Algo + C dos → 20-24 marks

Design & Analysis of Algorithms

20 — 8K (July)

I Fundamentals of Algo [4-6]
- ↳ Asympotic Notation
- ↳ TC/SC of loops
- ↳ TC/SC of Recursive algo
- ↳ Methods to solve Recurrence Relation [Substitution, Recursive tree, Master theorem / methods].

II Algo. design Techniques [6]
- ↳ Divide & Conquer [merge sort, Quicksort, Binary search etc.]
- ↳ Greedy method [single source shortest path Algo, MST [Prims, Kruskals], Huffman coding etc].
- ↳ Dynamic programming

III Tree, graphs, Binary heaps, [4-6]
- graph traversal,
- Sorting algo.

Introduction to Algo

- Algorithm is step by step representation of computer program (finite no. of instruction to perform some task)

- Criterias :-

(1) Input: 0 or more external I/P

(2) O/P : Atleast one O/P

(3) Finiteness :- Finite I/P → Algo → Halt [Terminal in finite time]

- Algo^n must terminate in finite amount of time

(4) Definiteness [Deterministic Algo] each step of algo must have unique solution

- Non-deterministic Algo.

each step of algo may have finite # of solution & algo must choose correct solution in 1st attempt

[not possible to run in comp uter system]

i.e Deterministic (DA) Algo :- a[1.....n] && x: searching element.

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

wc comp = n

i.e non-deterministic (NDA) Algo

```
j = choose(1,n);
if(x==a[j])
    return(j)
else return(-1)
```

wc comp = 1

NOTE
- NDA = DA in compare to power not in compare to time

Steps to Design Algo

1 Devise an algo:

- Design an Algo to solve an problem by using best possible design technique

Problem → Divide & conquer, greedy, DP, Brute force

2 Validation of an Algo:

- Validated Algo. to test Algo working Correctly or not

legal I/P → Algo → correct O/P

3 Analysis of an Algo [efficient]

- estimate CPU execution time & MM space required to complete execution of algo.

[Coding]

4 Testing of program :- system testing methods.

Decidable problem	undecidable problems.
• problem for which efficient algo exist	• problem for which no efficient algo exist
• Problem which can solve in polynomial time by using deterministic algo	• problem which required exponential time to solve by using Deterministic algo

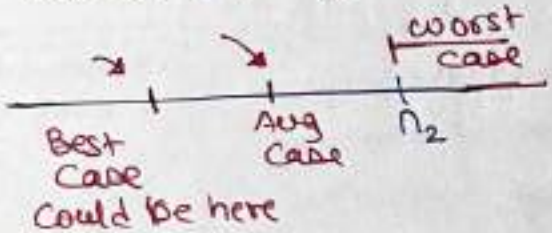
Major Misconceptions.

~~O is describing worst case~~

~~Ω is describing Best case~~

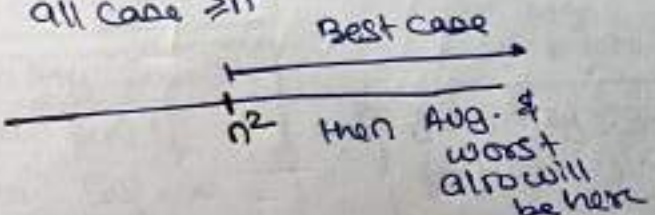
Best case, Worst case, Avg. case } can use any notation $O, \Omega, \Theta, o, \omega$

ex: in w.c : $\Omega(n^2)$
that means T.C $\Omega(n^2)$ (no)
worst case $\geq n^2$
all case $\geq n^2$ ✗

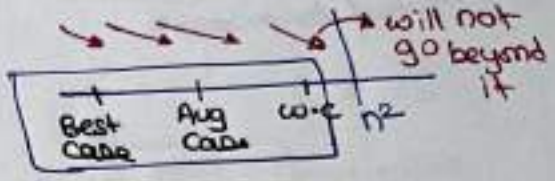


ex: Best case : $\Omega(n^2)$ (yes)
then T.C : $\Omega(n^2)$

Best case $\geq n^2$
all case $\geq n^2$

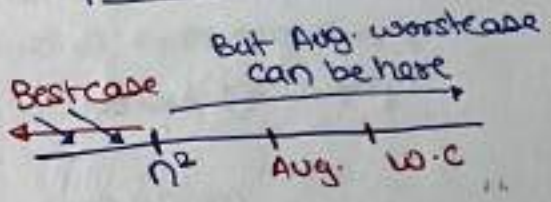


ex: in worst case my algo. is taking $O(n^2)$ does this mean my algo is having T.C $O(n^2)$ (yes)
worst case $\leq n^2$



ex: Best Case : $O(n^2)$
means T.C : $O(n^2)$ (no)

Best case $\leq n^2$
all case $\leq n^2$ ✗



Decidable problem (Tractable problem)	undecidable problem (un-tractable problem)
• for n I/p size ## Computation $n\log n, n^2, n^2\log n, n^3, n^k$ (polynomial functn)	• for n I/p size ## Computation $2^n, 3^n, n!, n^n$ (exponential functn)
finite I/p → Algo problem TC: n^2 → Halt [finite time]	Finite I/p → Algo TC: 2^n → Halt / may not Halt

Asymptotic Notations

$\{\leq, \geq, =, <, >\}$ → real
$\{O, \Omega, \Theta, o, \omega\}$ → Asy comp

• **Asymptotic comparision:-**
growth rate comparision of non negative fun $f(n), g(n)$ for large n values $(n \to \infty)$

• **g(n) Asy bigger than f(n) iff**
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$f(n) = 5n^2 + 1000n \quad g(n) = n^3 + n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2[5+1000/n]}{n^2[n+1/n]} = \frac{5}{\infty} = 0$

(g(n) Asy bigger)

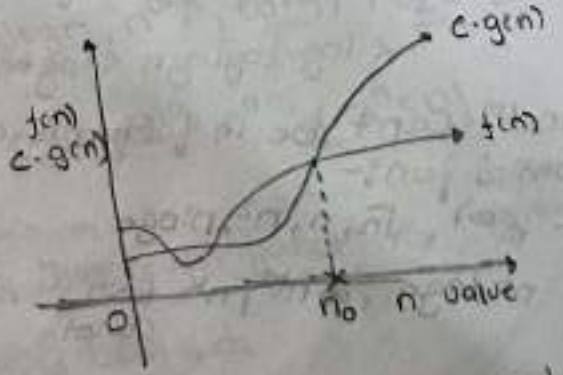
• **g(n) Asy Smaller than f(n) iff**
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

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

- $\frac{c}{\infty} = 0$
- $\frac{c}{c/0} = \infty$
- $\frac{c}{c} = c$

Big O'h Notation [O]

• f(n), g(n) non-negative functn $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all 'n' values, where $n \geq n_0$
[c, n_0 are constant]



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

• $f(n) = O(g(n))$ iff g(n) fun Asy. bigger or equal to f(n).

$f(n) = 10n + 100$

$g(n) = n$

$f(n) \leq c \cdot g(n)$

$10n + 100 \leq 11 \cdot n$ $\forall n$ where $n \geq n_0$

$10n + 100 = O(n)$

$f(n) = 10n + 100$

$g(n) = n^2$

$10n + 100 \leq 1 \cdot n^2$

$\forall n,\ n \geq 17$

$10n + 100 = O(n^2)$

ex: $1 + n + n^2 = O(n^2)$

$O(n^3/n^4 \ldots)$

as, $1 + n + n^2 \leq n^2/n^3/n^4 \ldots$

$n^2 \in O(n^3)$ abuse of notation

$f(n) = 10n + 100$ $\quad g(n) = \log_{10} n$

$f(n) \leq c \cdot g(n)$

$10n + 100 \nleq 20000 * \log_{10} n$

$10n + 100 \neq O(\log_{10} n)$

note: $2^n < n!$

- Categories of functn for Asymptotic comp:-

① $f(n) = \frac{c}{n}, \frac{10}{n^2}, \frac{n}{2^n}, \frac{2^n}{n!}, \frac{\log n}{n}$

Decrement function

$$\frac{2^n}{n!} < \frac{n}{2^n} < \frac{10}{n^2} < \frac{c}{n} < \frac{\log n}{n}$$

Decrement functn

ex: $f(n) = \frac{1000}{n^2}$, $g(n) = 10$ (constant) $\therefore f(n) < g(n)$

② Constant functn

$f(n) = 100$ [constant]

③ log fun's:

$f(n) = \log n$, $(\log n)^{10}$, $\log\log n$

$(\log\log n)^{10}$, $\log\log\log n$,

$(\log\log\log n)^{10}$, $\log n \cdot \log\log n$

$\log\log n \cdot \log\log\log n$

- assume K and then solve

$3^{\log n}$ ×

$(\log n)!$ ×

if $x = \log n$

$\Rightarrow \frac{\log n}{\log\log n}$

$\Rightarrow \frac{x}{\log x}$

$\log\log\log n < (\log\log\log n)^{10} < \log\log n < (\log\log n \cdot \log\log\log n) < (\log\log n)^{10}$

$< \log n \cdot \log\log n < \log n < (\log n)^{10}$

- log functn can't be in factorial and log functn can't be in power

④ polynomial fun:-

$f(n) :- n^{0.1}, \sqrt{n}, n, n^2, n\log n$

$n^2 \cdot \log n$, n^{10}, n^k $k > 0$ const.

- polynomial function are always bigger than log functn

$n^{0.1} < \sqrt{n} < n < n\log n < n^2 < n^2 \log n < n^{10} < n^k$

→ increasing with the value of power

(5) Exponential functn :-

$f(n) = (1.1)^n, (1.01)^n, 2^n, 3^n, n!, n^n$

$(1.01)^n < (1.1)^n < 2^n < 3^n < n! < n^n$

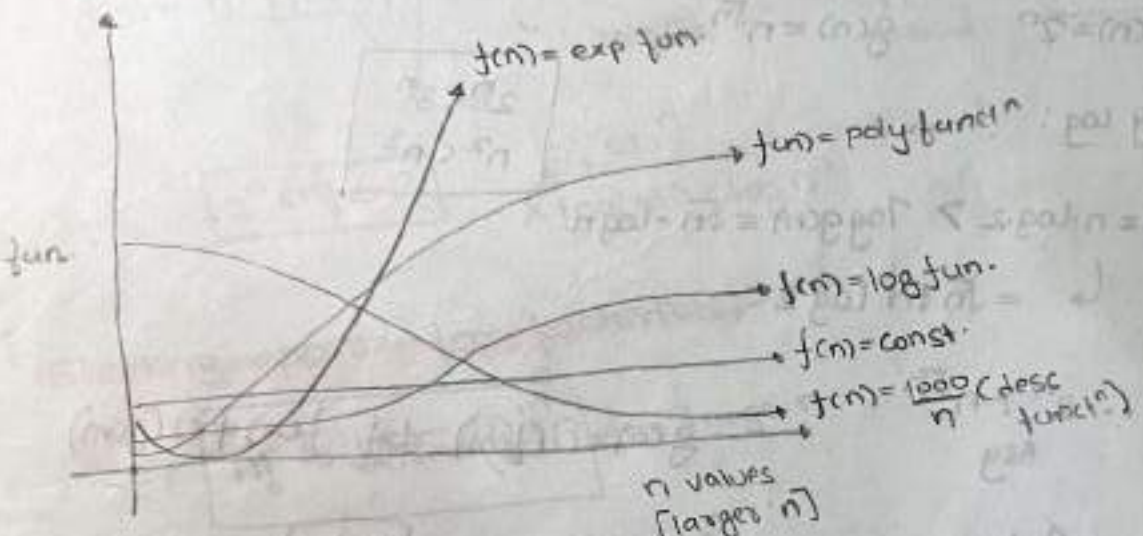
a^n < exp $a>1$ | Const $a=1$ | decfun $a<1$

i.e $(1.1)^n$: exp. fun

$(0.9)^n$: decrement funct

Asy Comp

decrement functn < Const functn < log functn < poly. functn < exp. functn



ans] write fun. Asy Inc. order

(1) $2^n, 10, n!, (\log n)^{10}, \log\log n, n^{0.9}, (0.9)^n, \frac{n}{2^n}$

$\frac{n}{2^n} < (0.9)^n < 10 < \log\log n < (\log n)^{10} < n^{0.9} < 2^n < n!$

$f(n) = (\log n)!$ → to check if given functn is logarithm functn

$\log n = (\log n)!$ we will compare with these as we are checking for logarithm

⇒ taking log both side

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

$= \log(n \log n)$

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

as it's greater than $\log n$ it's exponential functn

[$(\log n)! \neq n\log n$]

To compare Asy. can apply log :-

① if $\log f(n) \overset{Asy}{<} \log(g(n))$
Then $f(n) \overset{Asy}{<} g(n)$

② if $\log(f(n)) = \log(f(n))$
Then $f(n)$, $g(n)$ may/may not Asy equal.

③ if $f(n) < g(n)$
then $\log f(n) < \log g(n)$
(or)
$\log f(n) = \log g(n)$

ex:- $f(n) = n^2$ $g(n) = 2^n$

$\because \log f(n) = 2 \cdot \log(n) < \because \log g(n) = n \cdot \log 2$

ex: $f(n) = n^2$ $g(n) = n^3$

$\because \log f(n) = 2 \cdot \log n$ $= \because \log g(n) = 3 \cdot \log n$

④ if $f(n) = g(n)$
Then $\log f(n) = \log g(n)$

Ques] $f(n) = 2^n$ $g(n) = n^{\sqrt{n}}$

Apply log:

$2^n < 3^n$
$n^2 < n^3$

$\log f(n) = n \cdot \log 2$ > $\log g(n) = \sqrt{n} \cdot \log n$

↳ $= \sqrt{n}\sqrt{n} \log 2$

$\therefore 2^n > n^{\sqrt{n}}$
Asy

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

Ques] $f(n) = n^n \cdot \log n$

$g(n) = n^n \cdot 2^n$

Apply log.

$\log n \log n < n \log 2$
$f(n)$ $g(n)$

$\therefore n^n \cdot n^{\log n} < n^n \cdot 2^n$
$f(n)$ $g(n)$

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

note

- Before applying log funcn 1st remove the common part btw the two funcn so that to prove which is strictly increasing

Ques] $f(n) = n!$ $g(n) = 2^n$

$f(n) = n \cdot (n-1) \cdot (n-2) \cdots \cdots 3 \cdot 2 \cdot 1$ (Const)

$g(n) = 2 \cdot 2 \cdot 2 \cdots \cdots 2 \cdot 2 \cdot 2$ (Const)

both are equal

not equal

$n! > 2^n$ Asy

$2^n = O(n!)$ 44 $n! \neq O(2^n)$

Note:

$r(n) * c \overset{Asy}{\leftarrow} h(n)$ $\times n$

Ques] $f(n) = n!$ $g(n) = n^n$

$f(n) = n \cdot (n-1) \cdot (n-2) \cdots \cdots 3 \cdot 2 \cdot 1$

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

not equal

equal

$\therefore n! < n^n$

$n! = O(n^n)$ 44

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

$2^n < n! < n^n$

$n! < n^n$

$\times \log(n!) < \log n^n$

? **Sterling approx. for factorials:-**

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

Apply log both side.

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

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

$\log n! \approx \frac{1}{2}\log 2\pi e + \frac{1}{2}\log n + n \cdot \log n - n \log e$

$\log n! \overset{Asy}{\approx} n \log n$

$\log n!$ 44 $n \log n$ are Asy equal

$\log n! = O(n \log n)$ 44 $n \log n = O(\log n!)$

Ques] $f(n) = \log_2 n$ $\quad g(n) = \log_b n^a$

a, b are const.

$f(n) = \log_2 n$ $\quad g(n) = a \log_b n$

$\Rightarrow a \cdot \frac{\log_2 n}{\log_2 b}$ (const)

note: $\log_y x = \frac{\log_z x}{\log_z y}$

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

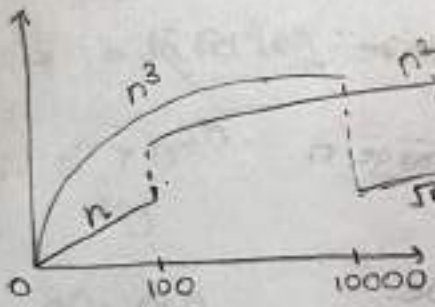
$\Rightarrow \frac{\log_2 n}{a \cdot \log_2 n / \log_2 b}$ } const.

$\therefore f(n)$ $g(n)$ Asy equal

note: $\log_2 n$, $\log_{10} n$, $\log_e n$, $\log_5 n$ — const base

Asy equal

Ques] $f(n) = \begin{cases} n^3 & 0 \le n \le 10000 \\ \sqrt{n} & n > 10000 \end{cases}$ $\quad g(n) = \begin{cases} n & 0 \le n \le 100 \\ n^2 & n > 100 \end{cases}$



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

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

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

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

Ques] $f(n) = \begin{cases} n & \text{for all even } n \\ 2^n & \text{for all odd } n \end{cases}$ $\quad g(n) = \begin{cases} n & \text{for all odd } n \\ 2^n & \text{for all even } n \end{cases}$

n	f(n)		g(n)
10	10	<	2^{10}
11	2^{11}	>	11
12	12	<	2^{12}
13	2^{13}	>	13

for n value ↓

$f(n)$ & $g(n)$ Asy not comparable

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

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

ex: $2^{n+1} = O(2^n)$

$2 \cdot 2^n \le c \cdot 2^n$

$2 \le c$

$c=3, n=1$

ex: $2^{2n} = O(2^n)$

$2^n \cdot 2^n \le c \cdot 2^n$

$2^n \le c$ false

ex: $f(n) = 2^n$ & $g(n) = n^{\sqrt{n}}$

$n \log_2 2$ $\sqrt{n} \log_2 n$

divided by $\sqrt{n}$

$n > \sqrt{n} \log_2 n$

ex: $f(n) = n^{\log_2 n}$ & $g(n) = n^{\log_{10} n}$

$n^{\log_2 n}$ $n^{\log_{10} n}$

taking log with base 2

$\log_2 n \cdot \log n$ $\log_{10} n \cdot \log n$

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

but we can't say $f(n) = g(n)$ ✗

ex: $\log_2 n$ $\log_{10} n$

$n = 2^{10} = 1024 \approx 10^3$

$\log_2 2^{10}$ $\log_{10} 10^3$

$\Rightarrow 10$ $\Rightarrow 3$

So, $n^{\log 2^n} > n^{\log_{10} n}$

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

as base increases, it's value keep on decreases

ex: $f(n) = \log_2 n$ & $g(n) = \log_{10} n$

$\log_{10} n$ same $\log_{10} n$

const. $\log_{10} 2$ so, ignore this

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

- $n^2 = O(n^3)$ set
  - n^2: one element of set
  - $=$: $\in$
- ideally:- $n^2 \in O(n^3)$
- abuse of notation
- mathematical convenience

$n^2 = O(n^3)$

n^2: just one function

$O(n^3)$: set of functions $\{f(n) / f(n) \le n^3\}$

$\{1, c, n, n^2, \log n \dots\}$

So, that's why it should be $\in$

ex: arrange in increasing order

① 2^n ② n^3 ③ $\binom{n}{n/2}$ ④ $n!$ ⑤ $\binom{n}{3}$

$\Rightarrow$ So $b_5 = b_2 < b_1 < b_3 < b_4$

$\binom{n}{3} = \frac{n!}{3!(n-3)!} = n(n-1)(n-2) \sim n^3$

$\binom{n}{n/2} = \frac{n!}{(n/2)!(n/2)!}$ ← $\frac{2^n}{\sqrt{n}}$ by Stirling approx.

comparing $\frac{n!}{((n/2)!)^2}$ with $n!$

multiply by $((n/2)!)^2$ & divide by $n!$

$\frac{((n/2)!)^2 * n!}{n! * ((n/2)!)^2}$

$\frac{n! * ((n/2)!)^2}{n!} > 1 < ((n/2)!)^2$

ex: if $f(n) = O(g(n))$ & $g(n) = O(f(n))$
then $f(n) = \Theta(g(n))$
$f(n) \neq g(n)$
can't say anything about these

ex: $f = \Omega(\log n)$ & $g = O(n)$ $\quad f \geq \log n$, $g \leq n$

~~(a)~~ $g(n) = O(f(n))$
$n \leq O(\log n)$

~~(b)~~ $f(n) = O(g(n))$
$n^3 \leq O(\log n)$

~~(c)~~ $f(n) = \Theta(\log g(n))$
$n^3 = \log n$

(d) $f(n) = \Omega(\log g(n))$
$f(n) \geq \log g(n)$
$\log n \geq \log n$

(e) $f(n) + g(n) = \Omega(\log n)$
$f(n) + g(n) \geq \log n$
$n^2 + \log n \geq \log n$

ex: $n \in \omega(2^{\sqrt{\log n}})$
$n = \omega(2^{\sqrt{\log n}})$
$n > 2(\log n)^{1/2}$
$\log n > (\log n)^{1/2}$

ex: funct^n f & g such that
$f(n)$ is in $O(g(n))$
then
$2^{f(n)}$ is $O(2^{g(n)})$

$2n$ → $f(n)$ ≤ $g(n)$ ← n
$2^{f(n)} \leq 2^{g(n)}$ so, $2^{2n} \notin O(2^n)$
2^{2n} ← → 2^n

ex: if $f(n) = \Theta(g(n))$
then $(f(n))^{g(n)} = \Theta(g(n)^{f(n)})$ } false

$f(n) = \Theta(g(n))$
$2n$ $\quad$ n

$(2n)^n$ $\quad$ $(n)^{2n}$
$2^n \cdot n^n$ $\quad$ $(n^n) \cdot (n^n)$
$2^n < n^n$

ex: $(n!)^{1/n}$
$\frac{1}{n} \log n!$
$\frac{1}{n} * n \log n$
$\log n$

$\log n^n$
$n \log n$
$\log n + \log\log n$
$\log n$

Can't say anything

if $(n!)^{1/n}$ $\quad$ $\log n^n$ → $n \log n$
taking a greater funct^n than $n!$
$(n^n)^{1/n}$ $\quad$ $n \log n$
$n^{n/n}$ $\quad$ $< n \log n$
so if greater funct^n is smaller than it must be small

ex: $2^{(\log n)^2} \neq O(n^{1.5})$
$(2^{\log n})^{\log n} = n^{\log n}$
so,
$n \log n > n^{1.5}$

ex: $\frac{e^{\log n}}{n}$, $2^{n\log n}$, $n^{\sqrt{n}}$

- $\frac{e^{\log n}}{n} = e^{n\log n - \log n}$
- $2^{n\log n} \Rightarrow e^{(\log 2)^{n\log n}}$
  $\Rightarrow e^{n\log n}$
- $n^{\sqrt{n}} = (e^{\log n})^{\sqrt{n}}$
  $\Rightarrow e^{\sqrt{n}\log n}$

$n^{\sqrt{n}} < \frac{e^{n\log n}}{n} < 2^{n\log n}$

ex: $f(n) = 5n\log n$ & $g(n) = n\log 5n$

$n(\log 5 + \log n)$
$n\log 5 + n\log n$
$= n\log n$

So, $\boxed{f(n) = \Theta(g(n))}$

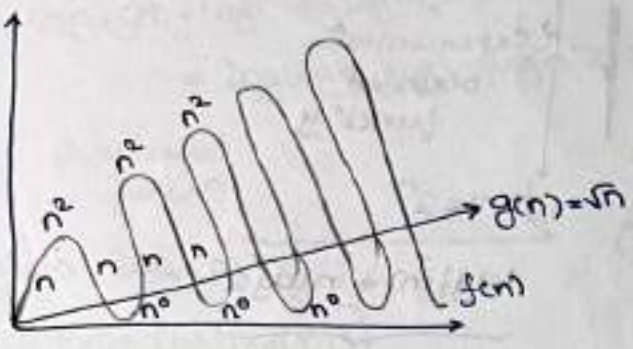
ex: $f(n) = O(3^{\log_{10} n})$ then $f(n)$ is $O(n^2)$ True

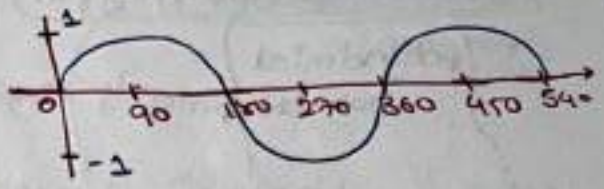
$f(n) \leq n^{\log_{10} 3}$

So, $f(n)$ is $O(n^2)$

So, if $f(x) \leq n^2$'s we can also say that $f(x) \leq n^2$

Ques] $f(n) = n^{1+\sin n}$ $g(n) = \sqrt{n}$





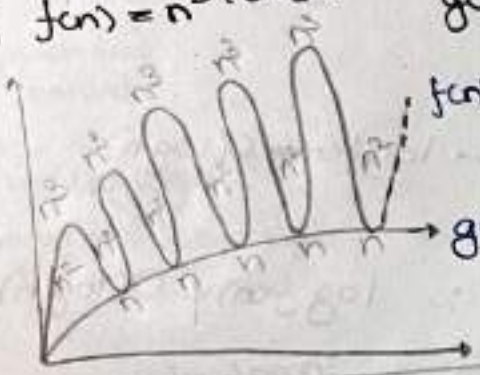
$f(n) \neq g(n)$ Asy not comp.

$f(n) \neq O(g(n))$ 44

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

Ques] $f(n) = n^2 + \sin n$ $g(n) = n$.

range $-1, 0, 1$



$f(n) \geq g(n)$

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

$f(n)$ is Asy bigger or equal to $g(n)$.

Ques] $f(n) = n^{1+\cos(2\pi n)}$ $g(n) = n^2$ n: the integers.

$\cos(\pi i) = (-1)^i$ for $i = 0, 1, 2, 3, \ldots$

i=even then 1
i=odd then -1

$\cos(2\pi n) = (-1)^{2n} = +1$ for all $n = 0, 1, 2, \ldots$

$f(n) = n^{1+\cos(2\pi n)} \approx n^2$ } Asy equal

$g(n) = n^2$

$f(n) = O(g(n))$ 44 $g(n) = O(f(n))$

cost < log fun < poly fun < exp fun

∈ polynomial bounded funcn

∈ exponential bounded funcn

$f(n)=c$ $f(n)=(\log n)^k$ $f(n)=n^k$ $f(n)=2^n$

$f(n)=n^k$: k is const

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

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

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

$\log f(n)>\log n$

* f(n) is polynomial bounded

iff $\log(f(n)) \leq_{n\geq y} \log n$ ∴ $\log f(n)=O(\log n)$

* f(n) is exponential bounded

iff $\log(f(n)) >_{n\geq y} \log(n)$ ∴ $\log f(n) \neq O(\log n)$

If

$\log(f(n))=\log n$ then poly fun

or

$\log(f(n))<\log n$ then log fun

Qu: find given fun poly bouned or exp bounded?

(i) $f(n)=n!$

exponential funcn

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

$n\cdot\log n>\log n$

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

∴ $m!=m^m$

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

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

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

exponential funcn

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

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

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

$= \log\log n \cdot \log\log\log n < \log n$

polynomial funct^n

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

k: any no. of time

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

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

$= \log[\ n\log n\ ]$

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

polynomial bounded.

(v) $f(n) = (\log n)^{\log n}$ exponential bound

$\log f(n) = \log[(\log n)^{\log n}]$

$= \log n \cdot \log\log n$

$\log n \cdot \log\log n > \log n$

(vi) $f(n) = (\log\log n)^{\log n}$

exponential bounded.

(vii) $f(n) = (\log n)^{\log\log n}$

polynomial bounded.

$\log n! = n\log n$

$n!$, $(\log n)!$

exp. bounded

$(\log\log n)!$

polybounded

$\log n! < (\log n)!$?

poly any Exp

- Arrange them in the Increasing order.

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

$< n!$

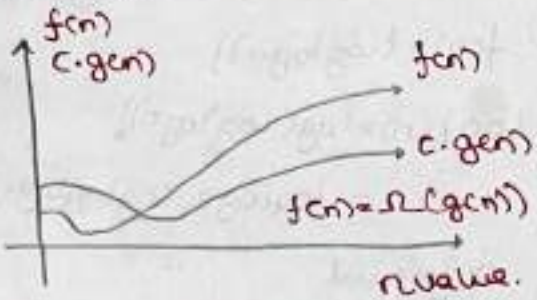
Omega Notation (Ω)

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

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

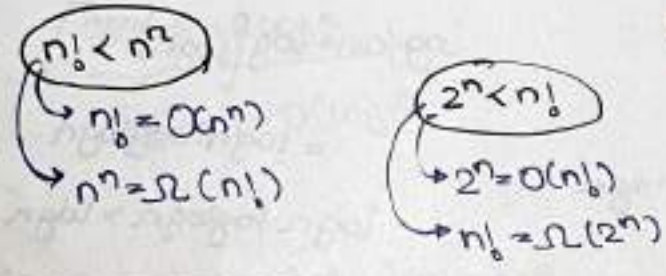
iff $f(n) \geq c \cdot g(n)$

for all n values, $n > n_0$



* $f(n) = \Omega(g(n))$
iff $g(n)$ Asy smaller or equal to $f(n)$

ex:



Theta Notation (Θ)

$f(n)$ $g(n)$ non negative funct^n

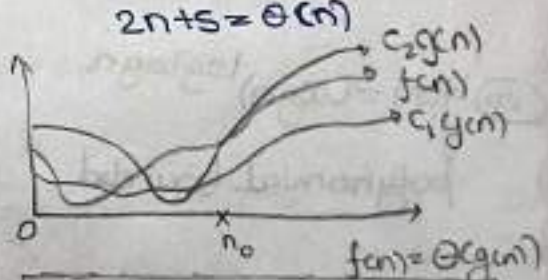
$f(n) = \Theta(g(n))$ iff

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n values where $n \geq n_0$

ex: $f(n) = 2n+5$ $g(n) = n$

$c_1 \cdot n \leq (2n+5) \leq c_2 \cdot n$

$2n+5 = \Theta(n)$



$f(n) = \Theta(g(n))$ iff $g(n)$ Asy equal to $f(n)$

Little o'h notation

$f(n) = o(g(n)) \longleftrightarrow f(n) < g(n)$

but not $f(n) \leq g(n)$

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

Little Omega notation

$f(n) = \omega(g(n)) \longleftrightarrow f(n) > g(n)$

but not $f(n) \geq g(n)$

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

Ques] which is true?

a) $f(n) + o(f(n)) = \Omega(f(n))$ True

$f(n) + o(f(n)) \geq f(n)$

$=$

$0, \Omega, \Theta$

$f(n)$ + any function $< f(n)$

bigger

$\{n^2 + o(n^2)\} = \{n^2\}$ Asy

ex: $n^2 + \Theta(n) = \Theta(n^2)$

$o(n^2)$ — any function $< n^2$

$n^2 + 2n + 3 = n^2$

$n + 0 = n$ — should be same here

$\omega(n^2)$ — $> n^2$

b) $f(n) + o(f(n)) = O(f(n))$ True

c) $f(n) + \omega(f(n)) = \Omega(f(n))$ True

$\{f(n) + > f(n)\}$

$\{n^2 + \omega(n^2)\} > \{n^2\}$ (Ω, ω)

d) $f(n) + \omega(f(n)) = O(f(n))$ False

e) $f(n) + O(f(n)) = \Theta(f(n))$ True

$\{f(n) + \text{any function} \leq f(n)\} = \{f(n)\}$

$\{\Theta, O, \Omega\}$

f) $f(n) + O(f(n)) = O(f(n))$ True

Asy. comparision

$F(n) = O(g(n)) \longleftrightarrow f(n) \leq g(n)$

$f(n) = o(g(n)) \longleftrightarrow f(n) < g(n)$

$F(n) = \Omega(g(n)) \longleftrightarrow f(n) \geq g(n)$

$F(n) = \omega(g(n)) \longleftrightarrow f(n) > g(n)$

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

** $F(n) = o(g(n)) \longrightarrow f(n) = O(g(n))$

** $F(n) = \omega(g(n)) \longrightarrow f(n) = \Omega(g(n))$

$F(n) = o(g(n)) \longleftrightarrow g(n) = \omega(f(n))$

$F(n) = O(g(n)) \longleftrightarrow g(n) = \Omega(f(n))$

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

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

$a \geq b$ & $b \leq a$

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

$a \leq b$ both are same

- $f(n)$ is $O(g(n))$ & $p(n)$ is $O(q(n))$

$f(n) + p(n)$ is $O(\max(g(n), q(n)))$

$f(n) * p(n)$ is $O(g(n) \cdot q(n))$

note

$O \rightarrow \leq$

$\Omega \rightarrow \geq$

$\Theta \rightarrow =$

$o \rightarrow <$

$\omega \rightarrow >$

Algorithm Analysis

- estimation of CPU execution time & main memory space required to complete execution of algorithm

Time Complexity

- estimation of CPU execution time to complete execution of algo

TC of Algo = { #of CPU computations required to complete ex. of algo [Run time] } = { sum of frequency count of each Instn of algo }

Space Complexity

estimation of MM space required to complete execution of algo

SC of algo =
1. Space for simple variable
2. Space for source code

(Const^n → ind-pendent of the I/P)

3. Space for DS used in Algo [Arrays, Linked lists]
4. Space for stacks which is used for recursive Algo

variable depend on I/O size

Linear Search Algo

a[1...n] array of n element. x Searching element

```
Algo Linear Search (a,n,x){
    for (i=1 ; i<=n ; i++){          → 2(n+1)
        if (x==a[i]) return(i);      → n
    }
    return(-1);                      → 1
}
```

(for: frequency count; n time)

i=1	i<=n	i++	TC 3n+3 = Θ(n)
1 Initialize	1<=n	1	
	2<=n	2	
	n+1<=n	n+1	
	n+1 comp	n Inc	
	2n+2		

frequent count Inner most stmt

TC of any algo

BC ≤ AC ≤ WC

✓ Worst Case TC of Algo: [max ex. time for n I/P]

if x is present last position or not in array

TC = 3n+3 = Θ(n)

[WC n element comp] (Θ, O, Ω)

- Best case TC of Algo
[min time to terminate algo for n i/p]

BC TC = constant = $\Theta(1)$

[Best case 1 comparison]

- Space compl" of LS:-

Simple Variable : 3 (i, n, x)

I/p array a[1...n] : n

{excluded I/p Space} $\Theta(1)$

SC of algo : $3+n = \Theta(n)$ {Including I/p}

- Aug TC of Algo :- $\Theta(n)$

$$\text{Aug Comp} = \frac{1+2+3+4+\cdots+n-1+n}{n}$$

$$= \frac{n(n+1)}{2*n} = \frac{n+1}{2} \text{ Comp.}$$

Matrix Addition

$A_{n\times n} + B_{n\times n} = C_{n\times n}$

BC, WC, AC — All case are $\Theta(n^2)$

```
Algo Matrix_Addition(A[n][n], B[n][n])
{
  for(i=1; i<=n; i++)          ---> 2(n+1)
  {                                          n time
    for(j=1; j<=n; j++)        ---> 2(n+1)*n
    {                                        n time
      C[i][j] = A[i][j] + B[i][j];   ---> 2*n*n
    }                          n^2
  }
  return(C[i][j]);             ---> 1
}
```

T.C : $4n^2+4n+3$

Space Comp. of Algo

i, j, n → 3

C[n][n], → $3n^2$

A[n][n],

B[n][n]

$3n^2+3 = \Theta(n^2)$

Include I/o space

excluding I/o space = $\Theta(n)$

TC of loops

I Loops var Inc/des interms of Add/Sub

1) `for(i=1; i<=n; i++)`

`x=x+1` → n times

TC = $\Theta(n)$

2) `for(i=n; i>=1; i--)`

`x=x+1` → n times

TC = $\Theta(n)$

3] for(i=2; i≤n; i=i+10)
x = x+1;
$\lfloor \frac{n-1}{10} \rfloor + 1$ times
optimes executed

i = 2, 12, 22, ... 2+K*10

K+1 times x=x+1 executed

4] for(i=n; i≥2; i=i-10) TC = Θ(n)
x = x+1;
i = n, n-10, ... n-K*10

$2+K*10 \le n$
$K \le \frac{n-2}{10}$
$K = \lfloor \frac{n-1}{10} \rfloor$
TC = Θ(n)

II] loop variable inc/dec in term of multiplication/division

5] for(i=2; i≤n; i=i*2)
x = x+1;
$\lfloor \log_2 n \rfloor + 1$ time

i = 2, 2.2, 2.2.2, ... 2^k

K+1 time loop executed

$2^k \le n$
$K \log 2 \le \lfloor \log n \rfloor \rightarrow K \le \log_2 n$
T.C = Θ(log n)

6] for(i=n; i≥2; i=i/2)
x = x+1;

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

$\frac{n}{2^k} \ge 1$
$K = \log_2 n$
T.C = Θ(log n)

6] for(i=2; i≤n; i=i*a)
x = x+1
TC = Θ(log_a n)

III] loop variable inc/dec in terms of $\sqrt[c]{i}$ / i^c

7] for(i=10; i≤n; i=i^2)
x = x+1;
$i = 10^{2^0}, 10^{2^1}, 10^{2^2}, \ldots 10^{2^k}$
K+1 time loop executed
$\lfloor \log_2 \log_{10} n \rfloor + 1$

$10^{(2^k)} \le n$
$K \le \log_2 \log_{10} n$
$K = \lfloor \log_2 \log_{10} n \rfloor$
$TC = \Theta(\log_2 \log_{10} n)$

8] for(i=n; i≥10; i=√i)
x = x+1;
$\lfloor \log_2 \log_2 n \rfloor + 1$ times
$i = n, n^{1/2}, n^{1/2^2}, n^{1/2^3}, \ldots n^{1/2^k}$
K+1 times loops executed

$n^{1/2^k} \ge 10 \Rightarrow \frac{1}{2^k} \log_{10} n \ge 1$
$\Rightarrow 2^k \le \log_{10} n \Rightarrow K = \lfloor \log_2 \log_{10} n \rfloor$
$T.C = \Theta(\log_2 \log_{10} n)$

9] for(i=5 ; i≤n ; i=i^10)

x = x+1 ; ⟶ $\lfloor \log_{10}\log_5 n \rfloor + 1$

T.C = $\theta(\log_{10}\log_5 n)$ time

10] for(i=n ; i≥5 ; i = $\sqrt[10]{i}$)

x = x+1

NOTE

if for(i=a ; i≤n ; i=i^b) then $\theta(\log_b \log_a n)$

	# estimated of time exec^n	TC of loop
(I) for(i=1 ; i≤n ; i=i+c) or for(i=n ; i≥1 ; i=i-c)	$= \lfloor \frac{n-i}{c} \rfloor + 1 \approx \frac{n}{c}$ times	$TC = \theta(n)$ (c: const.)
(II) for(i=1 ; i≤n ; i=i*c) or for(i=n ; i≥1 ; i=i/c)	$\lfloor \log_c n \rfloor + 1 \approx \log_c n$	$TC = \theta(\log_c n)$
(III) for(i=a ; i≤n ; i=i^c) or for(i=n ; i≥a ; i=$\sqrt[c]{i}$)	$\lfloor \log_c \log_a n \rfloor + 1$ $\approx \log_c \log_a n$	$TC = \theta(\log_c \log_a n)$

11] for(i=10 ; i ≤ 2^n ; i=i^2)

method 1:

$TC = \log_2 \log_{10} 2^n \approx \log_2 [n \cdot \log_{10} 2]$

$\approx \log_2 n + \log\log_{10} 2$

$\approx \theta(\log_2 n)$

method 2:-

$i = 10, 10^2, 10^3, \ldots 10^{2^k} \leq 2^n$

$10^{2^k} \leq 2^n$

$2^k \leq \log_2 \log_{10} 2^n$

$k \approx \log_2 n + \log\log_{10} 2$

$TC = \theta(\log n)$

12] for(i=n ; i≥1 ; i=i-n/2)

$TC = \frac{n}{n/2}$ times = 2 times = $\theta(1)$ (const)

$i = n, n-\frac{n}{2}, n-\frac{n}{2}-\frac{n}{2} \geq 1$

2 time, false

13] for(i=1 ; i≤n ; i=i+n/4)

$TC = \frac{n}{n/4}$ = 4 time = $\theta(1)$

$i = 1, 1+\frac{n}{4}, 1+\frac{n}{2}, 1+\frac{3n}{2}, \underbrace{1+n \leq n}_{\text{false}}$

4 time

15) for(i=n^10; i>1; i=i/6)

$TC = \log_6 n^{10} \approx 10 \cdot \log_6 n$

$\approx \Theta(\log n)$

16) for(i=10; i≤10^n; i=i*10) (almost)

$TC \approx \log_{10} 10^n \approx \Theta(n)$

17) for(i=100 (const); j≤n^3; i=i+10)

$TC \approx \frac{n^3}{10} \approx \Theta(n^3)$

18) for(i=10^n; i>5 (const); i=∛i)

$T.C = \log_6 \log_5 10^n$

$\approx \Theta(\log_6 n)$

19) for(i=n; i≤n^10; i=i+10)

$i = n, n+10, \ldots, n+k*10 \le n^{10}$

k+1 times

$\Rightarrow n + k*10 \le n^{10}$

$k = \frac{n^{10} - n}{10} \approx \Theta(n^{10})$

Nested Loops

- Independent nested loop:
  Inner loop value is independent of outer loop variable

T.C of nested loop = Θ(multiplication of frequency count of each loop)

```
for(i=n^10; i≥50; i=∛i) {x
  for(j=2^n; j≥1; j=j/10) {y
    for(k=10; k=n^20; k=k^3) {z
      x=x+1;
```

$\Theta(x * y * z)$

$\Theta[(\log \log n)^2 * n]$

$x = \log\log n^{10} \approx \Theta(\log\log n)$

$y = \log_{10} 2^n \approx \Theta(n)$

$z = \log\log n^{20} \approx \Theta(\log\log n)$

(21) for(i=1; i<=n; i=i+1)
  for(j=1; j<=n; j=j+1)
    for(k=1; k<=n; k=k+1)
      x=x+1;

$\Theta(n)$

(22) for(i=n²; i<=n²; i=sqrt(i)) $= \log_2 \log_2 n^2$
  for(j=2; j<=n²; j=j²) $= \log_2 n^2$
    for(k=n²; k>=1; k=k-n) $= n^2/n$
      x=x+1;

$\Theta[\log_2 \log_2 n * \log_2 n * n^2]$

(24) for(i=4; i<=n; i=i*4) $= \log_4 \log_4 n^2$
  for(j=n; j>=1; j=j/10) $= \log_{10} n^2$
    for(k=2; k<=n; k=k*9) $= \log_9 n^2$
      x=x+1;

$\Theta(\log\log n * (\log n^2))$

(26) for(i=2; i<=n; i=(i+10)) $= n$
  for(j=1; j<=n; j=j+j²) $= 2$
    for(k=n; k>=1; k=k-n/2) $= 2$
      x=x+1;

$\Theta(n)$

(23) for(i=1; i<=n; i=sqrt(i)) $= \sqrt{n}$
  for(j=n; j>=1; j=sqrt(j)) $= \log_2 \log_2 n$
    for(k=n²; k>=1; k=sqrt(k)) $\log_2 \log_2 n^2$
      x=x+1;

$\Theta(\sqrt{n} * (\log\log n^2))$

Mathematical Simplifications

(1) $\log_c a \cdot b = \log_c a + \log_c b$

(2) $\log_c a/b = \log_c a - \log_c b$

(3) $\log_c a^b = b * \log_c a$

(4) $\log_a a = 1$

(5) $\log_b a = \frac{\log_c a}{\log_c b}$

(6) $\log_b a = \frac{1}{\log_a b}$

(7) $a^{\log_c b} = b^{\log_c a}$

(8) $\log_a 1 = 0$

(9) $1+2+3+4+\cdots+n = \frac{n(n+1)}{2}$

(10) $1^2+2^2+3^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6}$

(11) $1^3+2^3+3^3+\cdots+n^3 = \frac{n^2(n+1)^2}{4}$

(12) $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n} = \log_e n$

(13) $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\cdots+\frac{1}{m} = \log_e \log_e m$

m is largest prime $\leq n$

(14) $n_{C_0}+n_{C_1}+n_{C_2}+\cdots+n_{C_n} = 2^n$

(15) $1+3+5+7+\cdots+(2n-1) = n^2$

(16) $1^2+3^2+5^2+\cdots+(2n-1)^2 = \frac{n(4n^2-1)}{3}$

Dependent nested loops

- Inner loop execution cost depending value of outer loop.

TC: expand loop & compute sum of frequency count of innermost Instⁿ

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

frequency count → TC

	y				
i →	1	2	3	----	n
j →	n times	n/2 times	n/3 times	----	n/n times

[x=x+1] → {n + n/2 + n/3 + ---- n/n}

frequency count = n{1 + 1/2 + 1/3 ---- 1/n}

= θ(nlogn) TC of program

Ques]
```
for(i=1; i<=n; i++)
    for(j=1; j<=i; j++)
        x=x+1;
```

i →	1	2	3 ----	n
j →	1 time	2 times	3 times ----	n times

(x=x+1) frequency count → [1+2+3 ---- n]

= n(n+1)/2 = θ(n²) TC of program

Ques]
```
x=0
for(i=1; i<=n; i++)
    for(j=1; j<=i; j++)
        for(k=1; k<=j; k++)
            x=x+1
```

(i) TC of program?

i=1	2	3	----	n
j=1	2/2	2/2/3	----	1/2/3 ... n
k=1	1/2	1/2/3	----	1/2/3 n times
	2 times	3 times		

(x=x+1) freq^n count → 2 + (1+2) + (1+2+3) + ---- (1+2+3+ n) = $S_n = \sum_{i=1}^{n} \frac{i(i+1)}{2}$

$S_n = \sum_{i=2}^{n} \left(\frac{i^2+i}{2}\right) \rightarrow \frac{1}{2}\left\{\sum_{i=2}^{n} i^2 + \sum_{i=2}^{n} i\right\}$

$S_n \rightarrow \frac{1}{2}\left\{\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}\right\}$

$S_n = \frac{1}{2} \cdot \frac{n(n+1)}{2}\left\{\frac{2n+1}{3} + 1\right\} \rightarrow S_n = \frac{1}{2} \cdot \frac{n(n+1)}{2} \cdot \frac{2(n+2)}{3}$

$S_n = \frac{n(n+1)(n+2)}{6} \rightarrow \boxed{\Theta(n^3)}$

(ii) if x=0, n=10 initially what is final val of x?

then n=10, $\frac{10(10+1)(10+2)}{6} \rightarrow \frac{10 \times 11 \times 12}{6} = 11 \times 20 \rightarrow \boxed{220}$

Ques
```
x=0
for (i=1; i<=n; i++){
    x = x+i^2
}
Return (x);
```

(i) TC of program?

TC = $\Theta(n)$

(ii) what is return value of program?

$x = 0+1^2+2^2+3^2+\cdots\cdots n^2$

$\rightarrow \frac{n(n+1)(2n+1)}{6} = \boxed{\Theta(n^3)}$

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

TC of program

$\Theta(n^2 \times (\log n))$

j =	1	2	3		n
k =	n	n/2	n/3		n/n
	time	time	time		time

$n \log n$

$n(1 + \frac{1}{2} + \frac{1}{3} + \cdots 1)$

$\rightarrow n \log n$

Ques] main() {

```
p=0;
for(i=1;i<=n;i++){
    q=0;
    for(j=1;j<=n;j=j*2)
    q++;          -log2n
    for(k=1;k<=q;k=k*2)
    p++;          log2log2n
}
return(p)
```

(i) TC of program? nlogn

(ii) return value of n(loglogn) the program?

$n(\log_2 n + \log\log_2 n)$ $\Rightarrow n(\log n)$

i = 1	2	3	------	n
$q=\log_2 n$	$q=\log_2 n$	$q=\log_2 n$		$q=\log_2 n$
$k \Rightarrow \log\log n$ time	$\log\log n$ time	$\log\log n$ time		$\log\log n$ time

$p = \log\log n + \log\log n$ ---------- $\log\log n^2$

value of $p = \boxed{\Theta(n \cdot \log\log n)}$

Ques] main(){

```
P=0; q=0;
for(i=1;i<=n;i++){
    for(j=1;j<=n;j=j*2)
    q++;
    for(k=1;k<=q;k=k*2)
    p++
}
return(p);
}
```

i = 1	2	3	--- n
$q=\log_2 n$	$q=\log_2 n \times 2$	$q=\log_2 n \times 3$	$q = n\log n$
$k=\log\log n$ time	$\log(2\log n)$	$\log(3\log n)$	$\log(n\log n)$

$p = \log\log n + \log\log n + \log\log n + \cdots + \log\log n$
$\log 2 + \log 3 + \cdots + \log n$

$\Rightarrow n \cdot \log\log n + \log(n!)$

$\Rightarrow n\log\log n + n\log n = \boxed{\Theta(n\log n)}$

i = 1 2 3 ---- n times

T.C → logn + logn + logn ---------- + logn
nlogn + log(2ⁿlogn) + ---- + log(nⁿlogn)

→ nlogn + nloglogn + logn! → $\Theta(n\log n)$

Ques] main()?
```
for(i=1;i≤n;i++){
    p=0; q=0;
    for(j=1;j≤n;j=j*2)
        q++;
        for(k=1;k≤q;k=k*2)
        p++;
    }
    return(p);
}
```

① TC of program?
$\Theta(n\log n)$

② return value of the program
$\Theta(n\log\log n)$

Ques] main()?
```
for(i=1;i≤n;i++){
    p=0;
    for(i=2;i≤n;i=i*2)
    q++;
    for(k=1;k≤q;k=k*2)
    p++;
}
return(p); // return(9) when
}
```

① TC of program?
$\Theta(n\log n)$

② return value of the program
$\Theta(\log n)$ | $\log_2 n$

(32) for(i=n, j=0; i>2; i=i/2, j=j+1)

① TC of loop: $\Theta(\log_2 n)$

② Value of j after termination of loop:-

i = n	n/2	n/2²	----	n/2^{log n}
j = 0	1	1+1		log n

$0 + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} \cdots \frac{n}{2^{\log n}}$

$n\left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} \cdots \frac{1}{2^{\log n}}\right)$

$\Theta(n)$ decreasing funct if GP series is less than 1 then it is const

$\Rightarrow n\left\{\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3}\right\}$

$\Rightarrow n\left\{\frac{\frac{1}{2}\left[1-\left(\frac{1}{2}\right)^k\right]}{1-1/2}\right\} = n\left\{1 - \frac{1}{2^k}\right\} < n\left\{1 - \frac{1}{n}\right\}$

$\Theta(n)$

→ $\Theta(n)$

(33) main()?
```
i=1
j=1
while(j≤n){
    i=i+1
    j=j+i
}
```
$\Theta(\sqrt{n})$

TC of loop

loop executed k times

i	1	2	3	4	k
j	1	1+2	1+2+3	1+2+3+4	1+2+3+...+k

$\sum_{i=1}^{k=n} \frac{k(k+1)}{2} \le n \Rightarrow k^2 \le n \Rightarrow k = \Theta\sqrt{n}$

(20) main(){

```
x=1
for(i=1; i<=n; i++)
    x=x+i^2
return(x) ⟶ θ(n^3)
}
```

i	1	2	3	----	n
x	1	1+2²	1+2²+3²	----	1+2²+3²...+n²

$$\sum_{i=1}^{n} \frac{i(i+1)}{2} = \Theta(n^3)$$

$1 \leq n \; ; \; n+2 \leq n$

(21) main(){

(i)

```
for(i=1; i<=n; i++) — 1 time
    for(i=1; i<=n; i++) — 1 time
        for(i=1; i<=n; i++) n times
            x=x+1
}
```

$\Theta(n)$

i
1
2
...
n

(ii)

```
for(i=1; i<=n; i++) — 1
    for(i=1; i<=n; i++) — 1
        for(i=1; i<=n; i=i*2)
            x=x+1
```

$\Theta(\log n)$

i
1
2
2^2
2^3
...
$2^k \leq n$

$2^{k+1} > n$

(iii) main(){

```
int i=n, flag=0;
while (i>2){
    if(flag==0){
        i=i-1
        flag=1;
    }
    else{
        i=i/2;
        flag=0;
    }
}
}
```

n

$n-1$

$\frac{n-1}{2} \approx \frac{n}{2}$

$\frac{n}{2}-1 \approx \frac{n}{2}$

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

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

$\frac{n}{2^3}$

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

$\approx 2 \cdot \log_2 n$

$\to k = \log_2 n$

$\Theta(\log_2 n)$

(iv) main(){

```
int i=2, flag=0;
while(i<=n)
    if(flag==0){
        i=i*2
        flag=1
    }
    else{
        i=i^2    → Loop is affected by these
        flag=0
    }
}
}
```

$\Theta(\log_2 \log_2 n)$

note

```
i=n
while(i>2){
 1. i=i-1;
 2. i=i-10;
 3. i=i-5;
 4. i=i/5;
 5. i=∛i;
}
```

TC: $\Theta(\log\log n)$

ex: Time complexity

```
for(k=1; k<n; k=k+1){ → n time
  for(i=1; i<n; i=i*3){ → log3n time
    j=i
    while(j>1){ } will run log i time
      sum=sum+1;
      j/=3;
    }
  }
```

for i=1 → log 1 time
i=3 → log 3 time
i=3² → log 3² time
⋮
i=3^k → log 3^k time (& k=log n)
assume to be n

So, inner for loop will run.

So T(n) = n(log n)²

log 1 + log 3 + log 3² + ... + log 3^k
= log(1·3·3²·.....·3^k) ⇒ log(3^{1+2+3...+k})
⇒ log 3^{k²} ⇒ k² = (log n)²

ex i=1;
k=1;
while(k<n){
  k=k+i;
  i=i+1;
}

0 1 2 3 4 5 ..... n → iteration : m
1 2 3 4 5 .....
1 1+1 1+1+2 1+1+2+3 1+1+2+3+4 ⋮
i
k

So, $1+\sum_{i=1}^{m} i$

So, $k = 1 + \frac{m(m+1)}{2}$

m to be total no. of iteration

$1+m(m-1)/2 < n \leq 1+\frac{m(m+1)}{2}$

So. n is $O(m^2)$ then m is $O(\sqrt{n})$

ex: for(int i=1; i<=n; i++) → if i++ O(n log n), i=i*2 O(n)
  for(int j=i; j<=n; j=j+i*2); → runs $\frac{n-i}{2i}$ time
}

For i=1, inner loop runs n/2 times
i=2, n/2×2
i=3, n/2×3
⋮
i=n n/2×n times

So, $\frac{n}{2} + \frac{n}{2\times2} + \frac{n}{2\times3} + \cdots + \frac{n}{2\times n}$

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

ex: For(i=1; i≤n; i*=2){ → 1 2 2^2 ... 2^k times ($2^k=n \Rightarrow k=\log n$)
For(j=1; j≤i; j++){ → 1 2 2^2 times
}
}

So, $1+2+2^2+\dots+2^k \Rightarrow \frac{2^{k+1}-1}{2-1} \Rightarrow 2^{\log n+1}-1$
$\Rightarrow n^{\log_2 2}-1$
$\Rightarrow 2n-1$

So, $\boxed{\Theta(n)}$

ex: int n;
int sum;
For(int i=1; i<n; i++){
For(int j=0; j<i*i; j++){ ——→ run 'n' time
If(j%i==0){ ——→ runs $\Theta(n) * \Theta(n^2) = \Theta(n^3)$ times
For(int k=0; k<j; k++){ → runs $\Theta(n) * \Theta(n) = \Theta(n^2)$ times
sum++; ——→ runs $\Theta(n^2) * \Theta(n^2) = \Theta(n^4)$ times
}
}
}
}

So, $\boxed{T(n)=\Theta(n^4)}$

ex: For(i=1; i≤n; i=i*2){
For(j=1; j≤i^2; j=j*2) → $\log_2 i^2$
sum++;
}

$\log 1^2 + \log 2^2 + \log 2^4 + \dots \log 2^{2k}$

$\Rightarrow 2\log 2 + 4\log 2 + 6\log 2 + \dots 2k\log 2$
$\Rightarrow 2+4+6+\dots+2k$
$2(1+2+3+\dots+k) = O(k^2)$
$= O(\log n)^2$

i=1 ---→ $\log_2 1^2$
i=2 ---→ $\log_2 2^2$
$i=2^2$ ---→ $\log (2^2)^2$
$i=2^3$ ——→ $\log (2^3)^2$
⋮ ⋮
$i=2^k$ $\log (2^k)^2$ ——→ 2^{2k}
$2^k=N$
$\boxed{k=\log_2 N}$

ex: For(j=n; j>0; j/=2){
For(k=j; k<n; k+=2);
}

j	Inner Iteration
2^k	1
2^{k-1}	$(2^k-2^{k-1})1/2$
2^{k-2}	$(2^k-2^{k-2})1/2$
⋮	⋮
2^0	$(2^k-2^0)1/2$

$\Rightarrow 1+k\cdot 2^{k-1}-(1+2+\dots+2^{k-1})\frac{1}{2}$
$\Rightarrow 1+k\cdot 2^{k-1}-(2^k-1)\frac{1}{2}$
$\Rightarrow 1.5+(k-1)\cdot 2^{k-1}$
$=(\log_2 n-1)n/2$

$\boxed{O(n\log n)}$

SC of Recursive Algo

- to execute Recursive Algo - or Subroutine call

Required Stack Data structure to store local variable & return address.

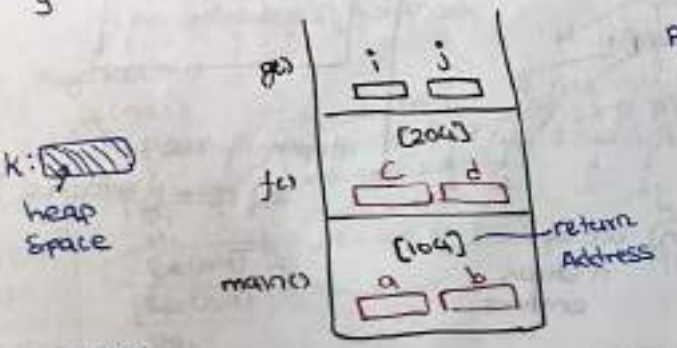
Rec Algo
1. SC
2. TC
3. # of rec calls
4. return value

Process:-
- Stack Space [static memory allocation]
- Heap space [dynamic memory allocation]
  - malloc, calloc

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

```
f(){
   int c,d
201 St1;
202 St2;   True
203 if(cond)g();
204 St3
205 St4
}
```

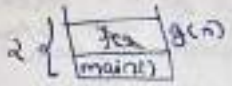
```
g(){
301 int i,j;
302 K=malloc( );
303 St1;
304 St2
   free(K)
}   EOP
```

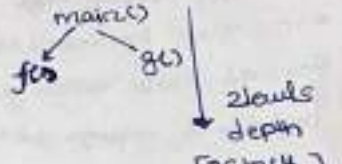


main()
↓
f()
↓
g()

3 level of fun call
[3 stacks entries]

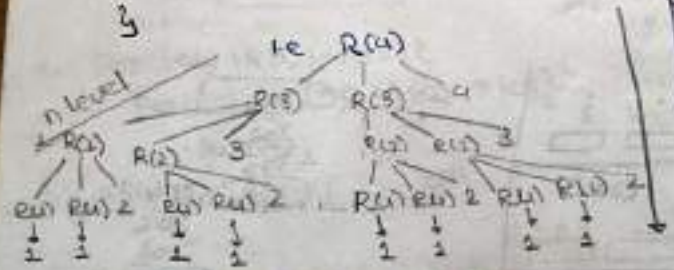
i.e main()?
int a, b;
f();
g();
}





How many [stack space / # entries required]? → it depend on depth of Recursive call
- not on the basis of no. of func^{n} call

i.e Algo rec(n)?
if (n ≤ 2)
Return(1);
else
return(rec(n-1)+rec(n-2)+n);
}



depth of rec : ## stack entries.

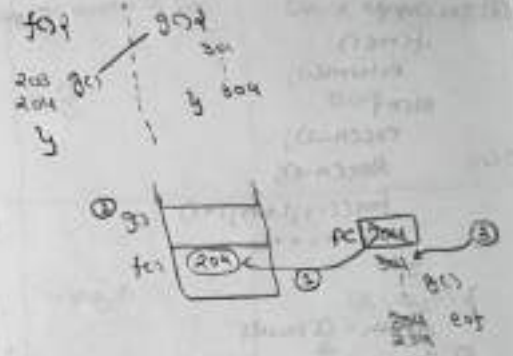
Space comp^{n} = Θ(n) b/c of n stack entries.

- Stack space required for Recursive Algo. is based on the depth of Recursion [no. of level of Recursion].

Space comp^{n} of Rec Algo	= Stack Space [order of depth of recursion]	+ Space required for other data structure used in algo

Overhead of fun call

1. Stack · top = [PC]
2. Stack · push();
3. PC ← new fun call address
   // execute fun
4. Stack · pop()
5. PC ← Stack · top · return addr.



- Recursive Algo are not efficient
  b/c 1. extra stack space required
  2. overhead of recursive call
  3. non-Recursive Algo [straight Algo] are more effic^{n} than Recursive Algo b/c no extra stack space & no overhead of funct^{n} call

note
Recursive Algo ⟷ non Recursive Algo
(more efficient, equivalent)
→ after the recursion funct^{n} no other statement when only you create an equivalent non-Recursive Algo else you can't create

ex: Algo Rec(n){
if(n≤1)
return
else{
St$_1$
St$_2$
Rec(n-1);
Rec(n-2)
St$_3$
St$_4$
}
}

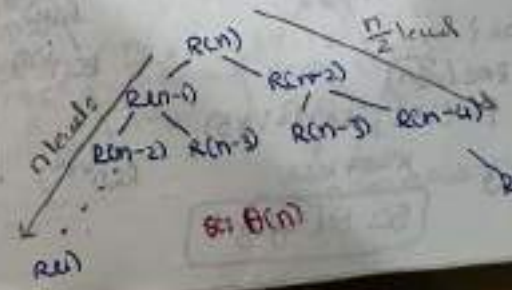
only these than are equivalent non Recursive Algo can be create with one Rec funct^{n}

You can't create non-Recursive Algo

i.e Algo rec(n){ ← no base condition
rec(n-1)
rec(n-1)
}
o/p: stack overflow

- Recursive Algo without base condition or termination condition throw stack overflow

i.e Algo Rec(n){
if(n≤1) Return(1);
O(n) else{
Rec(n-1);
Rec(n-2);
}
}



2 Rec(n) {
```
if(n<1)
  Return(1);
else{
  rec(n-2);
  Rec(n-2);
  for(i=1;i<=n;i+1)
    x=x+1;
}
}
```
$\Theta(n)$

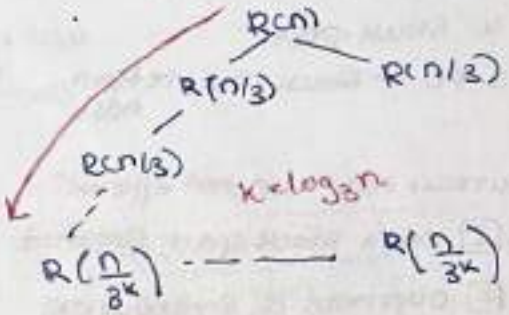
depth = $\frac{n}{2}$ levels of Rec

[$\frac{n}{2}$ stack entries]

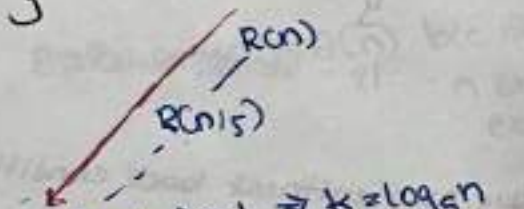
SC = $\Theta(n)$

3 Rec(n) {
```
if(n<=1)
  Return(1)
else{
  Rec(n/3);
  Rec(n/3);
}
}
```
$\Theta(\log n)$

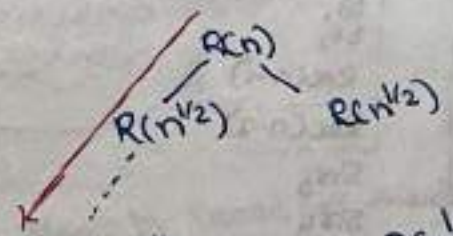
SC: $\Theta(\log n)$



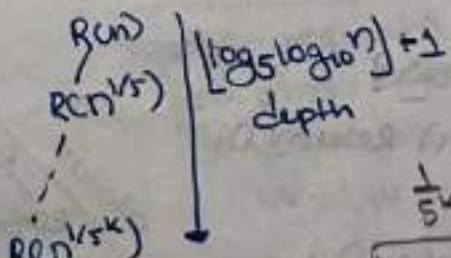
4 Rec(n) {
```
if(n<=1)
  Return(1);
else{
  Rec(n/5)
  for(i=1;i<=n;i=i*5)
}
}
```
SC: $\Theta(\log n)$



5 Rec(n) {
```
if(n<=2) Return(1);
else{
  Rec(√n);
  Rec(√n);
}
}
```
SC: $\Theta(\log\log n)$



6 Rec(n) {
```
if(n<=10) return;
else{
  Rec(∛n)
  for(i=1;i<=n;i=i*5)
    x=x+1
}
}
```
SC: $\Theta(\log\log n)$



$n^{1/5^k} = 10$

$\frac{1}{5^k}\log_{10} n = 1$

$k = \log_5 \log_{10} n$

Rec Algo	depth of Rec	Stack space
1. Rec(n){ … rec(n-c) } Rec(n){ … Rec(n-c); Rec(n-c); }	$\approx \frac{n}{c}$	$\theta(n)$
2. Rec(n){ … Rec(n/c); } Rec(n){ … Rec(n/c); Rec(n/c); }	$\approx \log_c n$	$\theta(\log_c n)$
3. Rec(n){ if(n≤a) Return; else{ rec(√n); } }	$\approx \log_c \log_a n$	$\theta(\log_c \log_a n)$

AGP series

AP/GP/AGP

- $a + (a+d) + (a+2d) + \cdots + a + (n-1)d = na + d\left[\frac{n(n-1)}{2}\right] = \theta(n^2)$

a, d const.

- $a + ar + ar^2 + \cdots + ar^{n-1} = \begin{cases} a\frac{[r^n - 1]}{r-1} & \text{if } r > 1 \rightarrow \theta(r^n) \\ a\frac{[1 - r^n]}{1-r} & \text{if } r < 1 \rightarrow \theta(1) \end{cases}$

$10 + \frac{1}{2^n}$ → Const (Decreasing func) then $\theta(1)$ if $10 > \frac{1}{2^n}$

▸ Shortcut for GP in Asymptotic analysis

a, r const.

$$1 + c + c^2 + \cdots + c^n = \sum_{i=1}^{n} c^i = \begin{cases} \theta(1), & \text{if } c < 1 \\ \theta(n), & \text{if } c = 1 \\ \theta(c^n), & \text{if } c > 1 \end{cases}$$

(first term) when decreasing

last term (when increasing)

note:- take the term depending on the increasing or decreasing

last term

take 1st term

① $T(n) = \sum_{i=0}^{n-1} 2^i * (n-i) \Rightarrow T(n) = 2^0 n + 2^1(n-1) + 2^2(n-2) + \cdots + 2^{n-3}(3)$ — (I)

n

leading term is 2^n

So, $\Theta(2^n)$

multiple it with common term

$T(n) = 2^0(n) + 2^1(n-1) + 2^2(n-2) + \cdots + 2^{n-2}(2) + 2^{n-1}(1)$ — (I)

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

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

(II) - (I)

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

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

$2^1(1)$ | $2^2(1)$

② $T(n) = \sum_{i=0}^{n-1} i * 2^i \Rightarrow T(n) = 2^0 * 0 + 2^1 * 1 + 2^2 * 2 + \cdots + 2^{n-2} * n-2 + 2^{n-1} * n-1$

$\Theta(2^n \times n)$

$T(n) = 2^0 * 0 + 2^1 * 1 + 2^2 * 2 + \cdots + 2^{n-2} * (n-2) + 2^{n-1} * (n-1)$ — (1)

$2T(n) = 2^2 * 1 + 2^3 * 2 + 2^4 * 3 + \cdots + 2^{n-1}(n-2) + 2^n(n-1)$ — (2)

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

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

$T(n) = 2^n(n-1) - \left[\frac{2(2^{n-1} - 1)}{2-1}\right] = \boxed{2^n(n-1) - 2^n + 2} = \boxed{\Theta(n \cdot 2^n)}$

③ $T(n) = \sum_{i=1}^{n} \frac{i}{3^i} \Rightarrow T(n) = \frac{1}{3} + \frac{2}{3^2} + \frac{3}{3^3} + \cdots + \frac{n}{3^n}$

leading term is const.

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

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

$$T(n) - \frac{T(n)}{3} = \frac{1}{3} + \underbrace{\left[\frac{1}{3^2} + \frac{1}{3^3} + \cdots + \frac{1}{3^{n-1}} + \frac{1}{3^n}\right]}_{\text{will be } \theta(1)} - \frac{n}{3^{n+1}}$$

So $T(n) - \frac{T(n)}{3} = -\frac{n}{3^{n+1}} \Rightarrow 2\frac{T(n)}{3} = \left(-\frac{1}{3} \times \frac{n}{3^n}\right) \Rightarrow \boxed{T(n) = \theta(n/3^n)}$

④ $T(n) = \sum_{i=1}^{n} \frac{n-i}{4^i}$

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

$$= \frac{n-1}{4} + \frac{n-2}{4^2} + \frac{n-3}{4^3} + \cdots + \frac{1}{4^{n-1}} + \frac{0}{4^n}$$

Subsitution method

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

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

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

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

$\vdots$

$T(n) = T(n-K)+\underbrace{1+\cdots+1+1}_{K \text{ time}}$

So, $T(n) = 1+K$

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

$= 1+n-1$

$n-K=1$ termination condition

$n-1=K$

So, $\boxed{\theta(n)}$

---

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

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

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

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

$\vdots$

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

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

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

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

$n-K=1$

$\boxed{K=n-1}$

---

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

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

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

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

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

$\vdots$

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

$n-K=1 \Rightarrow \boxed{K=n-1}$

replacing K by n-1

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

G.P series

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

$\Rightarrow 2^{n-1+1}-1 \Rightarrow 2^n-1$

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

④ gate 2004

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

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

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

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

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

$\vdots$

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

$n-k=1 \Rightarrow \boxed{k=n-1}$

$\Rightarrow 2^{n-1}(1)+2^{n-1-1}(n-(n-1-1))+\cdots+2(n-1)+n$

$\Rightarrow 2^{n-1}(1)+2^{n-2}(n-(n-2))+\cdots+2(n-1)+n$

$T(n) \Rightarrow 2^{n-1}(1)+2^{n-2}(2)+2^{n-3}(3)+\cdots+2(n-1)+n$

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

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

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

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

ex: $T(n) = \begin{cases} 2T(n-1)-1 & n>0 \\ 1 & n\le 0 \end{cases}$

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

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

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

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

$\vdots$

$T(n) = 2^kT(n-k)-2^{k-1}-\cdots-2^2-2-1$

$n-k=1 \Rightarrow \boxed{k=n-1}$

$T(n) = 2^k-(1+2+2^2+\cdots+2^{k-1})$

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

$\boxed{T(n)=\Theta(1)}$

ex: $T(n) = \begin{cases} T(n-1) \times n & n > 1 \\ 1 & n = 1 \end{cases}$

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

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

$T(n) = T(n-3) \times (n-2) \times (n-1) \times n$

$\vdots$

$T(n) = T(n-k) \times (n-(k-1)) \times \cdots \times (n-1) \times n$

$n - k = 1 \Rightarrow \boxed{k = n-1}$

Putting $k = n-1$

$T(n) = (n-(n-1-1)) \times \cdots \times (n-1) \times n$

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

$\Rightarrow 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$

$T(n) = n!$

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

gate 2009

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

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

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

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

$\vdots$

$T(n) = T\left(\frac{n}{3^k}\right) + \frac{n}{3^{k-1}} + \cdots + \frac{n}{3} + n$

$\frac{n}{3^k} = 1 \Rightarrow n = 3^k \Rightarrow \boxed{\log_3 n = k}$

$T(n) \Rightarrow n\left[\underbrace{1 + \frac{1}{3} + \frac{1}{3^2} + \cdots + \frac{1}{3^{k-1}}}_{\Theta(1)}\right]$

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

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

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

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

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

$\vdots$

$T(n) = T(n-2k) + \cdots + (n-2) + n$

$n - 2k = 1 \Rightarrow \boxed{k = \frac{n-1}{2}}$

$T(n) = n + (n-2) + \cdots + (n-2(k-0))$

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

rather than solving split the sequence

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

$\Rightarrow n + (n-2) + \cdots + (n-2n+2+2)$

$\Rightarrow n + (n-2) + \cdots + (-n+4)$

$\underbrace{n + n + n \cdots + n}_{k/2 \text{ time}} - (0 + 2 + 4 + \cdots + 2(k-1))$

$\Rightarrow \frac{kn}{2} - 2(0 + 1 + 2 + 3 + \cdots + (k-1))$

$\Rightarrow \frac{kn}{2} - 2(\underbrace{1 + 2 + 3 + \cdots + (k-1)}_{\frac{(k-1)(k-1+1)}{2} \Rightarrow \frac{k(k-1)}{2}})$

$T(n) = \frac{kn}{2} - \cancel{2}\frac{(k)(k-1)}{\cancel{2}} \Rightarrow \frac{(n-1)n}{2} - \frac{(n-1)}{2}\left[\frac{(n-1)+1}{2}\right]$

$\Rightarrow \frac{n^2-n}{4} - \frac{(n-1)(n-3)}{6}$

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

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

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

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

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

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

$\vdots$

$T(n) = 2^k T(n-2k)+(n-2(k-1))+\cdots+(n-2)+n$

$n-2k=1 \Rightarrow \boxed{k=\frac{n-1}{2}}$

$T(n) = n+(n-2)+\cdots+(n-2(k-1))+2^k$

$T(n) \Rightarrow 2^k + \underbrace{n+n+n\cdots+n}_{k/2 \text{ times}} - (2+4+\cdots+2(k-1))$

$\Rightarrow 2^k - \frac{kn}{2} - 2\underbrace{(1+2+3+\cdots(k-1))}_{\frac{k(k-1)}{2}}$

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

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

$\Rightarrow \frac{2^{n/2}}{\sqrt{2}} - \frac{n^2-n}{4} - \frac{(n-1)(n-3)}{6}$

$\boxed{T(n) = O(2^{n/2})}$ (missed few step)

wrong

ex: $T(n) = \begin{cases} 2T(n-3)+1 & n>1 \\ 1 & n\in 1 \end{cases}$

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

$T(n) = 2(2T(n-6)+1)+1$

$= 2^2 T(n-6)+2+1$

$T(n) \Rightarrow 2^3 T(n-3*3)+2^2+2+1$

$\vdots$

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

$n-3k=1 \Rightarrow \boxed{k=\frac{n-1}{3}}$

$T(n) = \underbrace{2^k+2^{k-1}+\cdots+2+1}$

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

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

putting $k=\frac{n-1}{3}$

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

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

$\boxed{T(n) = O(2^{n/3})}$

ex. $T(n) = \begin{cases} 3T(\sqrt[3]{n}) + 1 & n > 1 \\ 1 & n \le 1 \end{cases}$

Similarly to gate 2024

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

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

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

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

$\vdots$

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

$n^{1/3^k} = 3 \Rightarrow \frac{1}{3^k} \log_3 n = 1 \Rightarrow \log_3 n = 3^k$

$k = \log_3 \log_3 n$

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

$\Rightarrow 1\left(\frac{3^{k+1} - 1}{3 - 1}\right) \Rightarrow \frac{3^{k+1} - 1}{2}$

$T(n) = \frac{3^{\log_3 \log_3 n + 1} - 1}{2} \Rightarrow \frac{3^{\log_3 \log_3 n} \cdot 3^1 - 1}{2}$

$T(n) = \frac{\log_3 n^{\log_3 3} \cdot 3^1 - 1}{2} \Rightarrow \frac{3 \log_3 n - 1}{2}$

$T(n) = \log_3 n$

gate 2024

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

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

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

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

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

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

$\Rightarrow \frac{3}{n^4} T(n^{1/2^2}) + n + n$

$T(n) = \frac{3}{n^4}\left[n^{1/8} T(n^{1/2^3}) + n^{1/2^2}\right] + n + n$

$T(n) = \frac{7}{n^8}\left[T(n^{1/2^3})\right] + n + n + n$

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

$\vdots$

$T(n) = \frac{2^k - 1}{n^{2^k}}\left[T(n^{1/2^k})\right] + \underbrace{n + n \dots n}_{k \text{ times}}$

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

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

$k = \log_2 \log_2 n$

$T(n) = \frac{2^k - 1}{n^{2^k}} + kn$

$\Rightarrow \frac{n}{n^{1/2^k}} + kn \Rightarrow \frac{n}{n^{\frac{1}{2^{\log_2 \log_2 n}}}} + n \log_2 \log_2 n$

$\Rightarrow \frac{n}{n^{1/\log_2 n}} + n \log \log n$

$T(n) = n \log_2 \log_2 n$

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

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

Sol: $T(n) = T(n-2)+2\log n$

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

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

$\vdots$

$T(n) = T(\underbrace{n-2k}_{1}) + 2\log(n-2(k-1)) + \cdots + 2\log n$

$n-2k = 1 \Rightarrow \boxed{k = \frac{n-1}{2}}$

$T(n) = 2\log(n-2(k-1)) + \cdots + 2\log(n-2) + 2\log n$

$= 2\log(n*(n-2)*(n-4)*\cdots\cdots*(n-2(k-1)))$

putting $k = n-1/2$

$T(n) = 2\log(n*(n-2)(n-4)\cdots\cdots 4*2)$

$\Rightarrow 2\log(2^{n/2} * (\frac{n}{2})!)$

$\Rightarrow 2\log_2 2^{n/2} + 2\log \frac{n}{2}^{n/2}$

$\Rightarrow 2*\frac{n}{2} + 2*\frac{n}{2}\log\frac{n}{2}$

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

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

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

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

$\vdots$

$T(n) = T(n-2k)+(n-2(k-1))^2+\cdots+(n-2)^2+n^2$

$n-2k = 1$ Putting k value then

$\boxed{k = \frac{n-1}{2}}$

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

$= \frac{2n(n+1)(2n+1)}{3}$

$\boxed{T(n) = O(n^3)}$

doubt

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

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

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

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

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

$\vdots$

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

$n-2K=1 \Rightarrow \boxed{K=\frac{n-1}{2}}$

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

$\Rightarrow n(1+2+3+\cdots+2^{K-1})$

$2T(n) = +2n + 2^2(n-2) + \cdots + 2^{K-1}(n-2(K-2)) + 2^K(n-2(K-1)) + 2^{K+1}$ some thing wrong

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

$T(n) = 2^2+2^3+\cdots+2^{K-1}-n+K2^{K+1}$

$= 2^2(1+2+\cdots+2^{K-3})-n+K2^{K+1}$

$\Rightarrow 2^2(1)\left(\frac{2^{K-2}-1}{2-1}\right) - n + K2^{K+1}$

$\Rightarrow 2^2(2^{K-2}-1) - n + K2^{K+1}$

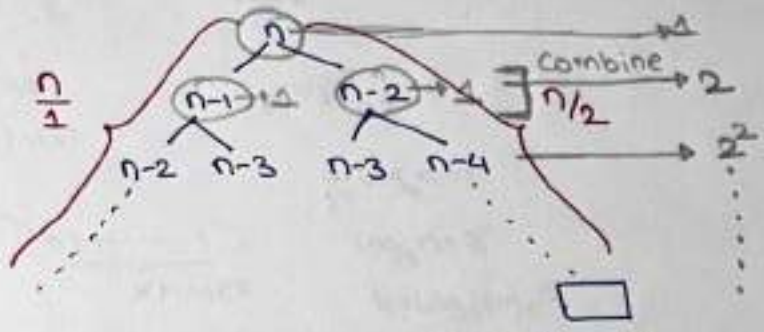
$T(n) \Rightarrow 2^K - 4 - n + K2^{K+1}$

putting $K=\frac{n-1}{2}$

$T(n) = \underbrace{2^{\frac{n-1}{2}}}_{\text{leading term } 2^{n/2}} - 4 - n + \underbrace{\frac{(n-1)}{2}2^{\frac{(n-1)}{2}+1}}_{\text{leading term } n2^{n/2}}$

$T(n) = O(n2^{n/2})$

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



- lower bound

$2^0 + 2^1 + \dots + 2^{n/2} \Rightarrow \Theta(2^{n/2})$

take last term as it's increasing G.P

upper bound

$2^0 + 2^1 + 2^2 + \dots + 2^n = \Theta(2^n)$

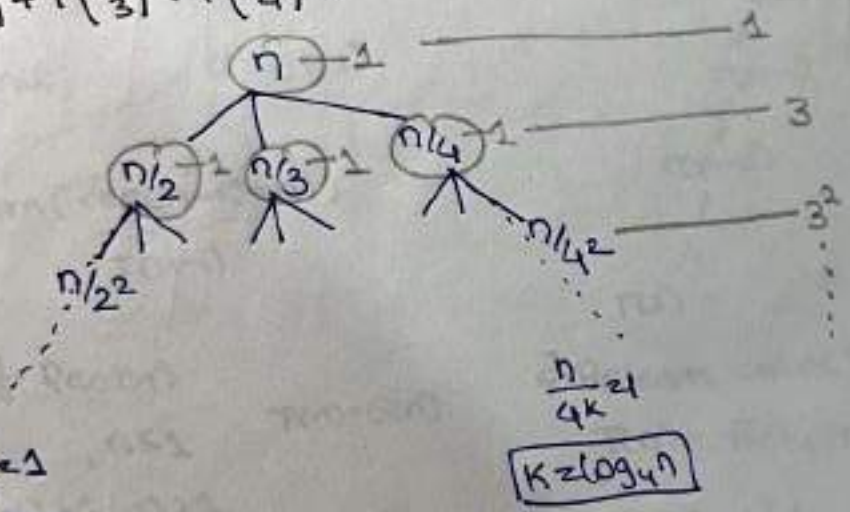
w/o solving

$T(n) = \Omega(2^{n/2})$

$T(n) = O(2^n)$

we can't combine (θ is not possible)

ex: $T(n) = T(\frac{n}{2}) + T(\frac{n}{3}) + T(\frac{n}{4}) + 1$



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

$k = \log_2 n$

$\frac{n}{4^k} = 1$

$k = \log_4 n$

lower bound

$\Rightarrow 1 + 3 + 3^2 + \dots + 3^k$

$\Rightarrow 3^k \Rightarrow 3^{\log_4 n}$

$\Rightarrow \Omega(n^{\log_4 3})$

upper bound

$\Rightarrow 1 + 3 + 3^2 + \dots + 3^k = \Theta(3^k)$

$\Rightarrow 3^{\log_2 n} \Rightarrow \Theta(n^{\log_2 3})$

$n^{\log_2 3}$ $n^{\log_4 3}$

$n^{\log_2 3}$ $n^{\frac{1}{2}\log_2 3}$

Divide by $n^{\frac{1}{2}\log_2 3}$

$\frac{n^{\log_2 3}}{n^{\frac{1}{2}\log_2 3}} > 1$

So, θ is not possible

(13) $T(n) = \begin{cases} T(\sqrt[3]{n}) + 1 & n > 3 \\ 3 & n \le 3 \end{cases}$

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

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

$\vdots$

$T(n^{(1/3)^k}) + \underbrace{1 + \cdots + 1}_{k \text{ times}}$

$n^{(1/3)^k} = 3$

$\frac{1}{3^k} \log_3 n = 1$

$\log_3 n = 3^k$

$k = \log_3 \log_3 n$

$T(n) = T(1) + k \cdot 1$

$= 1 + \log_3 \log_3 n$

$\rightarrow \Theta(\log_3 \log_3 n)$

T.C of Recursive Algo

1] Algo Rec(n){ → T(n)

```
if(n ≤ 1)          const(1)
    Return(1);
else                const(1)
    Return(rec(n-1)+n);
```

T(n-1)

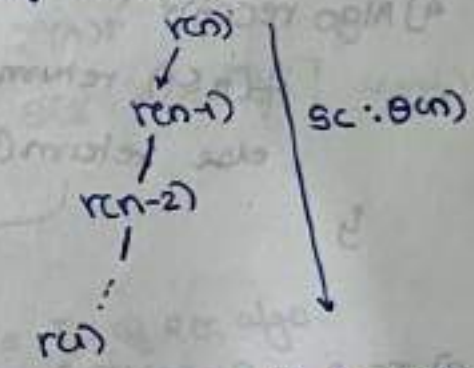
}

a] T(n): TC of Rec(n)

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

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

b] # recursive calls = $\Theta(n)$



c] Return value of Rec(n):-

T(n) → Return val of Rec(n)

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

$= \Theta(n^2)$

2] Algo Rec(n){ → T(n)

```
const  if(n≤2) return(1);
       else {
           rec(n-1); → T(n-1)
           for(i=1; i≤n; i++)  } n
               x=x+1;
       }
}
```

a] T(n): TC of Rec(n)

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

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

b] # of Recursive call = $\Theta(n)$

S.C: $\Theta(n)$ [n levels each level one call]

3] Algo rec(n)?

```
Const [if(n≤1) return(1);
       else return (2*rec(n-1)+n);
}
```

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

b] # of Recursive call = $\Theta(n)$

R(n) → R(n-1) → ... → R(1)

4] Algo rec(n)?

```
Const [if(n≤1) return 1;
       else return(rec(n-1)+rec(n-1)+n);
}
```

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

a] TC of Rec(n): T(n)

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

$$T(n) = 2^n - 1 = \Theta(2^n)$$

c] T(n): Return value of Rec(n)

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

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

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

a] TC of Rec(n)

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

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

c] Return value of Rec(n): T(n)

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

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

(b) # of rec. calls:-

[Complete BT of n levels]

R(n) — 2

R(n-1) R(n-1) — 2^2

R(n-2) R(n-2) R(n-2) R(n-2) — 2^3

R(1) + ... + R(1) — 2^{n-1}

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

5] Algo rec(n) { → T(n)

const [if(n≤1) Return(1);

else {

Rec(n-1); } 2T(n-1)

Rec(n-1);

for(i=1; i≤n; i++) } n

x = x+1

}

}

a] T(n): TC of Rec(n)

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

$2^{n+1} - n - 2 \Rightarrow \theta(2^n)$

b] # of rec calls

[complete BT of n level]

$1 + 2^2 + 2^3 \dots 2^{n-1} = \theta(2^n)$

- if extra cost of each recursion is constant other than cost of sub recursive call then TC of Recursive Algo is θ(recursive call)
- if extra time taken by each recursion is not constant [Asy^ greater than const.] then TC of Recursive Algo ≥ no. of recursive call

# of fun + loop

1 + n

2 + 2(n-1)

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

⋮

2^{n-1}

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

Q] Assume TC of gen() fun is θ(1/n) what is TC of Rec algo.

Rec(n) {

if(n≤1) Return(1); } const (1)

else {

Rec(n-1); → n

call gen(); → 1/n

}

}

due to these decrem^n func^n TC is ≤ then the no of recursive call

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

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

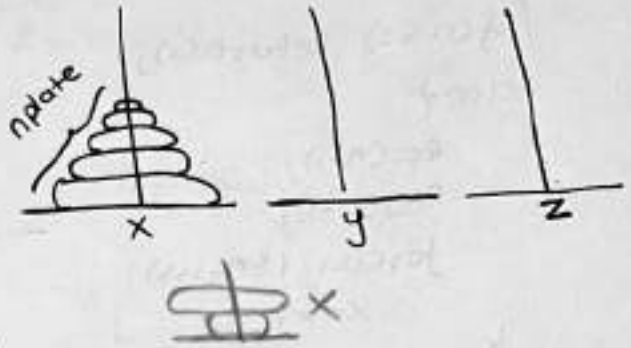
$T(n) = T(n-2) + \frac{1}{n-1} + \frac{1}{n}$

⋮

$= 1 + \frac{1}{2} + \frac{1}{3} \dots \frac{1}{n} = \theta(\log n)$

7] Tower of hanoi :-

- Given 3 tower [x, y, z] of n plates in tower x with decreasing order of diameter bottom to top

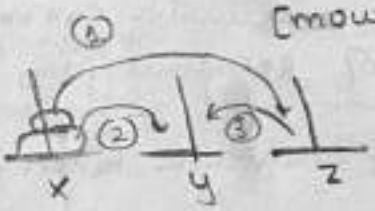


- How many min. plate move Required to transfer n plate from tower x to tower y.
  - i) only top plate is allowed to move at any time from any tower
  - ii) during the transfer lower diameter plate should not be below larger diameter plate

n=1

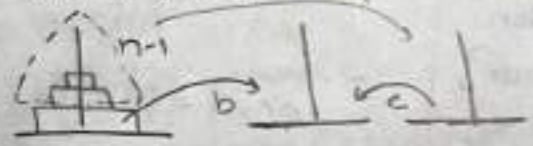
x → y [move 1]

n=2



{x → z, x → y, z → y;} moves 3

n=3



a] x → y, x → z, y → z
b] x → y
c] z → x, z → y, x → y

} 7 move

Procedure : To transfer n plates from x to y using z as intermidiate

(1) Transfer to (n-1) plates from x to z using y as intermidiate

(2) one plate move from x to y

(3) Trans (n-1) plates of z to y by using x as intermiate

TOH(Source, Dest^n, Intermn^, #of plates)

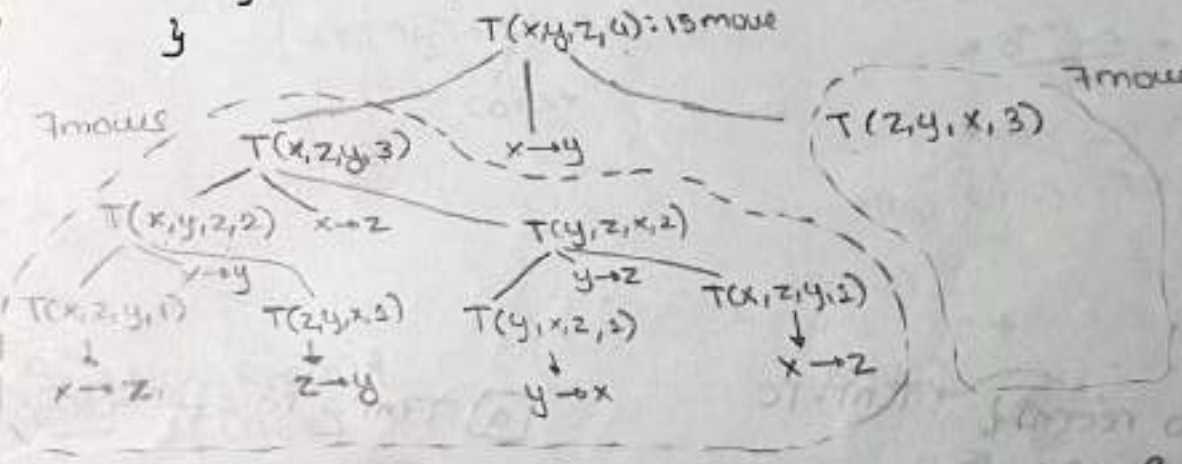
```
Algo TOH(x,y,z,n){
n-1 [2^n-1 moves]  if(n==1)
                       return(move x to y);
                   else{
                       1. TOH(x,z,y,n-1) ~ T(n-1)
                       2. move x to y  ——— 1 move
                       3. TOH (z,y,x,n-1) — T(n-1)
                   }
}
```

Recurrence Rel to find #f plate moves:

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

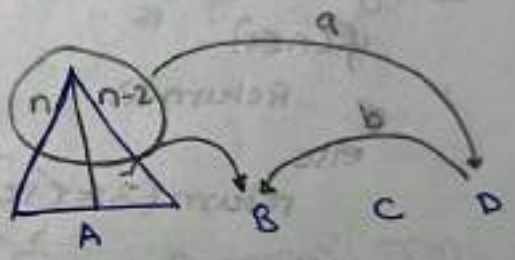
$T(n) = 2^n - 1$ moves

// exponential
// undecidable problem



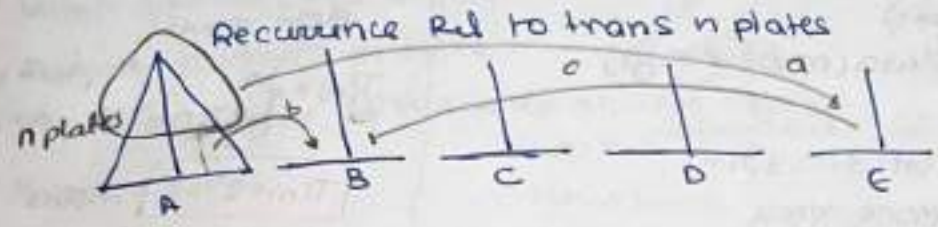
Ques] 4 tower [A B C D] 44 n plates
↑
Some

Recurrence Rel to transfer n plates from A to B using C,D is into.



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

Ques] 5 towers [A B C D E] if n plates
↑ Some



7.
Ques] Algo rec(n){ → T(n): TC

```
if(n<=1)
    Return(1)
else
    return(3*(rec(n/3)+n));
```

T(n/3)

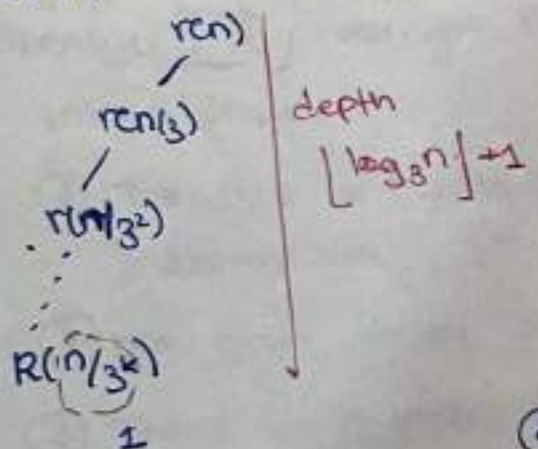
[a] T(n): Tc of rec(n)

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

Const.

$T(n) = \Theta(\log_3 n)$

[b] # rec calls: $\Theta(\log_3 n)$



[c] T(n): Return valu. of Rec(n)

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

$T(n) = \Theta(n \cdot \log_3 n)$

[d] Space comp^{n}
$\Theta(\log n)$

8.

Ques] Algo Rec(n) {

```
if(n≤1)
    Return(1);
else
    Return(rec(n/3)+rec(n/3)+rec(n/3)+n);
}
```

(a) T(n): TC of rec(n)

$$T(n) = \begin{cases} 1 & , n \le 1 \\ 3T(n/3) + \underbrace{1}_{\text{const}} & , n > 1 \end{cases}$$

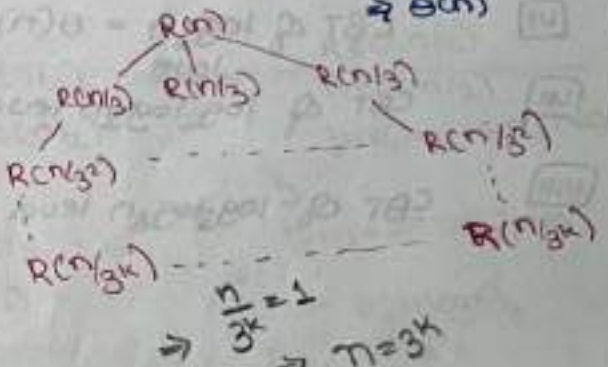
$T(n) = \Theta(n)$

(b) #f rec calls: complete Ternary Tree of $\Theta(\log_3 n)$ level

$= 1+3+3^2+\dots+3^k$

$\Rightarrow \frac{3^{k+1}-1}{2} = \frac{3n-1}{2}$

$\Rightarrow \Theta(n)$



9. Ques] Algo Rec(n) {

```
if(n≤1)
    Return(1)
else {
    1. Rec(n-1);
    2. Rec(n-1);
    for(i=1; i≤n; i++){
        x=x+1;
    }
}
}
```

(a) TC of rec(n): T(n)

$$T(n) = \begin{cases} 1 & ; n \le 1 \\ 3T(\frac{n}{3}) + n & ; n > 1 \end{cases}$$

$T(n) = \Theta(n \cdot \log_3 n)$

(b) #f rec calls

Complete ternary tree of $\Theta(\log_3 n)$ level.

$\Theta(n)$

Imp

Recursive Algo whose TC exponential

* we are assuming the remaining cost is constant

i) Rec(n){
:
R(n-1);
R(n-1);
}

TC $T(n) = 2T(n-1)+1$
$= \theta(2^n)$

ii) Rec(n){
:
Rec(n-1);
Rec(n-2);
}

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

iii) Rec(n){
:
Rec(n-2);
Rec(n-2);
}

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

*iv) Rec(n){
:
R(n-c);
R(n-c);
}

$T(n) = 2T(n-c)+1$
$\neq \theta(2^{n/c})$

v) rec(n){
:
Rec(n-1);
Rec(n-1);
Rec(n-1);
}

$T(n) = 3T(n-1)+1$
$= \theta(3^n)$

Qus) # Recursive call

complete BT of k level = $\theta(2^k)$

i) complete BT of n level = $\theta(2^n)$ $[T(n) = 2 \cdot T(n-1)+1]$
ii) complete BT of $n/2$ level = $\theta(2^{n/2})$ $[T(n) = 2T(n-2)+1]$
iii) complete BT of $n/3$ level = $\theta(2^{n/3})$ $[T(n) = 2T(n-3)+1]$
iv) complete BT of n/c level = $\theta(2^{n/c})$ $[T(n) = 2T(n-c)+1]$ } exp.

v) complete BT of $\log_2 n$ level = $\theta(n)$ $[T(n) = 2T(n/2)+1]$ } linear

vi) CBT of $\log_3 n$ level = $\theta(n)$ $[T(n) = 3T(n/3)+1]$

vii) CBT of $\log_2 \log_2 n$ level = $\theta(\log_2 n)$ $[T(n) = 2 \cdot T(\sqrt{n})+1]$ } logarithmic

viii) CBT of $\log_3 \log_3 n$ level = $\theta(\log_3 n)$ $[T(n) = 3T(\sqrt[3]{n})+1]$

vi) Rec(n)?

```
Rec(n-1);
Rec(n-1);
for(i=1;i<=n;i++)
}
```

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

$= \Theta(2^n)$

AGP

vii) Rec(n)?

```
Rec(n-2);
Rec(n-2);
Rec(n-2);
for(i=1;i<=n;i++)
}
```

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

$= \Theta(3^{n/2})$

# of rec calls complete Ternary tree of $n/2$ levels.

Recursive Algo TC are not exponential

$T(n)$: TC of Rec(n)

1] Rec(n)?

```
Rec(n/2)
Rec(n/2)
}
```

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

$= \Theta(n)$

2] Rec(n)?

```
Rec(n/2)
Rec(n/2)
for(i=1;i<=n;i++)
}
```

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

$= \Theta(n \log n)$

3] Rec(n)?

```
Rec(n/3);
Rec(n/3);
Rec(n/3);
}
```

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

$= \Theta(n)$

4] Rec(n)?

```
Rec(n/5);
for(i=1;i<=n;i++)
}
```

$T(n) = T(n/5)+n$

$= \Theta(n)$

5] Rec(n)?

```
Rec(√n)
Rec(√n)
}
```

$T(n) = 2T(\sqrt{n})+1$

$= \Theta(\log_2 n)$

6] Rec(n)?

```
Rec(∛n)
Rec(∛n)
Rec(∛n)
}
```

$T(n) = 3T(\sqrt[3]{n})+1$

$= \Theta(\log_3 n)$

7] Rec(n)?

```
Rec(n-1);
}
```

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

$= \Theta(n)$

8] Rec(n)?

```
Rec(n-1);
for(i=1;i<=n;i++)
}
```

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

$\Theta(n^2)$

Fast exponential Algo

for given 2 positive integer x & n compute x^n value

$$x^n = \begin{cases} x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} & \text{for } n \text{ is even} \\ x * x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} & \text{for } n \text{ is odd} \end{cases}$$

$y = x^{n/2}$

$$x^n = \begin{cases} y * y, & \text{if } n \text{ even} \\ x * y * y, & \text{if } n \text{ odd} \end{cases}$$

Ques] How many min multiplication required to compute x^{100}

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

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

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

$x^6 * x^6$

$x^3 * x^3$

$x * x * x$

min 8 multiplication

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

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

$x^8 * x^8$

$x^4 * x^4$

$x^2 * x^2$

$x * x$

$\lfloor \log_2 n \rfloor$ multiplication if n is power of 2

min 6 multiplication

$x^{127} = x * x^{63} * x^{63}$

$x * x^{31} * x^{31}$

$x * x^{15} * x^{15}$

$x * x^7 * x^7$

$x * x^3 * x^3$

$x * x * x$

min 12 multiplication

Fast Exponential Algo:

Algo FE(x,n) → T(n) : TC

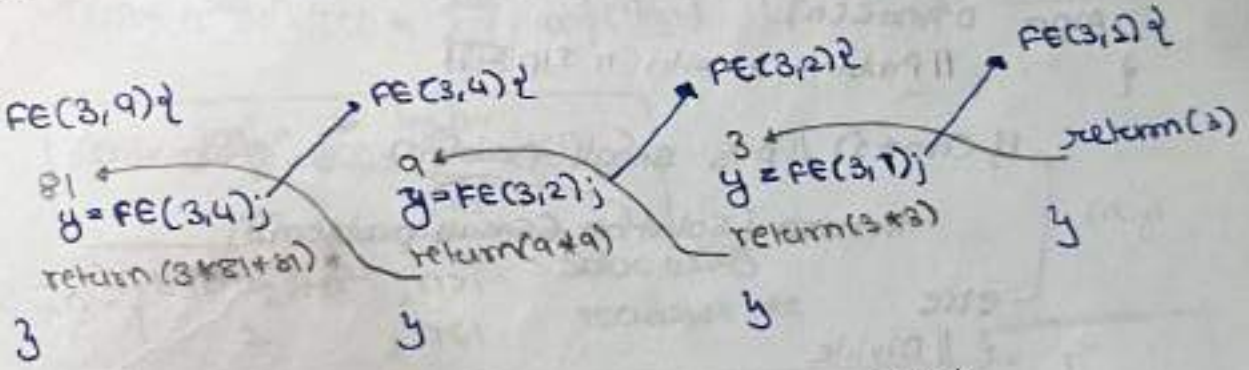
// x & n are two +ve integers

```
if(n==1)                                   Const (c)
    Return(x);
else{
    y=FE(x,n/2);   ----- T(n/2)
    if(n%2==0)
        return(y*y);        } const time
    else
        return(x*y*y);      }
}
}
```

T(n) = TC of FE(x,n)

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

$$\Theta(\log_2 n)$$



Assignment

- Solve given Recursions using substitution method

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

2. $T(n) = \begin{cases} 8T(n/2)+n^2 & n>1 \\ 1 & n\leq 1 \end{cases}$

3. $T(n) = \begin{cases} 4T(n/2)+n^2 & n>1 \\ 1 & n\leq 1 \end{cases}$

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

5. $T(n) = \begin{cases} 2T(n/2)+n\log_2 n & n>1 \\ 1 & n\leq 1 \end{cases}$

6. $T(n) = \begin{cases} 2T(n/2)+\frac{n}{\log_2 n} & n>1 \\ 1 & n\leq 1 \end{cases}$

7. $T(n) = \begin{cases} 2T(n/4)+\log_2 n & n>1 \\ 1 & n\leq 1 \end{cases}$

8. $T(n) = \begin{cases} 2T(\sqrt{n})+\log_2 n & n>2 \\ 1 & n\leq 2 \end{cases}$

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

Divide & Conquer

Small (problem): problem which are not allowed to possible divide further.

Algo design technique:

- problem "p" with n I/p if not small problem divide into "a" sub problem [$P_1 P_2 P_3 \cdots P_a$]. each Sub problem $\frac{n}{b}$ I/p [where a, b are constant]
- if subproblem not small problem then divide into sub-subproblem until each subproblem become small problem
- Solve the small problem, return solution of small problem & Conquer solution in Bottom up approach until we get solution of given problem

Control Abstraction of D And C:- (divide, conquer)

```
Algo DAndC(n)
// Problem p with n I/p size
{
    if (n==1) // p is small problem        g(n) = Θ(1)
        Return (Solution (Small problem));
    else
    { // Divide
        • Divide P into 'a' Sub problem
          P1 P2 P3 ... Pa with n/b i/p size each
        • Conquer (DAndC(n/b), DAndC(n/b), ---- DAndC(n/b));
    }
}
```

g(n) = one of the method; f(n) = T(n/b) + T(n/b) + ... + T(n/b)

T(n): TC of DAndC with n I/p

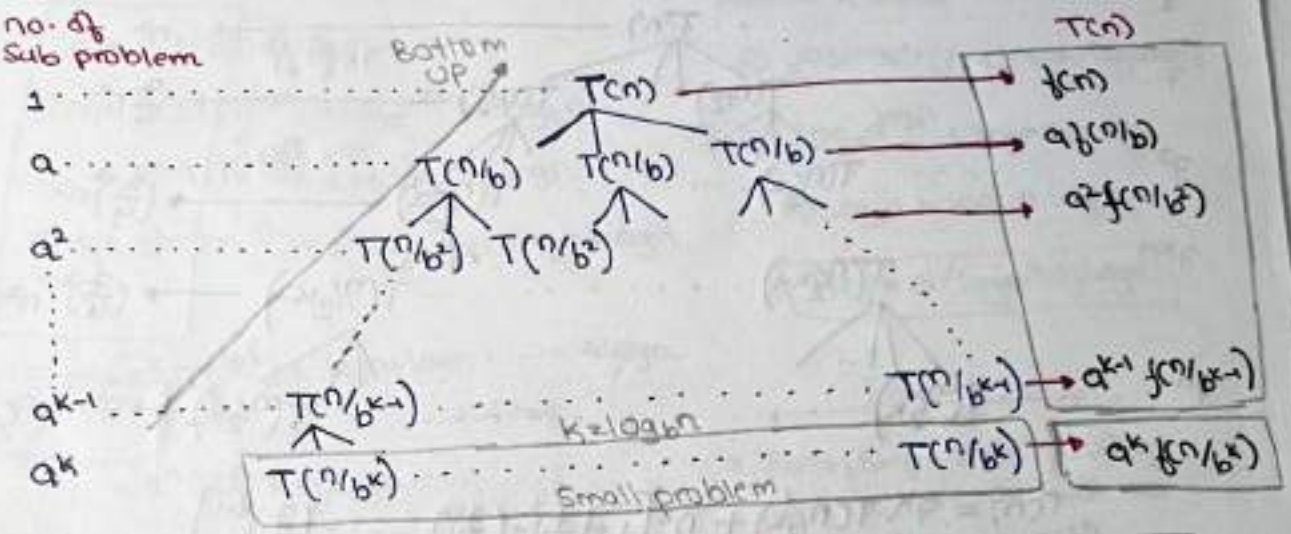
g(n): TC to solve one small problem

f(n): TC to divide & Conquer of n I/p

$$T(n) = \begin{cases} g(n) \approx \Theta(1) & , n = 1 \text{ // Small problem} \\ aT(n/b) + f(n) & , n > 1 \end{cases}$$

(g(n))

Recursive tree method $T(n) = aT(\frac{n}{b}) + f(n)$



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

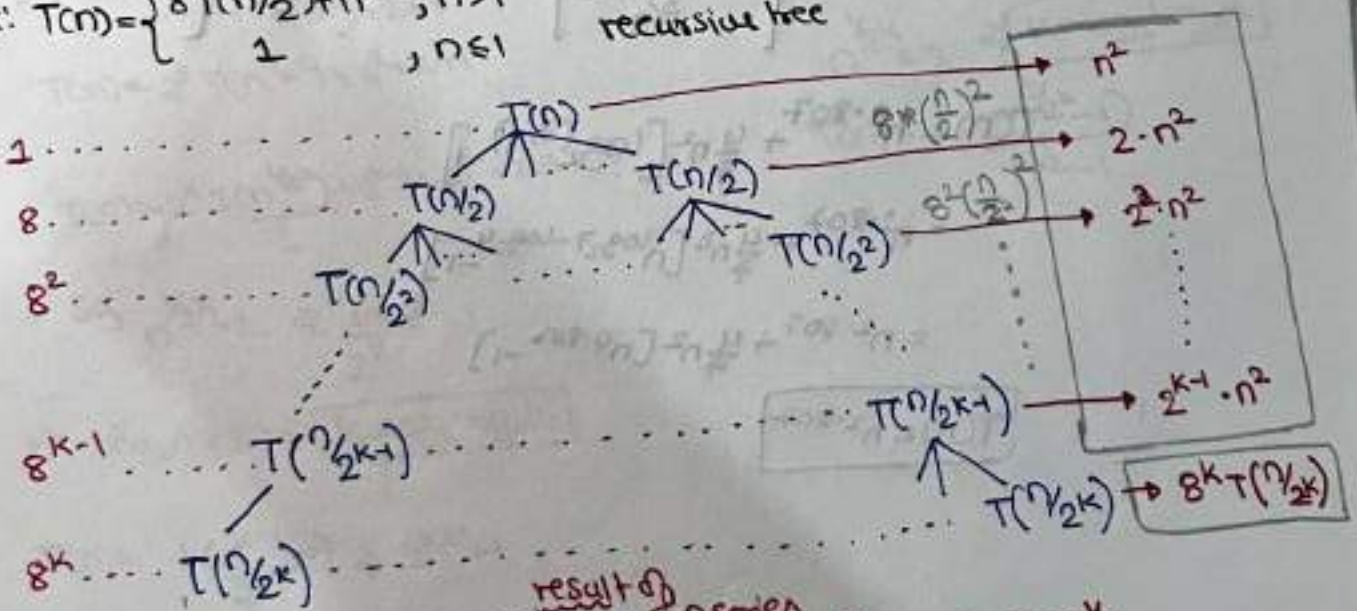
$n/b^k = 1$, $n = b^k$, $k = \log_b n$

Depth of Recursion: $\theta(\log_b n)$

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

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

ex: $T(n) = \begin{cases} 8T(n/2) + n^2 & , n > 1 \\ 1 & , n \le 1 \end{cases}$ Solve using recursive tree



$T(n) = 8^k \cdot T(n/2^k) + n^2 [2^k - 2]$ (result of G.P series)

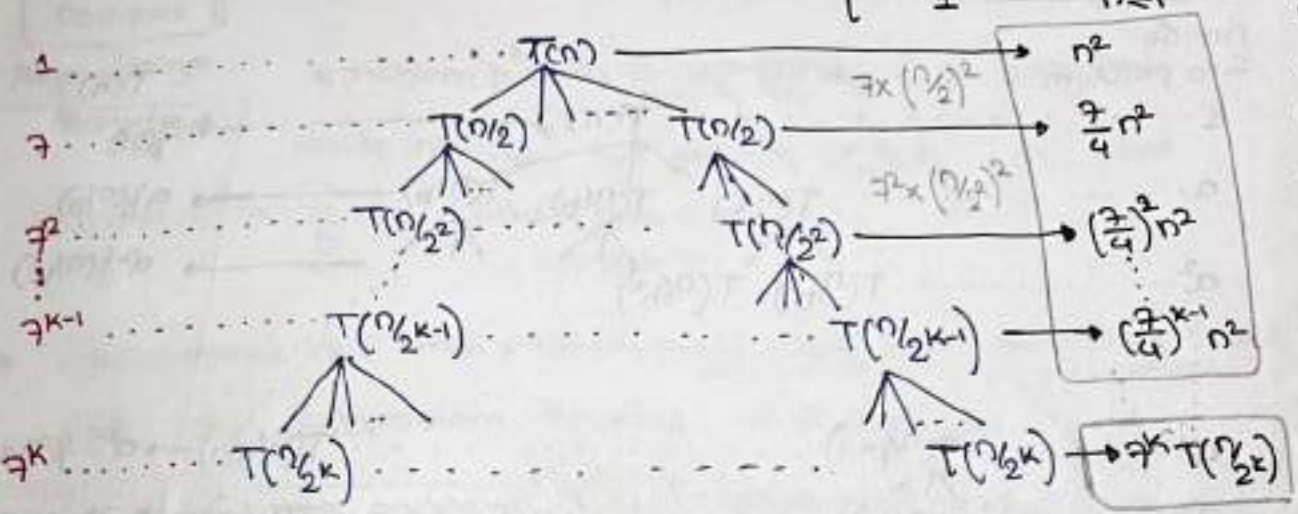
$= 8^{\log_2 n} \cdot 1 + n^2 [n-2]$

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

$T(n) = \theta(n^3)$

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

ex: Solve using recursive tree method $T(n) = \begin{cases} 7T(n/2)+n^2 & n>1 \\ 1 & n\le 1 \end{cases}$



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

$n/2^k = 1$

$n = 2^k \Rightarrow \boxed{k = \log_2 n}$

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

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

$$T(n) = n^{2.807} + \frac{4}{3}n^2\left[n^{\log_2(7/4)} - 1\right]$$

$$= n^{2.807} + \frac{4}{3}n^2\left[n^{\log_2 7 - \log_2 4} - 1\right]$$

$$= n^{2.807} + \frac{4}{3}n^2\left[n^{0.807} - 1\right]$$

$$\boxed{T(n) = n^{2.807}}$$

ex: $T(n) = \begin{cases} 2T(\sqrt{n}) + \log_2 n & n > 2 \\ 2 & n \le 2 \end{cases}$

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

$T(n) = 2(2T(n^{1/2^2}) + \log_2 n^{1/2}) + \log_2 n$

$= 2^2 T(n^{1/2^2}) + 2 * \frac{1}{2}\log_2 n + \log n$

$T(n) = 2^3 T(n^{1/2^3}) + \frac{2^3}{2^2} * \frac{1}{2}\log_2 n + \log n + \log n$

$\vdots$

$T(n) = 2^k T(n^{1/2^k}) + \log n + \log n + \cdots + \log n$ (K time)

$= 2^k T(1) + K \log n$

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

$\log_2 n = 2^k \Rightarrow \boxed{K = \log_2 \log_2 n}$

Can also be said as no. of level.

gate 2006

Q. $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + 1$, $T(1) = 1$

$T(n) = 2T(\sqrt{n}) + 1$

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

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

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

$\vdots$

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

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

$\Rightarrow \log_2 n = 2^k \Rightarrow \boxed{K = \log_2 \log_2 n}$

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

$\Rightarrow 2^{\log_2 \log_2 n} - 1 \Rightarrow \log_2 n^{\log_2 2} - 1$

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

$T(n) = 2^k T(1) + k \log n$

Putting 'k' value

$\Rightarrow 2^{\log_2 \log_2 n}(1) + (\log n \cdot \log\log n)$

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

$\Rightarrow \log_2 n + \log_2 n \cdot \log\log n$

$\boxed{T(n) = \Theta(\log n \cdot \log\log n)}$

note

- floor, step funct^n, ceiling will not effect alot so ignore those funct^n

gate 2007 → Variation.

ex: $T(n) = T(\lfloor \sqrt{n} \rfloor) + n$

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

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

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

$\vdots$

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

$\Rightarrow \boxed{K = \log_2 \log_2 n}$

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

Gate 2007

ex: $T(n) = T(\lfloor \sqrt{n} \rfloor) + 1$

$T(n) = T(\sqrt{n}) + 1$

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

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

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

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

$\vdots$

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

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

So $T(n) = \log_2 \log_2 n$

Gate 2021

ex: $T(n) = \begin{cases} T(n/2) + T(\frac{2n}{5}) + 7n & n > 0 \\ 1 & n = 0 \end{cases}$

level 0: $7n$ ——→ $7n$

level 1: $7(\frac{n}{2})$ $7(\frac{2n}{5})$ - - → $7(\frac{9}{10})n$

level 2: $7(\frac{n}{2^2})$ $7(\frac{2n}{10})$ $7(\frac{2}{10}n)$ $7(\frac{2}{5})^2 n$ $(\frac{9}{10})^2 n$

$K_1 = \log_2 n$ $K_2 = \log_{5/2} n$

$T(n) = 7n\left\{1 + (\frac{9}{10}) + (\frac{9}{10})^2 + \cdots (\frac{9}{10})^{K-1}\right\}$

$7(n) \left[\frac{(\frac{9}{10})^K - 1}{\frac{9}{10} - 1}\right]$ $7(n) \frac{1 \cdot \left[1 - (\frac{9}{10})^K\right]}{1 - \frac{9}{10}}$

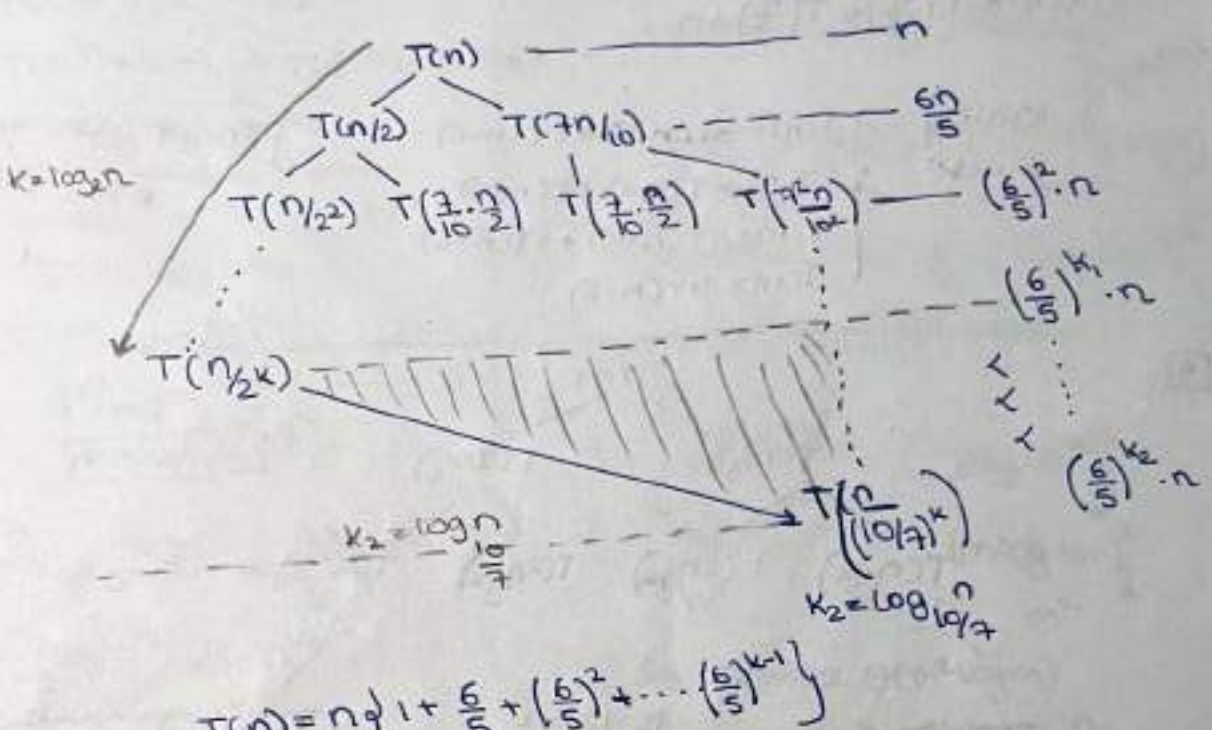
$\frac{9}{10} < 1$ → Smaller than 1

$T(n) = 7n\left[1 - (\frac{9}{10})^K\right] = 7n\left[1 - (\frac{9}{10})^K\right]$

$\frac{1}{10}$ Decreasing function

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

Q4] $T(n) = \begin{cases} T(n/2) + T(\frac{7n}{10}) + n & n>1 \\ 1 & n \le 1 \end{cases}$



$$T(n) = n\left\{1 + \frac{6}{5} + \left(\frac{6}{5}\right)^2 + \cdots \left(\frac{6}{5}\right)^{k-1}\right\}$$

$$5n\left\{\left(\frac{6}{5}\right)^k - 1\right\} \approx \boxed{n \cdot \left(\frac{6}{5}\right)^k}$$

$T(n) \ge n \cdot \left(\frac{6}{5}\right)^{\log_2 n}$

$T(n) \ge n \cdot n^{\log_2 6/5}$

$T(n) \ge n \cdot n^{0.26}$

$T(n) \ge n^{1.26}$

$\boxed{T(n) = \Omega(n^{1.26})}$

$T(n) \le n \cdot \left(\frac{6}{5}\right)^{\log_{10/7} n}$

$T(n) \le n \cdot n^{\log_{10/7} 6/5}$

$T(n) \le n \cdot n^{0.51}$

$T(n) \le n^{1.51}$

$\boxed{T(n) = O(n^{1.51})}$

but we can't say $\theta(n^{1.26}$ or $n^{1.51})$ b/c not equal

Similarly to gate 2021

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

5) $T(n) = T(\frac{3n}{5}) + T(\frac{4n}{5}) + n$

6) $T(n) = T(\frac{n}{8}) + T(\frac{n}{7}) + n$

generating funct^n

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

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

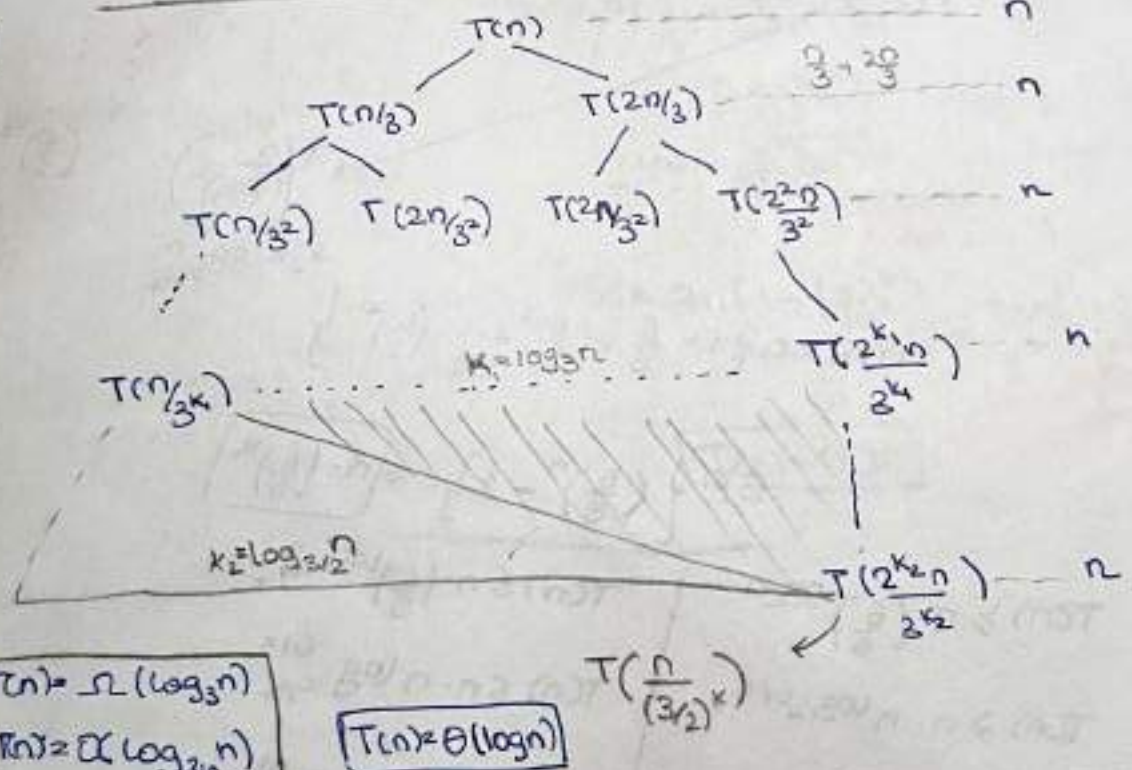
$T(n) = 2T(n-5) + 3T(n-6)$

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

$T(n) = T(\frac{n}{5}) + T(\frac{4n}{5}) + n$

Recursion Tree method

4

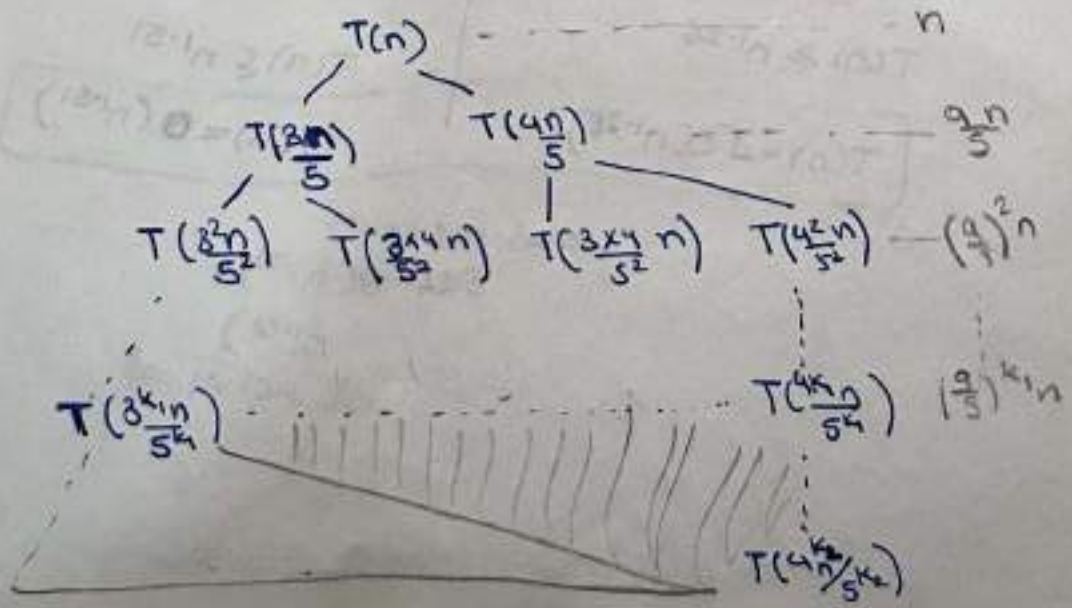


$T(n) = \Omega(\log_3 n)$

$T(n) = O(\log_{3/2} n)$

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

5



ex: $T(n) = T(\sqrt{n}) + \sqrt{n}$

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

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

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

$S(m) = S(m/2) + 2^{m/2}$

applying master theorem

we get

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

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

& $n = 2^m$

$\boxed{m = \log_2 n}$

So,

$2^{m/2} \Rightarrow 2^{\frac{\log_2 n}{2}} \Rightarrow 2^{\frac{1}{2}\log_2 n}$

$\Rightarrow 2^{\log_2 n^{1/2}} \Rightarrow n^{1/2}$

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

ex. $T(n) = 161^2\, T(\sqrt[16]{n}) + 161\,(\log n)^2$

$n = 161^m \Rightarrow \sqrt[16]{n} = 161^{m/161}$

$T(161^m) = 161^2\, T(161^{m/161}) + 161\,(\log 161^m)^2$

$T(161^m) = S(m) \Rightarrow T(161^{m/161}) = S(\frac{m}{161})$

So,

$S(m) = 161^2\, S(\frac{m}{161}) + (\log 161^m)^2$

applying master theorem

$m^{\log_{161} 161^2} = m^2$ $\qquad$ $(\log 161^m)^2 = (m \log 161)^2 = m^2$

$m^2 = m^2$

So $S(m) = \theta(m^2 \log m)$

& $n = 161^m \Rightarrow m = \log_{161} n$

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

ex: $T(n) = T(\sqrt[4]{n}) + T(\sqrt{n}) + \log n$

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

$\Rightarrow \sqrt[4]{n} = 2^{m/4}$

$T(2^m) = T(2^{m/4}) + T(2^{m/2}) + \log 2^m$

$T(2^m) = S(m) \Rightarrow T(2^{m/4}) = S(m/4)$

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

$S(m) = S(\frac{m}{4}) + S(\frac{m}{2}) + m$

using recursive tree

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

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

gate 2024

Q. $T(n) = \sqrt{n}\, T(\sqrt{n}) + 100n$

Simplify

$\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 100$ } divide n both side

$\Rightarrow \frac{T(2^m)}{2^m} = \frac{T(2^{m/2})}{2^{m/2}} + 100$

$S(m) \quad S(m/2)$

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

$m = \log_2 n$

$S(m) = \frac{T(2^m)}{2^m} \Rightarrow S(m) = S(m/2) + 100$

$S(m) = \Theta(\log m) \Rightarrow \frac{T(2^m)}{2^m} = \Theta(\log m)$

$\Rightarrow T(2^m) = \Theta(2^m \cdot \log m) \Rightarrow T(n) = \Theta(2^{\log_2 n} \log\log_2 n)$

$\Rightarrow \boxed{\Theta(n \log\log n)}$

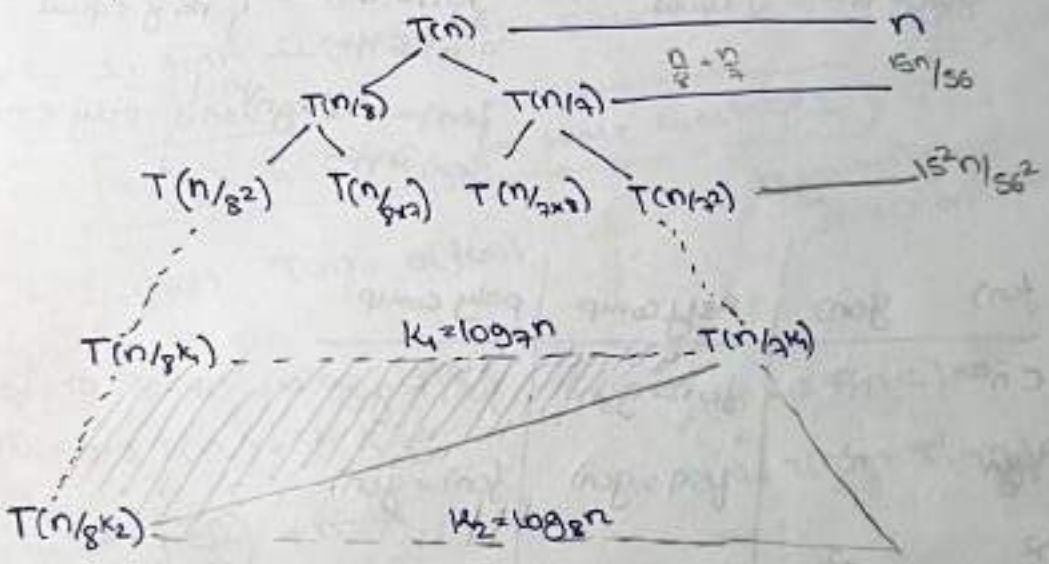
$T(n) = n\left\{1 + \frac{9}{8} + \left(\frac{9}{8}\right)^2 + \cdots\cdots + \left(\frac{9}{8}\right)^{k-1}\right\}$

$n\left\{\frac{\left(\frac{9}{8}\right)^k - 1}{\frac{9}{8} - 1}\right\} = \frac{8}{1} n\left\{\left(\frac{9}{8}\right)^k - 1\right\} \approx \Theta\left(n \cdot \left(\frac{9}{8}\right)^k\right)$

$k_1 \to \log_{8/3} n \qquad k_2 = \log_{8/4} n$

$T(n) \geq n$

⑥ $T(n) = T(n/8) + T(n/7) + n$



$T(n) = n\left\{1 + \frac{15}{56} + \left(\frac{15}{56}\right)^2 + \cdots\cdots + \left(\frac{15}{56}\right)^{k-1}\right\}$

$\Rightarrow n\left[\frac{\left(\frac{15}{56}\right)^k - 1}{\frac{15}{56} - 1}\right] \Rightarrow -\frac{56n}{41}\left\{\left(\frac{15}{56}\right)^k - 1\right\}$

$\Rightarrow -\frac{56}{41} n\left\{\left(\frac{15}{56}\right)^{\log_7 n} - 1\right\}$] for k_1

$\Rightarrow -\frac{56}{41} n\left\{n^{\log_8 15/56} - 1\right\} \Rightarrow \Theta(n)$

$k_1 = \log_7 n$

$\log_2 = \log_8 n$

for k_2, $-\frac{56}{41} n\left\{n^{\log_8 15/56} - 1\right\} \Rightarrow \Theta(n)$

Master theorem

Asy. comp

Const. < log fun < poly. functn < exp fun

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

[f(n) g(n) Asy. equal]

$\frac{f(n)}{g(n)} = \text{const. or log fun}$

f(n) g(n) poly equal

$f(n) = 10n^2$
$g(n) = n^2$
} Asy equal

$f(n) = 10 \cdot n^2$
$g(n) = n^2$
} poly equal

$f(n) = n^2 (\log n)^{10}$
$g(n) = n^2$
} poly equal

ex:

f(n)	g(n)	Asy comp	poly comp
$c \cdot n^2$	n^2	f(n) = g(n)	f(n) = g(n)
$n^2 \log n$	n^2	f(n) > g(n)	f(n) = g(n)
$\frac{n^2}{c(\log n)^2}$	n^2	f(n) < g(n)	f(n) = g(n)
$n^1 (\log n)^{10}$	$n^{1.1}$	f(n) < g(n)	f(n) < g(n)
n^2	$n^{2.7}$	f(n) < g(n)	f(n) < g(n)
e^n	n^3	f(n) > g(n)	f(n) > g(n)
$n!$	n^2	f(n) > g(n)	f(n) > g(n)

Master theorem

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

based Cond. $T(1) = \text{const.}$

i.e. n^c (poly time bigger than $f(n)$)

case 1: if $f(n) = O(n^{\log_b a - \epsilon})$ $\epsilon > 0$ const. $\longrightarrow$ $f(n) \leq \frac{n^{\log_b a}}{n^\epsilon}$

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

case 2: If $f(n) = \Theta(n^{\log_b a})$ ← depth of Rec

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

case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\epsilon > \text{const}$ $\longrightarrow$ $f(n) \geq n^{\log_b a + \epsilon}$, $\frac{f(n)}{n^\epsilon} \geq n^{\log_b a}$

(and)

$af\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some const. $c < 1$

regularity condition

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

- failed to solve using MTS's

1. $T(n) = 0.5\,T\left(\frac{n}{2}\right) + n^2$ ✗
2. $T(n) = 2T\left(\frac{3n}{2}\right) + n^2$ ✗
3. $T(n) = 2T\left\lfloor\frac{n}{3}\right\rfloor \rightarrow [n - n^2]$ ✗
4. $T(n) = 8T(n-2) + n^2$ ✗
5. $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n^2$ ✗
6. $T(n) = 2T(n) + n^2$ ✗

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

(case 2) if $n^2 = O(\log_2 7)$

$n^2 \leq n^{2.81 - \epsilon}$

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

$\Theta(n^{2.81})$

$T(n) = 16\,T\left(\frac{n}{4}\right) + n^2$

Case 1] $n^2 = O(\log_4 16 - \epsilon)$ ✗

$n^2 \leq n^2 - \epsilon$

Case ② $n^2 \cdot \Theta(n^{\log_4 16})$

$T(n) = \Theta(n^2 \cdot \log_4 2)$

$T(n) = 20\,T\left(\frac{n}{3}\right) + \Theta(n^3)$

Case 1: $n^3 = O(n^{\log_3 20 - \epsilon})$

Case 2: $n^3 = \Theta(n^{\log_3 20})$

Case ③ : $n^3 = \Omega(n^{\log_3 20 + 1})$

$n^3 \geq n^{2.7 - \epsilon}$

$\therefore \boxed{T(n) = \Theta(n^3)}$

$f(n) = O(n^{\log_b 9 - \epsilon})$

$f(n) \leq \frac{n^{\log_b 9 - \epsilon}}{n^\epsilon}$ ← poly

$f(n) \quad n^{\log_b 9}$

✗ $\frac{n^2}{\log n} \quad n^2$

✗ $10 n^2 \quad n^2$

$10 n^2 \quad n^3$

$f(n) \leq n^{\log_b} / n^\epsilon$

$\frac{n^2}{\log(n)} \leq \frac{n^2}{n^\epsilon}$

$10 n^2 \leq \frac{n^2}{n^\epsilon}$

$10 n^2 \leq \frac{n^3}{n^\epsilon}$

Master theorem $T(n) = aT(n/b) + f(n)$ · $a \geq 1$, $b > 1$ const.

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

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

case ② if $n^{\log_b a}$, $f(n)$ Asy equal

then $T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$ ← depth of Rec

case ③ if $f(n)$ polynomially bigger than $n^{\log_b a}$

(and)

regularity Condition: $af(\frac{n}{b}) \leq c \cdot f(n)$ for some const $c < 1$

Then · $T(n) = \Theta(f(n))$

ex: $T(n) = 7\left[T(n/2)\right] + n^2$

$f(n) = n^2$ | $n^{\log_b a} = n^{\log_2 7} = n^{2.81}$

$\Theta(n^{2.81})$ $\quad n^2 < n^{\log_2 7}$ poly

$n^{\log_b a} \underset{\text{Asy}}{=} f(n)$

ex: $T(n) = 16\,T(\frac{n}{4}) + n^2$

$f(n) = n^2$ · | $n^{\log_4 16} = n^2$

$T(n) = \Theta(n^2 \cdot \log_4 n)$

ex: $T(n) = 20\,T(n/3) + n^3$

$f(n) = n^3$ // poly bigger

$n^{\log_b a} = n^{\log_3 20} = n^{2.7}$

$af(\frac{n}{b}) \leq c \cdot f(n)$ for $c < 1$

$20 \cdot (\frac{n}{3})^3 \leq c \cdot n^3$

$\frac{20}{27} \cdot n^3 \leq \underbrace{c \cdot n^3}_{\text{for } c < 1}$

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

① $T(n) = 2T(n/2) + \sqrt{n}$

$n^{\log_2 2} \Rightarrow (n^1) > \sqrt{n}$

$\Rightarrow \boxed{\Theta(n)}$

② $T(n) = T(n/2) + \sqrt{n}$ $\boxed{\Theta(\sqrt{n})}$

$n^{\log_2 1} \Rightarrow n^0 < \sqrt{n}$

poly

$af(n/b) \leq c \cdot f(n)$

$1 \cdot \sqrt{\frac{n}{2}} \leq c \cdot \sqrt{n}$

for $c \leq 1$

③ $T(n) = T(n/5) + \text{const}^n$

$n^{\log_5 1}$ 1

$(n^0) \; 1 = 1$

$\Rightarrow \boxed{\log_5 n}$

④ $T(n) = 8T(n/2) + n^2 [\log n]^{10}$

$n^{\log_2 8}$ $n^2 (\log n)^{10}$

$n^3 >$

$\Rightarrow (n^3)$

⑤ $T(n) = T(n/5) + n$

$n^{\log_5 1}$ n

$n^0 < n$

$\Rightarrow \boxed{n} \Rightarrow \Theta(n)$

⑥ $T(n) = 6T(\frac{n}{3}) + n \log n$

$n^{\log_3 6} > n \log n$

$\Theta(n^{\log_3 6}) = \Theta(n^{1.11})$

⑦ $T(n) = 2T(n/2) + n!$

$n^{\log_2 2} < n!$

$n^1 < n!$

$\boxed{\Theta(n!)}$

⑧ $T(n) = 1.5 T(\frac{n}{2}) + n^2 \log_2 n$

$n^{\log_2 1.5} < n^2 \log_2 n$

$\boxed{\Theta(n^2 \log_2 n)}$

$af(n/b) \leq c \cdot f(n)$

$1.5 (n/2)^2 \log_2 (n/2) \leq c \cdot n^2 \log n$

$c < 1$

⑨ $T(n) = 16 T(n/4) + 2^n$

$n^{\log_4 16} < 2^n$

$\boxed{\Theta(2^n)}$

⑩ $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n$

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

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

11] $T(n) = T(\frac{n}{2}+5) + T(\frac{n}{2}-2) + n$ → for large value $\frac{n}{2}$, 2 is can be neglect

$\approx T(n/2)$ $\quad = T(n/2)$

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

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

here 2 can't be given bic 1 is the only one for the decrement

12] $T(n) = 2T(n/3 + 10) + n^2$

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

$\Rightarrow \Theta(n^2)$

13] $T(n) = T(n/5) + T(\frac{4n}{5}) + n \Rightarrow$ failed to solve by MT's

14] $T(n) = 2T(n-2) + n^2$ → failed to solve by MT's.

15] $T(n) = 2T(n/2) + n \log_2 n$

$n^{\log_2 2} = n \log_2 n$ poly — polyequal but Asy not equal

$\Theta(n \log_2 n)$ ×

$\Rightarrow \boxed{\Theta(n(\log_2 n)^2)}$ → modified formula

16] $T(n) = 64T(\frac{n}{4}) + n^3(\log n)^r$

$n^{\log_4 64} = n^3$ poly $= n^3(\log n)^r$

$\Rightarrow \Theta(n^3 \log_4 n)$ ×

$\Theta(n^3(\log_4 n)^6)$

17] $T(n) = 64T(\frac{n}{8}) + \frac{n^2}{\log n}$

$n^{\log_8 64}$ $\quad$ $n^2/\log n$

n^2 $\quad$ fcn

for $k = -1$

$\Theta(n^2 \cdot \log(\log n))$

Master theorem failure cond.

- $T(n) = aT(n/b) + f(n)$ $a \geq 1, b > 1$

If $n^{\log_b a}$ & $f(n)$ are polynomially equal but Asy not equal

Then MT's failed to solve Recurrence relation

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

$n^{\log_2 2}$ $\quad$ $n \log n$

n^1 $\quad$ $n \log n$

case ② if $f(n) = \Theta(n^{\log_b a} \cdot (\log n)^k)$

① for $k \geq 0$

$T(n) = \Theta(n^{\log_b a} \cdot (\log n)^{k+1})$ → if $f(n)$ Asy. bigger $(\log n)^k$ times than $n^{\log_b a}$

ex: $f(n) = n^2 \cdot (\log n)^3$

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

② for $k = -1$

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

If $f(n)$ Asy smaller $(\log n)$ time than $n^{\log_b a}$

ex: $f(n) = \frac{n^2}{\log n}$

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

Master theorem failure cond:

$T(n) = aT(n/b) + f(n)$ $a \geq 1, b > 1$

- If $n^{\log_b a}$ & $f(n)$ are polynomially equal but Asy. not equal

Then MT failed to solve recurrence relation

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

case ② true for $k = -1$

$n^{\log_b a} = \underbrace{n^{\log_2 2}}_{n^1}$ $\Theta(n \log\log n)$

$f(n) = \frac{n}{\log n}$

(19) $T(n) = 4T(n/2) + n^2(\log n)^7$

$n^{\log_b a} \geq \underbrace{n^{\log_2 4}}_{n^2}$

$f(n) = n^2(\log n)^7$

So, case ② true for $k \geq 0$

$\Theta(n^2 \cdot (\log n)^8)$

(20) $T(n) = 2T(n/2) + n(\log n)^5$

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

n^1

$f(n) = \frac{n}{(\log n)^r}$

no case for the as is k=5 here smaller.
(M.T.'s failed)

(21) $T(n) = 4T(n/2) + n^2 \cdot (\log\log n)^5$

$f(n) = n^2(\log\log n)^5$

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

(MT's failed)

(1) $T(n) = T(n/2) + n[2 - \cos 2\pi n]$

$f(n) = n[2 - \cos 2\pi n]$ // poly bigger.

$n^{\log_b a} = n^0 = 1$

* $af(n/b) \leq c \cdot f(n)$ & $c < 1$

$1 \cdot \frac{n}{2}[2 - \cos\frac{2\pi n}{2}] \leq c \cdot n[2 - \cos 2\pi n]$

n=5

$\frac{5}{2} \cdot [2 - (-1)] \leq c \cdot [2 - (+1)]$

$\frac{5}{2} * 3 \leq c \cdot 1 \cdot 5$

$\frac{3}{2} \leq c \cdot 2$

for $c > 1$

Master theorem failed

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

$T(n) = n^{\log_b a} \cdot T(1) + f(n) + af(\frac{n}{b}) + \cdots + a^{\log_b n - 1} f(\frac{n}{\log_b n})$

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

$T(n) = n^{\log_b a} \cdot T(1) + f(n) + a f(\frac{n}{b}) + \cdots + a^{\log_b n - 1} f(\frac{n}{\log_b n - 1})$

Case 1: $n^{\log_b a} > f(n)$ (poly) —— $T(n) = \theta(n^{\log_b a})$

Case 2: $n^{\log_b a} = f(n)$ (Asy) —— $T(n) = \theta(n^{\log_b a} \cdot \log_b n)$

Case 3: $n^{\log_b a} < f(n)$ (poly) —— $T(n) = \theta(f(n))$

$a f(\frac{n}{b}) \leq c \cdot f(n)$, $c < 1$

Sum of series equal to $f(n)$

$n^{\log_b a} < f(n)$ (poly)

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

$a f(n/b) \leq c \cdot f(n)$ not true for some $c > 1$

Sum of series. Asy. bigger than $f(n)$

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

$T(n) = a^k n^{\log_b a} T(1) + a^k f(n) + a^2 f(n/b) + a^2 f(n/b^2) + \cdots + a^{k-1} f(n/b^{k-1})$

1. $n^{\log_b a} > f(n)$ (poly)

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

i.e

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

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

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

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

$n^{\log_b a} \leq \left[\sum_{i=1}^{k-1} a^i f(n/b^i)\right]$

$k = \log_b n$

case 1

2. $\frac{n^{\log_b a}}{\log n} \overset{Asy}{=} f(n)$

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

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

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

$T(n) = 27T(n/3) + n^3/\log n$

$\therefore \{1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{\log n}\}$

Sum of series $= n^{\log_b a} \cdot \log\log n$

3. $\frac{n^{\log_b a}}{(\log n)^5} = f(n)$ Asy

$T(n) \geq \Omega(n^{\log_b a})$

$T(n)$: Asy equal to sum of series

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

$T(n) = 4T(n/2) + \frac{n^2}{(\log\log n)^5}$

$T(n) = 27T(n/3) + n^3/(\log n) \cdot \log\log n$

$n^{\log_b a}$ & $f(n)$ poly equal & Asy · not equal

$\Rightarrow$ MT's fails &

$n^{\log_b a} \overset{Asy}{>} f(n)$ & poly · equal

4. $n^{\log_b a} = f(n)$ Asy

depth

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

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

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

$T(n) = 27T(n/3) + n^3$

$T(n) = 64T(n/2) + n^6$

case (2)

[5] $n^{\log_b a} * (\log n)^k = f(n)$ (asy)

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

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

$T(n) = 4T(n/2) + n^2 [\log n]^5$

$T(n) = 8T[\frac{n}{2}] + n^3 [\log n]^9$

Sum of series = $n^{\log_b a} [\log n]^{k+1}$

$n^{\log_b a} <$ (asy) $f(n)$ ✓ poly equal

[6] $n^{\log_b a} \cdot (\log\log n)^k \overset{asy}{=} f(n)$

$f(n) >$ (asy) $n^{\log_b a}$

but not exactly $(\log n)^k$

bigger

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

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

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

Sum of series > $f(n)$

MT failed

[7] $f(n) >$ (poly) $n^{\log_b a}$

+

$af(n/b) \leq c \cdot f(n)$ for some $c < 1$

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

Sum of series = $\Theta(f(n))$

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

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

$T(n) = 26\,T(n/3) + n^3$

Case (3)

[8] $f(n) >$ (poly) $n^{\log_b a}$

$af(n/b) \leq c \cdot f(n)$ only true for $c > 1$

[not true for any/all $c < 1$]

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

Sum of series > $f(n)$

$T(n) = T(\frac{n}{2}) + x[2 - \cos 2\pi x]$

$T(n) = T[\frac{n}{2}] + x[2 + \cos \pi x]$

$T(n) = 2T(\frac{n}{2}) + n^2[2 - \cos \pi x]$

MT failed

∴ All series $T \cdot C$ = sum of series

the master theorem applies to recurrences

$T(n) = aT(n/b) + f(n)$ $\begin{cases} a \geq 1 \\ b > 1 \end{cases}$

3 cases

1. if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$

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

Extended

2.1 if $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$ with $k \geq 0$

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

2.2 if $f(n) = \Theta\left(\frac{n^{\log_b a}}{\log n}\right)$

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

2.3 if $f(n) = \Theta\left(\frac{n^{\log_b a}}{(\log n)^p}\right)$ with $p \geq 2$

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

3. if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ & $af\left(\frac{n}{b}\right) \leq c \cdot f(n)$

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

2.4 Asymptotic equal

if $f(n) = \Theta(n^{\log_b a})$

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

Change of Variable

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

$\sqrt[b]{n} = (b^m)^{1/b} = b^{m/b}$

Assume $n = b^m$

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

$T(b^m) = a[T(b^{m/b})] + f(b^m)$

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

$T(b^{m/b}) = S(m/b)$

$S(m) = a \cdot S(m/b) + f(b^m)$

Apply MT's

$S(m) = T(n) = \Theta(\quad)$

replace $m = \log_b n$

Ques] $T(n) = \begin{cases} T(\sqrt[5]{n}) + 1 & n \geq 5 \\ 1 & n < 5 \end{cases}$

$T(n) = T(\sqrt[5]{n}) + 1$

Assume $\boxed{n = 5^m} \Rightarrow \sqrt[5]{n} = 5^{m/5}$ $\left[\sqrt[5]{5^m} = (5^m)^{1/5}\right]$

$T(5^m) = T(5^{m/5}) + 1$

Assume ; $T(5^m) = S(m) \Rightarrow T(5^{m/5}) = S(\frac{m}{5})$

$S(m) = S(\frac{m}{5}) + 1$

Apply MT's

$\log_5^1 = \Theta(1)$ — Const Case 2

$S(m) = \Theta(\log_5 m)$

$S(m) = \boxed{T(n) = \Theta(\log_5 \log_5 n)}$

i.e $T(n) = \begin{cases} 3T(\sqrt[3]{n}) + 1 & n > 3 \\ 1 & n \le 3 \end{cases}$

$T(n) = 3T(\sqrt[3]{n}) + 1$

Assume $\Rightarrow \boxed{n = 3^m} \Rightarrow \sqrt[3]{n} = 3^{m/3}$

$\log_3 n = m$

$\Rightarrow T(3^m) = 3T(3^{m/3}) + 1$

Assume $T(3^m) = S(m)$

$\Rightarrow T(3^{m/3}) = S(m/3) \Rightarrow S(m) = 3S(\frac{m}{3}) + 1$

Apply MT.

$S(m) = \Theta(\quad, m)$

$S(m) \Rightarrow \boxed{T(n) = \Theta(\quad \log_3 n)}$

i.e $T(n) = \begin{cases} 16T(\sqrt[4]{n}) + n & n > 4 \\ 1 & n \le 4 \end{cases}$

$T(n) = 16T(\sqrt[4]{n}) + n$

Assume $\Rightarrow \boxed{n = 4^m} \Rightarrow \boxed{\sqrt[4]{n} = 4^{m/4}}$

$\Rightarrow m = \log_4 n$

$T(4^m) = 16T(4^{m/4}) + n$

Assume $\Rightarrow T(4^m) = S(m)$

$T(4^{m/4}) = S(m/4)$

$\Rightarrow S(m) = 16S(m/4) + 4^m$ } $\log_4 16 < 4^m$ case 3

Applying MT

$S(m) = \quad \Theta(4^m)$

$S(m) = \boxed{T(n) = \Theta(\log_4 n)} \Rightarrow 4^{\log_4 n} \Rightarrow n^{\log_4 4}$

$\Rightarrow \Theta(n)$

$$T(n) = \begin{cases} 16 \cdot T(\sqrt[4]{n}) + (\log_4 n)^2 & n > 4 \\ 1 & n \le 4 \end{cases}$$

$T(n) = 16 \cdot T(\sqrt[4]{n}) + (\log_4 n)^2$ $\quad \because n = 4^m$

$T(4^m) = 16 \cdot T(4^{m/4}) + m^2$

$[T(4^m) = S(m)]$

$S(m) = 16 \cdot S(m/4) + m^2$ Apply MT Case 2

$T(n) = S(m) = \Theta(m^2 \cdot \log_4 m) = \Theta((\log_4 n)^2 \cdot \log_4 \log_4 n)$

i.e $T(n) = 27\, T(\sqrt[3]{n}) + (\log_3 n)^2 \cdot \log_3 \log_3 n$

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

$T(n) = 27\, T(\sqrt[3]{n}) + (\log_3 n)^2 \cdot \log_3^2 \log_3 n$

$T(3^m) = 27\, T(3^{m/3}) + m^2 \quad \log_3 m$

$S(m) = 27\, T(m/3) + m^2 \log_3 m$ } case 1

$\Rightarrow \quad S(m) = \Theta(n^3)$

$n^{\log_3 27} > m^2 \log_3 m$

$m^3 > m^2 \log_3 m$

$\Rightarrow \Theta(\log_3 n)^3$

$\Theta(\log^3_3 n)$

note

if $T(\frac{2^m}{K}) = S(\log(\frac{2^m}{K}))$

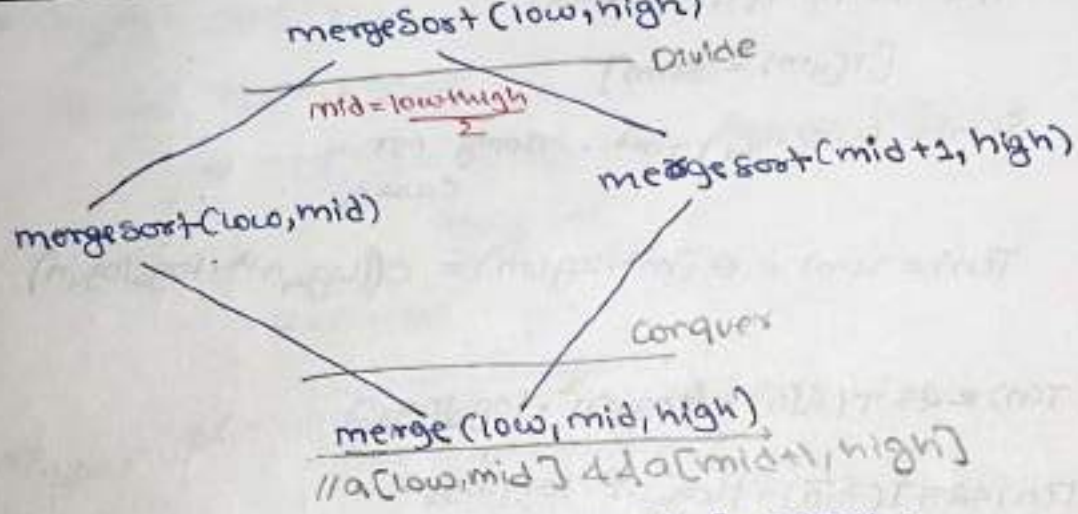
$= S(m - \log K)$

$T((2^m)^K) = S(mK)$

- Example of Divide & conquer.

Merge sort Algo

a[low, high] array of n element



- merge functn merge's two sort part of array a[low, mid] & a[mid+1, high] into single array

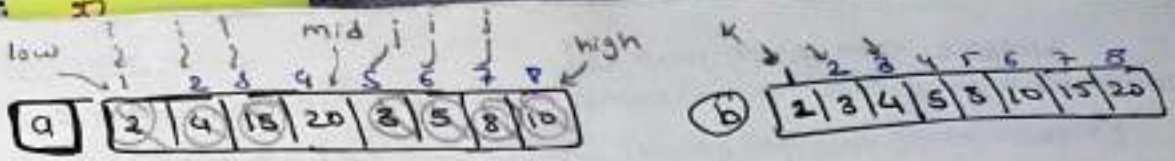
Algo MergeSort(low, high)// ——T(n)

```
if(low<high) //one than one element
{ //dividing cost ——O(1)
 mid=(low+high)/2;

 mergesort(low, mid); ——T(n/2)
 mergesort(mid+1, high); ——T(n/2)

 //Conquer
 // merge cost ——Θ(n)
 merge(low, mid, high);
}
}
```

Pascal

a[low, high] = high-low+1

a[1,1] - 1
a[2,1] - 0
a[1,10] - 10
} element



TC of merge fun : $\Theta(n)$

```
Algo. Merge(low,mid,high){
    i=low; j=mid+1; k=low;
    while(i≤mid && j≤high){
        if(a[i]≤a[j]){
            b[k]=b[i];
            i++; k++;
        }else{
            b[k]=b[j];
            k++; j++;
        }
    }
    if(i>mid)
        for(l=j; l≤high; l++)
        {b[k]=a[l]; k++;}
    else{
        for(l=i; l≤mid; l++)
        {b[k]=a[l]; k++;}
    }
    for(l=low; l≤high; l++){
        a[l]=b[l];
    }
}
```

TC recurrence Rel if merge sort to sort array of n elements:-

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

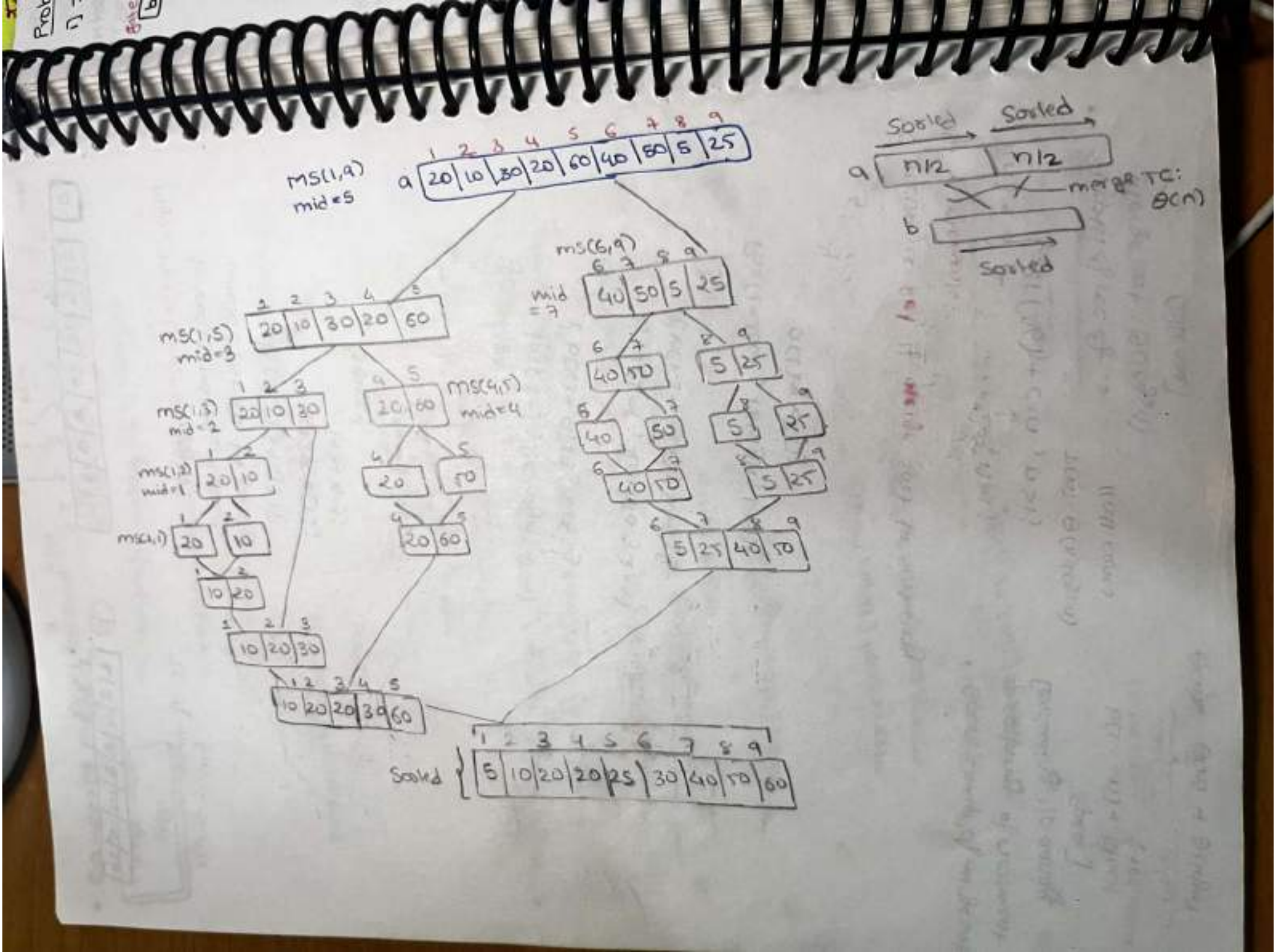
$T(n)=\Theta(n\cdot\log_2 n)$ // all cases

- ⇒ Depth of tree of merge sort : $\Theta(\log_2 n)$ [all case]

- Space comp^n of merge sort to sort array of n element [excluding i/p array space]

b[1....n] (extra array used) + stack if recursive also

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


Sorted
Sorted
a
n/2
n/2
merge TC: θ(n)
b
Sorted
ms(1,9)
mid=5
a
20 10 30 20 60 40 50 5 25
ms(1,5)
mid=3
20 10 30 20 60
ms(6,9)
mid=7
40 50 5 25
ms(1,2)
mid=1
20 10
ms(4,5)
mid=4
20 60
ms(1,1)
20
10
20
60
40
50
5
25
10 20
10 20 30
10 20 20 30 60
Sorted
5 10 20 20 25 30 40 50 60

Problem.

i) Inverse in an array is
$a[i] > a[j]$ && $i < j$

gate

[b] How many Avg. Inverse in array of n element.

$nC_2 * \frac{1}{2} = \frac{n(n-1)}{4}$

nC_2 pair, probability one pair Inv: 1/2 $\quad nC_2 * \frac{1}{2} = \frac{n(n-1)}{4}$

[c] What is TC required to compute #f Inverse in array of n element?

$\Theta(n \log n)$ // efficient algo

Method 1:

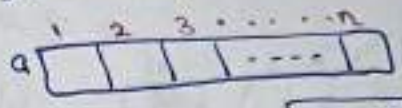
```
Inv=0
for(i=1; i<n; i++){
  for(j=i+1; j<=n; j++){
    if(a[i] > a[j]){ Inv++; }
  }
}
```

TC: $\Theta(n^2)$ SC: $\Theta(1)$

[Algo to find #f inverse]

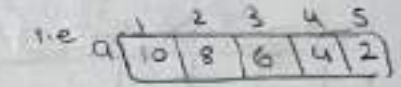
[a] How many max possible Inverse in array of n elements?

[every pair can form inverse]

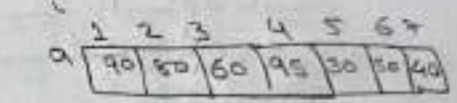


nC_2 pair = $\frac{n(n-1)}{2}$

i.e



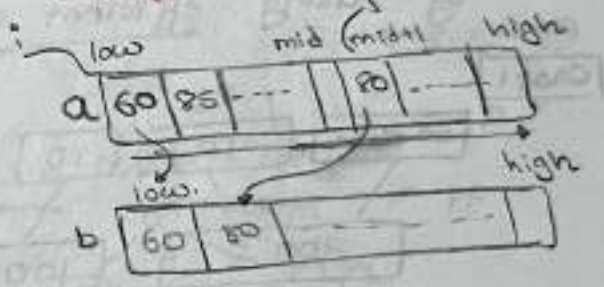
i



5+4+3+2+0+1 = [16 Inv]

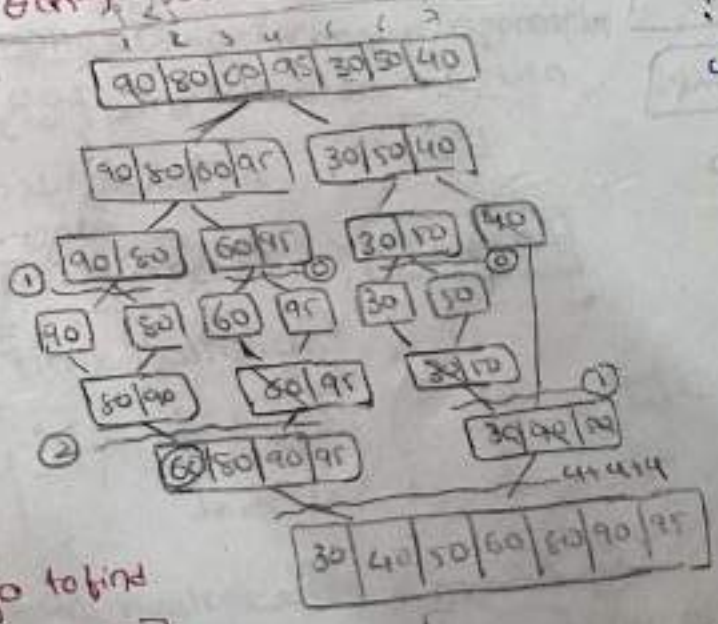
Method 2: [using merge sort]

change in merge fun:-



// rest of Merge sort

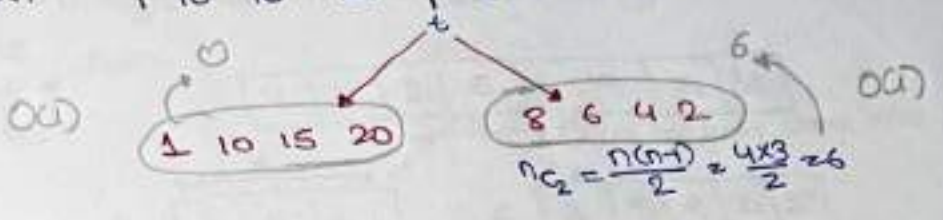
```
while(i<=mid && j<=high){
  if(a[i]<=a[j]){
    b[k]=a[i]; i++; j++; k++; }
  else{
    b[k]=a[j]; j++; j++; k++; }
  Inv = Inv + (mid-i+1);
}
}
}
```



↳ TC: $\Theta(n \log n)$ SC: $\Theta(n)$

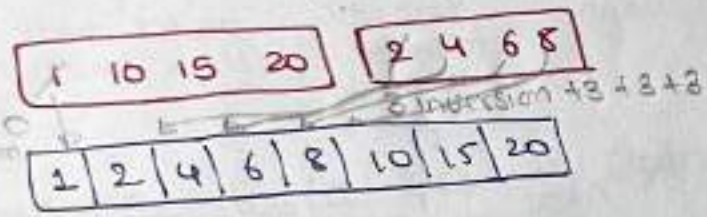
ex: An array A[1...n] is bitonic if there exists a t such that A[1...t] is increasing & A[t+1....n] is decreasing
what will be the time complexity to count all inversion in bitonic array?

ex: 1 10 15 20 8 6 4 2



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

- $O(n)$: to find t
- $O(1)$: to divide in 2 half
- $O(1)$: to solve right part
- $O(1)$: to solve left part
- $O(n)$: reverse the 2nd half
- $O(n)$: merge procedure



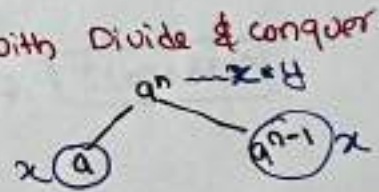
In total [3+3+3+3]+0+6
⇒ 12 inversion
+0+6
⇒ 18 inversion

- Exponent of a number

$a^n = a \cdot a \cdot a \cdot \dots \cdot a$

⇒ $O(n)$ time w/o Divide & Conquer

- with Divide & conquer



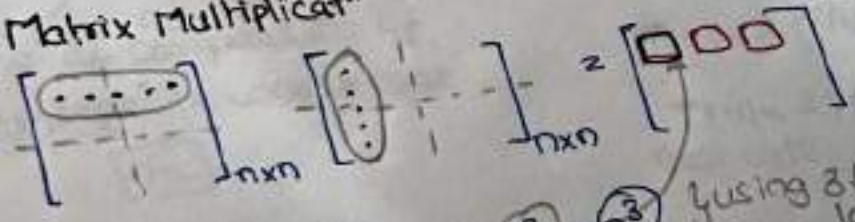
$T(n) = T(n-1) + 1 \Rightarrow \Theta(n)$

- another Divide & Conquer approach

$a^{n/2}$ { $a^{n/2}a^{n/2}$ when n is even
$a^{n/2}a^{n/2}a$ n is odd

$T(n) = T(n/2) + 1 \Rightarrow O(\log n)$

- Matrix Multiplicatn



total multiplication = $n \times n^2 = n^3$ (using 3 for loop)
n = no. of multiplication for 1 element
n^2 = no. of elements

8 multiplicatn

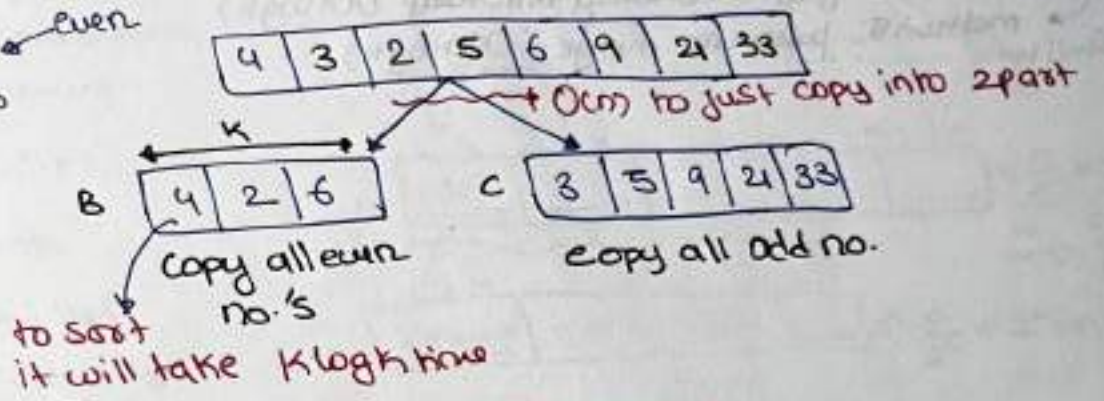
divide & Conquer ⇒ $T(n) = 8T(\frac{n}{2}) + n^2$

$\Rightarrow O(n^3)$ (n^2: combine cost of all matrix)

Strassens ⇒ $T(n) = 7(n/2)n^2$

ex: Integer array A is k-even mixed if there are exactly k-even integers in A, and the odd integers in A appear in sorted order. given a k-even mixed array A containing n distinct integers for $k = n/\log n$, describe an $O(n)$-time algo. to sort A.

Suppose even
$k = 10$
$n = 80$



$T(n) = n + K\log k + n$ ← to merge both arrays again

- n: to divide into two part (even & odd)
- $K\log k$: to sort k element of even array

$K\log k = \frac{n}{\log n} \cdot \log\left(\frac{n}{\log n}\right) \Rightarrow \frac{n}{\log n}(\log n - \log\log n) < n$ (ignore)

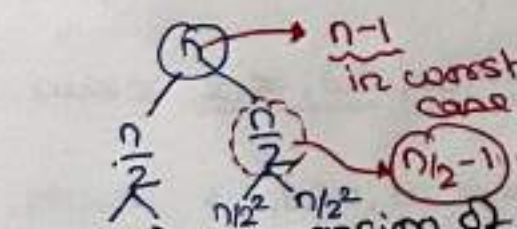
So, $T(n) = n + \Theta(n) + n \Rightarrow \boxed{T(n) = \Theta(n)}$

In general,

no. of comparsions that mergesort takes } recursive eqn

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

$T(n)$: no. of comparsion in mergesort for 'n' no.



- no. of comparsion of 'merge' in worst case

$\boxed{n-1}$

- no. of comparsion of mergesort in best case

$T(n) = 2T(n/2) + \min(\frac{n}{2}, \frac{n}{2})$

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

Best case

- To merge k sorted lists into one

method 1: trivial (append & sort) O(nlogn)

method 2: successive merge O(nk)

method 3: find min & put to array O(nk)

method 4: find min using min heap O(nlogk)

method 5: pairwise merge O(nlogk)

Similar

[3] (Iterative or Bottom up) 2-way merge sort

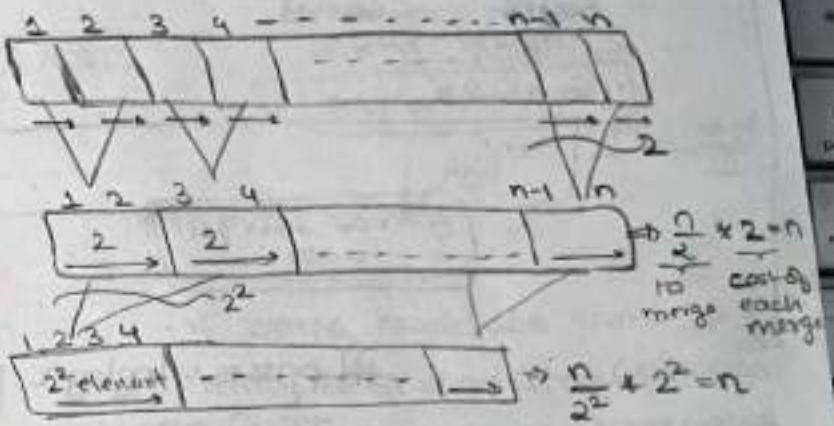
TC: $\Theta(n \log n)$

SC: $\Theta(n)$

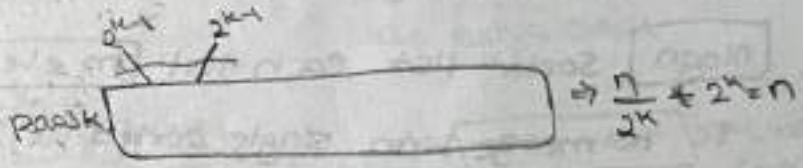
[Straight merge sort]

//non recursive mergesort

- Initially n elements
  n sorted parts each
  list 1 element
- $(n/2)$ Sorted lists each
  2 elements
- $\frac{n}{2^2}$ Sorted list
  each 2^2 elements



- $(n/2^k)$ Sorted list
  each 2^k element



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

$[k = \log_2 n]$ #f passes

Q] array of element are

2, 20, 5, 3, 90, 60, 70, 80, 25, 40, 65, 60, 15, 80, 95

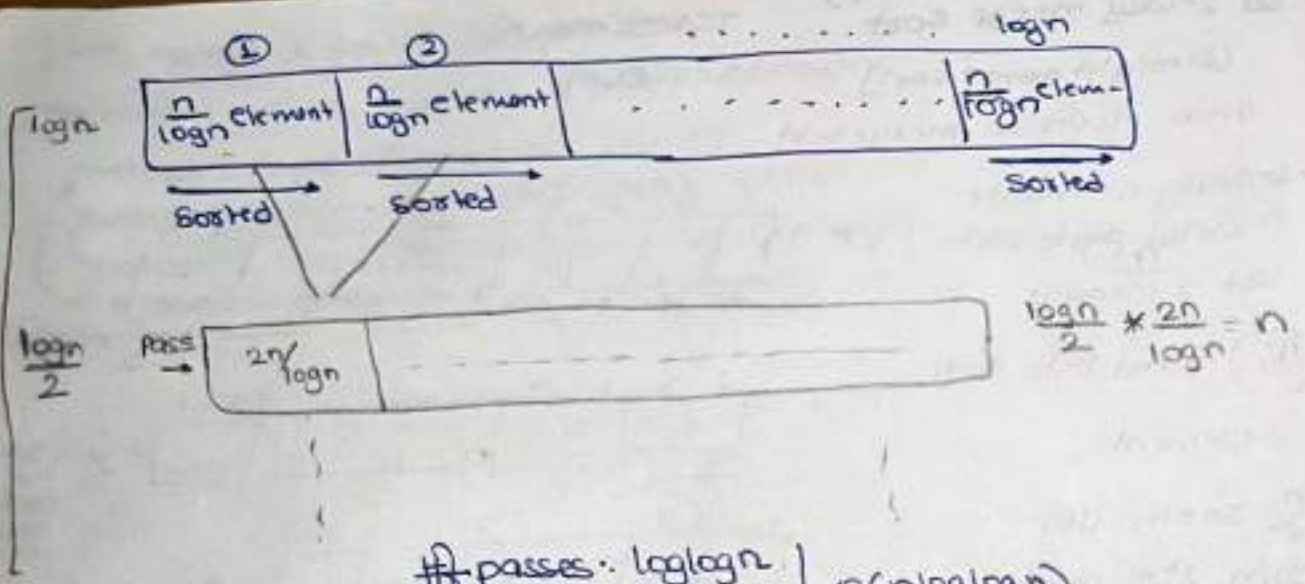
what is result array of pass 2 by using 2 way merge sort

pass 1: 2, 20, 3, 5, 60, 90, 70, 80, 25, 40, 60, 65, 15, 80, 95

pass 2: 2, 3, 5, 20, 60, 70, 80, 90, 25, 40, 60, 65, 15, 80, 95

Ques] what is TC required to merge $\log_2 n$ sorted part of array with $[n/\log n]$ element each into single sorted array.

$\log_2 n$ element



#f passes : loglogn
TC of one pass : n
} $\Theta(n \log\log n)$

i.e nlogn Sorted list each list m elements

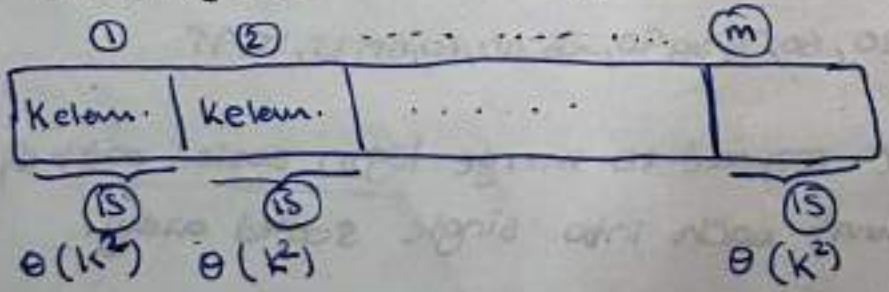
TC to merge into single sorted list?

Pass 1 TC $\Rightarrow \frac{n \cdot \log n}{2} * 2m = m \cdot n \cdot \log n$

#f passes : $\log[n\log n] = \log n + \log\log n$

TC : $mn \cdot \log n [\log n + \log\log n] \Rightarrow \Theta(m \cdot n \cdot \log^2 n)$

i.e Given m part of array each part K element . each K element Sorted using Insertion sort & followed by 2way merge sort to merge sort into single Sorted list what is TC?



1. TC of Insertion sort for m list each k element ⇒ $\theta(mk^2)$

2. merge sort

of pass : $\log_2 m$

TC of one pass :- $2k * \frac{m}{2} \approx mk$
(combine 2 element)

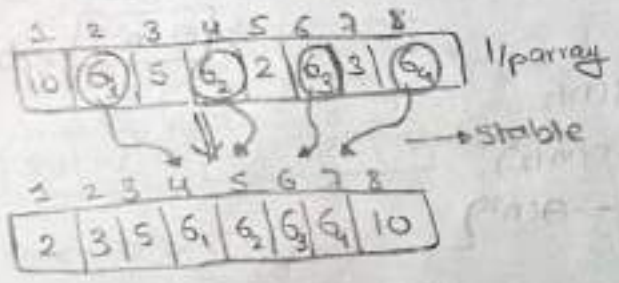
⇒ $mk * \log m$

$mk^2 + mk\log n$

$\theta(mk(k+\log m))$

Stable sorting Algo

- Identical element of given i/p array if occupied same relative order in Result of sorting Algorithm



- if not in relative space then unstable sorting

- merge sort is stable sorting Algo. but not inplace sorting Algo.

inplace sorting algo

- if sorting Algo. use same i/p array to sort an array (can use extra stack space if recursive Algo.) but logic should not use extra stack

a [n element] + stack can used if rec algo
i/p array

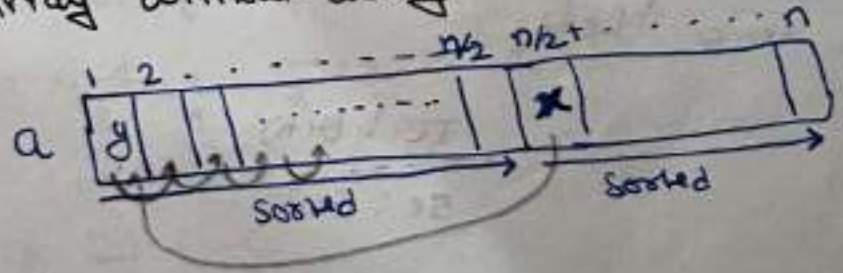
- non inplace sorting algo:-

a [n elements] + stack can used if rec algo

b [n elements] + extra array required
i/p array

ignoring the i/p array we still require $\theta(n)$ space

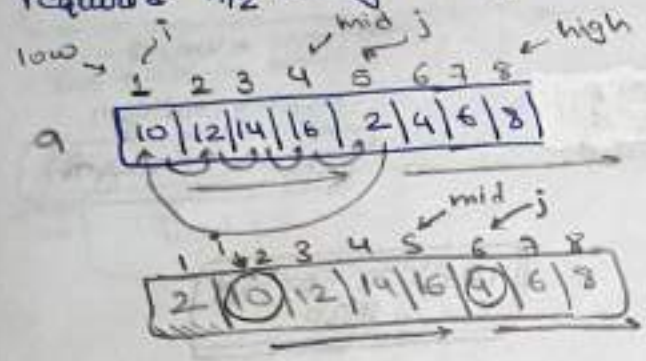
Ques/ WAC to merge two sorted part of array n/2 element each into single sorted array without using extra auxilary array?



w.c $n/2$ element of 2nd list to place in correct position

required $n/2$ array

Shift op each



n^2

Q) What is the worst case TC of sorting an nelement using merge sort without auxiliary array?

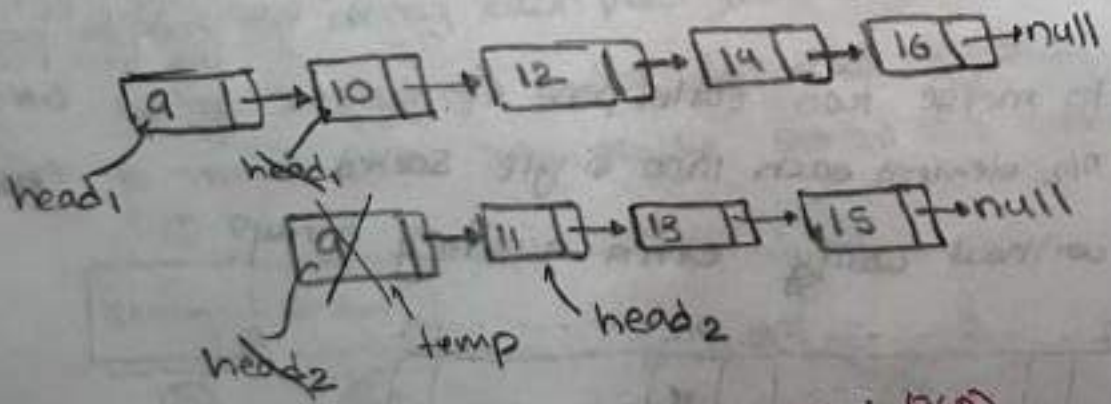
Algo MS(l,h) ⇒ T(n)

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

$\Rightarrow \boxed{\Theta(n^2)}$

```
{  if (l<h)?
     m=(l+h)/2;
     ms(l,m);      ——T(n/2
     ms(m+1,h);    ——T(n/2)
     merge(l,m,h); —Θ(n²)
   }
}
```

Ques] What is worst case TC to merge two sorted linked list of $n/2$ & $n/2$ element each into single sorted list?



TC : $\Theta(n)$

SC : $\Theta(1)$

- TC & SC to sort linked list of n element using merge sort

$T(n) = 2T(n/2) + n$ ← merge cost

$= \theta(n \log n)$

SC = $\theta(\log n)$ if recursive algo

- Merge sort is stable && Inplace sorting algo to sort linked list of n elements whose TC: $\theta(n \log n)$

Max-min Algo

a[1.....n] array of nelement find max, min element of array

	1	2	3	4	5	6	7
a	50	60	90	70	95	60	45

max = 50 90 95
min = 50 60 60 45

Straight Max Min Algo :- (not DandC)

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

comparison min: (n-1) if is true

comparison max: (n-1) if is false & 2nd is true

- min comparsion Required to find max-min of an element [n-1]
- max comparsion Required to find max-min of an element [2(n-1)]
- Avg comparsion Required to find max-min of an element.

$Avg = \frac{n-1}{2} * 1 + \frac{n-1}{2} * 2$ → 'if' condition always ful so you need ro go else part

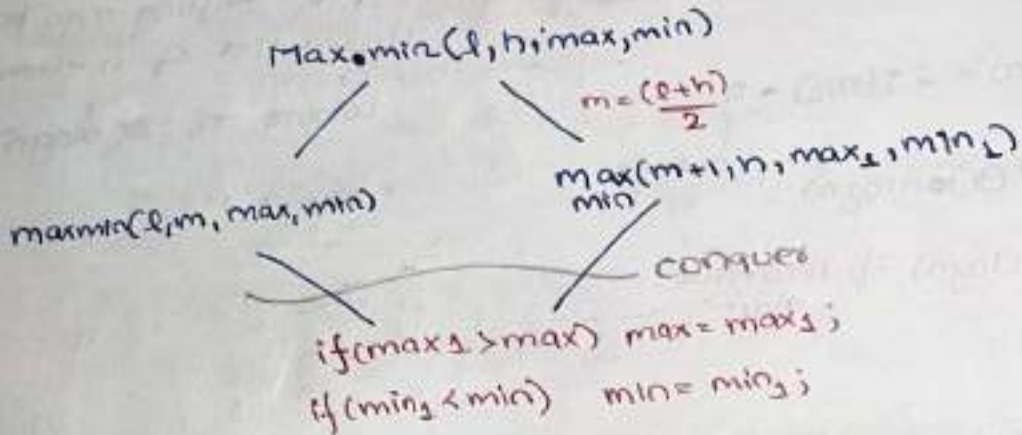
1 comp (always go to 'if')

2 comp

$\Rightarrow \frac{n-1}{2} + n - 1$

$\Rightarrow \frac{3n}{2} - 1.5$ Comp

D and C array of n elements. :- a[l,h] array of n element



- Algo RMax_Min(l,h,max,min){

```
if(l==h){
    min=max=a[l];
}
else if(l==h-1)
    {
        if(a[l]<a[h])
        {max=a[h]; min=a[l];}
        else
        {max=a[l]; min=a[h];}
    }
else{
    m=(l+h)/2;
    RMax_Min(l,m,max,min);
    RMax_Min(m+1,h,max1,min1);
    // conquers
    if(max1>max)  max=max1;
    If(min1<min)  min=min1;
}
return (max,min);
}
```

n=1 0 comp

n=2 1 comp

n>2 2T(n/2)+2

- Key observation:
  2nd largest can only be defeated by largest element ≡ 2nd max. is sibling of max. ($\log_2 n$ siblings)

- we can always find 2nd max. with $\log_2 n$ number

  Step 1 find max. using tournament method ≡ n-1 comparisons

  Step 2 find 2nd max. among logn siblings ≡ logn-1 comparisons

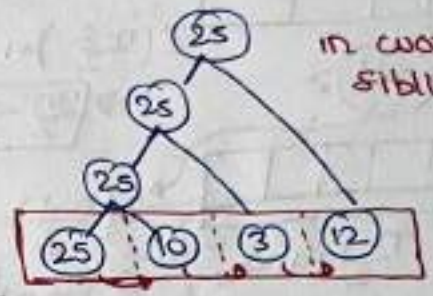
total comparisons ≡ n-1 + (n)-3

$n-1 + \log n - 1 = n + \log n - 2$

- $n-1$ for max
- $\log n - 1$ 2nd max. among siblings of max.

- max & 2nd Max in an array
  - worst case : $n + \lceil \log n \rceil - 2$

- skewed tournment



in worst case siblings = n-1

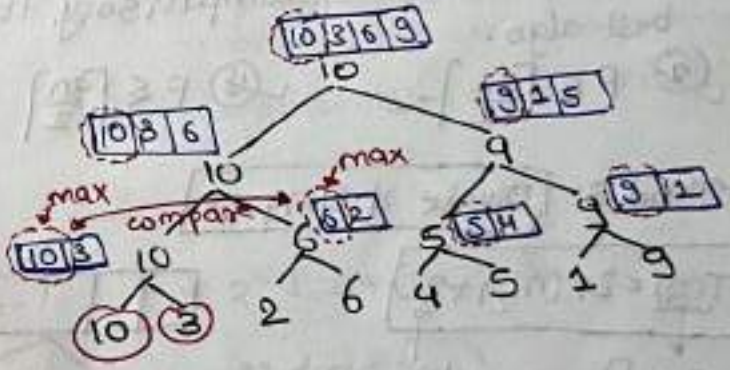
After we find 1st max ⇒ we only **logn** element are eligible for 2nd max

- $n-1$ ← for 1st max
- $\log n - 1$ ← for 2nd max

$\Rightarrow n-1+\log n-1$

$\Rightarrow n+\log n-2$

- Balanced tournment can be viewed as Divide & conquer merge sort



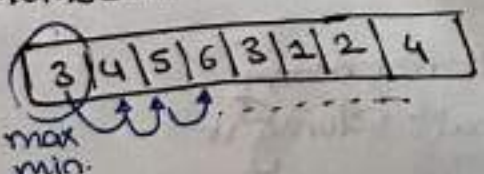
- max second max.
  - Skewed tournment
  - Balanced tournment

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

$\Rightarrow O(n)$

- **Max & Min from given array**

method 1:



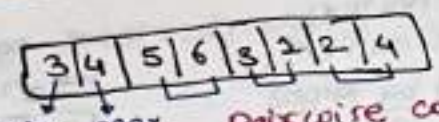

```
max = min = a[1]
for(i=2 → n)          → (n-1) time
    if(a[i] > max) max = a[i]
    if(a[i] < min) min = a[i]
```

Case:

- Best: $(n-1)$
- Avg: $\frac{3}{2}(n)$; $\frac{1}{2}(0+\frac{1}{2}(2))$ — only if condition
- worst: $2(n-1)$ if all 3 we run only else cond^n

$n-1 \leq 1.5n \leq 2(n-1)$

method 2: CLRS method
Book name



pairwise comparison

initialization
a[1] > a[2]

(min, max)
max = a[1]
min = a[2]

$\frac{n-2}{2}$ pairs
For every pairs we are doing 3 compares total compare

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

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

In all cases (n is even)

- if n is even:- $\frac{n-2}{3}$ pairs

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

$\Rightarrow \boxed{\frac{3n}{2}-2}$

- If n is odd:-

$3\left(\frac{n-1}{2}\right)$

$\frac{n-1}{2}$ pairs

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

$\Rightarrow \boxed{\frac{3n}{2}-1.5}$

we can say

$\left\lceil \frac{3n}{2}-2 \right\rceil$

$\Rightarrow \lceil \frac{3n}{2} \rceil - 2$

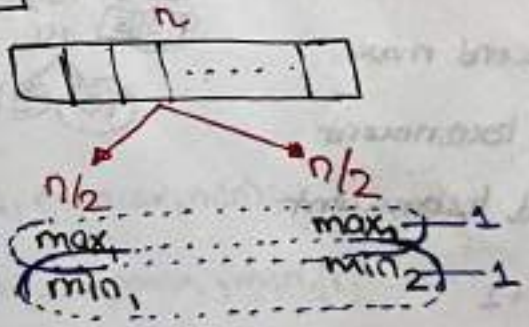
ex: let P be no. of comparisons to find min & max. using best algo.

(a) $P \le \lceil \frac{3n}{2} \rceil - 2$ (b) $P \le \lceil \frac{3n}{2} \rceil$ (c) $P \le \lfloor \frac{3n}{2} \rfloor$ (d) $P \le 1.5n$

method 3: Divide & Conquer

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

- T(n): no. of comparision
- 2: to combine both sub-problm

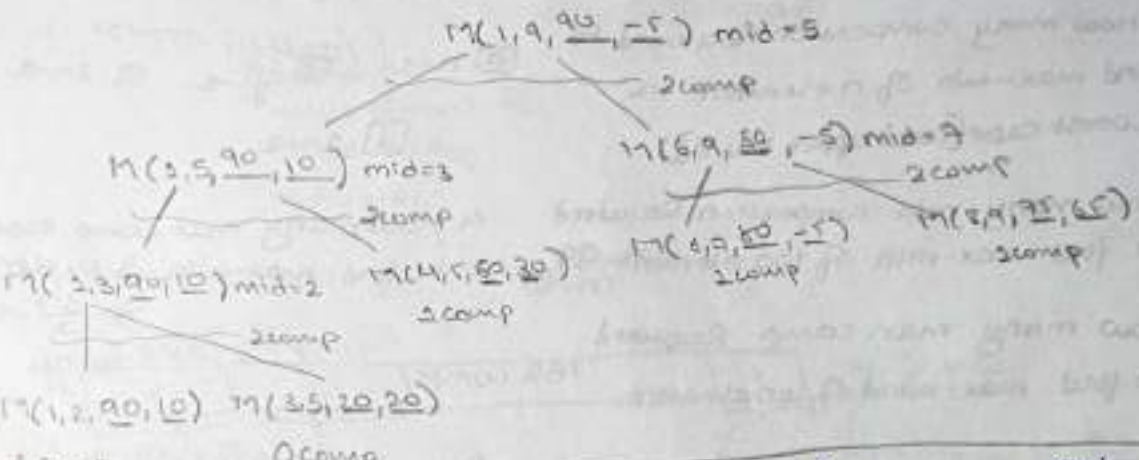


exit n is not power of 2

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 2$ } let's not worry about solving it

20

	1	2	3	4	5	6	7	8	9
9	10	90	20	60	80	-5	50	75	65



T(n): # comp. to find max, min of n element using DAC

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

? $T(n) = \frac{3n}{2} - 2$ All cases

if n is power of 2.

i.e $T(n) = 2T(n/2)+2$

$\Rightarrow T(n) = 2\left[2T(n/2)+2\right]+2$

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

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

1

:

: K time

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

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

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

$\Rightarrow 2^k T(2)+\left[2(2^k-1)\right]$

$\Rightarrow \frac{n}{2}\cdot 1+2\left(\frac{n}{2}-1\right)$

$\Rightarrow \frac{n}{2}+n-2$

$\Rightarrow \left(\frac{3n}{2}-2\right)$ comparsion.

we can also take 1 but max of the value is terminating by n=2

- when you have multiple terminating condition then choose the value which is more freq

MaxMin Algo	min Comp	Avg Comp	max Comp
Straight TC: $\Theta(n)$ SC: $\Theta(1)$	$n-1$	$\frac{3n}{2}-1.5$	$2n-2$
D And C TC: $\Theta(n)$ SC: $\Theta(\log n)$	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$

1. How many min comp. Required to find max-min of n element

~~a~~ $n-1$ b $\frac{3n}{2}-2$ c $2n-2$ d $2n+2$

2. How many comparsion Required to find max-min of n element in worst case [least upper bound]

a $n-1$ ~~b~~ $\frac{3n}{2}-2$ c $2n-2$ d $2n-2$

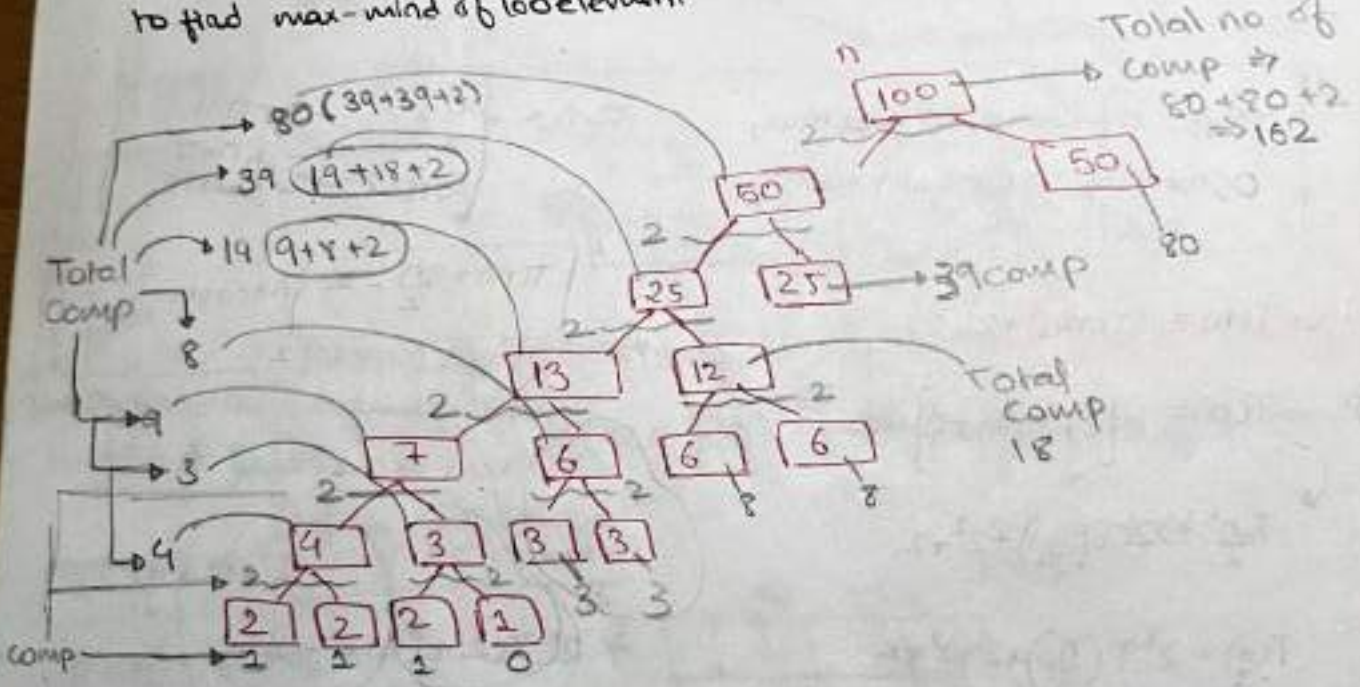
3. How many min comparsion Required to find max-min of 100 element 99 $(n-1)$

4. How many max comp. Required to find max-min of 64 element

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

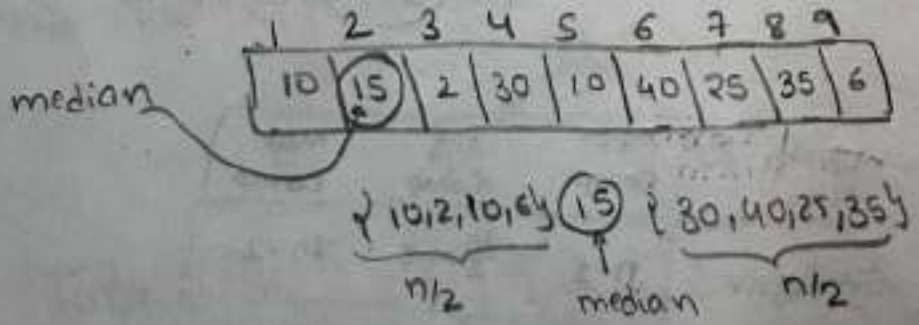
$\Rightarrow 94$

5. How many max. comp. Required to find max-mind of 100 element 162 comp.



- Algorithm to find median element of n elements

Median Element Element which is ~~greater~~ than equal to $n/2$ elements & less than equal to $n/2$ elements.



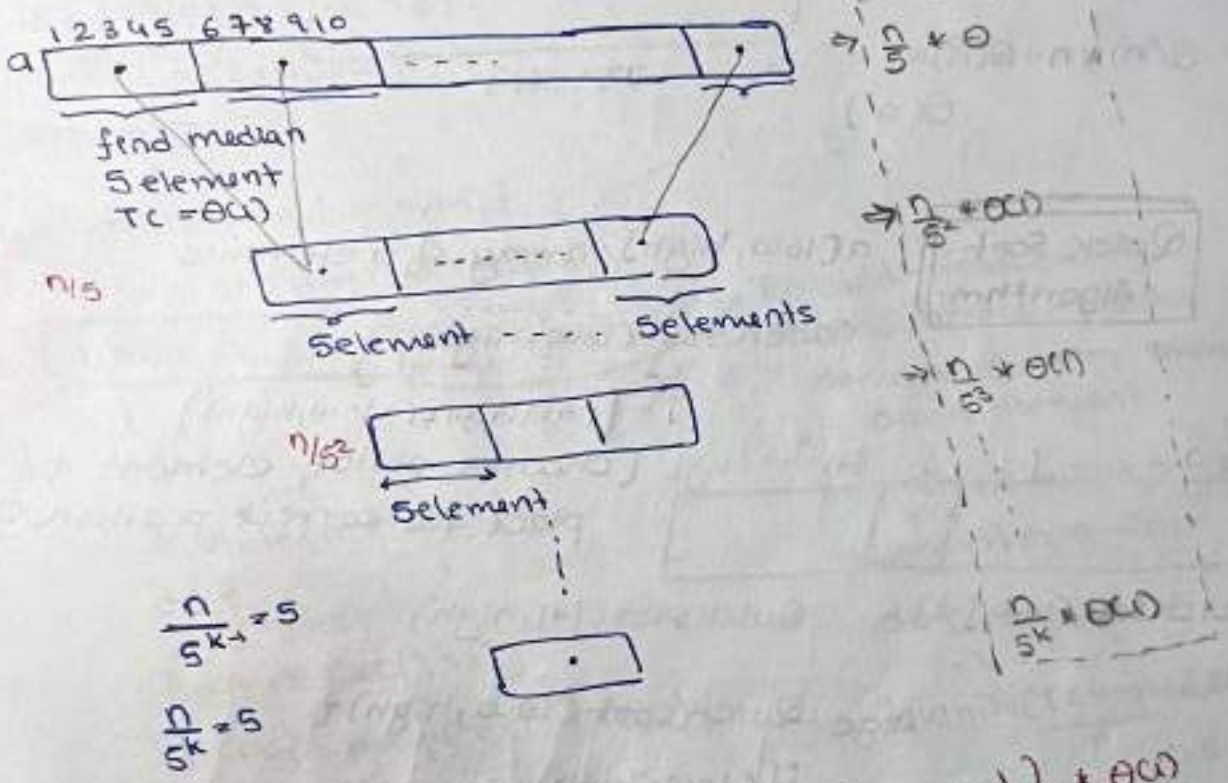
Method 1: [using sorting algo]

1. Sort array of n element.
2. return $(a[\frac{1+n}{2}])$

↑ middle element of sorted array is median

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

Method 2: [Median of Median Algo]



return

TC to find median of n element = $\theta(n)$

5 element

$T.C = n\left\{\frac{1}{5} + \frac{1}{5^2} + \cdots + \frac{1}{5^k}\right\} * \theta(1)$

$TC = \theta(n)$

note: no. of comparisons

$t < 2t < 3t$

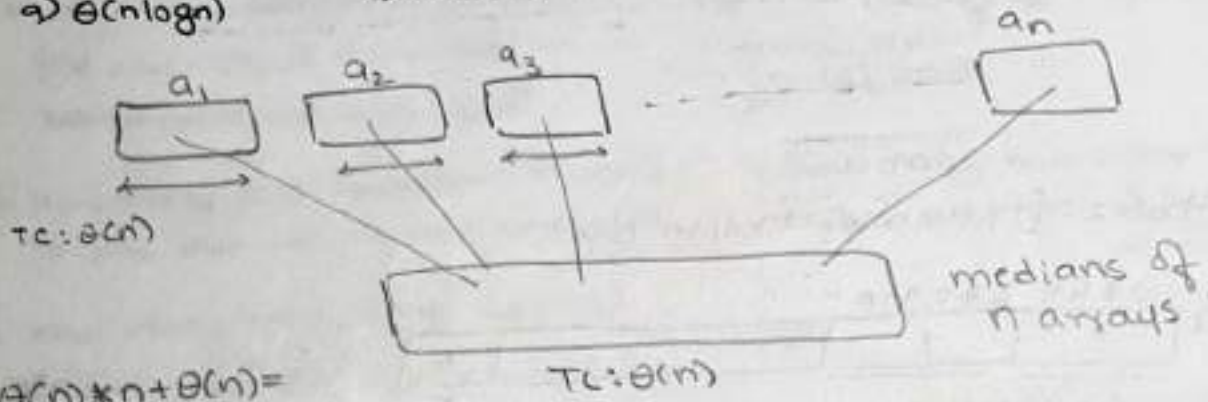
lowest upper bound

note

	Divide cost	merge cost
merge sort:	O(1)	O(n)
Quick sort:	O(n)	O(1)

* what is TC required to find median n arrays each arrays with n elements.

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



$\Theta(n) * n + \Theta(n) = \Theta(n^2)$

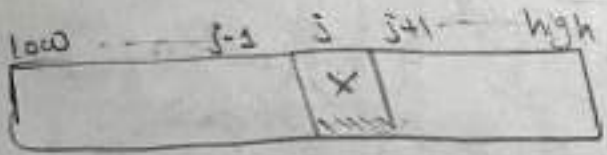
Quick Sort Algorithm

a[low, high] array of n elements

QuickSort(low, high)

j = [partition(a, low, high)]

[chooses pivot element & place to correct position (j)

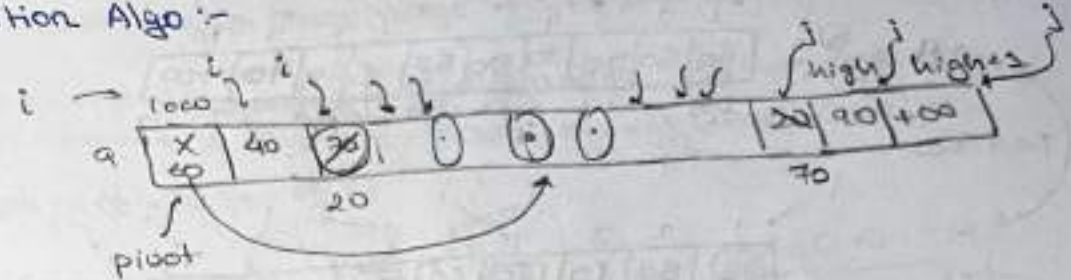


Quicksort(low, j-1) Quicksort(j+1, high)

until each subproblem ≤ 1 element

```
Algo Quicksort(low, high){
    if(low<high){
        j = partition(a, low, high);
        Quicksort(low, j-1);
        Quicksort(j+1, high);
    }
}
```

Partition Algo :-



1. Inc i until $a[i] \geq x$
2. Dec j until $a[j] \leq x$
3. If $i < j$ then swap $(a[i], a[j])$
4. Repeat 1, 2, 3 [until $i \geq j$]
5. swap $(a[j], a[low])$;
6. Return(j)

```
Algo partition (a, low, high) {
  i = low; j = high+1; x = a[low];
  a[high+1] = +∞ // main fun.
  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[low], a[j]);
  return (j);
}
```

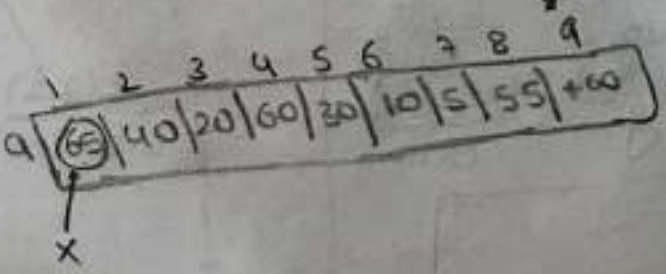
To avoid array out of bound access of pivot element greater than other element.

i.e can be avoid by using extra comp.

```
do {
  i++;
} while(i < high && a[i] < x)
```

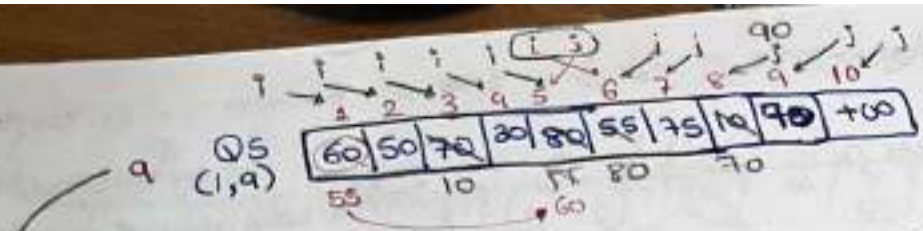
Stop's array out of bound

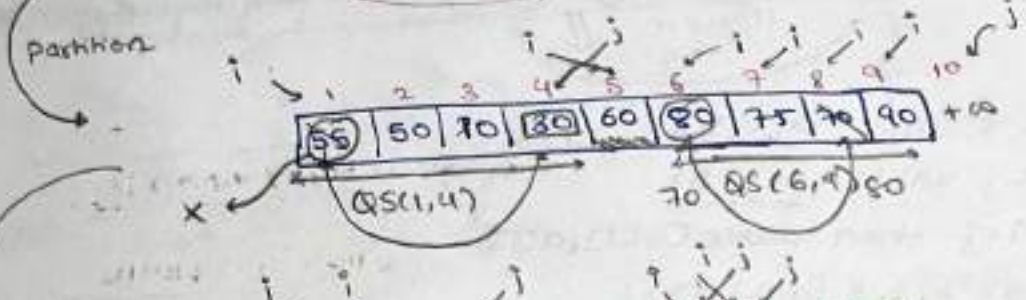
[only n extra comp. in worst case]

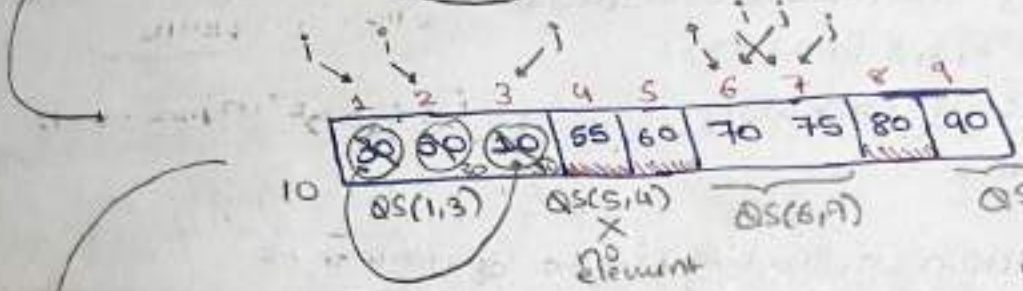


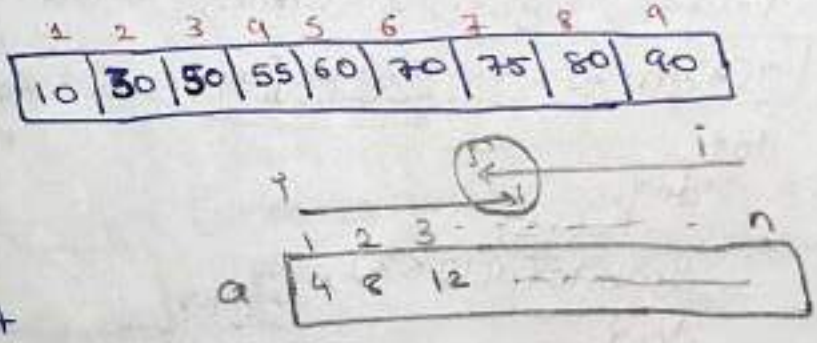
2nd do.

ex:-



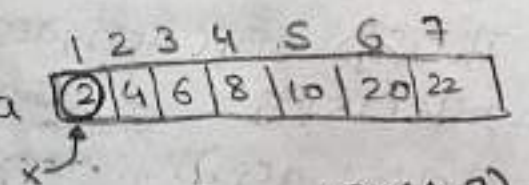


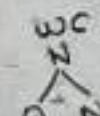




TC of partition of array of n element : $\Theta(n)$ // in all cases

TC of Quick sort

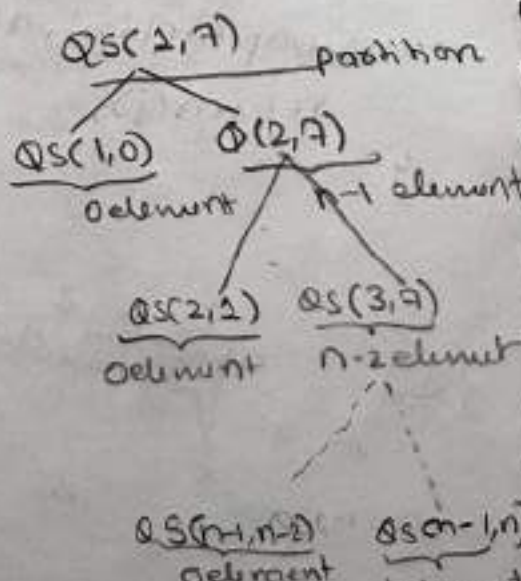




1 coorst case TC :-

- Quick sort behave worst case if input array is already in sorted order either ascending or descending.

max depth of rec of QS: $\Theta(n)$



WC TC recurrence Rel of Quick sort

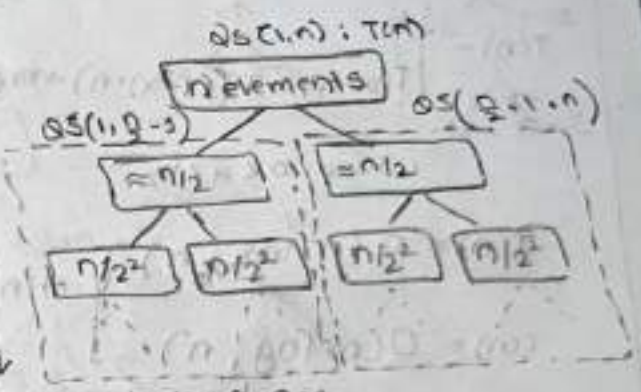
$$T(n) = \begin{cases} 1 & n \le 1 \\ T(n-1) + n & n > 1 \end{cases} \quad T(n) = \Theta(n^2)$$

- Quicksort take more time than merge sort if the array is already sorted.
- due to the unequal divide of partition such that one side is "0" element & other is "n-1" element [worst case]

[2] Best case TC :- BC

$n/2$ $n/2$

- Quick sort behave as Best Case for each partition pivot place should be placed in mid position [list should divide into 2 equal part]



min depth of Rec
$\Theta(\log n)$

BC TC reccurrence Rel of QS:

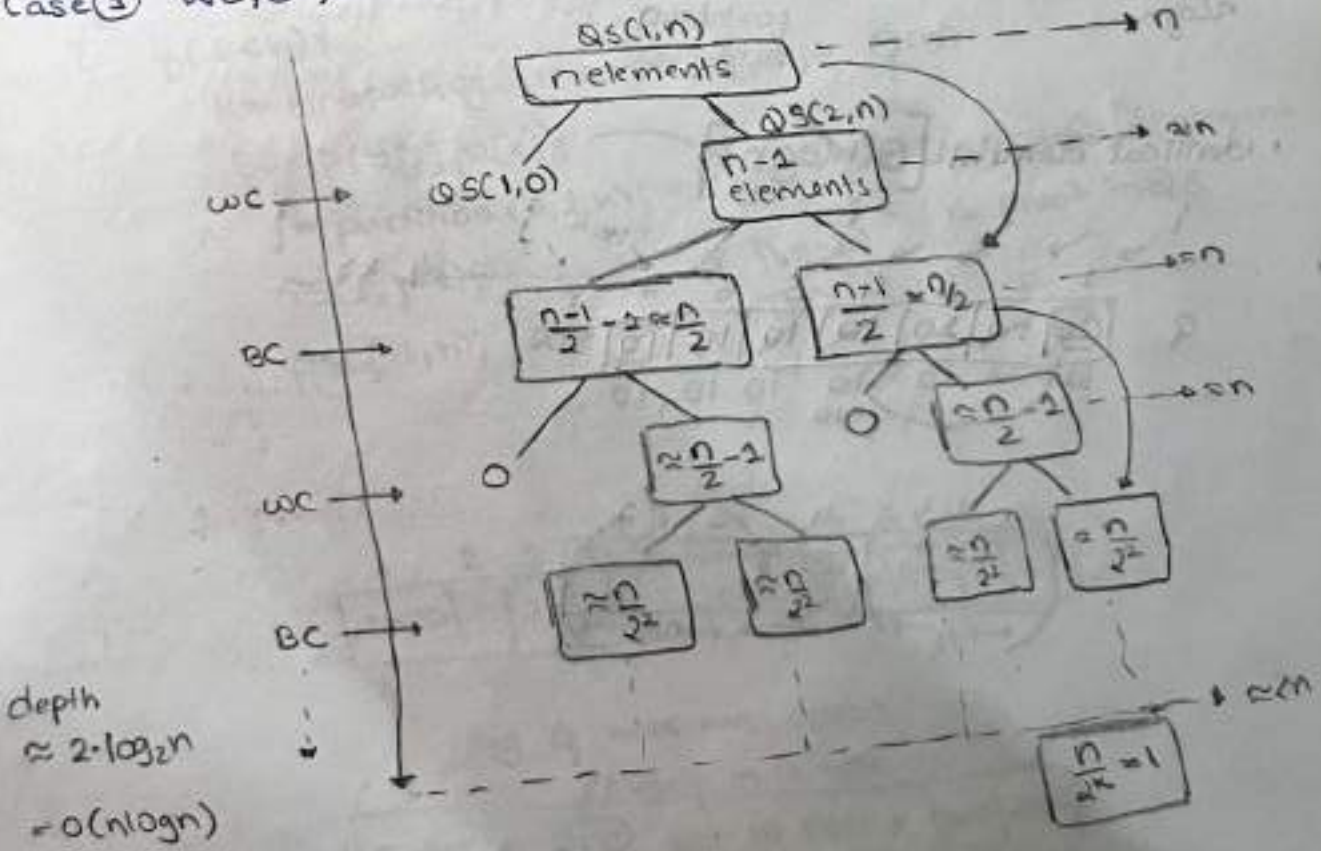
$$T(n) = \begin{cases} 1 & n \le 2 \\ 2T(n/2) + n & n > 1 \end{cases}$$

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

- if all element are identical then its Best case

[3] Aug case TC of QS:

Case (1) WC, BC, WC, BC .....



TC : $\Theta(n \log n)$

case ② for each partition of n elements is divided into

$\alpha \cdot n$ & $(1-\alpha)n$ where $0 < \alpha < 1/2$

Avg. case TC recurrence rel

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

where $0 < \alpha < 1/2$

$T(n) = \Omega(n \cdot \log_{1/\alpha} n)$ } $\Theta(n \log n)$

$T(n) = O(n \cdot \log_{\frac{1}{1-\alpha}} n)$

$T(n) = T(\frac{n}{5}) + T(\frac{4n}{5}) + n$ ($\alpha = 1/5$, $[1-x] = 4/5$)

$T(n) = T(\frac{299}{300}) + T(\frac{n}{300}) + n$

$\Theta(n \log n)$

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

after the combining if it is n then it's best case partition

& the cost of partition should be n

not every form of this is nlogn $\left[T(\frac{n}{8}) + T(\frac{4n}{5}) + n\right]$ ✗ not Avg. case of QS

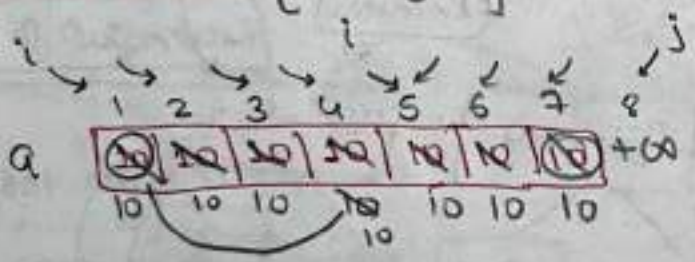
OR $\left[T(\frac{n}{3}) + T(\frac{2n}{3}) + n^2\right]$

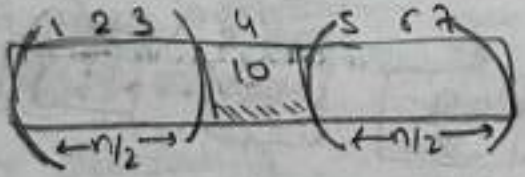
$T(\frac{n}{3}) + T(\frac{2n}{3})$ cost is n

n^2 but partition cost is n^2

still not nlogn

- identical element $[\Theta(n \log n)]$

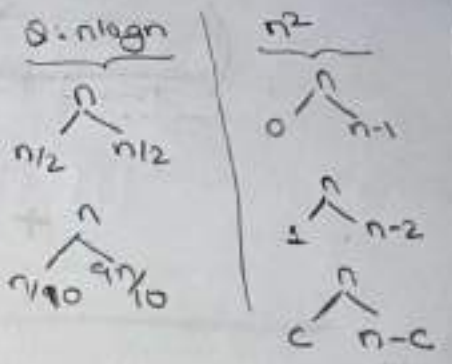




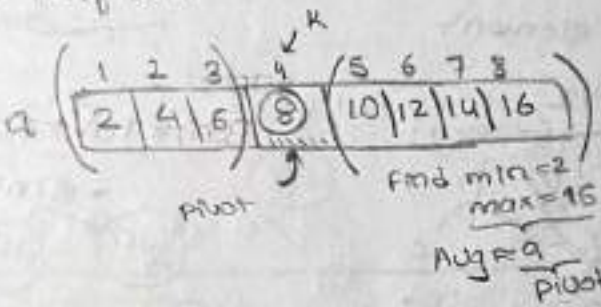
• Pivot selection of Quick sort:

① First/Last element is pivot
worst case TC of QS = $\Theta(n^2)$
// if ip array in ASC/desc order

② Avg of min & max element in pivot
worst case TC of QS $\Theta(n^2)$

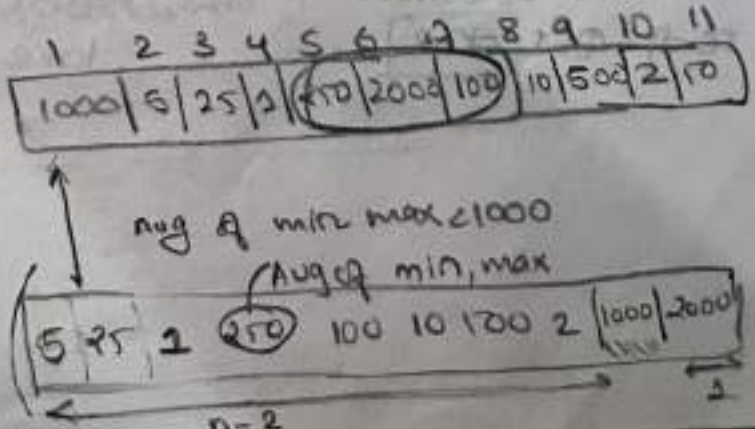


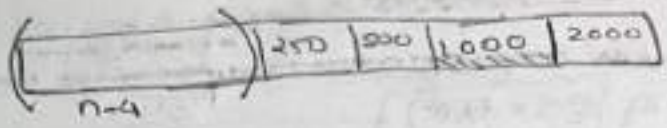
- the only case when it exceed n^2 is when partition itself is n^2



Algo QS(l,h) ⇒ T(n): wc TC

```
if(l<h){
  k = FindposAvgofMinMax(a[],h);→ Θ(n)
  swap(a[l],a[k]);
  j = partition(a,l,h);  →Θ(n)   b/c we assume 1st element is pivot in QS.
  QS(l,j-1);             →T(n-2)
  QS(j+1,h);             →T(1)
}
}
```





WC TC Rec. Rel.

$T(n) = T(n-2) + c \cdot n$

$\Rightarrow \Theta(n^2)$

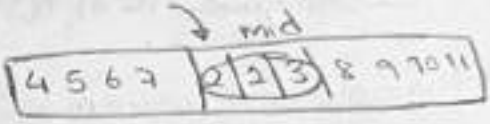
---

③ Pivot is Avg of a[mid], a[mid+1], a[mid-1]

a[low,high] array n element

mid is (low + high)/2

WC TC of QS : $\boxed{\Theta(n^2)}$ //



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

WC:

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

$= \Theta(n^2)$

? ④ median element of array is pivot element

WC TC of QS : $\Theta(n \log n)$

$T(n) = 2T(n/2) + \boxed{\Theta(n) + \Theta(n)}$

- $\Theta(n)$: find median
- $\Theta(n)$: partition

- Increasing the computation power

- Median pivot selection is not efficient b/c extra Cost of median computation for each partition increase Best case & Avg case execution of QS

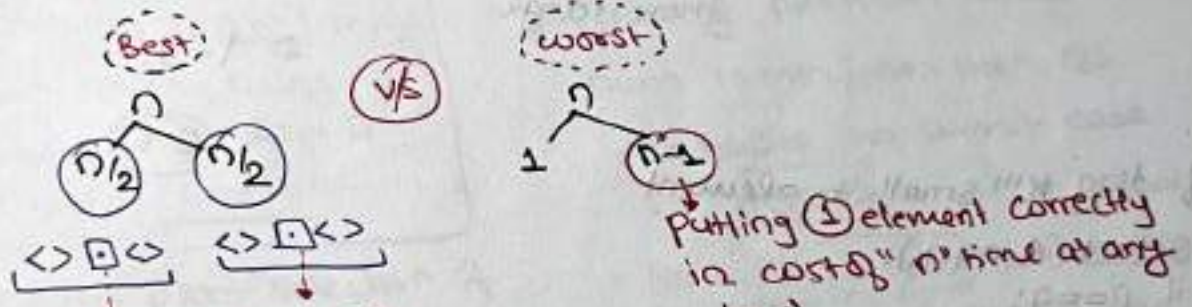
---

⑤ Random element from a[low, high] array if n element selection as pivot WC TC of QS: $\boxed{\Theta(n^2)}$

Random function take constant time $\Theta(1)$

- Partition Algo:-

```
partition (a, low, high){
  i=low+1, j=high;
  pivot=a[low]
  while(i≤j){
      while(i≤j and a[i]≤pivot) i++;
      while(i≤j and a[j]>pivot) j--;
      if(i≤j){
         swap(a[i], a[j]), i++, j--;
      }
  }
  swap(a[low], a[j]);
  return j;
}
```



Putting (1) element correctly in cost of "n" time at any level

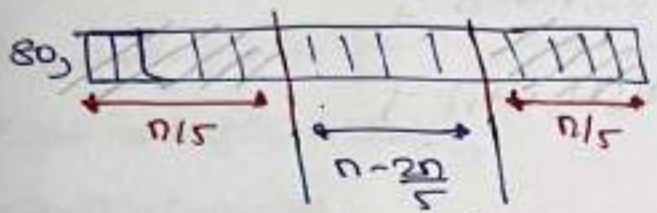
In 'n' time you are able to sort (2) item/element in cost of 'n'

↳ In next level you are going to put 4 element in cost of "n"

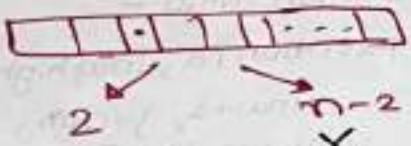
- worst case of selecting pivot in quick sort
  - first element
  - last element
  - middle element
  - choose min or max element
- Best case & Avg. case of quick sort to take pivot
  - median

ex: what is the probability that with a randomly chosen pivot element the partition subroutine produces a split in which the size of the smaller of the 2 subarray is $\geq \frac{n}{5}$?

let $\alpha = \frac{1}{5}$



pivot can be only selected from these range



not allowed as.

$n \geq 100$

$\Rightarrow \frac{n}{5} \geq 20$ → should be greater than (20)

So, probability

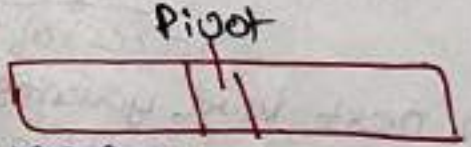
$\Rightarrow \frac{n - \frac{2n}{5}}{n}$ ← favourable case / ← total case

$\Rightarrow 1 - \frac{2}{5}$

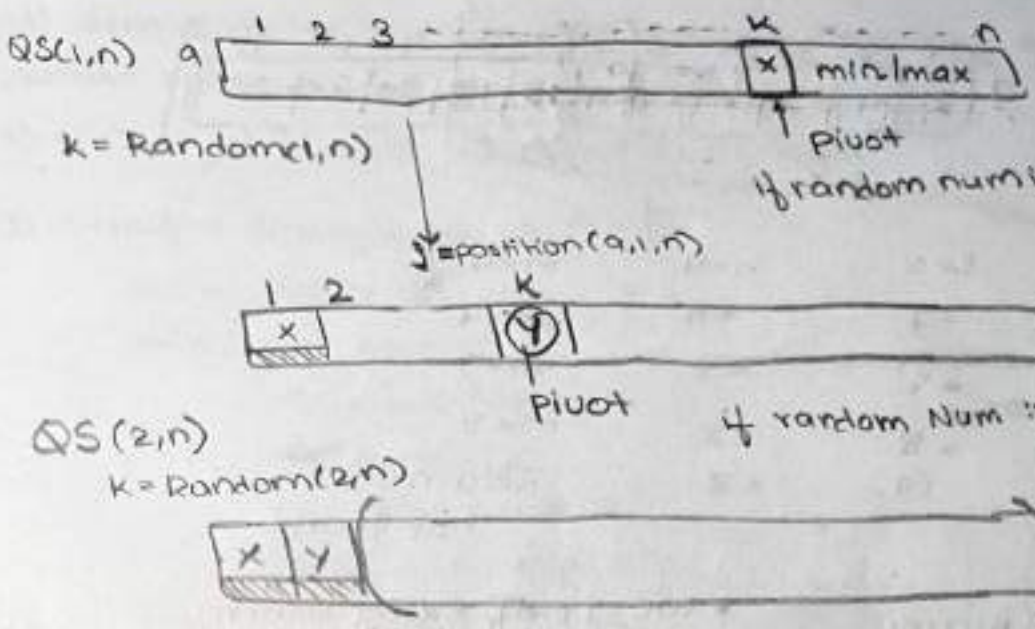
$\Rightarrow (0.6)$ //

ex finding k^{th} smallest element

```
Select(A,p,q,k):
if p==q:
   return A[0]
m = partition(A,p,q) } median of medians
                        O(n)
if m == k
    return a[m]
if k<m
    return select(A,p,m-1,k)
if k>m
    return select(A,m+1,q,k-m)
```



if k^{th} min is in left side then go only in Lts else vise versa



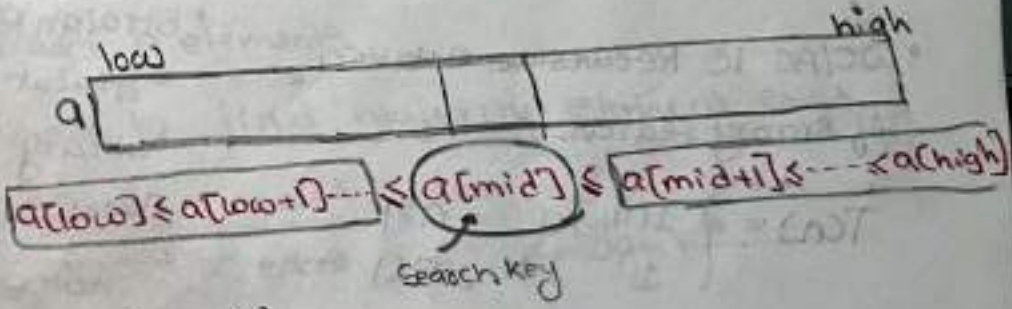
- probability of $O(n^2)$ TC of Quick Sort by using random pivot set is atmost close to 0.

- if every partition random num is min/max then QS behaves as worst case TC : $\Theta(n^2)$

- Random pivot selection of QS is efficient implementation of QS.

- QS is an inplace sorting Algo but not Stable Sorting Algo

Binary Search Algo [a and c]

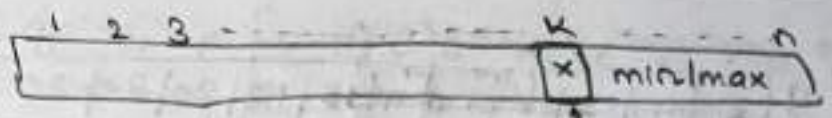


- array of n sorted element

```
Algo straight BinarySearch(a,n,x){
                                  // a[1...n] array
    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);
    }
    return(-1) // X not found in array
}
```

max log n time

QS(1,n) a

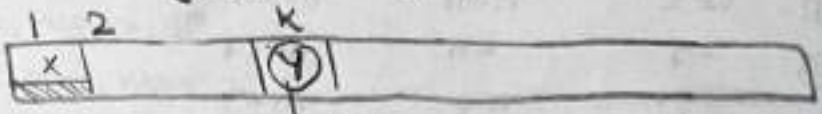


k = Random(1,n)

Pivot

if random num is min/max

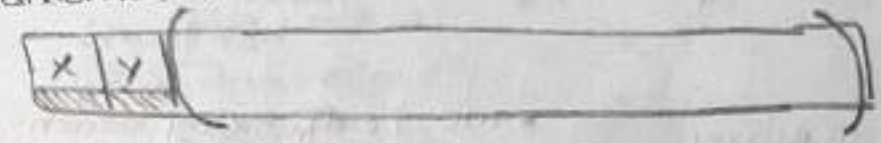
j = position(a,1,n)



Pivot

QS(2,n) if random Num is min/max

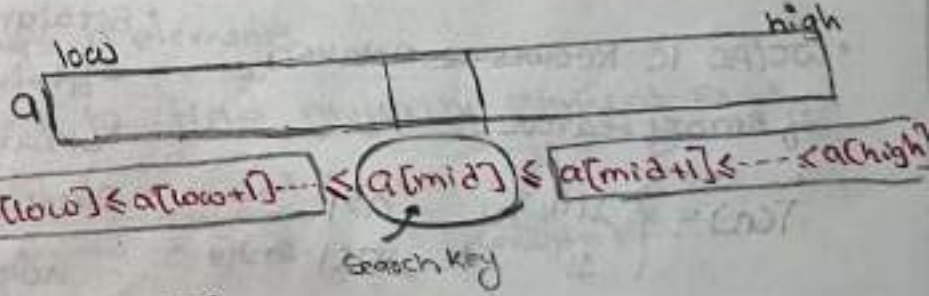
k = Random(2,n)



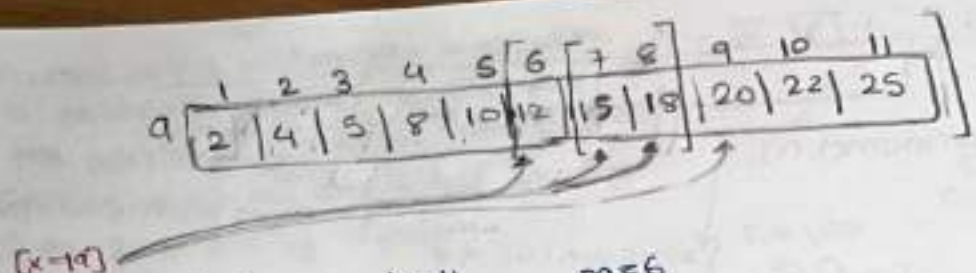
- probability of $\Omega(n^2)$ TC of Quick Sort by using random pivot set is atmost close to 0
- if every partition random num is min/max than QS behaves as worst case TC : $\Theta(n^2)$
- Random pivot selection of QS is a efficient implementation of QS.
- QS is an inplace sorting Algo but not stable sorting Algo

Binary Search Algo [a and c]

- array of n sorted element



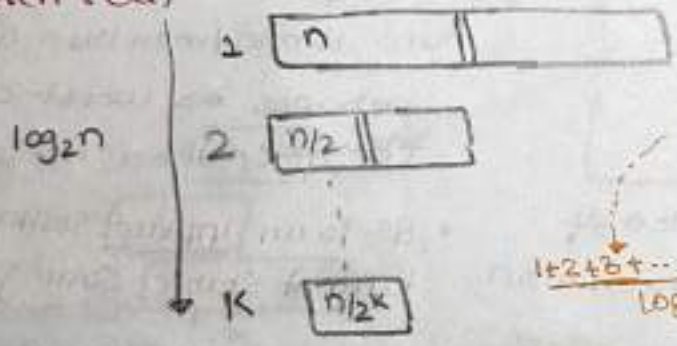

```
Algo straight BinarySearch(a,n,x){
        low=1; high=n          // a[1...n] array
        while(low ≤ high){
max       mid=(low+high)/2
logn      if (x < a[mid]){ high=mid-1; }
time      else if(x>a[mid]){ low= mid+1; }
          else return(mid);
        }
        return(-1) // x not found in array
}
```



[x=15]

	l	h	m
①	l=1	h=11	m=6
②	=7	=11	m=9
③	=7	=8	m=7
④	=8	=8	m=8
	(9	=8	Return (-1) not found

- BC TC of Binary Search : $\Theta(1)$
- WC TC of Binary Search : $\Theta(\log n)$



- AC TC of Binarch : $\Theta(\log n)$
- SC : $\Theta(1)$

$$\frac{1+2+3+\cdots+\log n}{\log n} = \frac{\log n(\log n+1)}{2*\log n} = \frac{1+\log_2 n}{2}$$

$\Rightarrow \Theta(\log n)$

- WC/AC TC Recursive Relation of Binary search.

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

$= \Theta(\log n)$

- Straigh Binary search more efficient than recursive Binary Search b/c : no Overhead of funcn call & no extra Stack space Required
- Every Reursive Relation is Divide & Conquer
- not every non Recursive Relation is not Divid & Conquer

- give a tree is BST that it's inorder will produce a increasin sequence

Qus) what is woc TC to return first occ of searching element position in sorted array of n elements?

a) $\Theta(1)$ b) $\Theta(\log n)$ c) $\Theta(n)$ d) $\Theta(n \log n)$

Algo straightBinarysearch(a, n, x){ // o.t.c

```
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(mid>1 && a[mid-1]==a[mid])
        high=mid-1;
    else
        return(mid);
}return(-1);
}
```

Qus) Majority element is the element which repeat more than $n/2$ times in arrays n elements

(i) what is TC required to find majority element exist or not in array of sorted element

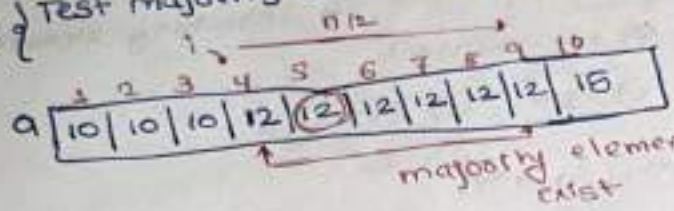
a) $\Theta(1)$ b) $\Theta(\log n)$ c) $\Theta(n)$ d) $\Theta(n \log n)$

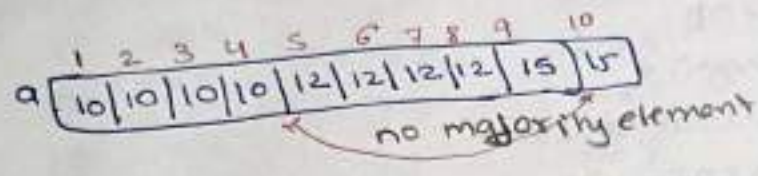
(ii) TC test majority element exist or not in array of n element whose range (1 to n) or (0 to n) with n integers

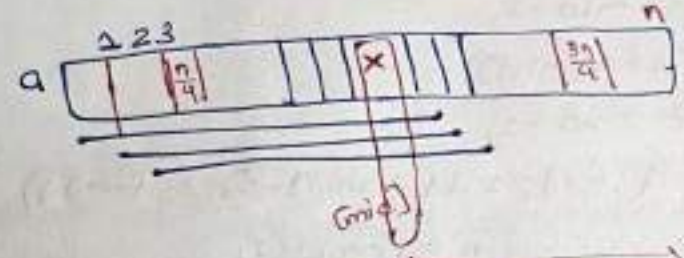
a) $\Theta(1)$ b) $\Theta(\log n)$ c) $\Theta(n)$ d) $\Theta(n \log n)$

(iii) what is TC required to test majority element exist or not in array of elements $\Theta(n)$

c) Test majority element exist/not in sorted array of n element







Candidate for majority only middle element of array.

- a[1...n] given sorted array $x = a[\frac{1+n}{2}]$ // middle element
  Candidate for majority.

  i = firstOccPoss_BS(a, n, x)

  if (a[i + n/2] == x) print ("majority exists")

  else print (no majority element)

(ii) Majority element exists/not in array of n element (sort) where range [0 to n]

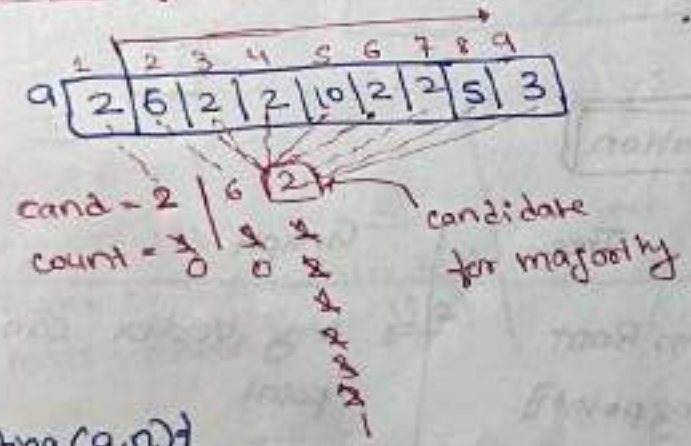
TC: $\theta(n)$
SC: $\theta(n)$


```
for(i=0;i<n;i++){ c[i]=0;}
for(i=0;i<n;i++){ c[a[i]]++;}
for(i=1;i<n;i++){ if(c[i]>n/2)
                    Print(majority element exist);
}
```

(iii) array of n element test majority exist or not

Moores Voting Algo:- Return candidates for the majority

- not used to calculate majority (only used to check whether it exist or not)



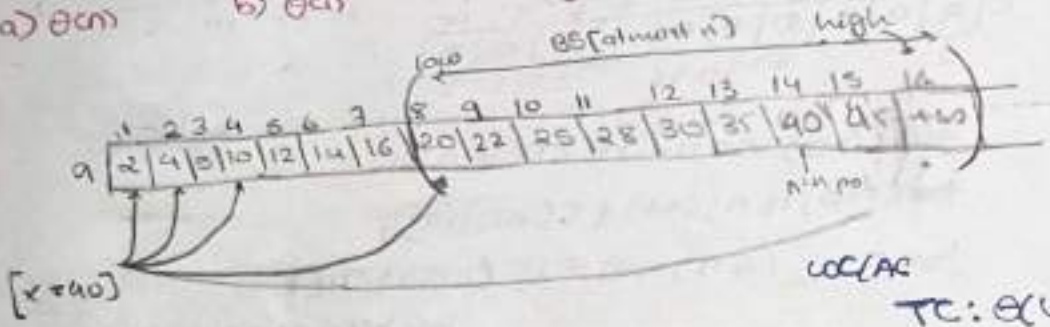

```
Algo MooresVoting(a,n){
    //a[i] to a[n] array
    cand = a[i]; count = 1;
    for(i=2;i<n;i++){
        if(cand==a[i]) count++;
        else{
            count--;
            if(count==0){cand=a[i];
                         count=1;
            }
        }
    }
    return(cand);
}
```

TC $\theta(n)$

SC $\theta(1)$

Ques] what is TC required to search element in sorted array of array size unknown [infinite size array] where remaining element of array is (+∞).

a) $\Theta(n)$ b) $\Theta(1)$ c) $\Theta(\log n)$ d) $\Theta(n \log n)$



WC/AC

TC: $\Theta(\log n)$

SC: $\Theta(1)$

```
i=1
while(A[i] < x){i = 2i;}
  l = i/2; h = i;
call BS(l, h, x);
```

WC: $\Theta(\log n)$ + $\Theta(\log n)$

TC $\Theta(\log n)$

Tree Representation

Tree	Graph.
1. Always start from Root [∴ single starting point]	1. Any vertex can be starting point
2. Connected & Acyclic & Directed	2. may not be connected, Acyclic Directed.
3. for n vertex tree #f edges exactly (n-1)	3. for n vertices simple undirected graph #f edges ∈ 0 to $\frac{n(n-1)}{2}$

- Simple graph
  Graph with no self loop & no parallel edges.

#f of edges in simple, undirected
Complete connected graph with "n" vertices are ${}^nC_2 \Rightarrow \left[\frac{n(n-1)}{2}\right]$ edges

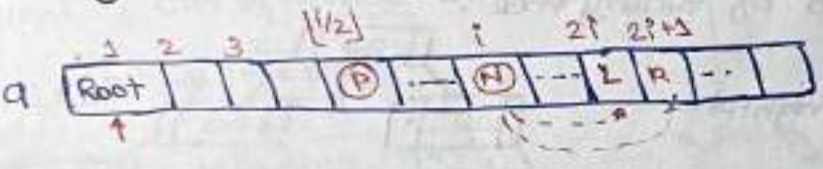
#f of edges in simple, directed
complete connected graph with "n" vertices are ${}^nC_1 \Rightarrow [n(n-1)]$ edges

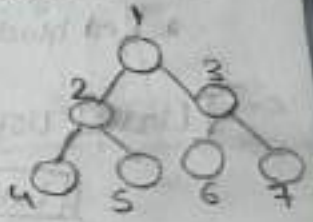
#f simple undirected graph with n vertices. $2^{\frac{n(n-1)}{2}}$

#f simple directed graph with n vertices, $2^{n(n-1)}$

Tree Representation
1. Array Representations
2. Linked list Representations

1] Array Rep. of Binary tree





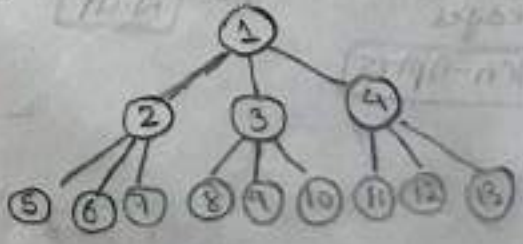
1. Root index 1
2. If Node N at index i
   then Left child of N index 2i
   Right child of N index 2i+1
   Parent ~~of root~~ of N index ⌊i/2⌋

1. Root index 0
2. If node n at index i
   then Left child 2i+1
   Right child 2i+2
   Parent — ⌊i-1/2⌋

2] Array Rep. of Ternary tree:-

If Root node index 1
(and) node N index is i

then index of
1st child of N 3i-1
2nd child of N 3i
3rd child of N 3i+1
Parent of N $\left[\frac{i+1}{3}\right]$



- If tree with n nodes their array index should be continous integers from 1 to n then array rep is preferred to store tree.

Otherwise, linkedlist rep. preferred [b/c in array there is wastage of memory if it is not continous allocation]

- * complete Binary tree
- * full Binary tree
- * min heap
- * max heap

→ array Rep. preferred.

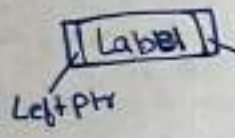
- * Binary search tree
- * strict Binary tree
- * AVL tree (it's BST)
- * Red black tree

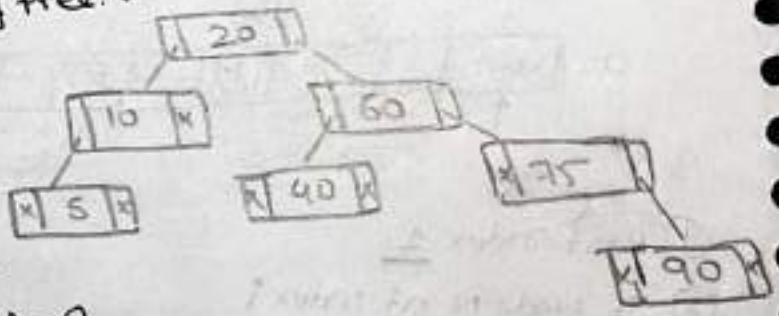
→ Linked list Rep.

Deletion in BST
if ② child then it will be replaced by inorder Predecessor & Successor.

Deletion in AVL Tree
if ① or ② child then replace by inorder precedecessor

2] Linked List Rep. of Binary tree :-





How many null ptr. in Binary tree Linked list rep. of n nodes?

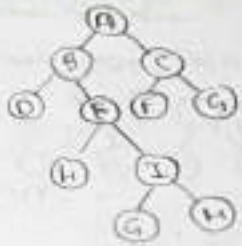
n nodes
[2n ptr]

for n-1 edges used (n-1) ptrs

(n+1) ptrs are pointed Null

- Strict Binary tree

Each node of the tree either 0 or 2 Child



Internal node : $\lceil n/2 \rceil$

Leaf node : $\lceil n/2 \rceil$

n: # of nodes

- Number node's in CBT of height h is $[2^h$ to $2^{h+1}-1]$
  min max

- height of CBT of n node $= \Theta(\log_2 n)$

$n = 2^h \Rightarrow \boxed{h = \log_2 n}$

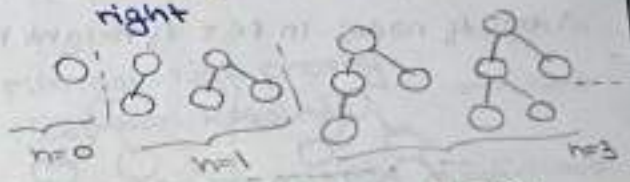
$n = 2^{h+1}-1 \Rightarrow \boxed{h = \log\left\{\frac{n+1}{2}\right\}}$

OR

$\boxed{h = \log_2(n+1)-1}$

- Complete Binary tree :- [CBT]

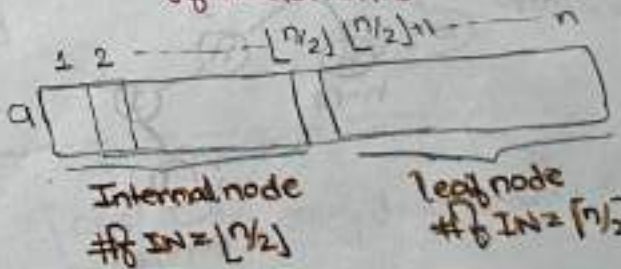
CBT of 'L' level with L-1 level full binary tree & last level node filled from left to right

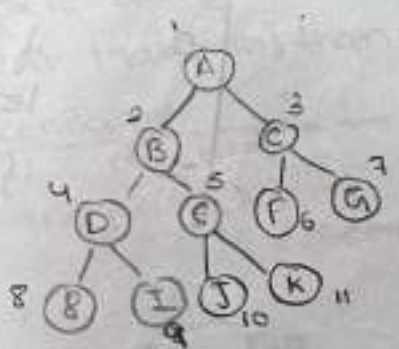


height(h)	min node	max node
0	1	1
1	2	3
2	4	7
3	8	15
...	...	...
h	2^h	$2^{h+1}-1$

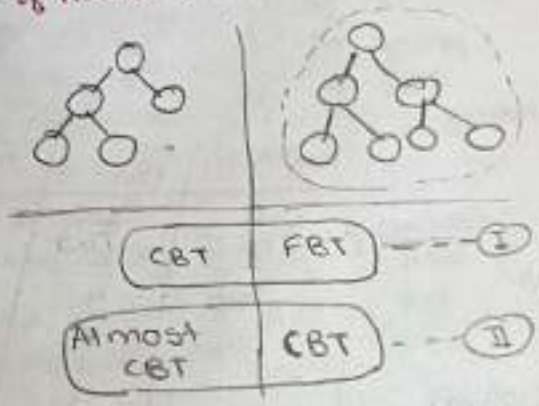
- Array Rep. used to Store CBT

CBT of n nodes are array of n elements

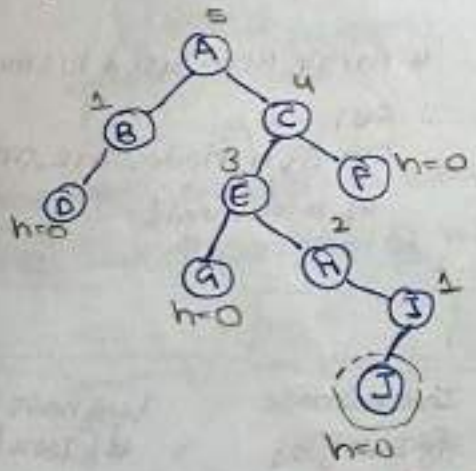




- Full Binary tree [FBT]
- Every node must be 0 or 2 child
  and
- All leaf node must at same level
- no. of nodes in FBT of height h: $2^{h+1}-1$



- Height of Binary tree .



$h(\text{Node } N) = \begin{cases} 1 + \max\{h(\text{left subtree of } N), h(\text{right subtree of } N)\} \\ 0 \quad \text{if } N \text{ is leaf node} \\ -1 \quad \text{if null node} \end{cases}$

- height of tree : height of Root node.
- height of the leaf node is '0'

Priority Queue DS DS (Data Structure) should support efficient way to perform

- 1. Repeated del min / del max
- 2. Repeated Insertion
- 3. Suppor to Store duplicate value
- 4. Key Inc / Dec [Repeated]

20

• Best DS for priority queue is Binary heap.

[max heap / min heap]

↓ Repeated Del. min — ↓ Repeated Del. max

A	B	C	D	E
20	10	15	20	40

F	G	H
25	10	35

5

Preferred minheap

* Min val: high priority [Del min Required]

* Max val: high priority [Del max Required] — maxheap

• Dictionary DS:-

DS should support efficient to perform

① Repeat searching

② Distinct key

③ Insertion / Deletion

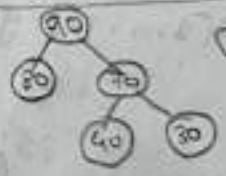
Best DS for Dictionary Implementation.

① Balanced Search tree [AVL / Red black / B tree]

② Hashing.

• Max [min] Heap:

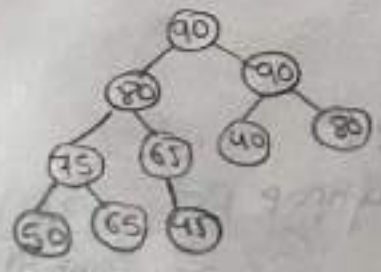
Complete Binary tree with every parent Max [min] than all descendants.



[not max heap] not CBT

Max heap

[CBT and parent ≥ every descendant]



Min heap

[CBT and parent ≤ every descendant]

	1	2	3	4	5	6	7	8	9	10
a	90	85	90	75	65	40	80	60	66	45

Max heap array

- Insert element into max heap:-

// a[1... n-1] already in max heap

Insert nth element into max heap

WC TC / AC TC: $\Theta(\log_2 n)$

BC TC: $\Theta(1)$

[max $\log_2 n$ comp]

array shift required

swap's

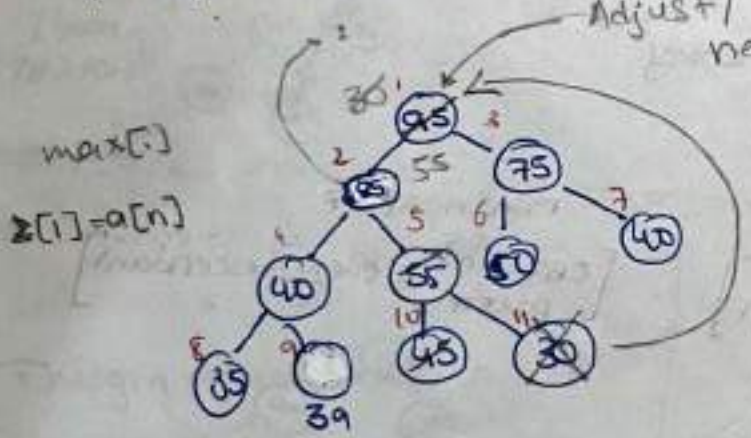
Algo Insert_Maxheap(a,n)

```
// a[1.... n-1] in max heap
// Insert a[n] into max heap
item= a[n] // inserting element
i= n;
while(i>1 && a[i/2] < item){
        a[i]=a[i/2];
        i=i/2;
}
a[i]=item
}
```

- Delename Name from max heap

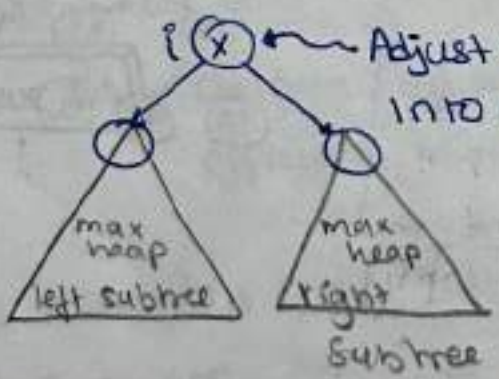
// a[1.... n] given max heap array.

- you can't delete any element you can only delete (2) element either the top most root or the bottom most right side element



- after removing root it will be replace with the right most bottom element.

Adj fun :-

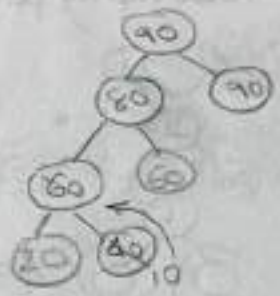


TC = Θ(height of a[i])

// a[i] Adjust into max heap

```
Algo DelMaxheap(a,n){                                   WC/AC TC: O(logn)
    // a[1..n] given max heap array
        max = a[1]
        a[1]=a[n]                                  O(logn)
        Adjust(a,1,n-1); // Adjust Root into max heap
        Return (max);
}
```

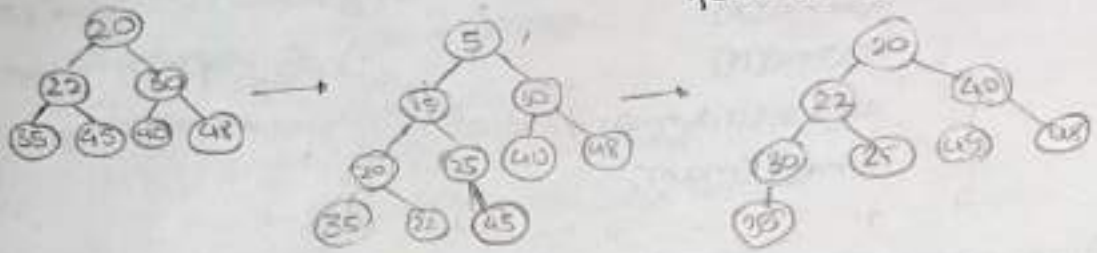
```
Algo adjust(a,i,n){ ..... .....                  TC of Adjust fun = θ(height of a(i))
    //Adjust a[i] into max heap
    //Left subtree & Right subtree
    of a[i] is already in max heap
    j=2i , item=a[i];
    while(j≤n){
        if(j<n && a[j]<a[j+1]) j++;
        if(a[j]≤item) break;
        a[j/2]=a[j]; j=2j;
    }
    a[j/2]=item
}
```



Out of Bound condition

Qus| given max heap array . what is resulted max heap array after insertion elements 98, 85, 75 & followed by two times delete max.
95, 80, 75, 40, 55, 50, 40

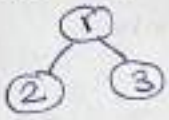
Qus] given min heap array : 20, 22, 40, 30, 35, 45, 48
Insert key: 5, 15, 25 into min heap
followed by 2 delete



*Qus] {1, 2, 3, 4, 5, 6, 7}
How many possible min heap?

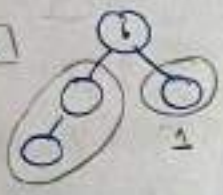
n=2

n=3



→ 2

n=4

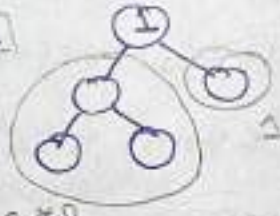


→ 3

$^3C_2 * 1$

[1 2 3 4]

n=5

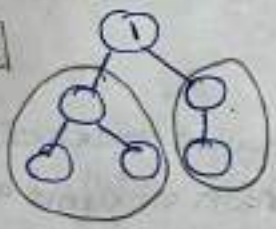


→ 8

$^4C_3 * 2$

[1 2 3 4 5]

n=6

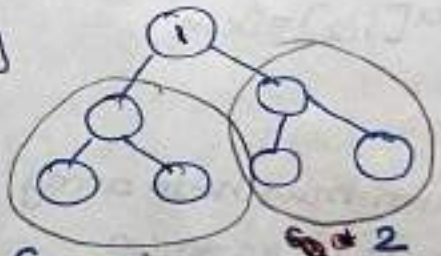


$1 * 2 * {}^5C_3$

→ 20

[1, 2, 3, 4, 5, 6]

n=7



$^6C_3 * 2$

→ 80

[1, 2, 3, 4, 5, 6, 7]

[n=15]
# of min heap?

general formal: $T(n) = {}^{n-1}C_1 * T(L) * T(R)$

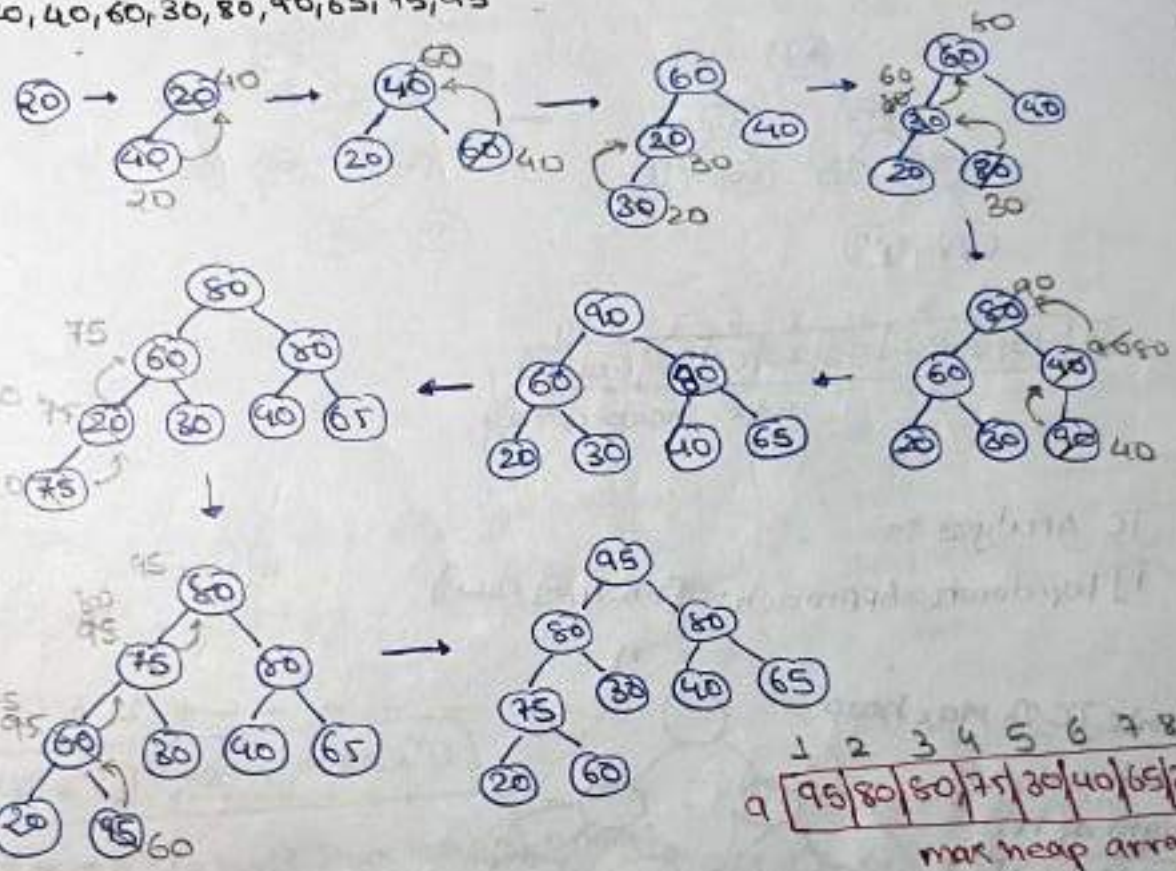
- Construction of Maxheap:-

[1] Top down approach (one by one :- O(nlogn)) [2] bottom-up approach O(n)

1] top down Approach :- [Insert a[2], a[3], ----a[n]]

ex: Construct max heap for given into maxheap sequence of element

20, 40, 60, 30, 80, 90, 65, 75, 95

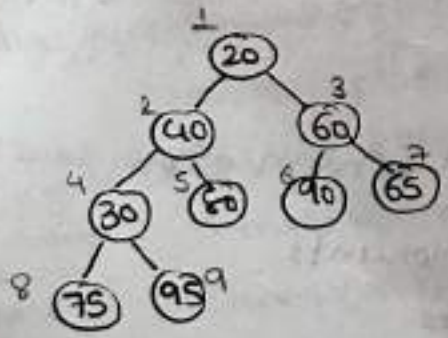


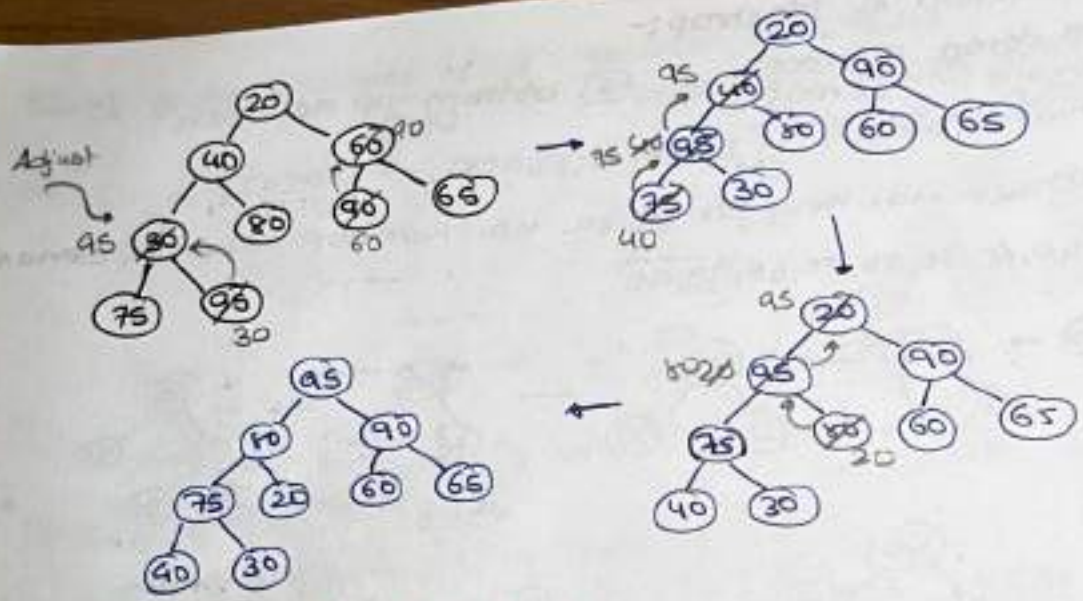
	1	2	3	4	5	6	7	8	9
a	95	80	80	75	30	40	65	20	60

maxheap array

2] Bottom-up approach [Heapify / Adjust method]

- Adjust all Internal node of CBT format of sequence of keys into maxheap.





1	2	3	4	5	6	7	8	9
95	80	90	75	20	60	65	40	30

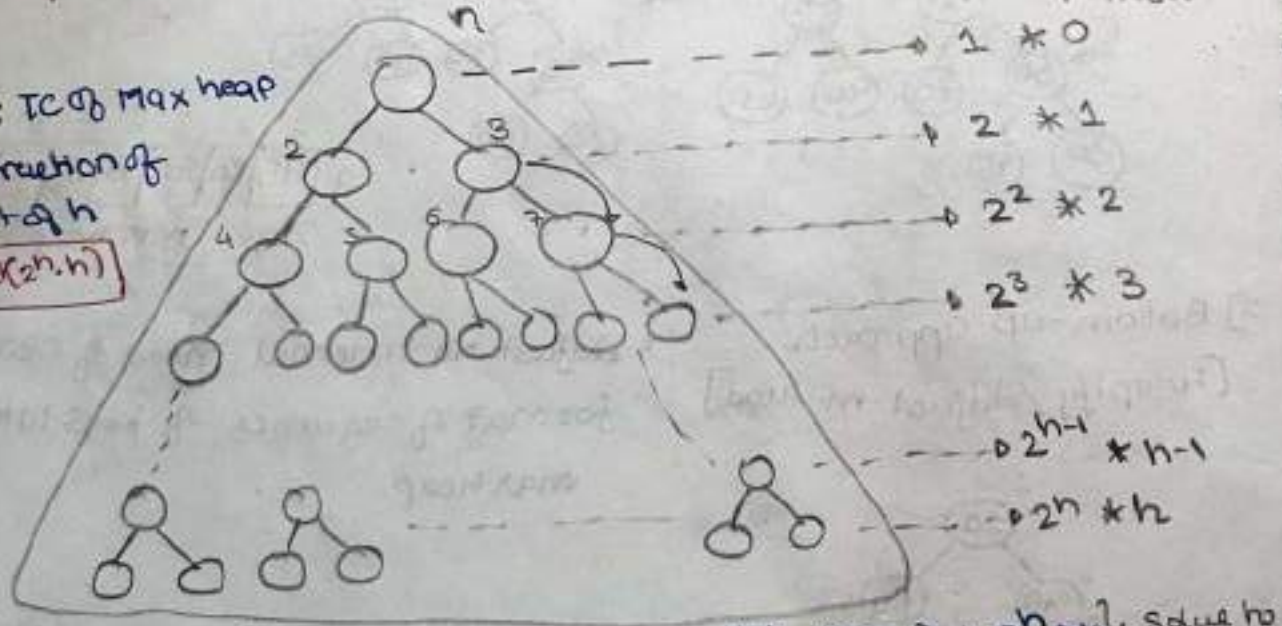
a

max. heap array.

TC Analysis :-

1] top down Approach : [WC / Avg case]

#insertion * #of swap (or array shift) for one insertion



- T(h): TC of max heap construction of height h

$T(h) = \Theta(2^h \cdot h)$

$T(h) = 2*1 + 2^2*2 + \dots + 2^{h-1}*(h-1) + 2^h*h$ solve to get max. no. of swap required

- T(n): TC of max heap construction for n elements

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

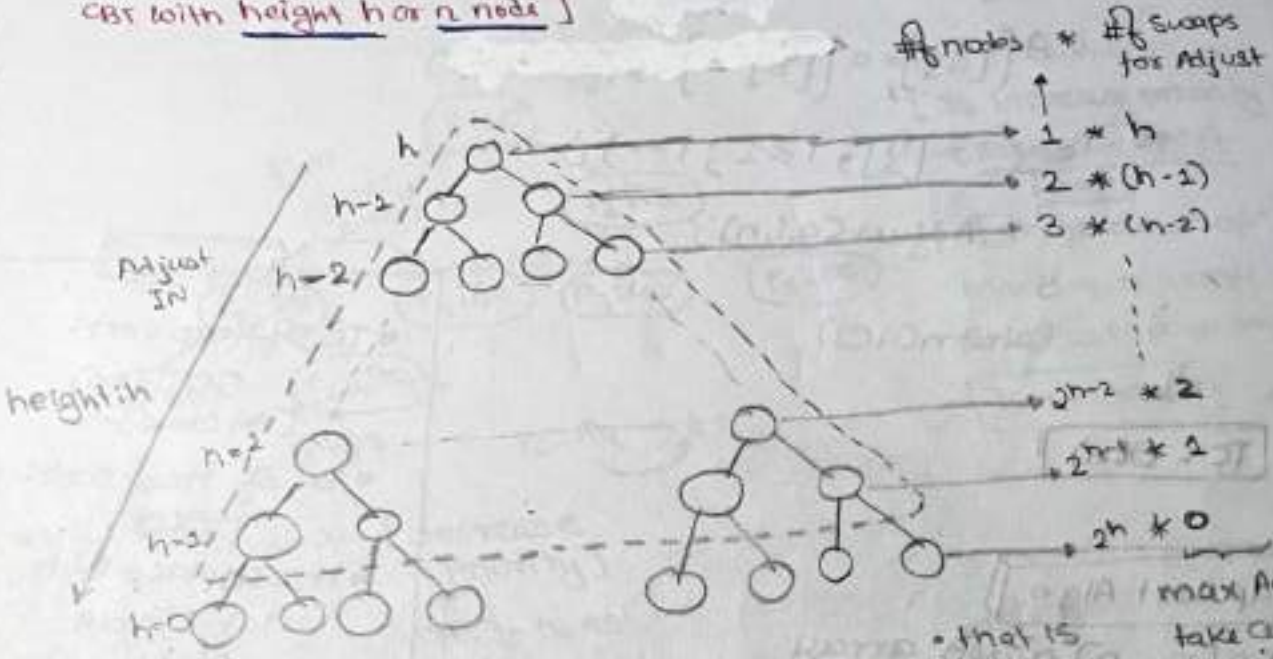
CBT of h : $2^{h+1}-1$ nodes

$n = 2^{h+1}-1 \Rightarrow 2^h = \frac{n+1}{2} = \Theta(n)$

$h = \log_2\left(\frac{n+1}{2}\right) = \Theta(\log n)$

2] Bottom-up Construction [Heapify method]

[Adjust all internal nodes of CBT with height h or n nodes]



- that is the reason Bottom up is better than top down

max Adjust take can not have it only few take logn cost (h)

- T(n): TC to build max heap for height h

$T(n) = 2^{h-1} * 1 + 2^{h-2} * 2 + \cdots + 2^2(h-2) + 2(h-1) + 1 * h$

$T(h) = 2^{h+1} - 2 - h \Rightarrow \Theta(2^h)$

T(n) = TC to build max heap of n nodes

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

Algo BuildMaxheapTD(a,n){

// Insert a[2] a[3] ---- a[n]

for(i=2; i<=n; i++){

insertionMaxheap(a,i)}

return (a[])

}

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

Algo BuildmaxheapBU(a,n){

// Adjust internal node

// a[⌊n/2⌋], a[⌊n/2⌋-1] ---- a[1]

for(i=⌊n/2⌋; i≥1; i--){

Adjust(a,i,n);

}

Return(a[1])

}

TC: Θ(n)

Heap Sort

- TC of heap sort: Θ(nlogn) [All case]
- SC of heap sort: Θ(1)
- It's Inplace but not stable Algo

Heap Sort Algo

// a[1...n] given array

1. build max heap
2. delete max from max heap of a[1...i] & place deleted element at a[i]

where i=n, n-1, ---- 2

Algo HeapSort(a,n){

// build max heap

Θ(n) [for(i=⌊n/2⌋; i≥1; i--){ Adjust(a,i,n); }]

// Del max repeat n-1 times

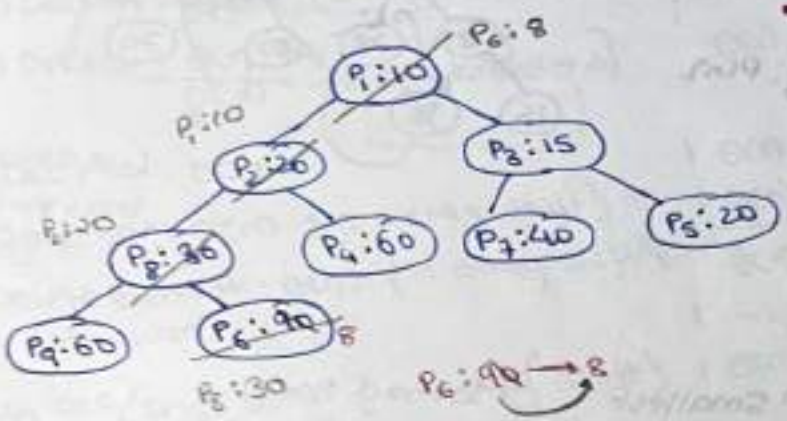
Θ(nlogn) [for(i=n; i≥2; i--){ a[i]=DeleteMax Maxheap(a,i); }]

Return(a[1]);

}

- Key decrement / Key Inc. of Min heap:- {ptr given}

[min heap]: Min val high priority



- Key decrement:

[To increase priority]

[Insertion op^n of heap]

[shift decon op^n]

Max $\log_2 n$ array

Shift or swap required

TC = O(logn)

- Key Increment: [decrease priority]

P_1: 10 → 95 a[1] = 10 95

Call Adjust (a, 1, n)

[shift up op]

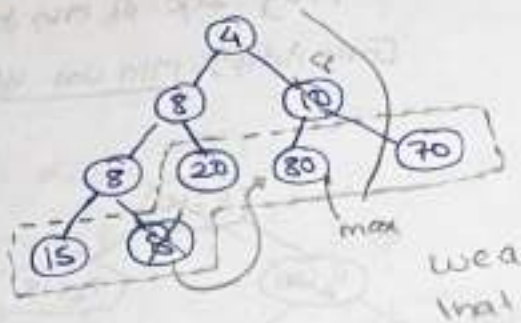
Max $\log_2 n$ array shift's

TC: O(logn)

- Key decrement / Key Inc. of Max heap:-

opposite of min heap

Ques] TC to find / delete max_element from min heap of n elements?



TC to find max in minheap : $\Theta(n)$

Del max : $\Theta(\log n)$

TC

TC = $\Theta(n)$

we assume that ptr is not given

Ques] TC to find k^{th} smallest element in min heap of n distinct elements?

Method 1: 1] Delete min. from given min. heap 4 Repeat (k-1) times.

2] Return (a[1])

↑

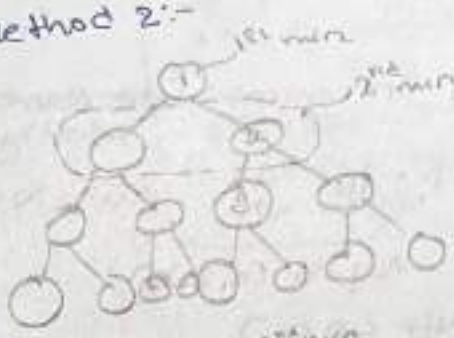
k^{th} smallest

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

$\Omega(\log n)$ to $O(n \log n)$

$k \approx Const$ $k \approx O(n)$

Method 2:-



2nd min : {a[2], a[3]} : 1 #f comp

3rd min {a[2], a[6], a[7]} : 2

⋮

k^{th} min {At most k elements} : k-1

$\Omega(1)$ to $O(n^2)$ TC = $\frac{k(k-1)}{2}$

$\Theta(k^2)$

① TC Required to find 7th smallest element in min heap of n distinct element

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

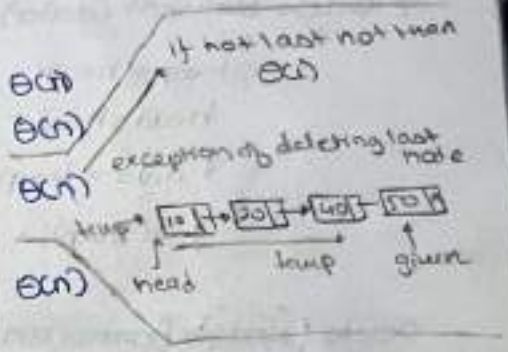
② TC Required to find n/5th Smallest element in min heap of n distinct element

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

Ques] Given sorted array of n elements [ASC]

TC of given operation:

	For sorted LinkedList in [ASC]
① Search: Log(n) [Binary search]	Θ(n)
② Del min: Θ(n) or O(n) [n array shift required]	Θ(1)
③ Del Max: Θ(1)	Θ(n)
④ Insertion: Θ(n) {n array shift}	Θ(n)
⑤ Delete element: Θ(n) {n array shift} [ptr given]	Θ(n)
⑥ Key dec/Inc: Θ(n) {n array shift} [ptr given]	Θ(n)



array

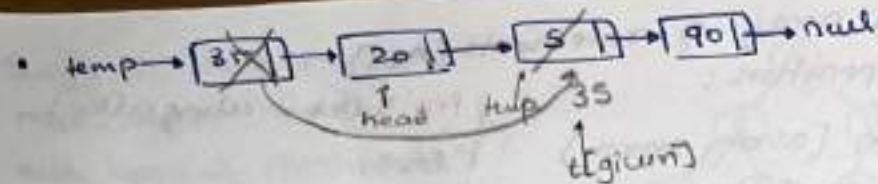
	1	2	3	4	5
a	10	20	30	40	50

- Given DS of n element

Operation	Sorted array ASC	unsorted array	Sorted SLL [ASC]	Unsorted SLL	sorted DLL [ASC]	unsorted DLL	min heap	Balanced BST [AVL tree]
Search	Θ(log n) B.S	Θ(n)	Θ(n) L.S	Θ(n) L.S	Θ(n)	Θ(n)	Θ(n)	Θ(log n)
Delete min	Θ(n)	Θ(n)	Θ(1)	Θ(n) [Find min]	O(1)	O(n)	O(log n)	O(log n)
Delete max	Θ(1)	Θ(n)	Θ(n)	Θ(n) [Find max]	O(n)	O(n)	O(n)	O(log n)
Insertion	Θ(n)	Θ(1)	Θ(n)	Θ(1)	O(n)	O(1)	O(log n)	O(log n)
Delete element (ptr given)	Θ(n)	Θ(1)	Θ(n)	Θ(1) ?	O(1)	O(1)	O(log n)	O(log n)
key dec/ key Inc [ptr given]	Θ(n)	Θ(1)	Θ(n)	Θ(1)	O(n)	O(1)	O(log n)	O(log n)

- minHeap/maxheap DS preferred to implement Priority Queue rather than Balanced BST (AVL tree) b/c if
  ① no overhead of pointers
  ② Should support for storing duplicate element

only support for no store distinct elements



⇒ Delete element (data) from unsorted SLL :-
[ptr given]

```
temp = head;
head = head -> next;
t->data = temp->data;
free(temp);
```

} TC = Θ(1)

Delete (Node) from unsorted SLL:-

```
temp = head
while (temp -> next != t)?
    temp = temp -> next;
;
Temp -> next = t -> next
free (temp)
```

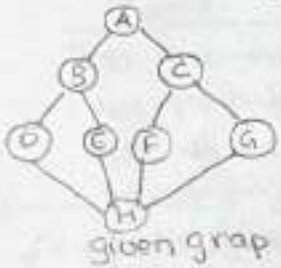
} TC : Θ(n)

Del by using ptr node address

Graph Representation

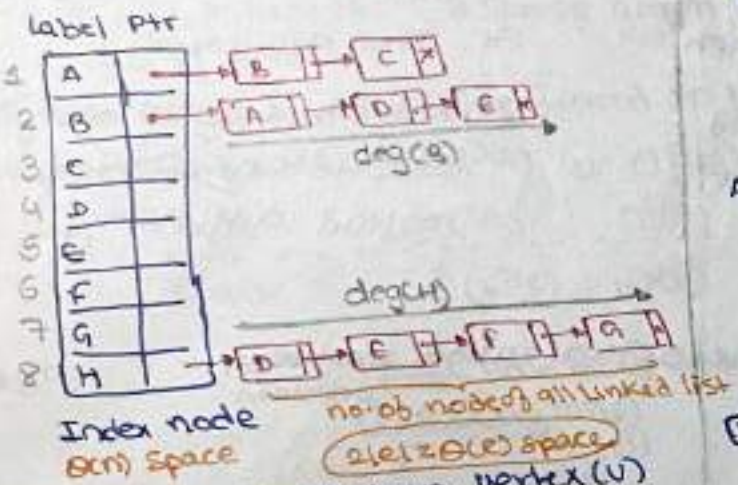
1. Adjacency List Rep.
   [using linkedlist]
2. Adjacency matrix Rep.
   [using 2D array]

- Sum of edges
- Undirected graph $\sum_{i=1}^{n} deg(v_i) = 2|e| = \Theta(e)$
- Directed graph $\sum_{i=1}^{n} deg(v_i) = \sum_{i=1}^{n} deg(v_i)$
  $\Rightarrow |e| \Rightarrow O(e)$



given grap

Adjacency List Rep.



1. TC to find Adj of given vertex(V)
   $\Theta(deg(v))$ $\Omega(1)$ to $O(n)$ B.C W.C
2. TC to find Adj of all vertices in loop
   $deg(v_1) + deg(v_2) + deg(v_3) + \cdots + deg(v_n)$
   $\Rightarrow \Theta(n+e)$
   atleast cost $[\Omega(n)]$ Atmost cost $[O(n^2)]$
   - If connected undirected graph with n vertices edges are
     $n-1 \le |e| \le \frac{n(n-1)}{2}$

B.C of edge $\Theta(n)$

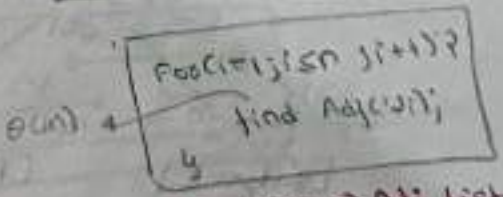
W.C of edge $\Theta(n^2)$

Adjacency Matrix Rep.

$Adj(i,j) = \begin{cases} 1, & \text{if } (i,j) \in \text{edge} \\ 0, & \text{if } (i,j) \notin \text{edge} \end{cases}$

Adj:

1. TC to find Adj of given vertex(V)
   $\Theta(n)$ All case
2. TC to find Adj of all vertices in loop:
   $= \Theta(n) + \Theta(n) + \Theta(n) + \cdots + \Theta(n)$
   $= \Theta(n^2)$ all case



TC: $\Theta(n+e)$ Adj list
TC: $\Theta(n^2)$ Adj matrix

Adjacency List Rep.	Adjacency Matrix Rep.
[3] SC to store graph → $\theta(n+e)$ Directory node → space for all SLL Space $\theta(n)$ $\theta(e)$ $\Omega(n)$ to $O(n^2)$ • if less edge then Best case is better than matrix	[3] SC to store graph : $\theta(n^2)$ All case (Space for 2D array) • no overhead of ptrs

• Dense graph: #f edges of the graph with n vertices are $O(n^2)$

Ex: Complete connected graph

• Adjacency Matrix is used bcz no overhead of ptr.

• Sparse graph: #f edeges of the graph with n vertes are $O(n)$

Ex: Tree, null graph

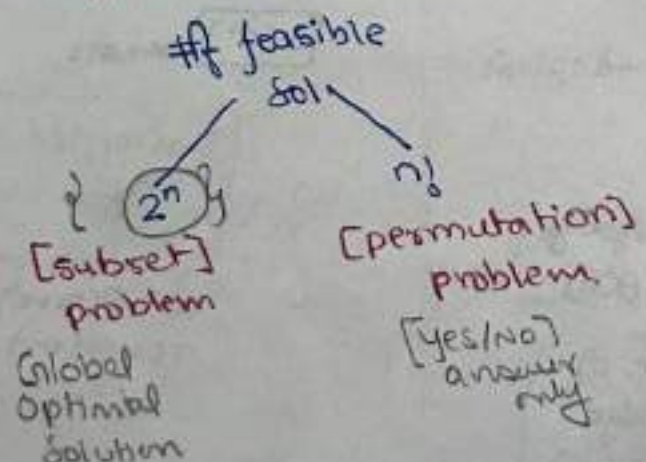
• Adjacency List is used bcz less time complⁿ required to find adjancency

Greedy Method • to solve optimization problem

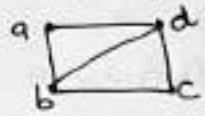
• Optimization Problem:-
Maximize / Minimize Result

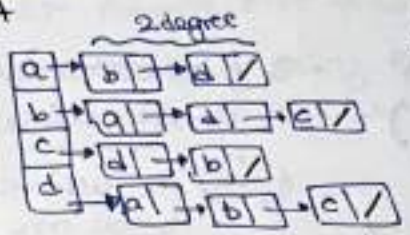
• Feasible solution:- Problem solution to leads optimal solution

• Mostly #f feasible solution to solve optimization problem with n i/p size is exponential.



- Adjacency list





$d_1+d_2+d_3+\cdots+d_n=2E$ } undirected

$d_1+d_2+\cdots+d_n=E$ } directed

Space	Adj. list	Adj. matrix
Test if $u \to v \in E$	$\Theta(1+\deg(u))$	$O(1)$
list v's neighbours	$\Theta(1+\deg(v))$	$\deg(v)$
list all edges	$\Theta(V+E)$	$\Theta(V^2)$
Insert edge uv	$\Theta(1)$	$\Theta(1)$
Delete edge uv	$\Theta(1+\deg(u))$ $+\deg(v)$	$O(1)$

- Density: no. of edges in term of no. of vertices
  - Dense graph, $|E| \approx |V|^2$
  - Sparse graph, $|E| \approx |V|$
- graph algorithm runtimes depend on $|V|$ & $|E|$

which is faster, $O(|V|^{3/2})$ or $O(|E|)$

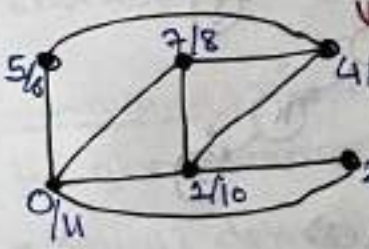
	$O(V^{3/2})$	$O(V^2)$
Dense	~~$O(V^{3/2})$~~	$O(V^2)$
Sparse	$O(V^{3/2})$	~~$O(V^2)$~~

- DFS

Start time ($s[v]$) and finish time ($f[v]$)

(discovery time) ($d[v]$): time at we visit vertex 1st time

time at we visited vertex last time

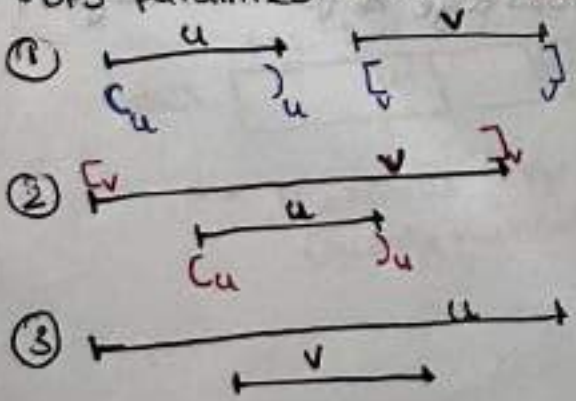


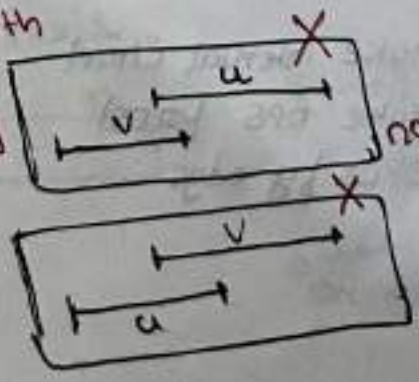
$d[v]/f[v]$

DFS forest

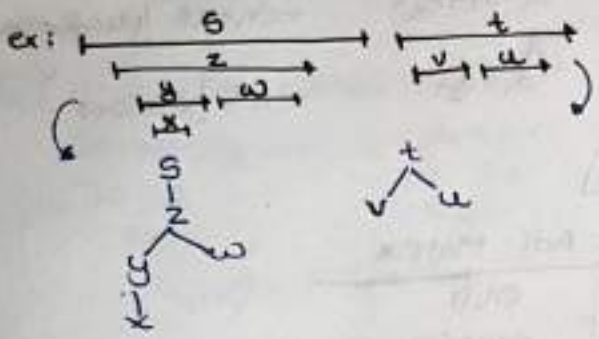
- getting multiple tree (set of disjoint tree)

- DFS parantheses theorem [to deal with start & finish time]





ex:

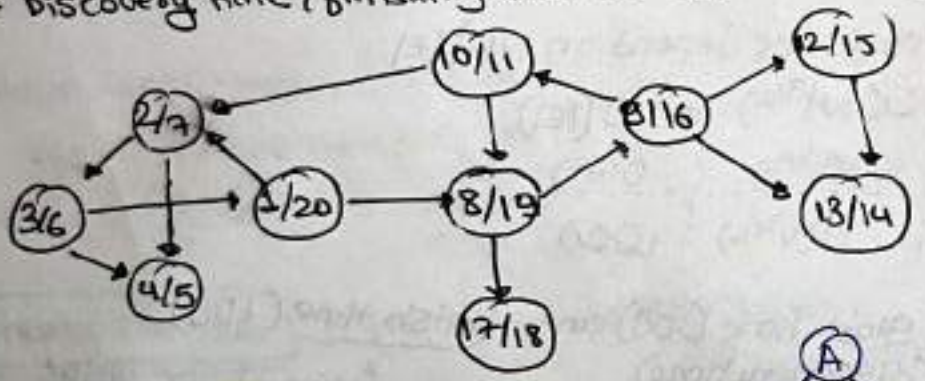


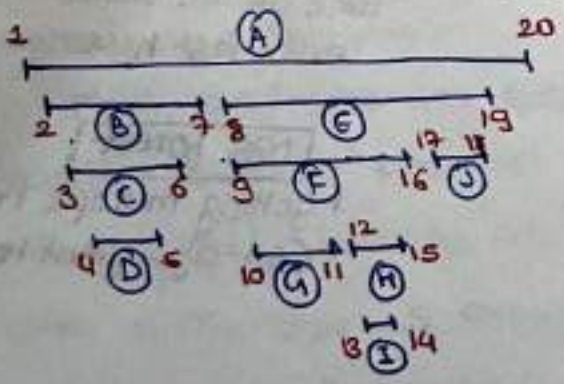
- s is ancestor of z
- x is descendant of y

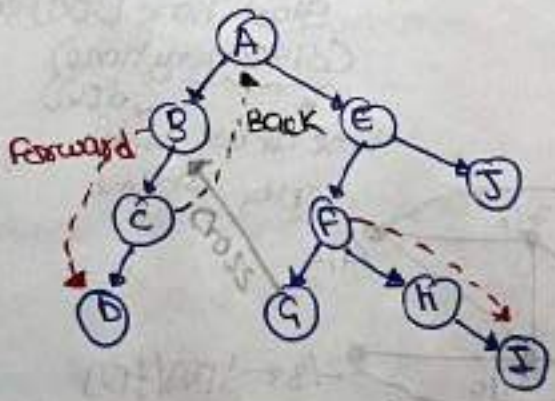
note: in undirected graph if you are getting forest then graph must be disconnected

- Edge classification
  - ↳ tree edge : edge in dfs forest
  - ↳ forward edge : an edge which is not a tree edge & goes from ancestor to descendant
  - ↳ Backward edge : not tree edge from descendant to ancestor
  - ↳ cross edge : no relation blw each other as ancestor or descendant

ex: Discovery time / finishing time no. of DFS are given for Directed graph







(undirected)

- DFS graph can't have an cross edge & forward edge

Step 1: make interval chart
Step 2: make DFS forest
Step 3: Classify edge

- **Greedy Method:-** Choose one feasible solution from solution space [All feasible sol.] by using predefined greedy strategy & Return as local optimal solution [Greedy]

- Adv
  - Greedy Algo always solves the problem in polynomial TC
- DisAdv.
  - not every optimize problem can be solved using greedy problem

[Greedy Method]

→ Single Source Shortest path (S.S.S.P)
  ① Dijstras Algo [Greedy] (Stack)
  ② Bell mans Ford Algo (queue) [not Greedy]
  ③ BFS Algo [Greedy]

→ Minimal cost spanning tree [MST]
  ① prims Algo [Greedy]
  ② kruskal Algo [Greedy]

→ Huffman Coding [optimal 2ways merge] (Greedy)

→ Knapsack problem
  ① Fraction KNP: Greedy
  ② 0/1 KNP: failed by greedy

→ Job scheduling based on deadline (Greedy)

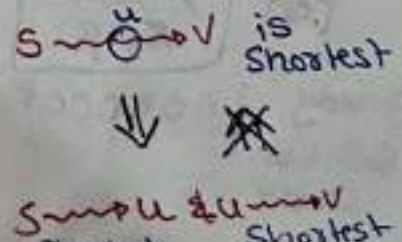
- MST & S.S.S.P are 2 different things

- **Single source Shortest path Algo:-** Given graph G(n,e) & edges cost cost[][] and Sources (s).

Problem Stmt: SSSP algo should compute Shortest path distance for given source(s) to every vertex in graph.

i.e Should determine dist[1...n] such that dist[i] is shortest path dist. from source(s) to vertex i

↳ Dijkstras Algo (Greedy method)



Dijkstra Algo

- For every node we create a separate table

① Initialization

dist[1...n] = +∞

Prev[1...n] = -1 & dist[s] = 0

List[Q] for all vertices

② Choose vertex(u) whose dist is min & Delete(u) from List[Q]

- apply relaxation to every edge exactly once

Apply edge Relaxation for Adjacencies of u

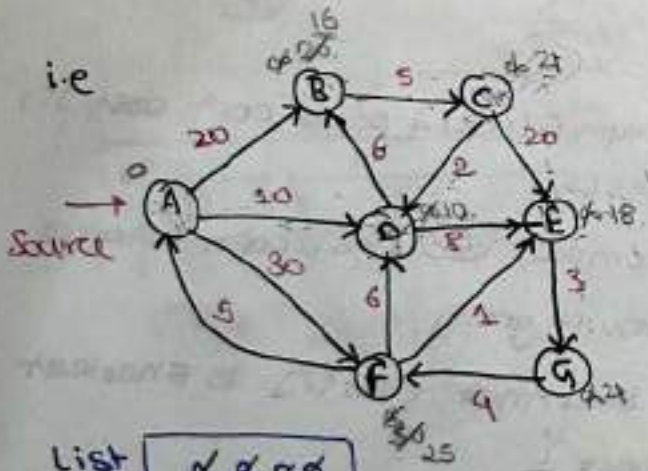
* edge relaxation of edge(u,v)

```
if(dist[u] + cost(u,v) < dist[v]){
    dist[v] = dist[u] + dist(u,v)
    Prev[v] = u;
}
```

- after every relaxation we are going to choose min. edge.

③ Repeat ② until list[Q] empt [n times].

i.e



	A	B	C	D	E	F	G	
dist	0	∞ 20 16	∞ 21	∞ 10	∞ 18	∞ 30 25	∞ 21	Dist

	A	B	C	D	E	F	G	
Prev	-1	~~X~~ A D	~~X~~ B	~~X~~ A	~~X~~ D	~~X~~ A G	~~X~~ E	edges

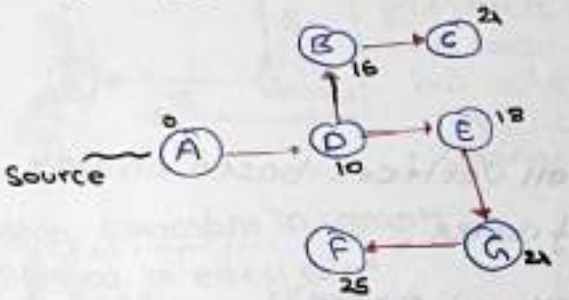
List [Q]: ~~A~~ ~~B~~ ~~C~~ ~~D~~ E F G

Greedy Selection Order: A D B E C G F

↳ what is greedy soln order of the vertices by using Dijkstra from A

Ans:- A D B E G C F
A D B E C G F

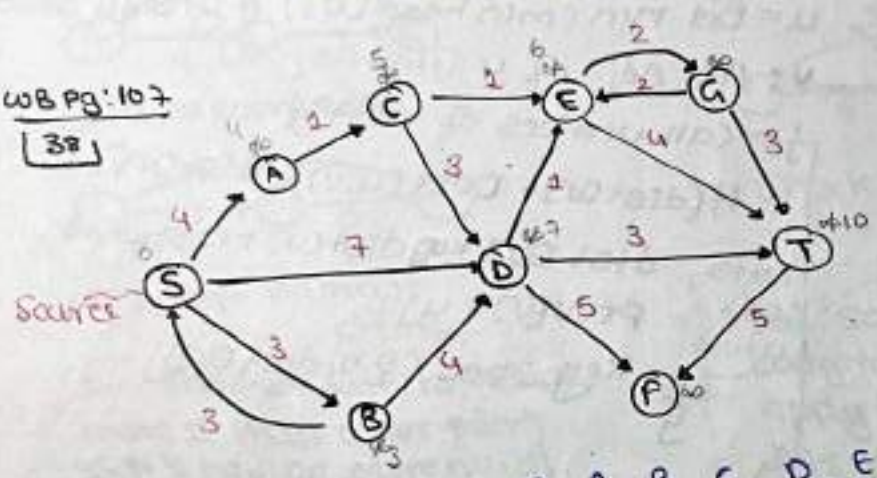
↳ Dijkstra shortest path spanning tree from (A)



A → F : A → D → E → G → F

WB Pg: 107

38



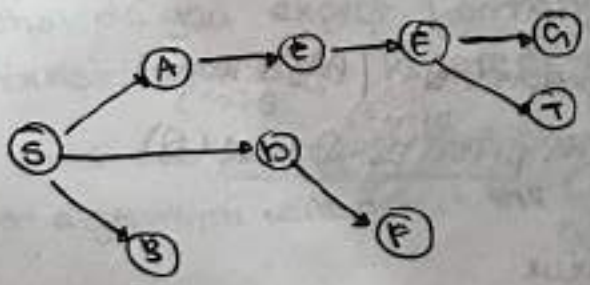
	S	A	B	C	D	E	F	G	T
dist	0	4	3	5	7	6	12	8	10

ABCDEF
GTS

SBACEDGTF

Greedy Sequence

	S	A	B	C	D	E	F	G	T
Prev	-1	S	S	A	S	C	D	E	E



Algo Dijkstras (G, n, e, cost[][], s) {

// Initialization

$\Theta(n)$ { for(i=1; i<n; i++){
dist[v] = +∞;
prev[v] = -1;
}

dist[s] = 0;

$\Theta(n)$ [Minheap[Q] for all vertices based on dist of vertex

no. of vertex

while (minheap(Q) not empty){

u = Del Min (min heap[Q]) // greedy selection

v: All Adj. of u [And Adj[u]]

for (all vertices of v set){

If(dist[u] + cost(u,v) < dist[v]) {

dist[v] = dist[u] + cost(u,v);

prev[v] = u;

Key_Dec(Q, v, dist[v]);

}

}

}

Return (dist[], prev[])

}

n

degree of u time (w.r.t. n)

worst case no. of edges or summation of all degree

- Main operation effect TC of Dijstras?

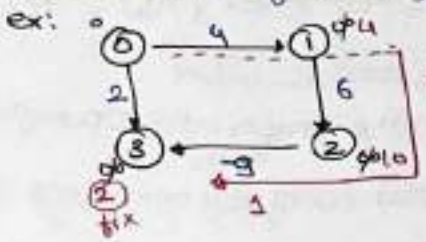
1. n times del min from priority queue (Q) which is used dijkstra greedy selection
2. find Adj of all vertices in graph [Adjust list $\Theta(n+e)$ / Adj Matrix $\Theta(n^2)$]
3. $\Theta(e)$ time key decrement from priority queue (Q).

if you find smaller value than the current from other path then change to new value

- Big Misconception

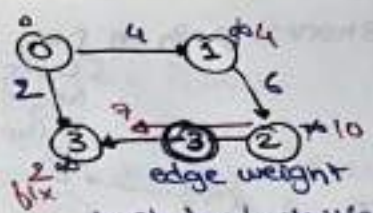
Dijkstra may or may not give correct ans. if -ve edge is present

ex:



from there path the ans. is (1) but 2 is already fix we can't change now

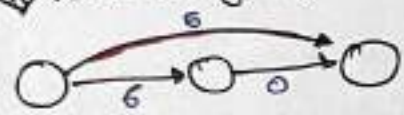
ex:



Just by luck it's working

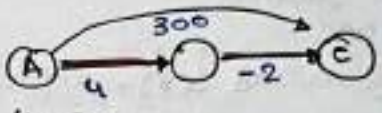
- Adding constant in graph applied to every edge

⇓ +2 to every edge

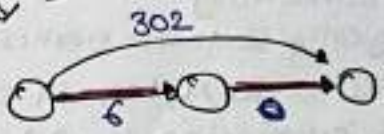


- Shortest path got change

(b/c shortest path is getting more 2 than other path) as it's having more no. of edges



⇓ +2

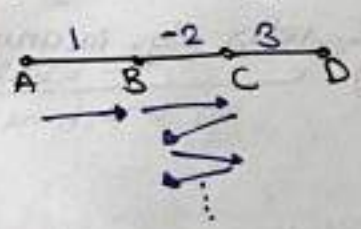


Shortest path remained same

Note

- add some weight to make every weight as non-negative & apply Dijkstra this doesn't work b/c adding a weight to every edge add more weight to long path than short paths.

- undirected graph



in case of undirected graph, just one (-ve) edge is like a (-ve) cycle

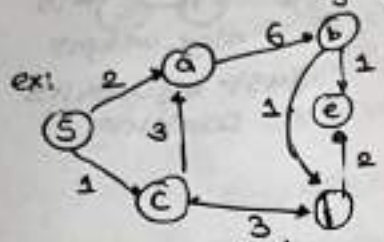
- So shortest path doesn't contain a cycle

ex: what do you expect from any shortest path algo. in case of -ve cycle?

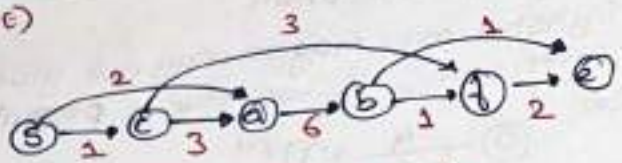
at max. we can expect that shortest path algorithm will detect the (-ve) cycle

• Shortest path in DAG (Direct acyclic graph) T.C: O(V+E)

ex:



redraw



Topological order

1. convert the graph into topological order
2. then relax every edge one by one

• Algo. for shortest path in DAG

1. Set d[s] = 0 and d[v] = ∞ for all v ≠ s
   (d: unsorted array)
2. topologically sort the vertices of G ← O(V+E)
3. for each vertex u, take in topologically sorted order do
4. For every edge (u,v) do u → v
5. Relax(u,v) O(1)
6. end for
7. end for

v_1	1+deg(v_1)
v_2	1+deg(v_2)
v_3	
⋮	
v_n	1+deg(v_n)

Sum of all ⇒ V+E

• Dijkstra: pick min & relax outgoing edge O(V+E) log V)

Special case

• DAG shortest path: pick in topologically & relax outgoing edge O(V+E)

• Bellman ford: relax edge in any order for V−1 time O(V·E)

- TC of Dykstras Algo:-

① By using Adj. list & min heap used for priority queue

$$n * \Theta(\log n) + \Theta(n+e) + e * \Theta(\log n) = \boxed{\Theta((n+e)\cdot \log n)} \approx \Theta(e \log n)$$

$n * \Theta(\log n)$: one del min; $\Theta(n+e)$: Adj list; $e * \Theta(\log n)$: key decrement

$\Omega(n \log n)$ to $O(n^2 \log n)$

② By using Adj Matrix graph & min heap prority Q.

$$n * \Theta(\log n) + \Theta(n^2) + e * \Theta(\log n) \approx \boxed{\Theta(n^2 + e \log n)}$$

$\Omega(n^2)$ to $O(n^2 \log n)$

③ By using array as priority Queue [unsorted array]

$$n * \Theta(n) + \begin{Bmatrix} \Theta(n+e) \\ \text{or} \\ \Theta(n^2) \end{Bmatrix} + e * \Theta(1) = \boxed{\Theta(n^2)} \text{ All cases}$$

④ by using sorted linkedlist as priority Q

$$n * \Theta(1) + \begin{Bmatrix} \Theta(n+e) \\ \text{or} \\ \Theta(n^2) \end{Bmatrix} + e * \Theta(n) \approx \boxed{\Theta(n \cdot e)}$$

$\Omega(n^2)$ to $O(n^3)$

⑤ by using unsorted linkedlist as priority Queue

$$n * \Theta(n) + \begin{Bmatrix} \Theta(n+e) \\ \text{or} \\ \Theta(n^2) \end{Bmatrix} + e * \Theta(1) \approx \boxed{\Theta(n^2)} \text{ All cases}$$

⑥ By using sorted array as priority Queue's

$$n * \Theta(1) + \begin{Bmatrix} \Theta(n+e) \\ \text{or} \\ \Theta(n^2) \end{Bmatrix} + e * \Theta(n) \approx \boxed{\Theta(n \cdot e)}$$

$\Omega(n^2)$ to $\Omega(n^3)$

we are build the Algo in such a way that it is desc order assume

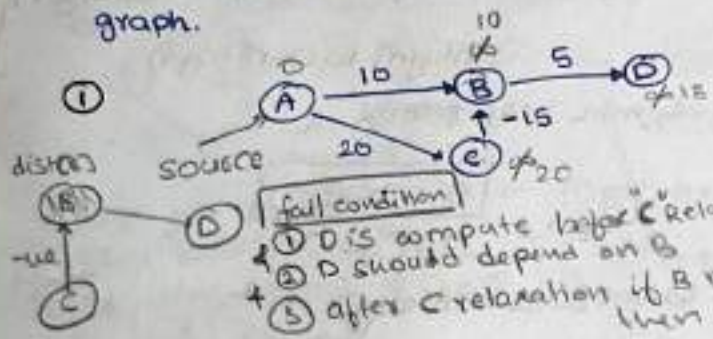
Main operation of Dijkstras Algo

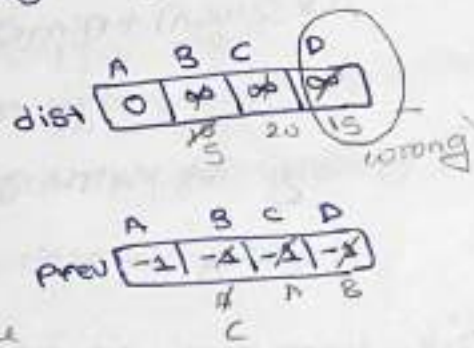
1. n time delete min from priority Queue
2. find Adj. of all (list) (matrix) vertices $\Rightarrow \Theta(n+e) / \Theta(n^2)$
3. $\Theta(e)$ times (key decrement)

whenever you find a min path that distance should also be reflected in the priority Queue

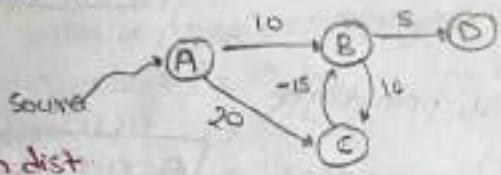
- Limitation of Dijkstra Algo:-

1. Dijkstra algo. may fail to compute shortest path distance from source if negative edges weight edges exist in the graph.





2. Dijkstra Algo fail to detect -ve cycle Reachable from source if exist



A to D
10+5=15
10+10-15+5=10
10+10-15+10-15+5=5

[Shortest path dist not well defined if vertices reachable though -ve cycle from source]

{Algo should detect -ve cycle reachable from Source if exist}

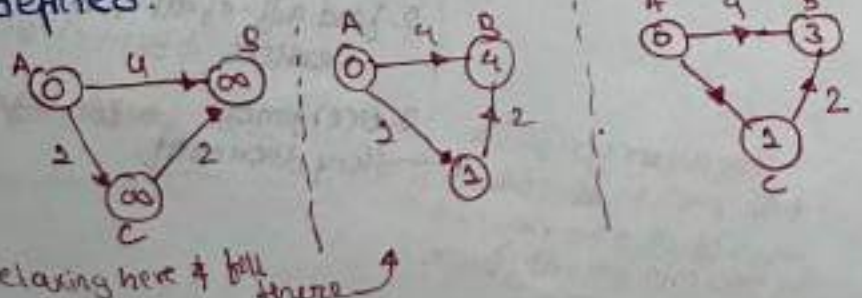
⇓

Dijkstras alro failed to detect same

- Bellman ford Algo:-

[not greedy method]
[uses Dynamic programing]

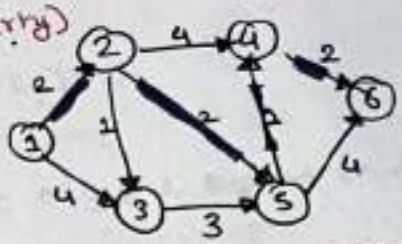
- Bellman ford Algo. compute correct Shortest path distance even if graph Consist -ve edges if Shoolest path distance is well defined.



- Bellmanford algo also detect -ve edge weight cycle reachable from the source if exist

- Bellman ford (Path relaxation property)
- if we relax in the order of shortest path (along with intermixed other relaxation) then we will still get shortest path cost.

ex:



(1) -ve edge are allowed in Bellman ford

- relaxation should be A/c to the Shortest path

but should contain the Shortest path

$s-v_1$ v_1-v_3 v_3-v_4 v_4-t

v_1-v_3 $s-v_2$ v_2-v_3 v_3-t

v_1-v_4

- Core idea behind Bellman-ford
  If we relax v-1 time then we are guarantee to relax in Shortest path order for every vertex

(2) • doing relaxation v^{th} time & if the value change's then there is a cycle

Algo:

```
Bellman-Ford (G,S):
d[v] = ∞ for all v in V
d[S] = 0
for i = 1 to n-1                      O(V·E)
    for each node (u,v) in E
        RELAX (u,v,w)
For each edge (u,v) in E              O(E)
    if (d[v] > d[u] + w)              O(1)
        return false
return true
```

```
RELAX (u,v,w):
    if (d[v] > d[u] + w)              O(1)
        d[v] = d[u] + w
        parent[v] = u
```

note:
- if a given sequence is repeated again & again (v-1) time then the Shortest path sequence while be present in that whole sequence

ex: 1 5 3 2 6

2 5 1 6 2 3, 2 5 1 6 2 3 2 5 1 6 2 3

(3) T.C = V·E

worst case: T.C = O(V·E)

Best case: T.C = O(E)

After k^{th} time relaxation

⇓

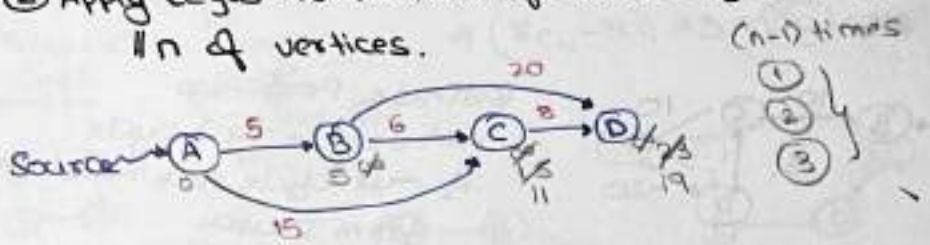
(4). we have ans. to all vertices which have atmost "k" Shortest path length (get Shortest path of length k) or less

① Initialize dist & prev. array.

dist[1...n] = +∞ | dist[s] = 0 (same)

prev[1...n] = -1

② Apply edges Relaxation of all e edges & Repeat (n-1) times
|| n of vertices.

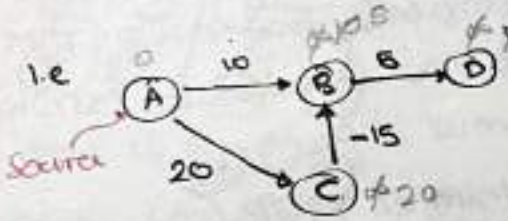


③ -ve cycle reachable from source detection.

Apply Edge Relaxation nth time

if dist[v] of nth time < dist[v] of (n-1)th time

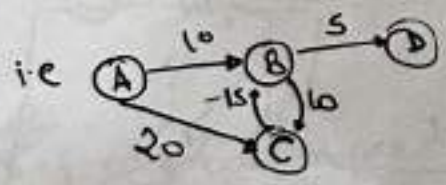
then -ve cycle reable from source exists.



	A	B	C	D
Dist	0	∞	∞	∞
prev	-1	-1	-1	-1
i=1	0	10	20	∞
	-1	A	A	-1
i=2	0	5	20	15
	-1	C	A	B
i=3	0	5	20	10
	-1	C	A	B
nth i=4	0	5	20	10
	-1	C	A	B

(n-1) time

Op :- no -ve cycle Reachable from Source return (dist[], prev[]);



	A	B	C	D
Dist	0	∞	∞	∞
i=1	0	10	20	∞
i=2	0	5	20	15
i=3	0	0	15	10
i=4	0	-5	10	5

Op. -ve cycle Reachable from source exists.

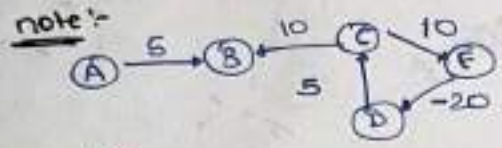
Return (false);

T.C of Bellman ford Algo : $\Theta(n \cdot e)$

worst case TC of Bellman ford Algo : $O(n^3)$

$\Theta(e) * (n-1)$

note :-



Bellman ford Algo not detects -ve cycle if -ve cycle not reachable from source

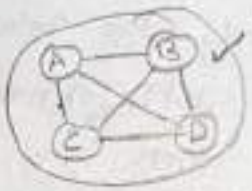
applied on undirected graph

Minimal Spanning tree [MST]

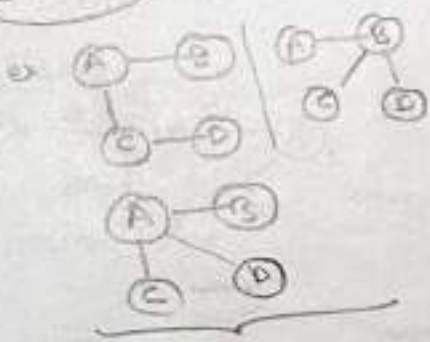
Spanning tree of graph $G(V,E)$ s.t $G'(V,E')$ $E' \subseteq E$

G' is tree

[connected & Acyclic]



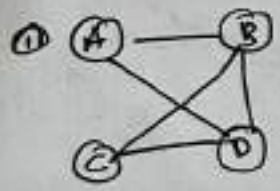
ST : $4^{4-2} = 16$



G' : ST

- no. of possible spanning tree in undirected complete connected graph of n vertice are n^{n-2} — Cayley's formula

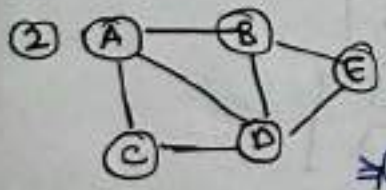
ex: How many possible SFs in given graph



total no. of edge

$^5C_3 - 2 \Rightarrow$ 8 Spanning tree

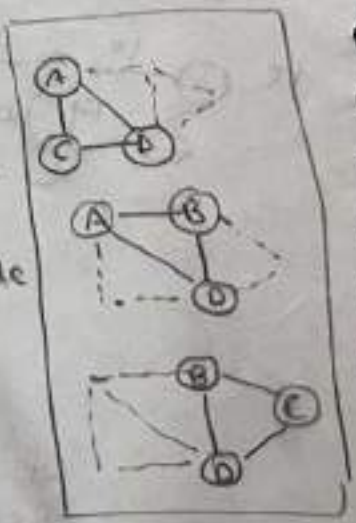
- $^5C_3$: min edge in ST
- 2 : form's cycle [ABD, CBD]

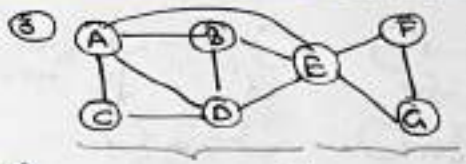


$^7C_4 - 2 - 12$

- $^7C_4$: cycle that formed by 4 edge [ABDC, ABED]
- 2 : form's 3 edge cycle

21





can be solved Separately as both are independent of each other.

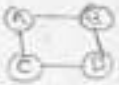
$^8C_4 - 5 - 5*5 * {}^3C_2$

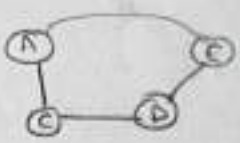
4 length cycle

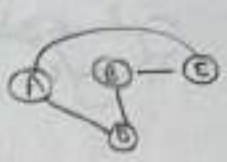
3 length cycle

ACD, ABD, BDE
BEC, AD, E

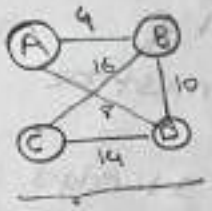
$\Rightarrow (^8C_4 - 30) * 3 = 120$ ST's

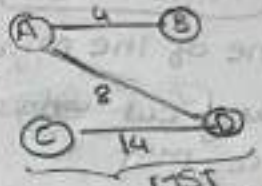






Minimal spanning Tree [MST]



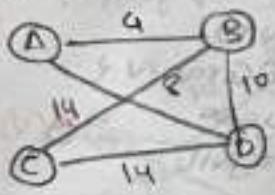


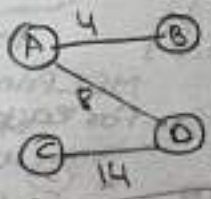
MST

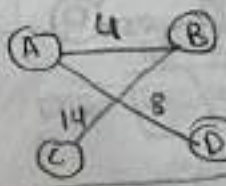
MST: given weighted graph G(V,E) edges costs cost[][] MST of Graph G is sum of edges cost of ST is minimum.

Cost of MST = 4+8+14 = 26

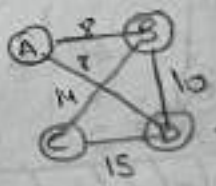
- If edge weight are not distinct in graph(G) then may not unique spanning tree
- if edge weight of the graph is distinct then unique MST for the given graph

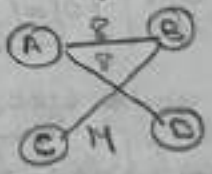






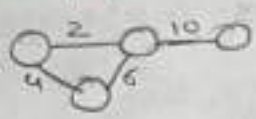
MST's

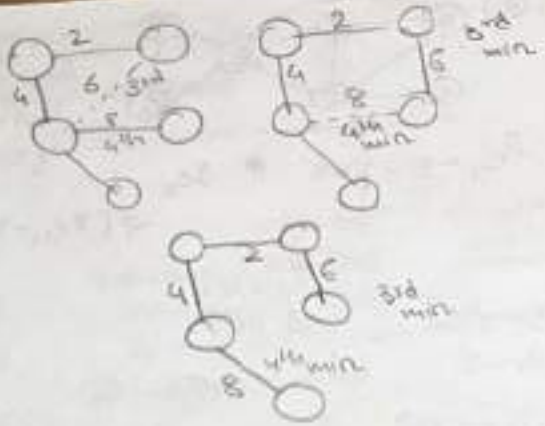




- If edge weight are distinct in graph G then
  - → min cost edge of Graph G must be in graph G
  - → 2nd min cost of G also must be in G
  - → either 3rd or 4th or both should be in the MST

→ if max cost edge in MST then Removal of the edge will disconnect the graph

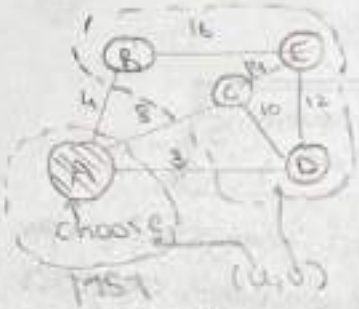


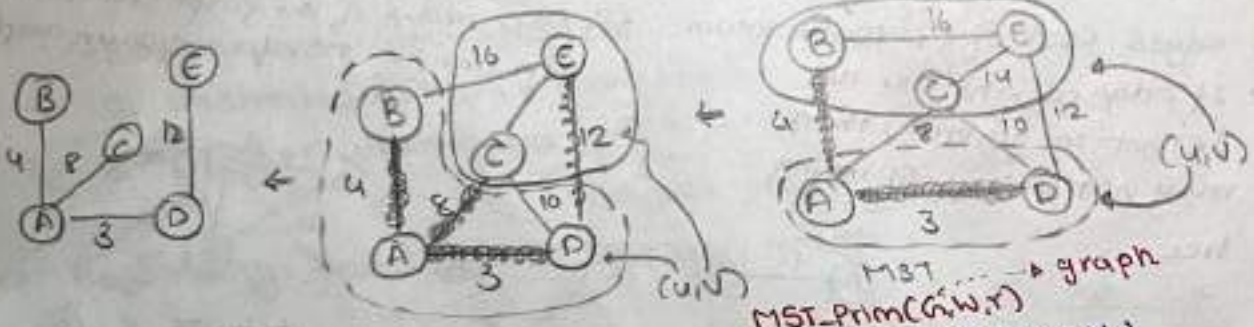


Prims Algo :- [Greedy]

- one of the min cost of every cut edges is must be in MST

cut edges : Removal of set edges disconnect graph





write code for prims Algo

- Starting point will be given.

So, $V+V+V\log V+V^2\log V$

$O(V^2 \log V)$

In worst case

b/c it's depending on edge's & then we perform decrease key

So T.C becomes

$O(E \log V)$

So E will lie b/w $O(V^2)$ to $O(V)$

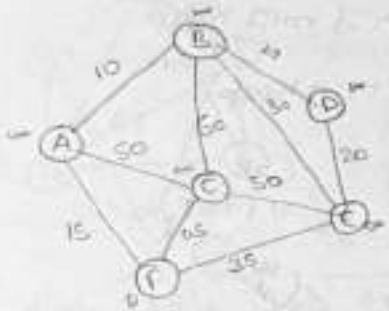
either include all — includes all vertex no. of edge

```
MST_Prim(G,W,r)
1. For each u ∈ Graph.V {
      u.key = ∞                    ⇒ O(V)
      u.π = NIL
   }
   r.key = 0
   Q = G.V  ← (Build heap)         ⇒ O(V)
   while Q ≠ ∅ {                   (Vertex (V) time)
      u = Extract_min(Q)           ← log V
      for each v ∈ G.Adj[u] {      ← (V-1) in worst case
         if v ∈ Q and
         w(u,v) < v.key {
            v.π = u
            v.key = w(u,v)         ← log V
         }
      }
   }
```

picking vertex one by one

modifying the weight

Decrease key

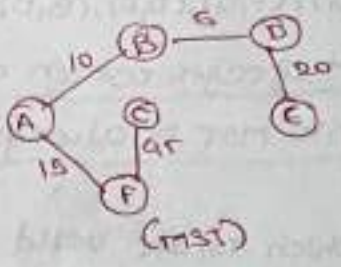


MST (V) (U) min distance nodeedge

```
if (dist[v] > cost(u,v) &&
    color[v]) {
    dist[v] = cost(u,v)
    prev[v] = u
}
```

	A	B	C	D	E	F
dist	∞	∞	∞	∞	∞	0
	15	10	45	6	35	80 20

MST edge weights



(MST)

	A	B	C	D	E	F
prev	-1	-1	F	-1	B	-1
	F	A	F	B	B	

MST edge

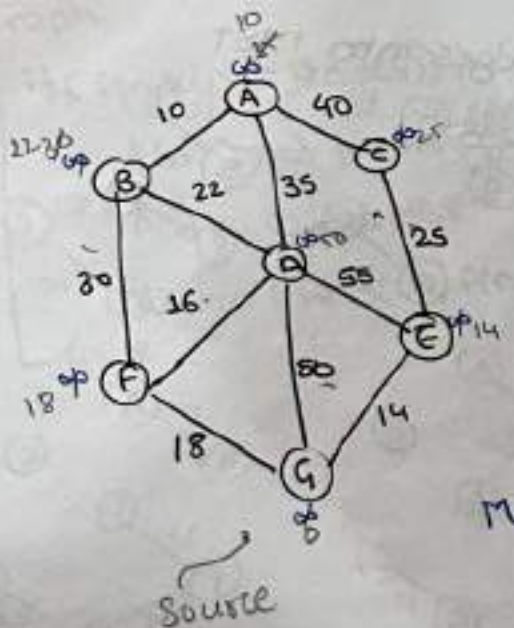
D

	A	B	C	D	E	F
color	W	W	W	W	W	W
	B	B	B	B	B	B

Min heap: A B D

F A B D E C

ex:



	A	B	C	D	E	F	G
Dist	0	22	35	50	39	14	18
		10	22		16		

MST edge cost

	A	B	C	D	E	F	G
Prev	-1	A	B	C	D	E	F
		B	D		F		

MST edge

	A	B	C	D	E	F	G
color	B	B	B	B	B	B	B

Min heap { A B C D E F G }

A B E D F G C

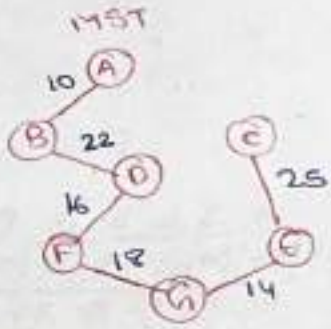
A B E D F G C

• Tc of prims Algo

$\Theta((n+e)\cdot \log n)$ // using Adj list graph & min heap

$\Theta(n^2+e\log n)$ // using Adj matrix & minheap

$\Theta(n^2)$ using array implementation



Prime edge sequence of MST

either E or G should be source → (E,G), (F,G), (D,F), (B,D), (A,B), (C,E)

[set of edges which are included into MST is always connected] } Imp

ex:- which is/are valid prime sequence of edges to build into MST

(a) (B,D) (A,D) (D,F) (F,G) (E,G) (C,E)

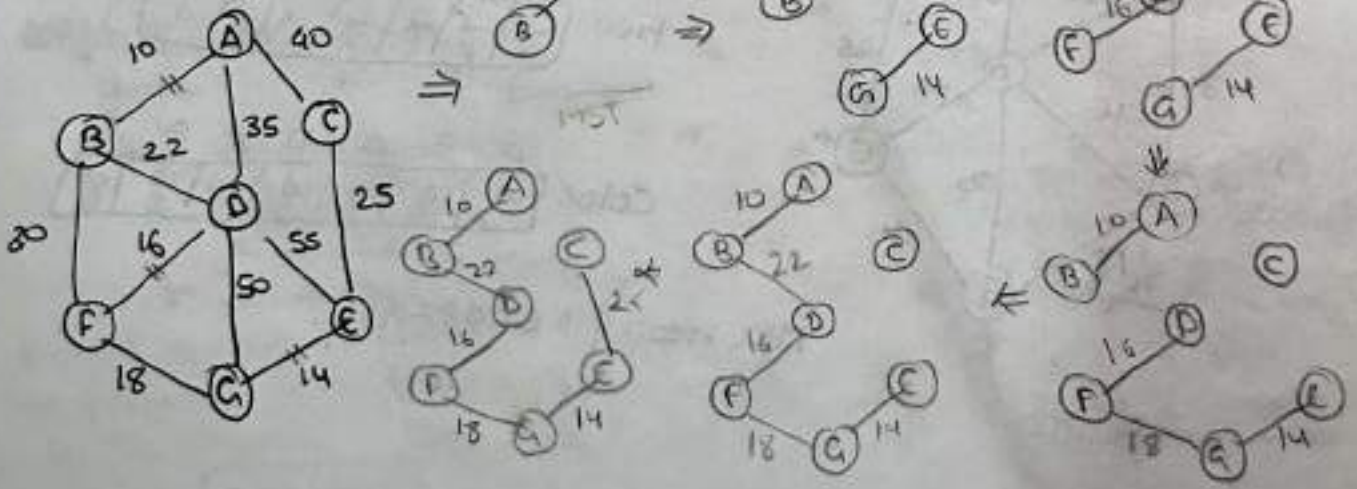
(b) (D,F) (F,G) (E,G) (B,D) (A,B) (C,E)

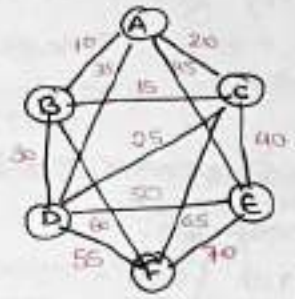
(c) (A,B) (B,D) (F,G) (D,F) (E,G) (C,E)

(d) (C,E) (E,G) (F,G) (B,D) (D,F) (A,B)

either C or E can be source

Kruskal Algo // Add min cost edges into MST





Disjoint set union, find algo used for to test next min cost C(u,v) of G is forms cycles or not in MST

- Union(i,j) // union of two set

TC : θ(1)

- Find(i) // Return set which contain?

TC: θ(n)

{A} {B} {C} {D} {E} {F} // n sets.

(A,B) → Find(A) ≠ Find(B) // Add(A,B) into MST
{A,B} {C} {D} {E} {F}

(B,C) → Find(B) ≠ Find(C)
{A,B,C} {D} {E} {F}

(A,C) → ×

(C,D) → Find(C) ≠ Find(D)
{A,B,C,D} {E} {F}

(B,D) → ×

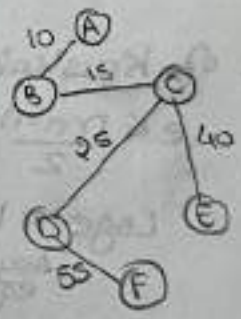
(A,D) → ×

(C,E) ✓
{A,B,C,D,E} {F}

(A,E) ×

(D,E) ×

(D,F)
{A B C D E F}



Min cost edge (u,v) del from Graph

```
if(find(u) ≠ find(v)){
    // (u,v) not forms cycles in MST
    Add (u,v) into MST
    Union (Find(u), Find(v))
}
```

TC: θ(1)

Algo Kruskals (cost[][], Edges[][], n, e){

θ(e) { • Build minheap(Q) for all e edges } Sort e edges in ASC order
based on edges cost

θ(e) → while(minheap(Q) not empty and MST edges (t) are < n-1){

loge ← (u,v) = DelminCostEdges(minheap(Q))

if(Find(u) ≠ Find(v)){ || (u,v) not forms cycle in MST(t)

• Add(u,v) in MST(t)

θ(elogn) • union(Find(u), Find(v));

}

}

if(MST(t) < n-1){

• Graph Disconnected

• no MST possible

} else

return (MST(t));

}

• forms unique MST if edges are distinct

TC of Kruskal Algo : θ(elogn)

$e \leq \frac{n(n-1)}{2}$

Loge ≈ logn (Asy equal)

Ω(nlogn) to O(n²logn)

• if cost[][] is in ASC[i/p given is ASC]

then TC : will be-...) θ(e)

now we don't need Delmin cost Edge as they array is already ASC (just iterate)

• set of edge Select so far during MST construction of the Kruskal algo may not connected

Time complexity for kruskal algorithm

Kruskal(G):

Sort edges in increasing order } ElogE

$T = \phi$

E { For each edge e_i — if $e_i \cup T$ is not making a cycle: $T = T \cup e_i$ } $E\alpha$ (α)

So, $T.C = ElogE + E\alpha$ (α = time to check cycle)

$ElogE + E\alpha$

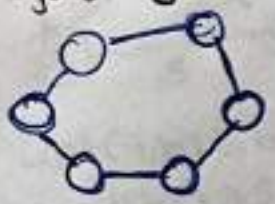
- DFS: $\alpha = O(V+E)$, $ElogE + VE + E^2$
- union-find data structure: $\alpha = O(1)$, $ElogE + E \cdot 1 \Rightarrow ElogE$

- Kruskal's algo time complexity $\Rightarrow ElogE$
- Dijkstra algo time complexity $VlogV + EO(1) \Rightarrow VlogV + E$ (Adj. list + fibanoi heap)
- if the edge weight are of appropriate value & Radix Sort can be applied instead

Sorting time + $E \cdot \alpha$ } $\Rightarrow$ $T.C = O(E)$

$O(E) + E$

- Can Kruskal handle -ve weight? yes (will not impact anything)
- no. of MST in cycle graph where all edge weight are same?

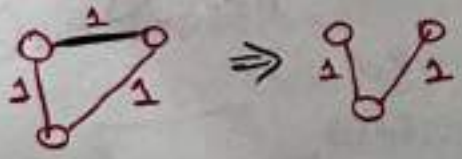


${}^{n}C_{n-1} \equiv {}^{n}C_{1}$

$\Rightarrow$ n possible spanning tree

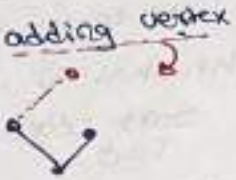
S_1: Min. weight edge is present in some MST yes

S_2: ..... " ..... " ... all MST no (if Duplicate) / yes (if distinct)



• Do we need to detect cycle in prims. no, b/c we are adding vertex

↳ we need to check cycle if we are adding edge's so, we can never have a cycle



Algo. for Dijkstra & prims are same

```
Algo Prims(G,n,e, cost[][],s){

    for(i=1 to n){
        dist[vi]=+∞
        prev[vi]=-1
    }
    Dist[s]=0
    minheap[Q] for all vertices based
                    on dist. of vertex

    while(minheap(Q) not empty){
        u=Del. min(minheap[Q])
        V:All Adj. of u
        for(all vertices of v set){
            if(cost(u,v) < dist[v]){      ! only difference
                dist[v]= cost(u,v);       ! comes when compared
                prev[v]=u;                !      by dijkstra
                key_dec(Q,v,dist[v]);
            }
        }
    }
    Return(dist[],prev[]);
}
```

Prime Time complexity = Same as Dijkstra time complexity

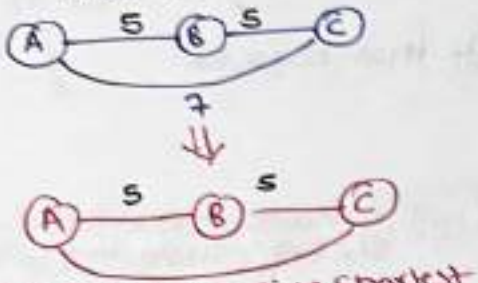
Prims (Vs) Dijkstra

• Both algo. takes one vertex at a time & relax outgoingedge

Prims	Dijkstra
Closest vertex to the tree	closest vertex to the source
(min. edge in the present graph)	(min from source) dist.

• MST (VS) shortest path

→ if we have MST then do we have shortest path? need not to be true



MST need not give shortest path

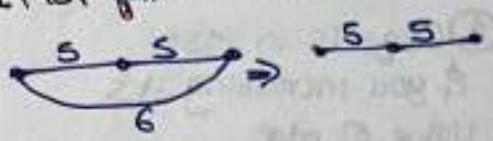
ex: if G has a cycle & there is unique edge 'e' which has the min. weight on the cycle then e must be part of every MST false

bic it may be possible that it's an edge from other cycle which is larges.

min in one cycle & max. at other

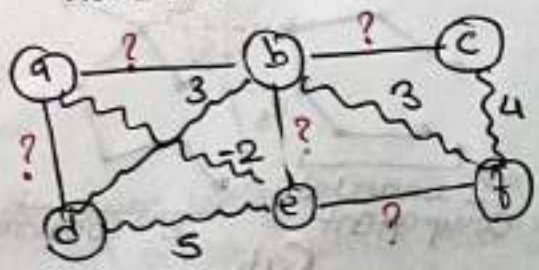
note → if the edge 'e' is not a part of any MST of G, then it must be the max. weight edge on some cycle in G'

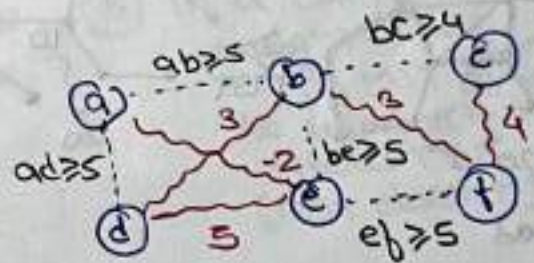
ex. edge weight are non-negative then the shortest path btw two vertice must be part of some MST false



TIFR 2014

ex wavy edges form MST for G then following inequalities need not hold?





(A) cost(a,b) ≥ 6 false

(B) cost(b,e) ≥ 5

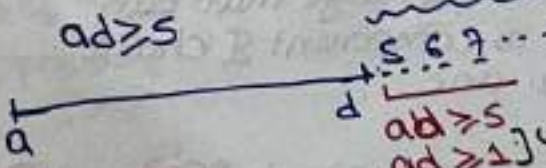
(C) cost(e,f) ≥ 5

(D) cost(a,d) ≥ 4

(E) cost(b,c) ≥ 4

} True

we can't take (5) Nc to option



ad ≥ 5
ad ≥ 1 } we can write even tho 'x' can never be 1
ad ≥ 4

even tho 'x' can never be 1,2,3,4

• range of some particular edge which is unknown

Step 1 :- draw MST (may be some edge weight are unknown here)
add non-MST edge one by one
↳ apply theorem

Step 2: if some edge is not in MST iff that edge is heaviest edge in some cycle.

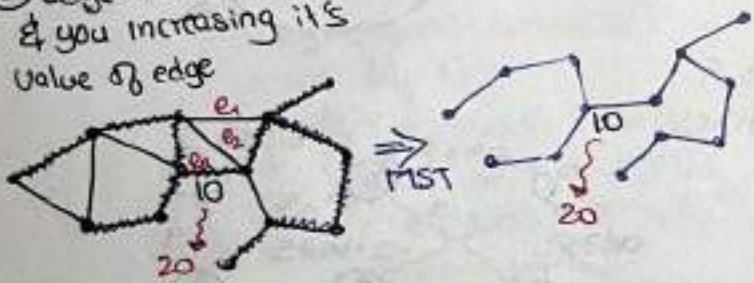
• 4 cases in MST

given a MST

↳ edge is in MST
① value is decreased } MST will remain same but cost of whole MST will decreased
② value is increased

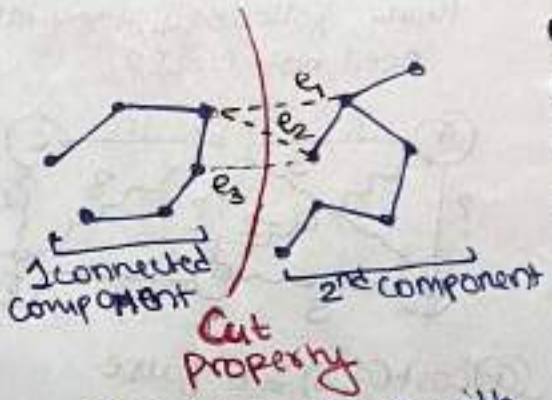
↳ edge is not in MST
③ value is decreased
④ value is increased } will still remain the same MST will not effect the original MST

② Edge is in MST & you increasing it's value of edge



Tad

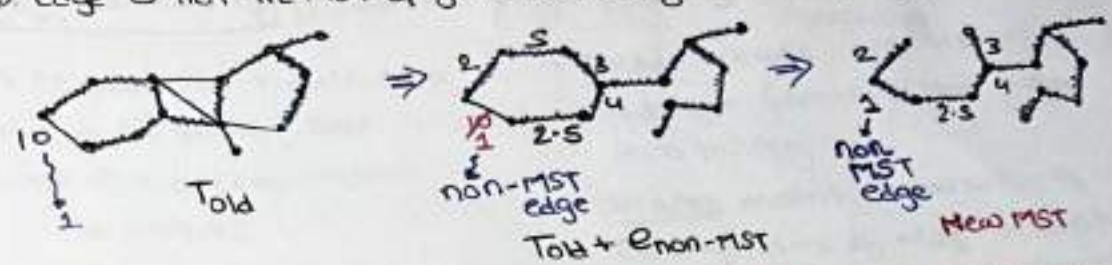
• when increasing an particular edge then we use cut property & divide it into 2 component & then compare every edge that can connect to component & choose an min. one



O(1) : to compare with MST edge.

T.C = O(V+E) + O(E) + O(E) * O(1)
O(V+E) : find connected component
O(E) : find all cross edge
O(E) : min among all the cross edge
O(1) : compare

③ Edge is not in MST & you decreasing value of edge



- by decreasing the weight we don't know wheather it's still is the min. to remain in the MST
- So, we add the non-MST edge which create a cycle then after that we will pick the max. & remove it.

$T.C = O(V+E) + O(E) + O(1)$
(V+E): to check for cycle; E: to find all edge weight; O(1): comparsh

ex: edge weight are different

(a) In a cut, min. edge weight will always be part of MST
True

(b) In a cut, max edge weight will be part of MST false
As we don't know about max.
(MST doesn't tell about max.)

ex: edge weight need not to be distinct in a cut, what can you say about min edge weight
↳ if it's unique → part of all MST
↳ if it's not unique → part of some MST

* ex. weight need not to be distinct in cycle, what can you say about heaviest edge
↳ if that is unique → never be a part of any MST
↳ if it's not unique then → will not be a part of some MST

Huffman coding (Optimal prefix encoding)

- It is used for reduce msg length in Binary Rep. [one of the Compression technique]
- 2 type of encoding & decoding technique
  - ↳ fixed length encoding
  - ↳ optimal prefix encoding / Huffman coding

- Encode means conversion of characters forms of the msg into binary
- Decodes means conversion of Binary form of msg into characters
- the order of the tree in which you choose can be anything

1. Fixed length encoding

- each character of the msg encode equal no. of bits

using 2 bits ⇒ 4 char encoding

00	a
01	b
10	c
00	d

using n bit ⇒ 2^n char encoding

To encode n char's $\lceil \log_2 n \rceil$ bit's required

ASCII [Standard fixed length encoding]

- 7-bit encoding for each char

#of printable char's ⇒ a-z, A-Z, 0-9, +, -, ÷, !, @, $ etc } ≈ 100 char's ⇒ $\lceil \log_2 100 \rceil$ = 7 bits

- more msg length w.r.t no. bits b/c no difference b/w most frequen used characters & least frequent characters w.r.t no. of bit

Huffman Coding [optimal prefix encoding]

- Most frequent used characters represented by less bit. less frequenet used characters represented by more bit, such that proper prefix of any character code should not be the code of other character.

101001

1	proper prefix
10	
101	
1010	
10100	

- Huffman code are always "prefix free code"
- For balanced tree variance will be lesser (or) greater? lesser.

2way optimal Merge [greedy Algo]

(Huffman Tree)

Pre-requirement : #f distinct char's[n] and frequency count of each char of msg should be given

- Forest : n trees with weight is frequency
- Build min heap (Q) for all n char based on frequency count of char to construct huffman code :- [2 way opt merge tree]

n-1 times
1. delete 2 min cost tree from the forest of tree. (2 log n) TC
2. Merge into one tree with weight is sum of weights (and) Insert into forest of tree — (log n) TC
3. Repeat 1 & 2 until forest becomes one tree.

---

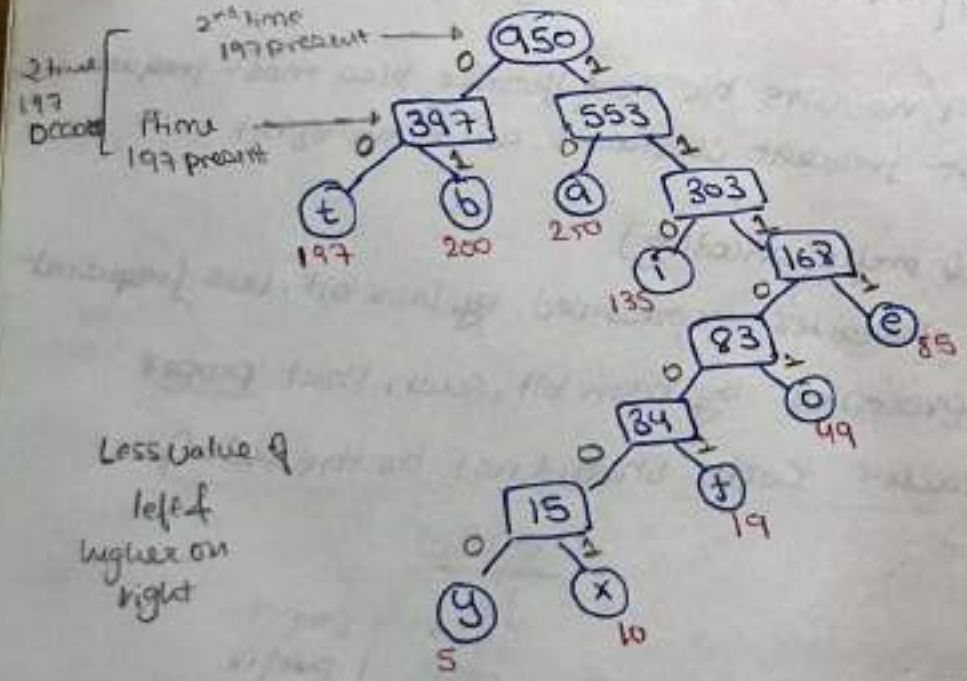
ex: Char: a e i o x b f t y

freq^n : 250 85 135 49 10 200 19 197 5

// Forest { 250 (a), 85 (e), 135 (i), 49 (o), 10 (x), 200 (b), 19 (f), 197 (t), 5 (y) }

Fixed length
$\lceil \log_2 9 \rceil$ = 4bit/char

Sum of frequency count [msg length] = 950 * 4 = 3800 bit



Less value of left higher on right

Char	Huffman Code
a	10 (2bit)
e	1111
i	110
o	11101
x	11110001
b	01
f	111001
t	00
y	11110000 (7bits)

- Huffman coding is a lossless coding & has an exact greedy algo.

Type 1

- Sum of frequency count [msg length] by using huffman coding : $\sum_{i=0}^{n} w_i * d_i$

  w_i = frequency of i^{th} char.
  d_i = depth of i^{th} char.

  = Sum of weights of internal node of Huffman code
  = 15 + 34 + 83 + 168 + 303 + 397 + 553 + 950
  ⇒ 2503 bits.

Type 2

- Avg. bits/char using huffman coding

$$\frac{\text{Sum of frequency counts bit}}{\text{Sum frequency count char}} = \frac{2503}{950} = 2.63 \text{ bit/char}$$

Type 3

- Min bits 2bits Max bits 7bits used for any char by using huffman coding.
- what is encoding bit sequence of word "bat" by using huffman coding & "tiya" of above msg.

Bat {01100} | tiya {0011011100010}

- what is decoding char of given bit sequence above msg.?

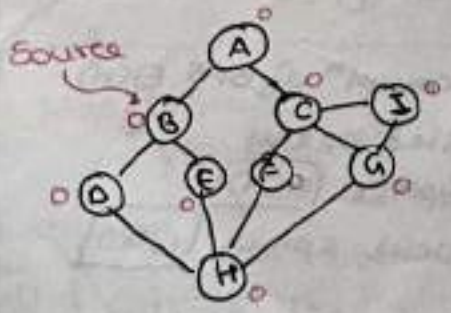
{01/110/1110001/110/1110001} ⇒ "bixix"

b i x l x

Graph Traversal Algorithms [BFS, DFS]

1] Breadth first search Algo: [level order traversal]

Visit source(s) & visit Adj of source followed visit vertices of two edge length from source & so on.



	A	B	C	D	E	F	G	H	I
Visited	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1

Queue

B	A	D	E	C	H	F	G	I

Queue (level order sequence)

Find Adj of — visit

1. (B)
2. (A) D E
3. B C
4. B H
5. B H
6. A F G I

BFS sequence: BADECHFGI

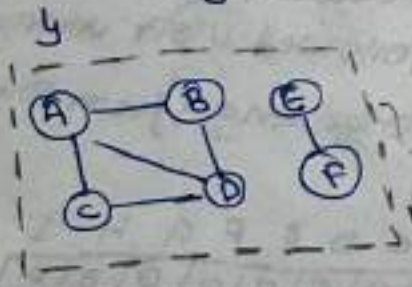
BFS spanning tree:-

Note:
BFS sequence is not unique

```
main(){
    for(i=1; i<=n; i++){
        visited[v_i]=0;
    }
    for(i=1; i<=n; i++){
        if(visited[v_i]==0){ → if it's a disconnected graph
            call BFS(v_i);
        }
    }
}
```

```
Algo BFS(v){ // v is source
    visited[v]=1;
    while(True){
        v = find_Adj(v);   Adj. list → θ(n+e)
                           θ(deg(v))
                           θ(n) Adj Matrix θ(n²)
        for(all vertices of v set){
            if visited[v]==0){
                visited[v]=1;
                Insert_Queue(Q,v);
            }
        }
        If (Queue (Q) is empty) → visit all Adj vertex
            Return (True);
        else
            u=Delete_Queue(Q);
    }
}
```

For all n vertices → Σ(v) sum of degree deg(v)

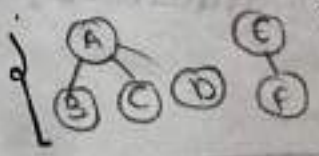


Visited

A	B	C	D	E	F
0	0	0	0	0	0
1	1	1	1	1	1

BFS(A) = {ABCD}

BFS(E) = {EF}



BFS's forest

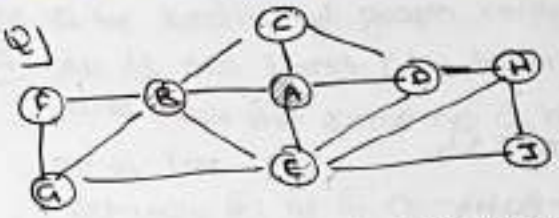
TC of BFS Algo. (n no. of vertices, e no. of edges)

- By using Adj list graph
  $\theta(n+e+e) \Rightarrow \theta(n+e)$
- By using Adj Matrix graph
  $\theta(n^2)$

Space compl^n of BFS $\theta(n)$

- visited array space $\theta(n)$
- queue space $O(n)$

Q]



How many possible BFS sequence from source

A [B C D E] [F G H I]

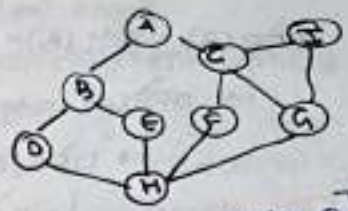
(B C D E) — independent of others

any order

4 possibility

A
- B D E → 3! → [FG, GF] → 2 × [HI] → 2
- B E D → 2 → 2 × [HI, IH] → 4
- D B E → 1 → 2 × [FG, GF] → 2
- D E B → 1 → 2 → 2
- E B D → 3! → [G, H, I] → 6 × [FG] → 6
- E D B → 3! → any order → 6

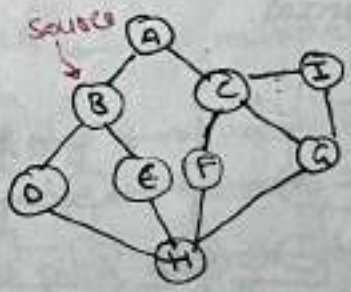
∴ 22*4 = 88 possible BFS sequence



[min: 3 entries Q
max: 5 entries Q]

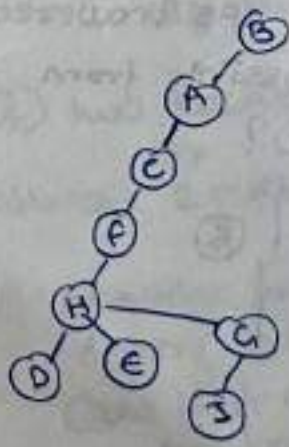
Depth First Search

- traverse each branch as deep as possible before backtracking



Max call = 8
min call = 5

BACFHDEGI
DFS sequence



DFS spanning tree

Depth: 6

```
main(){
Θ(n)[ for(i=1; i≤n; i++)
        visited[v_i]=0;

  for(i=1; i≤n; i++)
    if(visited[v_i]==0)
      call DFS(v_i);
}

Algo DFS(u){
  visited[u]=1
  v: find Adj(u)        deg = Θ(n+e)
                        Θ(n) = Θ(n²)
  for(all vertices of V set)
  deg(u) if(visited[v]==0)
    DFS(v)
  }
}
```

rec call

Θ(e)

DFS(B) Adj(B) = {A, D, E}

↳ DFS(A) Adj(A) = {B, C}

↳ DFS(C) Adj(C) = {A, D, G, I}

↳ DFS(F) Adj(F) = {C, H}

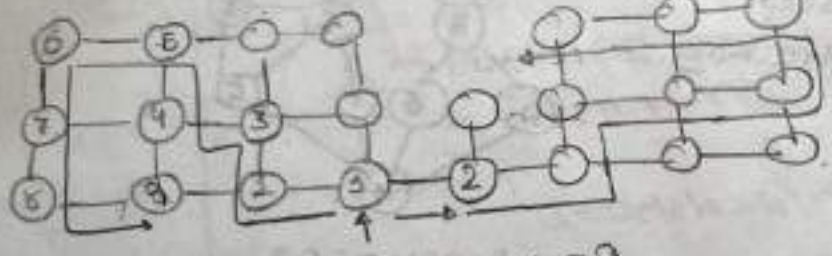
↳ DFS(H) Adj(H) = {D, E, F, G}

- DFS(D) Adj(D) = {B, H}
- DFS(E) Adj(E) = {B, H}
- DFS(G) Adj(G) = {C, H, I}
  - ↳ DFS(I) Adj(I) = {C, G}

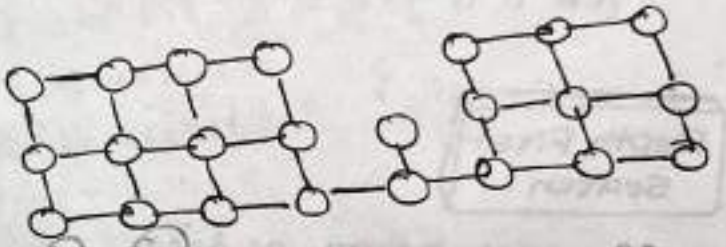
- T.C of DFS Algo
  - ↳ using Adj list graph: $\theta(n+e)$
  - ↳ using Adj Matrix graph: $\theta(n^2)$
- Sc of DFS Algo
  - $\Omega(1)$ to $O(n) = \theta(n)$

Code for DFS ?

ex:- How max/min stack entries to run DFS traversal.

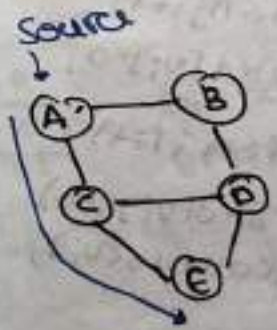


Min stack = 9



Max stack = 22

Ex: How many DFS traversal sequence possible from source (A) ?



A B D C E
A B D E C
A C D B E
A C D E B
A C E D B

(5)

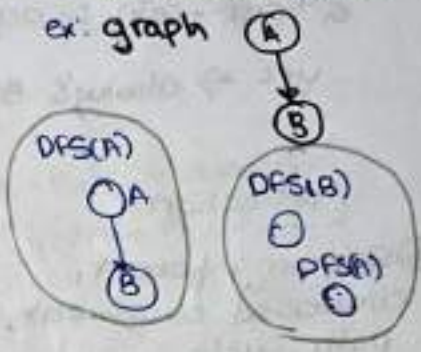
ex. G be undirected graph with 'n' vertices & 'm' edges

(a) All it's DFS forest (for traversal starting at different vertices) will have the same no. of tree? (yes)

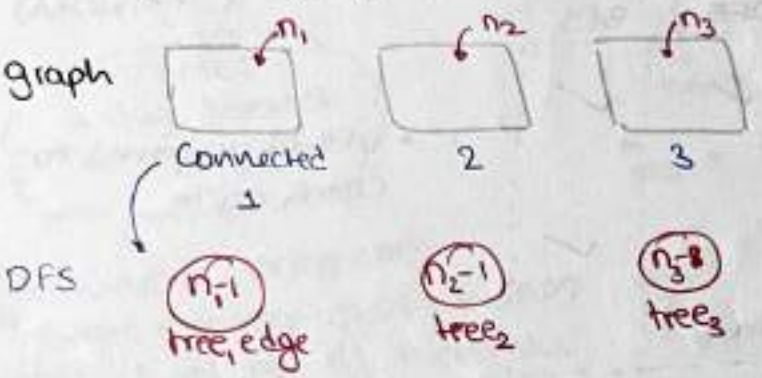
no. of tree determine the no. of connected component

- it will differ when it's directed graph

ex. graph



(b) All it's DFS forest will have the same no. of tree edge & the same no. of back edge?

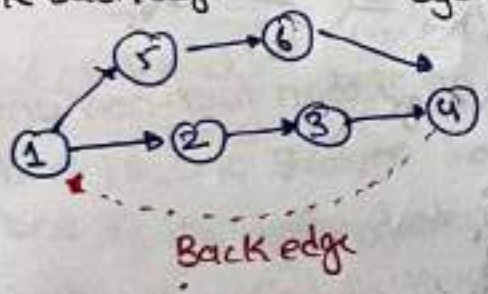


total edge of graph
⇒ tree + back

note
↳ A graph has cycle iff there is a back edge

graph can be directed or undirected

- one Back edge ⟷ multiple cycle



- Back edge ⟷ cycle
- one Back edge ⟷ multiple cycle
- one back edge ⟷ atleast one cycle
- 2 back edge ⟷ atleast 2 cycle
- Can we pick a single edge to destroy all cycles? that edge is always Back edge (or) common edge blw all cycle

(1) one back edge
⇓
(atleast) one cycle
⇓
one edge will always destroy All cycles (that edge will be back edge)
there could be other edge too.

(2) two back edge
⇓
(atleast) 2 cycle
⇓
one edge may or may not destroy all cycles.
(and that one edge can never be back edge)

ex: if DFS of directed graph $G=(V,E)$ produce exactly one back edge then it's possible to choose an edge $e \in E$ such that the graph $G''=(V, E-\{e\})$ is acyclic.

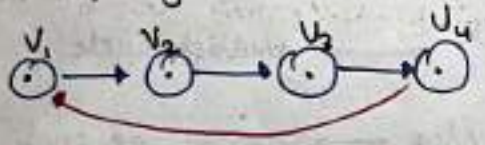
yes ⇒ always's Back edge

- Application of DFS
  - ↳ cycle in the graph [directed or undirected]
  - ↳ Find topological sort of DAG (Directed)
  - ↳ Cut Vertex (or) Articulation point (undirected)
  - ↳ Cut Edge (or) Bridge (undirected)

Application	DFS	BFS
① Spanning forest, connected component, path, cycle	✓	✓
② Shortest path (unweighted graph)		✓
③ Biconnected component	✓	

- DFS is preferred to check cycle

- Back edge → Cycle

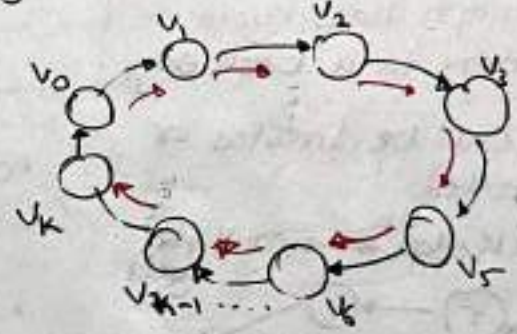


there is a path from any ancestor to descendent using tree edge

&

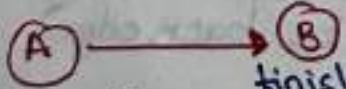
there will be a path from descendent to ancestor using Back edge

- Cycle → Backedge



Back edge

- In a DAG, we will always have: for any edge A → B



finish time [larger] than B

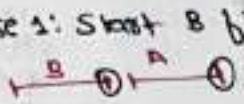
finish time [smaller] than A

Intuitively:

you can not gaurantee from S, you choose A first or B first

you don't know relationship b/w start time

Case 1: Start B first



Start B → end B

Start A → end A

Case 2: Start A first

Start A

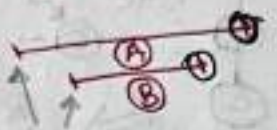
Start B

end B

end A

[b/c B is reachable from A]

B is unvisited & B is reachable



Start time

- in both the case the finish time is more for A.

note: A dag doesn't contain cyclic

Articulation point (AP)

ex: when can root of DFT be AP?

- root is an AP iff there are atleast 2 children of root

- only one child of root → root is not A·P

ex: when can non-root of DFT be AP?

a non-root node will be A·P iff none of it's descendent are directly connected to it's ancestor

i.e (6) is a A·P

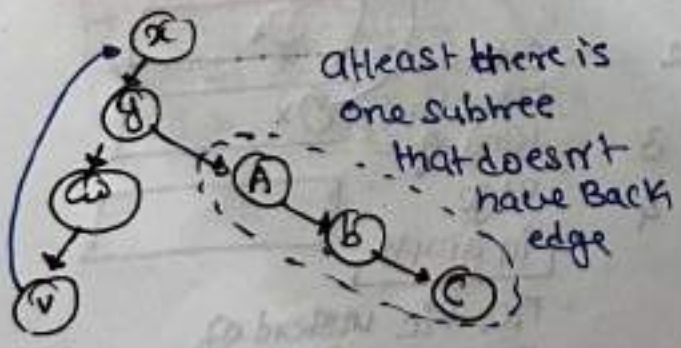
Slightly wrong

Correct one

- any (internal) non-root node is an A·P iff there is atleast one subtree such that none of it's descendent are directly connected to it's ancestor. (Back edge)

- A leaf node can never be A·P

ex:

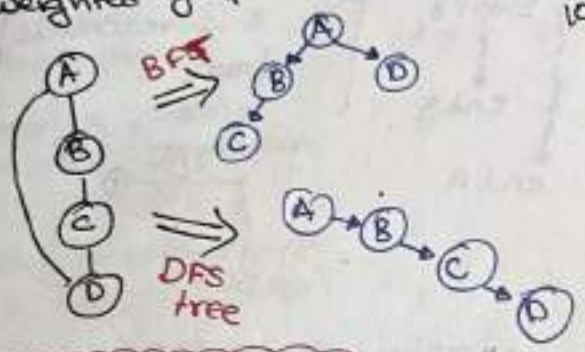


atleast there is one subtree that doesn't have Back edge

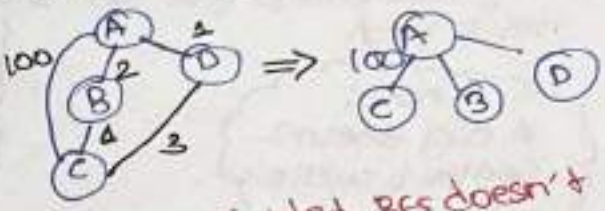
note:
DFS tree are long & skinny
while BFS tree are short & fat

DFS & BFS are twins
DFS → stack
BFS → queue
instead of queue if we use priority queue it will become dijkstra Algo

- BFS for shortest path in unweighted graph (only)



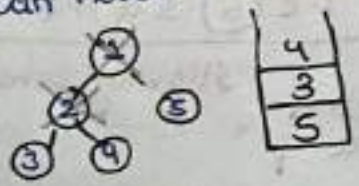
but in weighted graph



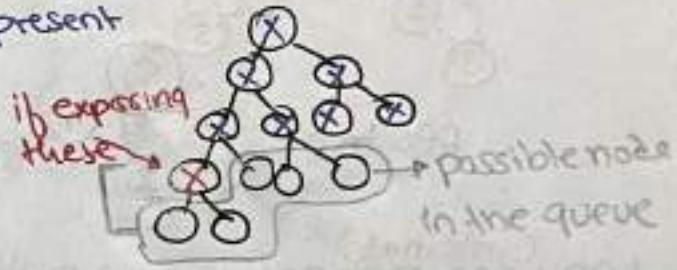
in weighted BFS doesn't give you shortest path b/c it doesn't take care about weight

note: BFS will always give you shortest path (unweighted graph)

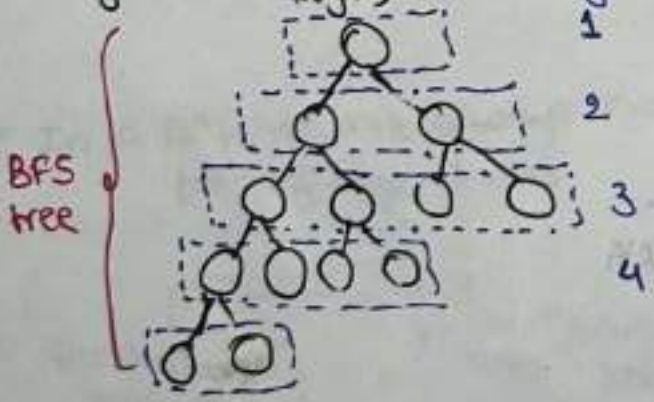
- BFS Queue state (frontiers)
just by looking at the partial BFS tree we can say about queue state ⇒ ① Internal node can never be in queue



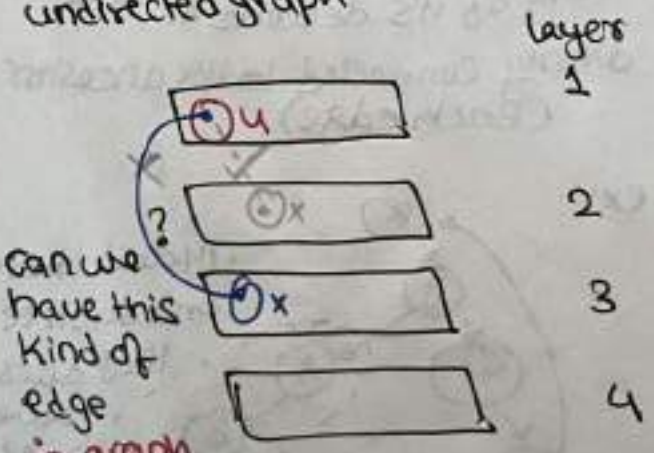
② only last two levels leaves could be present



- BFS as Layer (possible edge in b/w layer)

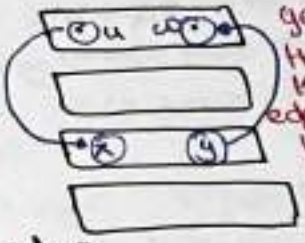


undirected graph



no, b/c instead of present in ③ it would be in layer ②

Directed graph



no, there kind of edge will still not be present (Forward edge)

yes, there kind of edge can be present (backward edge)

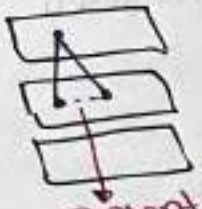
- Skip level edges in BFS graph
  - undirected ⇒ no skip level edge
  - in directed ⇒ From upper to lower

From lower to upper (yes) : (back edge)

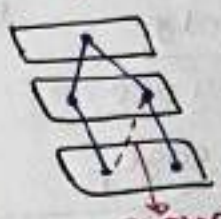
lower (no) : (forward edge)

- within level edge in (directed/undirect) are present

cross edge



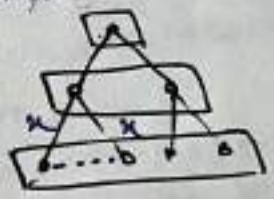
present



present

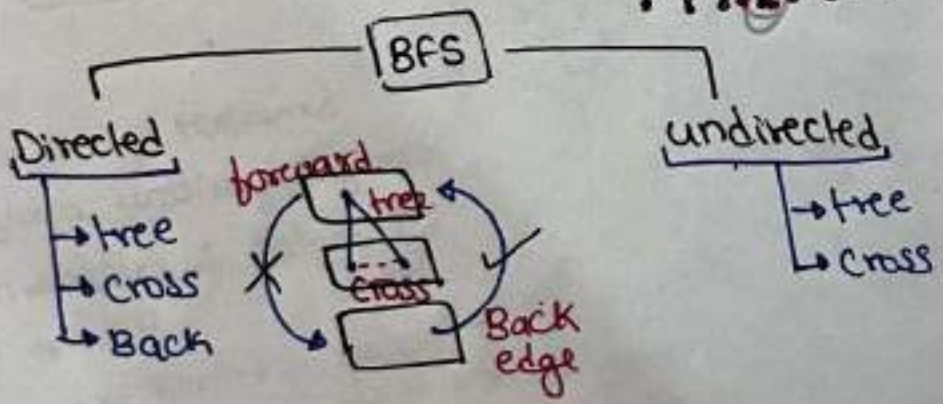
- Cross edge
  - → within level
  - → across (continious) level

- Cross edge [undirected graph cycle due to cross edge]
  - → within level (odd length cycle)



2x+1 odd length

- → across (continious) level → form's even length cycle (2x)



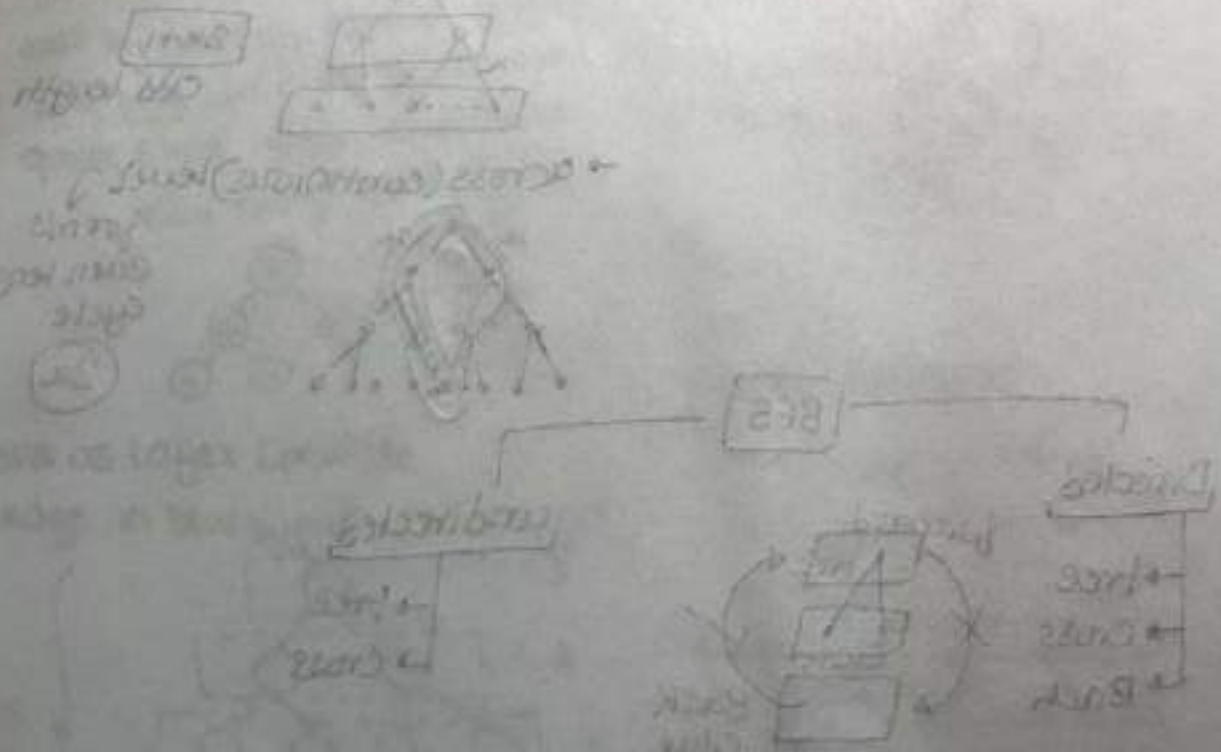
• BFS Application (in undirected graph)

note: tree is always Bipartite

① Bipartite graph : a graph is Bipartite iff there is no odd length cycle

ex: given a undirected graph? check if it's Bipartite or not

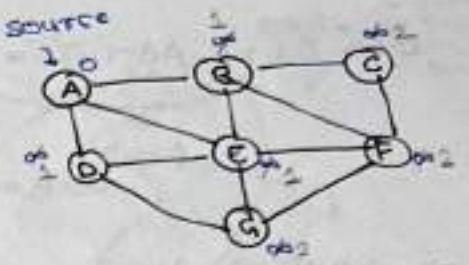
- if graph has no cross edge then it's a Bipartite graph
  - only tree edge
  - then graph is a tree
- if graph has cross edge,
  - in b/w level
    - it create odd length cycle so, no Bipartite graph
  - across different level [i to i+1]
    - it create even length cycle so, it can create Bipartite graph



Ques] what is TC required for finding shortest path dist for every vertex from given source(s) over unweighted graph?

a) $\Theta((n+e)\cdot \log n)$ b) $\Theta(n \cdot e)$

c) $\Theta(n+e)$ d) $\Theta(n)$



dist

A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞

prev

A	B	C	D	E	F	G
-1	A	B	A	D	E	F

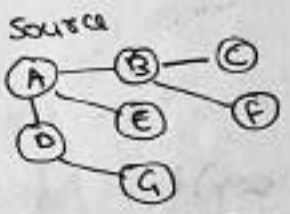
visited

A	B	C	D	E	F	G
0	0	0	0	0	0	0

- Distance of (u,v) in DFS $\geq$ Distance of (u,v) in BFS

** [BFS Algo can used as single source shortest path algo over unweighted graph]

T.C :. $\Theta(n+e)$ (or) $\Theta(n^2)$



B D E G A F G

Queue

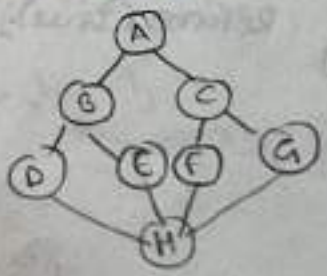
Edge Classification of graph traversal

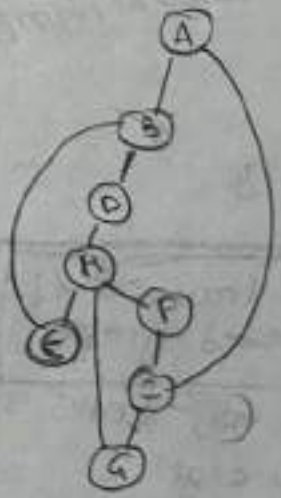
① Edges classification DFS traversal of undirected graphs:

1] tree edges [DFS ST edges] (n-1)

2] Back edges → e-(n-1) = e-n+1 } Connected graph

- no. of back edges in DFS traversal of Connected undirected graph or e-n+1





code for this?

i.e what is TC required to test given undirected graph with n vertices & edges is cyclic or not?

a) $\Theta(n^2)$ ~~b) $\Theta(n)$~~ → to find if cycle present or not

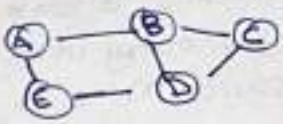
c) $\Theta(n+e)$ d) $\Theta(n \cdot e)$

to count Back edge

- if there exist a Back edge during DFS traversal of undirected graph than the graph is cyclic.

DFS (u, x) · Adj(u) = {v ..... }
v visited

(u,v) back edge



DFS(A ->) Adj(A) = {B, E}
↳ DFS(B,A) Adj(B) = {A, C, D}
↳ DFS(C,B) Adj(C) = {B, D}
↳ DFS(D,C) Adj(D) = {B, C, E}

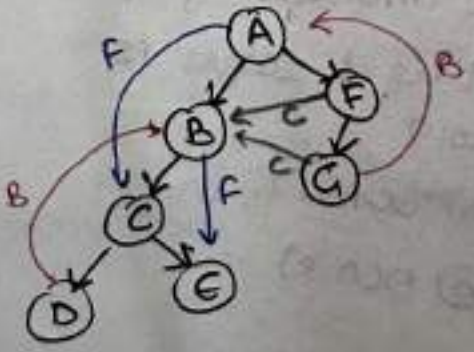
Return (true)
Cycle exist.

```
DFS(U,P){
    visited[U] = 1
    V : And_Adj(u)
    for(all vertices v set){
        if(visited[V]==0)
            DFS(V,U);
        else if(P!=V){
            (u,v) back edges
            G is cyclic
            Return (True);
        }
    }
}
```

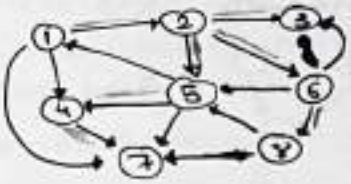
(2) DFS traversal of Directed graph

1. tree edge [DFS ST edges]
2. back edge
3. cross edge
4. forward edge

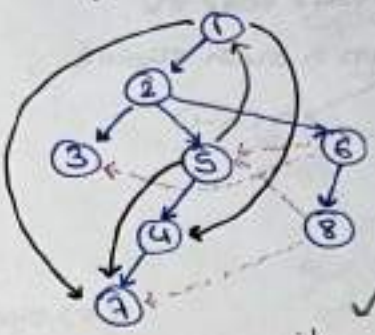
Assume DFS ST



ex:



How many T, B, C, F edges in DFS Traversal of given Directed graph from Source ① And Adj Selected priority of ASC order of vertex lable?



edge { = Forward = Tree = Cross = Back

T : 7
B : 1
C : 4
P : 3

if present then forms cycle

- If there exist a Back edge in DFS traversal of directed graph then graph G is cyclic

colour [v] = { white ; if v not visited (not begins explanation
Gray ; if v is begins explanation
Black ; if v finished explanation

while { 1 2 3 4 5 6 7 8 }

gray { 1, 2, 3, 5 }

Black { 3 4 }

v → p : back edge
graph G cycle

DFS(v) Find Adj(v) = { v } ← white
↳ DFS(v) Adj(v) = { p } ← gray

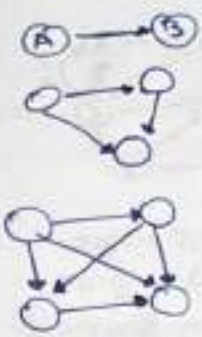
Color [v;]
DFS(v) {
Color[v] = Gray
v = Find Adj (v)
for(all vertices of v adj) {
if (color [v] == white) DFS(v);
else if (color [v] == gray) {
G is cyclic
(v, v) is back edge
Return (True);
} } }

WC
TC: $\Theta(n+e)$
(or)
$\Theta(n^2)$

Q: How many max edge in Directed graph n such should not cycle? → $n(n-1)$

↓

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



Q: X_s exploration begins, X_e explanation end of vertix x in DFS traversal of Directed graph what is explanation beginning end sequence for the vertices u, v. Such that there exist edges (u, v)

where

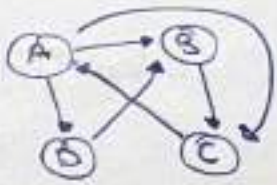
(i) (u,v) Tree edges. : $u_s < v_s < v_e < u_e$

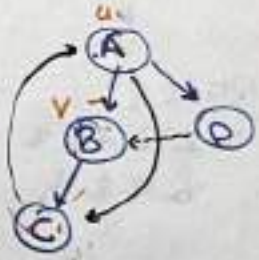
(ii) (u,v) back edges: $v_s < u_s < u_e < v_e$

(iii) (u,v) forward edges: $u_s < v_s < v_e < u_e$

(iv) (u,v) Cross edges. $v_s < v_e < u_s < u_e$

only when new path to complete path





3 BFS Traversal of undirected

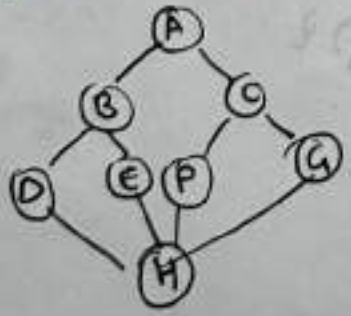
(no forward & back edge here)

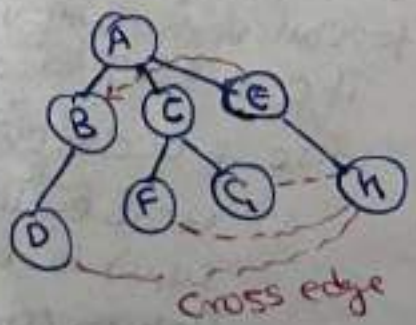
Graph edge classification.

(1) Tree edges [n-1]

(2) Cross edges [e-n+1]

- no of cross edge in BFS traversal of undirected connected graph are [e-n+1]





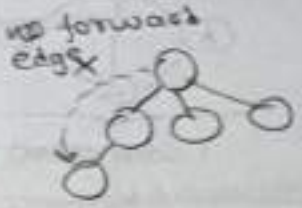
- if there exist cross edge in BFS traversal of undirected graph then graph is cyclic
  Otherwise Acyclic

T.C : $\Theta(n)$
[for checking of cycle]

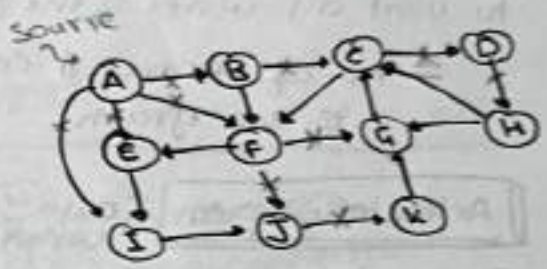
(4) **BFS traversal Directed graph**

edge classification :-

(1) Tree edges (T)
(2) back edges (B)
(3) cross edges (C)

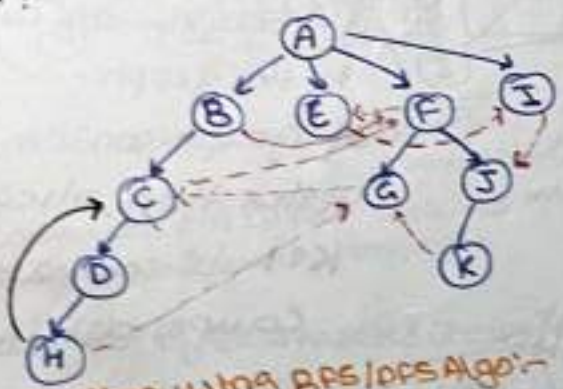


How many T, B, C edges in BFS traversal of given Directed graph from sources vertex (A) & Adjacencies selected based on lexiographic order of vertex label?



Tree edge : 10
Back edge : 1
Cross edge : 8

BFS



Note

BFS Algo not preferred to test directed graph is cyclic or not b/c Back edge guarantn for ani kind cycle but even no Back edge, still graph can be cyclic

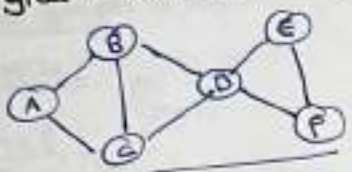
graph Application using BFS/DFS Algo :-

Undirected Graph
- ↳ Connected components
- ↳ Bi-connected component

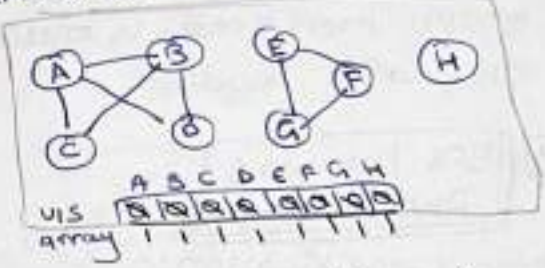
Directed graph
- ↳ weakly connected component
- ↳ strongly connected component
- ↳ uni-laterly connected comp.

TC:
$O(n+e)$ or $O(n^2)$ (to test no. of components)
using DFS, BFS

- TC to test # of connected component of given undirected graph is $\Theta(n+e)$ or $\Theta(n^2)$



1 connected component



- no. of DFS or BFS call required to visit all vertex of the graph = no. of connected component of the graph

DFS(A) = {A B D C}
BFS(E) = {E F G}
DFS(H) = {H}

- G has 3 connected component

Articulation point (Cut vertex)

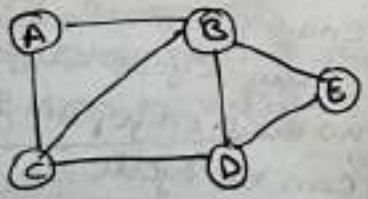
- Removal of vertex is disconnect given undirected graph

Bi-Connected Component

- given undirected graph is single Bi-connected component if & only if no cut vertex in the graph
- If graph consist K cut vertex then atleast K+1 bi-connected component in the graph

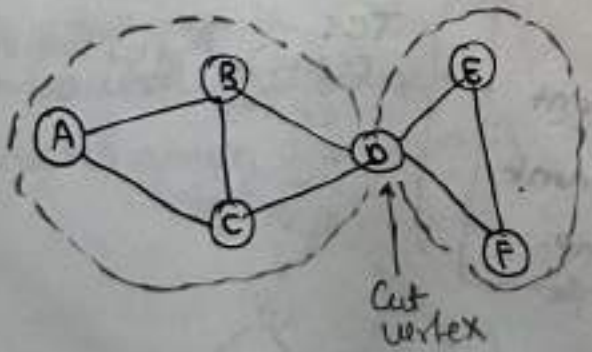
- one Bi-connected component is maximum possible subgraph which has no cut vertex

(1)



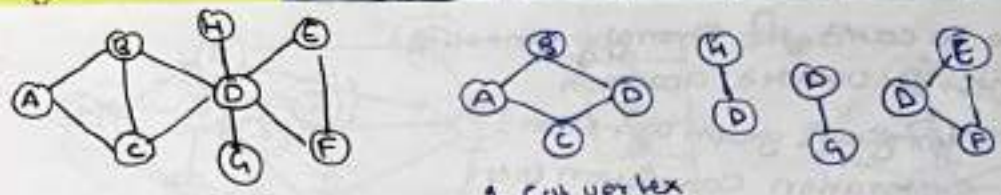
- no cut vertex
- G is single Biconnected component

(2)



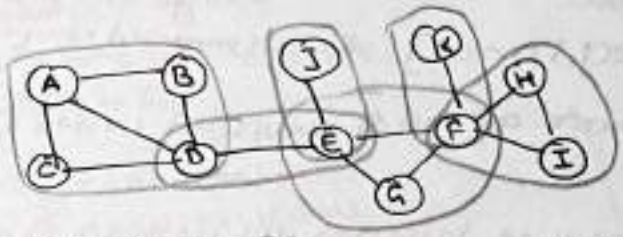
no. of
(1) Cut vertex
(2) Bi-connected component.

(3)



1 cut vertex
4 Biconnected components.

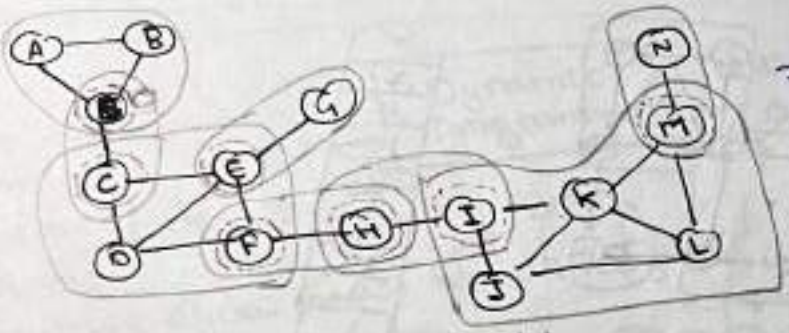
(4)



3 cut vertex
6 Bi-connected components

- we can't take EF individual bcoz we need max subgraph

(5)



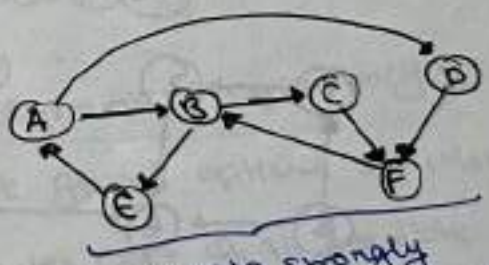
7 cut vertex
8 Biconnected components.

- if n cut vertex then n+1 Biconnected component
- if no common cut vertex then no more than 2 component from that

- TC to test no. of Biconnected Component in a graph using DFS/BFS $\theta(n^2)$

Strongly Connected Component

- given directed graph is singlely strongly connected component if for all pair of vertice u and v such that there must be path from u to v and v to u.



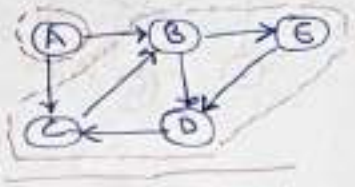
Single strongly Connected component

- Procedure to count #f Strongly Connected components of Directed Graph G

1] call DFS for given graph G & store vertex exploration completion order in stack[S]

2] build Transpose graph G^T
[edge (i,j) in G is edge(j,i) in G^T]

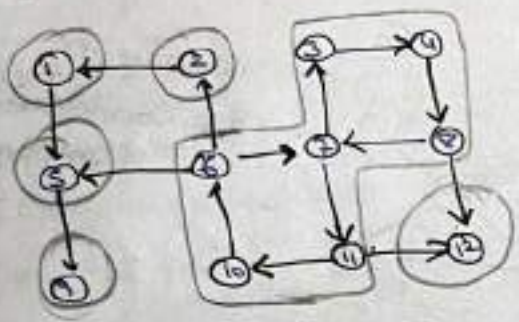
3] call DFS for G^T by using source as top of unvisited vertex of stack[S]

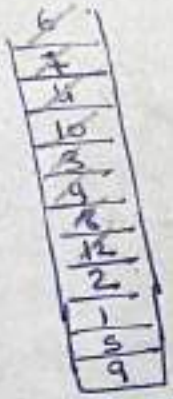


2 single strongly connected component

Result:- #f times DFS calls required to visited all vertices of G^T = #f Strongly connected component of given graph G

ex:- G:



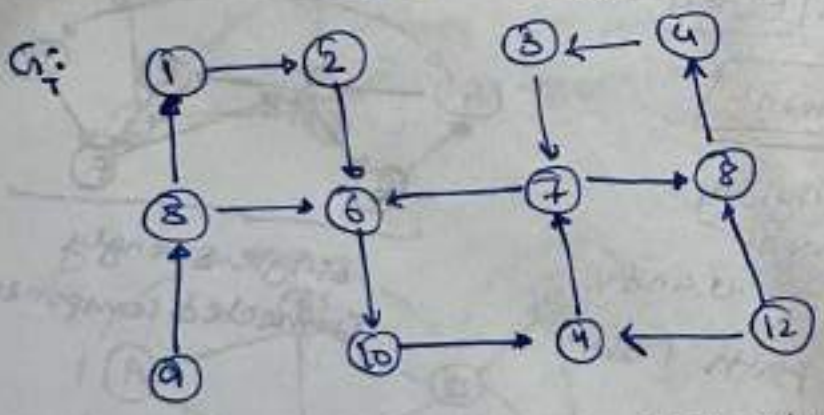


Stack

DFS(1) → 1_s 5_s 9_s 9_e 5_e 1_e

DFS(6) → 6_s 2_s 2_e 7_s 3_s 4_s 8_s 12_s 12_e 8_e 4_e 3_e 11_s 10_s 10_e 11_e 7_e 6_e → put in stack

pop from the stack



DFS(6) = ? 6, 10, 11, 7, 8, 4, 3 ✓

DFS(12) = ? 12 ✓

DFS(2) = ? 2 ✓

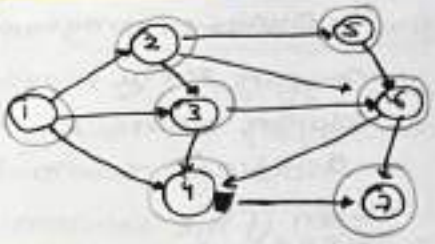
DFS(1) = ? 1 ✓

DFS(5) = ? 5 ✓

DFS(9) = ? 9 ✓

6 Strongly Connected Component in graph

ex

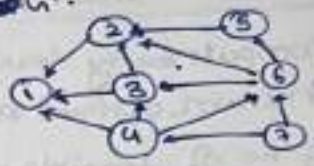


Sequence
$1_s 2_s 8_s 4_e 7_s 7_e 4_e 6_s 6_e 3_e$
$5_s 5_e 2_e 1_e$

1
2
5
3
6
4
7

Stack

GT:



DFS(1) = {1} DFS(2) = {2}
DFS(3) = {3} DFS(4) = {4}
DFS(5) = {5} DFS(6) = {6}
DFS(7) = {7}

7 strongly connected Component in given graph

- weakly connected
  either U to V or V to U

ex:

} weakly connected

Dynamic programming [Also called principle of optimality design technique]

Optimization problem:
Sequence of decision problems
[sequence of dec must be optimal]

- principle of optimality
  whatever initial state & decision cost of remaining decision must be optimal, state result from 1st decision
  [the result is global optimal]

Bellmanford

1. recursive eqn
2. general rec. Rel with base condition
3. Implementation of DP algo by using extra space
4. Compute opt sol in bottom up approach

Greedy method	Dynamic Programming
• choose each decision irrespective cost of remaining decision [local optimal]	• choose global optimal solution
• Computes cost of one feasible solution based on greedy strategy	• Computes cost of all feasible solutions & returns optimals solution
• Always runs in polynomial time	• may required exponential time
• not every optimization problem can solve by greedy method	• every optimization problem can solve using DP

Brute force Algo.

- Compute every feasible solution & return optimal solution
- If no of feasible solution [solution space] to solve optimization problem is exponential then Brute force Algo" TC to solve the problem is also exponential
- less space complexity

2^n or $n!$: $\Theta(2^n)$ or $\Theta(n!)$
TC of BFA

Dynamic program"

- Compute every feasible solution recursively by avoiding recomputation
- even if the solution space is exponential, DP may solve the problem in polynomial time by avoiding re computation.
- more space complexity.
- to write code we use top down
- while solving we use bottom up

① nth Fib num:-

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

Depth of Recur" is n level

rec. calls atmost CBT of n level : $\Theta(2^n)$

- TC of Fib(n): $\Theta(2^n)$
- SC of Fib(n): $\Theta(n)$

```
Algo Fib(n){
    if(n≤1) return (1);
    else Return( Fib(n-1)+Fib(n-2));
}
```

- DP nth Fib Num:

```
main(){
    for(i=0; i≤n; i++) {a[i]=-1;}
    Fib(n);
}
Algo 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]);
    }
}
```

(n+1) rec calls
each rec $\Theta(1)$ time

TC of nth Fib : $\Theta(n)$ using DP

SC of nth Fib : $\Theta(n)$ using DP

Stack : $\Theta(n)$
+
a[0...n] : $\Theta(n)$

Imp

Matrix Chain Multiplication problem [MCP]

1. $A_{p \times q}$ $B_{q \times r}$ Cost of matrix :- $p*q*r$
   [# of element multiplication]

2. Associativity applicable for matrix mul.
   A,B,C matrix $\Rightarrow$ $(A*B)*C = A*(B*C)$

ex: $A_{10 \times 50}$ $B_{50 \times 6}$ $C_{15 \times 40}$

① $(A*B)*C$ (10×15)
- $10 \times 50 \times 15 = 7500$
- $10 \times 15 \times 40 = 6000$
- 13500

ex: $A*(B*C)$
- $50 \times 15 \times 40 = 30000$
- $10 \times 15 \times 40 = 6000$
- 36000

n=4 A_1 (2×5) A_2 (5×8) A_3 (8×3) A_4 (3×1)

Problem Stmt MCP :-

Given n matrix $A_1 A_2 \cdots A_n$ of order set $\{P_0, P_1, P_2 \cdots P_n\}$ integer such that order of A_i is $P_{i-1} \times P_i$

parenthesize given n matrix such that minimize cost of multiplication of matrices.

Min. cost for multiplication of matrix.

5 feasible solution for n=4:
1. $A_1(A_2(A_3A_4))$
2. $A_1((A_2A_3)A_4)$
3. $(A_1A_2)(A_3A_4)$
4. $(A_1(A_2A_3))A_4$
5. $((A_1A_2)A_3)A_4$

ex: A_1 (2×5) A_2 (5×8) A_3 (8×3) A_4 (3×1)

① $A_1(A_2(A_3A_4))$ (A_3A_4: 8×1, $A_2(A_3A_4)$: 5×1, $A_1(\cdots)$: 2×1)
$= 24 + 40 + 10$ (8×3×1, 5×8×1, 2×5×1)
$= 74$

② $A_1((A_2A_3)A_4)$ (A_2A_3: 5×3, $(A_2A_3)A_4$: 5×1, $A_1(\cdots)$: 2×1)
$\Rightarrow 120 + 15 + 10 = 145$ (5×8×3, 5×3×1, 2×5×1)

③ $(A_1A_2)(A_3A_4) = 80 + 24 + 16$ (A_1A_2: 2×8, A_3A_4: 8×1)
$= 120$

④ $(A_1(A_2A_3))A_4 = 120 + 30 + 6 = 156$ (A_2A_3: 5×3, $A_1(A_2A_3)$: 2×3)

⑤ $((A_1A_2)A_3)A_4 \Rightarrow 80 + 48 + 6$
$\Rightarrow 134$

Given n matrix		#ℓ parthenization of matrix Mult"
n=1	A	1
n=2	$A_1 * A_2$	1
n=3	$A_1A_2A_3$	2
n=4	$A_1A_2A_3A_4$	5
n=5	$A_1A_2A_3A_4A_5$ 1*5+1*2+2*1+5*1	14
n=6	$A_1A_2A_3A_4A_5A_6$ 1*14+1*5+2*2+5*1+ 14*1	42

Catalan series

T(n) = #ℓ parathnesis to multiply n matrices

T(1) = 1
T(2) = 1

$A_1^{k=1} \cdot A_2^{k=2} \cdot A_3 \cdots A_{n-1}^{k=n-1} \cdot A_n$

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

$$T(n) = \begin{cases} 1 & , n=1 \\ 1 & , n=2 \\ \sum_{k=1}^{n-1} T(k) * T(n-k) & , n \geq 2 \end{cases}$$

T(n) — no of feasible solution to solve Matrix multi"

Brute force.

$$T(n) = \frac{1}{n} * \frac{(2n-2)!}{(n-1)!(n-1)!} > 2^n$$

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

#ℓ feasible solution to Soln [MCP]

- Greedy method failed to solve MCP
- Brute force Algo. TC to solve MCP is $\Omega(2^n)$ (exponential.

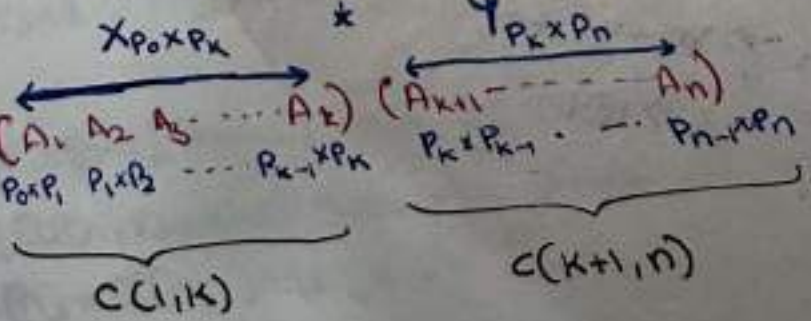
DP MCP Given $A_1 A_2 A_3 \cdots A_n$ matrices

$P_0\ P_1\ P_2 \cdots P_n$ order

C(1,n): min cost of Mul of $A_1, A_2, A_3, \cdots A_n$ matrix

C(1,n): Recursive function

Consider as optimal parthenization



$$C(1,n) = \min_{1 \le K < n} \{ C(1,K) + C(K+1,n) + P_0 P_K P_n \}$$

Generalize rec Rel of MCP:-

$A_i\ A_{i+1} \cdots A_j$: Matrices

$P_{i-1}\ P_i\ P_{i+1} \cdots P_j$: order

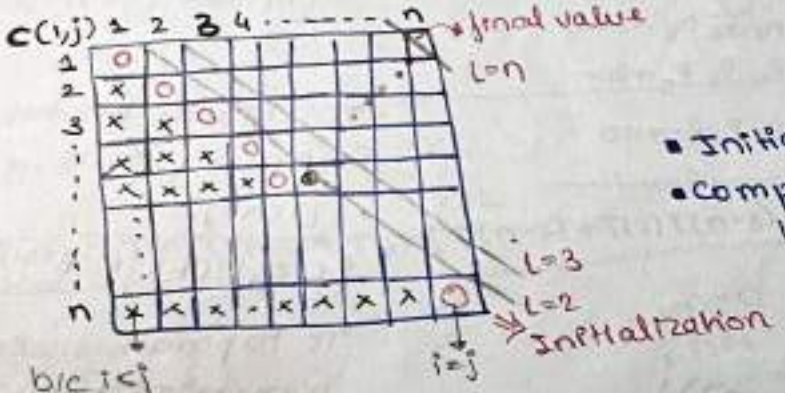
$$C(i,j) = \begin{cases} \min_{i \le K < j} \{ C(i,K) + C(K+1,j) + P_{i-1} P_K P_j \}, & i < j \\ 0 \text{ // only one matrix in subproblem}, & i = j \end{cases}$$

$C(1,n)$ // final value

avoid recomputation using DP

- Implementation DP MCP:-

2D array $C[1 \ldots n, 1 \ldots n]$, $K[1 \ldots n, 1 \ldots n]$



- no need to initialize the whole array with 0, just base condition [diagonal element]
- Initialize diagonal by 0s $C[i,i]=0$
- Compute bottom up
  $L=2, 3, \ldots, n$
  Sub problems
- Return $C[1,n]$ final value.

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

$\frac{n(n+1)}{2}$ distinct subproblem

$\Rightarrow \Theta(n^2)$ //

*** Q] $A_1 A_2 A_3 A_4 A_5$ matrices

order {P_0 P_1 P_2 P_3 P_4 P_5} =

{4 8 6 9 3 10}

How many min Mul to multiply given 5 matrices?

what is optimal parathesization?

C(i,j)	1	2	3	4	5
1	0	192 K=1			
2	x	0	432 K=2		
3	x	x	0	162 K=3	
4	x	x	x	0	270 K=4
5	x	x	x	x	0

$$C(1,2) = \begin{cases} K=1 & C_{11} + C_{22} + P_0 P_1 P_2 \\ & 4 \times 8 \times 6 \\ 0 & \end{cases}$$

$\rightarrow 192$

$C(2,3) = \{ K=2\ C_{22} + C_{33} + P_1 P_2 P_3$

$= 432$

$C(3,4) = 162$

$C(4,5) = 270$

$C(1,3) = \begin{cases} K=1 & C_{11} + C_{23} + P_0P_1P_3 \\ K=2 & C_{12} + C_{33} + P_0P_2P_3 \end{cases}$ 408

$C(2,4) = \begin{cases} K=2 & C_{22} + C_{34} + P_1P_2P_4 \\ K=3 & C_{23} + C_{44} + P_1P_3P_4 \end{cases}$ 306

$C(3,5) = \begin{cases} K=3 & C_{33} + C_{45} + P_2P_3P_5 \\ K=4 & C_{34} + C_{55} + P_2P_4P_5 \end{cases}$ 342

$C(1,4) = \begin{cases} K=1 & C_{11} + C_{24} + P_0P_1P_4 \\ K=2 & C_{12} + C_{34} + P_0P_2P_4 \\ K=3 & C_{13} + C_{44} + P_0P_3P_4 \end{cases}$ 402

$C(2,5) = \begin{cases} K=2 & C_{22} + C_{35} + P_1P_2P_5 \\ K=3 & C_{23} + C_{45} + P_1P_3P_5 \\ K=4 & C_{24} + C_{55} + P_1P_4P_5 \end{cases}$ 546

$C(1,5) = \begin{cases} K=1 & C_{11} + C_{25} + P_0P_1P_5 \\ K=2 & C_{12} + C_{35} + P_0P_2P_5 \\ K=3 & C_{13} + C_{45} + P_0P_3P_5 \\ K=4 & C_{14} + C_{55} + P_0P_4P_5 \end{cases}$ 522

$(n-1)$

to find min from $(n-1)$ is $\Theta(n)$

Algo MCP(n, P[0....n])

```
for(i=1; i<=n; i++)
    c[i][i]=0;
for(l=2; l<=n; l++)                                   Θ(n)
    for(i=1; i<=n-l+1; i++)                           Θ(n)
        j=i+l-1;
        c[i][j] = min_{i<=k<j} { c[i][k] + c[k+1][n] + p[i-1]·p[k]·p[j] }   Θ(n)
Return(c[1][n]);
```

TC: $\Theta(n^3)$

- Min Matti to multiply $A_1A_2A_3A_4A_5$: 522
- Opt. parathesization

$(A_1A_2A_3A_4)A_5$

c[1,5] for k=4

$A_1(A_2A_3A_4)$

c[1,4] for k=1

$A_2(A_3A_4)$

c[2,4] for k=2

$(A_1(A_2(A_3A_4)))A_5$

- TC required to solve MCP using DP is $\Theta(n^3)$

? Sc of MCP: $\Theta(n^2)$

TC to parathesization 'n' matrice by using k[1...n][1...n] array which is calculate already is $\Theta(n)$.

⇒ #f Binary search tree for n distinct key:

		#f BTS
n=0		1
n=1	k_1	1
n=2	$k_1 < k_2$; k_2 / k_1 ; k_1 / k_2	2
n=3	$k_1 < k_2 < k_3$; k_2 / k_1 k_3 ; k_1 / k_3 / k_2 ; k_3 / k_1 k_2 ; k_3 / k_2 / k_1 ; k_3 / k_1 / k_2	5
n=4	$k_1 < k_2 < k_3 < k_4$	14
⋮		
n		$\frac{1}{n+1} \cdot {}^{2n}C_n$

5 = 1*2 + 1*1 + 2*1

14 = 1*5 + 1*2 + 2*1 + 5*1

MCP

DP		Brute force
TC	SC	TC
$O(n^3)$	$\Theta(n^2)$	$\Omega(2^n)$

$O(n^3)$	$\Theta(n^2)$	$\Omega(2^n)$
TC	SC	TC
DP		Brute force

Optimal BST Construction.

no. of ways to multiply 'n' matrices = no of ways to put n-1 balanced parenthesis = Stack permutation

$k_1\ k_2\ k_3 \ldots\ldots\ldots k_n$

$$T(n) = T(0)T(n-1) + T(1)T(n-2) + T(2)T(n-3) + \cdots + T(n-1)T(0)$$

($k=1$, $k=2$, $k=3$, $k=n$)

$$T(n) = \begin{cases} 1 & , n=0 \\ 1 & , n=1 \\ \sum_{k=1}^{n} T(k-1) * T(n-k) & , n>1 \end{cases}$$

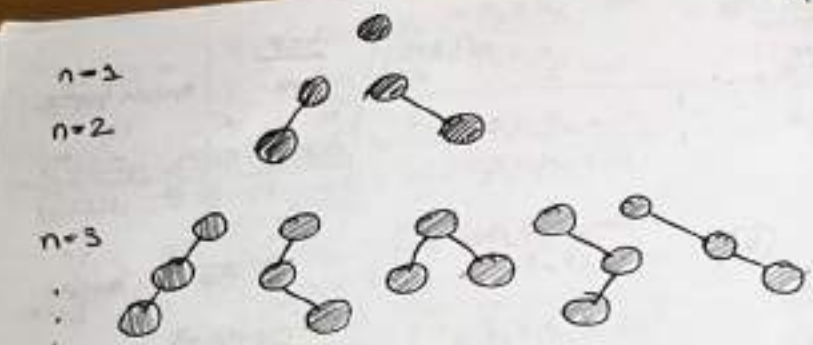
exponential

$$T(n) = \frac{1}{n+1} * \frac{(2n)!}{n! \cdot n!} = \frac{1}{n+1} = {}^{2n}C_n$$

{ #f BST's for n distinct key } = { #f parenthesizations to multiply (n+1) matrices } $= \frac{1}{n+1} \cdot {}^{2n}C_n$

no. of unlabled Binary tree for n nodes :-

{ #f unlabled Binary tree for n nodes } = { #f BST's for n distinct key } $= \frac{1}{n+1} \cdot {}^{2n}C_n$



#f unlabled BT
1
2
5
$\frac{1}{n+1} \cdot \frac{(2n)}{n! \cdot n!}$

#f labled BT's
1
2×1 = 2
5×3×2 = 30
$\frac{1}{n+1} * \frac{(2n)!}{n!}$

{#f labled BT's for n nodes} = {#f unlabled BT's for n nodes} * n!

$\Rightarrow \frac{1}{n+1} \cdot {}^{2n}P_n$

All pair shortest path Algo

- Graph G with n vertices & e edges
  Cost [1...n][1...n] are edges
  Cost such that

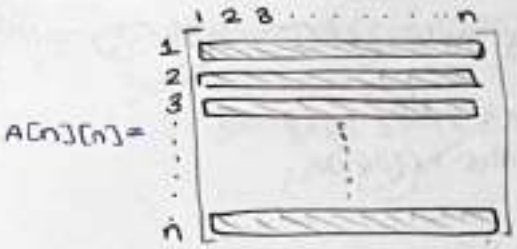
$$cost[i][j] = \begin{cases} 0 & \text{if } (i=j) \\ \text{edge cost} & \text{if } (i,j) \in \text{edge} \\ +\infty & \text{if } (i,j) \notin \text{edges} \end{cases}$$

→ All pair shortest path algo. should compute shortest path distance for every pair of vertices in the graph i.e should compute 2D array A[x][x] such that

A[i][j] = Shortest path dist. from (i) to (j) vertex

Ques] How many way can labled given/unlabeled Binary tree of n distinct keys such that result must be a Binary search tree?

(a) $\frac{1}{n+1} \cdot {}^{2n}C_n$
(b) $\frac{1}{n+1} \cdot {}^{2n}P_n$
(c) n!
(d) 1



A(i,j) is shortest path dist from vertex i to j

- All pair shortest-path Algo by using single source sp algo.
- By using n invocation (execution) of Dijkstra or Bellman ford can compute All pair Shortest path result.

```
Algo APSP (G,n,e,cost[][])
  for(i=1; i≤n; i++)
    A[i][1...n] = Dijkstra(G,n,e,cost[][],i);
  Return(A[][])
}
```

(i: source)

- TC of APSP Algo by using n-Invocation of **Dijkstra** (weighted graph with no negative edges; tot of operation of priority queue; not work for -ve edges)
  - $\Theta(n \cdot (n+e) \cdot \log n)$ — using min heap
  - $\Theta(n^3)$ — using array
- TC of APSP Algo by using n-Invocation of **Bellman-ford Algo** (weighted graph with may consist -ve edges)

  $\Theta(n^2 \cdot e)$
- TC of APSP Algo for n-invocation of **unweight graph** BFS Algo : $\Theta(n \cdot (n+e))$ or $\Theta(n^3)$

APSP Algo by DP

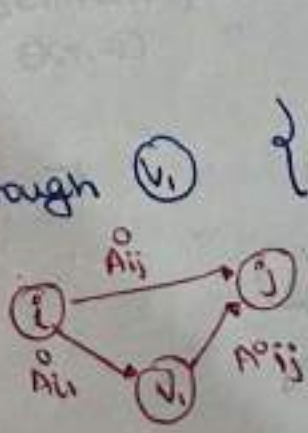
[floyd warshall Algo]

A[1....n, 1....n] 2D array

A^0_{ij} = cost[i,j] // initialization

A^1_{ij} = SP dist from i to j going through v_1

$A^1_{ij} = \min\{A^0_{ij}, A^0_{i1} + A^0_{1j}\}$



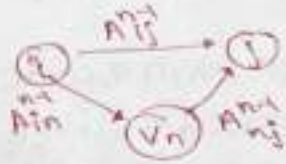
A^0_{ij} = SP dist from i to j going through (V_1) and/or (V_2)

$A^2_{ij} = \min\{A^1_{ij}, A^1_{i2} + A^1_{2j}\}$

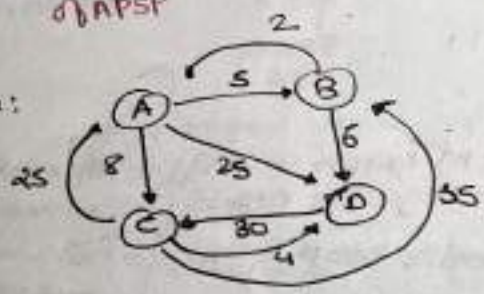
A^n_{ij} = SP dist from i to j going through (V_1) and/or (V_2) ... and/or (V_n)

$(A^n_{ij}) = \min\{A^{n-1}_{ij}, A^{n-1}_{in} + A^{n-1}_{nj}\}$

Final Result of APSP



eg:



Cost

	A	B	C	D
A	0	5	8	25
B	2	0	∞	6
C	20	35	0	4
D	∞	∞	30	0

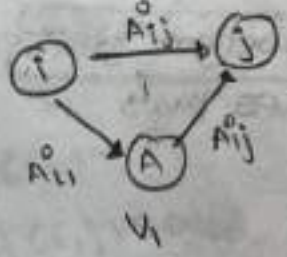
[given edge cost]

$A^0 =$

	A	B	C	D
A	0	5	8	25
B	2	0	∞	6
C	20	35	0	4
D	∞	∞	30	0

$A^1 =$

	A	B	C	D
A	0	5	8	25
B	2	0	10	6
C	20	25	0	4
D	∞	∞	30	0

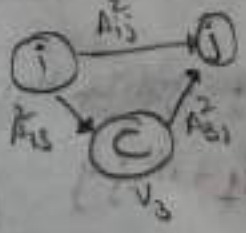


$\Rightarrow A^2 =$

	A	B	C	D
A	0	5	8	11
B	2	0	10	6
C	20	25	0	4
D	∞	∞	30	0

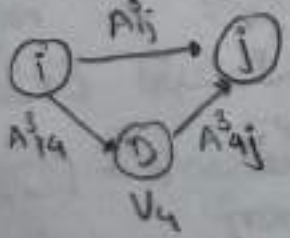
$A^3 =$

	A	B	C	D
A	0	5	8	11
B	2	0	10	6
C	20	25	0	4
D	50	55	30	0



$A^4 =$

	A	B	C	D
A	0	5	8	11
B	2	0	10	6
C	20	25	0	4
D	50	55	30	0



```
Algo floydwarshal(cost[][],n){
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++)
            A[i][j]=cost[i][j]; P[i][j]=-1;
    }
    for(k=1;k<=n;k++){
        for(i=1;i<=n;i++){
            for(j=1;j<=n;j++){
                if(A[i][j]>A[i][k]+A[k][j]){
                    A[i][j]=A[i][k]+A[k][j];
                    P[i][j]=k;
                }
            }
        }
    }
    Return(A[1....n][1...n], P[1...n][1...n]);
```

- APSP Algo works with weighted graph with some -ve edge weight
- also used to detect -ve weight edge cycle

TC of APSP By using DP : $\Theta(n^3)$

SC of APSP By using DP : $\Theta(1)$ (exclude space used for I/P & O/P)

- Shortest Path Algo (2-4 marks)

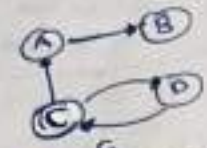
- SSSP Algo for unweighted graph
  BFS Algo [Greedy]
  [Queue DS used $\Theta(n+e)$ / $\Theta(n^2)$]

- Floyd warshal Algo [DP]
  $\Theta(n^3)$

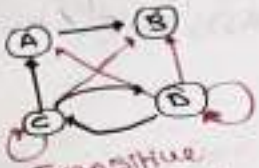
- SSSP for weighted graph
  - no -ve edge
    Dijkstra [Greedy]
    $\Theta((n+e)\cdot\log n)$
    Minheap used
  - with may have -ve edges
    Bellman ford [DP]
    $\Theta(n \cdot e)$

i → j && j → k then i → k

Transitive closure of graph



G
[Given graph]



Transitive closure of G

Adj Matrix

	A	B	C	D
A	0	1	0	0
B	0	0	0	0
C	1	0	0	1
D	0	0	1	0

I/P

0	1	0	0
0	0	0	0
1	1	1	1
1	1	1	1

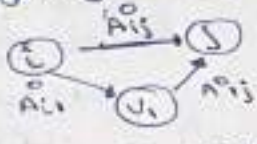
O/P

TC: $\theta(n^3)$ SC: $\theta(1)$

Floyd warshal algo: DP

A^0_{ij} = Adj matrix rep of

$A^0_{ij} = 1$ if path from i to j going through v_1



$A^1_{ij} = A^0_{ij} \vee (A^0_{i1} \wedge A^0_{1j})$

$A^2_{ij} = A^1_{ij} \vee (A^1_{i2} \wedge A^1_{2j})$

i to j group'th v_1 and/or v_2

$A^n_{ij} = A^{n-1}_{ij} \vee (A^{n-1}_{in} \wedge A^{n-1}_{nj})$

final result =

$A^k_{ij} = A^{k-1}_{ij} \vee (A^{k-1}_{ik} \wedge A^{k-1}_{kj})$

where k<=1 to n
i<=1 to n
j<=1 to n

Largest Common Subsequence problem [LCS]

- **Subsequence:** Sequence of character from string(s) which may not be consecutive.

ex: x: [abadac]

aadc
aadac
badc
} subsequence of x

- **Common Subsequence:** Subsequence which is common for 2 string x & y

x: [abeadc]
y: [debcaedc]
} Common subsequence one of the subsequence abadc

2^n no. of subsequence

ex: abadac
2 2 2 2 2 2
choose

- **LCS:** Problem stmt:- max. length subsequence which is common for 2 string x & y with m & n character Respectively.

- Greedy Method fail to solve LSC problem.

Rec calls CBT of n+m level

- Brute force Algo TC: $O(m*2^n)$ (time taken to check one subsequence × total time)

TC: $\theta(2^{n+m})$

SC: $\theta(n+m)$

DP LCS problem.

x, y two strings n, m char each

x: $x_1 x_2 x_3 \cdots x_n$

y: $y_1 y_2 y_3 \cdots y_m$

L(n,m): max length common sequences of x, y strings with n, m char.

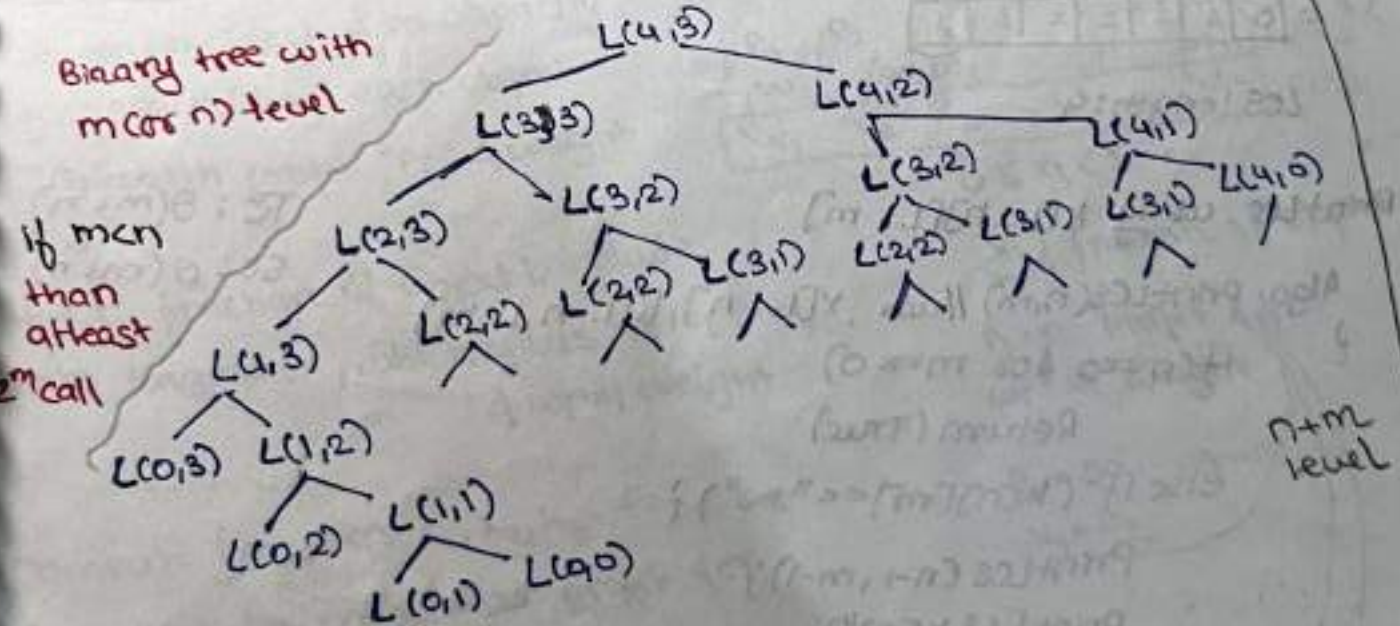
$$L(n,m) = \begin{cases} 0 & \text{; if } n==0 \text{ or } m==0 \\ 1+L(n-1,m-1) & \text{; if } n>0 \&\& m>0 \ x_n==y_m \\ \max\{L(n-1,m), L(n,m-1)\} & \text{; if } n>0 \&\& m>0 \ x_n \neq y_m \end{cases}$$

(L(n,m): these is length of string not for index; 1+L(n-1,m-1): index)

x[1...n] y[1...m] two strings

$$L(i,j) = \begin{cases} 0 & \text{; } i==0 \text{ || } j==0 \\ 1+L(i-1,j-1) & \text{; } x[i]==y[j] \\ \max\{L(i-1,j), L(i,j-1)\} & \text{; } x[i] \neq y[j] \end{cases}$$

(1+L(i-1,j-1): diagonal; L(i-1,j): left; L(i,j-1): top)



- no. of unique call

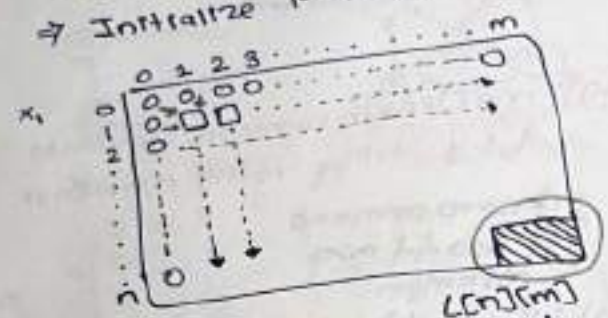
LCS(m,n)

m×n ⇒ unique call

• DP implementation of LCS:-

L[0...n, 0...m] path[1...n, 1...m]

⇒ Initialize 1st Row, 1st column by 0



• Compute LCS either row major order or column major order

TC using DP: $\Theta(nm)$

SC using DP: $\Theta(n*m)$

X = [a b d a c e] n=6

Y = [e a c b d e] n=6

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

X

final value

LCS length: 4

TC: $\Theta(m+n)$

SC: $\Theta(m*n)$

PrintLCS using K[1...n][1...m]

```
Algo printLCS(n,m) // use x[1...n], K[1...n,1...m]
{
  if(n==0 && m==0)
      Return (True)
  else if (K[n][m]=="↖"){
      printLCS(n-1,m-1);
      Printf("X[n]");
  }
  else if (K[n][m]=="←")
      printLCS(n,m-1);
  else
      printLCS(n-1,m);
}
```

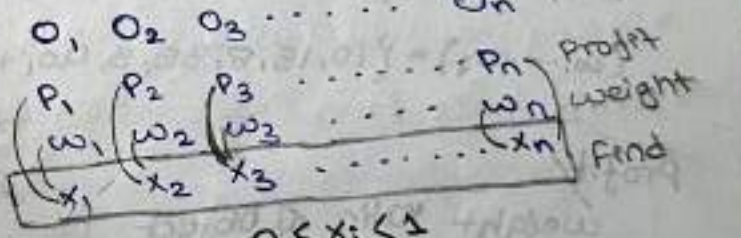
```
Algo LCS( x[m], y[n], m, n) { x[1...m] y[1...n]
    for(i=0; i<=m; i++) L[0][i]=0;
    for(i=1; i<=n; i++) L[i][0]=0;
    for(i=1; i<=n; i++){
        for(j=1; j<=m; j++){
            if(x[i]==y[j]){
                L[i][j] = 1+L[i-1][j-1]
                K[i][j]='K'
            }
            else if(L[i][j-1] >= L[i-1][j]){
                L[i][j]=L[i][j-1];
                K[i][j]="←"
            }
            else{
                L[i][j]=L[i-1][j]
                K[i][j]="↑"
            }
        }
    }
    Return(L[n][m]; K[1...n][1...m])
}
```

T.C : $\theta(m*n)$
S.C : $\theta(n*m)$

Knapsack problem [KNP]

1) Fraction Knapsack problem [FKNP]:

- Given n object & m capacity knapsack (bag) each object "O_i" with profit "P_i" & weight "ω_i"



$0 \le x_i < 1$

- if x_i fraction of Object i, include into knapsack, which gives $P_i \cdot x_i$ profit & $\omega_i \cdot x_i$ weight

x_i fraction of object i

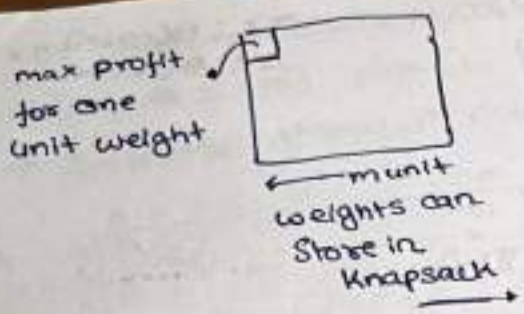
$P_i \cdot x_i$ profit &
$\omega_i \cdot x_i$ weight

$P_1 = 50 \quad \omega_1 = 10$
$x_1 = 1/2$

- Compute fraction $x_1, x_2, x_3 \cdots x_n$ such that to maximize profit for m capacity knapsack

Maximize $\sum_{i=1}^{n} P_i x_i$

Subjected to $\sum_{i=1}^{n} \omega_i x_i \le m$



Greedy Algo [FKNP] — $\Theta(n \log n)$

1] Sort obj decreasing order of their P/w ratios — $\Theta(n \log n)$

$P_1/w_1 \geq P_2/w_2 \geq \cdots \geq P_n/w_n$

2] Fill knapsack by object in above order. — $\Theta(n)$

P_i profit for w_i weight of object i

Profit for one unit weight of object i $= \frac{P_i}{w_i}$

Greedy Sol:
Fill Knapsack based highest P/w ratio objects.

Ques] n=7 · m= 75

$(P_1 \cdots P_7) = \{40, 50, 35, 85, 20, 65, 15\}$

$(w_1 \cdots w_7) = \{10, 15, 8, 35, 3, 40, 7\}$

(i) Max profit?

(ii) what are fractions $(x_1 \cdots x_n)$ which is optimal profit?

Profit/weight ratio of Object

$O_1 \; O_2 \; O_3 \; O_4 \; O_5 \; O_6 \; O_7$

$\Rightarrow \{\frac{40}{10}, \frac{50}{15}, \frac{35}{8}, \frac{85}{35}, \frac{20}{3}, \frac{65}{40}, \frac{15}{7}\}$

$\Rightarrow \{4, 3.33, 4.37, 2.42, 6.66, 1.62, 2.14\}$

Greedy selection order of object:-

$\{O_5, O_3, O_1, O_2, O_4, O_7, O_6\}$

$\Sigma P_i x_i$

$20*1 + 35*1 + 40*1 +$
$50*1 + 85*1 + 15*\frac{4}{7}$

$\Rightarrow 238.57$ (max profit)

$\Sigma w_i x_i \leq 75$

included fully b/c there is space

$3*1 + 8*1 + 10*1 +$
$15*1 + 35*1 + 7*\frac{4}{7} = 75$

Fraction $\{x_1 \; x_2 \; x_3 \; x_4 \; x_5 \; x_6 \; x_7\}$

$\Rightarrow \{1, 1, 1, 1, 1, 0, 4/7\}$

[2] 0/1 Knapsack problem [0/1 KNP]:-

Given; n objects & m capacity

Knapsack profit p[1...n]

weight w[1...n]

Maximize $\sum_{i=1}^{n} p_i \cdot x_i$

subjected to $\sum_{i=1}^{n} w_i x_i \leq m$

$x_i = 0$ or 1

ex: m=20

$(p_1\ p_2\ p_3) =$

(30 16 15)

$(w_1\ w_2\ w_3) =$

(14 9 10)

Greedy sol:

profit [30]

profit: 30

$(x_1 x_2 x_3) = (1\ 0\ 0)$

(not optimal)

- greedy sol^n failed to solve 0/1 Knapsack.

- Brute force Algo for 0/1 Knapsack problem

Compute profit for every subset of object and return subset of object whose sum of profit is maximum & sum of weight is $\leq m$

TC of KNP using Brute force $\Theta(2^n)$

object	profit	weight ≤ 20
{1}	30	14
{2}	16	9
{3}	15	10
{1,2}	46	~~23~~
{1,3}	45	~~24~~
{2,3}	31	19 optimal
{1,2,3}	61	~~33~~

- DP of 0/1 KNP:

p[1...n] profits, w[1...n] weights m

Knp capacity

ex V(2,10) / V(n,m): max profit for n obj's & m capacity KNP.

$$V(n,m) = \begin{cases} 0 & ; n=0, m \geq 0 \\ V(n-1,m) & ; w_n > m \\ \max\{V(n-1,m), & ; w_n \leq m \\ \quad V(n-1,m-w_n)+p_n & \end{cases}$$

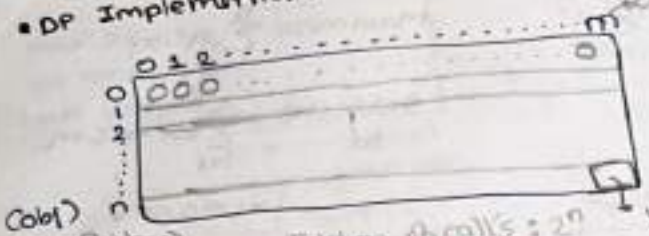
OR

p[1...n] w[1...n] // I/P n obj m KNP

$$V(i,j) = \begin{cases} 0 & ; i=0, j \\ V(i-1,j) & ; w_i < \\ \max\{V(i-1,j), & ; w \\ \quad V(i-1,j-w[i])+ \\ \quad p[i]\} & \end{cases}$$

value V(n,m)

• DP Implementation: V[0...n, 0....m] 2D array



• Initialize 1st Row by 0's
Compute V(i,j) in Row major order.
V[n][m] final value.

Total no of calls : 2^n

TC of KNP using DP : $\Theta(n*m)$

SC of KNP using DP : $\Theta(n*m)$

Sum of subset problem

• Given n positive integers $\{a_1, a_2, a_3 \cdots a_n\}$ & target integer (m)
SOS problem should return TRUE if any subset of given n integers sum equal to m exists . otherwise return false

$\{a_1\ a_2\ a_3\ a_4\} = \{5, 2, 8, 6\}$

m = 15 → SOS ⇒ TRUE

m = 17 → SOS ⇒ False

• Greedy method failed to solve SOS problem

• Brute force Algo SOS :- Compute sum of every possible subset of given $\{a_1, a_2, a_3 \cdots a_n\}$ integers. If any subset sum equal to m then return true

Given: n=4 : [positive Integers] $\{a_1\ a_2\ a_3\ a_4\} = \{4, 2, 6, 5\}$

If Target integer m : 13

If Target integer m : 14

Subset	Subset Sum
a_1	4
a_2	2
a_3	6
a_4	5
$a_1 a_2$	6
$a_1 a_3$	10
$a_1 a_4$	9
$a_2 a_3$	8
$a_2 a_4$	7
⋮	⋮

m = 14

SOS : false

TC of SOS Using Brute force

$\Theta(2^n)$

[Exponential]

DP Solution: given n positive integers $\{a_1, a_2, a_3 \cdots a_n\}$ && target integer m.

SOS(n,m) : True if any subset sum of n integers equal to m

[function]

$$SOS(n,m) = \begin{cases} \text{TRUE} & ; n=0 \,\&\&\, m=0 \\ \text{false} & ; n=0 \,\&\&\, m>0 \\ SOS(n-1,m) & ; n>0 \text{ and } a_n > m \\ SOS(n-1,m) \vee (SOS(n-1,m-a_n) \wedge a_n \le m) & , a_n \le m \end{cases}$$

base

General Recurrence Relation

SOS(i,j) : TRUE if any subset of i positive int sum equal to target j exists

$$SOS(i,j) = \begin{cases} \text{True} & \text{if } i=0 \,\&\&\, j=0 \\ \text{false} & \text{if } i=0 \,\&\&\, j>0 \\ SOS(i-1,j) \vee (SOS(i-1,j-a_i) \wedge a_i \le j) & , \text{if } i>0 \end{cases}$$

final value → SOS(n,m)

SOS(n,m) → T/F

DP Implementation

- Define SOS[0....n, 0...m]
- Initialize 1st Row by false [F] except SOS[0][0]
- S[0][0] = True
- Compute S[i][j] in Row major order
- Return SOS[n][m]

S(0,0)=T S(0,1)=F S(0,2)=F S(0,3)=F S(1,0)= S(0,0)

S(2,0) = S(1,0)

S(4,12) = S(3,12) ∨ S(3,7)
T ∨ F

S(4,7) = S(3,7) ∨ S(3,2)
F ∨ T

Q) n=4 $(a_1 a_2 a_3 a_4) = (4,2,6,5)$ target int : m=13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	F	F	F	F	F	F	F	F	F
2	T	F	T	F	T	F	T	F	F	F	F	F	F	F
3	T	F	T	F	T	F	T	F	T	F	T	F	T	F
4	T	F	T	F	T	T	T	T	T	T	T	T	T	T

final value

$S[i,j] = S[i-1,j] \vee (S[i-1, j-a_i] \wedge a_i \leq j)$

DP Algo for SOS:-

```
Algo SOS(a[1...n],n,m){ // a[1...n] are positive int, m target int
    S[0][0] = true
    For(i=1; i<=m; i++){
        S[0][i] = false;
    }                                   } Initialization of 1st Row
    For(i=1; i<=n; i++){
        for(j=0; j<=m; j++){
            if(a[i]>j)
                S[i][j] = S[i-1][j]
            else
                S[i][j] = S[i-1][j] || S[i-1][j-a[i]]
        }
    }
    Return S[n][m]
}
```

TC of SOS problem using DP: $\Theta(n \cdot m)$

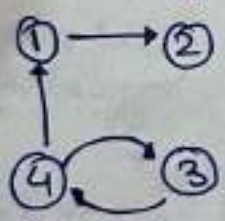
SC of SOS problem using DP: $\Theta(n \cdot m)$

Transitive closure of graph:-

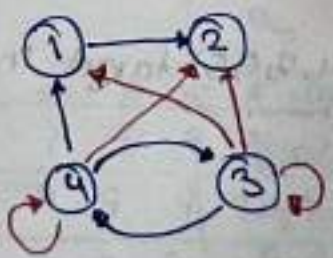
- If in graph G there exists edges from i to j & j to k then in [transitive closure of G] add edges from i to k also.

[also can compute by floyd warshal algo of DP]

Graph G



$\Rightarrow$ G' = transitive closure of G



Already done before

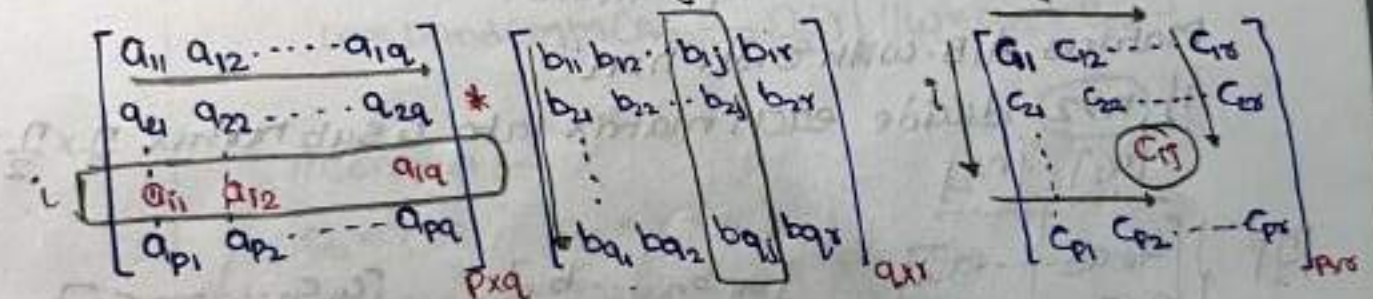
```
Algo Transitive_floydwarshals(n, Adj[][])
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            Aij = Adj[i][j]
    }
    for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                Aij = Aij V (Aik ^ Akj)
            }
        }
    }
    Return(A[][])
}
```

- TC of Transitive closure of graph using OP: $\theta(n^3)$ [Floyd warshal]
- SC of Transitive closure of graph using OP $\theta(n^2)$

Straight Matrix Multiplication

$A_{p \times q}$ $B_{q \times r}$

q multiplication



$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} * \cdots + a_{1q} * b_{q1}$$ (q multiplication)

$$c_{ij} = \sum_{k=1}^{q} a_{i1} * b_{1j} + a_{i2} * b_{2j} + \cdots + a_{iq} * b_{qj}$$

(k=1 to q)

```
for(k=1;k<=q;k++)
    cij = cij + aik * bkj
```

```
Algo Matrix Mul(A[p][q], B[q][r]){
    for(i=1; i≤p; i++){
        for(j=1; j≤r; j++){
            C[i][j]=0;
            for(k=1; k≤q; k++){
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
            }
        }
    }
    Return(C[][])
}
```

p, r, q

TC = $\Theta(p*r*q)$

A p×q B q×r

cost of Matrix Mul = (p*q*r) element multiplication

A n×n B n×n matrices

TC of Matrix Mul = $\Theta(n^3)$

② D And C Matrix Multiplication:-

Matrices A, B with order n×n

if (n>2) divide each matrix into 4 sub matrix $\frac{n}{2} \times \frac{n}{2}$ orders.

$C_{11} = (A_{11} * B_{11}) + (A_{12} * B_{21})$

$C_{12} = (A_{11} * B_{12}) + (A_{12} * B_{22})$

$C_{21} = (A_{21} * B_{11}) + (A_{22} * B_{21})$

$C_{22} = (A_{21} * B_{12}) + (A_{22} * B_{22})$

To multiply A/B matrices of n×n orders

- 8 sub matrix mul of orders $n/2 \times n/2$
- 4 Matrix Additions

DandC Matric Mul Algo

```
Algo DandCMM(A,B,n) {  ⟶ T(n)
    // A,B two matrices of order nxn
    if(n≤2) // small
        Return(A*B) ⟶ TC = θ(1)
    else {
        Divide Matrices A,B with order nxn into sub
        matrices of order n/2 x n/2 each
        A11 A12 A21 A22 , B11 B12 B21 B22
        P = DAndCMM(A11, B11, n/2)   // P = A11 * B11
        Q = DAndCMM(A12, B21, n/2)   // Q = A12 * B21
        r = DAndCMM(A11, B12, n/2)
        .
        .
        .
        W = DAndCMM(A22, B22, n/2)   // W = A22 * B22
        // Conquer
        C11 = P+q   C12 = r+s
        C21 = t+u   C22 = v+w        } 4 matrix Addition
        Return([C11 C12; C21 C22]);
    }
}
```

n≤2 (Const)

n>2

8T(n/2)

4n²

DAndC Matric Mul Recurrence Rel

$$T(n) = \begin{cases} \text{const} & , n \le 2 \\ 8T(\frac{n}{2}) + 4n^2 & , n > 2 \end{cases}$$

$\Rightarrow \theta(n^3)$

Strassen's DandC Matrix Mul:- [efficient algo]
Matrix A,B with order (n×n) divided into 4 sub matrix each with order $\frac{n}{2} \times \frac{n}{2}$ each

$p = (A_{11} + B_{12}) * (B_{11} + B_{22})$
$q = (A_{21} + B_{22}) * B_{11}$
$r = A_{11} * (A_{12} - B_{22})$
$s = A_{22} * (B_{21} - B_{11})$
$t = (A_{11} + A_{12}) * B_{22}$
$u = (A_{21} - A_{11}) * (B_{11} + B_{22})$
$v = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = p + q - t + v$
$C_{21} = q + s$
$C_{12} = r + t$
$C_{22} = p + r - q + u$

T(n)
⇒ To multiply two matrix of order n×n
① 7 sub matrix $\frac{n}{2} \times \frac{n}{2}$
② 18 matrix Addition / subtraction

Recurrence Rel- of Strassen's DAndC Matrix Mul

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T(n/2) + 18 * n^2 & n > 2 \end{cases}$$

$\Rightarrow \theta(n^{\log_2 7}) = \theta(n^{2.81})$

Job scheduling based on deadline

- given n task [jobs] each task T_i with profit P_i & deadline d_i If task T_i completed on or before deadline d_i then which gives profit P_i otherwise no profit
- Inorder to complete given n task maximize profit subjected to constraint.

1) All task ready to execute
2) each task required one unit execution time
3) only one task can execute at any time.

Q]

Task	DBMS	Algo	DS	CO	OS	CN
Profit (marks)	12	15	5	10	9	8
Deadline (Days)	4	4	6	2	2	4

→ max marks?
⇒ schedule task to get max marks?

discard

D_1	D_2	D_3	D_4	D_5	D_6
OS	CO	DBMS	Algo		OS

deadline

9 + 10 + 12 + 15 + 5 = (51) max possible marks

Time complexity

Greedy Solution Job Scheduling based on DL

$O(n \log n)$ { 1) sort tasks decreasing order of their profit

$P_1 > P_2 > P_3 > \dots > P_n$

Greedy Selection { Highest profit task scheduled first }

$\Theta(n^2)$ in worst case { 2) schedule task in above order Task T_i with profit P_i and deadline d_i schedule T_i at last possible slot of deadline based free slot from d_i to 1. If no slot free from d_i to 1 then discard task T_i.

gate Ques]

Task no	1	2	3	4	5	6	7	8	9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

① what is max profit ? 147

② what are task discard inorder to schedule task to get max profit? $T_4 T_6$

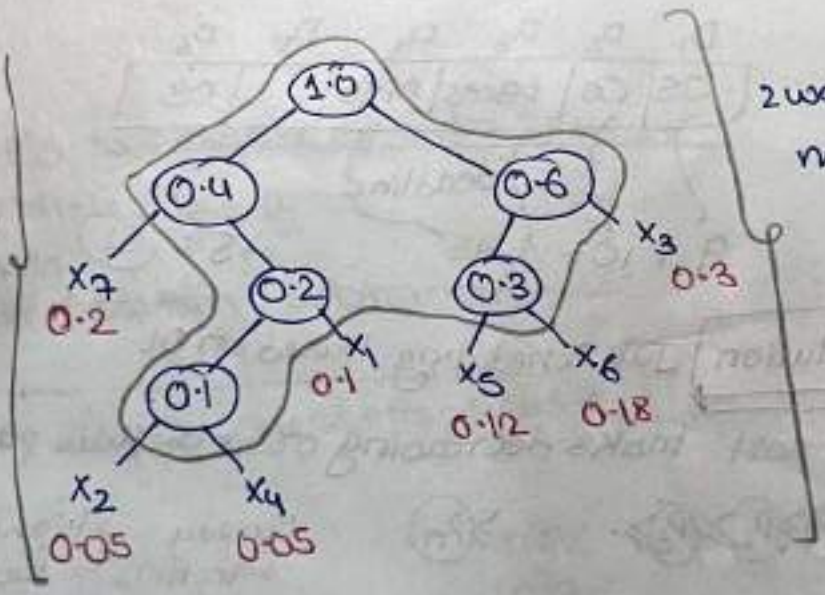
Ques] given 7 character $x_1, x_2, x_3, x_4, x_5, x_6, x_7$ with frequency count 10%, 5%, 30%, 5%, 12%, 18%, 20% respectively. what is Avg no. of bits/char using huffman coding

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
260 / 100 = 2.6	10%	5%	30%	5%	12%	18%	20%
	0.1	0.05	0.3	0.05	0.12	0.18	0.2

Avg bit/char

$= 0.1 + 0.2 + 0.3 + 0.4 + 0.6 + 1.0$

$\Rightarrow 2.6$ bit/char

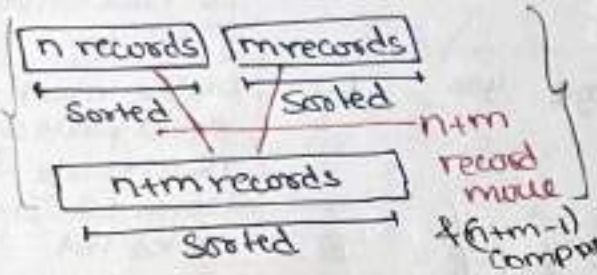


2 way optimal merge tree

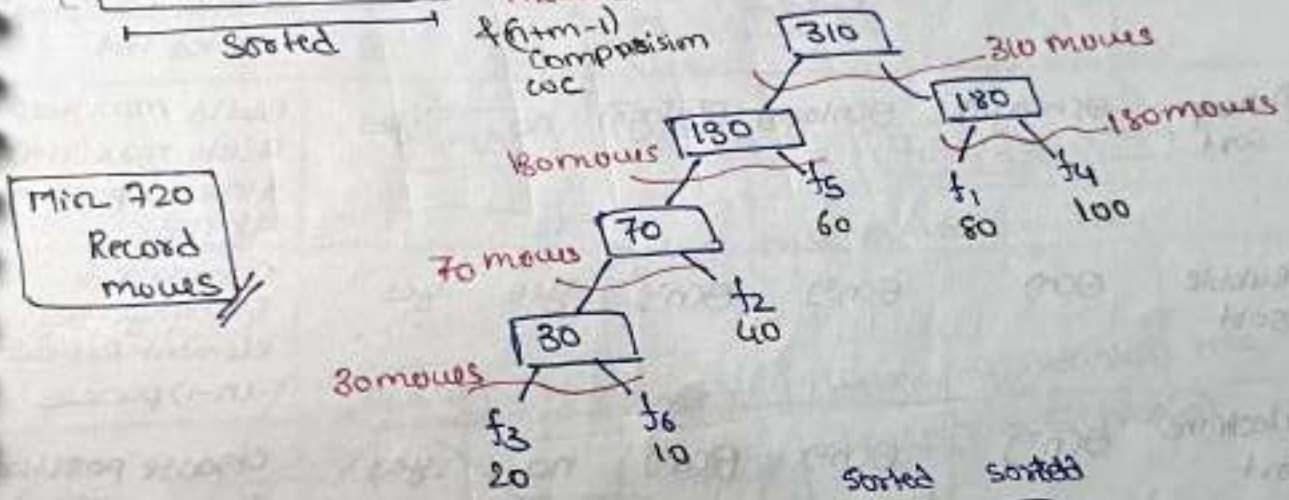
Ques] Assume [n+m records moves] required to merge two sorted file of n*m records each. How many min. records moves gives 6 sorted file [f_1 f_2 f_3 f_4 f_5 f_6] with [80,40,20,100,60,10] sorted record into single sorted file?

Greedy soln (Optimal) : merge two files which has min #f record

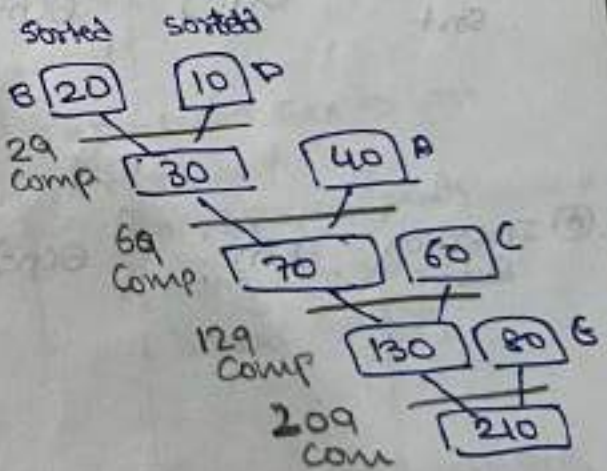
2 way optimal merge



f_1	f_2	f_3	f_4	f_5	f_6
80	40	20	100	60	10



Min 720 Record moves



Ques] suppose A,B,C,D,E sorted array with 40,20,60,10,80, elements each. these are to be merge into a single sequence by merging two sequnce at a time? the no. of comparision that will be in worst case by the optimal algo?

Optimal Algo (Greedy) ⇒ 2 way optimal merge

[merge 2 array which has min possible elemtts]

COC comp. to merge array using optimal algo:

⇒ 29+69+129+209

⇒ 436 comparision

Sorting Algorithms

[1] Comparision based sorting algo.

[2] non-comparision based sorting algo.

1) Comparision based sorting Algo:-

Sorting Algo	Best case time complexity	Avg case	Worst case	Stable	Inplace	Algo logic
① Quick sort	$\theta(n\log n)$	$\theta(n\log n)$	$\theta(n^2)$	No	yes	Choose pivot & place at correct position & Divide list base on pivot position
② Merge sort	$\theta(n\log n)$	$\theta(n\log n)$	$\theta(n\log n)$	yes	no	Divide into two equal parts sort recursively & merge into single sorted list
③ Heap sort	$\theta(n\log n)$	$\theta(n\log n)$	$\theta(n\log n)$	no	yes	Build max heap & delete max (n-1) times & place max at end.
④ Bubble sort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$	yes	yes	compare & Exchange Adj. element Repeat (n-1) passes
⑤ Selection sort	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$	no	yes	Choose position of min. from a[K] to a[n] & swap with a[K] where K: 1, 2, 5 ... n-1
⑥ Insertion Sort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$	yes	yes	Place a[i] into correct position of sorted part of array a[1] to a[i-1] where i: 2, 3 ... n

② Non Comparision based Sorting algo:-

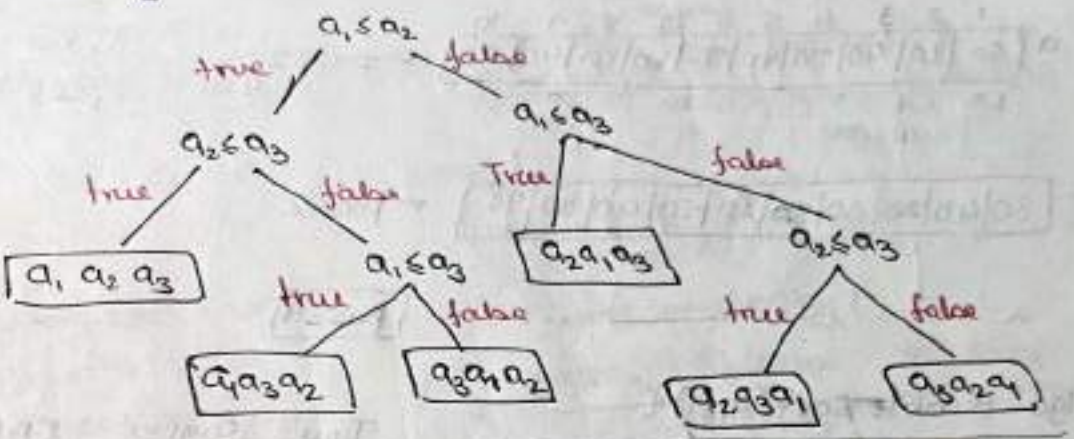
(i) Counting sort : TC & SC : $\theta(n+K)$

Stable sorting algo but not inplace sorting algo.

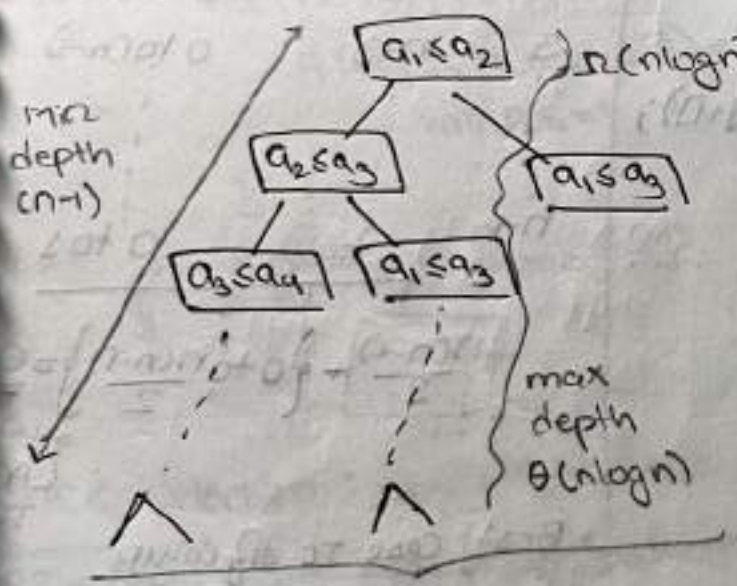
(2) Radix sort : TC = $\theta(nd)$ SC = $\theta(n)$

Stable sorting algo & inplace sorting algo.

- Max. (worst case) comparision Required to sort nelement using Comparision based sorting algo is $\Omega(n\log n)$.
- Decision Tree to sort 3 elements $[a_1, a_2, a_3]$ using comparision based sorting.



3! leaf node of decision tree



n! leaf node

Height of decision tree
$\log(n!) = \theta(n\log n)$

- Decision Tree to sort n element $\{a_1, a_2, a_3 \ldots\ a_n\}$ using Comparison based sorting :-

- worst case no. of comparision to sort array of n element using Comparision based sorting

$\Omega(n\log n)$ → or min comparision Required in worst case

- min no. of comparisons to sort array of n elements using comparision based sorting algo. : n-1

Bubble sorting

- Compare adj. element a[j], a[j+1]

if (a[j] > a[j+1]) swap (a[j], a[j+1])

where j = 1 to n-1 and i = 1, 2, 3 ... n-1 passes

// Stable sorting & Inplace sorting

Comp Adj elements

	1	2	3	4	5	6	7	8	9	10	
a	60	80	30	90	20	90	30	50	50	40	90
	30	60	80	20	80	30	80	50	50	40	

- In 1st iteration largest is placed at then end by just comparing adj element

	1	2	3	4	5	6	7	8	9	10	
a	60	30	40	20	80	30	80	50	40	90	← pass 1
	30	60	60	60	30	50	50	50	40		
		40	20								

a | 30 | 40 | 20 | 60 | 30 | 80 | 50 | 40 | 80 | 90 | ← pass 2

⋮

pass 9

```
Algo BubbleSort(a, n)
{
    for(i=1; i<=n-1; i++)
    {
        flag=0;
        for(j=1; j<=n-i; j++)
        {
            if(a[j]>a[j+1])
            {
                Swap(a[j], a[j+1]);
                flag=1;
            }
        }
        if(flag==0) return;   // if no swap then stop the loop
    }
}
```

Time complexity : [AC & WC]		
pass	# comp	# swaps
1	n-1	0 to (n-1)
2	n-2	0 to (n-2)
3	n-3	0 to (n-3)
⋮	⋮	⋮
n-1	1	0 to 1

$\frac{n(n-1)}{2}$ + {0 to $\frac{n(n-1)}{2}$} = $\Theta(n^2)$ WC / AC TC

- Best case TC # comp

min (n-1) comp in 1st pass

⋮

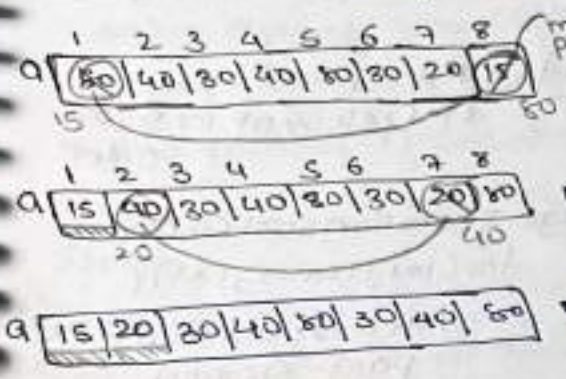
(n-1) = $\Theta(n)$ BC TC

# of swap's required to sort array using bubble sort = # of inverse of the array

# of comp of : $\frac{n(n-1)}{2}$ except Best case bubble Sort

Avg swap = $\frac{n(n-1)}{4}$

Selection Sort Find position of min from a[i] to a[n] & swap with a[i] where i=1,2,3...n-1 passes



[Selection sort is inplace sorting but not stable sorting algo]

```
a[1-n]
mp=1
for(j=2 to n){
  if(a[j]<a[mp]){
    mp=j
  }
}
```

```
Algo selection sort (a,n){
  for(i=1;i<=n-1;i++){
    // find pos of min from
    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]);
  }
}
```

Time complexity:

passes	#f comp	#f swap
1	(n-1)	1
2	(n-2)	1
3	(n-3)	1
...	...	...
n	1	1
		(n-1)

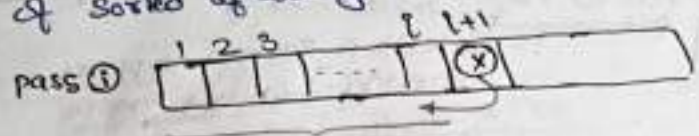
Time compln $\frac{n(n-1)}{2} + (n-1) \Rightarrow \Theta(n^2)$ [for all cases]

- TC of selection : $\Theta(n^2)$ in all case
- only (n-1) swaps required to sort array
- Selection Sort special case of Quick sort with pivot element selected as min element

```
SortA(a,n){
  if(n<20) // small size array
    selectionSort(a,n)
  else
    QuickSort(a,n);
}
```

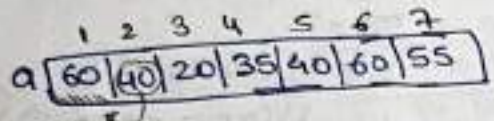
- Posteriory Analysis: less element to sort : Selection sort Runs faster than QS more element to sort: QS run faster then SS.

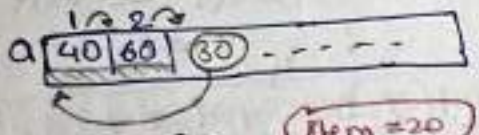
Insertion Sort :- Insert element a[i+1] into correct position of sorted of array a[0] to a[i]. where i=1,2,3. ... n-1 (passes)

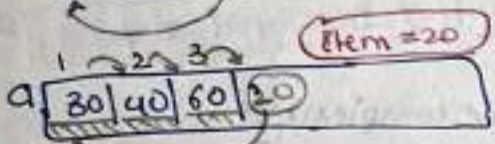


After pass i (i+1) element in sorted order

Sorted part of array







Pass (n-1)

```
Algo InsertionSort(a,n){
  for(i=1;i<=n-1;i++){
    // a[i+1] places into sorted part of array
    // a[0] to a[i]
    item=a[i+1]; j=i
    while (j>=1 and a[j]>item){
      a[j+1]=a[j]
      j--;
    }
    a[j+1]=item;
  }
}
```

array shift

Element Comp.

#f no. of swap or #f no. of comparision of Insertion sort of n element — (=) Equal — #f Inv. of array — (≈) Almost

Time Complexity:-

Posses	min comp	min Shift	max Comp.	max array Shift	Avg Comp.	Avg array Shift
1	1	0	1	1	1	1
2	1	0	2	2	2/2	2/2
3	1	0	3	3	3/2	3/2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
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

BC TC of IS : θ(n)

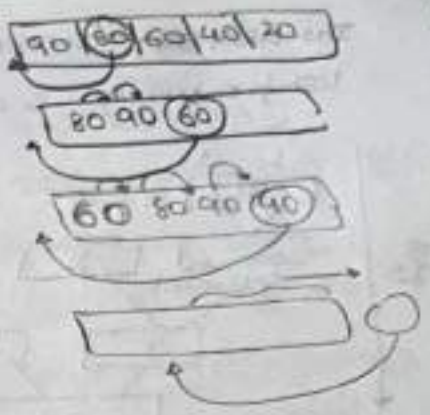
worst case TC of Insertion Sort : θ(n²)

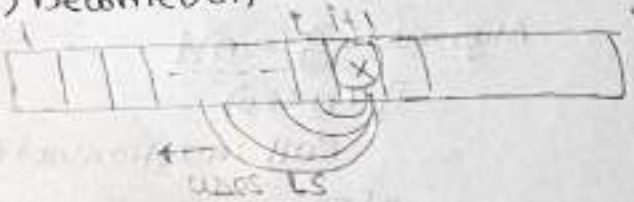
Avg. Case TC of Insertion Sort : θ(n²)

- Insertion sort WC TC $\Theta(n^2)$ use linear search to find position of inserting element into sorted part of array. if Binary search uses to find position of Inserting element then (WC) TC of modified Insertion Sort ?

a) Remain $\Theta(n^2)$ b) become $\Theta(n\log n)$

c) become $\Theta(n)$ d) become $\Theta(\log n)$





BS to find pos

$\Rightarrow \log i$ comp)

$\Rightarrow i$ array shift

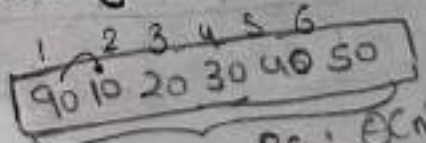
	worst case Comparision	Array shift worst case
pass 1	1	1
pass 2	$\log 2$	2
pass 3	$\log 3$	3
⋮	⋮	⋮
pass $n-1$	$\log(n-1)$	$n-1$
	$\log(n+1)!$	$\frac{n(n-1)}{2} = \Theta(n^2)$ array

$\approx \Theta(n\log n)$ Comparision + Shift operation

$\Theta(n^2)$

Qus] array of n element with nInverse in array which sorting algo sorts array in $\Theta(n)$ time

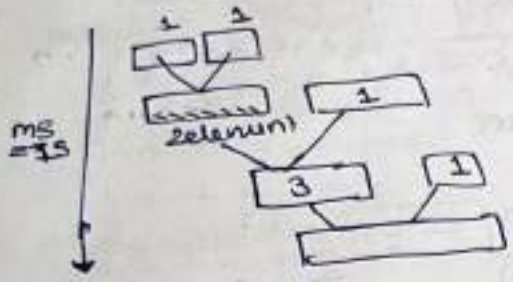
a) QS
b) BS
c) IS
d) SS



TC of : BS : $\Theta(n^2)$
SS : $\Theta(n^2)$

TC of : IS : $\Theta(n)$
QS $\Theta(n^2)$

Insertion Sort special case of merge sort if 2nd sorted list for merge is always one element



Insertion Sort is Stable algo & inplace sorting algo

posterior Analysis:-

* for small array to perform sort Insertion sort runs faster than merge sort.
* for large array to performs sort merge sort Runs faster than Insertion sort.

```
Algo sorting (a,n){
    if (n<20)
        Call insertionSort(a,n)
    else
        Call mergeSort(a,n)
}
```

Non-comparision based sorting Algo

- Array of integer elements & Range of element [0,K]

Array of 12 integer elements Range [0...8]

```
For(i=0;i<K;i++)
    C[i]=0
For (i=1;i<n;i++)
    C[a[i]]++;

For(i=1;i<K;i++)
    C[i]=C[i]+C[i-1]

For(i=n;i>=1;i--){
    b[C[a[i]]]=a[i];
    C[a[i]]--;
}
```

	1	2	3	4	5	6	7	8	9	10	11	12
a	4	7	4	8	6	8	0	3	4	8	7	6

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

	0	1	2	3	4	5	6	7	8
count	1	1	1	2	5	6	7	9	12

	1	2	3	4	5	6	7	8	9	10	11	12
b	0	3	4	4	4	5	6	7	7	8	8	8

```
Algo CountingSort(a,n,K){
  // A[1...n] array of n integer & Range of element (0,K)
Θ(k){ For(i=0;i<=K;i++)
        c[i]=0
Θ(n){ For(i=1;i<=n;i++)
        C[a[i]]++
Θ(k){ For(i=1;i<=K;i++)
        C[i]=C[i-1]+C[i]
Θ(n){ For(i=n;i>=1;i--){
        b[c[a[i]]]=a[i]
        c[a[i]]--
    }
  Return b[1...n]
}
```

- Counting sort TC: $\Theta(n+k)$
- Best case TC of counting sort: $\Theta(n)$
- SC of Counting sort: $\Theta(n+k)$

C[0...K] & b[1...n] array used

* counting sorting stable sorting but not Inplace sorting

→ these run from n to 1 b/c to make it stable Algo

Radix Sort :- [bucket Sort]

Radix[r] : r different Symbols used to represent any no. whose value are 0,1,2,3.....r-1

r=10 ⇒ {0,1,2,3,4,5,6,7,8,9}

r=16 ⇒ {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} // [16 Symbols]

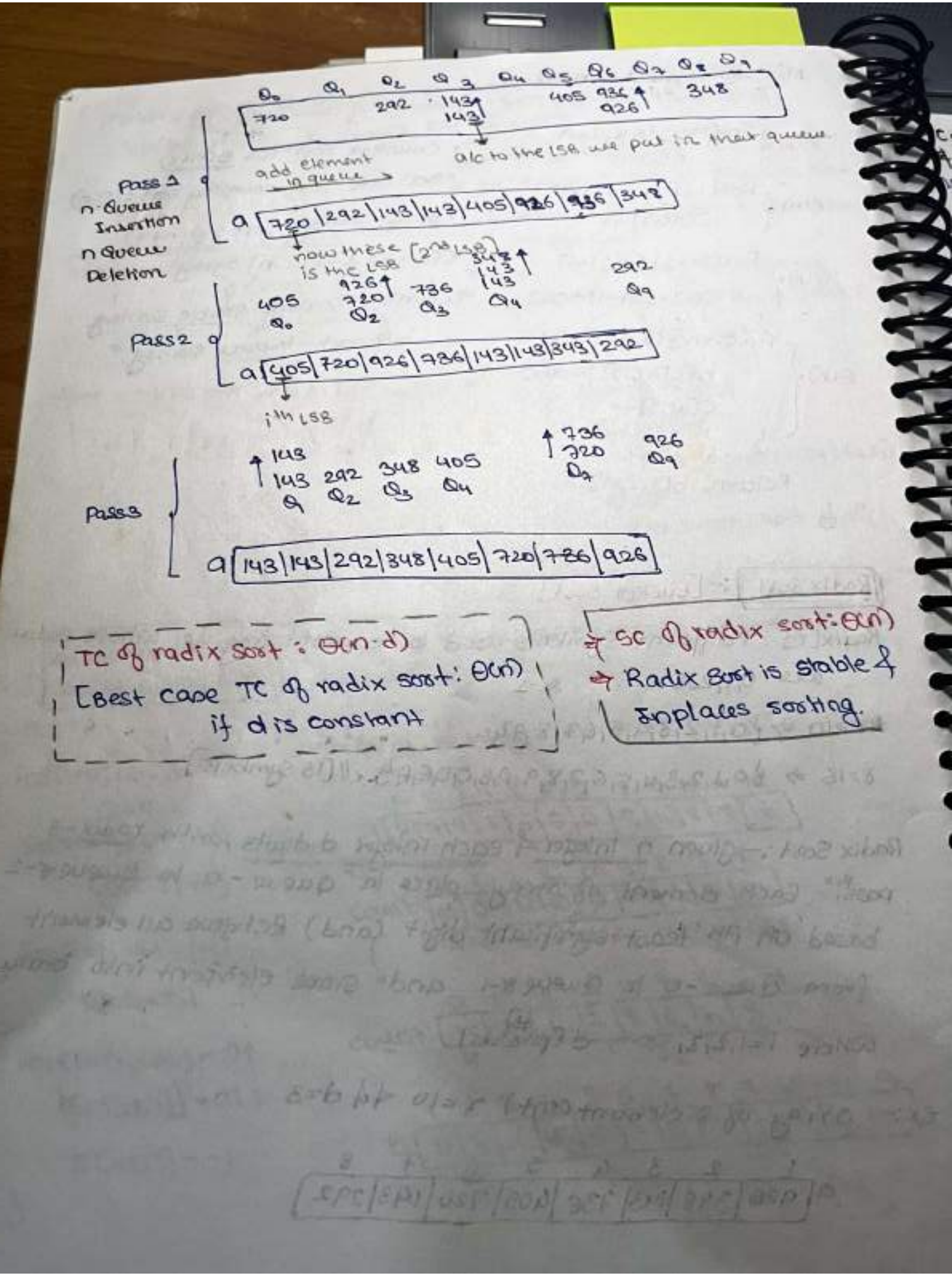
Radix Sort :- given n Integer & each integer d digits, with radix-r

pass :- Each element of array place in queue-0 to Queue r-1 based on ith least-significant digit (and) Retrieve all element from Queue-0 to Queue r-1 and store element into array

where i=1,2,3,.....d[passes] (# of passes)

ex:- array of 8 element (int) r=10 & d=3

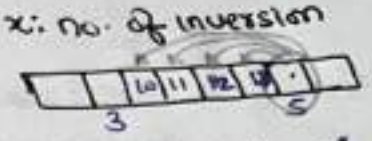
	1	2	3	4	5	6	7	8
a	926	348	143	736	405	720	143	292


Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
720 292 143 143 405 926 926 348
add element in queue
a/c to the LSB we put in that queue
Pass 1
n Queue Insertion
n Queue Deletion
a 720 292 143 143 405 926 936 348
now these is the LSB
405 Q0
926 720 Q2
736 Q3
348 143 143 Q4
292 Q9
Pass 2
a 405 720 926 736 143 143 343 292
ith LSB
143 143 Q
292 Q2
348 Q3
405 Q4
736 720 Q7
926 Q9
Pass 3
a 143 143 292 348 405 720 736 926
TC of radix sort : Θ(n·d)
[Best case TC of radix sort: Θ(n) if d is constant
SC of radix sort: Θ(n)
Radix Sort is stable & Inplaces sorting.

- Counting Inversions & Insertion Sort

Insertion sort runs in time $O(n+x)$ (x = no. of inversion)

x: no. of inversion



no. of time you are shifting one element = no. of inversion of that one element

one element → i_1
2 element → i_2
3 element → i_3
$i_1+i_2+i_3$ = x

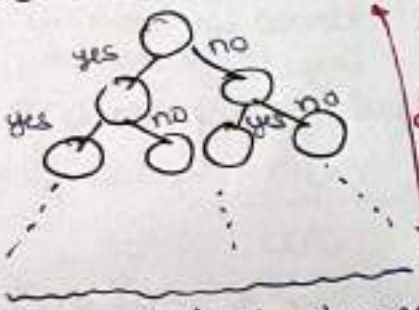
{all elements combindy
{ we are shifting 'x' time or swap

x = total no. of inversion

time complexity of insertion sort = $O(x)$ or $O(x+n)$ (whichever is max)

B.C : $O(n)$
W.C : $O(n^2)$

- Decision tree



this is a binary tree with 'L' leaves & 'd' depth

$2^d \geq L \Rightarrow d \geq \log_2 L$

max. leaves

note

2^d leaves (max)

$O(\text{depth})$ & $n!$ → leaves

Thus, in all binary tree with $n!$ leaves the longest path has length atleast $\log(n!)$ (longest path → depth)

In worst case, $\log(n!)$ comparsion

≡ in worst care you need $\Omega(n\log n)$ comparsion

- every possible execution of the algo. is represented as one path from the root to leaf.

i.e $f(n) = O(n^3)$

$f(n) \leq n^3$

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

$g(n) \geq n^2$

my algo. is having time complexity of $\Omega(n^2)$

- in worst case it take $O(n^3)$ ≡ in worst case $\leq n^3$ (upper bound)
- in all cases it $O(n^3)$ → time complexity $O(n^3)$

Best case → $\geq \Omega(n^2)$
↳ not lower than n^2

worst case → $\leq O(n^2)$
↳ not more than n^2

- Bubble Sort
  - Compare pair of adjacent items
  - Swap if the items are out of order [smaller → larger] or [larger → smaller]

[✓Inplace ✓stable]

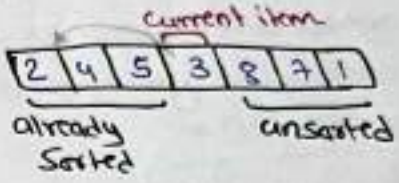
Best case : Ω(n) $O(n^2)$ (already sorted) using modification

Avg. case : $O(n^2)$ $O(n^2)$

worst case : $O(n^2)$ $O(n^2)$

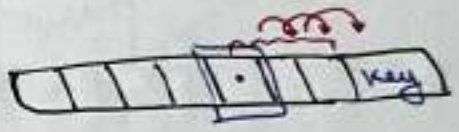
- Insertion Sort
  - like playing card, new element is inserted in sorted order
  - one by one trying to insert element to it's correct position in left side sub array

[✓Inplace ✓stable]



Best case $O(n)$
Avg. case $O(n^2)$
worst case $O(n^2)$

- using Binary search in Insertion sort
  - we can find correct position in $O(\log n)$



#of comparsion – $\log n$ } using array
#of swap – $O(n)$ } using linked list

total : $O(n)$ $O(1)$ $O(n)$

time complexity for one element

for n element : $n * O(n) \Rightarrow n^2$

but total no. of comparism in worst case $O(n \log n)$

- Selection Sort (most natural algo.)
  - find smallest or largest item x
  - Swap x with (first or last) element

All Case $O(n^2)$

[✓Inplace ✗stable]

- Heap sort

$T(n) = O(n) + \log n + \log(n-1) + \log(n-2) \cdots \log 1$

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

$O(n)$ to Build heap; $n \log n$ to sort

All Case

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

[✓Inplace ✗stable]

- Counting sort

$T(n) = O(n+K)$ ← max. no. in the i/p

[✗ inplace ✓ stable]

- Radix sort

$T(n) = O(n \cdot d)$ → no. of digit; n → no. of element

[✓ inplace ✓ stable]

Ques: using which algo an array of elements in range [1—n^3] will be sorted using O(n) time?

(a) Merge sort $\Theta(n \log n)$
(b) Quick sort $\Theta(n^2/n \log n)$
(c) Radix sort
(d) Insertion sort $\Theta(n^2)$
(e) Counting sort $\Theta(n^3)$ $\Theta(n+k)$ ↑ 1 to n^3

#digits to store elements x in radix-x = $\lceil \log_r x \rceil + 1$ digit

Method ①

Array of n elements Range [1....n^3]
TC to sort by using Radix sort
$= \Theta(n*d) = \Theta(n \log n)$

$d = \lceil \log_{10} n^3 \rceil$ digits

How many digit for element x = (n^3) in decimal format (r=10) → method ①

#decimal digit ≈ $\log_{10}(x) = \log_{10}(n^3)$ to represent $x = n^3$
$= \lceil 3 \cdot \log_{10} n \rceil$

Method 2:

given array of n element where range [1...n^3]

Step 1] convert each element of array into radix-n number system

$\Theta(n)$ element $x = (n^3)_{10}$

#digit required to convert x into radix-n number system $= \log_n x = \lceil \log_n n^3 \rceil$
$= 3$ digits (Const)

Step 2] Apply Radix sort for step ① Result array // sorted array is result of Radix sort

$\Theta(n)$

Step 3] Convert each element of sorted array of radix-n to radix-10 [given format]

$\Theta(n)$

TC: Sort array: $\Theta(n)$

MCQ

Ques] If Radix sort to sort n integer in the range $[n^{k/2}, n^k]$ for some $k>0$ which is independent of n. Then time taken would be

a) $\theta(n)$ b) $\theta(k \cdot n)$

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

Direct Radix sort for n element Range $[n^0 - n^k]$

$x = n^k$

(d) #f digits for each element $= \log_{10} n^k = \lceil k \cdot \log_{10} n \rceil$

TC of Radix sort : $\theta(n \cdot d)$

$\theta(n k \log n)$

- covert element into Radix-n number system

$x = n^k$ [element]

#f digit for x in radix-n $\} \approx \lceil \log_n n^k \rceil = k$ digit

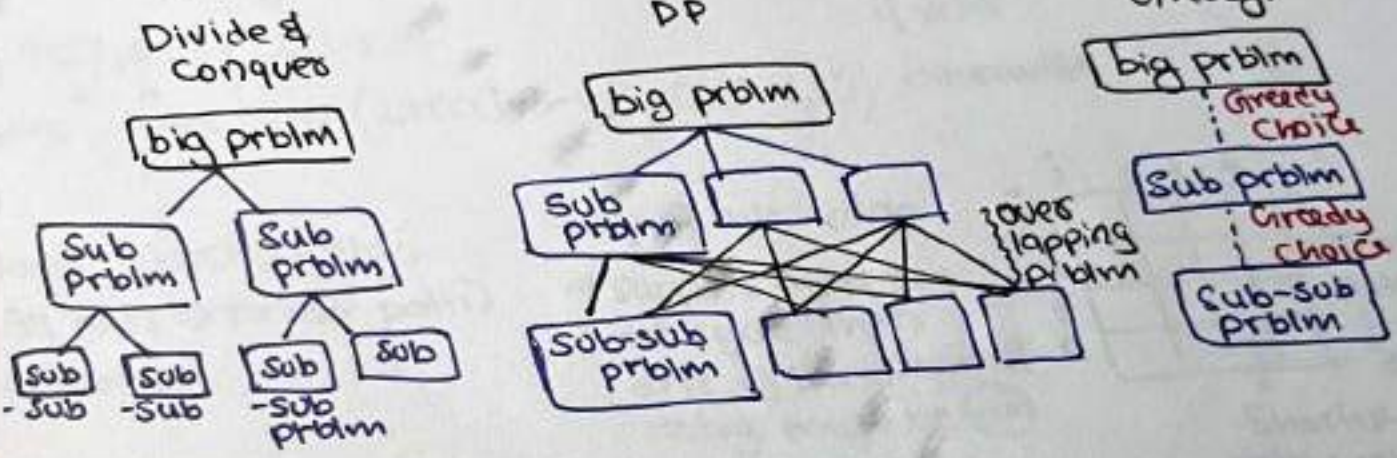
Apply Radix sort

TC : $\theta(n \cdot d) = \theta(nk)$

• DP (Vs) Greedy (Vs) Divide & conquer

• optimal substructure
↳ can you divide problem into subproblem

• overlapping subproblem
↳ Do you have common subproblem



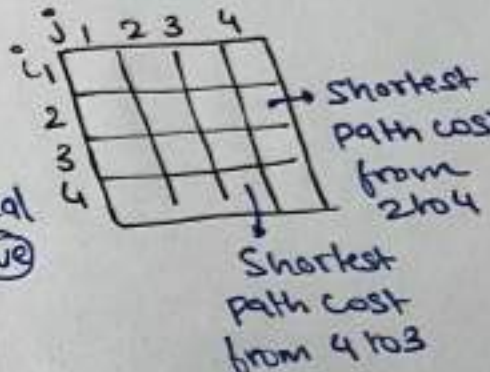
- coin change problem [CC]
  (min. no. of coins)
  → you have notes of ---- rupees
  → min. no. of notes you give

$$CC(i,S) = \begin{cases} 0 & \text{if } n=0 \text{ or } S=0 \\ CC(i-1,S) & \text{if } v_i > S \\ \min(1+CC(i,S-v_i), CC(i-1,S)) & \text{otherwise} \end{cases}$$

i coins, total sum

- Floyd-Warshall Algo.
  (All pair shortest path)
- DP problem

- o/p of this algo.
  → suppose there is a -ve cycle then one of the diagonal entry must be -ve



Time complexity:

All pair
- Floyd ---- V^3 ⟶ V^3 (for one source) ---- vtime ⟶ V^3 (for V vertices)
- Dijkstra --- $(V+E)\log V$ ⟶ $V^2 \log V$ ---- vtime ⟶ $V^3 \log V$
- Bellman --- VE ⟶ V^3 ---- vtime ⟶ V^4 (for V^2)

Single Source

- Travelling salesman problem (TSP)
  NP hard problem
  T.C.: only exponential time algo. $(n-1)! > 2^n$
  but no polynomial time