deg(v_2) + \ldots + deg(v_n) = 2 \cdot e$

$= \theta(n+e)$

```
for(i=1; i≤n; i++)
{ Find Adj (Vi) }
```

Tc = θ(n+e)

[SL(n) to O(n²)]

SL(n): Graph is free

O(n²): Graph is complete connected graph.

3) Sc to store graph with n vertices & e edges

$= \theta(n+e)$

{SL(n) to O(n²)}

n: Space for directory node.

2*e: Space for all linked list.

3) SC to store graph with n vertices
& e edges : $\theta(n^2)$

Sparse graph :- Graph with n vertices and $O(n)$ edges [less possible edges]
Eg→ Tree,

Dense graph :- Graph with n vertices $O(n^2)$ edges (Max^m possible edges)
Eg→ Complete connected graph.

* Adjacency list representation prefers to store sparse graph.
* Adjacency Matrix representation prefers to store Dense graph.
(Because no overhead of pointers)
* Adjacency Matrix representation takes less time & space complexity.

Greedy Algorithm :-

⇒ Uses to solve optimization problems.
⇒ optimization problems :-
Maximization / Minimization of result of solution.
⇒ Usually #f feasible solution to solve optimization problem with n i/p.
⇒ 2^n feasible Solution [Subset problem] eg→ 0/1 KnapSack Problem.
⇒ n! feasible Solutions [Permutation problem] eg→ Travelling Salesman Problem.

⇒ Greedy Algo Solves optimization Problem by using Predefined strategy [Static Approach].
Greedy Solution choose one feasible sol from 2^n or n! feasible Solutions and return cost of solution.
⇒ Greedy Algo Always Run in Polynomial Time Comp.
⇒ Every optimization problem may not possible to solve using Greedy Approach.
[Some optimization problem may not result optimal value using greedy algo]

* **Coins change Problem :-**

Domination of coins :-
{50, 25, 20, 10, 5, 2, 1} Rs

How many min coins to denominate Rs 143?

Greedy Soln :-
Max^m domination coin place first :-
1st dec : 2×50 = 100
2nd dec : 1×25 = 25
3rd dec : 0×20 = 0
4th dec 1×10 = 10
5th dec 1×5 = 5
6th dec 1×2 = 2
7th dec 1×1 = 1
Total = 7 Coins.

Dynamic Soln:—

Compute all possible Sol and return optimal.

$2 \times 50 = 100$
$0 \times 25 = 0$
$2 \times 20 = 40$
$0 \times 10 = 0$
$0 \times 5 = 0$

$1 \times 2 = 2$
$1 \times 1 = 1$

Total 6 coins.

*** Single Source shortest Path algo:— 30/08/17

1) Dijkstras Algo (Greedy Algo)
2) Bellman Ford Algo (Not Greedy Algo)

} Imp: For exam

1) Dijkstras Algo:—

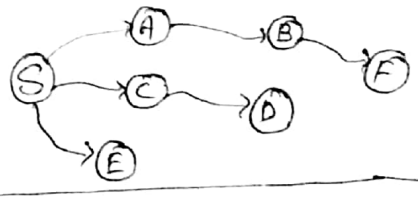
Graph (n,e) n: #f vertices
e: #f edges

Cost [1...n, 1...n] // Edge costs

$$Cost(i,j) = \begin{cases} \text{Edge cost} & \text{if } (i,j) \in \text{edge} \\ \infty & \text{if } (i,j) \notin \text{edge} \end{cases}$$

S: Source Vertex

Compute shortest path from Source (S) vertex to every vertex of graph.
Determine dist [1...n] array s.t
dist [i] shortest path cost from S to i



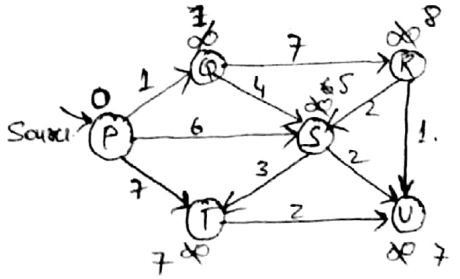
Dijkastaras Procedure

// Initialization

1. dist [1...n] = $+\infty$
   Prev [1...n] = -1
   dist [s] = 0
   list of all vertices which are not completed Relaxation // all vertices in list

// Relaxation Procedure

2. Choose vertex (u) from list which is Min dist & not relaxed before
   V is set Adj of vertex (u)
   for all v set
   { if (dist [u] + cost (u,v) < dist [v])
   { dist [v] = dist [u] + cost [u,v]
   Prev [v] = u
   }
   } Repeat ② for all vertices.



Dist

P	Q	R	S	T	U
0	~~∞~~ 1	~~∞~~ 8	~~∞~~ 5	~~∞~~ 7	~~∞~~ 7

Prev

P	Q	R	S	T	U
-1	~~N~~ P	~~N~~ Q	~~N~~ R	~~N~~ P	~~N~~ S

list

~~R~~ Q R	P, Q, S, T, U, R
S T U	

empty.

Order of vertices completed ⇒ Dijkstras Spanning tree:-

Relaxation: { P Q S T U R
P Q S U T R

Algo Dijkstras (Graph(G), n, e, cost[][], s)

```
{
    // Initialization
    for(i = 1; i ≤ n; i++)
    {
        dist[vi] = +∞;
        Prev[vi] = -1;
    }
    dist[s] = 0
    build MinHeap(Q) for all vertices based
    dist of vertex is Priority.
    // Apply edge Relaxation for each vertex
    while(Q is not empty)
    { del vertex (u) whose dist is Min^n [θ(log n)]
      Find Adj of vertex (u) are v set [TC: deg(v) or O(n)]
      for(each vertex of v set)
      {
          if(dist[u] + cost(u,v) < dist[v])
          { dist[v] = dist[u] + cost(u,v)
            Prev[v] = u
            Key decremnt(Q, v, dist[v]); θ(log n)
          }
      }
    }
    return(dist[], prev[])
```

⇒ TC of Dijkstras Algo

a] Assume Graph G is Adj list Representation & & Min heap used:

$$\theta(n) + \theta(n) + n[\log n + \deg(u) + \deg(u)\cdot\log n]$$

(Initialization of dist[] · build min heap · del Min · Find Adj of u · key dec)

$$\theta(n) + \theta(n) + n\log n + \sum_{i=1}^{n} \deg(u) + \sum_{i=1}^{n} \deg(u)\log n$$

$$\theta(n) + \theta(n) + n\log n + e + e\cdot\log n$$

$$= \boxed{\theta((n+e)\cdot\log n)}$$

b] Assume graph G is Adj Matrix representation and Min heap used for priority Queues.

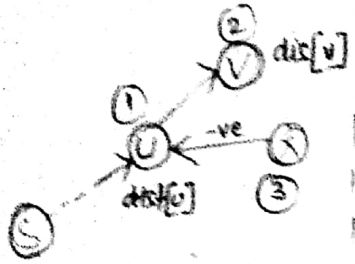
$$\theta(n) + \theta(n) + n[\log n + \theta(n) + \deg(u)\cdot\log n]$$

$$\theta(n) + \theta(n) + n\log n + \theta(n^2) + \theta(n)\cdot\log n$$

$$= \boxed{\theta(n^2 + e\log n)}$$

⇒ Limitations of Dykastras Algo:-

1) Dykstras algo may fail to find shortest path from source to other vertices if graph consist -ve edges. [No -ve Cycle]



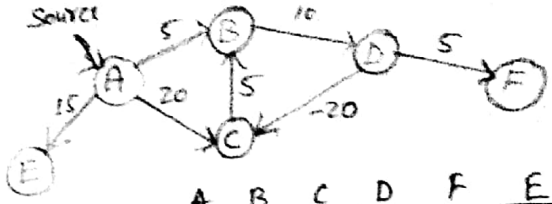
① Vertex U completed Relaxation

② dist[v] depends on dist[u].

③ because of relaxation of vertex(x) dist[U] decreases.

[then only dijastras fails]

2) Dykstras Algo fail to detect -ve cycle if -ve cycle reachable from source.



	A	B	C	D	F	E
dist	0	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	5	20	15	20	15	
	0	-5				

Wrong dist

Not possible to compute shortest path from A to BCDF (Cycle detection required).

---

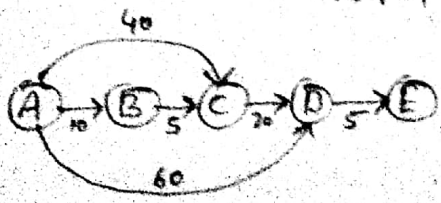
2) Bellman Ford Algo:-

⇒ Computes shortest path cost from source correctly even -ve Edge Exist.
[Assume No Negative Cycle]

⇒ Detects -ve cycle if -ve cycle is reachable from source.

If no -ve cycle exist then shortest path from Source (S) to vertex (A) should consist at most (n-1) edges.
[n: vertices]

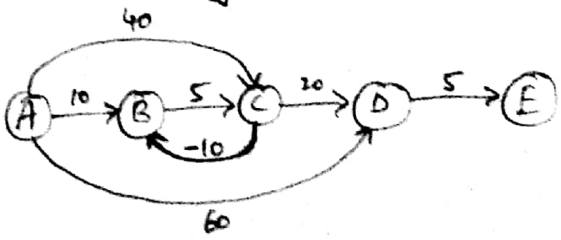
← Max (n-1) edge in shortest path →



n=5

A to E ⇒ [4 edges b/w shortest path from A to E]

If dist[A] from source with (n-1) edges is more than dist[A] from source with more than [n-1] edges. Then vertex 'A' is reaching from source through -ve cycle.



[A-B-C-D-E] 40 [n-1] edges.

[A-B-C-B-C-D-E]: 35 [more than n-1 edges]

Procedure of BellManford Algo:-

1) Initialize.

dist[vi] = ∞

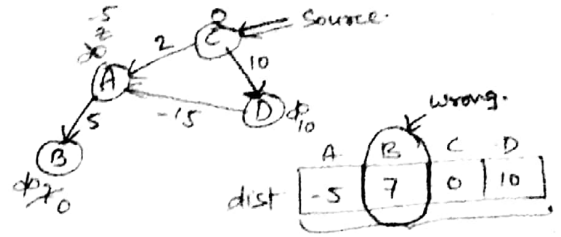
Prev[vi] = -1 for all vertices.

dist[s] = 0

2) Apply edge relaxation for all edges of graph 4 4 [Repeat (n-1) times]

3) Cycle detection: Apply edge relaxation nth time:- if any vertex dist[v] reduces then negative cycle going through source.

Eg1)



Bellmanford Algo:-

	A	B	C	D
dist	∞	∞	∞	∞

edges
1. C-A
2. C-D
3. A-B
4. D-A

n-1 times

	A	B	C	D
i=1	-5	7	0	10
i=2	-5	0	0	10
i=3	-5	0	0	10

} dist result after (n-1) edge relaxation

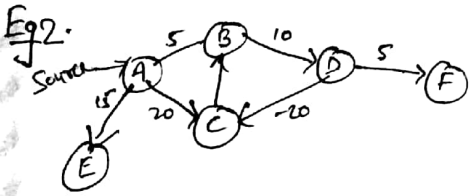
nth time

	A	B	C	D
i=4	-5	0	0	10

} nth time edge relaxation.

← correct dist val →
no -ve cycle.

Eg2.



	A	B	C	D	E	F
dist	0	0	-5	15	15	20

Result of dijkstras.

Bellman Ford:

	A	B	C	D	E	F
dist	0	∞	∞	∞	∞	∞

n-1 times

	A	B	C	D	E	F
i=1	0	0	-5	15	15	20
i=2	0	-5	-10	10	15	15
i=3	0	-10	-15	5	15	10
i=4	0	-15	-20	0	15	5
i=5	0	-20	-25	-5	15	0

nth time

	A	B	C	D	E	F
i=6	0	-25	-30	-10	15	-5

BCDF going through -ve cycle. from source.

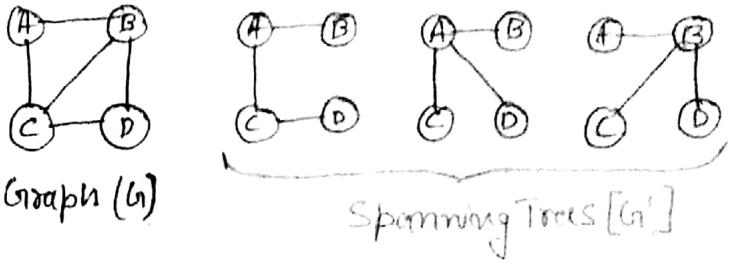
```
Algo Bellman ford (n,e, cost[][], S, E[][])
{
  // Initialization
  for(i=1; i≤n; i++)
  { dist[vi] = +∞;
    Prev[vi] = -1;
  } dist[s] = 0;
// Apply edge relaxation for each edge of graph
 4 4 repeat (n-1) times.
  for(i=1; i≤n-1; i++)
  { for ( All edges of graph (u,v))
    { if (dist[u]+cost[u,v] < dist[v])
      { dist[v] = dist[u] + cost[u,v]
        Prev[v] = u;
    } }
  } // Testing of -ve cycle reachable from source.
 for( All edges of Graph(u,v))
 { if dist[u] + cost[u,v] < dist[v])
   { // Negative cycle exist
     return false;
   }
 } return (dist[], Prev[])
   // No negative Cycle.
}.
```

TC of Bellman Ford Algo $= \theta(n \cdot e)$

⇒ W.C TC of Bellman ford Algo: $= \theta(n^3)$

Minimal Spanning tree :— 31/08/17

Spanning tree :— Spanning tree of graph ($G(V,e)$) is G' such that $G'(V,e')$ where $e' \subseteq e$ && G' is connected && No cycle.



Graph (G) — Spanning Trees [G']

⇒ n^{n-2} Spanning Trees possible in complete connected graph with n vertices.

complete connected graph G_4 [4 vertices] ⇒ #f possible ST's $4^{4-2} = 16$

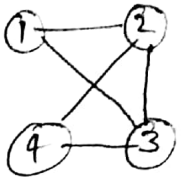
Kirchoff theorem :—

[To find #f ST's of graph]

1) Represent graph G in Adj Matrix formate
2) Multiply (-1) to Adj Matrix
3) Diagnal element replace
4) degree of Adj(i,i) = deg(i)
5) Co-factor of any diagonal element is #f ST's of graph.

Eg ⇒

$$Adj = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix} \end{array}$$

Cofactor of $Adj(1,1) = (-1)^{1+1} \det \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{bmatrix}$

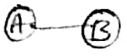
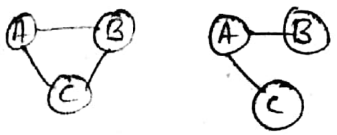
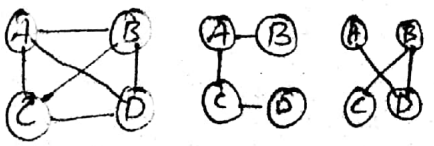
$\{3(6-1) - (-1)[-2-1] + (-1)[1+3]\}$

$= 8$ Spanning tree of the graph.

GATE.

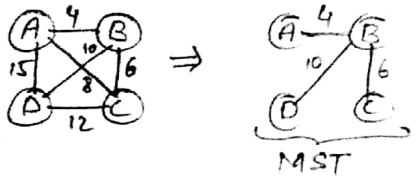
How many edge disjoint ST's possible in complete connected graph of n vertices?

$\frac{n(n-1)}{2}$ edge $\frac{\left[\frac{n(n-1)}{2}\right]}{n-1} = \left[\frac{n}{2}\right]$ Edge disjoint ST's

Complete connected graph	#f Edge disjoint ST's
	1
	1.
	2.

Minimal Spanning Tree –

Sum of edge of ST of graph is minimum.



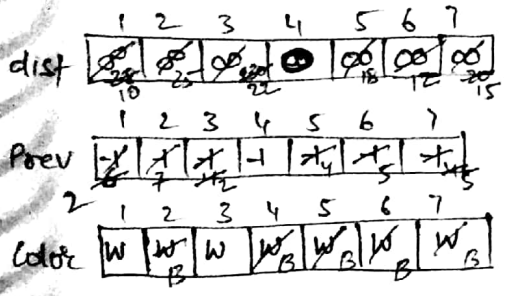
1) Prims algo
2) Kruskals Algo

} Greedy Method Algorithms.

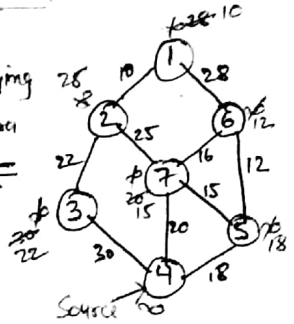
Prims Algo:–
- MST constructs Edge by Edge
- Set Edges which are included in MST must be always connected.

{ } 'X' is set of edges included into MST so far which must be connected.

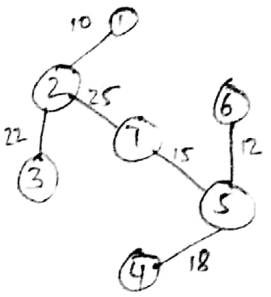
(u,v) next edge of MST such that { X ∪ (u,v) } must be connected & + mm Inc in Sum of Edge costs.



Applying dijkstra ⇐



Applying Prims. ⇒

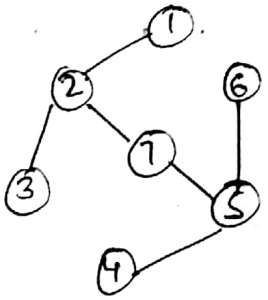


Color[vi] = { W, if vi not included into MST; B, if vi included into MST }

MST Cost: [18+12+15+25+10+22] = 102

Minheap (Q): All vertices based on dist[v] is priority.

dijkstra result ⇒



MST: using Prev[

Cost MST: Using dist[

List:-

1	3	4
5	6	7

```
Algo Prims (Graph, n, c, cost[][])
{  for (i=1; i≤n; i++)
   {  dist[vi] = ∞;
      Prev[vi] = -1;
      Color[vi] = W;
   } choose one vertex as source
   dist[s] = 0
   Build Minheap(Q) for all vertices
    based on dist[vertex]
   while (Q. Not empty)
   {  del vertex (U)
      Color[U] = B // U included in MST
      V is Adj of U
      for (all vertices (v) adj of U)
      {  if (dist[v] > cost(u,v) + + Color[v] == '
         {  dist[v] = cost(u,v)
            Prev[v] = U
            Keydec (Q, v, dist[v])
         }}
   } return (Prev[], dist[])
}
```

TC of Prims Algo.

a) Assume Adj list Graph ++ Priority Queue Min heap

$O((n+e)\cdot \log n)$

b) Assume adj matrix ++ priority Queue minheap

$O(n^2 + e\log n)$

c) Assume Adj list graph ++ Priority Queue unsorted array

$O(n^2)$

Set Algo :-

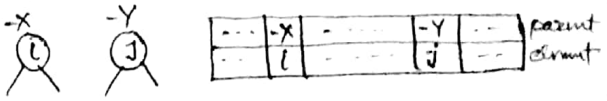
Used in Kruskals algo to detect cycle in MST construction]

Representation of sets [disjoint set]

$S_1\{1,3,5\}$ $S_2\{2,4\}$ $S_3\{8,9\}$ $S_4\{15,6,10\}$

P[i]	-3	-2	1	2	1	-3	9	-2	6	6
i	1	2	3	4	5	6	8	9	10	15

⇒ Union Algo :-



if set with Root i have more element than set with Root j

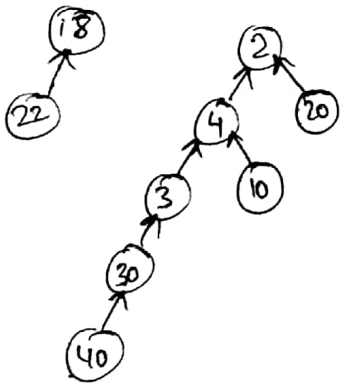
then (i) become parent of (j).

```
Algo Union (i, j)
{  z = P[i] + P[j]
   if ( P[i] ≤ P[j])
   {  P[j] = i;
      P[i] = z;
   }
   else { P[i] = j; P[j] = z;}
}
```

TC of Union Algo: $\boxed{\theta(1)}$

⇒ Find Algo :-

Find[i] : should root set tree of element i present.



P[i]	18	4	2	4	2	3	30	-2	18
i	2	3	4	10	20	30	40	18	22

Find (22) = 18 Find (2) = 2

Find (10) = 2

Find (40) = 2

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

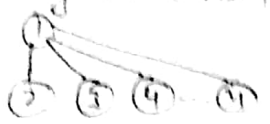
TC of Find operation

O (Height of Set tree)

⇒ Initially n sets having each set one element.

P[i] | 1 | 2 | 3 | 4 | 5 | ... | n-1 | n |

a) "Min Height" of set tree if n sets merges into one set using Union(i,j)



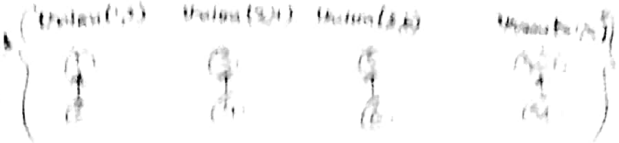
Union(1,2)
Union(1,3)
Union(1,4)
Union(1,5)

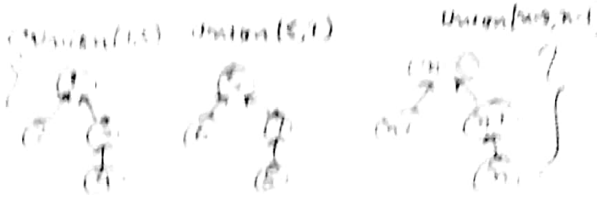
b) "Max height" of set tree if n sets merges into one set using Union(i,j)

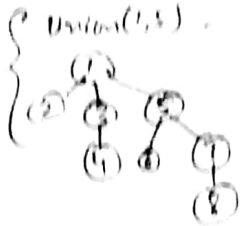
⇒ $n/2^k$ Sets each set 2^k elements

Height : K : $\Theta(\log_2 n)$

$n/2^k = 1$ $K = \log_2 n$. Worst Case TC : $O(\log n)$







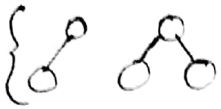
Union(n-2,n-3)

n/2³ sets each set 2³ elements

Kruskals Algo:

- Constructs MST edge by edge
- Let X be set of edges included into MST so far.

These X edges may not be connected

$X = \{$  $\}$

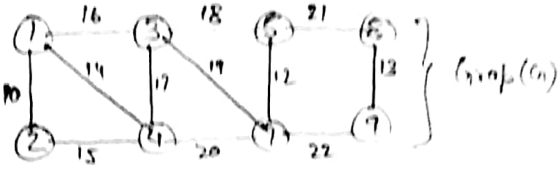
(U,V) next edge into MST s.t $\{X \cup (U,V)\}$ should not form cycle

Min Increment in Sum of edge cost.

Procedure :

n: #of vertices e: #of edges of graph G.

1) Del Min cost edge (U,V) from G.
2) Add (U,V) into MST if (U,V) not forms cycle in MST
3) Repeat ① ② until MST becomes (n-1) edge or Graph G becomes 0 edges.



Set algo used to detect cycle in MST Construction using Kruskals algo.

if (u,v) edge Min cost Edge deleted from graph
  if (Find(u) != Find(v))
    // (u,v) not forms cycle in MST
  else
    // (u,v) forms cycle in MST

{ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ } // initially each vertex in own set

Edge del from graph		Edge forms cycle in MST
1) (1,2)	Find(1) ≠ Find(2)	No, [Add(1,2) to MST]
2) (6,7)	Find(6) ≠ Find(7)	No, "
3) (8,9)	Find(8) ≠ Find(9)	No, "
4) (1,4)	Find(1) ≠ Find(4)	No, "
5) (2,4)	Find(2) = Find(4)	Yes, [discard(2,4)]
6) (1,3)	Find(1) ≠ Find(3)	No, [Add(1,3)]
till (6)		
7) (3,4)	Find(3) = Find(4)	Yes, [discard(3,4)]

↓ continue until MST(n-1) edges or Graph G becomes '0' edges.

Algo Kruskals (n, e, cost[][])
{
O(e) Build Min heap (Q) for all edges of graph based on Priority edge cost.
O(n) Initialize set array each vertex of graph is one set. // Initially n sets each set one element.
  i = 0 // # of MST Edges
  Min cost = 0 // cost of MST
  while (i < (n-1) && Q not empty)
  {
    Edge Del Min cost Edge (u,v) from Min heap (Q)
    Logn X = Find(u) ; Y = Find(v).
    if (X ≠ Y) // (u,v) not forms cycle in MST
    {
      Min cost = Min cost + cost(u,v)
      i++;
      t[i,1] = u; t[i,2] = v
      Union (X,Y);
    }
  } e times
  if (i < n-1) // No ST in Graph
    return (false);
  else
    return (t[][], Min cost)
}

TC of kruskal

$\boxed{O(e \log e) \approx O(e \log n)}$

Huffman coding :-

- [Optimal Prefix encoding]
- Compression Technique && Encoding Technique
- Encoding {char to binary}
- Encoding Technique
  1] Fixed length Encoding
  2] Prefix encoding.

1) Fixed length Encoding: [ASCII Encoding]
[Every char of message encoded by same # of bits]

n bits ⇒ 2^n distinct char can Encode

Ex: [xxx] ⇒ 8 char can encode

To encode n char ⇒ $\lceil \log_2 n \rceil$ bits per char.

aaabababbaaacdaa
bbbcxxyyzaaab
cccdddyzzz

a: 11
b: 7
c: 5
d: 4
x: 2
y: 3
z: 3

7 distinct char $\Rightarrow \lceil \log_2 7 \rceil = 3$ bits

Using Fixed length Encoding

Message length $\Rightarrow$
$[11+7+5+4+2+3+3] * 3$ bits
$= 35 * 3 = \underline{105 \text{ bits}}$

disAdv:- Msg length is high becoz No diff b/w Most frequently used & least frequently used with respect to #f bits.

2] Prefix Encoding :- [Huffman encoding]

- Most frequently used char represented by less #f bits and
- least frequently used char represented by more #f bits.
- and Proper prefix of any char code is not code of any other char.

P: [11010]

Should not be code for other char: 1, 11, 110, 1101

P: 11010
Q: 110
R: 11

MSG = 11010 1101 } Decoding not possible.

Adv:- Message length [Sum of frequency count] is optimal [Min] if Huffman coding used.

---

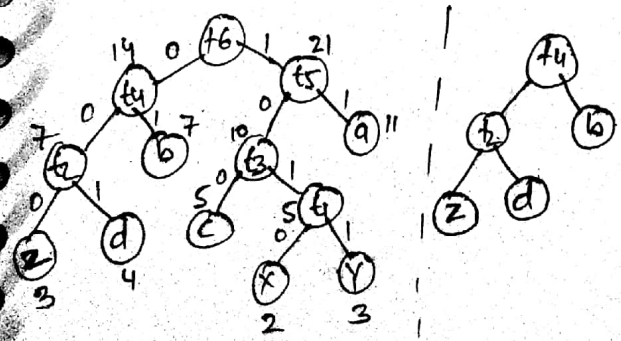
2 way optimal merge tree :- [$\theta(n \log n)$]

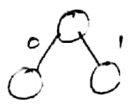
n distinct char.

1) each char denote tree with weight is frequency count
2) del 2 Min cost trees & Merge into one tree with weight Sum of weights of both trees & Insert merged tree into list of trees.
3) Repeat (2) until list becomes single tree.

Char	a	b	c	d	x	y	z
freq	11	7	5	4	2	3	3

{ (a)11 (b)7 (c)5 (d)4 (x)2 (y)3 (z)3 }





a ⇒ 11
b ⇒ 01
c ⇒ 100
d ⇒ 001
x ⇒ 1010
y ⇒ 1011
z ⇒ 000

} Encoding

*** (1) Sum of frequency count [Message length] using Huffman coding

$11*2 + 7*2 + 5*3 + 4*3 + 2*4 + 3*4 + 3*3$

$= 92$ bits.

*** (2) Avg bits char using Huffman code

$= \frac{\text{Sum of freq}}{\#f \text{ char}} = \frac{92}{35} = 2.63$

(3) Min & Max bits for char

[Min 2 bits Max 4 bits]

Eg: char a e i o u sp nl ? x
freq 200 150 80 40 12 5 8 10 4

1) Sum of frequency count using fixed length
= no. of distinct char : ⌈log 9⌉ = 4 bits/char
= [200+150+80+40+12+5+8+10+4]*4
= [509]*4 = 2036

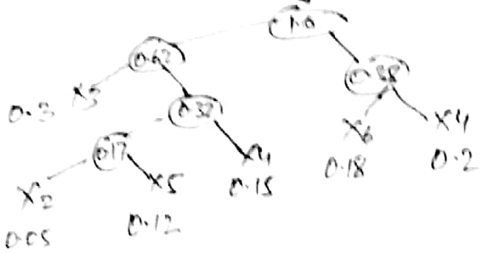
2) Sum of frequency count using Huffman encoding?

Sum of frequency count
9+17+29+ [illegible]
179+159+309+509
= 1145 bits [Sum of all internal nodes]

② Avg bits/char => $\frac{1145}{509}$ = 2.24 bits

③ Min: 1 bit Max: 7 bits

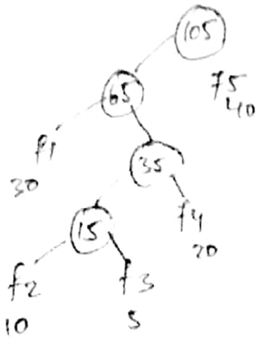
Eg. char {x_1, x_2, x_3, x_4, x_5, x_6}.
freq {20%, 5%, 30%, 15%, 12%, 18%}.
What is sum of frequency count using Huffman coding?



=> Sum of frequency count:
= 0.17 + 0.32 + 0.62 + 0.38 + 1.0
= 2.49

Eg. Assume (n+m) record moves required to merge two files with n + m sorted records each into single file of sorted record.
How many min record moves required to merge n=5 [files] ⟨f1 f2 f3 f4 f5⟩ with ⟨30, 10, 5, 20, 40⟩ sorted records each into single sorted file?

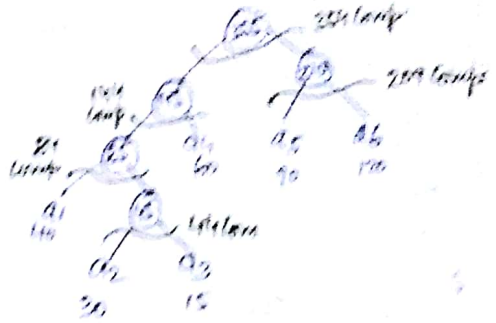
Soln.



=> 15 + 35 + 65 + 105
= 220 [Min record moves to merge 5 files into single files].

Eg: 6 sorted arrays
$[a_1\ a_2\ a_3\ a_4\ a_5\ a_6]$ each array $\{40, 20, 15, 60, 90, 120\}$ elements in sorted order respectively.
⇒ How many max elements comp. required to merge 6 sorted arrays into single sorted array using optimal algo?

Sol: [merge 2 arrays which are having min elements and so on]



Max comp to merge using optimal algo
$[40 + 75 + 135 + 200 + 335] = 785$ comp

Fraction knapsack problem

Given n objects & a capacity knapsack (bag) profits $\langle P_1, P_2, P_3 \dots P_n \rangle$
weights $\langle w_1, w_2, w_3 \dots w_n \rangle$
if x_i fraction of obj i included into knapsack which require $w_i * x_i$ weigh & result $P_i * x_i$ profit
where $0 \le x_i \le 1$
Determine fraction value $\{x_1, x_2, \dots x_n\}$ such that [Sum weights included into knapsack] $\le m$ & Maximum Profit

$$\begin{cases} \text{Maximize} \sum_{i=1}^{n} P_i * x_i \text{ Profit} \\ \text{Subjected to} \sum_{i=1}^{n} w_i * x_i \le m \\ \text{& } 0 \le x_i \le 1 \end{cases} \Big\} \text{ fraction knapsack problem}$$

⇒ $n = 6$ [# objects] & $m = 90$

$\langle P_1, P_2, P_3, P_4, P_5, P_6 \rangle = \langle 70, 30, 50, 60, 60, 80 \rangle$

$\langle w_1, w_2, w_3, w_4, w_5, w_6 \rangle = \langle 20, 10, 20, 25, 35, 50 \rangle$

Sol: i^{th} object
w_i unit weight = P_i profit
Profit / unit weight ⇒ P_i/w_i

Greedy Sol:— Max P_i/w_i objects should choose first and so on.

$\left\langle \frac{P_1}{w_1}, \frac{P_2}{w_2}, \frac{P_3}{w_3}, \frac{P_4}{w_4}, \frac{P_5}{w_5}, \frac{P_6}{w_6} \right\rangle = \left\langle \frac{70}{20}, \frac{30}{10}, \frac{50}{20}, \frac{60}{25}, \frac{60}{35}, \frac{80}{50} \right\rangle$
$= \langle 3.5, 3, 2.5, 2.4, 1.71, 1.6 \rangle$

Order of obj = $\langle O_1, O_2, O_3, O_4, O_5, O_6 \rangle$
Obj order based on decreasing order P_i/w_i ratio.

Fill knapsack by obj in above order.

$[w_i * x_i \le 90]$: $20*1 + 10*1 + 20*1 + 35*1 + 25*\frac{1}{5}$

$[P_i * w_i]$: $70*1 + 30*1 + 50*1 + 60*1 + 40*\frac{1}{5}$
$= (218)$ Max profit

$$\begin{cases} \langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle \\ \langle 1, 1, 1, 1/5, 1, 0 \rangle \end{cases} \Big\} \text{ Optimal Solution}$$

Max profit = {218} optimal profit.

TC

```
Algo knapSack (P[], W[], n, m) ⇒ θ(nlogn)
{
  1. Sort objects based on                    θ(nlogn)
     decreasing order of their P/w ratio
     profit = 0, m1 = m
  2. for (i=1; i≤n; i++)                      θ(n)
     { if (Wi ≤ m1)
       { xi = 1
         Profit = Profit + Pi
         mi = m1 - Wi
       }
       else
       { xi = m1/Wi ;
         Profit = Profit + Pi * Xi
       }
     }
  3. return (x[], Profit);
}
```

0/1 Knapsack Problem

$\sum_{i=1}^{n} P_i x_i$ Maximize profit

$\sum_{i=1}^{n} W_i x_i \leq m$ [Subjected to]

$x_i = 0$ or 1

Eg: $n = 3 \quad m = 30$

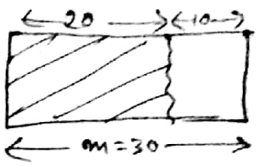
$\langle P_1, P_2, P_3 \rangle = \langle 60, 40, 30 \rangle$

$\langle W_1, W_2 W_3 \rangle = \langle 20, 15, 11 \rangle$

Soln Order obj [dec P/w ratio] = $\{O_1, O_3, O_2\}$

Profit = 60 / $\langle x_1 x_2 x_3 \rangle$ Not opt.

$\langle 1, 0, 0 \rangle$



Brute Force Algo :—

Comput cost of all fesible solution & returns fesible sol whose result optimal. {TC = θ(2^n)}

⇒ 0/1 Knapsack Problem [Brute Force Algo]

Compute profit for every subset of obj && return subset of obj whose sum of weighs ≤ m && sum of Profits maximum.

Obj	Profit	Weight ≤ m
{ }	0	0
{O1}	60	20
{O2}	40	15
{O3}	30	11
{O1,O2}	100	35 X
{O1,O3}	90	31 X
{O2,O3}	70	26 ✓ optimal Soln
{O1,O2,O3}	130	46 X

Job Scheduling based on deadline

Given n Tasks [T1, T2, ... Tn]

&& Profits ⟨P1 P2 .... Pn⟩ && deadline ⟨d1, d2 .... dn⟩

If Ti executed on / before di time then resulted profit Pi otherwise no profit.

Maximize Profit in order to Schedule Tasks with Constraints

a) Only one task can execute at a time

b) Every task ready to execute

c) One unit execution time for each task

Eg → {OS, CO, Algo, DS, DBMS}

Marks {20, 15, 25, 30, 18}

deadline {3, 2, 3, 1, 2}

Schedule task to get Max marks

Each assignment required 1 day to complete

day1	day2	day3
DS	OS	Algo
30	20	25

Max Marks = 75 } optimal

Greedy Soln :- 1) Sort tasks based on [TC: $\theta(n \log n)$] decreasing order of profits.

{$T_1, T_2, T_3 \dots T_n$} Order of Tasks

$P_1 \geq P_2 \geq P_3 \geq \dots P_n$

2) Choose task T_i in above order (d_i deadline)

& Schedule T_i at d_i^{th} time if free

if d_i time not free $(d_i - 1)^{th}$ time and so on.

if no time slot free from d_i to 1 then discard T_i

Sorting Algo :-

1) Comparison based sorting algo

Algo	Time Complexity BC	AC	WC	Stable Sorting	Inplace Sorting
1) Quick Sort	$\theta(n \log n)$	$\theta(n \log n)$	$\theta(n^2)$	No	Yes
2) Merge Sort	$\theta(n \log n)$	$\theta(n \log n)$	$\theta(n \log n)$	Yes	No
3) Heap Sort	$\theta(n \log n)$	$\theta(n \log n)$	$\theta(n \log n)$	No	Yes
4) Bubble Sort	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$	Yes	Yes
5) Selection Sort	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$	No	Yes
6) Insertion Sort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$	Yes	Yes

WC time Comp^x of any comparison based Sorting algo for n elmnts = $\Omega(n \log n)$

2) Non Comparison based Sorting Algo :-

1) Counting Sort :-

TC ⇒ $\theta(n+k)$

[Stable && Not Inplace Sorting]

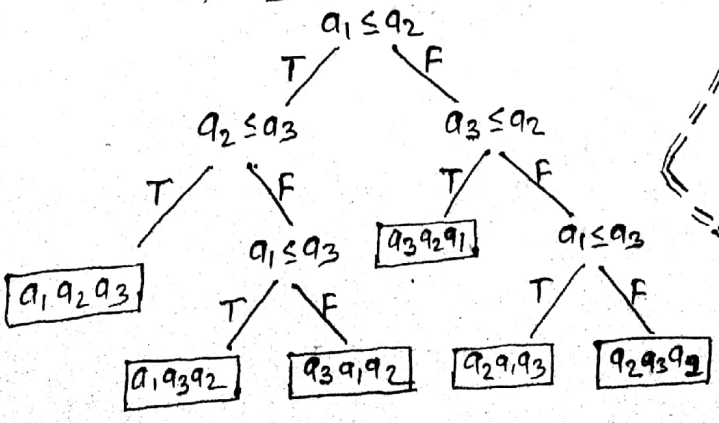
2) Radix Sort [Bucket Sort]

TC ⇒ $\theta(d*n)$

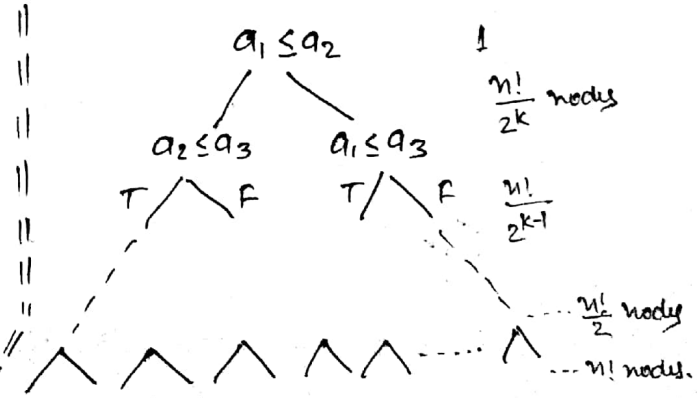
[Stable && Inplace Sorting].

Let n = 3 [$a_1\ a_2\ a_3$]

Decision tree



← 3! = 6 leaf nodes in decision tree →



← n! leaf nodes for decision tree →

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

$k = \log n! = \theta(n \log n)$

{WC atleast $n \log n$ Comp required for any comparison based Sorting algo}

Bubble Sort Algo:—

Compare & Exchang Adjancecy elements

Procedure:—

pass 1: comp a[j] & a[j+1]
if a[j] > a[j+1] then
swap (a[j], a[j+1])
for j=1 to n-1
// a[n] gets sorted

pass 2: comp a[j] & a[j+1]
if (a[j] > a[j+1]) then
swap (a[j], a[j+1])
for j=1 to n-2
// a[n-1...n] gets sorted

pass(n-1) [Repeat]

Its stable & inplace algo

```
Algo Bubble Sort (a,n)
{ for (i=1; i ≤ n-1; i++)
  { for (j=1; j ≤ n-i; j++)
    { if (a[j] > a[j+1])
      { swap (a[j], a[j+1]);
      }
    }
  }
```

	#f comp	#f Swap
pass 1	n-1	[0 to n-1]
pass 2	n-2	[0 to n-2]
:	:	:
pass n	1	[0 to 1]

$TC = \frac{n(n-1)}{2} + \{0 \text{ to } \frac{n(n-1)}{2}\}$

$= \Theta(n^2)$ All Cases.

Selection Sort Algo:—

Find Position of min elemnt from a[j] to a[n] and swap (a[minPos], a[j])
where j is 1, 2, ... n-1 for n-1 passes.

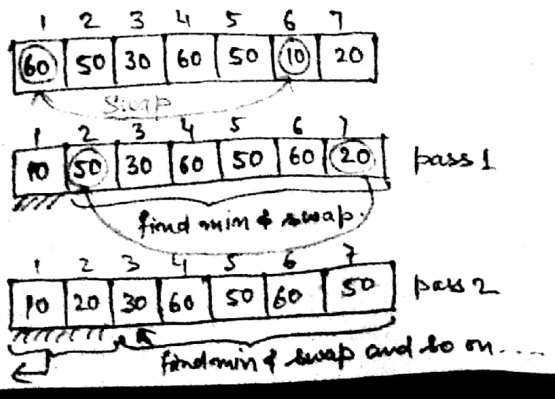
Procedure:—

Pass 1 :- Find pos of min element a[1] to a[n] & swap with a[1]

pass 2 :- Find pos of min ele a[2] to a[n] & swap with a[2]

pass n-1 :- Find pos of min ele a[n-1] to a[n] & swap with a[n-1]

Eg




```
Algo Selection Sort (a,n)
{ for (i=1; i ≤ n-1; i++)
  { // Find pos of min ele a[i] to a[n]
    Minpos = i;
    for (j = i+1; j ≤ n; j++)
    { if (a[j] < a[minpos])
        minpos = j
    }
    Swap (a[i], a[minPos])
  }
}
```

TC

	#f comp	#f Swap
pass 1	n-1	1
pass 2	n-2	1
pass 3	n-3	1
:	:	:
pass n-1	1	1

Time comp $= \{\frac{n(n-1)}{2} + (n-1)\} = \underline{\Theta(n^2)}$ All Cases.

⇒ Selection sort can be treated as special case of Quick sort where Pivot element is Min element

⇒ Selection Sort TC ⇒ $C_1 \cdot n^2$ "AC"

Quick Sort TC ⇒ $C_2 \cdot n\log n$ "AC"

{ Selection sort runs faster than Quick sort for less I/p array.

{ Quick Sort runs faster than Selection sort for large I/p array

```
Sorting(a,n)
{ if(n<20)
    Selection sort(a,n)
  else
    Quick sort(a,n)
}
```

Insertion Sort :-

a[i] places correct position in sorted part of array [1... i-1]

where i=2 to n for n-1 passes.

Procedure :-

pass1 place a[2] in correct pos of sorted array[1...1] // a[1...2] in sorted order.

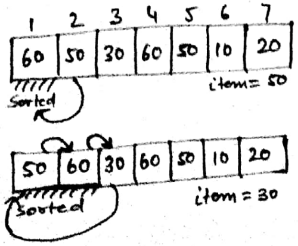
pass2 place a[3] in correct pos of sorted array a[1...2] // a[1...3] in sorted order.

⋮

pass3 place a[n] in correct pos of sorted array a[1...n-1] // a[1...n] in sorted array.

```
Algo Insertion Sort(a,n)
{ for(i=1; i≤n; i++)
  { item = a[i+1]; j = i;
    while(j≥1 && a[j] > item)
    { a[j+1] = a[j]
      j = j-1
    }
    a[j+1] = item;
  }
}
```

Eg:



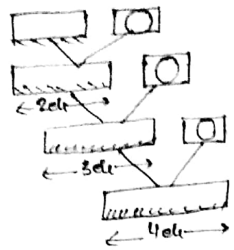
goes like this

	Min comp	Min Shift	Max comp	Max Shift	Avg comp	Avg Shift
Pass1	1	0	1	1	1	1
Pass2	1	0	2	2	2/2	2/2
pass3	1	0	3	3	3/2	3/2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
pass n-1	1	0	n-1	n-1	(n-1)/2	(n-1)/2
	(n-1)(comp)		n(n-1)/2 + n(n-1)/2		n(n-1)/4 + n(n-1)/4	
	Best Case TC of Insertion Sort : θ(n)		Worst Case TC of Insertion Sort θ(n²)		Avg Case TC of Insertion Sort : θ(n²)	

If binary search is used instead of linear search to place an element in sorted position in Insertion sort, then the worst case TC of the Algorithm remains same = θ(n²)

Because it may reduce the time of # comparisons but shifting of elements will take same time as previous.

⇒ Insertion Sort Special Case of Merge Sort where:-
Every merge is merging of 2 sorted array with only one element 2nd sorted array



⇒ If I/p array with less ele
Insertion sort runs faster than Merge sort

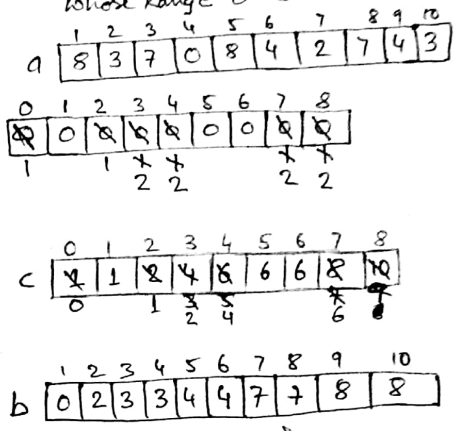
⇒ If I/p array with more Ele
Merge sort runs faster than Insertion sort.

Q. If almost all the element of an array is sorted then name the sorting method which will sort the array in $\theta(n)$ times.
a) SS b) QS c) MS d) IS

Non Comparison based Sorting:-

1) Counting Sort Algo:-
array a[1...n] are n integers whose element value Range 0...k
i) Initialize count array c[0...k] by 0's
ii) for each element of array "a" increase count val of count array
iii) Inc count array by Previous index of count array.
$c[i] = c[i] + c[i-1]$
iv) a[i] element place $b[c[a[i]]]$ && decrease $c[a[i]]$ by 1.

Eg) array of 10 elements (int) whose Range 0-8




```
Algo CountingSort(a,n,k)
{ // a[1...n] are n integers whose
  // element Range 0 to K.
  for(i=0; i≤K; i++)          } θ(k)
  { c[i]=0;                   }
  } for(i=1; i≤n; i++)        } θ(n)
  { c[a[i]]=c[a[i]]+1;        }
  }
  for(i=1; i≤K; i++)          } θ(k)
  { c[i]=c[i-1]+c[i];         }
  }
  for(i=n; i≥1; i--)          } θ(n)
  { b[c[a[i]]]=a[i]           }
    c[a[i]]=c[a[i]]-1;        }
  }
  return(b[1...n]);
}
```

TC of counting sort : $\theta(n+k)$
Best case TC of counting sort $\theta(n)$ if $K \leq n$

[It is stable sorting but not inplace]

Radix sort Algo!—

Radix (r) :- r different symbols used to represent any number using Number systems

Symbol value $\{0, 1, 2, \ldots r-1\}$

radix-16 $\{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$.

radix-20 $\{0, 1, 2, \ldots, 9, A, B, C, D, E, F, G, H, I, J\}$

Eg → $n = 8$ $d = 3$ $r = 10$ (decimal numbers)

	1	2	3	4	5	6	7	8
a	956	743	143	650	836	985	143	743

743 143 143 743 985 956 836

Pass 1.

650 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9

	1	2	3	4	5	6	7	8
a	650	743	143	143	743	985	956	836

Pass 2

743 143 143 956 836 743 650 985 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9

	1	2	3	4	5	6	7	8
a	836	743	143	143	743	650	956	985

Algo!—

a[1... n] n elmnts in decimal val of each element (0 -- n^3)

1) Convert each elmnt of array from radix-10 to radix-n
[x is elmnt of array]

$\log_n^x = \log_n^{n^3} = \underbrace{3}_{\text{constant}}$ digits

2) Run Radix Sort for above resulted array.

3) Convert each array elmnt of above result from radix-n to radix-10

---

radix Sort :-

⇒ n : #f elmnts of array a[1... n]
r : is radix of Number System
d : #f digits of each element.

Step 1. define r- Queues.
[Queue 0, Queue 1, ... Queue (r-1)]

Step 2: For each elmnt of array place element into Queue(x) based k^{th} least significant digit(x) of elmnt and retrive elmnt from Queue 0 to Queue r-1 Store in array "a".

[Repeat "d" times]

Pass 3

143 143 Q0 Q1 Q2 Q3 Q4 Q5 650 Q6 743 743 836 Q7 Q8 985 956 Q9

	1	2	3	4	5	6	7	8
a	143	143	650	743	743	836	956	985

→ Sorted order.

Tc of Radix Sort
n : #f elmnts of array.
d : #f digits of each elmnt
r : radix

$O(dn)$

Best case Tc of RadixSort
$O(n)$ if d is const.

[1 - n^3]
elmnt(x) = n^3
val
#f digits to dnote x in decimal

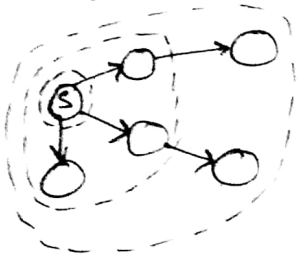
Graph Traversal Algo:-

02/09/17

1) Breadth first Search (BFS)
2) Depth first Search (DFS)

1) Breadth first Search (BFS)
[level order Traversal]

Graph vertices visits level by level from source

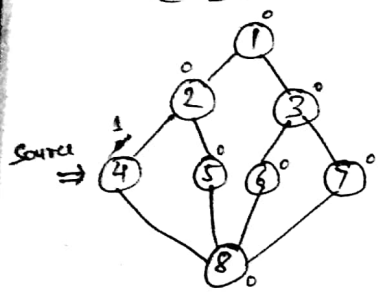


Visited[vi] = { 0 vi not visited
1 vi visited.

Unexplored [white] vertex :- Vertex 'v' not visited [Not started exploring]

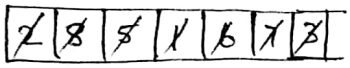
Exploring [Gray] vertex :- (V) - Vertex "V" visited but Adj "V" not visited yet

Explored [Black] vertex :- (V) :- Vertex 'V' visited & Adj "V" also visited.



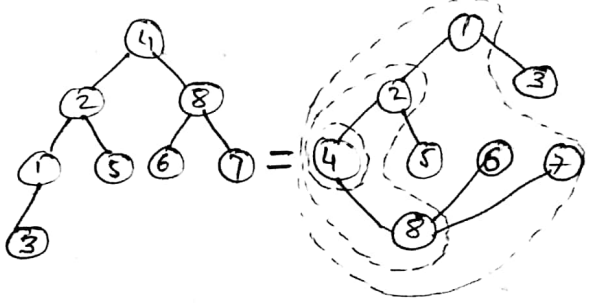
BFS Sequence.

4 2 8 5 1 6 7 3



4s 4e 2s 2e 8s 8e 5s 5e 1s 1e
6s 6e 7s 7e 3s 3e

BFS spanning tree




```
Algo (BFS (G, n, e, s))
{  for(i=1; i<=n; i++)
θ(n) [{  visited[vi] = 0;
     }
   visited[s] = 1.
   while (TRUE)
n  [{  v is set of Adj of s  ---> θ(deg(s))
times    for (all vertices of v)  ---> θ(n)
deg(s)   {  if (visited[v] == 0)
            {  visited[v] = 1
               Insert (Queue(Q), v);
            }
         }
      if (Empty(Queue(Q)) return;
      else
         s = delete(Queue(Q));
   }
```

TC of BFS

a) Using Adj list Graph :-
$\theta(n+e)$ [TC of BFS]

b) Using Adj Matrix Graph
$\theta(n^2)$ [TC of BFS]

SC of BFS

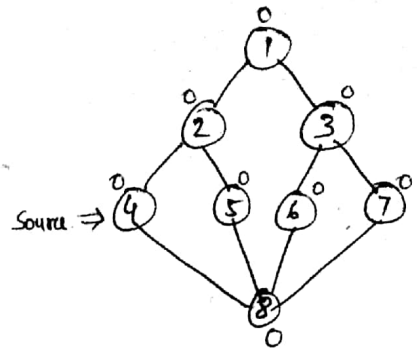
- $O(n)$ Queue DS Space.

2) Depth first search :-

Graph vertices visited based on depth.

Vertex (v) which started Exploration first is the vertex complete exploration at last.

{Stack DS used for DFS traversal}.

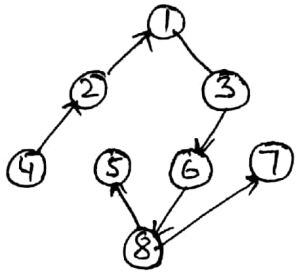


4s 2s 1s 3s 6s 8s 5s 5e 7s 7e 8e 6e 3e 1e 2e 4e

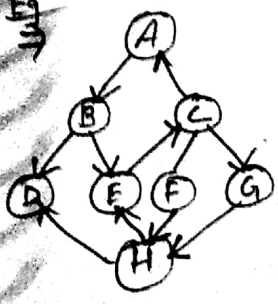
DFS Sequence :- 4, 2, 1, 3, 6, 8, 5, 7 [visited order]

DFS(4) Adj(4) = {2, 8}
↳ DFS(2) Adj(2) = {1, 4, 5}
↳ DFS(1) Adj(1) = {2, 3}
↳ DFS(3) Adj(3) = {6, 7}
↳ DFS(6) Adj(6) = {3, 8}
↳ DFS(8) Adj(8) = {4, 5, 6, 7}
↳ DFS(5) Adj(5) = {2, 8}
↳ DFS(7) Adj(7) = {3, 8}

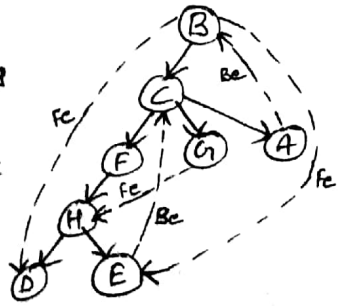
DFS Spanning tree :-



Eg ⇒



Spanning tree ⟹ Source "B"



Back edge :- Edge which is connected Child to parent.

Forward edge :- Graph edge from prev level to next level.

Tree edge :- DFS Spanning tree edges

Cross edge :- Graph edge which connect same level tree vertices.

Cross edge = 0
Back edge = 2
Forward edge = 3
Tree edges = 7.

```
Algo DFS(G, n, e, s)
{  for(i=1; i≤n; i++)
   { visited[vi] = 0}
   DFS(s)
   { Print(Ss)
     visited[s] = 1;
     v is set of Adj of vertex(s)
     for(vertices of vset)
     { if(visited[v] == 0)
       { DFS(v)
       }
     }
   } print(Se)
```

TC of DFS Algo

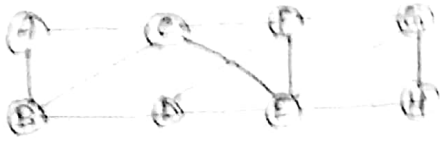
{ θ(n+e) if graph is Adj List Rep
{ θ(n²) if graph is Adj Matrix Rep.

SC of DFS Algo

O(n) // Stack Space.

Applications

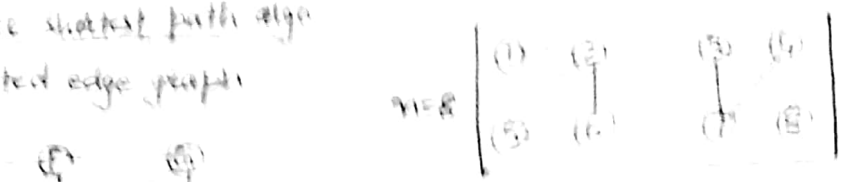
1) BFS Algo can used as
Single source shortest path algo
for unweighted edge graph.



v	0	1	2	3
d[v]	0	1	2	3

Result of Single Source
Shortest path

2) BFS/DFS can be used to test #f connected components of undirected graph.

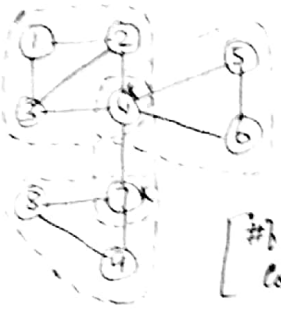


```
k=0
for(i=1; i<=n; i++)
    visited[vi] = 0
for (i=1; i<=n; i++)
{ if (visited[vi]==0)
  { call DFS(vi);
    k++;
  }
}return(k);
```

DFS(1)
DFS(3)
DFS(8)

G has 3 connected components
[#f DFS Algo calls to become all elements of visited array is]

3) BFS/DFS can be used to Test #f "Bi-Connected Components" of undirected graph.



K=2

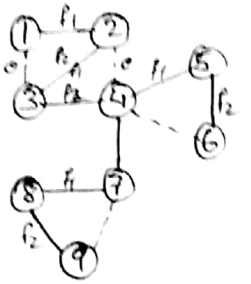
[#f Biconnected Components] ≥ K+1

If graph G with one cut vertex [articulation point]
then graph G is Single Bi-Connected Component

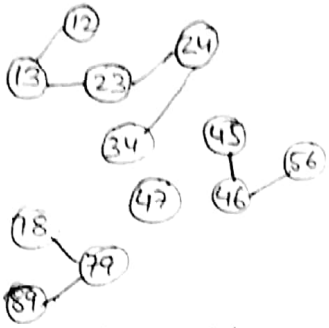
If graph has K cut vertices then
G has at least (k+1) Bi-Connected components

Procedure to test #f Bi-Connected Comp of G:-

1) Call DFS & f divide edges as tree edge & f Back edge
2) Construct auxilary graph



Bi-Connected Components
"Graph G"



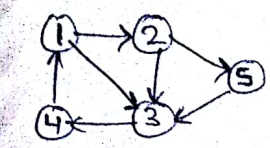
4 Connected Comp.
"auxilary graph" G.

⇒ Edge of G are vertices of G'
⇒ edge of G' are (e, f1, f2, ...fx) loop in G then ef1, ef2 ... efx are edges of G'

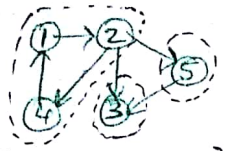
3) # of connected components of G' is #f Biconnected component of Graph.

4) DFS algo can be used to test #f strongly connected components of Directed graph.

if every pair of vertices (u,v) there must be path from u to v & v to u then strongly connected component.



One Strongly Connect. Component.



3 Strongly Connected Compt.

5) DFS algo can be used to detect cycle in Directed graph.

6) DFS algo can be used to detect cycle in undirected graph

Procedure:-

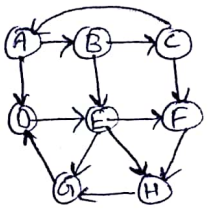
1) Call DFS and Store finish time of exploration of vertices in stack (S)

2) Compute G^T: Transpose of G: G^T

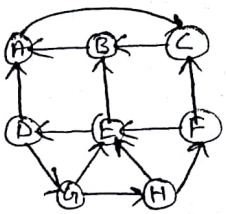
3) Call DFS for G^T based on unvisited vertex from top of stack (S) as source.

[#f DFS calls to visit all vertices of G^T is #f strongly connected component of G]

Eg:



G



G^T

Call DFS for G.

Calling DFS for G^T

DFS(A) = {A B C}

DFS(F) = {F E D G H}.

2 times DFS called to visit all vertices of G'

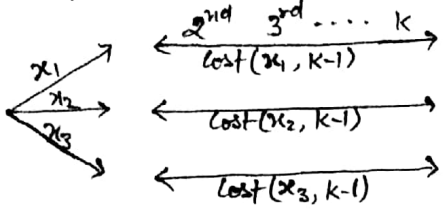
G' has 2 Strongly connected components.

03/09/17

Dynamic Programming :-

[Also called principle of optimality design technique]

Whatever cost of first decision, Cost of remaining decision along with first decision must be optimal.



$$= \text{Min} \begin{cases} x_1 + cost(x_1, k-1) \\ x_2 + cost(x_2, k-1) \\ x_3 + cost(x_3, k-1) \end{cases}$$

Greedy Algo	Dynamic Prog. Algo
⇒ Choose one feasible solution using predefined method return optimal solution.	⇒ Computes all feasible solutions & returns optimal solution.
⇒ Runs in Polynomial Time Complexity.	⇒ May required exponential Time comp. [Almost poly TC]
⇒ Fail to solve some optimization problems. Eg: 0/1 knapsack problem	⇒ Solve every optimization problem because it uses principle of optimality.

Brute force algo	Dynamic Prog. Algo
⇒ Computes all feasible Soln & return optimal.	⇒ Computes all feasible sol such that avoids recomputation & returns optimal Soln.
⇒ If #f feasible sol exponential $\{2^n, n!\}$ Then TC of solution using Brute force algo also exponentially	⇒ Even exponential feasible sol problem may solves in poly time becoz of avoiding recomputation
⇒ Uses less space comp	⇒ More space comp.

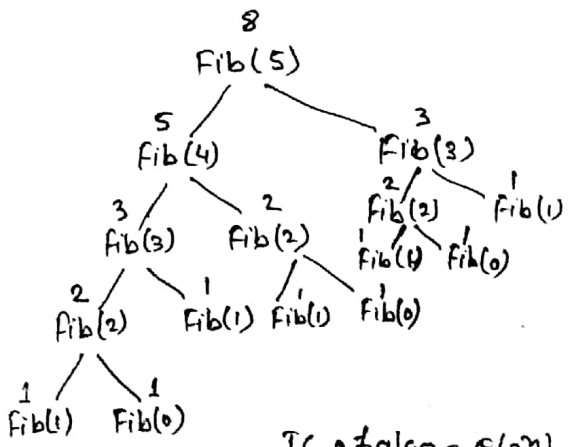
Steps to design Dynamic Programming.

1) Solution of problem be formulated Recursively. [To avoid recomputation]
2) Formulate recurrence relation for solution of Problem.
3) Solve recurrence Relation for given I/p in Bottom-up approach
   - return optimal solution
   - return optimal sequence.

① nth Fib Number :-

$$Fib(n) = \begin{cases} 1 & , n \le 1 \\ Fib(n-1) + Fib(n-2) & , n > 1 \end{cases}$$

```
Algo Fib(n)
{
  if (n≤1)
     return (1);
  else
     return (Fib(n-1) + Fib(n-2));
}
```



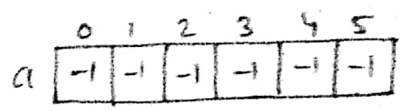
TC of algo = $O(2^n)$ expo

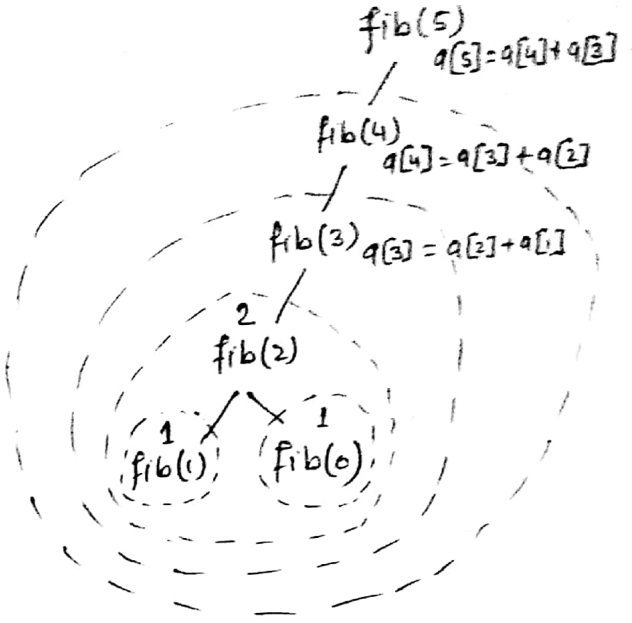
SC of algo = $\theta(n)$

// no depth of recursion

DP of nth Fib Num Algo :-

```
Algo nth Fib(n)
{  for(i=0; i<=n; i++)
   {  a[i] = -1 }
   Fib (n)
   {  if (n<=1)
      {  a[n] = 1.
         return (1)
      }
   else
      {  if (a[n-1] == -1)
      {  a[n-1] = fib(n-1); }
       if (a[n-2] == -1)
       {  a[n-2] = fib(n-2); }
       a[n] = a[n-1] + a[n-2];
       return(a[n]);
      }
   }
}
```





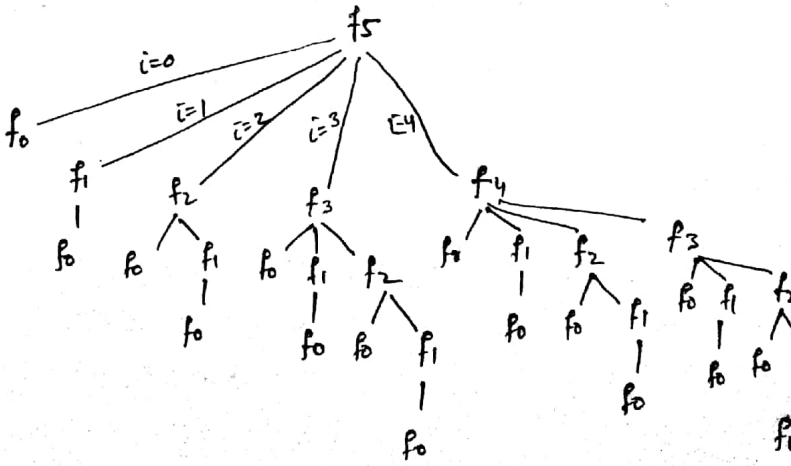
#f fun calls : n+1
[each fun call required constant time]

TC of algo : $\theta(n)$

SC of algo
a[0...n] : (n+1) memory units
Stack : n entries
$\theta(n)$

WB
Q No-61

```
Algo foo(i)
{  if (n == 0)
      return (1)
   else
    {  sum = 0
       for (i=0; i<n; i++)
       {  sum = sum + foo(i) }
       return (sum)
    }
}.
```



#f fun calls : $[1+2+2^2+\dots 2^{n-1}] + 1. = 2^n$

TC of algo = $\theta(2^n)$

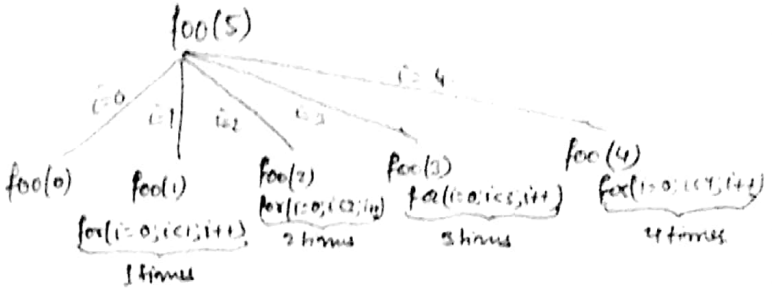
SC of algo : $\theta(n)$ // Stack space

Q62

```
for(i=0; i<n; i++)
{   a[i]=1 }

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

0	1	2	3	4
-1	-1	-1	-1	-1



#f fun call $[n+1]$

T.C of algo $[n+1] + \{1+2+\ldots+(n-2)+(n-1)\}$

$= \Theta(n^2)$

S.C of algo $a[0 \ldots n-1] \Rightarrow \Theta(n)$

Stack space $\Rightarrow \Theta(1)$ // 2 entries

Sc of algo $\Theta(n)$

Matrix chain Problem :-

$\Rightarrow A_{p \times q}$ $B_{q \times r}$

cost of Matrix Mul: $P \cdot q \cdot r$ (elemnt Mul. required)

=> Matrix Mul Associativity [A, B, C, Matrices]

$(A * B) * C \equiv A * (B * C)$

Eg $A_{100 \times 75}$ $B_{50 \times 75}$ $C_{75 \times 10}$

① cost Mul $(A * B)_{100 \times 75} * C_{75 \times 10}$

$\rightarrow 100 \times 75 \times 50 = 375000$

$\rightarrow 100 \times 75 \times 10 = 75000$

$4,50,000$ (Scalar Mul)

optimal:

② cost Mul $(A * (B * C)_{50 \times 10})_{100 \times 50}$

$\rightarrow 50 \times 75 \times 10 = 37500$

$\rightarrow 100 \times 50 \times 10 = 50000$

87500 (Scalar Mul)

Problem Statement:-

Given n matrices

$A_1\ A_2\ A_3 \ldots A_n$ and

$\langle P_0\ P_1\ P_2\ P_3 \ldots P_n \rangle$ order set

s.t A_i order $P_{i-1} \times P_i$

Paranthesize n matrices

is minimum [n=4]

$A_1\ A_2\ A_3\ A_4$

$A_1 (*(A_1 * A_2) * A_4)$

$A_1 (* A_2 * (A_3 * A_4))$

$(A_1 * A_2) * (A_3 * A_4)$

$((A_1 * A_2) * A_3) * A_4$

$(A_1 * (A_2 * A_3)) * A_4$

} 5 different paranthesizations for 4 Matrices.

#f Matrices (n)		#f Paranthesization (T(n))
1	A_1	1
2	$(A_1 A_2)$	1
3	$A_1(A_2A_3)$ $(A_1A_2)A_3$	2
4	$A_1A_2A_3 \cdot A_4$ 1*2+1*1+2*1	5
5	$A_1A_2A_3A_4A_5$ 1*5+1*2+2*1+5*1	14
	⋮	⋮
n		$\frac{1}{n}\left[\frac{(2n-2)!}{(n-1)!\,(n+1)!}\right]$

T(n): #f paranthesizations for n matrices.

$$T(n) = \begin{cases} 1 & , n \le 2 \\ \sum_{K=1}^{n-1} T(K) \cdot T(n-K) & , n > 2 \end{cases}$$

$A_1\ A_2\ A_3\ A_4 \ldots A_{n-1}\ A_n$

$$T(n) = \underset{K=1}{T(1)} \cdot T(n-1) + \underset{K=2}{T(2)} \cdot T(n-2) + \ldots \underset{K=(n-1)}{T(n-1)} \cdot T(1)$$

$$T(n) = \frac{1}{n}\left\{\frac{(2n-2)!}{(n+1)!\,(n-1)!}\right\} > 2^n$$

[#f feasible sol for matrix chain Prob]

Brute force algo: Matrix chain Problem

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

DP Solution of matrix chain Problem.

[Recursive soln]

$A_1\ A_2 \ldots A_n$ are n matrices.

$\langle P_0\ P_1\ P_2 \ldots P_n \rangle$ are order set

Assume C(1,n) is Min cost of Mul of Matrices.

$A_1\ A_2 \ldots A_n$

$(A_1\ A_2 \ldots A_K)$ — $P_0 \times P_1$ ... $P_{K-1} \times P_K$ — Min cost of Mul of $A_1 \ldots A_K$ is C(1, K)

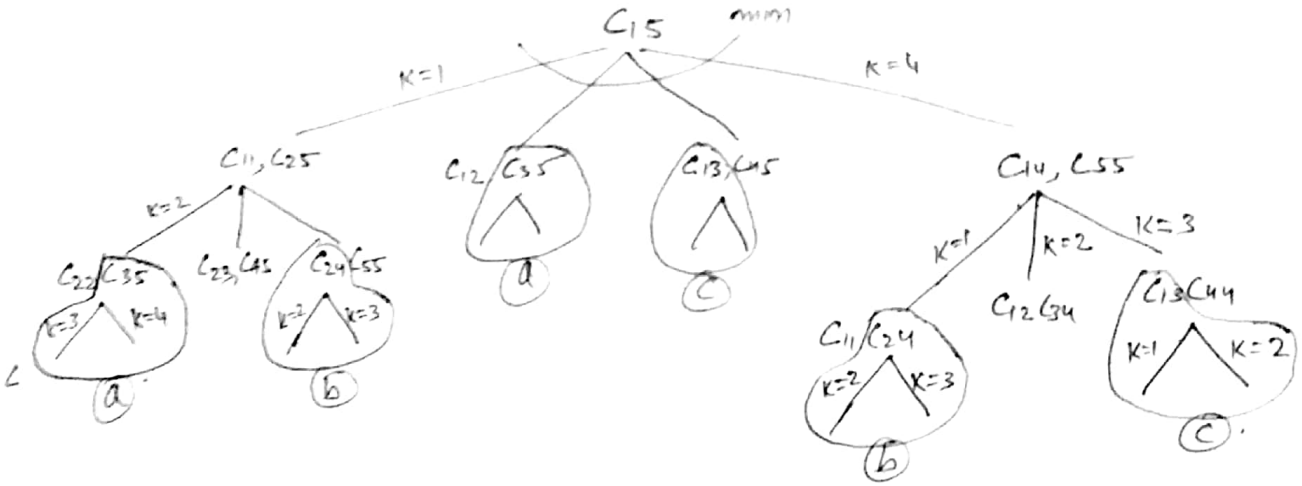
$(A_{K+1} \ldots A_n)$ — $P_K \times P_{K+1}$ ... $P_{n-1} \times P_n$ — Cost of Mul $A_{K+1} \ldots A_n$ matrices is C(K+1, n)

$$C(1,n) = \min_{1 \le K < n} \{ C(1,K) + C(K+1,n) + P_0 \cdot P_K \cdot P_n \}$$

General Recurrence Rel of Matrix chain Problem

$A_i\ A_{i+1} \ldots\ldots A_j$ Matrices

$$C(i,j) = \begin{cases} \min_{i \le k < j} \{ C(i,k) + C(k+1,j) + P_{i-1} \cdot P_k \cdot P_j \}, & i < j \\ 0, & i = j \end{cases}$$



Eg ⇒ $A_1\ A_2\ A_3\ A_4$

$\langle P_0\ P_1\ P_2\ P_3\ P_4 \rangle = \langle 40, 50, 25, 60, 20 \rangle$

Soln $c(1,4) = ?$ // Find val.

Cost Array $c[][]$

	1	2	3	4
1	0	50,000 k=1	110000 k=2	95000 k=1
2	X	0	75000 k=2	55000 k=2
3	X	X	0	30,000 k=3
4	X	X	X	0

j

Find val

$|j-i| = 3$

$|j-i| = 2$

$|j-i| = 1$

Initial value

TC of matrix chain using DP $= \theta(n^3)$

SC of Matrix chain using DP

$c[1 \ldots n]\ [1 \ldots n]$ // to store $c(i,j)$

$k[1 \ldots n]\ [1 \ldots n]$ // to store k^{th} of min cost of $c(i,j)$

$= \theta(n^2)$

Min cost of Mul of $A_1\ A_2\ A_3\ A_4$ Matrices : 95000 [Scalar Multiplications]

⇒ Optimal paranthesization of $A_1\ A_2\ A_3\ A_4$ Matrix Mul.

k^{val} of $c[1,4] = 1$

$A_1 (A_2\ A_3\ A_4)$

$(A_2 (A_3\ A_4))$ ← k^{val} of $c[2,4] = 2$

$\Rightarrow A_1 (A_2 (A_3 \cdot A_4))$

(1) ⇒ #f Binary Search trees for n distinct key:-

[#f distinct key]	[#f BST's]
0	1
1	1
2	2
3	5
4	14

1 : K_1

2 : $K_1 < K_2$; $K_2 < K_1$

3 : $K_1 < K_2 K_3$

$[K_2 K_3]$ K_1 $[K_1 K_3]$ K_2 $[K_1 K_2]$ K_3

$1*2 + 1*1 + 2*1$

4 : $K_1 < K_2 < K_3 < K_4$

$K_2K_3K_4$ K_1 K_3K_4 K_1K_2 K_4 $K_1K_2K_3$

$1*5 + 1*2 + 2*1 + 5*1$

T(n): #f BST's for n distinct key.

K_1 K_2 K_3 .... K_{n-1} K_n

$$T(n) = \underset{K=1}{T(0)\cdot T(n-1)} + \underset{K=2}{T(1)T(n-2)} + \underset{K=3}{T(2)T(n-3)} + \ldots \underset{K=n}{T(n-1)\cdot T(0)}$$

$T(0) = 1$, $T(1) = 1$ } Initial value

$$T(n) = \begin{cases} 1 & , n \le 1 \\ \sum_{k=1}^{n} T(k-1) * T(n-k) & , n > 1 \end{cases}$$

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

#f BST's possible for n distinct key

{ #f paranthesization for n+1 matrices ≡ #f BST's for n distinct keys. }

(2). #f unlabled Binary Trees for n nodes.

#f nodes	#f unlabled BT's
1.	1
2	2
3	5
⋮	⋮
n	$\frac{1}{n+1}\left[\frac{(2n)!}{n!\cdot n!}\right]$

{ #f unlabled BT's for n nodes equal to #f BST's for n distinct keys }

(3). #f labled BT's for n keys.

#f keys	#f labled BT's.
1	1.
2	$2*2! = 4$
3	$5*3! = 30$
⋮	⋮
n	$\frac{1}{n+1}\left[\frac{(2n)!}{n!}\right]$

#f labled BT's for n keys ≡ n! [#f unlabled BT's of n nodes].

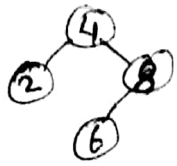
Q How many ways can label given Unabled BT of n nodes ++ n distinct keys · into Binary Search tree?

a) $n!$ b) $\frac{1}{n+1}\left[\frac{(2n)!}{n!}\right]$

c) $\frac{1}{n+1}\left[\frac{(2n)!}{n!n!}\right]$ d) 1.

[2, 4, 6, 8]

BST ⇒



Largest Common Subsequence

Sub Sequence :-

Sequence of char. from string which may not consecutive

X = "a b a d e"

Some Subsequences of X: abd, abde, aae, abade

String X with n char, ## possible Subsequence $= 2^n$

X = "a b c d" : 4 char $\Rightarrow 2^4 = 16$ possible sub-sequences of X

Common Subsequence :- Subsequence which is common for two String X & Y

X = abade } bd, bad, aad } common Subsequences for X & Y
Y = baada }

Largest common Subsequence of X, Y is of length 3

Largest common Subsequence (Statement)

Common Sub Sequence of string X & Y which is maximum length.

Recursive Solution of LCS :-

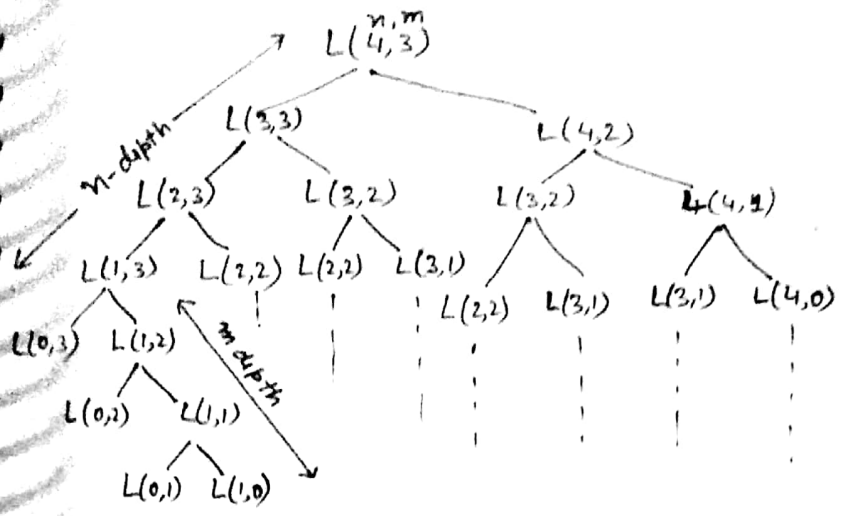
LCS (n, m) : Largest Common Subsequence length of two strings.
n & m char respectively.

$$X = [x_1\ x_2\ x_3\ \ldots\ x_{n-1}\ x_n]$$

$$Y = [y_1\ y_2\ y_3\ \ldots\ y_{m-1}\ y_m]$$

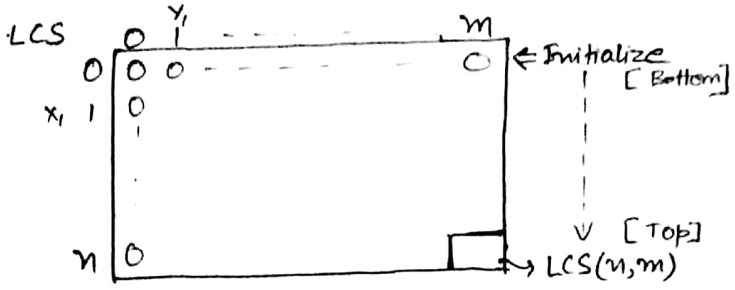
$$LCS(n,m) = \begin{cases} 0 & , n=0 \,||\, m=0 \\ 1 + LCS(n-1, m-1) & , n>0 \,\&\&\, m>0 \,\&\&\, x_n == y_m \\ \max\begin{cases} LCS(n-1, m) \\ LCS(n, m-1) \end{cases} & , n>0 \,\&\&\, m>0 \,\&\&\, x_n \neq y_m \end{cases}$$

$L(n,m) = ?$ Final value.



DP implementation of LCS :—

X & Y two strings of n & m char.



⇒ 1st Row & 1st column of LCS should initialize 0's

		j-1	j
X_{i-1}	i-1	L(i-1,j-1)	L(i-1,j)
X_i	i	L(i,j-1)	L(i,j)

if $(x_i == y_j)$
  $L(i,j) = 1 + L(i-1,j-1)$
else if $(L(i,j-1) \geq L(i-1,j))$
  $L(i,j) = L(i,j-1)$
else
  $L(i,j) = L(i-1,j)$

⇒ LCS array can compute either in row major order or in column major order.

$X = [a\ b\ c\ d\ e]$ $n=5$

$Y = [b\ c\ a\ e\ c\ d]$ $m=6$

		b	c	a	e	c	d
	0	0	0	0	0	0	0
a	1	0	0	0	1	1	1	1
b	2	0	1	1	1	1	1	1
c	3	0	1	2	2	2	2	2
d	4	0	1	2	2	2	2	3
e	5	0	1	2	2	3	3	3

final value

LCS(5,6) = 3

[b, c e] LCS of given string.

```
Algo LCS(X[], Y[], n, m)
{
O(m)  for(i=0; i<=m; i++)
      { L[0][i] = 0 }
O(n)  for(i=0; i<=n; i++)
      { L[i][0] = 0 }
      for(i=1; i<=n; i++)
      { for(j=1; j<=m; j++)
        { if(X[i] == Y[j])
          { L[i][j] = 1 + L[i-1][j-1]
            Path[i][j] = "↖";
          }
          else if(L[i][j-1] >= L[i-1][j])
          { L[i][j] = L[i][j-1]
            Path[i][j] = "←";
          }
          else
          { L[i][j] = L[i-1][j];
            path[i][j] = "↑"
          }
        }
      } return(L(n,m), path[][])
}
```

(n, m)

⇒ TC of LCS using DP ⇒ $\boxed{\theta(n*m)}$

⇒ SC of LCS using DP ⇒ $\boxed{\theta(n*m)}$

L[0..n, 0..m] ⇒ (n+1)(m+1)

path[1..n, 1..m] ⇒ n*m

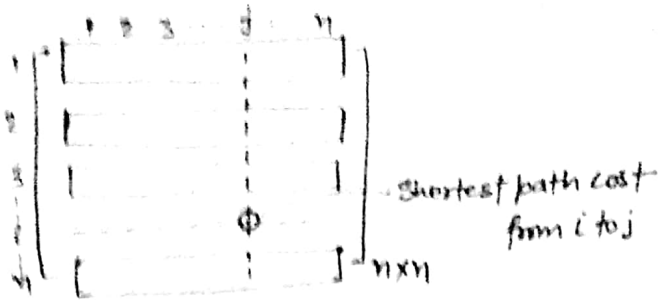
$\theta(n \cdot m)$

```
Algo Print(n, m)
{ if(n==0 || m==0)
  { return; }
  else
  { if(path[n][m] == "↖")
    { Print(n-1, m-1);
      Print("X[n]");
    }
    else if(Path[n][m] == "←")
    { Print(n, m-1); }
    else
    { Print(n-1, m); }
  }
}
```

TC → $\theta(n+m)$

SC → $\theta(n+m)$

TC to print LCS using patharray : $\theta(n+m)$



Greedy Algo [Dijkstras]

```
for (i=1; i<=n; i++)
{ A[i][1..n] = Dijkstras(G, n, e, i) }
                                  source
```

$TC = n(n+e)\log n$

$= \Theta(n^2 + n \cdot e \log n)$

DP All pair shortest path algo :-

$A^0_{ij} = cost(i,j)$ // Edge Cost
Initial val.

1) $A^1_{ij} \Rightarrow$ 

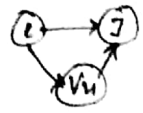
$A^1_{ij} = \min\{A^0_{ij}, A^0_{i1} + A^0_{1j}\}$

// Shortest path from ① to ⓙ going through ①

2) $A^2_{ij} \Rightarrow$ 

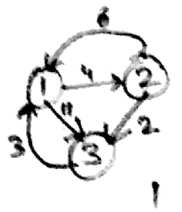
$A^2_{ij} = \min\{A^1_{ij}, A^1_{i2} + A^1_{2j}\}$

// Shortest path cost from ① to ⓙ going through ① and/or ②

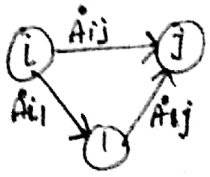
n) $A^n_{ij} = \min\{A^{n-1}_{ij}, A^{n-1}_{in} + A^{n-1}_{nj}\}$ 

final val

$$A^k_{ij} = \begin{cases} \min\limits_{1 \le k \le n} \{A^{k-1}_{ij}, A^{k-1}_{ik} + A^{k-1}_{kj}\} \\ A^0_{ij} = cost\,ij \quad \text{// Edge Cost} \end{cases}$$

Eg. 

$cost = \begin{matrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & \infty & 0 \end{matrix}$

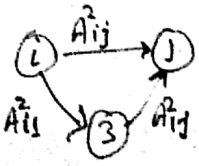


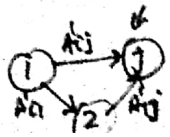
$A^0 = \begin{matrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & \infty & 0 \end{matrix}$

$A^1 = \begin{matrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{matrix}$,

$A^2 = \begin{matrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 6 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{matrix}$,

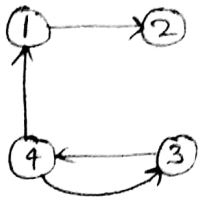
$A^3 = \begin{matrix} & 1 & 2 & 3 \\ 1 & 0 & 4 & 6 \\ 2 & 5 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{matrix}$ → Result of all pair shortest path.



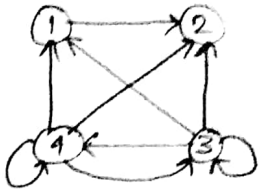


```
Algo APSP_FloydWarshals(n, cost[][])
{
    for(i=1; i≤n; i++)                    TC: θ(n³)
    {
θ(n²)   for(j=1; j≤n; j++)                SC: θ(n²)
        {
            Aij = cost ij;
        }
    }
    for(k=1; k≤n; k++)
    {
        for(i=1; i≤n; i++)
        {
θ(n³)       for(j=1; j≤n; j++)
            {
                if(Aij > Aik + Akj)
                    Aij = Aik + Akj;
            }
        }
    }
}
```

Transitive closure of graph:-



Graph [G]

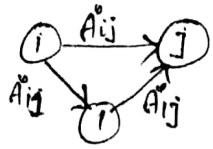


G': Transitive closure.

$$A \Rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$A^0_{ij} = Adj\ ij$

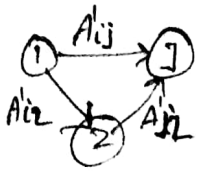
1) $A^1_{ij} \in$



$A^1_{ij} = A^0_{ij} \vee (A^0_{i1} \wedge A^0_{1j})$

A^1_{ij} is 1 ⇒ if there exist path from i to j going through ①

2) $A^2_{ij} \Rightarrow$



$A^2_{ij} = A^1_{ij} \vee (A^1_{i2} \wedge A^1_{2j})$

n) $A^n_{ij} = A^{n-1}_{ij} \vee (A^{n-1}_{in} \wedge A^{n-1}_{nj})$

```
Algo TCGFloydWarshall(n, Adj[][])
{
    for(i=1; i≤n; i++)
    {   for(j=1; j≤n; j++)
            Aij = Adj ij;
    }
    for(k=1; k≤n; k++)
    {   for(i=1; i≤n; i++)
        {   for(j=1; j≤n; j++)
            {
                Aij = Aij ∨ (Aik ∧ Akj)
            }
        }
    }
    return(A[][])
}.
```

0/1 Knap sack Problem

Given n obj && m, capacity knapsack && $P_1 \dots P_n$ profits
$w_1 \dots w_n$ weights

Maximize $\sum_{i=1}^{n} P_i * x_i$

Subjected to $\sum_{i=1}^{n} w_i * x_i \le m$

where $x_i = 1$ & 0

⇒ 0/1 knapsack Problem failed to solve using Greedy Method.

⇒ 0/1 knapsack Problem TC using Brute force Algo : $\theta(2^n)$

DP sol of 0/1 knapsack Problem

$V(n,m)$ = Max profit for n obj && m capacity knapsack

$V(n,m) = \max\{ V(n-1,m), V(n-1,m-w_n)+P_n \}$
$\leftarrow x_n=0 \rightarrow \quad \leftarrow x_n=1 \rightarrow$

$V(i,j)$: Max profit for i objects && j knapsack capacity.

$$V(i,j) = \begin{cases} 0 & , i=0 \,\&\&\, j \ge 0 \\ V(i-1,j) & , i>0 \,\&\&\, w_i > j \\ \max\{ V(i-1,j),\ V(i-1,j-w_i)+P_i \} & , i>0 \,\&\&\, w_i \le j \end{cases}$$

$(V(n,m) = \text{final val})$

Eg, $n=3 \quad m=6$

$\langle P_1 P_2 P_3 \rangle = \langle 20, 10, 15 \rangle$

$\langle w_1 w_2 w_3 \rangle \quad \langle 2, 1, 4 \rangle$

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	20	20	20	20	20
2	0	10	20	---->			
3			---->				final value.

. define $V[0 \dots n, 0 \dots m]$

. Initialize 1st Row 0's,

$V(1,0) = V(0,0) \quad w_1 > 1$

$V(1,2) = \max\{ V(0,2), V(0,0)+20 \}$
$\underset{20}{}$

$V(2,1) = \max\{ V(1,1), V(1,0)+10 \}$
$\underset{10}{}$

$V(2,2) = \max\{ \underbrace{V(1,2)}_{20}, \underbrace{V(1,1)+10}_{10} \}$
$\underset{20}{}$

```
Algo 0/1 Knapsack (n, m, w[1...n], P[1...n])
{ // V[0...n, 0...m] define array.
  for(i=0; i<=m; i++)
  { V[0][i] = 0 }
  for(i=1; i<=n; i++)
  { for(j=0; j<=m; j++)
    { if(wi > j)
        V[i][j] = V[i-1][j]; x[i] = 0
      else
        V[i][j] = max[V[i-1][j],
                      V[i-1][j-wi]+Pi]
    }
  } return (X[], V[n][m])
      obj Seque   max profit
}.
```

TC of 0/1 knapsack using DP $= \theta(n \cdot m)$

SC of 0/1 knap Sack using DP $= \theta(n \cdot m)$

Sum of Subset problem (SOS):—

$[a_1 a_2 a_3 \ldots a_n]$: n integers.

Test any subset of given n Integers.
Sum equal to "m" is exists / notexists.

Ex ⇒ n=4

$\{a_1\ a_2\ a_3\ a_4\}$

{5, 4, 8, 6}. && m=16

SOS(n,m) = false.

Ex $\{a_1 a_2 a_3 a_4\} = \{5, 4, 8, 6\}$

m = 15

SOS(n,m) = True.

---

1) Greedy method fails to solve SOS problem

2) Brute force Algo to solve SOS problem
$\theta(2^n)$ Time Comp.

3) DP Sol of SOS:—

SOS(n,m) = True / false
any subset of n elmnts
Sum equal to m exists / not exists

$SOS(n,m) = SOS(n-1,m) \vee (SOS(n-1), m-a_n) \wedge m > a_n)$

$$SOS(i,j) = \begin{cases} True \ldots\ldots, & j=0 \ \&\& \ i \geq 0 \\ false \ldots\ldots, & j>0 \ \&\& \ i=0 \\ SOS(i-1,j), & j < a_i \\ SOS(i-1,j) \vee (SOS(i-1, j-a) \wedge j \geq a_i) \end{cases}$$

Eg DP

n=3 m=7.

$[a_1 a_2 a_3] = [3, 2, 5]$

	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F
1	T	F	F	T	F	F	F	F
2	T	F	T	T	F	T	F	F
3	T	F	T	T	F	T	F	T

- First column initialize : True

First row SOS[0][1...n] is false

SOS(1,2) = SOS(0,2) j<ai

~~SOS(2,2) = SOS(1,2~~

TC of SOS using DP $\theta(n \cdot m)$

SC of SOS using DP : $\theta(n \cdot m)$

```
Algo SOS(n,m, a[1..n])
{
  for(i=0; i<=n; i++)
      SOS[i][0] = True
  for(i=1; i<=m; i++)
   { SOS[0][i] = false }
  for (j=0; j<=m; j++)
   { if(ai > j)
       SOS[i][j] = SOS[i-1][j];
     else
       SOS[i][j] = SOS[i-1][j] V SOS[i-1][j-ai]
   }
 }return(SOS[n][m])
}.
```