Recursion Tree Method:-

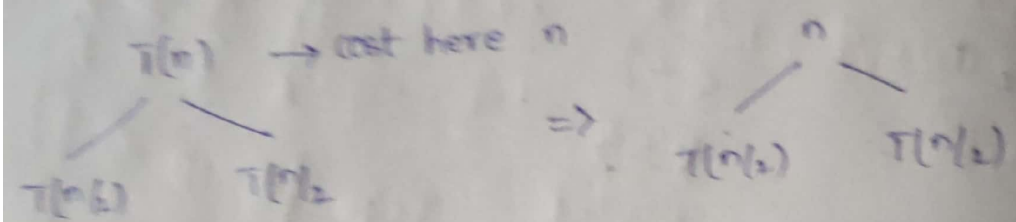
1) $x^{\log_b a} = a^{\log_b x}$

2) $x^0 + x^1 + x^2 + \dots + x^n = \frac{x^{n+1} - 1}{x-1}$ $\quad x \neq 1, |x| > 1$

3) $\quad = \frac{1}{1-x}$ $\quad |x| < 1$

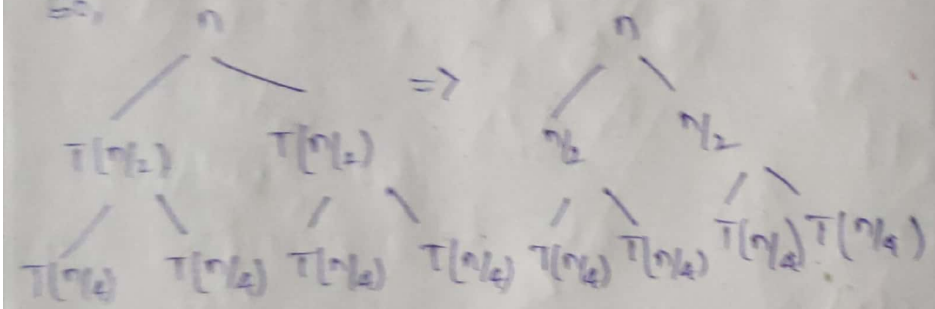
4) $T(n) = 2T\left(\frac{n}{2}\right) + n$

$T(n)$ → cost here n

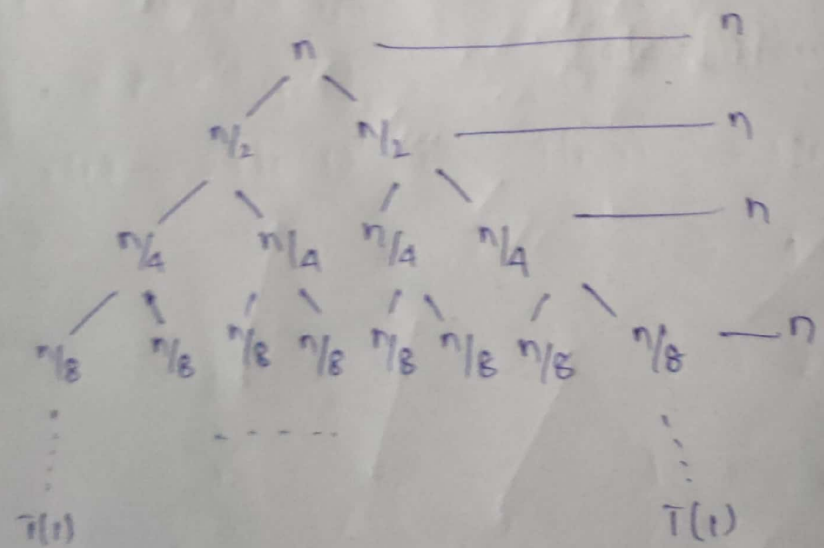


For cost of $T(n/2) = 2T\left(\frac{n}{2^2}\right) + \frac{n}{2}$

So,



The recursive tree is



Cost of the tree = cost of all leaf nodes + cost of all internal nodes.

$= L_C + I_C$

No: of internal nodes = ?

No: of levels be k

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

$\Rightarrow k = \log_2 n$

No: of leaf nodes $= 2^{k-1}$

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

$[\because x^{\log_y n} = n^{\log_y x}]$

So, $L_C = \frac{n}{2}$

For I_C, we have $k-1$ internal levels.

Each level cost is ~~1~~ n

So, $I_C = (k-1)n = (\log_2 n - 1)n$

$= n \log_2 n - n$

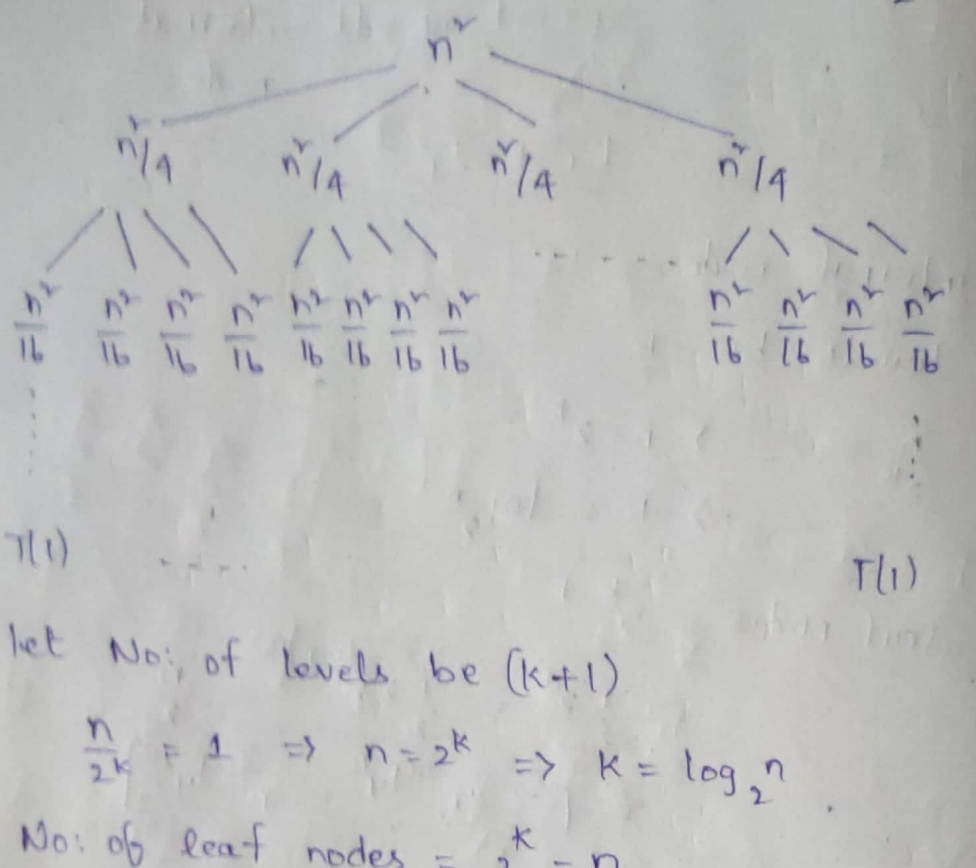
Now, $T(n) = L_C + I_C$

$= \frac{n}{2} + n \log_2 n - n$

$= n \log_2 n - n/2$

$\in O(n \log_2 n)$

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



let No: of levels be $(k+1)$

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

No: of leaf nodes $= 2^k = n$

$L_c = n(1) = n$

Cost at each level $= n^2$

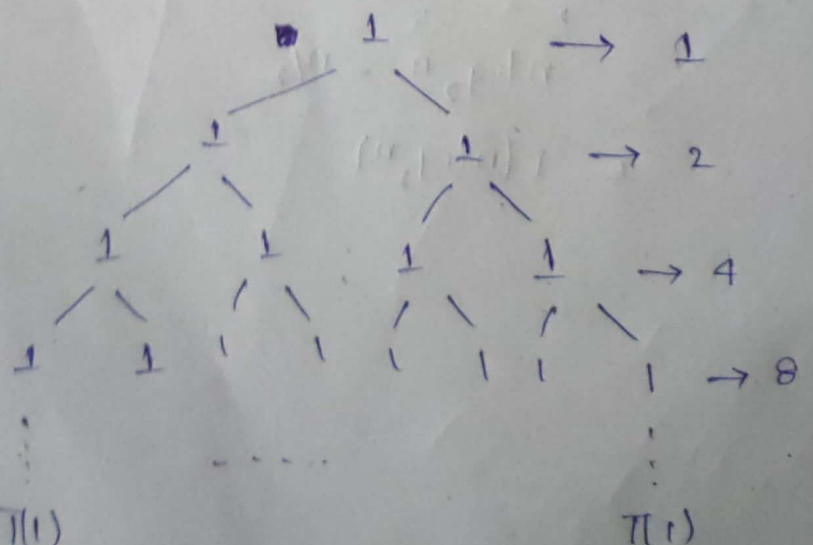
No: of internal nodes levels $= k$

$I_c = kn^2 = n^2 \log_2 n$

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

$\in \Theta(n^2 \log_2 n)$

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



let No: of levels will be n

No: of leaf nodes = 2^{n-1}

$L_c = 2^{n-1}$

I_c (≠) (n-1) internal levels.

$I_c = 2^0 + 2^1 + 2^2 + \cdots + 2^{n-2}$

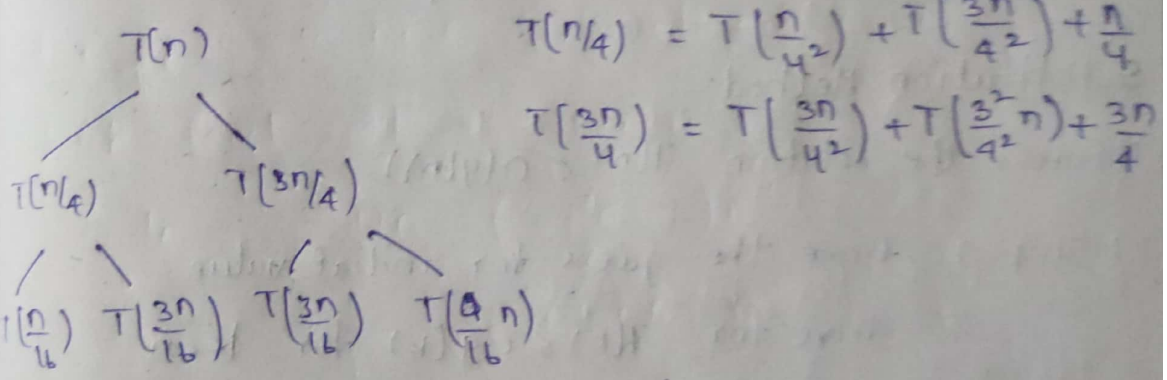
$= \frac{2^{n-1} - 1}{1}$

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

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

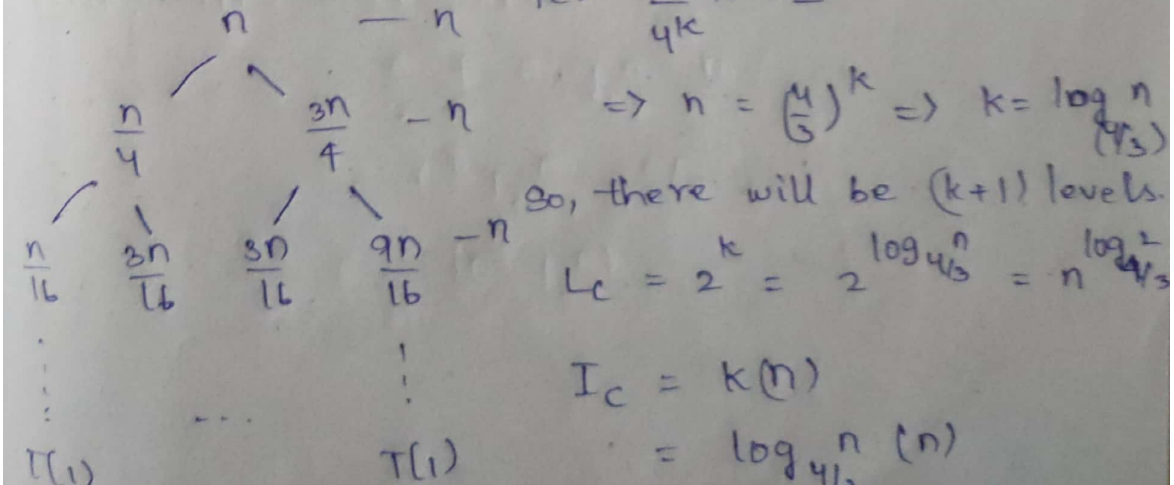
$\in O(2^{n+0})$

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



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

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



let $\frac{3^k}{4^k} n = 1$

$\Rightarrow n = \left(\frac{4}{3}\right)^k \Rightarrow k = \log_{(4/3)} n$

So, there will be (k+1) levels.

$L_c = 2^k = 2^{\log_{4/3} n} = n^{\log_{4/3} 2}$

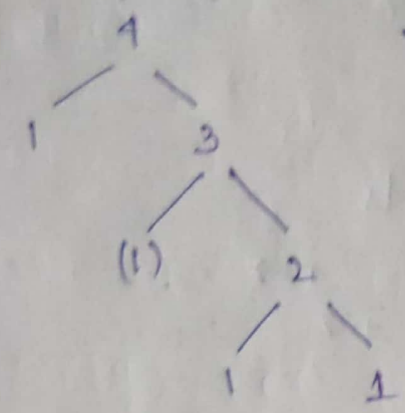
$I_c = k(n)$

$= \log_{4/3} n \; (n)$

$\therefore T(n) = n^{\log_{4/3} 2} + n \log_{4/3} n$

$\in O(n \log_{(4/3)} n)$

$n=4 \quad T(4) = T(1) + T(3) + 4$



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

13

$4 \log_{4/3} 4$

$\frac{n}{4^{k_1}} = 1 \qquad k_1 = \log_4 n$

$k = \log_{4/3} n$

$\frac{3^k}{4^k} n$

$\frac{3^{\log_4 n}}{4^{\log_4 n}} n = \frac{n^{\log_4 3}}{n} \times n$

Substitution Method:-

1) Guess the solution $T(n) \in O(g(n))$

2) Step-1:- Prove the guess for initial value.

2:- Prove for $T(k) \leq c\,g(k)$ for $k<n$

then $T(n) \leq c\,g(n)$, $n \in N$, $n > n_0$

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

Guess:- $T(n) \leq c.n\log_2 n$

Hypothesis :- $T(k) \leq c.k\log k \;\forall\; k<n \quad k = \frac{n}{2}$

Inductive step :- $T(n) \leq 2T(\frac{n}{2}) + n$

$\leq 2.c\frac{n}{2}\log(\frac{n}{2}) + n$

$= cn\log\frac{n}{2} + n$

$= cn\log n - cn\log 2 + n$

$= cn\log n - cn + n$

$\leq c\,n\log n \quad \forall c \geq 1$

Base step :- $T(1) \leq c.1\log_2 1 \Rightarrow T(1) = 0$

for $n=2$, $n=3$, we get $T(1)$ on the RHS

So, check for both of them.

$T(2) = 2(T(1)) + 2 \qquad T(3) = 2T(1) + 3$

$= 4 \qquad\qquad = 5$

Acc to our guess, $T(n) \leq c\,n\log_2 n$ for some $c>0$ & $n>n_0$

Find c, n_0 accordingly

$T(2) \leq C(2) \qquad T(3) \leq c(3)\log_2 3$

$\Rightarrow 2c \geq 4 \qquad \Rightarrow c \geq 1.052$

$\Rightarrow c \geq 2$

$\Rightarrow c \geq 2, n_0 = 2$

$T(n) \in O(n\log_2 n) \quad c \geq 2, n_0 = 2$

$n > n_0$

Wrong guess:-

Guess:- $T(n) \in O(n)$ i.e., $T(n) \leq cn$ for some $c > 0$ and $n \geq$

Hypothesis:- let $T(k) \leq kc$ for $\forall k < n$

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

Now, $T(\frac{n}{2}) = \frac{cn}{2}$ [$\because \frac{n}{2} < n$, $T(k) \leq kc$ $\forall k < n$]

$\therefore T(n) \leq 2\frac{cn}{2} + n$

$= (\frac{c+1}{1})n$

Now, $T(n) \leq cn \rightarrow$ This should hold for some $c > 0$

But we got $T(n) \leq (\frac{c+2}{2})n$

So, $\frac{c+2}{2} n \leq cn$

$\Rightarrow c+2 \leq 2c \Rightarrow c \geq 2$

For $c \geq 2$, $T(n) \leq cn$

Now, check the base case:-

$T(1) \leq c = 2$

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

Guess:- $T(n) \in O(n^2)$ $T(n) \leq cn^2$

Hypothesis:- $T(k) \leq ck^2$ for $k<n$

Inductive step:- $T(n) = T(n-1)+n$

$T(n-1) \leq c(n-1)^2$ [$\because n-1<n$]

so, $T(n) \leq c(n-1)^2+n$

$\leq c(n^2+1-2n)+n$

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

$\leq cn^2$ $n-2cn+c<0$

for $c>1$ if $c \geq 1$ $n-2cn+c$

Base step:- $T(1) = 1$

From $T(n) \leq cn^2$

$T(1) \leq c$

$\Rightarrow c \geq 1$

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

Guess:- $T(n) \in O(n^2 \log_2 n) \leq cn^2 \log_2 n$

Hyp:- $T(k) \leq ck^2 \log_2 k$ $\forall k<n$

Ind:- $T(n) = 4T(n/2)+n^2$

$T(n/2) \leq c\frac{n^2}{4}\log_2(n/2)$ $\because n/2<n$

$\therefore T(n) \leq 4c\frac{n^2}{4}\log_2(n/2)+n^2$

$\leq cn^2\log_2 n - cn^2+n^2$

$\leq cn^2\log_2 n + (1-c)n^2$

$\leq cn^2\log_2 n$ for $c \geq 1$

Base case :-

let $n_0 = 1$ $T(1) = 1$ (from que)

$T(1) \leq c(1)\log_2 1$

≤ 0

But $1 \leq 0$, So, $T(1)$ fails

Hence $n_0 \neq 1$

For $n_0 = 2$, $T(2) = 4T(1) + 4$

$= 4 + 4 = 8$

$T(2) \leq c \times 4 \log_2 2$

$\leq 4c$

So, $8 \leq 4c \Rightarrow c \geq 2$ for $n_0 = 1$

So, $c \geq 2$, $n_0 = 1$, $T(n) \leq cn^2 \log_2 n$

Wrong guess :-

Guess :- $T(n) \in O(n \log_2 n)$

Hyp :- $T(k) \leq ck \log_2 k$ some $c > 0$

Ind :- $T(n) = 4T(n/2) + n^2$

$T(n/2) \leq c\frac{n}{2} \log_2 \frac{n}{2}$

$T(n) \leq 2cn \log_2 n - 2cn + n^2$

$\leq 2cn \log_2 n + (n^2 - 2cn)$

$\not\leq cn \log_2 n$ for $c > 0$ & $n \geq 2$

$n^2 - 2cn > 0$

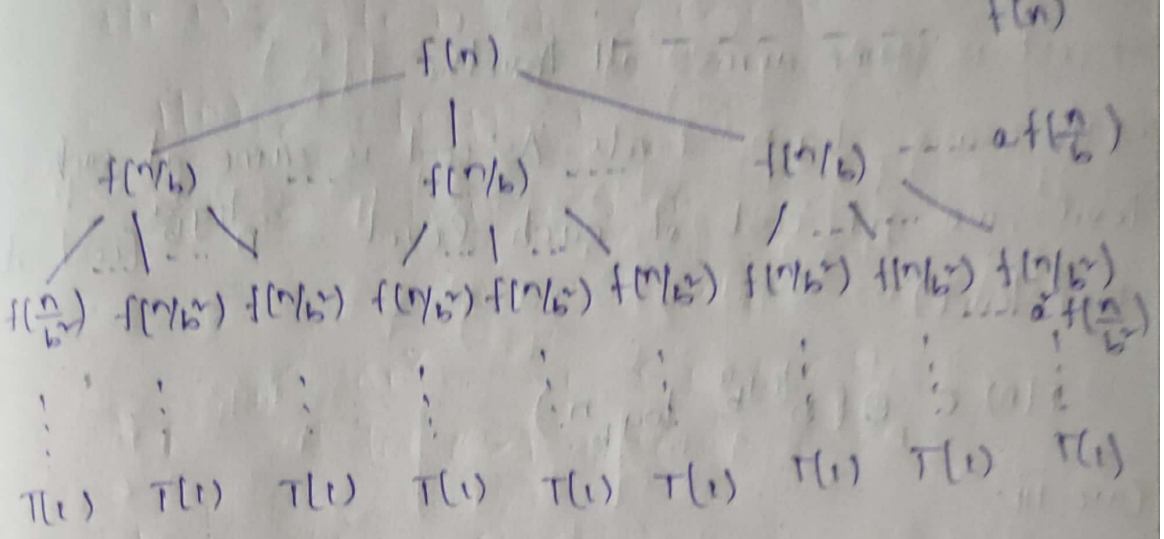
Master Method:

→ used for solving recurrences of the form:

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

$f(n)$ is a positive function.

$$T(n) = \begin{cases} 1 & , n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & , n > 1,\ a \geq 1,\ b > 1 \end{cases}$$



$$\frac{n}{b^k} = 1 \Rightarrow k = \log_b n$$

No: of levels $= (k+1)$

No: of leaf nodes $= a^k = a^{\log_b n} = n^{\log_b a}$

$$\therefore L_c = n^{\log_b a}$$

$$I_c = f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + \cdots + a^{k-1} f\left(\frac{n}{b^{k-1}}\right)$$

$$= \sum_{k=0}^{\log_b n - 1} a^k f\left(\frac{n}{b^k}\right)$$

$$\therefore \text{Cost} = L_c + I_c = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} a^k f\left(\frac{n}{b^k}\right)$$

Three cases:-

1) cost increases geometrically from root to leaves.
2) " decreases
3) same at each level.

Case-i:-

→ Cost increases geometrically from root to leaves.

→ $n^{\log_b a}$ is asymptotically greater than $f(n)$ by a factor n^{ϵ}

Then $T(n) \in \theta(n^{\log_b a})$

Case-ii:-

→ Cost is same across all levels.

→ There are $\log_b n$ levels. As cost is same at each level, we get cost at each level = $n^{\log_b a}$ (cost at leaf level)

$T(n) \in \theta(n^{\log_b a} \log_b n)$

Case-iii:-

→ Cost decreases geometrically.

→ Then $n^{\log_b a}$ is asymptotically smaller than $f(n)$ by a factor n^{ϵ}

$T(n) \in \theta(f(n))$

Theorem:-

Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a function and $T(n) = aT(\frac{n}{b}) + f(n)$

Then $T(n)$ can be bounded as

(i) If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \theta(n^{\log_b a})$

(ii) If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} \log_b n)$

(iii) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and if $a \cdot f(\frac{n}{b}) \leq cf(n)$ for some $c < 1$ and $n > n_0$ then

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

Proof:-

Case-1:- $n^{\log_b a} > f(n)$

$$\Rightarrow \left(\frac{n^{\log_b a}}{f(n)}\right) \in \Omega(n^{\epsilon}) \rightarrow (1)$$

$$\Rightarrow \frac{n^{\log_b a}}{f(n)} \geq n^{\epsilon} \Rightarrow f(n) \leq \frac{1}{c}(n^{\log_b a - \epsilon})$$

$$\Rightarrow f(n) \in O(n^{\log_b a - \epsilon})$$

Suppose, $T(n) = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} a^k f\left(\frac{n}{b^k}\right)$

$f(n) = n^{\delta}$ where $\delta < \log_b a$ (or) $b^{\delta} < a$ then

$$T(n) = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^{\delta}}\right)^k n^{\delta}$$

$$= n^{\log_b a} + n^{\delta}\left[\frac{\left(\frac{a}{b^{\delta}}\right)^{\log_b n} - 1}{\left(\frac{a}{b^{\delta}} - 1\right)}\right]$$

$$= n^{\log_b a} + n^{\delta}\left(\frac{n^{\log_b a - \delta} - 1}{c}\right)$$

$$= n^{\log_b a} + \frac{n^{\log_b a} - n^{\delta}}{c}$$

$$= \Theta(n^{\log_b a}) \quad (\because \delta < \log_b a)$$

Case-ii:- $n^{\log_b a} = f(n)$

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

$$T(n) = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^{\delta}}\right)^k n^{\delta}$$

Suppose $f(n) = n^{\delta}$ $\delta = \log_b a$ (or) $b^{\delta} = a$

$$T(n) = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^{\delta}}\right)^k n^{\delta} = n^{\log_b a} + n^{\delta} \log_b n$$

$$= n^{\log_b a}(1 + \log n) \in \Theta(n^{\log_b a} \log_b n)$$

Case-iii :- $n^{\log_b a} < f(n)$

$$\frac{f(n)}{n^{\log_b a}} \in \Omega(n^{\epsilon}) \Rightarrow \frac{f(n)}{n^{\log_b a}} \geq cn^{\epsilon}$$

$$\Rightarrow f(n) \geq cn^{\log_b a + \epsilon}$$

$$\Rightarrow f(n) = \Omega(n^{\log_b a + \epsilon})$$

$$T(n) = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} a^k f\left(\frac{n}{b^k}\right)$$

Suppose $f(n) = n^{\delta}$ where $\delta > \log_b a$ (or) $b^{\delta} > a$

$$T(n) = n^{\log_b a} + \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^{\delta}}\right)^k n^{\delta}$$

$$= n^{\log_b a} + n^{\delta}\left[\frac{1}{\frac{a}{b^{\delta}} - 1}\right] = n^{\log_b a} + \frac{n^{\delta}}{c}$$

$$= \Theta(n^{\delta}) \quad [\because \delta > \log_b a]$$

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

→ This method doesn't cover all possibilities.

1.) There is a gap between cases 1 and 2 when $f(n)$ is smaller than $n^{\log_b a}$ but not polynomially ~~larger~~ smaller

2.) Also, a gap blw 2 & 3 when $f(n)$ is larger than $n^{\log_b a}$ but not polynomially larger.

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

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

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

Now, $f(n) = n^{\delta} = n$ where $\delta = \log_b a = 1$

falls under case -ii

$\therefore\ T(n) \in \Theta(n \log_2 n)$

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

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

$L_c = n^{\log_2 2} = n$

Now, $f(n) = n^{\delta} = n^2$ $\quad \delta > \log_b a \Rightarrow \delta > 1$

Case-iii applies

Checking regularity condition. $T(n)$

$af\left(\frac{n}{b}\right) \leq cf(n) \quad c < 1$ & $n > n_0$

$2f\left(\frac{n}{2}\right) = 2 \times \frac{n^2}{4} = \frac{n^2}{2} \leq cn^2$

$\Rightarrow c \geq 1/2$

So, regularity condition holds.

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

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

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

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

$f(n) = n^{\delta} = n \quad \delta < \log_b a \Rightarrow \delta < 2$

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

$= \Theta(n^2)$

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

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

$L_c = n^{\log_b a} = n^0 = 1$

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

$T(n) \in \Theta(n^{\log_2 1} \log_2 n)$

$= \Theta(\log_2 n)$

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

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

$L_c = n^{\log_2 4} = n^2$

$f(n) = n^\delta = n^3 \quad \delta > \log_b a$

Case iii :-

$f(n) \in \Omega(n^{\log_b a} + 1)$

Regularity check

$af(\frac{n}{b}) \leq cf(n) \quad c<1,\ n>n_0$

$\Rightarrow 4f(\frac{n}{2}) \leq cn^3$

$\Rightarrow 4 \times \frac{n^3}{8} \leq cn^3$

$\Rightarrow c \geq 1/2 \quad [c = 1/2 < 1]$

Holds.

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

$= \Theta(n^3)$

6) $T(n) = 7T(\frac{n}{3}) + n^2$

$a=7,\ b=3,\ f(n)=n^2$

$L_c = n^{\log_3 7} = n^{1.77}$

$f(n) = n^{\delta} = n^2 \qquad \delta > \log_b a$

Case-iii applied.

$f(n) \in \Omega(n^{\log_b a + 0.23})$

Regularity check.

$af(\frac{n}{b}) \leq cf(n)$

$\Rightarrow 7f(\frac{n}{3}) \leq c\,n^2$

$\Rightarrow 7 \times \frac{n^2}{9} \leq cn^2$

$\Rightarrow c \geq 7/9 \qquad c = 7/9\ [c<1]$

Holds

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

7) $f(n) = 2T(\frac{n}{2}) + \sqrt{n}$

$L_c = n \qquad f(n) = n^{1/2}$

$\delta < \log_b a$

$T(n) \in \Theta(n)$

8) $T(n) = 3T(\frac{n}{4}) + n\log_2 n$

$L_c = n^{\log_4 3} = n^{0.79} \qquad f(n) = n\log_2 n$

$f(n) = n^{\delta} \qquad \delta > 0.79$ Case-iii applied.

Regularity check :- $3f(\frac{n}{4}) \leq cf(n)$

$\Rightarrow 3\frac{n}{4}\log_2 n - \frac{3n}{4} \times 2 \leq cn\log_2 n$

$\Rightarrow [\log_2 n - 2] \leq \frac{4}{3} c \log_2 n$

Loop Invariance:-

→ Loop is a property that remains true before and after each iteration of the loop.

It has three main properties:-

1) Initialization:-

→ It is true prior to initialization of the first iteration.

2) Maintainance :- If it is true before an iteration of the loop it remains true before the next iteration of the loop.

3) Termination:- When loop finishes, the loop invariant provides a useful property that helps to demonstrate the algorithm is crct.

Strassen Multiplication of Matrices:-

→ Brute force – $O(n^3)$

Divide And conquer:- $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ $B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$

M(n): $C = AB$

if $n \leq 2$:

return brute_force (n); → constant time

else:

$C_{11} = M(n/2) + M(n/2)$ $[ae + bg]$

$C_{12} = M(n/2) + M(n/2)$ $[af + bh]$

$C_{21} = M(n/2) + M(n/2)$ $[ce + dg]$

$C_{22} = M(n/2) + M(n/2)$ $[cf + dh]$

return C

end M

$T(n) = 8T(n/2) + \theta(n^2)$ $[\theta(n^2) \rightarrow$ for adding Matrix$]$

$\Rightarrow T(n) \in \theta(n^3)$

→ Same as brute force.

→ Try to reduce the No: of multiplications:-

$ae + bg = (a+d)(e+h) + d(g-e) - (a+b)h + (b-d)(g+h)$

$af + bh = a(f-h) + (a+b)h$

$ce + dg = ce \cdot (c+d)e + d(g-e)$

$cf + dh = a(f-h) + (a+d)(e+h) - (c+d)e - (a-c)(e+f$

$P_1 = (a+d)(e+h)$ $P_5 = a(f-h)$

$P_2 = d(g-e)$ $P_6 = (c+d)e$

$P_3 = (a+b)h$ $P_7 = (a-c)(e+f)$

$P_4 = (b-d)(g+h)$

$cf + dh = P_5 + P_1 - P_6 - P_7$

$af + bg = P_1 + P_2 - P_3 + P_4$, $af + bh = P_5 + P_3$, $ce + dg = P_6 + P_2$

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

$T(n) \in \Theta(n^{2.81})$

Prims Algorithm:-

→ used to find MST.

→ Given a graph with V vertices and E edges with weights upon each edge.

Let V[] vertices, vis[] visited

W[] weights. W[i] = {V[j], w_{ij}), ... }

Key[] key values for vertices.

Neighbour[] neighbour vertices of vertices.

Clauses:-

1.) Key[i] array holds the minimum weight/cost to be paid to reach the present vertex V[i] from the corresponding neighbour vertex in N[i].

2.) N[i] array holds the minimum cost neighbour for a vertex V[i]

Steps:-

→ V, W holds the vertices and weights.

→ Initialize Key array to theoritical infinity.

→ Initialize Neighbour array to Null, visited array to false.

→ Choose any random vertex as source vertex.
source = V[i]

→ Add this source to our tree i.e; Mark it as visited. and put key[source] = 0

→ Construct a min_heap of all vertices based on their key values.

→ Loop while the min_heap is not empty

→ Every time, 1) extract the min of the heap
2) Mark the vertex as visited.

3) for every vertex adjacent to the present vertex do the following

(i) If it is unmarked and key[i] > weight associated with that edge, then key[i] = wbij

and N[i] = V[j]

(ii) Reconstruct MinHeap

Time complexity :- O(ElogV)

$$f(n) = \sum_{k=1}^{n-1} f(k) f(n-k)$$

$$\Omega(2^n)$$

Pri Kruskal's Algorithm:-

V[] vertices

E[u,v,w] u,v are vertices of the edge
w is weight of that edge.

1.) Make every vertex as a separate set.

2.) Sort edges based on increasing order of weights.

3.) For each edge, check if vertices u, v are present in same set.

4.) If not, include the current edge in our solution and perform union on u, v

Algo:-

```
for i in V:
    make_set(i)
sort(E[u,v,w])
for j in E[u,v,w]:
    if find_set[j[u]] != find_set[j[v]]:
        solution += [u,v]
        union(u,v) ⟶ O(lgV)
end & return solution.
```

Time Complexity :- $O(E \lg V)$

Dijkstra's Algo:-

$S \leftarrow \phi$

Put key value for source vertex as 0.

$Q \leftarrow V[G]$

while $Q \neq \phi$

do $u \leftarrow$ extract_min(Q)

$S \leftarrow S \cup \{u\}$

for each vertex $v \in Adj[u]$:

do Relax(u, v, w)

```
Relax(u,v,w){
    if (d[v] > d[u] + w(u,v))
        d[v] = d[u] + w[u,v]
}
```

04-08-2022

Chained Matrix Multiplication:-

Paranthesize the matrices so that # of scalar multiplications are min.

Ex:- M_1 18×5

M_2 5×89

M_3 89×3

M_4 3×12

Greedy:- First compress the multiplication with higher number of columns.

compute M_2M_3 5×89×3 Multiplication

→ Next choose the highest col and compress that product.

→ This is greedy approach.

→ But this also doesn't help.

Dynamic Programming:-

→ Construct a table with rows and columns representing the matrices to be multiplied.

	M_1	M_2	M_3	M_4
M_1	x	5785	1530	2295
M_2	x	x	1335	1845
M_3	x	x	x	9078
M_4	x	x	x	x

x → inappropriate Mult.

→ Each cell represents the no: of scalar multiplications req.

```
EMM ( A[0 ... n] , n):
  for i = 1 to n :
      T[i,i] = 0
  for d = 1 to n-1 :
      for i = d to (n-1):
          T[i, i+d

for (d = 1 ; d <= (n-1) ; d++) {

    for (i = 1 ; i <= (n-d) ; i = i+1) {
        T[i, i+d] = minSum = +∞
                    j = i+d
        for (k = i ; k < j ; k++) {
            if( minSum >
            pSum = T[i, k] + T[k+1, j]
                   + (A[i-1] * A[k] * A[j])
            if ( minSum > pSum ) {
                minSum = pSum
            }
        }
        T[i, i+d] = minSum
    }
}
```

```
CMM (A[0... n], n):
    for (i = 1 ; i <= n ; i++) {
        T[i, i] = 0

    sol [n][n] ← 0

    for (d = 2; d <= (n-1); d++) {          n-1
        for (i = 1 ; i <= (n-d) ; i++) {    n-d
            minSum = +∞
            j = i+d
            for (k = i ; k < j ; k++) {     i+d-i
                                            d
                pSum = T[i,k] + T[k+1, j] +
                       A[i-1] * A[k] * A[j]
                if (pSum < minSum) {
                    minSum = pSum
                    sol[i][i+d] ← k
                }
            }
            T[i][i+d] = minSum
        }
    }

    for (i = 1 ; i <= (n-1) ; i++) {
        T[i, i+1] = A[i-1] x A[i] x A[i+1]
    }

end CMM
```

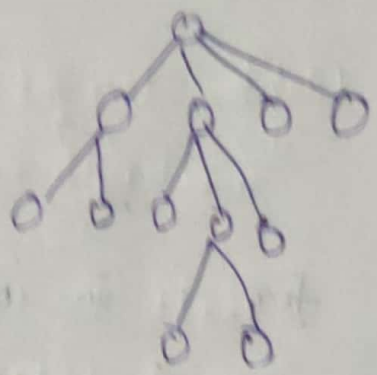
Time:- $O(n^3)$

```
findPartition(i,j):
    if sol[i][i] == 0 :
        return
    printf(sol[i][j] + ',')
    findPartition(i, sol[i][j])
    findPartition(sol[i][j]+1, j)
end findPartition.
```

Time :- O(n)

Q:
Given layout of the building
→ Install cameras at intersections
→ Minimal number of cameras required.

Sol:
→ ~~Arrange~~ the corners in a hierarchical manner.



Number of cameras

$$N[i] = \min\{1 + \sum_{c \in child(i)} N(i), \quad children(i) * \sum_{c \in child(n)} \sum_{g \in grand\ child(n)} N(g)\}$$

1) Matching strings (brute force & Normal code)

2) Travelling saint problem

3) Problem of satisfy (Boolean Logic)

Match_Strings (S_1, S_2):

```
prStr = min(|S1|, |S2|)
subStr = []
for (i = |prStr|; i >= 1; i--) {
    for (j = 0; j+i <= |prStr|; j = j+i) {
        subStr.append(prStr[j : j+i])
    }
}

for string in subStr {
    if (contains(string, S2)) {
        return string
    }
}
return null
end
```

$O(n^2)$

n^2

m

$O(mn^2)$

Time: $O(mn^2)$

```
Contains (S, S1):
    j = 0
    for (i = 0; i < |S1|; i++) {
        if (S1[i] == S[j]) {
            j++;
            if (j == |S|+1) {
                return true
            }
        }
        else {
            j = 0
            if (S1[i] == S[j]) {
                j++
                if (j == |S|+1) {
                    return true
```

$O(|S_1|)$

```
        }
    }
}
return false
}
```

1, 2, 3, 4

e[2,4]

2	2
3	1
4	4

```
Match_Strings(S1, S2):
    maxlength = 0

    for(i=1; i<=|S1|; i++){
        for(j=1; j<=|S2|; j++){
            if[i][j]==0
            if( i-1>0 && j-1>0){
                T[i][j] = T[i-1][j-1]
                          +(S1[i-1] == S2[j-1])
            else {
                T[i][j] = (S1[i-1] == S2[j-1])
            }
            if ( maxlength < T[i][j] ){
                maxlength = T[i][j]
            }
        }
    }
    return maxlength
end.
```

~~Max Flows:-~~

Backtracking:-

1) N-Queen's

2) Path finding

N Queen's:-

```
nQueen's (c[1....n], int n, int r){
    if (r==(n+1)):
        return c[]
    else:
        for (j=1 to n){
            c[r] = j
            if (isfeasible(c[], j, r)){
                nQueens(c[], n, r+1)
            }
        }
}
```

isfeasible:-

```
isfeasible (c[1...n], j, r){
    for (k=1 to r-1){
        if (c[r]==c[k] || (c[r]-c[k])==(r-k))
            return false
    }
    return true
}
```

Graph Coloring:-

$G_1(V, E)$

Objective:-

Assign one colour to each vertex of the graph such that no two vertices have which are adjacent have same colour.

k → No: of colors.

Coloring (G(V,E)): i → vertex to be coloured

```
coloring (G(V,E), C[1...n] ,i,k) {
    let |V| = n
    if ( i == n+1) {
        return C
    else {
        for (i=1 to k) {
            c[i] = i
            if (isValid(G, C, i)) {
                coloring (G, C[], i+1, k)
            }
        }
}
```

```
isValid( G[V, E], C[1,...n], j ){
    let |V| = n
    for ( i=1 to j-1 ) {
        if ((j, i) ∈ E || (i, j) ∈ E) {
            if (u<j || C[u] == C[j]){
                return false
            }
        }
    }
    return true
}
```

Ex:-

$V = \{1, 2, 3, 4\}$

$E = \{(1,2), (1,3), (2,3), (2,4), (3,4)\}$

Solutions

Vertex	A	B	C	D	E	F
1	1	1	2	2	3	3
2	2	3	1	3	1	2
3	3	2	3	1	2	1
4	1	1	2	2	3	3

Microprocessor GPR liveliness Analysis.

Daemons & Resources

→ No: of ext cnctions for 128 x 64 org Memory chip

7 → address lines

64 - bits output

2 - R/W

2 → Ext cnction Power

75

512 K x 8

2^{19}

19 → add

8 → bits

2 → R/W

2 → Power

27
1
31

4 - color Theorem:-

→ Any planar graph can be covered with 4 colors.

Enumerating Independent sets of graph:-

→ This is MIS ← Min Ind set

→ A vertex set is ind set if there is not edge involving the vertices in set.

Ex:- $V = \{1,2,3,4,5,6\}$

$E = \{(1,2), (1,4), (1,5), (2,3), (2,6), (3,6), (4,6), (5,6)\}$

$$IS(n) = \sum_{S \in \{1,2,\ldots,n\}} P(S \text{ is a IS})$$

$$P(S \text{ is a IS}) = \binom{1}{2}^{|S|C_2}$$

$$= \frac{1}{2}^{[(|S|-1)(|S|)/2]}$$

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

$$\therefore IS(n) = \sum_{k=0}^{n} {}^{n}C_k \, \frac{1}{2}^{\frac{(k-1)k}{2}}$$

Branch & Bound 01-04-2022

→ The diadv is worst-case time complexity is intractable.

→ The best case complexity is minimized.

→ find the most promising node to the soln and visit in that order.

→ Prune out remaining possibilities.

TSP:-

→ Goal is to minimize :-

$$C = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$x_{ij} = \begin{cases} 1 & , i \to j \text{ edge included} \\ 0 & , \text{otherwise} \end{cases}$$

Constraints:-

$x_{ij} \in \{0, 1\}$

$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j$ → A city should be reached from only one city

$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i$ → Move from a city to another city only once.

→ Check for subtours

Consider tours 1-3-2-1
4-5-4

→ These accept above constraints but are not optimal.

for length of subtours:-

$l = 1$:- $x_{ii} = 0 \quad \forall i$

$l = 2$:- $x_{ij} + x_{ji} \leq 1 \quad \forall j, i$

$l = 3$:- $x_{ij} + x_{jk} + x_{ki} \leq 2 \quad \forall i, j, k$

⋮

$l = n-1$:-

→ But these constraints are combanitorially large

→ To reduce them,
if subtours of length 1 are eliminated, then trivially subtours of $n-1$ are also eliminated.

→ using this, we reduce the const^{n}s from n to $\lfloor n/2 \rfloor$

→ It is also large.

n^r constraints:-

$u_i - u_j + n x_{ij} \leq (n-1)$

$\forall i \in \{1, 2, \ldots, n\}$

$\forall j \in \{2, \ldots, n\}$

Partial tours

$1-2-3-1$

$4-5-4$

for $1-2-3-1$

$u_1 - u_2 + 5 \leq 4$

$u_2 - u_3 + 5 \leq 4$

$u_1 - u_3 + 10 \leq 8$

$u_4 - u_5 + 5 \leq 4$

$u_5 - u_4 + 5 \leq 4$

Not possible.

Total tours

$1-2-3-4-5-1$

for this tour

$u_1 - u_2 + 5 \leq 4$

$u_2 - u_3 + 5 \leq 4$

$u_3 - u_4 + 5 \leq 4$

$u_4 - u_5 + 5 \leq 4$

$u_5 - u_1 + 5 \leq 4$

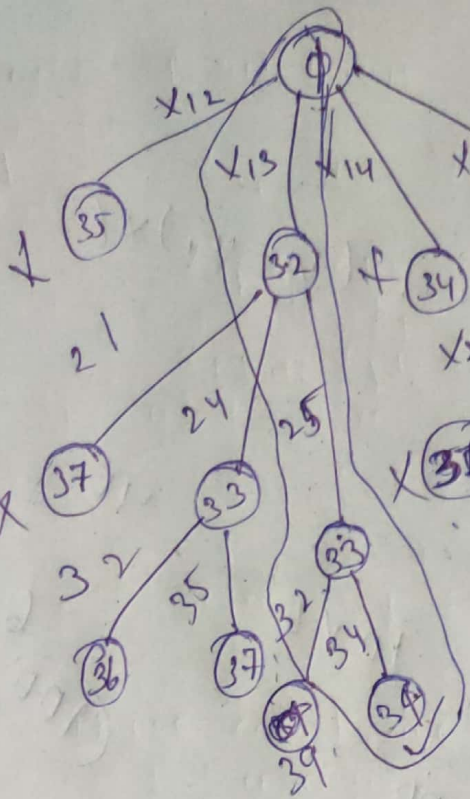
$u_1 - u_5 + 20 \leq 16$

possible

→ This formulae prevents subtours and promotes full tours.

Ex: TSP

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



1 - 3 - 4 - 2 - 5

1 - 3 - 2 - 5 - 4

1 - 3 - 5 - 2 - 4

2 4 5

3 10 8 9

4 5 8 6

5 6 9 -

1 2 5

3 8 10 9

4 9 5 6

5 7 6 -

8 + 5 + 6

19 8 + 9

8 + 6

27 + 5

37 + 9

8 + 10 + 5 + 6 + 7

36

8 + 10 + 6 + 6 + 9

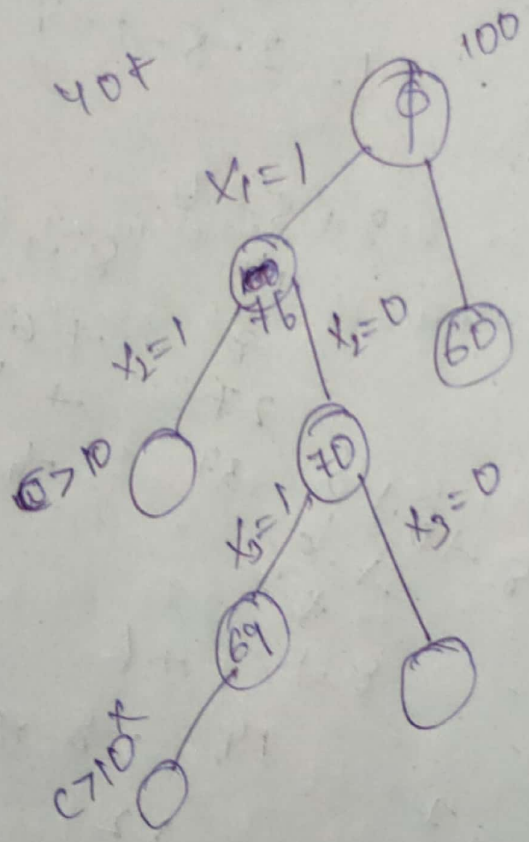
$v_i + (w - w_i) \times 6$

40 (6)

40+ 10×6 6

100

40+



0/1 Knapsack :-

item :-	weight	value	V/w
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

Capacity = 10 10 $10 \times 10 = 100$

Heirustic :-

$UB = V_i + (W - w_i) \times \max(R)$

Set cover :-

$X = \{1, 2, 3, 4, 5, 6\}$

$S = \{S_1, S_2, S_3, S_4\}$ $\forall s \in S \subseteq X$

Greedy logic :- ← Approximation prgrm

$Y = \phi$

while $\bigcup_y y \in Y \neq X$ {

$s \leftarrow$ pick a set from S that covers the largest number of remaining uncovered elements in $X - \bigcup_y y \in Y$

$Y \leftarrow Y \cup \{i\}$

return Y

}

String Matching 05-04-2022

→ Worst case for brute force - O(m,n)

```
naiveMatch (char * text, char * p):
    int i=0 ; j=0;  k=0;

/*while (i < (|text| - |p|)){
                j<|p|
        if (text[i] == p[j]) {
            i++;  k = i;
            j++;
        }
        else {
            i= k+1;
            j=0;
        } */

while ( i<(|text| - |p|) && j < |p| ) {
    //int i=0; j=0; k=0;
        if (text[i] == p[i]) {
            i++;
            j++;
            k=i;
        else{
            i=k+1;
            j=0;
        }
    }
    if (j == |p|){
        return k
    return -1
```

T(naiveMatch)
= O(mn)

Knuth - Morris Pratt Algo (KMP)

→ for the pattern, create an array (prefix array) which notes the suffices that are also prefixes.

→ Then start matching with text and slide the pattern in case of mismatch using the prefix array.

Ex:-

```
      j=1
  j=0 i=1 i=2
   x  x  y  x  x  y  x  x  x
   0  1  0  1  2  3  4  5  2
```

This is the prefix array.

```
compute PrefixArr(char * P):
    int i=0, j=1;
    int prefixArr[|P|];
    while (i < len(P)){
        if (P[i] == P[j]){
            prefixArr[i] = j+1
            i++; j++;
        }
        else {if (j!=0){
            j = prefixArr[j-1];
            }
            else {
                prefixArr[i] = 0;
            } i++;
    }
}
```

T(

return prefixArr

KMP_Match (char * text, char * patt):

```
int i=0, j=0, k=0;
while (j<|P| && i<(|text|-|P|)){
    if (text[i]==patt[j]){
        i++;
        j++;
        k = i
    }
    else {if (j!=0){
        j = prefixArr[j-1];
        }(k≠i)
        else { i = 0;}
            i++;
    }
}
if (j==|P|){
    return k // return i - |P|
}
return -1
```

end KMP_Match

T(KMP_Match) = O(mn) ← worst case

= O(m+n) ← best case.

T(compute_prefix) = O(m)