Dt- 16.01.25

Computer - System + I/O Devices

= (CPU + main memory) + I/O Device
(RAM/ROM)

ALU is a IC

= (CALU+CU) + MM) + I/O Device.

+ → indicates direct connec^n b/w devices.

job → program in secondary memory.

process → transfer to main memory from secondary memory.
or
program in execution.

Algo → set of finite no. of steps to solve a particular problem.

Algo must satisfy following criteria :-

1) I/O Criteria
a) must take 0 or more than 1 i/p for generating o/p.
b) at least 1 o/p.

2) Effectiveness
each step must contribute to generate o/p.

3) Finiteness
no. of steps should be countable by execution not seem to be countable i.e infite loop is generated.

4) Definiteness
No ambiguity in the steps or no duality on steps. It should be a unique value in each step.
x = x - y ; a = y/z ; print x

Execution time -
The amount of time to generate o/p.
Let P is a program.

exec^n time of P = compile time + run time
E(P) = C(P) + R(P)

E(P) α R(P) if we once compile a C Program
In C Program when we compile a obj. code will be generated.

Process to write algo.

Algorithm Name of Algo. (Parameters)
// Define Parameters
I/P :-
O/P :-
BEGIN
1.
2. . .
END

Time depends on no. of i/p

ex Algorithm Add_two_nos (x, y, z)
// x & y are 2 i/p and z is O/P
Input - x, y
Output - z
BEGIN
1. Read x & y
2. z = x + y
3. Print z
END

Time depends on no. of i/p

E(P) -
Time(P) $\propto$ R(P)
$T(n) \longrightarrow \propto R(n)$
n → n i/p program
$T(n) \propto n$
$T(n) = kn$
k → +ve constant

So There are 3 situations arises :-

i) $T(n) \leq kn$ (Big Oh)
ii) $T(n) \geq kn$ (Big Omega)
iii) $k_1 n \leq T(n) \leq k_2 n$ (Theta)
n_0 → intial value
$k \geq n_0$

If $T(n) \leq kn \rightarrow O$
$T(n) < kn \rightarrow o$
If $T(n) \geq kn \rightarrow \Omega$
$T(n) > kn \rightarrow \omega$

} Asymptotic symbol.

Find Order of the eq^n's

$T(n) = 5n + 7$

$T(n) \leq kn$ for Big Oh

$T(n) \leq 5n + 7$

$T(n) \leq 5n + n$

$T(n) \leq 6n$ $\quad k = 6$

$\Rightarrow 5n + 7 \leq 6n$ $\quad [42 \leq 42]$

for $n_0 = 7$, the eq^n is satisfying.

$T(n) = 612$
$T(n) \leq 612 n^0$ $\quad k = 612$
$612 \leq 612 n^0$
$n_0 \geq 1$ $\quad O(n^0) = O(1)$

If it is constant it is always 1.

$T(n) \leq k\,g(n) \Rightarrow T(n) = O(g(n))$

where n → no. of inputs
$n_0 \geq n_0$, $n_0 \geq 0$

$T(n) = 2n^2 + 5$

$T(n) \leq 2n^2 + 5$

$\Rightarrow T(n) \leq 2n^2 + n$

$\Rightarrow T(n) \leq 2n^2 + n^2$

$\Rightarrow T(n) \leq 3n^2 \quad k = 3$

$\Rightarrow 2n^2 + 5 \leq 3n^2 \quad n_0 = 3$

$T(n) = O(n^2)$ constant get ignored.

$T(n) = 325$ 01.28.01.25

$T(n) \leq 325 \times n^0$

$325 \leq 325 \times n^0$

$k = 1, \; n_0 = 1$

$T(n) = O(1^0) = O(1)$

If $(T(n)) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_0$

$O(x^n)$

$T(n) = 2n^3 + n^2 + 2 = 2n^3 + n^2 + n^{n^2} = 2n^3 + 2n^{n^3} \leq 2n^3 + 2n^3 = 3n^3$

$T(n) \leq 3n^3$

$\Rightarrow 2n^3 + n^2 + 2 \leq 3n^3 \quad k = 3, \; n_0 = 2$

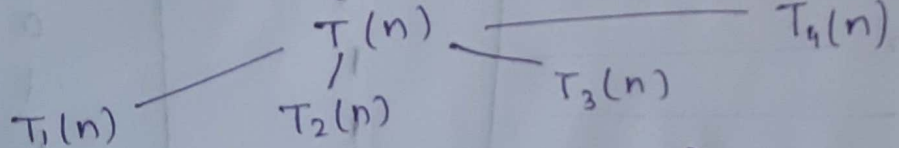
$O(n^3)$

$$T(n) = T_1(n) + T_2(n) = O(\max(g_1(n), g_2(n)))$$

$$T(n) = T_1(n) \cdot T_2(n) = O(g_1(n) \cdot g_2(n))$$

$$T(n) = T_1(n) / T_2(n) = O\left(g_1(n) \cdot \frac{1}{g_2(n)}\right)$$

$T(n) = 2n^2 \log(n^2 + 5n + 7) \log\log(n^3 + 7) + 2n^3 + \log(n^3 + 8)\, n^2 + n$



$T_1(n) = 2n^2 \log(n^2 + 5n + 7) \log\log(n^3 + 7)$

$T_2(n) = 2n^3$

$T_3(n) = \log(n^3 + 8)\, n^2$

$T_4(n) = n$

$T_4(n) = n = O(n)$

$T_2(n) = 2n^3 = O(n^3)$

$T_3(n) = \log(n^3 + 8)\, n^2$

Let $T_4(n) = \log(n^3 + 8)$, $T_6(n) = n^2$

$T_5(n) = \log(n^3 + 8) \qquad T_6(n) = O(n^2)$

$n^3 + 8 = O(n^3)$

$\log(n^3 + 8) = O(\log n^3) = O(3 \log n)$

$= O(\log n)$

$T_3(n) = O(n^2) \cdot O(\log n)$

$= O(n^2 \log n)$

$T(n) = 2n^2 \log(n^2+5n+7) \log\log(n^3+7)$

$T_7(n) = 2n^2 = O(n^2)$

$T_8(n) = \log(n^2+5n+7) = O(2\log n) = O(\log n)$

$T_9(n) = \log\log(n^3+7) = O(\log\log n)$

$T_1(n) = O(n^2 . \log n . \log\log n)$

$T_2(n) = O(n^3)$ Final order $O(n^3)$

$T_3(n) = O(n^2 \log n)$

$T_4(n) = O(n)$

Growing Order

$O(1) < n < \log n < n^2 < n\log n < n^3 < 2^n$

Step Count Method to derive a eqⁿ from a program / algo :-

Iterative method

Algo :- Sum of two nos (x_1, x_2, z)

// x_1 and x_2 are 2 nos as i/p and z is integer

i/p → Two nos x_1, x_2

o/p → Addⁿ of x_1 & x_2 as z.

BEGIN

1. Read x_1 & x_2
2. $z = x_1 + x_2$
3. print z

END

steps of execⁿ (s/e)	freq. of execⁿ (f/e)
0	0
0	0
0	0
0	0
0	0
1	1
1	1
1	1
0	0
Total freqⁿ = 3	

This will result $O(3) = O(1)$

$T(n) = 3$

For a for loop statement — for(int i=0; i<n; i++)

body of loop will execute n time

for will execute n+1 times.

BEGIN			
1. for (i=0; i<n; i++)	1	n+1	
2 {	0	0	
3. Read x_1 & x_2	1	n	
4. $z = x_1 + x_2$	1	n	
5. }	0	0	
6. Print z	1	1	
END	0	0	
	Total	$T(n) = 3n + 2 = O(n)$	

D1 - 29.01.25

control flow depend on program counter.

$$f(n) = \begin{cases} 2n+5 & n>3 \\ 5 & \text{otherwise.} \end{cases}$$

$$f(n) = \begin{cases} 4f(n/2)+7 & n>0 \\ 3 & \end{cases}$$ recursive

base value → termin^n of recursive program

Iterative method of solving eq^n:-
Direct recurance, Indirect recurance.

$$T(n) = \begin{cases} 2T(n/2) + cn & ; n>1 \\ a & ; \text{otherwise.} \end{cases}$$

cn → combine time & divide time.

a → base value.

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

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

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

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

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

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

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

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

At 3rd step,

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

At ith step,

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

$$= n\,T\left(\frac{n}{n}\right) + \log_2 n \cdot cn$$

$$= nT(1) + \log_2 n \cdot cn$$

$$= n \cdot a + cn \log_2 n$$

Let $2^i = n$

$i = \log_2 n$

$= an + cn \log_2 n$

$O(n \log_2 n)$

Iterative method of solving recurrance eq^n:-

Algorithm Recursiveaddn(a, n) — 0

// a is an array contain n no. of elements. — 0

I/p - array a with 'n' no.s. of elements. — 0

O/p - sum of 'n' no.s — 0

BEGIN — 0

1. sum = 0
   if (n ≤ 0)
2. {
3. Return (sum)
4. }
5. else
6. {
7. return (sum + a[n] + Recursiveadd (a, n-1))
8. }

END.

s/e	
n ≤ 0	n > 0
1	1
1	
1	
0	
0	

f/e	
n ≤ 0	n > 0
1	1
1	0
1	0
	2 + T(n-1)
3	3 + T(n-1)

$$T(n) = \begin{cases} 3, & n \le 0 \\ 3 + T(n-1), & n > 0 \end{cases}$$

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

2nd $T(n-1) = 3 + (3 + T(n-2))$
$= 6 + T(n-2)$

3rd $T(n-2) = 9 + T(n-3)$

at 3rd step :-

$T = 3 \cdot 3 + T(n-3)$

At i^{th} step

$= i \cdot 3 + T(n-i)$

let $i = n$

$3n + T(n-n)$

$= 3n + T(1)$

$= 3n + 3$

$\Rightarrow O(n)$

storage → matrix

R - loc^n → no.

C - cell

consequetively storage - linear.

fragmented storage - non-linear.

Binary :- 2-1 Tree.

B-Tree - N - (N-1) Tree

↓ no. of child ↓ Data

H - Longest path from root to leaf.

Depth - leaf to node.

Binary - no. of node - 2^{level} - no. of child.

Recurrance eq^n solving :-

i) Substitution method

ii) Tree / Recurrance tree method

iii) Master method

iv) Changing variable method.

Dt 30.01.25

Tree / Recurrance Tree Method

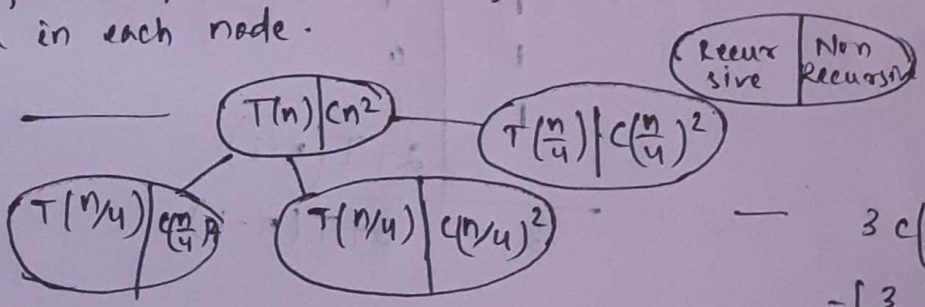
$T(n) = 3T(n/4) + cn^2$

Prob is divided into 3 problems of $T(n/4)$

cn^2 - (cost / const → non recursive part)

3 → no. of child of a node → no. of subproblems

2 → data in each node.



$3c\left(\frac{n}{4}\right)^2$

$= \frac{3}{16} cn^2$

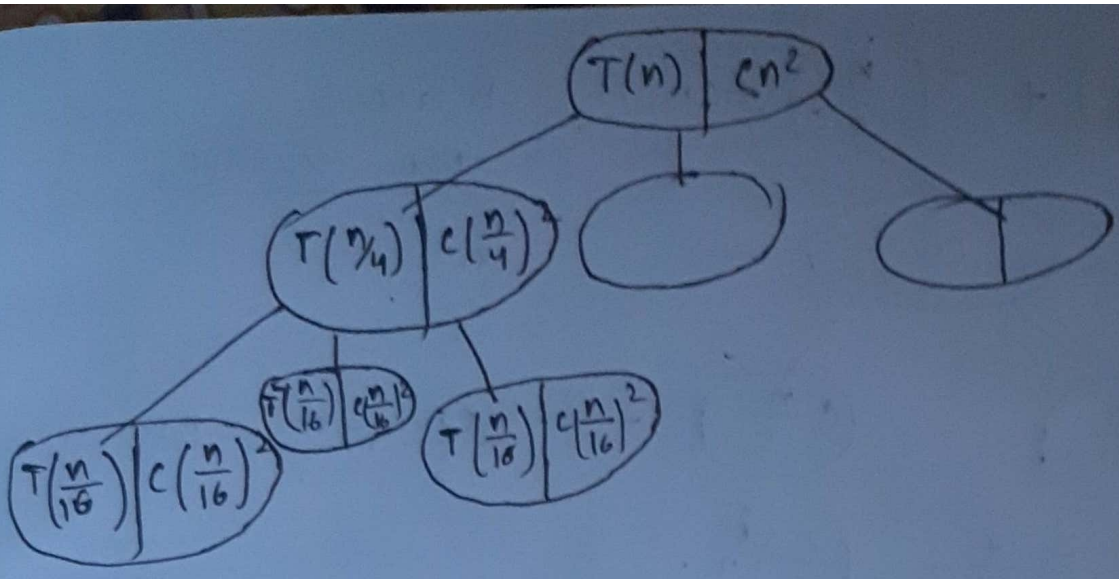
calculate the cost →

cost is the non-recursive part.

$9 \times T\left(\frac{n}{16}\right) c\left(\frac{n}{16}\right)^2$

2nd $9 \times c\left(\frac{n}{16}\right)^2$

$= \left(\frac{9}{16}\right)^2 \times cn^2 = \left(\frac{3}{16}\right)^2 cn^2$



$\left(\frac{3}{4}\right)^0 cn^2$

$\left(\frac{3}{4^2}\right)^1 cn^2$

~~$9 \times \frac{n^2}{16^2} cn^2$~~

$9 \times c\left(\frac{n}{16}\right)^2$

$= \frac{9}{16^2} \times cn^2$

$= \left(\frac{3}{16}\right)^2 \times cn^2$

$= \left(\frac{3}{16}\right)^2 \times cn^2$

$= \left(\frac{3}{4^2}\right)^2 \times cn^2$

For i, $= \left(\frac{3}{4^i}\right)^2 \times cn^2$

No. of nodes in i^{th} level $= 3^i$

$n^2 \to \left(\frac{n}{4}\right)^2 \to \left(\frac{n}{4^2}\right)^2 \to \cdots \left(\frac{n^2}{4^i}\right)^2 = 1$

$\left(\frac{n}{4^i}\right)^2 = 1 \Rightarrow \frac{n}{4^i} = 1 \Rightarrow n = 4^i \Rightarrow \boxed{i = \log_4 n}$

no. of leaf nodes $= 3^{\log_4 n} = n^{\log_4 3}$

$i \to n-1$ level.

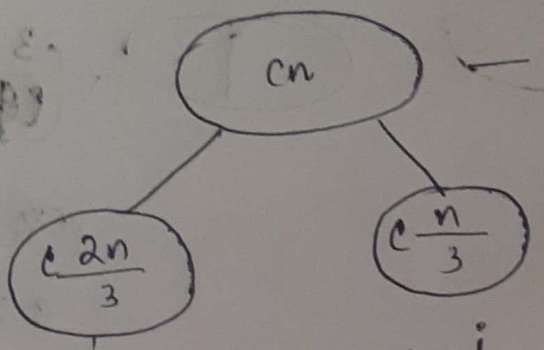
cost $= \sum_0^i \left(\frac{3}{4^i}\right)^2 cn^2 + n^{\log_4 3} = \sum_0^{\log_4 n} \left(\frac{3}{4^i}\right)^2 cn^2 + n^{\log_4 3}$

$= \sum_0^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + n^{\log_4 3}$

$= an^2 + b \quad = O(n^2)$

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

B Tree
non- recursive



cn

$\frac{2cn}{3} + \frac{cn}{3} = cn$

$n \to \frac{2n}{3} \to \frac{4}{9}n \to \frac{8}{16} \to \cdots \left(\frac{2}{3}\right)^i n$.

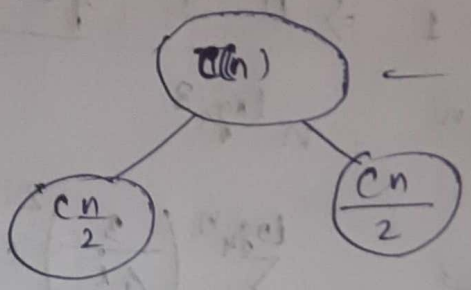
$\left(\frac{2}{3}\right)^i n = 1 \Rightarrow n = \left(\frac{3}{2}\right)^i \Rightarrow i = \log_{\frac{3}{2}} n$

leaf node - $T(1)$ - cn

i times cn $\to \log_{\frac{3}{2}} n \, (cn) = cn \log_{\frac{3}{2}} n$

$= O(n \log_{\frac{3}{2}} n)$

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



B-Tree -
cn

cn

$n \to \frac{n}{2} \to \frac{n}{4} \to \frac{n}{2^i} = 1$

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

$cn + cn \cdots \log_2 n$ times.

$= cn \log_2 n \Rightarrow cn \log_2 n = O(n \log_2 n)$

Dt-04-02-25

Master Method

Solving recurrance eq^n by using Master Method

Cond^{n}s :-

$T(n) = a\,T(n/b) + f(n)$ — $a \geq 1$: coefficient — no. of subproblems

i) $a \geq 1$ → no. of subproblem

ii) $b > 1$

iii) $f(n)$ → combined time of 'n' i/p & divide time of 'n' i/p

$f(n) = C(n) + D(n)$

There 3 Theorems :-

i) If $f(n) = O(n^{\log_b a - \epsilon})$ and $\epsilon > 0$

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

ii) If $f(n) = O(n^{\log_b a})$

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

iii) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $\epsilon > 0$,

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

ex: i) $T(n) = 9\,T(n/3) + n$

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

Q1 $T(n) = 9\,T(n/3) + n$ solve using master method.

$T(n) = 9\,T(n/3) + n$

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

given $a = 9$, $b = 3$, $f(n) = n$

i) Calculate $f(n)$

$f(n) = n^{\log_b a} = n^{\log_3 9} = n^2$

Increasing given $f(n)$ i.e n to n^2

Applying the 1st Theorem if $f(n) = O(n^{\log_b a - \epsilon})$ and $\epsilon > 0$

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

So the answer will be $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$ or $O(n^2)$

Q2) $T(n) = 2(T(n/2)) + O(n)$

$a = 2$; $b = 2$ $\quad f(n) = \Theta(n)$

Calculate $n^{\log_b a}$

$f(n) = n^{\log_b a} = n^{\log_2 2} = n$ both the given & calculated $f(n)$ are same.

Applying Theorem 2, i.e if $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$

So,

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

Q $T(n) = 2T\left(\frac{2n}{3}\right) + 1 = 2T\left(\frac{2n}{3}\right) + n^0$

Given $a = 1$, $b = \frac{3}{2}$ $\quad f(n) = n^0 = 1$

Calculate $f(n) = n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$

Given and calculated are same.

So applying Theorem. 2, $f(n) = \Theta(n^{\log_b a})$ Then

$T(n) = \Theta(n^{\log_1 a} \log n)$

Then,

$T(n) = \Theta(n^{\log_b a} \cdot \log n) = \Theta(1 \cdot \log n) = \Theta(\log n) = O(\log n)$

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

$a = 3 \qquad f(n) = n\log n$

$b = 4$

Cal. $f(n) = n^{\log_4 3} = n^0 = 1$ $\qquad \log_4 3 < 1$

$1 < n\log n$

By Apply Theorem 3,

if $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ and $\epsilon > 0$, $T(n) = \Theta(f(n))$

so,

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

Assignment

Q: i) $T(n) = 4T\left(\frac{n}{2}\right) + n^2$ iv) $T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$

ii) $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$ v) $T(n) = 8T\left(\sqrt{n}\right) + n$

iii) $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$

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

Given $a = 4$, $b = 2$, $f(n) = n^2$

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

Th2 $T(n) = \Theta(n^{\log_2 4}\log n) = \Theta(n^2\log n)$

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

Given $a = 2$, $b = 2$, $f(n) = n\log n$

Calculate $f(n) = n^{\log_b a} = n^{\log_2 2} = n$

Th-3 $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $\epsilon > 0$, $T(n) = \Theta(f(n))$

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

iii) $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$

$a = 1$, $b = 2$, $f(n) = \Theta(1) = n^0 = 1$

Calculate $f(n) = n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

$T(n) = \Theta(n^{\log_2 1}\log n) = \Theta(n^0\log n) = \Theta(\log n) = O(\log n)$

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

$a = 2, \quad b = 2, \quad f(n) = \sqrt{n}$

Calculate $f(n) = n^{\log_b a} = n^{\log_2 2^2} = n$

Th-1, If $f(n) = n^{\log_b a - \epsilon}$, $T(n) = n^{\log_b a}$

So $T(n) = \Theta(n^{\log_2 2}) = \Theta(n^1) = \Theta(n)$

v) $T(n) = 2T(\sqrt{n}) + n$

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

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

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

$T(2^k) = 2T(2^{k/2}) + 2^k$ let $T(2^k) = S(k)$

$S(k) = 2S(k/2) + 2^k$

$a = 2; \quad b = 2; \quad f(k) = 2^k$

$f(k) = k^{\log_b a} = k^{\log_2 2} = k^1 = k$

As $k < 2^k$

vi) $T(n) = 3T(n^{1/3}) + \log_3 n$ let $n = 3^m$, $m = \log_3 n$

$\Rightarrow T(3^m) = 3T(3^{m/3}) + m$ let $T(3^m) = S(m)$

$\Rightarrow S(m) = 3S(m/3) + m$

$a = 3, \quad b = 3, \quad f(m) = m$

$f(m) = m^{\log_3 3} = m$

$O(m \log m) = (\log_3 n \log \log_3 n)$

Algorithm Insertion (A, n)

// A is an array with 'n' no. of ele.

I/P - array with n -

O/P - .

BEGIN

1. for j = 2 to length(A)	1	n
2. {		
3. key = A[j]	1	n-1
4. i = j-1	1	n-1
5. while (i>0 and A[i]>key)	1	$\sum_{j=2}^{n} t_j$
6 {		
7. A[i+1] = A[i]	1	$\sum_{j=2}^{n} t_j - 1$
8. i = i-1	1	$\sum_{j=2}^{n} t_j - 1$
9. }		
10. A[i+1] = key	1	n-1
11 }		

END

1	2	3	4	5
2	1	7	5	4

i. j = 2 to 5

j = 2

key = A[j] = A[2] = 1

i = 2-1 = 1

1 2 7 5 4

2. j = 3

key = 7

i = 2-2 = 1

A[3] = 7

1 2 7 (5) 4

3. j = 4

key = 5

i = 4-1 = 3

Total cost = $n + n-1 + n-1 + \sum_{j=2}^{n} t_j + \sum_{j=2}^{n} t_j - 1 + \sum_{j=2}^{n} t_j - 1 + n-1$

Case-I

For all elements are same and sorted.

$\sum_{j=2}^{n} t_j - 1 + \sum_{j=2}^{n} t_j - 1$ not executed

Case-II (Reverse order) (worst case)

Decreasing order all elements costs are executed (Reverse

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

Case-III

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

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

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

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

$$= 4n - 3 + \frac{n^2+n-2}{2} + n^2 - 2n = O(n^2)$$

Case-I (Best Case)

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

$$= 4n - 3 + n = 5n - 3 = O(n)$$

Average case (some are sorted and some are unsorted)

$O(n^2)$

Techniques to Design Algorithm

i) Divide & Conquer
ii) Dynamic Programming
iii) Backtracking
iv) Greedy Method
v) Branch and Bound

Divide & Conquer

It is a bottom-up approach. (solve) divide (top-down)

General concept of algorithm :-

```
Algorithm DandC(P)
// P is a problem
BEGIN
1. If small(P)
2. return solution(P)
3. else
4. Divide the (P) into P1, P2 .... Pk
5. return (combine (solution(P1), S(P2) .... S(Pk)))
END.
```

$$T(n) = \begin{cases} g(n) & n \text{ is small} \\ T(n_1) + T(n_2) + \cdots + T(n_k) & \text{when } n \text{ is large.} \end{cases}$$

$$T(n) = \begin{cases} T(1) & ; n = 1 \\ aT(n/b) + \dfrac{c(n) + D(n)}{f(n)} f(n) \end{cases}$$

a → no. of subproblem

n/b → each subproblem size

C(n) combine time D(n) - divide time

General complexity of Divide and Conquer :-

$$\Rightarrow T(n) = \begin{cases} T(n) & ; n = 1 \\ aT(n/b) + f(n) \end{cases}$$

// Algorithm Quick Sort

⇒ if $U \cdot B > L \cdot B$, problem is large
before
⇒ if $L \cdot B < U \cdot B$, and have to divide.
if $L \cdot B = U \cdot B$ no divide

$T(n) \leq c\, g(n) \Rightarrow T(n) = O(g(n))$

If $T(n) < c\, g(n) \Rightarrow T(n) = o(g(n))$

$T(n) = 5n^2 + 6n + n = O(n)$ check whether it is big oh of n.

**

$\lim_{n \to \infty} \frac{f(n)}{g(n)}$ { constant $\neq 0$, zero, infinite

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

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

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

$f(n) = o(g(n))$

$T(n) = 5n^2 + 6n + n$

$g(n) = O(n) = cn$

$\Rightarrow \lim_{n \to \infty} \frac{5n^2 + 6n + n}{n}$

$= \lim_{n \to \infty} \frac{5n^2}{n} + \lim_{n \to \infty} \frac{6n}{n} + \lim_{n \to \infty} \frac{n}{n}$

$5n + 6 + 1$ is not constant. So it is false.

Dt 06/02/25

Algorithm Quicksort(A, P, r) P → starting index, r → ending index

//A is an array

input: array

output: sorting of elements.

BEGIN

1. If $p<r$
2. q = partition(A, P, r)
3. Quicksort(A, p, q-1)
4. Quicksort(A, q+1, r)

END

Algorithm Partition(A, P, r)

// A is an array

input: array

output: sorting of elements

BEGIN

1. $x \leftarrow A[r]$	1	1
2. $i \leftarrow p-1$	1	1
3. for $j \leftarrow p$ to $r-1$	1	$r-1-p+1 = (r-p)$
4. if $A[j] \leq x$	1	$r-1-p$
5. $i = i+1$	1	$r-1-p$
6. exchange $A[i] \leftrightarrow A[j]$	1	$r-1-p$
7. exchange $A[i+1] \leftrightarrow A[r]$	1	1
8. Return $i+1$	1	1

END

$4 + 4r - 4p - 3$

$1 + 4r - 4p$

$O(n)$

Worst case if pivot element is from one end

$T(n) = T(n-1) + T(0) + O(n)$

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

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

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

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

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

$O(n^2)$

if all elements are same
3, 3, 3, 3
all elements are in one side.

Average case

when pivot element divide into 2 equal sub array

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

$a = 2 \quad b = 2 \quad f(n) = O(n)$

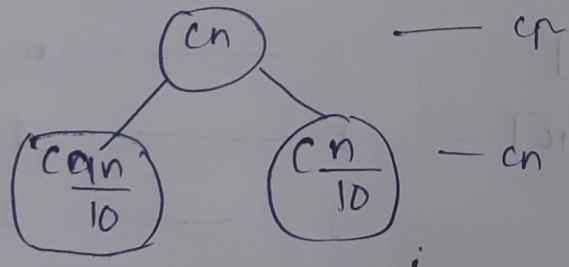
$f(n) = n^{\log_2 2} = n$

As they are same as given then $T(n) = n \log n$

Best case

$T(n) = T\left(\frac{90n}{100}\right) + T\left(\frac{10n}{100}\right) + cn$

$= T\left(\frac{9}{10}n\right) + T\left(\frac{n}{10}\right) + cn$



$\frac{9n}{10} \times \frac{9n}{10} = \frac{81}{100}n$ in 2nd step

$n \to \frac{9n}{10} \to \frac{81n}{100} \to \cdots \left(\frac{9}{10}\right)^i n = 1$

$\left(\frac{9}{10}\right)^i n = 1$

$\Rightarrow \left(\frac{10}{9}\right)^i = n$

$\Rightarrow i = \log_{10/9} n$

$cn + cn + \cdots$ ($\log_{10/9} n$ time) $= cn \times \log_{10/9} n$

$T(n) = O(n \log_{10/9} n)$

D1 - 21.02.25

Cond^n for the problem

If LB < U.B, then the problem is large.

Algorithm MergeSort (A, p, r)

// A is an array where p is the lower bound and r is upper bound.

i/p -

o/p - Sorted array.

```
BEGIN
1. If (p < r)                          ——————   1
2. mid(q) = (p+r)/2                    ——————   1
3. Mergesort(A, p, q)                  ——————   (p+r)/2
4. Mergesort (A, q+1, r)                        (p+r)/2
5. Merge (A, p, q, r)                           O(n)
END.
```

Algorithm Merge (A, p, q, r)

//

```
BEGIN                                           1
1. n1 = q - p + 1                               1
2. n2 = r - q
3. create arrray L[1 .... n1+1] and R[1 ... n2+1]   1
4. for i = 1 to n1                              n1+1
5.      L[i] = A[p + i - 1]                     n1
6  for j = 1 to n2                              n2+1
7.      R[j] = A[q + j]                         n2
8. L[n1+1] = ∞ , R[n2+1] = ∞                    1
```

```
9.  i ← 1                                   1
10  j ← 1                                   1
11. for k = p to r                          r-p+1
12      if L[i] ≤ R[j]                      r-p
13.         A[k] = L[i]                     r-p
14          i ← i+1                         r-p
15      else                                0
16          A[k] = R[j]                     r-p
17          j ← j+1                         r-p
18  Forend
19. END
```

Total = $6r - 6p + 9 + 2n1 + 2n2$

$= 6r - 6p + 9 + 2(q-p+1) + 2(r-q)$

$= 6r - 6p + 9 + 2q - 2p + 2 + 2r - 2q$

$= 8r - 8p + 11$

$$T(n) = \begin{cases} O(1) & ; n = 1 \\ 2T(n/2) + O(n) + d(n) & ; \text{otherwise.} \\ = 2T(n/2) + O(n) + 1 \\ = 2T(n/2) + O(n) \end{cases}$$

Solving By master method.

$a = 2, b = 2, f(n) = O(n)$

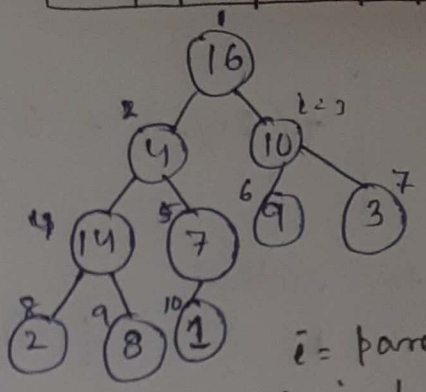
$f(n) = n^{\log_2 2} = n = O(n)$

$T(n) = n \log n$ in all cases → It is better.

Insertion sort $T(n) = n^2$

Algorithm maxHeapify (A, i) i=2

1	2	3	4	5	6	7	8	9	10
16	4	10	14	7	9	3	2	8	1



i = parent
2×i - Left child
2i+1 - right child

1. l = left [i] (indexes)
2. r = right [i]
3. If l ≤ heap-size (A) and A(l) > (A(i)
4. then largest = l

else

largest = r

5. if r ≤ heap-size(A) and A[r] > A[largest]
6. largest = r
7. if largest ≠ i // i = largest terminate from algo
8. exchange A[i] ⟷ A[largest]
9. maxHeapify(A, largest)

Algorithm minHeapify(A, i)

// A is the array

i → index where the distortion of min heap starts.

1. l = left [i]
2. r = right [i]
3. If l ≤ heap-size (A) and A[l] < A[i]
4. then smallest = l
5. If r ≤ heap-size (A) and A[r] < A[smallest]

smallest = r

if smallest ≠ i

exchange A[i] ⟷ A[smallest]

minHeapify (A, smallest)

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

Master method,

$a = 1, \quad b = \frac{3}{2}, \quad f(n) = O(1)$

$f(n) = n^{\log_{3/2} 1} = n^0 = 1$ which is same with given $f(n)$

$T(n) = n^{\log_{3/2} 1} \log n = 1 \cdot \log n$

$T(n) = O(\log n) = O(h)$ → height of the binary tree.

Algorithm BUILD_MAX_HEAP(A)

// A → Array consist of set of element

i/p - Array element

o/p - To build max heap

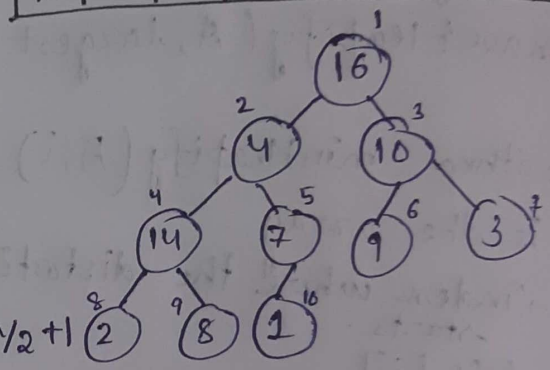
BEGIN

1. heap-size(A) = length(A) — 1
2. for i = length(A)/2 down to 1 — n/2+1
3. MAX-HEAPIFY(A, i) - n/2(log n)

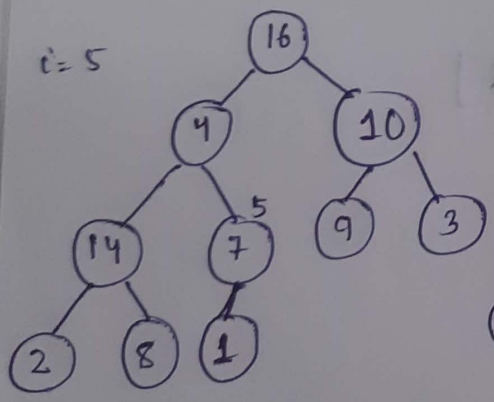
END

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

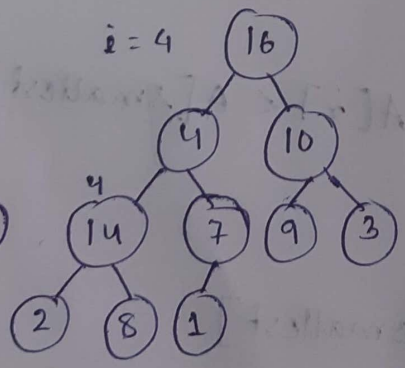
16	4	10	14	7	9	3	2	8

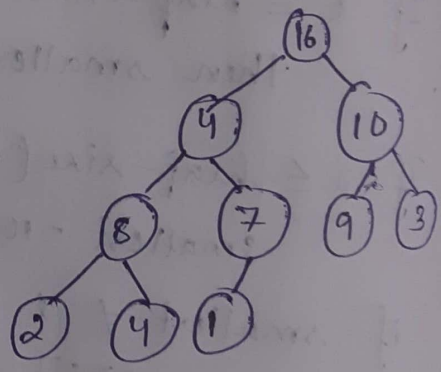


i=5



i=4





Algorithm Heapsort (A)

BEGIN

1 BUILD-MAXHEAP (A)

2. heap_size = length(A)

3. exchange A[1] $\longleftrightarrow$ A[heap_size]

4. ~~for~~ heapsize = heapsize - 1

5 for i=1 to heapsize

6 maxheapify(A, i)

7. for end

END

Algorithm Heapsort (A) ——— nlogn

Build Heap(A)

1 for i = length(A) down to 2 — n+1-2 = n-1

2. exchange A[1] $\leftrightarrow$ A[i] —— n-2

3. heap-size(A) = heapsize(A) - 1 — n-2

4. Heapify (A, 1) — ~~n-2 (logn)~~ $\sum_{h=0}^{\log_2 n} \left(n/2^{h+1}\right) O(h)$

$n\log n + n - 1 + n - 2 + n - 2 + (n-2)(\log n)$

$= n\log n + 3n - 5 + n\log n - 2\log n = O(n\log n)$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}$$

$$\sum_{h=0}^{\log n} \frac{n}{2^{h+1}} \cdot ch = \sum_{h=0}^{\log n} n\left(\frac{h}{2^{h+1}}\right) = \sum_{n=0}^{\log n} \frac{n}{2}\left(\frac{h}{2^h}\right)$$

$$= \frac{n}{2} \sum_{h=0}^{\log n} \left(\frac{h}{2^h}\right) = \frac{n}{2}$$

$O(n)$

Application of Heap

i) maintain priority by
ii) return root in max heap
iii) new arrival of program/ increasing key.
iv) job of replacement the old job by new job
new job priority > old job priority.

i) Algorithm Heap Maximum (A)
1. return (A[1])

$T(n) = O(1)$

ii) Algorithm Heap-EXTRACT_MAX (A) (find max element & remove from tree)

1. Check underflow or overflow
1 if heapsize (A) < 1 ——— 1
2 print underflow ——— 1
~~if heapsize~~
3 max ← A[1] ——— 1
4 A[1] ⟷ A [heapsize(A)] ——— 1
5. heapsize (A) = heapsize(A) − 1 ——— 1
6. Heapify (A, 1) ——— O(log n)
7 return max ——— 1

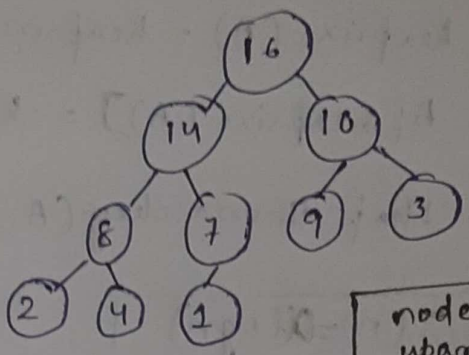
$T(n) = O(\log n)$

iii) increase key.
Can be inserted if key value > existing key.

iii) Algorithm maxHeapincreasekey(A, i, key) (with replacement)

i = posn

k = key value



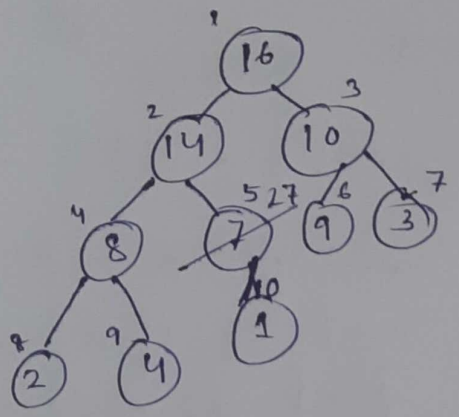
1. If key < A[i]
2. cant insert (error)
   else
3. A[i] ← key
4. while i > 1 and A[parent[i])] < A[i] O(log i)

node se upar jana pot tak (depth)

exchange A[i] ⟷ A[parent(i)]

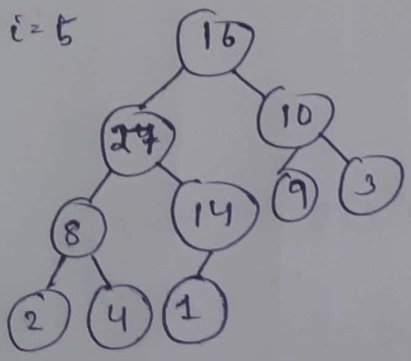
i ← parent(i)

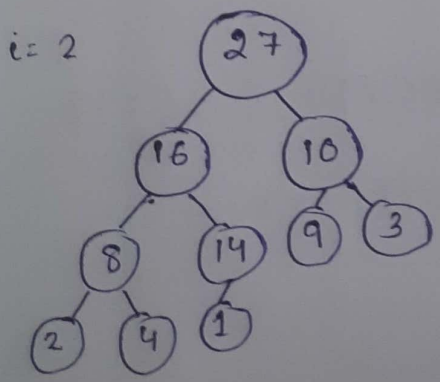
O(log n)



i = 5

key = 27





Algorithm Heap-Insert (A, key) (no replacement but ending)

1. heapsize (A) = heapsize (A) + 1
2. A[heapsize (A)] = key
3. Heap-Increasekey (A, heapsize, key)

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

Dynamic Programming

Dt - 18/02/25

used to calculate the optimal sol^n.

Characterise the optimal structure.

A_1 2×3

A_2 3×5

A_3 5×4

$((A_1 A_2) A_3) \rightarrow 70$

$(A_1 (A_2 A_3)) - 84$

chain matrix multiplc^n → order how to multiply matrix so that the cost will be minimum.

$A_1 \ldots 4$

$i < j$ → Trival sol^n - Trival sol^n.

$i = j$ → Non-trival - '0' cost

$A_i \ldots 4$

$i \ldots k, k+1 \ldots j$ k → middle element that from 2 group.

$(A_i \ldots k)(A_{k+1} \ldots j)$

$A_1 \ldots k \cdot k+1 \ldots j$.

$$P(n) = \begin{cases} 1 & ; i = j \\ \sum_{k=1}^{n-1} P(k) P(n-k) & i < j \end{cases}$$

If matrix is 'n' in no. the total no of parentheris

$$P(n) = \begin{cases} 1 & ; i = j \\ \sum_{k=1}^{n-1} P(k) P(n-k) & ; i < j \end{cases}$$

$$P(3) = \sum_{k=1}^{2} P(k) P(3-k)$$

$$= P(1) P(3-1) + P(2) P(3-2) = P(1) P(2) + P(2) P(1)$$

$$= 2 P(2) + P(2)$$

$$= 1 + 1 = 2$$

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

$$= P(1) P(2-1) = P(1) P(1) = 1 \cdot 1 = 1$$

$$P(4) = \sum_{k=1}^{n-1} P(k)\,P(n-k)$$

$$= \sum_{k=1}^{4} (P(k))\,P(4-k)$$

$$= P(1)P(3) + P(2)P(2) + P(3)P(1)$$

$$= 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 \quad = 2 + 1 + 2 = 5$$

$(A_1 A_2)(A_3 A_4)$

$(A_1(A_2 A_3) A_4)$

$(A_1 (A_2 (A_3 A_4)))$

$((A_1((A_2 A_3) A_4))$

$((A_1 A_2) A_3) A_4)$

Calculate optimal substructure value.

$A_i \cdots k \cdots j$

$(A_{i \cdots k})(A_{k+1 \cdots j})$

$$\text{cost} = m(i,j) = \begin{cases} 0 & i = j \\ \min_{i \le k < j} \{ m(i,k) + m((k+1), j) + P_{i-1} P_k P_j \} & i < j \end{cases}$$

Q size of matrix :-

$A1_{30 \times 35}$ $P_0 \times P_1$

$A2 - 35 \times 15$ $P_1 \times P_2$

$A_3 - 15 \times 5$ $P_2 \times P_3$

$A_4 - 5 \times 10$ $P_3\ P_4$

$A_5 - 10 \times 20$ $P_4\ P_5$

$A_6 - 20 \times 25$ $P_5\ P_6$

i ↓ \ j →	1	2	3	4	5	6
1	0					
2	x	0				
3	x	x	0			
4	x	x	x	0		
5	x	x	x	x	0	
6	x	x	x	x		0

$$m[1,2] = \min_{1 \le k < 2} m[1,1] + m[2,2] + P_0 \times P_1 \times P_2$$

$$= 0 + 0 + 30 \times 35 \times 15 = 15750$$

Dt - 19.10.21

algorithm - Matrix-CHAIN-MULTIPLICATION (p, n)

//

```
for i = 1 to n
    m[i, i] = 0

for i = 2 to n
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] = q, s[i, j] = k
```

$O(n^3)$

return m & s

	p_0		p_1		p_2		p_3		p_4		p_5		p_6
A_1	30	x	35										
A_2			35	x	15								
A_3					15	x	5						
A_4							5	x	10				
A_5									10	x	20		
A_6											20	x	25

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

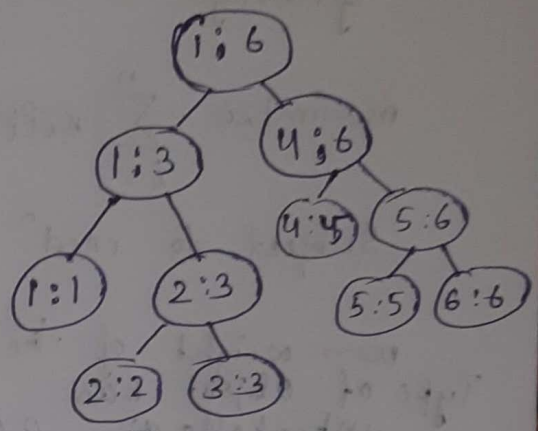
(parenthesis possible ways to put parenthesis.

	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

Algorithm Print-optimal-par(s, i, j)

```
1  if i = j
2     Print "Ai"
3  else print "("
4     print-optimal-par(s, i, s[i, j])
5     print-optimal-par(s, s[i, j]+1, j)
6     print ")"
```

((A1(A2A3))



Dt- 25/02/25

Knapsack Problem :-

The object $x_1, x_2, x_3 \ldots x_n$

weight $w_1, w_2, w_3 \ldots w_n$

profit of each obj $- p_1, p_2, p_3 \ldots p_n$

maximize $\sum_{i=}^{n} x_i p_i$

Subject to cond^n $\sum_{i=1}^{n} x_i w_i \leq w$

$w \rightarrow$ weight of the knapsack.

Type of Object basis

unbrakable obj - 0/1 knapsakk

brakable obj - focactional knapsack

0/1 knapsack :-

Sol^n Tabular method
Set Method.

Obj → x_1 x_2 x_3 x_4

feasible sol^n = 2^4 (possible ways)

Q $w = 8$

x_1	x_2	x_3	x_4	$w_1 = 2$	$w_2 = 3$,	$w_3 = 4$,	$w_4 = 5$
0	1	0	1	$p_1 = 1$	$p_2 = 2$,	$p_3 = 5$,	$p_4 = 6$

Table name-V

p	w	i \ w	w_0 0	w_1 1	w_2 2	w_3 3	w_4 4	w_5 5	w_6 6	w_7 7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

$$V[4,6] = \max[V[3,6], V[3, 6-5] + 6]$$

$$= \max[6, \quad 0+6] \quad = 6$$

$$V[4,7] = \max[V[3,7], V[3, 7-5] + 6]$$

$$= \max[7, \quad 1+6] \quad = \max[7, \quad 7] = 7$$

$$V[4,1] = \max[V[3,1], V[3, 1-5] + 6] = \max[0, \text{not exist}]$$

(not exist)

$$= 0$$

$$V[i,w] = \max[V[i-1, w], V[i-1, w - w[i]] + P_i]]$$

Q-1 Find optimal cost of putting the object into knapsack

Q-2 Find sol^n vector space for selecting/inserting obj. into the knapsack.

Set theory

P = (1 2 5 6) KW = 8 ; n = 4

W = (2 3 4 5)

we have to create 4 sets. n = 4

S^0 S^1, S^2, S^3, S^4 → S^n sets for n object.

S^0 → initial set when object we are adding is zero.

$S^0 = \{(0, 0)\}$ (P, W)

1) weight doesnot exicead KW
2) weight should be in increasing o
3) If P↑ W↓ th discard:

S_1^0 → subset of S^0 = $\{(0,0), (1,2)\}$

$S^1 = \{(0,0), (1,2)\}$ $(S_0 \cup S_1^0)$

$S_1^1 = \{(2,3), (3,5)\}$

$S^2 = S^1 \cup S_1^1 = \{(0,0)(1,2)(2,3), (3,5)\}$

$S_1^2 = \{(5,4)\ (6,6)\ (7,7)\ (8,9)\}$

$S^3 = S^2 \cup S_1^2 = \{(0,0)(1,2)(2,3)(3,5)(5,4)(6,6)$ (P↑ W↓)

$(7,7)\ (8,9)\}$

cond^n → P↑ W↑
W should not exceed KW

$S^3 = \{(0,0)(1,2)(2,3)(3,5)(6,6),(7,7)\}$

$S_1^3 = \{(6,5)\ (7,7)\ (8,8)\ (9,10)$ ~~(11,9)~~ $(12,11)$

$(13,12)$ ~~(14,13)~~ $\}$

$S^4 = \{(0,0)\ (1,2)\ (2,3)\ (3,5)\ (6,6)\ (7,7)$.

$(6,5)\ (8,8)\}$

Select^n criteria to be added

$(8,8) \in S^4$

If $(8,8) \in S^3$ the S^3 will be selected else S^4 is taken.

$(8,8) - (6-5) = (2,3)$

check if $(2,3)$ in S^3 and check if $(2,3)$ also in S^2

so S^2 will be considered.

$(2,3) - (2,3) = (0,0)$

soln vector space

x_1	x_2	x_3	x_4
0	1	0	1

Note :- Make sorting acc^n to profit then order it.