Algorithm

#) Analysis :

*) TC = CT + RT

CT → compiler → Lang.

RT → CPU → H/w

*) Apostriory
- Depends on compiler lang.

Apriory
- Independent of compiler lang.
- Determin.n of order of magnitude of a stmt.

*) Ex
```
while (i<n)
{ i = i+1
}
```
$n + n-1 = \Theta(n)$

*) Updation type | TC

1. $i = i \pm 20$	$i/20$
2. $i = i * / 20$	$\lg_{20} i$
3. $i = i^2$ or $i^{1/2}$ (where init $i = 3$)	$\lg_2 \lg_3 i$ i.e. Up^n. Init.^{n}

*) Note: In 1. if i goes till n^2 then TC $n^2/20$

*) Note: $\lg_2 \lg_3 n^2 = \lg_2 (2 \lg_3 n)$
$= \lg_2 2 + \lg_2 \lg_3 n$
$= \Theta(\lg_2 \lg_3 n)$

*) Note: Sometimes cond.^{n} itself is init.^{n}. Ex:- while(n>2)
```
{ n = n^(1/3)
}
```
$\Theta(\lg_3 \lg_2 n)$

*) Ex
```
i=5
while(i<n)
{ i = i+2
  i = i*2
  i = i^2
}
```
$((i+2) \times 2)^2$
$(2i+4)^2$
$= i^2$
$\rightarrow \Theta(\lg_2 \lg_5 n)$

*) Ex
```
for (k=5; k<n^7; k=k^3)
{ pf(hi);
}
```
$= (\lg_3 \lg_5 n^7)$
$= \lg_3 (7 \lg_5 n)$
$= \Theta(\lg_3 \lg_5 n)$

*) Inner loop dependency

Ex
```
for (i=1; i≤n; i++) → n
  for(j=1; j≤i; j++) → i
```

value of i:	1	2	3	... n
TC of j loop:	1	2	3	... n

overall TC = Σ TC of j loop
$= \Theta(n^2)$

Ex
```
for (i=1; i≤n; i++) → n
  for(j=1; j≤n; j=j+i) → n/i
```

i:	1	2	...	n
TC j:	n/1	n/2	...	n/n

overall TC $= n\left[\frac{1}{1} + \frac{1}{2} + \cdots \frac{1}{n}\right]$
$= \Theta(n \lg_2 n)$

*) Nested loop on same var.
```
for(i=1; i≤n; i++) ⇒ 1
  for(i=1; i≤n^2; i++) ⇒ 1
    for(i=1; i≤n^3; i++) ⇒ n^3
      {pf(Hi)}
```
$\Theta(n^3)$

*) Ex
```
i=1, j=1
while(i≤n)
{ j++;
  i=i+j;
}
```
i: 1, 3, 6 ; j: 1, 2, 3

$1 \le n$
$1+2 \le n$
$1+2+3 \le n$
...
$1+2+3+\cdots k \le n$
$\frac{k(k+1)}{2} \le n$
$k^2 \le n$
$k = \sqrt{n}$
$\Theta(\sqrt{n})$

*) Ex
```
for(i=1; i≤n; i++) ⇒ n
  {if(n % i==0) ⇒ 2
    for(j=1; j≤n; j++) ⇒ n
      {pf(hi)}
  }
```
n = Prime no.

i=1	i=2		= 3	⋮ = n
j=1	j=1, 2		= 1, 2, 3	⋮ = 1, 2, ... n
k=1	k=1, 1, 2		= 1, 1, 2, 1, 2, 3	⋮ = 1, 1, 2
1	2		3	

$TC = 1 + (1+2) + (1+2+3) + \cdots (1+2+3+\cdots n)$
$= 1 + (1+2) + (1+2+3+ \cdots \frac{n(n+1)}{2}$
$= \sum_{i=1}^{n} \frac{i(i+1)}{2}$
$= \frac{1}{2}\left[\sum i^2 + \sum i\right]$
$= \frac{1}{2}\left[\frac{1}{6} n(n+1)(2n+1) + \frac{n(n+1)}{2}\right]$
$= n^3$
$= \Theta(n^3)$

i=1	$\Theta(1)$	$\Theta(1)$	j=1,2...n

$n + (n-1) \times 1 + n$
$= \Theta(n)$

*) Ex ... same
n = anything

BC = $\Theta(n)$ [n = Prime]
WC = $\Theta(n^2)$ [n = not Prime]
$O(n^2)$, $\Omega(n)$

*) Ex
```
for(i=2; i≤√n; i++)
{ if(n % i==0)
  {pf(not prime)
   return(0)
  }
}
```
$O(\sqrt{n})$ [n ∉ Prime]
$\Omega(1)$ [n ∈ Prime]

*) Ex
```
for(i=1; i≤n; i++)
  for(j=1; j≤i; j++)
    for(k=1; k≤j; k++)
      pf(Hi);
```

*) Ex for (i=n, j=1; i>1; i/=2, j+=i)

Termination condⁿ on = i TC = ?
j = ?

Initial i value = n
then n/2
then n/4
...
n/n = 1

$\Rightarrow TC = \Theta(\lg_2 n)$

Initial j value = 1
then 1+n
then $1+n+\frac{n}{2}$
then $1+n+\frac{n}{2}+\frac{n}{4}$
...

$1+n+\frac{n}{2}+\frac{n}{2^2}+\cdots\frac{n}{2^{\lg_2 n}}$

$1+n\left(1+\frac{1}{2}+\frac{1}{2^2}+\cdots\frac{1}{2^{\lg_2 n}}\right)$

1+n (Decreasing GP)

$1+n\times\Theta(1)$

Final value of j $\Rightarrow \Theta(n)$

*) Asymptotic notation

•) Big OH: Right will be greater or equal with the help of C from n_0 onwards.

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

$f(n) \le c\cdot g(n);\ \forall n;\ n \ge n_0$

Two +ve const. → $c > 0$ (R), $n_0 \ge 1$ (I)

Ex $f(n) = n+10$; $g(n) = n$; $f(n) = O(g(n))$

$n+10 \le \underset{2}{c}\cdot n$ $n_0 = 10$

$\therefore n+10 \le 2n;\ \forall n,\ n \ge 10$

$n+10 = O(n)$

•) Omega: $f(n) = \Omega(g(n))$

$f(n) \ge c\cdot g(n);\ \forall n;\ n \ge n_0$

Two +ve const → same...

•) Theta: $f(n) = \Theta(g(n))$

i) $f(n) \le c_1 g(n)$; $\forall n;\ n \ge n_0$

ii) $f(n) \ge c_2 g(n)$

Note:) If Θ possible, there's no chance that o & ω are possible

Ex $f(n) = n+10$; $g(n) = n$

i) $n+10 \le \underset{2}{c_1} n$ | $n_0 = 10$

ii) $n+10 \ge \underset{1}{c_2} n$ | $n_0 = 10$

$\Rightarrow n+10 = \Theta(n)$; $c_1 = 2$, $c_2 = 1$, $n_0 = 10$

•) Note: ω ✓ ⇒ Ω ✓
Ω ✓ ⇏ ω ✓

•) Note: Bigger fxⁿs equal ⇒ Θ possible
$f(n) = n^2$
$g(n) = n^2+n+10$
$\Rightarrow f(n) = \Theta(g(n))$

•) Note:
n^2 → TUB = n^2
→ Non-TUB = n^3, n^{100}

O ⇒ upper bound (TUB or NTUB)
o ⇒ upper bound (NTUB)

n^2 → TLB = n^2
→ NTLB = $n, 1$

Ω ⇒ Lower bound (TLB, NTLB)
ω ⇒ Lower bound (NTLB)

n^2 → TLB = n^2
→ TUB = n^2 ⇒ $\Theta(n^2)$

Note:) $n^n > n! > 2^n > n^{\lg n}$
but $\lg(n!) = \Theta(n\lg n)$

*) Complexity classes :-
① Decreasing fxⁿs $\left(\frac{1}{n}, \frac{1}{n^2}, \cdots\right)$ $< \Theta(1)$
② constant fxⁿ (10crore, 1^n) $= \Theta(1)$
③ Logarithmic fxⁿ ($\lg n$, $(\lg n)^2$ ...)
④ Linear (n)
⑤ Quadratic (n^2)
⑥ Cubic (n^3)
⑦ Polynomial (n^c ; c = const & c > 0)
$[n^{0.1}, n^{0.5}, n^{200}, n^2, n^3]$
⑧ Exponential (c^n ; c = const. & c > 1)
$[2^n, 3^n, 5.6^n, \cancel{1^n}, \cancel{0.5^n}, \cancel{n^n}]$

•) $\lg\lg n < \lg n < \sqrt{n} < n$ •) $4^n > 2^n$
•) $n < n\lg n < n\sqrt{n} < n^2$ •) $a^{\lg_b n} = n^{\lg_b a}$
•) $n > (\lg n)^c$ but $n < (\lg n)^{\lg n}$ & $n < (\lg n)^n$

CamScanner

•> $\lg_2 n > \lg_3 n$

but asym. both equal

•> Ex: $f(n) = n^{1-\cos n}$

$g(n) = n^{4+\sin n}$

Rel.^{n} b/w $f(n)$ & $g(n)$?

n	$f(n)$	$g(n)$
0	0	4
90	1	5
180	2	3
270	1	4
360	0	5

$f(n) \to 0-2$, $g(n) \to 3-5$] => $f(n)$ & $g(n)$ are comparable, $f(n) = O(g(n))$

Note: If $f(n) \to 0-2$, $g(n) \to 2-4$] => Not comparable

•> $n! = O(n^n)$

$\lg(n!) = O(n \lg n)$

•> TC of Rec = TC of non-Rec

•> Note: Reflexive $\to O\ \Omega\ \Theta$

Transitive $\to o\ O\ \omega\ \Omega\ \Theta$

Symmetric $\to \Theta$

•> Note:

→ In recursion from one fx^n call to another only parameter values will change, but not other things.

→ one recursive program may have more than one recurrence relation

*> Recurrence relation solving:

1] Substitution (use when 1 distinct fx^n call is there)

Ex:) $T(n) = \begin{cases} 1 & ; n=1 \\ T(n-1)+n & ; n>1 \end{cases}$

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

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

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

$\vdots\ K$

$= T(n-K) + [n-(K-1)] + \dots + n$

$n-K = 1 \Rightarrow K = n-1$

$= T(1) + 2 + 3 + \dots + n$

$= 1 + 2 + \dots n$

$= O(n^2)$

Ex: $T(n) = \begin{cases} T(n-2) + \lg n & ; n>0 \\ 2 & ; n=0 \end{cases}$

$T(n) = T(n-2) + \lg n$

$= T(n-4) + \lg(n-2) + \lg n$

$= T(n-6) + \lg(n-4) + \lg(n-2) + \lg(n-0)$

$\vdots\ K$

$= T(n-2K) + \lg[n-(2K-2)] + \lg[n-(2K-1)] + \dots \lg n$

$n-2K = 0 \Rightarrow n = 2K \Rightarrow K = n/2$

$= T(0) + \lg(n-(n-2)) + \lg(n-(n-4)) + \dots \lg n$

$= 2 + \lg 2 + \lg 4 + \lg 6 + \dots \lg n$

$= 2 + \lg(2\times1) + \lg(2\times2) + \lg(2\times3) + \dots \lg(2\times n/2)$

$= 2 + (\lg 2 + \lg 1) + (\lg 2 + \lg 2) + (\lg 2 + \lg 3) + \dots (\lg 2 + \lg n/2)$

$= 2 + \frac{n}{2} \cdot \lg_2 2 + (\lg[1\cdot2\cdot3\dots n/2])$

$= 2 + \frac{n}{2} + \lg[(n/2)!] = n/2 \lg n/2$

$= \Theta(n \lg n)$

Ex: $T(n) = \begin{cases} T(n-2) + n^2 & ; n>0 \\ 2 & ; n=0 \end{cases}$

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

$= T(n-4) + (n-2)^2 + n^2$

$= T(n-6) + (n-4)^2 + (n-2)^2 + n^2$

$\vdots\ K$

$= T[n-2K] + [n-(2K-2)]^2 + [n-(2K-4)]^2 + \dots + n$

$n-2K = 0 \Rightarrow K = n/2$

$= T(0) + [n-(n-2)]^2 + [n-(n-4)]^2 + \dots n^2$

$= 2 + 2^2 + 4^2 + 6^2 + \dots n^2$

$= 2 + 2^2 [1^2 + 2^2 + 3^2 + \dots (n/2)^2]$

$= 2 + 2^2 [\frac{1}{6}(n/2)(n/2+1)(n+1)]$

$= \Theta(n^3)$

Note: If in option you've O, Ω, Θ all n^3 then select Θ for mcq, & all for msq

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

$$T(n) = T(n/2) + n$$
$$= T(n/2^2) + n/2 + n$$
$$= T(n/2^3) + n/2^2 + n/2 + n/2^0$$
$$\vdots\ k$$
$$= T(n/2^k) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} + \cdots \frac{n}{2^0}$$

$1 = n/2^k \Rightarrow k = \lg_2 n$

$$= T\left(\frac{n}{2^{\lg_2 n}}\right) + \frac{n}{2^{\lg_2 n - 1}} + \frac{n}{2^{\lg_2 n - 2}} + \cdots \frac{n}{2^0}$$
$$= 1 + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \cdots \frac{1}{2^{\lg_2 n - 1}}\right]$$
$$= 1 + n \times \Theta(1)$$
$$= \Theta(n)$$

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

$$T(n) = 7T(n/2) + n^2$$
$$= 7\left[7T(n/2^2) + (n/2)^2\right] + n^2$$
$$\hookrightarrow 7^2 T(n/2^2) + \frac{7}{4}n^2 + n^2$$
$$= 7^2\left[7T(n/2^3) + (n/2^2)^2\right] + \frac{7}{4}n^2 + n^2$$
$$\hookrightarrow 7^3 T(n/2^3) + \left(\frac{7}{4}\right)^2 n^2 + \left(\frac{7}{4}\right)^1 n^2 + \left(\frac{7}{4}\right)^0 n^2$$
$$\vdots\ k$$
$$= 7^k T(n/2^k) + n^2\left\{\left(\frac{7}{4}\right)^0 + \left(\frac{7}{4}\right)^1 + \cdots + \left(\frac{7}{4}\right)^{k-1}\right\}$$
$$\hookrightarrow 7^k T(n/2^k) + n^2\left[\frac{1\left((7/4)^k - 1\right)}{(7/4 - 1)}\right]$$

$\frac{n}{2^k} = 1 \Rightarrow k = \lg_2 n$

$$= 7^{\lg_2 n} \cdot 1 + n^2\left[\frac{(7/4)^{\lg_2 n} - 1}{(7/4 - 1)}\right]$$
$$= 7^{\lg_2 n} + n^2 \frac{7^{\lg_2 n}}{4^{\lg_2 n}}$$
$$= n^{\lg_2 7} + n^2 \cdot \frac{n^{\lg_2 7}}{n^2} = \Theta(n^{2.81})$$

EX: $T(n) = \begin{cases} 2T(n/2) + n\lg n & ; n > 2 \\ 2 & ; n = 2 \end{cases}$

$$T(n) = 2T(n/2) + n\lg n$$
$$= 2\left[2T(n/2^2) + n/2 \lg n/2\right] + n\lg n$$
$$\hookrightarrow 2^2 T(n/2^2) + n\lg\frac{n}{2} + n\lg n$$
$$= 2^2\left[2T(n/2^3) + n/2^2 \lg n/2^2\right] + n\lg\frac{n}{2} + n\lg n$$
$$\hookrightarrow 2^3 T(n/2^3) + n\lg\frac{n}{2^2} + n\lg\frac{n}{2} + n\lg\frac{n}{2^0}$$
$$\hookrightarrow 2^3 T(n/2^3) + n\left[\lg\frac{n}{2^2} + \lg\frac{n}{2^1} + \lg\frac{n}{2^0}\right]$$
$$\vdots\ k$$
$$= 2^k T(n/2^k) + n\left[\lg\frac{n}{2^{k-1}} + \lg\frac{n}{2^{k-2}} + \cdots \lg\frac{n}{2^0}\right]$$

$\frac{n}{2^k} = 2 \Rightarrow n = 2^{k+1} \Rightarrow k = \lg_2 n - 1$

$$= 2^{\lg_2 n - 1} \cdot T\left(n/2^{\lg_2 n - 1}\right) + n\left[\lg\frac{n}{2^0} + \lg\frac{n}{2^1} + \cdots \lg\frac{n}{2^{\lg n - 1}}\right]$$
$$= \frac{2^{\lg_2 n}}{2} \cdot 2 + n\left[(\lg n - \lg 2^0) + (\lg n - \lg 2^1) + \cdots + (\lg n - \lg 2^{\lg n - 1})\right]$$
$$= 2^{\lg_2 n} + n\left[(\lg n \times \lg n) + (0 + 1 + 2 + \cdots \lg_n - 1)\right]$$
$$= n + n\left[\lg n \cdot \lg n + \frac{(\lg_n)(\lg_n + 1)}{2}\right]$$
$$= \Theta\left[n(\lg n)^2\right]$$

EX: $T(n) = \begin{cases} \sqrt{n} \cdot T(\sqrt{n}) + n & ; n > 2 \\ 2 & ; n = 2 \end{cases}$

$$T(n) = n^{1/2} T(n^{1/2}) + n$$
$$= n^{1/2}\left[n^{1/4} T(n^{1/4}) + n^{1/2}\right] + n$$
$$\hookrightarrow n^{3/4} T(n^{1/4}) + n + n$$
$$= n^{3/4}\left[n^{1/8} T(n^{1/8}) + n^{1/4}\right] + n + n$$
$$\hookrightarrow n^{7/8} T(n^{1/8}) + n + n + n$$
$$\hookrightarrow n^{\frac{2^3 - 1}{2^3}} T(n^{1/2^3}) + 3n$$
$$\vdots\ k$$
$$= n^{\frac{2^k - 1}{2^k}} T(n^{1/2^k}) + kn$$

$n^{1/2^k} = 2 \Rightarrow \frac{1}{2^k}\lg n = 1 \Rightarrow \lg n = 2^k \Rightarrow k = \lg_2 \lg_2 n$

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

$$= n^{\left(1-\frac{1}{2^{\lg\lg n}}\right)} \cdot 2 + n\lg\lg n$$

$$= \frac{n}{n^{\frac{1}{2^{\lg\lg n}}}} \cdot 2 + n\lg\lg n$$

$$= \frac{n}{n^{1/2}} \cdot 2 + n\lg\lg n$$

$$= \frac{n}{2} \cdot 2 + n\lg\lg n$$

$$= \Theta(n\lg\lg n)$$

Ex: $T(n) = \begin{cases} 2T(n-1)+n; & n>1 \\ 1 & ; n=1 \end{cases}$

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

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

$$\hookrightarrow 2^2T(n-2)+2(n-1)+2^0(n-0)$$

$$= 2^2[2T(n-3)+(n-2)]+2^1(n-1)+2^0(n-0)$$

$$\hookrightarrow 2^3T(n-3)+2^2(n-2)+2^1(n-1)+2^0(n-0)$$

$$\vdots\ k$$

$$= 2^kT(n-k)+2^{k-1}[n-(k-1)]+\cdots 2^0[n-0]$$

$n-k=1 \Rightarrow k=n-1$

$$= 2^{n-1}T(1)+2^{n-2}[n-(n-2)]+2^{n-3}[n-(n-3)]+\cdots 2^0[n-0]$$

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

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

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

$$\frac{x}{2} = 0+\frac{1}{2^2}+\frac{2}{2^3}+\cdots\frac{(n-1)}{2^n}$$

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

$$= \Theta(1)$$

$$\Rightarrow x = \Theta(1)$$

$$= 2^n \cdot \Theta(1) \Rightarrow O(2^n)$$

Note: constants of non recursive terms can be deleted.

If for $T(n)=2T(n/2)+n$ TC is $\Theta(n\lg n)$ then TC is same for

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

$$= 2T(n/2)+n+50$$

$$= 2T(n/2-10)+n$$

$$= 2T(n/2+10)+n$$

$$= 2T(\lfloor n/2\rfloor)+n$$

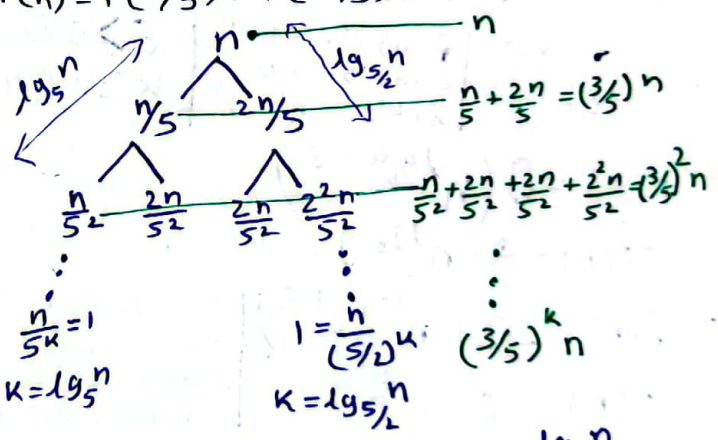
$$= 2T(\lceil n/2\rceil)+n$$

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

फालतू delete करो & solve करो

2] Recursive Tree (use when more than one distinct fx^n call)

Ex: $T(n)=T(n/5)+T(2n/5)+n$



UB: $T(n) \le (3/5)^0 n+(3/5)^1 n+\cdots(3/5)^{\lg_{5/2} n} n$

$n[(3/5)^0+(3/5)^1+\cdots(3/5)^{\lg_{5/2} n}]$

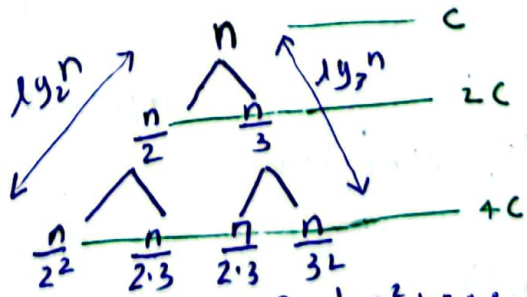
$nc \Rightarrow O(n)$

LB: $T(n) \ge n[(3/5)^0+(3/5)^1+\cdots(3/5)^{\lg_5 n}]$

$nc \Rightarrow \Omega(n)$

$\Rightarrow \Theta(n)$

Ex: $T(n)=T(n/2)+T(n/3)+c$



UB: $T(n) \le c[2^0+2^1+2^2+\cdots 2^{\lg_2 n}]$

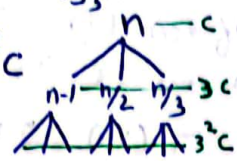
LB: $T(n) \ge c[2^0+2^1+2^2+\cdots 2^{\lg_3 n}]$

CamScanner

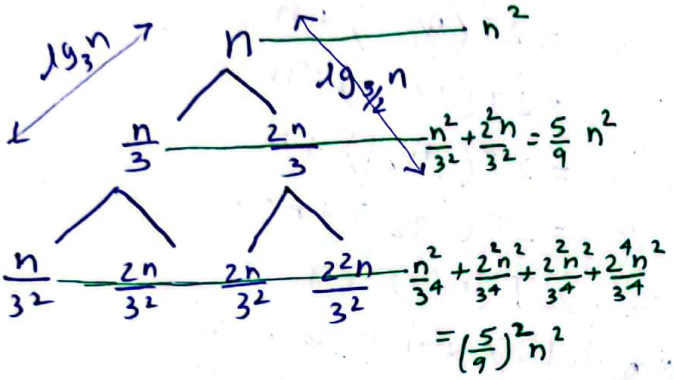
Ex: $T(n) = T(n-1) + T(n/2) + T(n/3) + c$

$K = n$ ($\lg_2 n$, $\lg_3 n$)

Cost $= 3^0 c + 3^1 c + 3^2 c + \dots 3^K c$



Ex: $T(n) = T(n/3) + T(2n/3) + n^2$



3] Master Theorem: (θ (जो भी बड़ा हो))

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

a, b ∈ const. $a, b \geq 1$ | $f(n)$ ∈ +ve fx^n, can't be −ve like $-n^2$ | $\varepsilon > 0$ & const. $K \geq 0$ & const.

I) $f(n) = O(n^{\lg_b a - \varepsilon})$

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

II) $f(n) = \Omega(n^{\lg_b a + \varepsilon})$ | $a f(n/b) \leq c \cdot f(n)$, $0 < c < 1$ [99% this cond.^{n} always satisfied]

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

III) $f(n) = \Theta(n^{\lg_b a} \cdot (\lg n)^K)$; $K \geq 0$

$T(n) = \Theta(n^{\lg_b a} \cdot (\lg n)^{K+1})$

I & II ⇒ Polynomial time small or big

III ⇒ logarithmic " small

•) Ex: $T(n) = T(n/2) + c$

$f(n)$	$a=1$ $b=2$
c	$n^{\lg_2 1}$
c	c

Apply case 3

$f(n) = \Theta(n^{\lg_2 1} \cdot (\lg n)^K)$; $K=0$

$c = \Theta(c)$

$\Rightarrow T(n) = \Theta(\lg n^1)$

Note: $n \lg n$ | n ⇒ Logarithmic time smaller

$n^4 \lg n$ | n^3 ⇒ $n \lg n$ ⇒ Polynomial time smaller

$\lg n$ | n ⇒ Polynomial time smaller

•) Ex: (Root wala)

$T(n) = T(\sqrt{n}) + \lg_2 n$

let, $n = 2^K \Rightarrow K = \lg_2 n$

$T(2^K) = T(2^{K/2}) + K$

let, $T(2^K) = S(K)$

$S(K) = S(K/2) + K$

$f(K)$	$a=1$ $b=2$
K	$K^{\lg_2 1}$
$K \not> 1$	

$\Rightarrow S(K) = \Theta(K)$

$T(2^{\lg_2 n}) = \Theta(\lg_2 n)$

$T(n) = \Theta(\lg_2 n)$

*) Recursive program TC:

Ex: fact(n)

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

c; $T(n-1)$

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

$= T(n-K) + Kc$ (K)

$n - K = 1 \Rightarrow K = n-1 \Rightarrow SC = \Theta(n)$

$= T(1) + (n-1)c = \Theta(n)$

Ex: fib(n)

```
{ if (n==0 || n==1) return(n)
  else return(fib(n-1) + fib(n-2))
}
```

c; $T(n-1)$, $T(n-2)$

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

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

$\Rightarrow SC = \Theta(n)$

$TC = O(2^n)$

Ex: A(n)

```
{ if(n≤1) return(n^10) → c
  else{
      for(i=1; i≤4; i++) → 4T(n/2)
          A(n/2)
      for(j=1; j≤n^2; j++) → n^2
          x = y+z
      ret(5A(n/2) * 3A(n/2) + LS(n))   [c]
      }   [T(n/2)  T(n/2)  n]
  }
```

$$T(n) = \begin{cases} c & ; n \le 1 \\ 6T(n/2) + n^2 & ; n > 1 \end{cases}$$

•) GCD of two numbers:

→ GCD(m, n) ⇒ 1. n % m = x
[where m < n] 2. GCD(x, m)

→ Rec. Program

```
GCD(m,n){
    if(m==0) ret(n)
    if(n==0) ret(m)
    ret(GCD(n%m, m))
}
```

→ BC: O(1) [m & n are multiple of one another]

GCD(50, 100 000)

1000 00% 50 = 0

GCD(0, 50) ⇒ 50

→ AC: O(lgn)

→ WC: O(lgn) [m & n are consecutive fibonacci no.s]

GCD(8, 13)
13 % 8 = 5
GCD(5, 8)
8 % 5 = 3
GCD(3, 5)
5 % 3 = 2
GCD(2, 3)
3 % 2 = 1
GCD(1, 2)
2 % 1 = 0
GCD(0, 1) ⇒ 1

For WC, $T(n) = T(n/2) + c$

$c \mid n^{\lg 1}$

$c \neq c$

$c = \Theta(c \cdot (\lg n)^k); k = 0$

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

→ Actually

$GCD(m,n) = O(\lg n)$
$\Omega(1)$

⇒ Divide & Conquer:

*) DAC Template:

```
DAC(a,i,j) → T(n)
{ if(Small(a,i,j)) → c
      return(Solution(a,i,j))
  else{
      k = Divide(a,i,j) → f1(n)
      S1 = DAC(a,i,k) → T(n/2)
      S2 = DAC(a,k+1,j) → T(n/2)
      S = Combine(S1,S2) → f2(n)
      return(S)
  }
}
```

} General code

Small(){..} Divide{..}
Combine{..} Solution{..}

} Specific codes

$$T(n) = \begin{cases} c & ; n = \text{small} \\ \underbrace{2}_{\text{no. of subproblem}} \overbrace{T(\underbrace{n/2}_{\text{size of each subproblem}})}^{\text{Time of one subproblem}} + \underbrace{f(n)}_{\text{Time for dividing \& combining}} & ; n = \text{big} \end{cases}$$

In General,
$T(n) = aT(n/b) + f(n)$; $a \ge 1, b > 1$

*) DAC Appⁿs:

1. Finding maxᵐ & minᵐ
2. Power of an element
3. Binary search
4. Merge Sort
5. Counting no. of inversions
6. Quicksort
7. Strassen's matrix multiplication
8. Maxᵐ continuous subarray sum
9. Selection procedure

maxᵐ - minᵐ
a^n
BS
QS
MS
SP
SMM
MCSS
inv.

*) Finding max.m & min.m :

→ Straight min-Max (used in BC)

```
f(a, i, j)
{ if(n==1) {max = min = a[n]
            return(max, min)}

  if(n==2) { max = bigger one
             min = smaller one
             return(max, min)}

  else
  { max = min = a[0]
    for(i=1; i≤n-1; i++)
    { if(max < a[i]){max=a[i]}
      if(min > a[i]){min=a[i]}
    }  TC: Θ(n)
  }
}
```

when array sorted in Asc. order

desc. order

comparisons: BC → (n-1), WC → 2(n-1), AC → 1.5(n-1)

a[50 70 60 45 35 25 75 12]
0 1 2 3 4 5 6 7
i

max = ~~50~~ ~~70~~ 75
min = ~~50~~ ~~45~~ ~~35~~ ~~25~~ 12

→ DAC min-max (used in WC)

```
f(a, i, j)
{ if(i==j) { max = min = a[i]
             return (max, min)}

  if(i==j-1) { max = bigger one
               min = smaller one
               ret(max, min)
  }

  else{ k = ⌊(i+j)/2⌋ // Divide → C

  (max1, min1) = f(a, i, k)
  (max2, min2) = f(a, k+1, j)   Conq. → 2T(n/2)

  if(max1 < max2) max = max2
                  else max = max1
  if(min1 > min2) min = min2
                  else min = min1   Comb. → C

  ret(max, min)
}}
```

TC: $T(n) = \begin{cases} c & ; n \le 2 \\ 2T(n/2) + c & ; n > 2 \end{cases}$

$\lg n$

$= \Theta(n)$

SC: $\Theta(\lg n)$

comparisons:

$T(n) = \begin{cases} 0 & ; n = 1 \\ 1 & ; n = 2 \\ \underbrace{0}_{\text{Div.}} + \underbrace{2T(n/2)}_{\text{conq.}} + \underbrace{2}_{\text{comb.}} & ; n > 2 \end{cases}$

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

$= \Theta(n)$

Note:) comparisons at top level $= 2^1$

" " 2nd " $= 2^2$

" " $\lg_2 n - 1$ " $= 2^{\lg_2 n - 1}$

Note:) Straight min max better cuz. same TC & using no stack space

Q:) 150 elements, finding min.m - max.m

i) Best case how many comparisons?

ii) Max.m how many comparisons?

i) ⇒ BC ⇒ $n-1$ ⇒ 149

ii) ⇒ WC ⇒ $\frac{3n}{2} - 2$ ⇒ 223

Q:) Finding neither 1st max.m nor 1st min.m in A[n]

i) All elements distinct ⇒ $\Theta(1)$ [3 comparison]

ii) All elements not distinct ⇒ $\Theta(n)$ [one scan needed]

Q:) _____ 2nd max.m nor 2nd min.m in A[n]

Take any 5 elements [ξ ξ (ξ) ξ ξ]

↓ sort

↓ middle ⇒ $\Theta(1)$

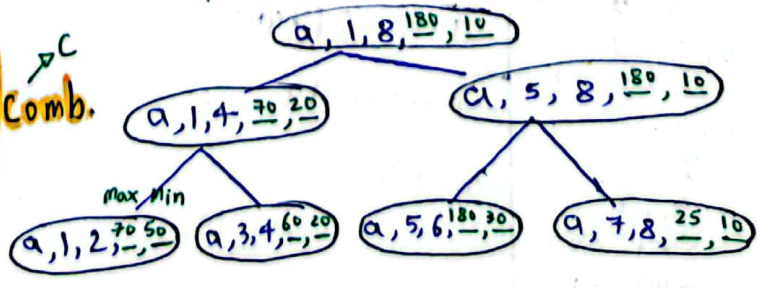
Q:) _____ k^{th} max.m nor k^{th} min.m in A[n]

Take 2k+1 elements [ξ ξ ξ (ξ) ξ ξ ξ] (k, k)

↓ sort

↓ k^{th} element ⇒ $\Theta(k \lg k)$

a[50 70 20 60 30 180 10 25]
1 2 3 4 5 6 7 8



*) Power of an element:

→ DAC Power of an element:

```
f(a,n)
{ if(n==1) {return (a)}   → c
  else{
      K = ⌊n/2⌋            → c
      S1 = f(a,K)          → T(n/2)
      S = S1*S1            → c
      if (n%2==0) //even power
          return (S)
      else //odd power
          return (a*S)
  }
}
```

TC: $T(n) = \begin{cases} c & ; n=1 \\ T(n/2)+c & ; n>1 \end{cases}$

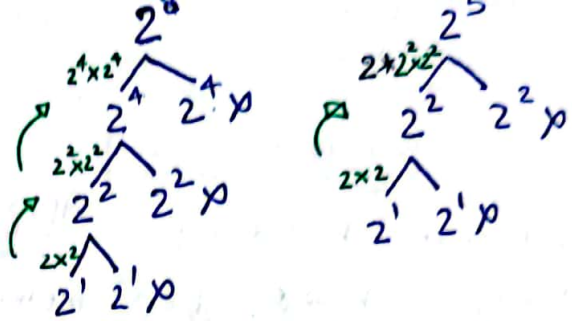
$= \Theta(\lg_2 n)$

SC: $\Theta(\lg_2 n)$

Multiplications:

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

$= \Theta(\lg_2 n)$



→ Non recursive Power of an element

```
s=1
for(i=1; i≤n; i++)
    s=s*a
ret(s)
= Θ(n)
```

or

```
s=a
for(i=1; i≤n; i=2*i)
    s=s*s
ret(s)
= Θ(lg2 n)
```

*) Linear Search: (Best algo when array unsorted)

```
LS(A[n], x)
{ for(i=0; i<n; i++)
    { if(A[i]==x)
          ret(i)
    }
  ret(-1)
}
```

Return position 'i' if x is found o/w '-1'

$T(n)$ (TC) $\begin{cases} BC = \Theta(1) \\ AC = \Theta(\frac{n+1}{2}) \\ WC = \Theta(n) \end{cases}$

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

$T(n) = \frac{n+1}{2}$ in Avg. case (comparisons)

*) Binary Search: (Best Algo when array sorted)

→ DAC Binary Search:

```
f(Sorted A[n], i, j, x)
{ if(i==j) { if (a[i]==x) ret(a[i])
             else ret(-1) }
  else{
      K = ⌊(i+j)/2⌋          → c
      if(a[K]==x) ret(K)
      else{
          if(x<a[K])
              f(A[n], i, K-1, x)
          else
              f(A[n], K+1, j, x)
      }
  }
}
```

c (or) $T(n/2)$

TC: • WC & AC

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

$= \Theta(\lg_2 n)$

• BC

$T(n) = c+c = \Theta(1)$

SC: • WC & AC → $\Theta(\lg_2 n)$

• BC → $\Theta(1)$

→ Non recursive Binary Search:

```
f(a,i,j,x)
{ while(i≤j)
  { if(i==j)
      { if (a[i]==x)
            ret(i)
        else
            ret(-1)
      }
    else{ K=⌊(i+j)/2⌋
          if(x==a[K])
              ret(K)
          else
              if(x<a[K])
                  j=K-1
              else
                  i=K+1 }
```

TC: $O(\lg_2 n)$

SC: $O(1)$

Note:) Non recursive binary search better cuz. same TC & using no stack space.

Q: A[n], Sorted with distinct elements
Find any element such that a[i] == i
Best algo, wc TC?

A [-50 -40 -30 0 3 5 7 90 100 200 220]
1 2 3 4 5 6 7 8 9 10 11

method-1: Linear search ⇒ $\Theta(n)$ [wc]

method-2: Binary search ⇒ $\Theta(\lg n)$ [wc]

```
f(){ if (k > value)
        go right
     else
        go left
}
```

$T(n) \leftarrow O(\lg n)$, $\neq \Theta(\lg n)$, $\Omega(1)$

Q: A[n]; until pos.^{n} x all increasing afterwards decreasing. Find x

A [10 20 30 40 50 60 50 40 30 20]
1 2 3 4 5 6 7 8 9 10

method-1: Linear Search ⇒ $\Theta(n)$

```
{while (a[i] < a[i+1])
        i++;
 ret(i)
}
```

Method-2: Binary search ⇒ $\Theta(\lg n)$

```
{ if(a[k-1] < a[k] &&
     a[k] > a[k+1])
     ret(k)
  else{ if (a[k] < a[k+1])
           go right
        else go left
}
```

Q: A[n]; until pos.^{n} x all are ∞ afterwards all integers. Find x

A [∞ ∞ ∞ ∞ ∞ ∞ 20 -10 30 -5]
1 2 3 4 5 6 7 8 9 10

Method-1: LS(){while(a[i] == ∞) ⇒ $\Theta(n)$
```
        i++;
 ret(i-1);
}
```

Method-2: BS(){ if (a[k-1] == ∞ &&
```
     a[k+1] == INT)
     ret (k)
  else{ if (a[k+1] == ∞)
           go right
        else go left
}
```

$\Theta(\lg n) \Leftarrow$

Q: A[n]; until pos.^{n} x all are integers afterwards ∞. Array size unknown, but after array all are $.

A[10 20 30 -5 11 3 ∞ ∞ ∞ ∞ ∞ $ $ $ $ $]
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Method-1: LS(){ while(a[i] != ∞)
```
        i++;
 ret(i-1);
}
```
⇒ $\Theta(n)$

Method-2

1. Find Approx. array size
```
{while(a[i] != $)
      i *= 2;
}
```
$\Theta(\lg_2 n) \Leftarrow$

2. BS() ⇒ $\Theta(\lg n)$
```
{ if(a[k] == ∞ &&
     a[k-1] == INT)
     ret (k)
  else{ if(a[k] == INT)
           go right
        else go left
}
```

Q: A[n] Sorted. Find any 2 elements such that a+b > 1000

Method-1: Select one element & add it iteratively with rest of the elements
⇒ n × LS ⇒ n·n ⇒ $\Theta(n^2)$

Method-2: n × BS ⇒ $\Theta(n \lg n)$

A [100 (200 300 400 500 600 700 800)]
1 2 3 4 5 6 7 8
a b

and so on..

Method-3: Return Last 2 elements ⇒ $\Theta(1)$

Q: A[n] unsorted. Same..

Method-1: n × LS ⇒ n^2

Method-2: 1) sort ⇒ n lg n
2) Return Last 2 elements ⇒ 1] ⇒ n lg n

Method-3: 1) Apply 2 passes bubble sort ⇒ 2n
2) Return Last 2 elements ⇒ 1] ⇒ $O(n)$

Q: A[n] Sorted. a+b < 1000
Return first 2 elements ⇒ $O(1)$

Q: A[n] unsorted. a+b < 1000
2 passes selection sort ⇒ 2n
or
2 LS for 2 min.^{m} elements ⇒ 2n] ⇒ $\Theta(n)$

CamScanner

Q: A[n] sorted. a+b == 1300

m1: n×LS ⇒ n^2

m2: n×BS ⇒ n lgn

m3: Greedy

A[100 200 300 400 500 600 700 800]

a a a a a b

1. Set 'b' such that its just less than 1300 ⇒ O(n) [∵ LS]
2. Move 'a' L to R ⇒ O(n)

i.e. if (a+b > want) b--; else a++;

TC : O(n)

Q: A[n] unsorted. a+b == 1300

m1: n×LS ⇒ n^2

m2: Sort ⇒ n lgn, n×BS ⇒ n lgn] 2n lgn ⇒ n lgn

m3: Best: Sort ⇒ n lgn, Greedy ⇒ n] ⇒ n lgn

Q: A[n] sorted. a+b+c = 1000

m1: n^2×LS = n^3

m2: n^2×BS = n^2 lgn

a b c

After all 'c' over 'b' will move

After all 'b', 'c' over 'a' will move

m3: Greedy = n×n = n^2

a → one loop

b → one loop

c → Found by LS

a b c

Q: A[n] sorted. 10 element sum = 1000

m1: n^9×LS = n^{10}

m2: n^9×BS = n^9 lgn

m3: Greedy = n^8×n = n^9 →
1. a - one loop
2. b - one loop
...
9. i - one loop
10. j - found by LS

Q: A[n] sorted. 10 element sum < 1000

Return first 10 elements ⇒ O(1)

Q: A[n] unsorted. 5 element sum = 1000

m1: n^4×LS = n^5

m2: sort = n lgn, n^4×BS = n^4 lgn] = n^4 lgn

m3: Greedy - sort = n lgn, - n^3×n = n^4] = n^4

Q: A[n] sorted or unsorted. Find any subset such that their sum = 1000

Don't know size of subset ⇒ NP complete

⇒ O(2^n)

Note: Probes in BS = $\lceil \lg_2 n \rceil$

*) Merge Sort :

→ No. of divisions = No. of combines = No. of internal nodes

No. of internals = Total − Leafs

→ Merge Algorithm:

case-1 Outplace & worst case: Elements coming from alternate groups.

a: 10 20 30 40 | 11 21 31 41 = i/p = Two sorted subarrays

$\frac{n}{2}$ or m $\frac{n}{2}$ or n

b: 10 11 20 21 30 31 40 41

comparisons = $\frac{n}{2}+\frac{n}{2}-1$ or m+n−1

Moves = $\frac{n}{2}+\frac{n}{2}$ or m+n

TC (merge Algo) = Comparisons + Moves ≃ Moves

= Θ(m+n) or Θ(n)

case-2 Outplace & best case: Always elements are coming from same group, which is small group. And at the end you copy leftout elements of bigger group.

a: 50 60 70 80 90 100 | 10 20 30

m n

b: 10 20 30 50 60 70 80 90 100

comparisons = min(m, n)

Moves = m+n

TC (merge Algo) = Comparisons + Moves ≃ Moves

= Θ(m+n)

→ Merge Algo (outplace) always take Θ(m+n)

Merge Algo (inplace) = O(m·n) [W.C]

SC (merge algo) = O(n) [Taking an array for merging]

→ Merge Algo:

f(a, i, K, K+1, j) ⟹ Θ(m+n)

```
{ create b[j-i+1]
  int p=i, q=K+1, r=0;
  while( p ≤ K-1 && q ≤ j)
  { if(a[p] < a[q]) { b[r] = a[p];
                      r++;
                      p++;
    else { b[r] = a[q];
           r++;
           q++;
  return(b)
```

→ Merge Sort(a, i, j)

```
{ if(i==j)  return(a[i])
  else
  { mid = ⌊(i+j)/2⌋ → c
    mergeSort(a, i, mid) → T(n/2)
    mergeSort(a, mid+1, j) → T(n/2)
    MergeAlgo(a, i, mid, mid+1, j) → n
    return (a)
  }
}
```

TC: $T(n) = \begin{cases} c & ; n=1 \\ 2T(n/2)+n & ; n>1 \end{cases}$

$= \Theta(n \cdot \lg n)$ [E.C]

SC: merge + stack space

$= n + \lg n$

$= \Theta(n)$

Q: $\frac{n}{a}$ arrays each of size $\frac{n}{b}$. each subarray sorted using bubble sort. Find single sorted array.

Size of 1 subarray $= \frac{n}{b}$

bubble sort cost of 1 subarray $= (n/b)^2$

$\frac{n}{a}$ subarrays $= \frac{n}{a}\left(\frac{n}{b}\right)^2$

Now we're gonna merge them → n/a groups

$\overset{1}{n/b} + \overset{2}{n/b} + \cdots \overset{n/a}{n/b}$

$\frac{n^2}{ab}$ elements ⇒ $\Theta\left(\frac{n^2}{ab}\right)$ to merge

$2n/b + 2n/b \cdots$ → $\frac{n}{2a}$ groups

$\frac{n^2}{ab}$ elements

K

n^2/ab → $\frac{n}{2^K a}$ groups

$\frac{n}{2^K a} = 1 \Rightarrow K = \lg_2(n/a)$

TC = Bubble sorting + merging

$= \frac{n^3}{ab^2} + \frac{n^2}{ab}\lg_2(n/a)$

Q: lgn sorted subarrays, each of size $\frac{n}{\lg n}$. Find single sorted array with all elements → lgn groups

$\overset{1}{n/\lg n} + \overset{2}{n/\lg n} + \cdots \overset{\lg n}{n/\lg n}$

n elements ⇒ merge = $\Theta(n)$

n elements

$2n/\lg n + 2n/\lg n \cdots$ → $\frac{\lg n}{2}$ groups

K

n → $\frac{\lg n}{2^K}$ groups

w.k.t, $\frac{\lg n}{2^K} = 1 \Rightarrow K = \lg_2 \lg_2 n$

TC = Merge cost at any level × no. of levels

$= \Theta(n) \times \Theta(\lg_2 \lg_2 n)$

$= \Theta(n \cdot \lg \lg n)$

Q: a sorted arrays each of size n/b. Find single sorted array → a groups

$\overset{1}{n/b} + \overset{2}{n/b} + \cdots \overset{a}{n/b}$

$\frac{an}{b}$ elements ⇒ $\Theta\left(\frac{an}{b}\right)$ to merge

$2n/b + 2n/b \cdots$ → $a/2$ groups

$\frac{an}{b}$ elements

K

an/b → $a/2^K$ groups

w.k.t, $\frac{a}{2^K} = 1 \Rightarrow K = \lg_2 a$

TC = merge cost at any level × no. of levels

$= \frac{an}{b} \times \lg_2 a$

$= \Theta\left(\frac{an}{b} \cdot \lg_2 a\right)$

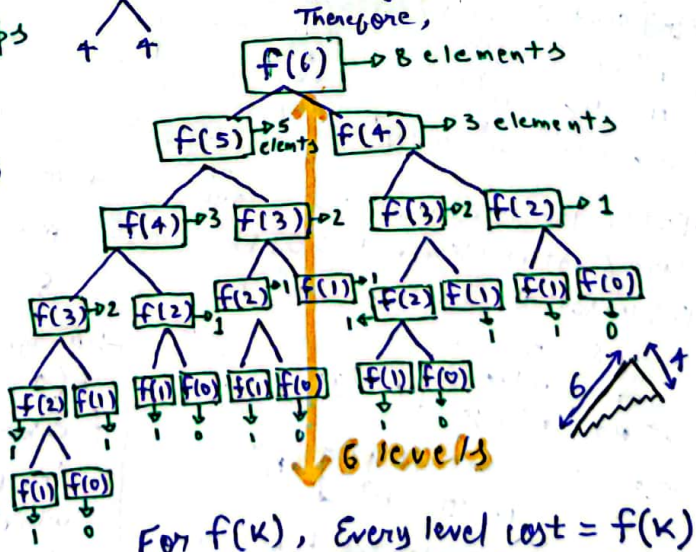
→ Fibonacci merge sort: Instead of dividing in two equal parts, divide in fibonacci series manner.

n	0	1	2	3	4	5	6	7	8	9
f(n)	0	1	1	2	3	5	8	13	21	34

Normal merge sort: 8 elements → 4 | 4

Fibonacci merge sort: f(6) ⇒ 8 elements

Therefore,



For f(K), Every level cost = f(K)

No. of levels = K

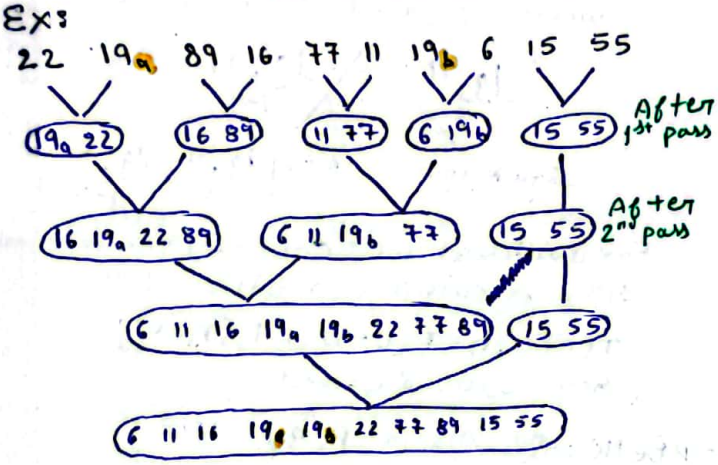
TC: $\frac{K}{2} \cdot f(K) \le T(K) \le K \cdot f(K)$

$T(K) = \Theta(K \cdot f(K))$

→ Straight 2-way merge Sort:

Instead of dividing by 2 recursively and then merging, just take two-two elements directly & combine. It is stable also.

Ex:



No. of levels = $\lg_2 n$

Every level cost = n

TC: $O(n \lg n)$

Q: Two sorted arrays A[m], B[n] with distinct elements. $A \cup B$, $A \cap B$ = ?

A[10 20 30 40 50 60 70]
(1 2 3 4 5 6 7)

B[11 21 31 40 45 50 60 65 80]
(1 2 3 4 5 6 7 8 9)

m1: $m \times LS(n) \Rightarrow \Theta(n^2)$ [WC]

m2: $m \times BS(n) \Rightarrow \Theta(n \lg n)$ [WC]

m3: i) For $A \cap B$:

Modified Merge Algo → print if equal o/w leave

WC	BC
$\Theta(m+n)$ $= \Theta(n)$	min(m,n): ∵ if one array is over, no need to look further

ii) For $A \cup B$:

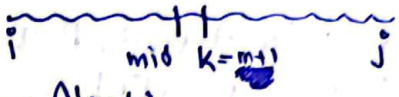
WC $\Theta(m+n)$	BC
$= \Theta(n)$	$\Theta(m+n)$ $= \Theta(n)$ ∵ You'll have to mandatorily run a loop to copy leftout elements

*) Counting no. of inversions: (merge sort ETC)

→ Inversion: $i < j$ && $a[i] > a[j]$

→ By Brute force: $n \times LS = O(n^2)$ [EC]

→ By DAC:



Modified Merge Algo()

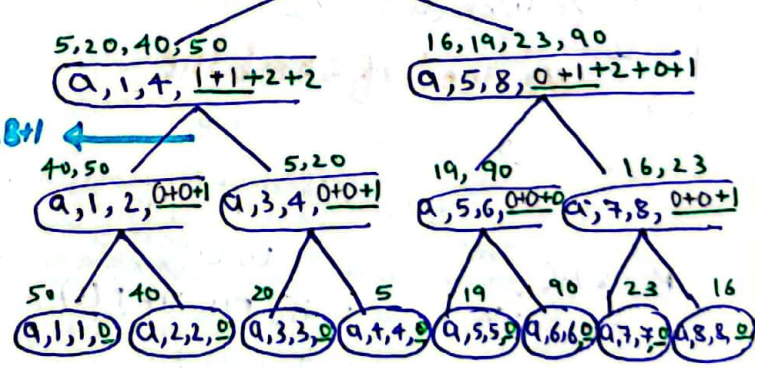
```
{ l = 0
  Inversions = Left Inversions + Right Inversions
  // Now merging left & right
  while( i ≤ mid && k ≤ j )
  { if(a[i] < a[k]) // No inversion
      { b[l] = a[i]
        i++ , l++
      }
    else // Inversion occured
      { b[l] = a[k]
        k++ , l++
      }
    Inversions = Inversions + (mid - i + 1)
  }
}
```

UB-LB+1 of left part

→ Ex: A[50 40 20 5 19 90 23 16]
(1 2 3 4 5 6 7 8)

5, 16, 19, 20, 23, 40, 50, 90

(a, 1, 8, 6+4+0+3+3+0+2+0+0 = 18)
LB UB Inversions



→ TC: $T(n) = O(1) + 2T(n/2) + n$
Divide conquer combine

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

$\lg n$: Stack Space

$= \Theta(n \lg n)$ [EC]

SC: Extra space = Stack + array b to merge

$= \lg n + n = \Theta(n)$

→ Conclusion: It is merge Sort only, just merge algorithm is modified.

*) Quick Sort Algorithm:

→ Quicksort is Inplace, Not-Stable

→ Quicksort is mostly used (practical)

→ PartitionAlgo(a, P, q)

```
{ i = P ,  x = a[P]
  for( j = P+1; j ≤ q; j++)
  {  if(a[j] ≤ x)
     {  i++
        swap(a[i], a[j])
     }
  }
  swap(a[P], a[i])
  return(i)
}
```

$\boxed{TC: \Theta(n)}$ (∵ Scanning n-1 elements)

Partition algorithm returns correct position of pivot.

→ QuickSort(a, P, q)

```
{ if(p==q) return(a[P])   → c
  else
  {  m = PartitionAlgo(a, P, q)   → n
     QuickSort(a, P, m-1)   → T(m-1-P+1)  UB-LB+1
     Quicksort(a, m+1, q)   → T(q-m-1+1)
     return(a)
  }
}
```

No need of combine

$$TC: T(n) = \begin{cases} c & ; n=1 \\ n + T(m-p) + T(q-m) & ; n>1 \end{cases}$$

BC & AC (Random or Unsorted array)

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

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

$\lg n$

$= \Theta(n \lg n)$

WC (Array already sorted almost sorted)

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

$= T(n-1) + n$

n

$= \Theta(n^2)$

→ Ex: A[56, 16, 10, 86, 8, 96, 29, 10, 80, 90, 8, 50, 80, 89, 54, 16]

Pivot $x = 50$

p, i, i, i, i, i, i, i, i, j

$\underline{5}$ = (16, 10, 8, 29) 50 (80, 89, 54, 90)

1, 2, 3, 4 — $\underline{5}$ — 6, 7, 8, 9

Smaller than 50 ⇓ recursion

Greater than 50 ⇓ recursion

Note: • To apply partition algorithm at least 2 elements required.

• Partition algorithm division is based on selected pivot

→ Note:) For Quicksort BC & AC: Some fraction of n on one side, remaining on other side.

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

where; $0 < \alpha < 1$ & α = const.

$\lg_{\frac{1}{\alpha}} n$ $\lg_{\frac{1}{1-\alpha}} n$

Stack space = $\max\left[\lg_{\frac{1}{\alpha}} n, \lg_{\frac{1}{1-\alpha}} n\right]$

For Quicksort WC: Constant elements one side, remaining on other side.

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

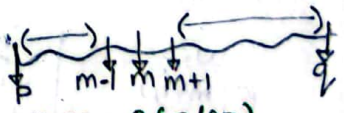
here, $c > 0$ & c = const.

→ Better-QS-Algo(a, P, q)

```
{ if(P==q) return(a[P])
  else { while(P ≤ q)
         {  m = PartitionAlgo(a, P, q)
            if(m-P < q-m)
            {  // i.e. left side smaller
               QS(a, P, m-1)
               P = m+1
            }
            else // i.e. right side smaller
            {  QS(a, m+1, q)
               q = m-1
            }
         }
  }
}
```

Recursion taking care of smaller part

while loop taking care of bigger part



BC & AC = $\Theta(n \lg n)$

WC = $\Theta(n^2)$

TC: Same

Stack space: min = 1 (worst partition); max = $\lg n$ (Best & Avg. partition)

By default take this one

Note: Quicksort takes max^m time (n^2) when:-

1) Array already sorted in same order

2) ——— reverse "

3) All elements same

Q: i) 1, 2, 3, ... n
ii) n, n-1, n-2, ... 1
iii) n, n, n ... n } I/P's

C_1, C_2, C_3 → comparisons made
S_1, S_2, S_3 → swaps done

To keep I/P's in Asc. order using Quicksort
Find relation b/w C's & S's

i) 10 20 30 40 50 60 70
P i j

⇓ Par(7) ⇒ 6 Comparison, 1 Swap

(10) (20 30 40 50 60 70)

⇓ Par(6) ⇒ 5C, 1S

(20) (30 40 50 60 70)

⇓ Par(5) ⇒ 4C, 1S

(30) (40 50 60 70)

⇓ par(4) ⇒ 3C, 1S

(40) (50 60 70)

⇓ par(3) ⇒ 2C, 1S ...

(50) (60 70)

⇓ par(2) ⇒ 1C, 1S

(60) (70)

Total comparisons $= n-1+n-2+\cdots 3+2+1$
$= \frac{n(n-1)}{2} = O(n^2) = C_1$

Total swaps = Swap in each pass × no. of passes
$= 1 \times (n-1) = O(n) = S_1$

ii) 70 60 50 40 30 20 10
P i j

⇓ Par(7) ⇒ 6C, 7S

(10 60 50 40 30 20) (70)
P i j

⇓ Par(6) ⇒ 5C, 1S

(10) (60 50 40 30 20)

⇓ Par(5) ⇒ 4C, 5S

...

Total comparisons $= n-1+n-2+\cdots 1$
$= \frac{n(n-1)}{2} = O(n^2) = C_2$

Total swaps $= \frac{n}{2}$ passes × 1 swap + $\frac{n}{2}$ passes × n swaps
$= O(n^2) = S_2$

iii) 10 10 10 10 10 10
P i j

⇓ Par(6) ⇒ 5C, 6S

(10 10 10 10 10)(10)

⇓ Par(5) ⇒ 4C, 5S

...

Total Comparisons $= n-1+n-2+\cdots 1$
$= \frac{n(n-1)}{2} = O(n^2) = C_3$

Total swaps = swaps in each pass × no. of passes
$= n(n-1) = O(n^2) = S_3$

Therefore; $C_1 = C_2 = C_3$
mathematically: $S_1 < S_2 < S_3$
Asymptotically: $S_1 < S_2 = S_3$

→ Randomized quicksort:
1. Generate random no. from 1 to n
2. Store to x
3. swap (a[p], a[x])
4. QS(a, p, q)

Ex:) 10 20 30 10 50 60 70
P

x = RG(1, 7)
4 (say)

10 20 30 10 50 60 70
P i j

(10 20 30)(40)(50 60 70)

TC: O(n lg n) [Some best partitions, Some worst "]

*) Strassen's Matrix multiplication:

→ Normal matrix multiplication:

```
for(i=1 to n)
for(j=1 to n)
for(k=1 to n)
    c[i][j]+=a[i][k]*b[k][j]
```

⇒ $\Theta(n^3)$

→ DAC matrix multiplication:

if (size 2×2 or 1×1) {Small & Solution}
else {1. Divide the square matrix n×n recursively until you get submatrices of size 2×2.
2. Conquer by mul.ns & add.ns}

→ Ex:

$$A = \begin{bmatrix} \underbrace{\begin{matrix}1 & 2\\ 3 & 4\end{matrix}}_{A_{11}} & \underbrace{\begin{matrix}2 & 3\\ 4 & 5\end{matrix}}_{A_{12}} \\ \underbrace{\begin{matrix}3 & 4\\ 5 & 6\end{matrix}}_{A_{21}} & \underbrace{\begin{matrix}4 & 8\\ 9 & 10\end{matrix}}_{A_{22}} \end{bmatrix}_{\frac{4}{2}\times\frac{4}{2}} \quad B = \begin{bmatrix} \underbrace{\begin{matrix}11 & 21\\ 30 & 41\end{matrix}}_{B_{11}} & \underbrace{\begin{matrix}51 & 61\\ 71 & 81\end{matrix}}_{B_{12}} \\ \underbrace{\begin{matrix}31 & 88\\ 42 & 36\end{matrix}}_{B_{21}} & \underbrace{\begin{matrix}1 & 2\\ 5 & 7\end{matrix}}_{B_{22}} \end{bmatrix}_{\frac{4}{2}\times\frac{4}{2}} \quad C = A*B = \begin{bmatrix} \underbrace{\begin{matrix}5 & 6\\ 7 & 8\end{matrix}}_{C_{11}} & \underbrace{\begin{matrix}9 & 10\\ 11 & 12\end{matrix}}_{C_{12}} \\ \underbrace{\begin{matrix}13 & 14\\ 15 & 16\end{matrix}}_{C_{21}} & \underbrace{\begin{matrix}17 & 18\\ 19 & 20\end{matrix}}_{C_{22}} \end{bmatrix}_{\frac{4}{2}\times\frac{4}{2}}$$

1 Matrix Mul^n (4×4) ⇒ 8 MM (2×2) + 4 Matrix Add^n (2×2)

1 MM (16×16) ⇒ 8 MM (8×8) + 4 MA (8×8)

1 MM $(n\times n)$ ⇒ 8 MM $(n/2\times n/2)$ + 4 MA $(n/2\times n/2)$

Note: for matrices like 2X3 or 3X3 or 3X4 Pad zeros & make it to closest power of 2. like :-

$$\left[\begin{array}{cc|cc} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]_{\frac{4}{2}\times\frac{4}{2}}$$

TC: $T(n) = \begin{cases} c & ; n \le 2 \\ 8T(n/2) + 4(n/2)^2 & ; n > 2 \end{cases}$

∵ No of add^{n}s for $A_{n\times n} + B_{n\times n} = n^2$ ⇒ for $n/2 \times n/2 \to (n/2)^2$

$= 8T(n/2) + n^2$

$\log_2 n$ = Stack space

$\Theta(n^3)$

but acc. to Strassen's

$= 7T(n/2) + 18(n/2)^2$

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

$\log n$

$= \Theta(n^{2.81})$

* Maximum Continuous Subarray Sum :

→ A[n] with integer elements, You've to return max^m subarray sum and indices.

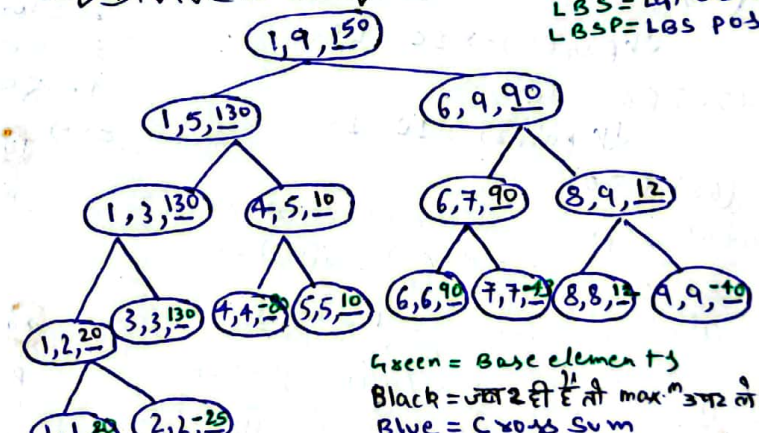
→ Brute force : ⇒ $O(n^2)$

20	-25	130	-80	10	90	-43	12	-40
1	2	3	4	5	6	7	8	9

```
f(){ maxSum = A[1], currentSum = 0
     p, q = 0   // indices of max^m subarray
     for(i=0; i<n; i++)
       { for(j=i; j<n; j++)
           {currentSum += A[j]
            if(currentSum > maxSum)
              { maxSum = currentSum
                p = i
                q = j
              }
           }
       }
}
```

→ Divide & conquer :

LTS = Left total sum
LBS = Left Best sum
LBSP = LBS position



Green = Base elements
Black = जब 2 ही हैं तो max^m उठा लेनी
Blue = Cross Sum

Cross Sum(1,3)

20	-25	130	
1	2	3	

LTS = ~~-25~~ -5 R
LBS = 0
LBSP =
RTS = 130
RBS = ~~0~~ 130
RBSP = 3
LBS + RBS = 130 = Cross Sum
130 > 90

Cross Sum(1,5)

20	-25	130	-80	10
1	2	3	4	5

LTS = ~~130~~ ~~105~~ 125
LBS = ~~0~~ 130
LBSP = 3
RTS = ~~-80~~ -70
RBS = 0
RBSP =
LBS + RBS = 130
130 > 10

Cross Sum(6,9)

90	-43	12	-40
6	7	8	9

LTS = ~~-43~~ 47
LBS = ~~0~~ 47
LBSP = 6
RTS = ~~12~~ -28
RBS = ~~0~~ 12
RBSP = 8
LBS + RBS = 59
90 > 59 > 12

Cross Sum(1,9)

20	-25	130	-80	10	90	-43	12	-40
1	2	3	4	5	6	7	8	9

LTS = ~~20~~ ~~-70~~ ~~60~~ ~~35~~ 55
LBS = ~~0~~ ~~0~~ 60
LBSP = ~~5~~ 3
RTS = ~~90~~ ~~47~~ ~~59~~ 19
RBS = ~~0~~ 90
RBSP = 6
LBS + RBS = 150
150 > 130 > 90

```
f(a, p, q)
{ if (p==q) return (a[p])
  else {
    mid = ⌊(p+q)/2⌋ → c
    LeftSum = f(a, p, mid) → T(n/2)
    RightSum = f(a, mid+1, q) → T(n/2)
    CrossSum = Cross(mid, p, mid+1, q) → n
    return(max[LeftSum, RightSum, CrossSum])
  }
}
```

```
Cross(mid, p, mid+1, q) → Θ(n)
{ RTS = 0   RBS = 0   RBSP = 0
  LTS = 0   LBS = 0   LBSP = 0
  for(i = mid; i ≥ p; i--) → n/2
  { LTS = LTS + a[i]
    if(LTS > LBS) { LBS = LTS
                    LBSP = i }
  }
  for(j = mid+1; j ≤ q; j++) → n/2
  { RTS = RTS + a[j]
    if(RTS > RBS) { RBS = RTS
                    RBSP = j }
  }
  return(LBS + RBS)
}
```

$$TC: T(n) = \begin{cases} c & ; n = 1 \\ 2T(n/2) + n & ; n > 1 \end{cases}$$

$\lg n$ = Stack space

$= O(n \lg n)$ [EC]

Selection Procedure:

→ A[n] (+ve distinct integers); $k \in$ int

Find k^{th} smallest element

→ Method-1: 1. Sort ⇒ $n \lg n$
2. return(a[k]) ⇒ c] $\Theta(n \lg n)$

Method-2: 1. k passes of selection sort ⇒ $k \cdot n$
2. return(a[k]) ⇒ c
⇒ $\Theta(k \cdot n)$

→ By Devide & Conquer: f(a,1,12,7) → 7th smallest

A [50 25 85 45 30 62 88 98 110 15 29 69]
(1 2 3 4 5 6 7 8 9 10 11 12)

K = 7

⇓ Partition Algo(n)

(25 45 30 15 29)(50)(85 62 88 98 110 69)
(1 2 3 4 5)(6)(7 8 9 10 11 12)

⇓

f(a, 7, 12, 7)

⇓

Partition Algo(n)

(62 69)(85)(88 98 110)
(7 8)(9)(10 11 12)

⇓

f(7, 8, 7)

⇓

Partition Algo(n)

(62)(69)
(7)(8)

⇓

return(a[7])

```
f(a, p, q, k)
{ if (p==q) return (a[p])
  else
  { m = Partition Algo(a, p, q) → n
    if (m==k) ret (a[k])
    else { if (k < m) f(a, p, m-1, k) → T(m-p)
           else f(a, m+1, q, k) → T(q-m)
    }
  }
}
```

$$TC: T(n) = \begin{cases} c & ; n = 1 \\ n + T(m-p) \text{ or } T(q-m) \end{cases}$$

BC AC

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

$\lg n$

$= O(n)$

WC

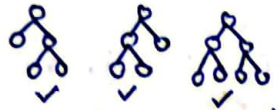
$T(n) = n + T(n-1)$

n

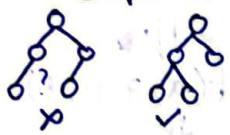
$= O(n^2)$

#) Binary Trees :

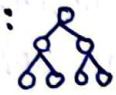
*) BT :



ACBT : no gaps allowed



C BT :



0 or 2 child
no level incomplete

*) 1) K-level CBT
Total nodes $(n) = 2^K - 1$

2) n-nodes CBT or ACBT
Leafs $= \lceil n/2 \rceil$
Non-Leafs $= \lfloor n/2 \rfloor$

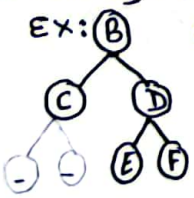
3) K-level, n-nodes CBT
$n = 2^K - 1 \Rightarrow$ Levels $= K = \lg_2(n+1)$

4) CBT or ACBT
Height(h) = Levels − 1
$= K - 1$
$\boxed{h = \lg_2(n+1) - 1}$

*) Representation of BT :

1. Array : Use when BT is CBT or ACBT

Ex:

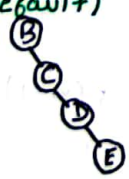


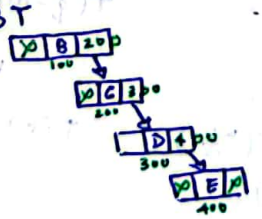
as

1	2	3	4	5	6	7
B	C	D	–	–	E	F

2. Linked List : Use when BT is not CBT or ACBT (Default)

Ex:

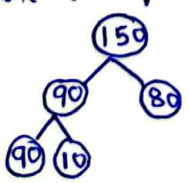




Note:) For BT of n nodes represented by Array
min^m array size $= n$
max^m " " $= 2^n - 1$

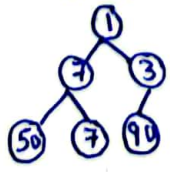
*) Heap Tree :

Maxheap
ACBT or CBT where at every node root is max^m or equal.

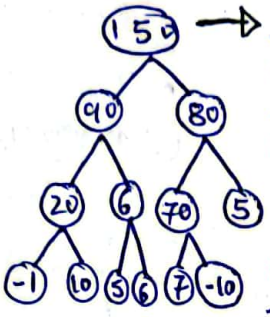


$\Theta(\log n)$

Minheap
ACBT or CBT where at every node root is min^m or equal.



*) Finding max^m in maxheap :



→ To Find 1st max^m ⇒ 0 comparisons ⇒ O(1)
(1 person fighting)

To Find 2nd max^m ⇒ 1 comparisons ⇒ O(1)
(2 person fighting)

. . . .

To Find 10th max^m ⇒ 9 comparisons ⇒ O(1)
(10 Person Fighting)

Can't Find directly 10th max^m, so doing from 1st to know what 10 persons are fighting $= \frac{9 \times 10}{2} = 45$ comparisons ⇒ O(1)

. . .

To Find K^{th} max^m ⇒ (K−1) comparisons

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

when K = n, it's actually 1st minimum.

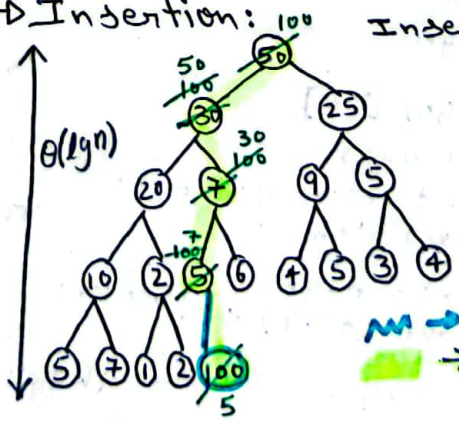
→ If you've to find min^m element in maxheap array, instead of doing it in $O(K^2)$ use one pass of selection sort O(n)

→ conclusion: In maxheap
max^m element finding = O(1) [EC]
min^m " " = O(n) [EC]

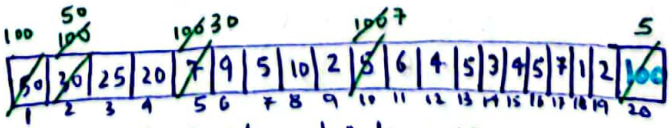
*) Finding min^m in minheap :
Similar.

*) Insertion, Deletion, Build Heap in Heap Tree :

→ Insertion: Insert 100 = x



→ insertion
→ Maxheapify



m = last element index = 19
n = Array size = 20
m++ , a[m] = x

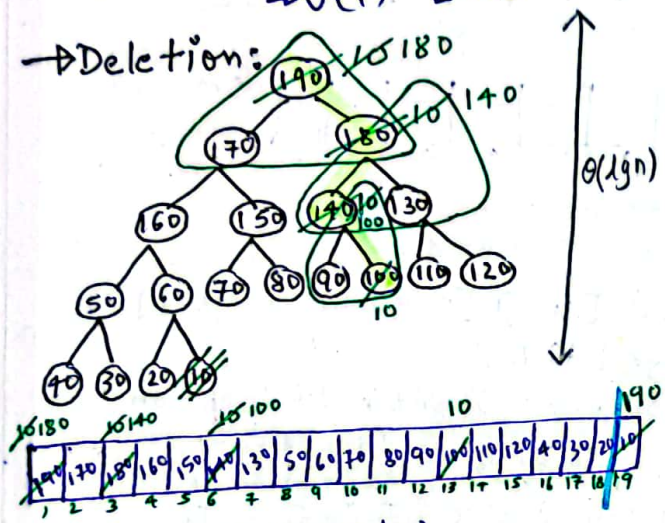
Maxheapify ⇒ lgn comparisons ⇒ O(lgn) [WC]
lgn Swaps

$\underset{BC}{\Rightarrow} O(1)$

$\underset{AC}{\Rightarrow} \frac{1}{2}\lg n + \frac{1}{2}\lg n$
$= O(\lg n)$

Therefore,
Insertion → $O(\lg n)$ [WC, AC]
→ $O(1)$ [BC]

→ Deletion:



$m = 19$ (Last index)
$n = 19$ (array size)

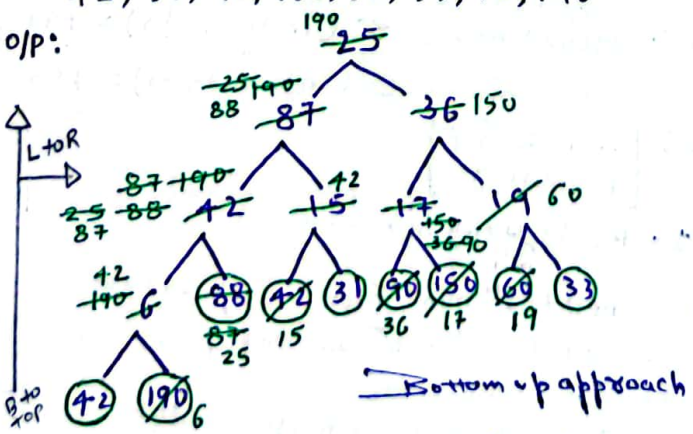
$x = a[1]$
Replace root by last element
Maxheapify ⇒ $\left\{\begin{matrix} 2 \text{ comparison} \\ 1 \text{ swap} \end{matrix}\right\}$ in each level
⇒ $3 \lg n$
$= O(\lg n)$ [WC, AC]
$= O(1)$ [BC]

Note:) Maxheap deletion n-times → Asc. order sorted array
Minheap " " → Desc. order sorted array

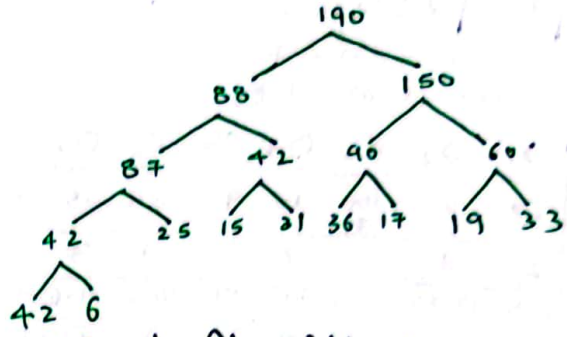
→ Buildheap: ⇒ $O(n)$ [EC]

i/p: 25, 87, 36, 42, 15, 17, 19, 6, 88, 42, 31, 90, 150, 60, 33, 42, 190

o/p:



Starting from first non-leaf in bottom
अपना successor ढूंढते जाओ
level-by-level left to right



> HeapSort Algorithm:

*) 1. Create maxheap by BuildHeap → n
2. Delete one-by-one element & store from RHS continuously n-times. → $n \times \lg n$

*) TC: $O(n \lg n)$
But if all elements are same $O(n)$

*) Heapsort is not-stable
Heapsort is inplace

> Greedy Algorithm approach:

*) Greedy: n-no. of inputs & our objective is finding subset which will satisfy our conditions & maximizes our goal.

*) Applications of Greedy:
1) Knapsac (fractional)
2) Job sequencing with deadline
3) Optimal merge pattern
4) Huffman coding
5) Minimum Cost Spanning Tree (MST)
   a) Prims b) Kruskal
6) Single source shortest path
   a) Dijkstra b) Bellmanford c) BFT

*) Knapsac problem:

No. of elements = n = 7
Sac size = m = 17

Object	O_1	O_2	O_3	O_4	O_5	O_6	O_7
Profit	25	55	100	15	95	65	35
weight	2	4	3	2	5	4	3
1. P/w	12.5	13.75	33.3	7.5	19	16.2	11.6 → n
2. Sort	5	4	1	7	2	3	6 → n lgn

3. m = 17 | P = 0

$17 - 1\times3 = 14$ | $0 + 1\times100 = 100$
$14 - 1\times5 = 9$ | $100 + 1\times95 = 195$
$9 - 1\times4 = 5$ | $195 + 1\times65 = 260$
$5 - 1\times4 = 1$ | $260 + 1\times55 = 315$
$1 - \frac{1}{2}\times2 = 0$ | $315 + \frac{1}{2}\times25 = 327.5$ → n

4. Sequence:

1/2	1	1	0	1	1	0
O_1	O_2	O_3	O_4	O_5	O_6	O_7

TC: O(n lgn) [EC]

*) Job sequencing with deadlines:

→ single CPU, Non-premption
BT = 1, AT = 0 for all process

→ Method-1: n = 7

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Profit	80	20	150	25	95	15	70
Deadline	5	3	1	4	3	2	4

O(n)	O(n)	O(n)	O(n)	O(n)
J_3	J_4	J_5	J_7	J_1
1	2	3	4	5

Filling from Last to first

Fighting for 1 → $\cancel{J_1}$ J_2 J_3 $\cancel{J_4}$ $\cancel{J_5}$ J_6 $\cancel{J_7}$ → n
" " 2 → $\cancel{J_1}$ J_2 J_4 $\cancel{J_5}$ J_6 $\cancel{J_7}$ → n
" " 3 → $\cancel{J_1}$ J_2 J_4 J_5 $\cancel{J_7}$ → n
" " 4 → $\cancel{J_1}$ J_4 J_7 → n
" " 5 → J_1 → n

Bottom to top

max.m profit = 420
Penalty = 35 [Jobs which are not in sequence table]

TC: T(n) = no. of Jobs × Highest deadline
= n × n = n^2 [WC]
= n × 1 = n [BC]

→ Method-2:

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
Profit	55	77	11	39	65	95	25	10	88
Deadline	7	3	4	5	1	5	4	2	3

1. Sort jobs in decreasing order of profit

J	J_6	J_9	J_2	J_5	J_1	J_4	J_7	J_3	J_8
P	95	88	77	65	55	39	25	11	10
D	5	3	3	1	7	5	4	4	2

2.

O(1)	O(1)	O(1)	O(n)	O(1)		O(n)
J_5	J_2	J_9	J_4	J_6	—	J_1
1	2	3	4	5	6	7

are you 7 or 7 greater?

max.m profit = 419
Penalty = 46

TC: n^2 [WC] [no. of Jobs = no. of slots] i.e. n
n lgn [AC, BC]

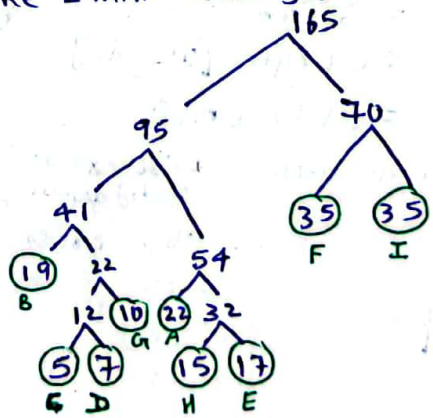
Note: If sequence doesn't match in option
J_5 J_2 J_9 J_4 J_6 = J_5 J_2 J_9 J_6 J_4
J_5 J_9 J_2 J_4 J_6
J_5 J_9 J_2 J_6 J_4

Jobs with same deadline can be swapped in sequence table

*) Optimal merge pattern: (uses Heap DS)

→ A B C D E F G H I ⇐ Files
22 19 5 7 17 35 10 15 35 ⇐ Records in File

Take 2 min.m & merge, order doesn't matter



min.m record moves = Σ (internals) = 491
max.m comparisons = Σ (internals − 1) = 483

→ TC: A B C D E F G H I

Do only one time: 1. Build min Heap of ⇒ O(n)

Repeat n−1 times:
2. Delete 1st min.m from minheap ⇒ lg n
3. Delete 2nd min.m from minheap ⇒ lg n
4. Add them ⇒ c
5. Insert into min heap ⇒ lg n

T(n) = n + (n−1) × 3 lg n
= O(n lg n) [EC]

*) Huffman coding: (Non-Uniform Coding)

→ Data encoding & compression technique

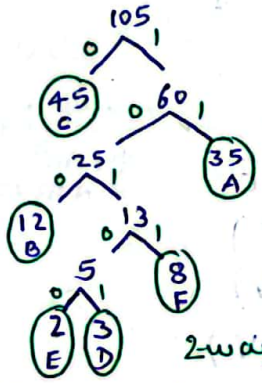
→ chars | Freq. | bits

chars	Freq.	bits
A	35	11 → 2 bits
B	12	100 → 3 bits
C	45	0 → 1 bits
D	3	10101 → 5 bits
E	2	10100 → 5 bits
F	8	1011 → 4 bits
	105	

max.m freq. ⇒ Less bits

Less freq. ⇒ more bits

Take 2 min.m & merge, Order matters



2way encoded tree

Total bits $= 35\times2 + 12\times3 + 45\times1 + 3\times5 + 2\times5 + 8\times4$
$= 208$ bits

Avg. (bits/char.) $= \frac{208 \text{ bits}}{105 \text{ chars}}$
$= 1.981$ b/c

→ If ASCII (uniform coding) used

Each char. = 8 bits i.e. 8 b/c

Total bits = $8\times105 = 840$ bits

→ Percentage compression:

characters = 6

Bits required to represent = 3 bits

Huffman code = 1.981 bits

$\% \text{ comp.} = \frac{\text{bits saved}}{\text{actual bits}} \times 100$

$= \frac{3-1.981}{3} \times 100$

$\simeq 34\%$

→ TC: when minheap DS used

= Create minheap + Delete 1st min.m + Delete 2nd min.m + Insert in minheap

$= n + (n-1)\,3\lg n = O(n\lg n)$ [EC]

TC: when array DS used

= Search 2 min.m in array × (n-1) times

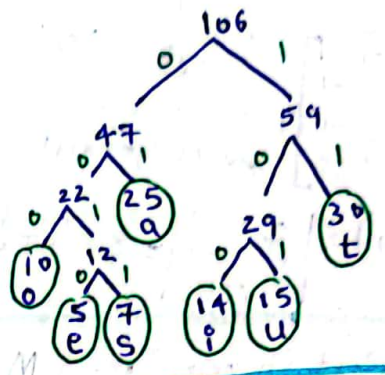
$= 2n \times (n-1)$

$= O(n^2)$ [EC]

Note: Huffman coding is optimal prefix notation, i.e. in Huffman coding no character is the exact prefix of some other character.

→ Ex-2

a	e	i	o	u	s	t
25	5	14	10	15	7	30



@) Simple Graph (G):

→ Simple graph
- parallel edges ✗
- Self loop ✗

→ Types of simple graph: complete, Null (no edges, only vertices), Normal

→ For simple graph:
- min.m degree = 0
- max.m " " = n-1
- min.m edges = 0
- max.m edges $= \frac{n(n-1)}{2}$

→ $\sum \deg(v_i) = 2|E|$

$|E| \le \frac{V(V-1)}{2} \le c\cdot V^2 = O(V^2)$

$\lg E = 2\lg V = O(\lg V)$

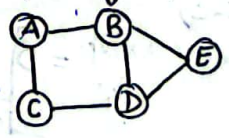
@) Spanning tree (S):

→ Spanning tree (S)

$S \subseteq G$ such that S has
- All vertices of G
- n-1 edges
- no cycle

→ For K_n, No. of ST's $= n^{n-2}$

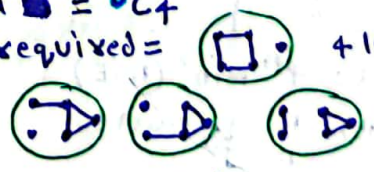
→ No. of Spanning Trees from G:



: G

method-1: n = 5; edges to take in ST = e_t = 4
Total edges in G = e_T = 6

- Possibility of cycle by e_t: 4 length, 3 length
- Total = $^6C_4$
- Not required =



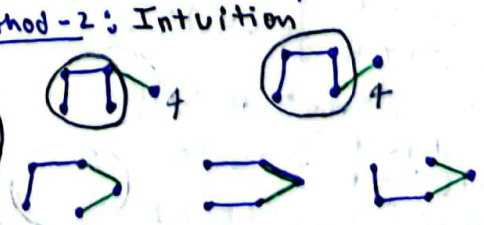
4 length cycle

3 length cycle

= 4

- Required = $^6C_4 - 4 = 11$

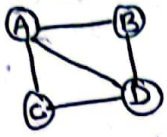
method-2: Intuition



= 11

Method-3: Kirchoff theorem

No. of Spanning trees for



$$M = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 3 & -1 & -1 & -1 \\ B & -1 & 2 & 0 & -1 \\ C & -1 & 0 & 2 & -1 \\ D & -1 & -1 & -1 & 3 \end{array}$$

~ → Degree of ABCD

~ → edge exist ⇒ -1

$$\text{cofactor}(M_{11}) = \begin{vmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 3 \end{vmatrix} = 8 = \text{No. of ST's}$$

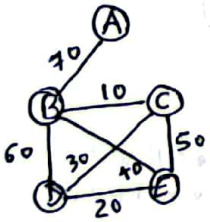
*) Minimum Cost Spanning Tree (MST):

→ To construct MST we have 2 Algo.
1) Kruskal 2) Prims

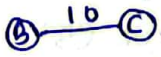
→ Kruskal Algorithm: (concentrate edges wherever min.m)

1. Create minheap of edges → O(E)
2. Delete min.m edge weight from minheap iteratively until you get (V-1) edges such that no cycle there → O((V-1) lgE) → O(E · lgE)

→ Ex: Kruskal 0. Create minheap → O(E)

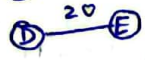


1. Delete 1st min.m



| BC | lgE
| A D E |

2. Delete 2nd min.m

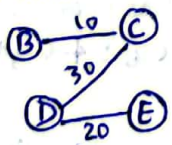


| BC | lgE
| DE A |

3. From now onwards, Delete-check cycle-Add

Delete 3rd min.m (D-C) | lgE

Find (D) = First element of group to which D belongs = D
Find (C) = B
⇒ no cycle, Add DC



| BCDE |
| A |

4. Delete 4th min.m (B-E) | lgE

Find (B) = B
Find (E) = B
⇒ cycle

Delete 5th min.m (C-E) | lgE

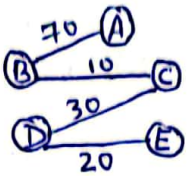
Find (C) = B
Find (E) = B
⇒ cycle

Delete 6th min.m (B-D) | lgE

Find (B) = B
Find (D) = B
⇒ cycle

Delete 7th min.m (B-A) | lgE

Find (B) = B
Find (A) = A
⇒ no cycle, Add BA



| ABCDE |

It took E + 7 lgE time

→ TC of Kruskal:

Best case → (No cycle arrived)

$= E + (V-1)\lg E$

creating minheap — Adding edges to MST (No cycle arrived at any step)

$= O[E + (V)\lg E]$

$= O[E + V \cdot \lg V]$ ∵ [$\lg E = O(\lg V)$

$= O(V^2)$ ∵ [$E = O(V^2)$

worst case → (Always cycle arrived so, सारी edges ले-लेकर देखनी पड़ी)

$= E + E \cdot \lg E$

$= O(E \cdot \lg E)$

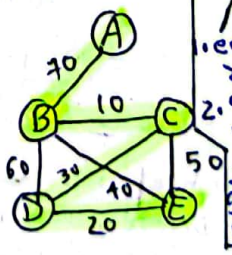
$= O(E \cdot \lg V)$

$= O(V^2 \cdot \lg V)$

→ Find(x) takes constant time. For cycle checking purpose we use Disjoint set DS, allowed operations are Find & Union

→ Note: • Every MST contains 1st & 2nd min.m edge weight
• Some MST's may contain max.m edge weight also.

→ Kruskal to MST



Note: In Gr if
1. edge weights repeated ⇒ Multiple MST's possible
2. edge weights distinct ⇒ 0 or 1 MST possible
3. connected graph & distinct edge weights ⇒ exactly 1 MST possible

edge selection in Kruskal = BC, DE, DC, BA

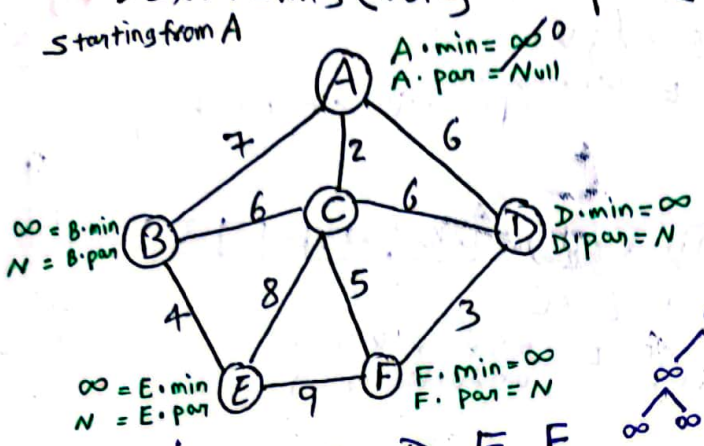
→ Prim's Algorithm (using minheap & adjacency list):

1. Create minheap for vertices
2. Find min.m from minheap & delete
3. Find Adj(A) in adjacency list
4. Decrease key operation

Repeat until minheap is empty

→ Ex: Prims (using minheap & adj. list)

Starting from A



	A	B	C	D	E	F	
2. Find min^m from minheap & Delete lgV	~~0~~ / ~~N~~	∞ / N	∞ / N	∞ / N	∞ / N	∞ / N	⇒ 1. Create minheap for vertices O(V)
A		7 / A	~~2 / A~~	6 / A			⇒ 3. Find Adj(A) in adjacency list 3; 4. Decrease key operation 3·lgV
lgV C		6 / C		6 / A	8 / C	~~5 / C~~	⇒ 5; 4·lgV
lgV F		6 / C		~~3 / F~~	8 / C		⇒ 3; 1·lgV
lgV D		~~6 / C~~			8 / C		⇒ 3; 0·lgV
lgV B					~~4 / B~~		⇒ 3; 1·lgV
lgV E							⇒ 3; 0·lgV

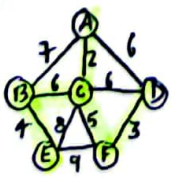
$$TC = \underbrace{V}_{\text{Build Heap}} + \underbrace{V \times \lg V}_{V \times \text{Find min}^m \text{ in minheap \& delete}} + \underbrace{(3+5+3+3+3+3)}_{\text{sum of adjacents} = 2E} + \underbrace{E \times \text{Decrease key}}_{= E \times \lg V}$$

$$= V + V\lg V + 2E + E\lg V$$

$$= V\lg V + E\lg V = (V+E)\lg V$$

1. Simple graph $= O[(V+E)\lg V]$
2. connected graph $= O[E \cdot \lg V]$
3. complete graph $= O[V^2 \lg V]$

→ Prims to MST



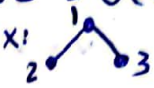
edge selection in prim = AC, CF, FD, BC, BE

@) Graph representation:

I) Adjacency matrix: • Used when edge more (dense graph like Kn)

• SC = $O(V^2)$ [EC]

• Ex:

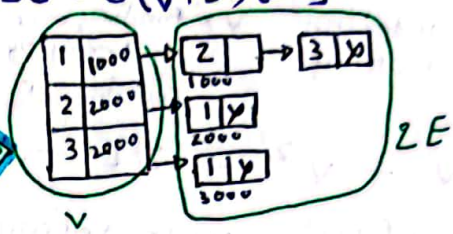


$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 \end{array}$$

II) Adjacency List: • Used when edges less (sparse graph like Null G)

• SC = $O(V+E)$ [EC]

•



@) To Find Adj(X) we use Degree (X):

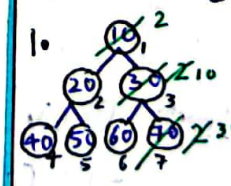
1. Matrix
   Deg (X) → O(V) [EC]
2. List
   Deg (X) → BC: O(1) (next → null); WC: O(V−1) = O(V) (next → Long chain)

@) Decrease key operation:

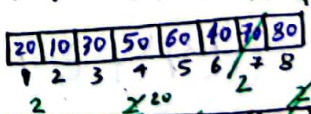
1. In Heap → O(lgV)
2. Unsorted array → O(1)
3. Sorted array → O(V)
4. Sorted DLL → O(V)

decreaseKey(a, 7, 2) ⇒ 7 index पर जो 5 value उसे 2 कर दो

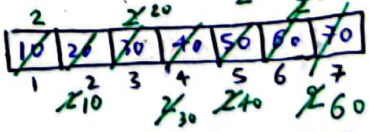
1.



2.



3.



4. Same like 3

one Scan & swaps

→ Prims (using minheap & adj. matrix):

$TC = V + V \times \lg V + V \times V + E \times \lg V$

- V: Build minheap
- $V \times \lg V$: Finding min^m in minheap & delete
- $V \times V$: $V \times$ Find adj(x) in adj. matrix; $V \times 1$ entire row scan
- $E \times \lg V$: $E \times$ Decrease key operation on min heap

$= O(V^2 + E \cdot \lg V)$

→ Prims (using unsorted array & adj. list):

∞	∞	∞	0	∞	∞	∞

$= V \times 1 + V \times V + 2E + E \times C$

- $V \times 1$: $V \times$ insertion in array
- $V \times V$: $V \times$ Finding min^m in Array by LS & delete it.
- $2E$: sum of adjacents
- $E \times C$: $E \times$ Decrease key in unsorted array

$= V + V^2 + 2E + CE = O(V^2)$

→ Prims (using unsorted array & adj. matrix):

$= V \times 1 + V \times V + V \times V + E \times C$

- $V \times 1$: $V \times$ insertion in array
- $V \times V$: $V \times$ Finding min^m in Array by LS & delete it
- $V \times V$: $V \times$ Find Adj(x) in adj. matrix
- $E \times C$: $E \times$ Decrease key in unsorted array

$= O(V^2)$

→ Prims (using sorted array & adj. list):

0	∞	∞	∞	∞	∞	∞

$= (V \times \text{insertion in Array} + \text{Sorting}) + V \times 1 + 2E + E \times V$

- Sorting: $O(V)$ time by Insertion Sort ∵ Array almost sorted
- $V \times 1$: $V \times$ Finding min^m in Sorted array & delete it
- $2E$: sum of adjacents
- $E \times V$: $E \times$ decrease key in Sorted array

$= 3V + 2E + EV = O(EV)$

→ Prims (using sorted doubly L.L. & adjacency matrix)

0	∞	∞	∞

$= (V \times \text{insertion in DLL} + \text{Sorting}) + V \times 1 + V \times V + E \times V$

- $V \times 1$: $V \times$ Finding min^m in Sorted D.L.L & delete it
- $V \times V$: $V \times$ Find Adj(x) in adj. matrix
- $E \times V$: $E \times$ decrease key in Sorted D.L.L

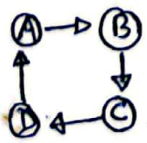
$= (V \times 1 + V)$

$= O(V^2 + EV)$

if complete graph $\Rightarrow O(V^2 + V^2 V)$

$= O(V^3)$

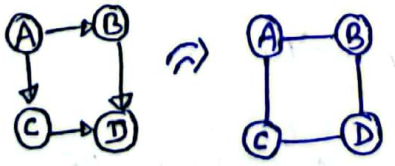
@) Strongly connected graph



SCG ✓ ⇒ WCG ✓

Path exists b/w every 2 pair of vertices

Weakly connected graph



SCG ✗ WCG ✓

After removing direction, path exists b/w every 2 pair of vertices.

*) Single Source Shortest path:

→ Given single vertex to ~~all~~ any other ~~vertices~~ vertex shortest path determination purpose we have 3 algorithms:
1) Dijkstra 2) Bellman ford 3) BFT

→ ① Dijkstra [+ve edge weight] $O[(V+E)\lg V]$
② Bellman ford [-ve or +ve] $O(VE)$
③ BFT [unweighted graph] $O(V+E)$

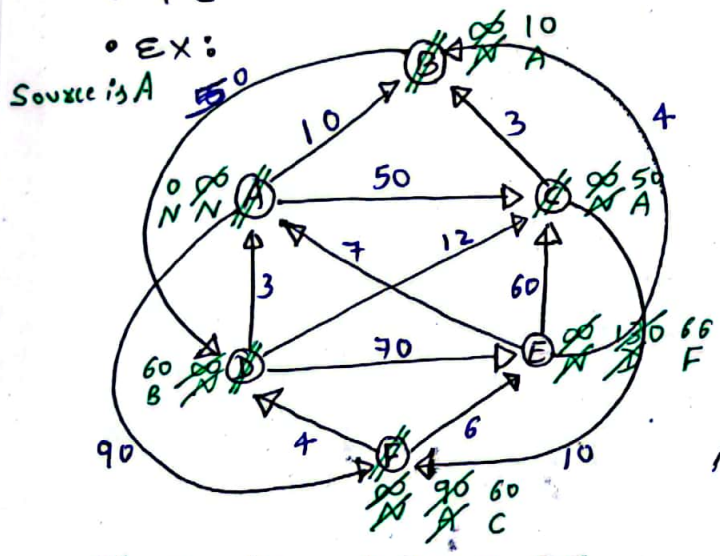
25

→ Dijkstra = Greedy
Bellman ford = DP
BFT =

→ Dijkstra (Find shortest path b/w any two vertices in G with no-ve weights)

- Algorithm = Same as Prims
- TC = Same as Prims
- Ex:

Source is A



1) Sequence of vertices identified by Dijkstra Algo.
⇒ ABCDFE

2) Cost & shortest path from A to E

E ↑ 6 F ↑ 10 C ↑ 50 A

cost = 66
Path = ACFE

Note:) ∅ के बाद सिर्फ उन vertices को देखो which are not over

∅ का order A B C D F E
0 10 50 60 60 66

#) Some important notes :-

•) To find median of unsorted arrays, use selection algorithm ⇒ $O(n)$ [WC]

•) For two matrix multiplication A×B, where A & B are 2D arrays :-
→ TC is independent of RMO or CMO
→ Program execution time is dependent
i.e. its best if A in RMO & B in CMO

•) In graph with 'n' vertex, if k-edges are tree edges then in DFT of G no. of connected components = n-k

•) Logic to find A^T :-

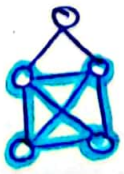
method-1 Outplace :-
```
B[rows][cols]
for(i: 0 to rows-1)
for(j: 0 to cols-1)
    B[i][j] = A[j][i]
```

method-2 Inplace :-
```
for(i: 2 to N)
for(j: 1 to (i-1))
    swap(A[i][j], A[j][i])
```
i.e. Traverse half matrix & swap.

•) Min^m no. of comparisons to sort 'n' no.s = $\lceil \lg(n!) \rceil$

•) To get TC $\Theta(n \cdot \lg n)$ [EC] in Quicksort :-
→ Median of 'n' elements can be found in $O(n)$ time
→ In every pass find median & select it as pivot.

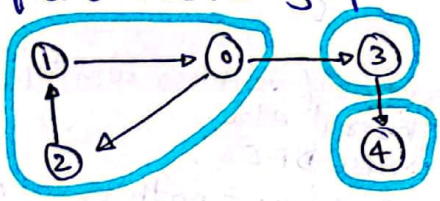
•) Clique : Subset of vertices of undirected graph such that every 2 distinct vertices of clique are adjacent



i.e. a complete subgraph inside original graph

CamScanner

- In the BFT tree T, if cycle is there, then G has strongly connected component. Each cycle in T forms SCC.
- In the BFT tree T, if any cycle is there, then topological sort is not possible.
- Strongly connected components :-
  - A directed graph is strongly connected if there is a path b/w all pairs of vertices.
  - A SCC of a directed graph is a maximal strongly connected subgraph.
  - Ex:



3 SCCs in G

#) Dynamic programming:

*) → It is intelligent brute force

→ DP = Optimal Sol^n | more time | All possibilities covered | Always correct answer

Greedy = Optimal Sol^n | Less time | few " " | Few times correct

→ Applications of DP:

1. Fibonacci series
2. Longest common Subsequence
3. 0/1 knapsack
4. Matrix chain mul.^{n}
5. Floyd warshall
6. Sum of subsets
7. Bellman ford

→ when all function calls are distinct DP becomes Brute force. So, first draw the normal brute force tree & observe if repeating fx^n calls are there or not.

*) Fibonacci series:

→ By Brute force:

```
f(n) // Recursive
{ if (n ≤ 1) return(n)
  else
  { a = f(n-1)   → T(n-1)
    b = f(n-2)   → T(n-2)
    c = a+b      → c
    return(c)
  }
}
```

```
f(n)//non-Rec
{ a=0
  b=1
  for(2 to n)
  { c=a+b
    a=b
    b=c
  }
}
```

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

$\}\, n : SC$

$= O(2^n) : TC$

TC = O(n)

SC = O(1) (Be the one)

→ By DP:

```
f(n) ⟹ C
{ if (n ≤ 1) return(n)
  else
  { if (Table[n-2] == null)
        Table[n-2] = f(n-2)
    if (Table[n-1] == null)
        Table[n-1] = f(n-1)
    Table[n] = Table[n-1] + Table[n-2]
    return (Table[n])
  }
}
```

C

Table	null	null	null	. . . .	null
	0	1	2		n

TC: No. of distinct fx^n calls × cost of one fx^n call

$(n+1) \times C \Rightarrow O(n)$

SC: Stack + Table

$n + n+1 \Rightarrow \Theta(n)$

*) Longest Common Subsequence:

→ x = Sequence 1 of length m

y = Sequence 2 of length n

→ LCS(m,n) = 0, if m=0 or n=0 or both zero

→ Approach:

x = A B A A B A

y = A B B A

LCS(6,4) = 1 + LCS(5,3)

LCS(5,3) = 1 + LCS(4,2)

LCS(4,2) = max{LCS(4,1), LCS(3,2)}

→ By Brute force:

```
f(m,n)
{ if (m=0 or n=0) ret(0)
  else
  { if (x[m] == y[n]) // BC (unary tree)
        ret(1 + f(m-1, n-1)) → T(m-1, n-1)
    else
    { // WC, AC (Binary Tree)
      a = f(m-1, n)  → T(m-1, n)
      b = f(m, n-1)  → T(m, n-1)
      c = max(a,b)   → c
      ret(c)
    }
  }
}
```

WC, AC

$T(m,n) = T(m-1,n) + T(m,n-1) + c$

$\}\, m+n$

$= c \cdot 2^{m+n}$

$= O(2^m \cdot 2^n)$

BC

$= T(m-1, n-1)$

$\}\, \max(m,n)$

$= \min(m,n)$

→ By DP

```
f(m,n){ //Recursive
if (m=0 or n=0) ret(0)
else{ if(x[m]==y[n])
        { if (T[m][n]==null)
              ret(1+f(m-1,n-1))
        }
      else
        { if(T[m-1][n]==null)
              a = f(m-1,n)
          if(T[m][n-1]==null)
              b = f(m,n-1)
          c = max(a,b)
          ret(c)
        }
}
```

//i.e. evaluate only if Table cell doesn't contain the value

```
f(m,n){ // non-Recursive
if(m=0 or n=0) ret(0)
else{ for(i=m; i≤n; i++)   actually m to 1
      for(j=n; j≤m; j++)   & n to 1
        { if(x[i]==y[i])
              T[i][j]=1+T[i-1][j-1]
          else{ a = T[i-1][j]
                b = T[i][j-1]
                T[i][j]= max(a,b)
              }
        }
}
```

TC: WC & AC — DFCs × Cost of 1 FC

$m \cdot n \times 1$

$= O(m \cdot n)$

BC — Always matching

$= O(\min(m,n))$

SC: Stack + Table

$(m+n) + m \cdot n$

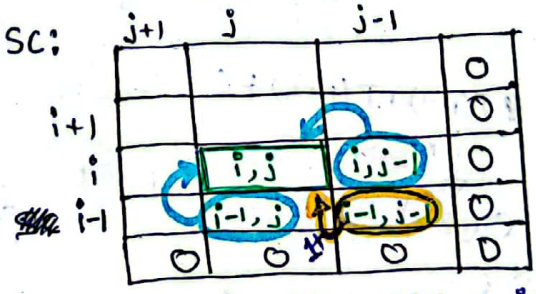
$= O(m \cdot n)$

Sample Table
m=4 n=3

T	3	2	1	0
4	null	n	n	0
3	n	n	n	0
2	n	n	n	0
1	n	n	n	0
0	0	0	0	0

T.C.: $O(m \cdot n)$ [EC]

SC:



at a time 2 rows or 2 cols required

⇒ $= O(2 \text{ or } 2C)$

→ if

→ else

Note:
- Only diagonal values are required in BC (always match)
- Non-Rec. → Fill table in any order (RM/CM)
  Rec → Fill table in postorder manner (RM/CM)
- Recursive program (without DP) → TC by RR for time
  Recursive Program (with DP) → TC by calculating DFCs & cost of 1 FC

*) 0/1 Knapsack:

→ Using Greedy it may fail, so we use DP.

→ By Brute force $\left(\begin{array}{ccc} O_1 & O_2 & O_3 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}\right)$ 2^0

Sac size = m = 16

Items = N = 5

O_i	O_1	O_2	O_3	O_4	O_5 ← as of now here
P_i	25	70	50	10	60
w_i	5	3	2	4	5 a

b

f(m,n) = max^m profit we'll get with 'm' capacity & 'n' objects

As of now we're at n^{th} position

```
f(m,n)
{ if(m=0 or n=0) ret(0)
  else
  { if (w[n]>m) // simply throw (BC)
        ret( f(m,n-1)) → T(m,n-1)
    else //WC,AC
    { a = f[m-w[n],n-1]+P[n] // object taken → T(m-w[n],n-1)
      b = f(m,n-1) // object not taken → T(m,n-1)
      c = max(a,b) → C
      return(c)
    }
  }
}
```

TC: WC, AC $O(2^n)$ BC $O(n)$

SC: $O(n)$ $O(n)$

Example: m = 8, n = 4

Profit → 0 1 2 5 6

weight → 0 2 3 4 5

i ≡ which weight is available

j ≡ My available capacity

i \ j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
2	0	0	1	1	1	1	1	1	1
3	0	0	1	2	2	3	3	3	3
4	0	0	1	2	5	5	6	7	7
5	0	0	1	2	5	6	6	7	8

cell[4][7] = max{3, 5+2}

didn't take 4th item so, ऊपर से 3 आया

Took 4 weighted item so, Profit of this item is 5

current row से एक ऊपर जाओ and see (7-4) i.e. 3rd entry i.e. "2"

→ By DP:-

```
f(m,n)
{if(m=0 or n=0) ret(0)
 else
 {if (w[n]>m) {if(T[m][n-1] == Null)
                 T[m][n-1] = f(m,n-1)
                 T[m][n] = T[m][n-1] }
  else
  {if(T[m-w[n]][n-1] == Null)
       T[m-w[n]][n-1] = f(m-w[n], n-1)+P[n]
   if(T[m][n-1] == Null)
       T[m][n-1] = f(m, n-1)
   if(T[m][n] == Null)
       T[m][n] = max{T[m-w[n]][n-1], T[m][n-1]}
  }
 }
return(T[m][n])
}
```

TC: WC, AC BC $= O(n)$

DFCs × Cost of each

$m \cdot n \times 1$

$= O(m \cdot n)$

SC: Stack + Table Stack

$n + m \cdot n$ n

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

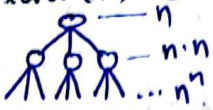
Note: 0/1 Knapsac by DP TC is actually $\simeq 2^n$ bec. very less repetitions

*) Matrix chain Multiplication:

→ In a chain of matrices multiplied, what should be the order of mul.^{n} of the matrices to get optimal no. of mul.^{n}s

→ TC w/o DP

n level (n-1) ary tree



$= O(n^n)$

with DP

DFC's × cost of 1 DFC

$n^2 \times n$

$= O(n^3)$

→ SC

$= O(n)$

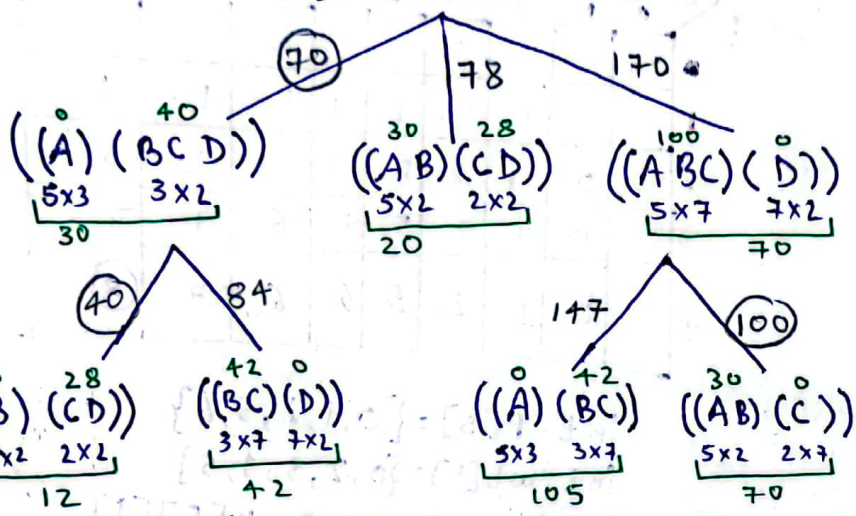
Stack + Table

$n + n^2$

$= O(n^2)$

→

A (5×3) B (3×2) C (2×7) D (7×2)



optimal mul.^{n}s = 70

order:

((A)((B)(CD)))

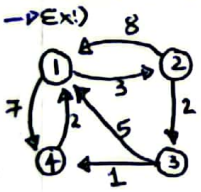
*) Floyd warshall (all pair shortest path):

→ works on directed weighted graph

→ TC: $O(V^3)$

SC: $O(V^2)$

→ Ex:)



$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & \infty \\ 3 & 5 & \infty & 0 & 1 \\ 4 & 2 & \infty & \infty & 0 \end{array} \qquad \pi^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & - & 1 \\ 2 & 2 & 2 & 2 & - \\ 3 & 3 & - & 3 & 3 \\ 4 & 4 & - & - & 4 \end{array}$$

$$D^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array} \qquad \pi^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & - & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 3 & 3 & 1 & 3 & 3 \\ 4 & 4 & 1 & - & 4 \end{array}$$

$$D'[2,3] = \min\{D^0[2,3],\ D^0[2,1] + D^0[1,3]\}$$
$$2 < 8 + \infty$$

$$D'[2,4] = \min\{D^0[2,4],\ D^0[2,1] + D^0[1,4]\}$$
$$\infty > 8 + 7$$

→ By DP:-

```
for(k=1; k≤n; k++)
{
    for(i=1; i≤n; i++)
        for(j=1; j≤n; j++)
            D[i,j] = min{D[i,j], D[i,k]+D[k,j]}
}
```

i.e.

$$D^k[i,j] = \min\{D^{k-1}[i,j],\ D^{k-1}[i,k] + D^{k-1}[k,j]\}$$

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array} \qquad \pi^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 3 & 3 & 1 & 3 & 3 \\ 4 & 4 & 1 & 2 & 4 \end{array}$$

$$D^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 7 & 0 & 2 & 3 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array} \qquad \pi^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 2 & 3 \\ 2 & 3 & 2 & 2 & 3 \\ 3 & 3 & 1 & 3 & 3 \\ 4 & 4 & 1 & 2 & 4 \end{array}$$

→ copied from previous matrix

$$D^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 5 & 0 & 2 & 3 \\ 3 & 3 & 6 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array} \qquad \pi^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 3 & 5 & 3 \\ 2 & 4 & 2 & 2 & 3 \\ 3 & 4 & 4 & 3 & 3 \\ 4 & 4 & 1 & 2 & 4 \end{array}$$

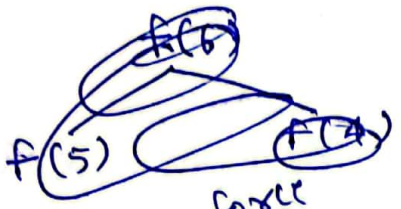
Q → path: 3 to 2 ?

3 → 4 → 2 i.e. 3 → 4 → 1 → 2

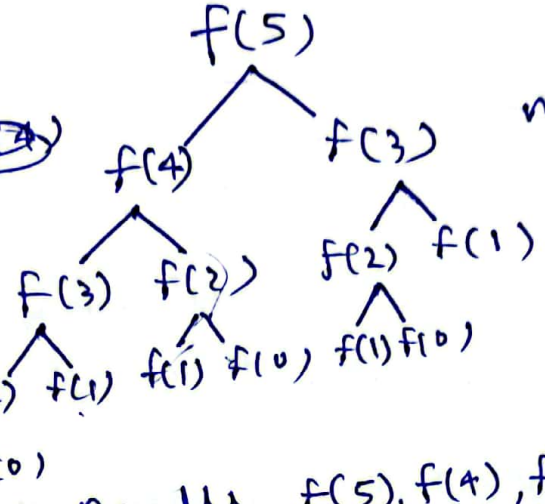
cost = 1 + 2 + 3 = 6

CamScanner

Fibonacci Repeating fx^n calls



Brute force tree



many fx^n calls repeating

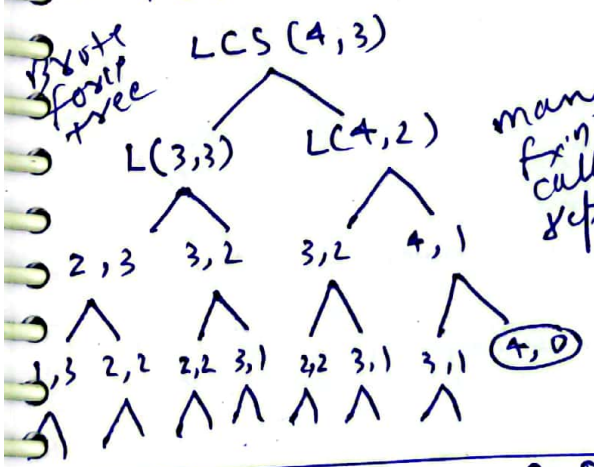
Distinct fx^n calls f(5), f(4), f(3), f(2), f(1), f(0)

LCS Repeating fx^n calls

W C

x = A A A A (1 2 3 4) y = B B B (1 2 3)

Brute force tree



many fx^n calls repeating

Distinct fx^n calls

LCS (4, 15)

3 2 1 0 14 13 ⋮ 0

(m+1) ways (n+1) ways

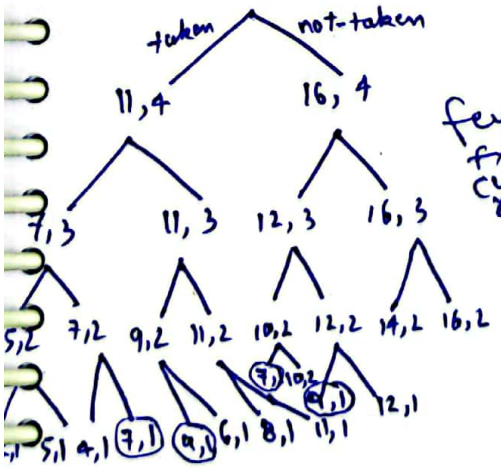
total = (m+1)(n+1) ways

=> (m·n) DFC

0/1 Knapsack

	O_1	O_2	O_3	O_4	O_5
P	25	70	50	10	60
W	5	3	2	4	5

f (m, n) = f (16, 5)



few fx^n calls repeating

DFCs

f(m, n) = f(16, 5)

15 14 ⋮ 0 4 3 ⋮ 0

m+1 n+1

DFC = O(m · n)

*) Sum of subsets problem: (Similar to 0/1 Knapsack)

→ A[n] में से find any subset such that sum = m

→ By DP

$f(m, n)$ — m: required sum; n: as of now we're at n^{th} position

Given A[n]

```
{ SS = ϕ  // Selected Subset
  if (m == 0) return (SS)
  if (n == 0) return (-1)
  if (A[n] > m) ret(f(m, n-1))  // simply throw
  f(m - A[n], n-1); SS = SS ∪ A[n]  // object taken
  f(m, n-1)  // object not taken
}
```

→ TC: By DP

DFC's × cost of 1

$m \cdot n \times 1$

$= O(m \cdot n)$

By Brute force $= O(2^n)$

→ SC: Stack + Table

$n + m \cdot n$

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

Note:) 0/1 KS $\leq_p$ SOS

i.e. 0/1 KS $\xrightarrow{NPC}$ (Poly) $\xrightarrow{NPC}$ SOS

Note:) Table

	0	1	2	. . .			n
0	T	F	T	T	F	F	T
1	T	T	F	...			
2	T	F	T	...			
. . .	F	F	F	...			
n	.	.	.				

T → yes, such subset is there

*) Bellman ford (Single source shortest path):

→ Relax edges n-1 times; n = no. of vertex

→ $\overset{x}{(u)} \xrightarrow{\alpha} \overset{y}{(v)}$

if $(d[u] + c[u,v] < d[v])$ $\{d[v] = d[u] + c[u,v]\}$

i.e. if $(x + \alpha < y)$ $\{d[v] = x + \alpha\}$

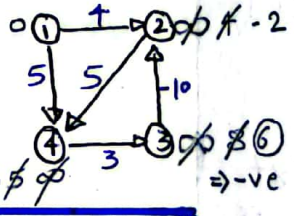
Here, d = distance, c = cost matrices resp

→ Algo

1. Prepare edge list
2. Select any vertex as source vertex
3. Do relaxation of all edges (n-1) times.

→ TC: $E \times (V-1) \Rightarrow O(VE)$

→ Ex:

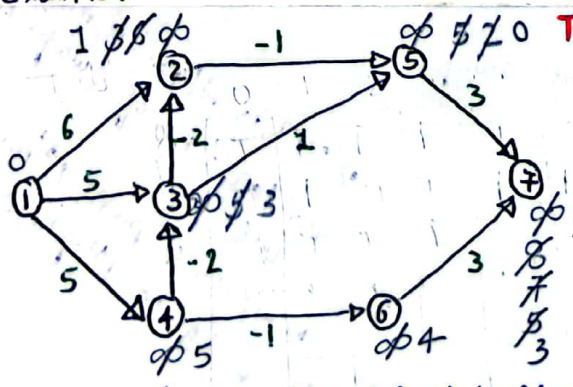


gets relaxed n^{th} time ⇒ -ve wt. cycle there

→ To detect -ve wt. cycle, first do relaxation n-1 times, then try to relax one more time, if it gets relaxed one more time, -ve wt cycle detected.

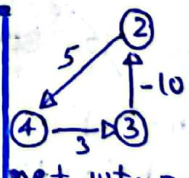
→ Bellman ford doesn't fails on -ve edge wt's, but fails on -ve wt. cycles.

→ Example:



Table

	①	②	③	④	⑤	⑥	⑦
	0 N	∞ N	∞ N	∞ N	∞ N	∞ N	∞ N
i=0	0 ①	~~6~~ 3 ~~0~~ 3	~~5~~ 3 ~~0~~ 1	5 ①	5 ②	4 ④	~~8~~ 7 ~~0~~ 6
i=1							
. . .		.	.	.			
i=5							

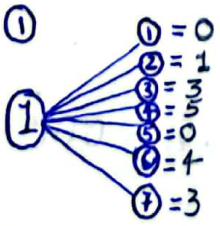


net wt. = -2

Edge list: (1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3), (4,6), (5,7), (6,7)

Source : ①

Shortest Distances:



Relaxations done = 7 - 1
= 6 times

#) Sorting techniques (contd.):

*) Bubble Sort:

i/p: 80 20 15 70 10 11 39
(1 2 3 4 5 6 7)

After Ist pass [20 15 70 10 11 39] 80

IInd pass [15 20 10 11 39] 70 80

IIIrd pass [15 10 11 20] 39 70 80

4th pass [10 11 15] 20 39 70 80

5th pass [10 11] 15 20 39 70 80

6th pass [10] 11 15 20 39 70 80

→ Bubble Sort on n elements:-

1. passes = n-1
2. Comparisons = $(n-1)+(n-2)+\dots 3+2+1 = \frac{(n-1)n}{2}$
   $= O(n^2)$
3. Swaps → max.m $= \frac{(n-1)n}{2} = O(n^2)$
   → min.m $= 0$
4. TC : comparisons + Swaps
   $n^2 + \begin{cases} 0 \\ n^2 \end{cases}$
   $= O(n^2)$ [EC]
5. Inplace
   Stable

→ Optimized bubble Sort:

- while doing bubble sort if we find this pass (current pass) have zero swaps, Stop the algo.

P0: (60 20 30 40 50) swaps = 0

P1: (20 30 40 50)(60)
swaps = 4
swaps = 0 [again initialize to 0]

P2: (20 30 40)(50 60)
swaps = 0
Stop.

TC: $O(n)$ [BC]
$O(n^2)$ [WC, AC]

*) Selection Sort: (Target = no. of swaps as min.m as possible)

i/p: 50 30 40 80 15 60 45
(1 2 3 4 5 6 7)

	comparisons	swaps
P1: i=1, min = i = ~~1~~ 3, swap[a[i], a[min]] → 10 (30 50 80 15 60 45)	(n - 1)	1
P2: i=2, min = ~~2~~ 5, swap(2, 5) → 10 15 (50 80 30 60 45)	(n - 2)	1
P3: i=3 → 10 15 30 (80 50 60 45)	(n - 3)	1
... (n-1) times	...	...

→ This Algo. is meant for least no. of swaps in WC

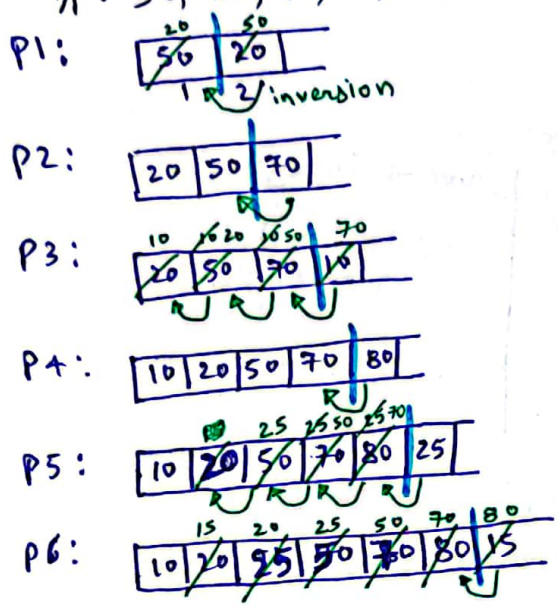
→ After 1st pass, 1st min.m will go to 1st place

1. passes = n-1
2. comparisons = $n(n-1)/2$
3. Swaps = n-1
4. TC = comparisons + swaps = $n^2+n = O(n^2)$ [EC]
5. Inplace
   Unstable

CamScanner

*) Insertion Sort:

i/p: 50, 20, 70, 10, 80, 25, 15



Passes: n−1

Every pass (TC): O(1) [BC]
O(n) [WC]

TC: BC [Asc. order]
comparisons + swaps
$1 \times (n-1) + 0$
$= O(n)$

WC [Desc. order]
c + s
$n^2 + n^2$
$= O(n^2)$

Comparisons $= 1+2+\cdots n-1 = (n-1)n/2$
Swaps $= 1+2+\cdots n-1 = (n-1)n/2$

AC
n/2 BC | n/2 WC
$n/2 * c$ | $n/2 * n$
$= O(n^2)$

Notes:

•) If array already sorted or almost sorted then Quicksort is not preffered cuz it takes $O(n^2)$ but Insertion sort is preffered cuz it takes $O(n)$ | Comparisons = n−1, Swaps = 0

•) If random arrays, Quicksort better
→ QS ⇒ $n \lg n$
→ IS ⇒ n^2

•) wc swap purpose ⇒ Selection sort
BC " " ⇒ Insertion sort
practical " ⇒ QS

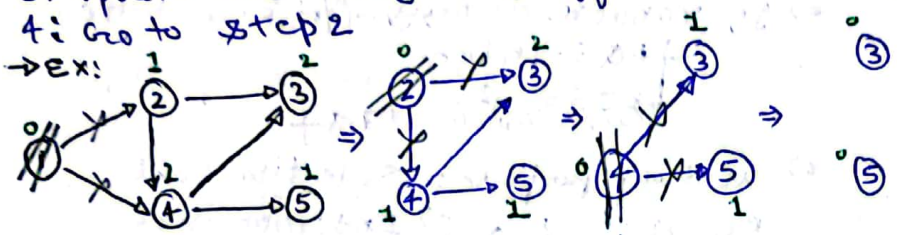
•) If array contain atmost 'n' inversions, then insertion sort takes O(n) time cuz. array almost sorted.

Topological Sort: [O(V+E)]

→ G should be directed & acyclic (i.e. DAG)
→ A DAG have one or more topological ordering
→ Algo:

1: For every vertex calculate indegree.
2: Delete the vertex with indegree zero.
3: Update the indegree of affected vertices.
4: Go to step 2

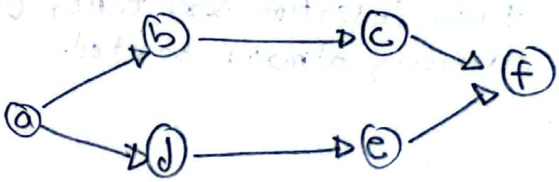
→ Ex:

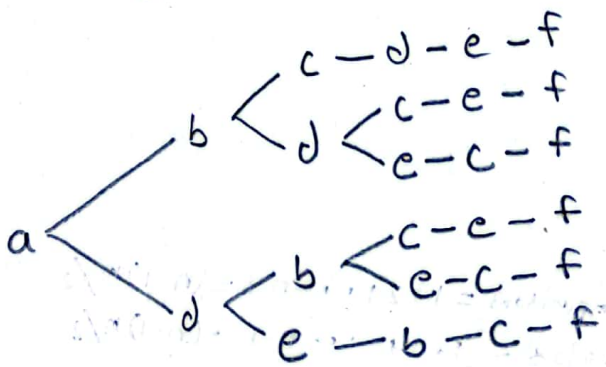


~~ → indegree

Order: 1 2 4 3 5
or 1 2 4 5 3

Ex: Different topological ordering of are





6 ✓

Ex:

*) Radix Sort:

- It is non-comparison based sorting algo.
- It is outplace & stable
- i/p:

	0	1	2	3	4	5	6	7	8	9
A	237	146	259	348	152	163	235	48	36	62

- To extract one's, ten's, hundred's digit.

1's ← (A[i]/1) % 10

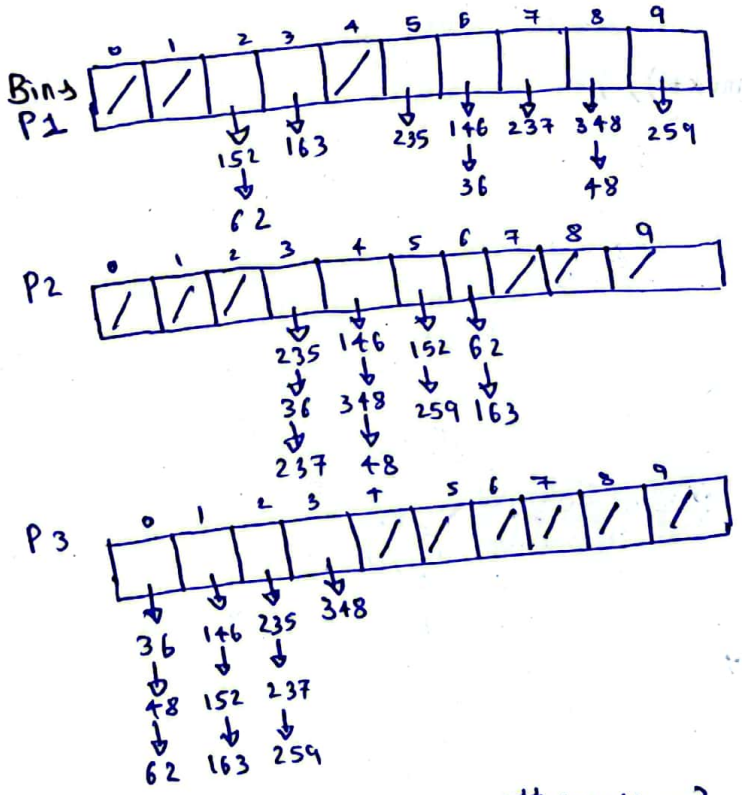
10's ← (A[i]/10) % 10

100's ← (A[i]/100) % 10

P1: 152, 62, 163, 235, 146, 36, 237, 348, 48, 259

P2: 235, 036, 237, 146, 348, 048, 152, 259, 062, 163

P3: 36, 48, 62, 146, 152, 163, 235, 237, 259, 348



TC: No. of passes = d {d ⇒ d^{th} digit no.}

Every pass ⇒ n insertions into bin ⇒ O(n)

⇒ TC = O(dn) ≃ O(n)

$d = \lceil \log_b K \rceil$

where, K = max'm possible value

b = base {like 10 for decimal}

SC: Array with 10 indices and containing n key in chained manner ⇒ O(n)

Counting Sort:

	0	1	2	3	4	5	6	7	8	9
A	6	3	9	10	15	6	8	12	3	6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C	0	0	0	2	0	0	0	3	0	0	1	1	1	0	0	0	0	1

	0	1	2	3	4	5	6	7	8	9
A	3	3	6	6	6	8	9	10	12	15

TC: $O(n+m)$; $n \rightarrow$ no of elements in A
$m \rightarrow$ value of max element in A

```
void f(int A[], int n)
{ int max,i,j ;
  int *C;
  max = findMax (A, n);
  c = (int *) malloc (sizeof(int) * (max+1)) ;
  for ( i=0; i< max+1 ; i++)
          c[i] = 0
  for (i=0; i<n; i++)
          c [A[i]] ++;
  i=0; j=0;
  while ( i < max+1)
  { if (c[i]>0)
          { A [j++] = i;
            c[i]--;
          }
    else
          i++;
  }
}
```