B.Ramesh

Page No. :

Date : / /

Master Theorem

Recurrence : $T(n) = aT(n/b) + \theta(n^k \log^p n)$

p : Real number

$a \geq 1$, $b > 1$, $k \geq 0$

case 1) If $a > b^k$ then $T(n) = \theta(n^{\log_b a})$

case 2) If $a = b^k$

a) if $p > -1$ then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

b) if $p = -1$ then $T(n) = \theta(n^{\log_b a} \log \log n)$

c) if $p < -1$ then $T(n) = \theta(n^{\log_b a})$

case 3) If $a < b^k$

a) if $p \geq 0$ then $T(n) = \theta(n^k \log^p n)$

b) if $p < 0$ then $T(n) = \theta(n^k)$

Case 1

1) $T(n) = 4T(n/2) + n$

$a = 4, \quad b = 2, \quad k = 1, \quad p = 0$

$T(n) = \theta(n^{\log_2 4})$

$= \theta(n^2)$

2) $T(n) = 16\,T(n/4) + n$

$a = 16, \quad b = 4, \quad k = 1, \quad p = 0$

$T(n) = \theta(n^{\log_4 16})$

$= \theta(n^2)$

3) $T(n) = \sqrt{2}\,T(n/2) + \log n$

$a = \sqrt{2}, \quad b = 2, \quad k = 0, \quad p = 1$

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

$= \theta(n^{0.50})$

$= \theta(\sqrt{n})$

4) $T(n) = 3T(n/3) + \sqrt{n}$

$a = 3, \quad b = 3, \quad k = 1/2, \quad p = 0$

$T(n) = \theta(n^{\log_3 3})$

$= \theta(n)$

5) $T(n) = 3T(n/2) + n$

$a = 3, \quad b = 2, \quad k = 1, \quad p = 0$

$T(n) = \theta(n^{\log_2 3})$

$= \theta(n^{\lg 3})$

$= \theta(n^{1.58})$

6) $T(n) = 4T(n/2) + cn$

$a = 4, \quad b = 2, \quad k = 1, \quad p = 0$

$T(n) = \theta(n^{\log_2^4})$

$= \theta(n^2)$

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

$a = 4, \quad b = 2, \quad k = 1, \quad p = -1$

$T(n) = \theta(n^{\log_2 4})$

$= \theta(n^2)$

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

$a = 4, \quad b = 2, \quad k = 0, \quad p = 1$

$T(n) = \theta(n^{\log_2 4})$

$= \theta(n^2)$

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

$a = 9, \quad b = 3, \quad k = 1, \quad p = 0$

$T(n) = \theta(n^{\log_3 9})$

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

Case 2

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

$a = 4, \; b = 2, \; k = 2, \; p = 0$

$T(n) = \theta(n^{\log_2 4} \log^{0+1} n)$
$= \theta(n^2 \log n)$

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

$a = 2, \; b = 2, \; k = 1, \; p = 1$

$T(n) = \theta(n^{\log_2 2} \log^{1+1} n)$
$= \theta(n \log^2 n)$

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

$a = 3, \; b = 3, \; k = 1, \; p = 0$

$T(n) = \theta(n^{\log_3 3} \log^{0+1} n)$
$= \theta(n \log n)$

Case 3

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

$a = 3, \quad b = 2, \quad k = 2, \quad p = 0$

$T(n) = \theta(n^2 \log^0 n)$

$= \theta(n^2)$

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

$a = 1, \quad b = 2 \quad n = 2, \quad k = n \text{ \& } p = 0$

Comparing a & b^k i.e 1 and 2^n

so if $n = 1, 2, \ldots$ then $a < pow(b,k)$ & $p = 0$ then, $T(n) = \theta(n^k \log^p n)$ by substituting $p=0$ we get $T(n) = \theta(n^k)$ then by substituting n & k value we have $n = 2 \quad k = n \quad \therefore T(n) = \theta(2^n)$

3) $2T(n/2) + n^2 \log n$

$a = 2, \quad b = 2, \quad k = 2, \quad p = 1$

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

4 $T(n) = 2T(n/4) + n^{0.51}$

$a = 2, \quad b = 4, \quad k = 0.51, \quad p = 0$

$T(n) = (n^{0.51} \log^0 n)$
$= \theta(n^{0.51})$

5 $T(n) = 16T(n/4) + n!$

$a = 16, \quad b = 4, \quad f(n) = n!$

$\log_b a = \log_4 16 = 2$

Third case of master theorem
$f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$

and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ & all sufficiently large n, then $T(n) = \theta(f(n))$ (regularity condition)

Hence $n! = \Omega(n^{2+\epsilon})$

This could be shown to satisfy for some value of $\epsilon > 0$ using stirling's approximation

$$n! \sim \sqrt{2\pi n}\left(n/e\right)^n$$

For regularity condition

$16f(n/2) <= cf(n)$

i.e. $16f(n/4)! <= cn!$

Take $c = 0.5$ and we find the regularity condition does satisfy

Hence according to case 3

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

6 $T(n) = 3T(n/4) + n\log n$

$a = 3,\ b = 4,\ k = 1,\ p = 1$

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

B.Ramesh
Page No. :
Date : / /

7 $T(n) = 6T(n/3) + n^2 \log n$

$a = 6, \quad b = 3, \quad k = 2, \quad p = 1$

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

8 $T(n) = 7T(n/3) + n^2$

$a = 7, \quad b = 3, \quad k = 2, \quad p = 0$

$T(n) = (n^2 \log^0 n)$

$= \theta(n^2)$

Substitution Method

I Given the solution

II Verify using properties of mathematical induction

Eg 1 $$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n \geq 1 \end{cases}$$

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

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

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

$T(2) = 1 + 2$

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

$T(3) = T(2) + 3$

$T(3) = \quad + 3$

$T(4) = T(4-1) + 4$

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

$T(4) = 1 + 2 + 3 + 4$

$T(4) = 4 + 3 + 2 + 1$

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

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

B.Ramesh
Page No. :
Date : / /

Or

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

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

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

$$T(n-2) = T(n-3) + (n-2) \quad \text{---} \quad (3)$$

Substituting equation (2) in 1

$$T(n) = T(n-2) + (n-1) \quad \text{---} \quad (4)$$

substituting equation (3) in 4

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

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

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

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

$T(1) = 1$

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

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

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

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

substituting equation (2) & (3) in (1)

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

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

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

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

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

Ex 3 $T(n) = T(n-1) + \log n$ if $n \geq 1$ --(1)

$T(1) = 1$ if $n = 1$

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

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

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

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

$= T(n-3) + \log(n-2)$ --(3)

substitute eq (2) & (3) in (1)

$T(n) = \log n + \log(n-2) + \log(n-3) + T(n-3)$

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

$\log(n-(n-2)) + \log(n-(n-3)) + \ldots$

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

$= 1 + \log 2 + \log 3 + \ldots + n$

$= 1 + \log(2 * 3 * 4 * \ldots * n)$

$T(n) = 1 + \log(n!) = 1 + \log n^n$

$T(n) = 1 + n \log n$

$T(n) = n \log n$

Ex 4 $T(n) = 2T(n/2) + n$ — (1) if $n > 1$

$T(1) = 1$ if $n = 1$

$$\left[T(n/2) = 2T(n/4) + n/2 \right]$$

$$T(n) = 2\left[2T(n/4) + n/2 \right] + n$$

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

$$T(n) = (2)^2 T(n/4) + 2n \quad \text{---} \quad (2)$$

$$\left[T(n/4) = 2T(n/8) + n/4 \right]$$

$$T(n) = (2)^2 \left[2T(n/8) + n/4 \right] + 2n + n$$

$$= (2)^3 T(n/8) + n + 2n$$

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

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

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

$k = \log n$

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

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

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

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

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

Ex 5 $T(n) = 2T(n/2) + 2, \quad T(1) = 0$

$$\left[T(n/2) = 2T(n/4) + 2 \right]$$

$$T(n) = 2\left(2T(n/4) + 2 \right) + 2$$

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

$$\left[T(n/4) = 2T(n/8) + 2 \right]$$

$$T(n) = (2)^2 \left(2T(n/8) + 2 \right) + 4 + 2$$

$$T(n) = (2)^3 T(n/8) + 8 + 4 + 2$$

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

$$= 2^k T(n/2^k) + \sum_{i=1}^{k} 2^i$$

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

$$\log_2 n = k$$

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

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

$$= n \cdot 0 + 2(2^{\log_2 n} - 1)$$

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

$$= 2n - 2$$

$$= 2n$$

$$T(n) = O(n)$$

Ex 6 $T(n) = 3T(n/3) + n/2$ ... (1)

$$\left[T(n/3) = 3T(n/9) + \frac{n}{6}\right]$$

$$T(n) = 3\left(3T(n/9) + \frac{n}{6}\right) + \frac{n}{2}$$

$$= 3^2 T(n/9) + \frac{n}{2} + \frac{n}{2}$$

$$\left[T(n/9) = 3T\left(\frac{n}{27}\right) + \frac{n}{18}\right]$$

$$T(n) = 3^2\left(3T(n/27) + \frac{n}{18}\right) + \frac{n}{2} + \frac{n}{2}$$

$$= 3^3 T(n/27) + \frac{n}{2} + \frac{n}{2} + \frac{n}{2}$$

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

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

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

$$k = \log_3 n$$

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

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

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

$$T(n) = n$$

Ex 7 $T(n) = 7T(n/3) + n^2$ --(1)

$$\left[T(n/3) = 7T(n/9) + (n/3)^2\right]$$

$$T(n) = 7\left(7T(n/9) + (n/3)^2\right) + n^2$$

$$= 7^2 T(n/9) + 7(n/3)^2 + n^2$$

$$\left[T(n/9) = 7T(n/27) + (n/9)^2\right]$$

$$T(n) = 7^2\left(7T(n/27) + (n/9)^2\right) + 7(n/3)^2 + n^2$$

$$= 7^3 T(n/27) + 7^2(n/9)^2 + 7(n/3)^2 + n^2$$

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

$$= 7^3 T(n/3^k) + 49\frac{n^2}{81} + 7\frac{n^2}{9} + n^2$$

$$= 7^3 T\left(\frac{n}{3^k}\right) + \frac{49n^2}{81} + \frac{63n^2}{81} + \frac{81n^2}{81}$$

$$= 7^3 T\left(\frac{n}{3^k}\right) + \frac{49n^2 + 63n^2 + 81n^2}{81}$$

$$= 7^3 T\left(\frac{n}{3^k}\right) + \frac{193n^2}{81}$$

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

$$= 343\, T(1) + \frac{193}{81} n^2$$

$$= \frac{193}{81} n^2 = n^2$$

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

Ex 8 $T(n) = T(n/2) + c$ if $n > 1$

$= b$ if $n \leq 1$

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

$[T(n/2) = T(n/4) + c]$

$T(n) = [T(n/4) + c] + c$

$= T(n/4) + 2c$

$[T(n/4) = T(n/8) + c]$

$T(n) = [T(n/8) + c] + 2c$

$= T(n/8) + 3c$

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

$T(n) = T(n/2^k) + k \cdot c$

$n/2^k = 1 \quad n = 2^k \quad k = \log_2 n$

$T(n) = T(1) + \log_2 n \cdot c$

$= b + c \cdot \log_2 n$

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

Ex 9 $T(n) = 3T(n/3) + n$

$[T(n/3) = 3T(n/9) + n]$

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

$= 3^2 T(n/9) + 2n$

$[T(n/9) = 3T(n/27) + n]$

$T(n) = 3^2(3T(n/27) + n) + 2n$

$= 3^3 T(n/27) + 3n$

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

$T(n) = 3^k T(n/3^k) + kn$

$n/3^k = 1 \qquad n = 3^k \qquad k = \log_3 n$

$T(n) = 3^k T(1) + 3(\log_3 n)$

$= n + (\log_3 n)3$

$T(n) = O(\log_3 n)$

call to itself several times

Recursion tree Method

B.Ramesh

Page No. :

Date : / /

step 1 : cost of each level

step 2 : Find depth of the tree

step 3 : Find the no of leaves

It has 3 cases

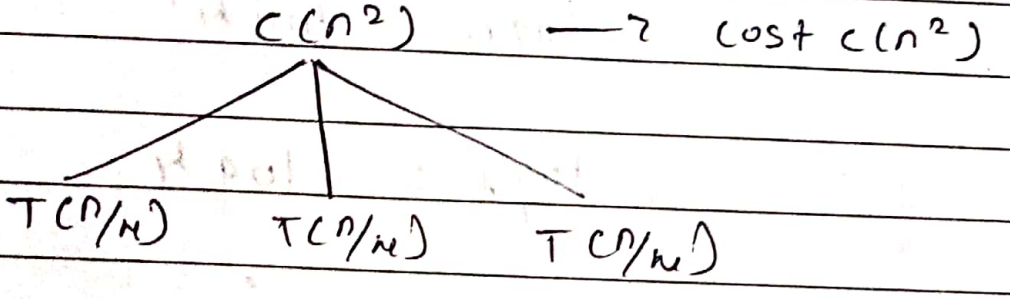
1) Cost of root node is max

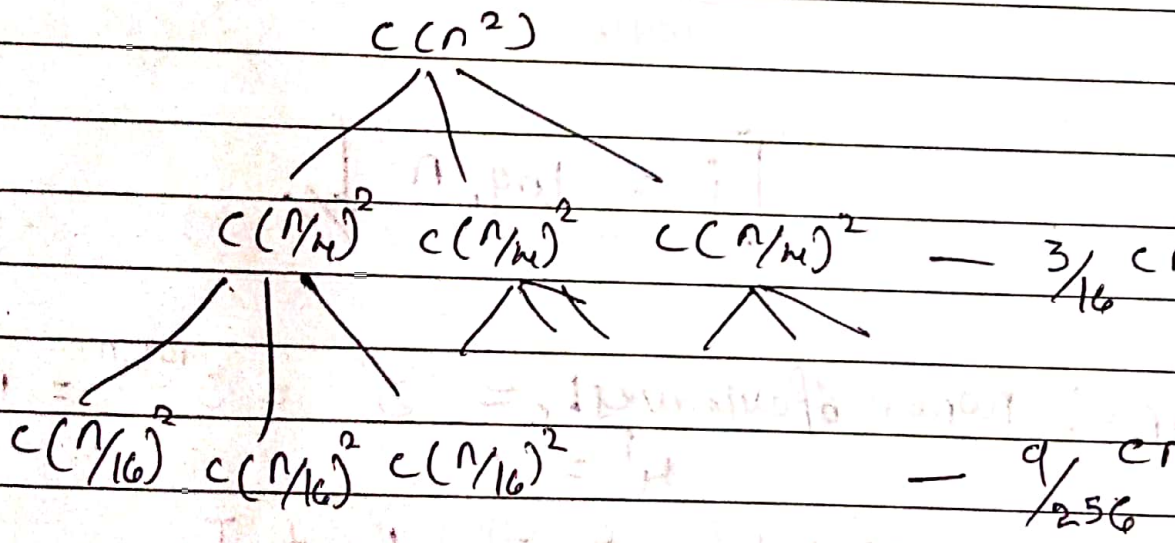
2) Cost of leaf node is max

3) Cost of each node is same

Case 1

1) $3T(n/4) + n^2$ → root





step 1: cost of each level : $(3/16)^i cn^2$

$I_c = 1/1 - 3/16$

$i = 0$ level 1
$i = 1$ level 2
$i = 2$ level 3

step 2: depth of the tree $= n/4^i = 1$

$n = 4^i$

Applying log on both sides

$$\log n = \log 4^i$$

$$\log n = i \log 4$$

$$\frac{\log n}{\log 4} = i$$

$$\boxed{i = \log_4 n}$$ //

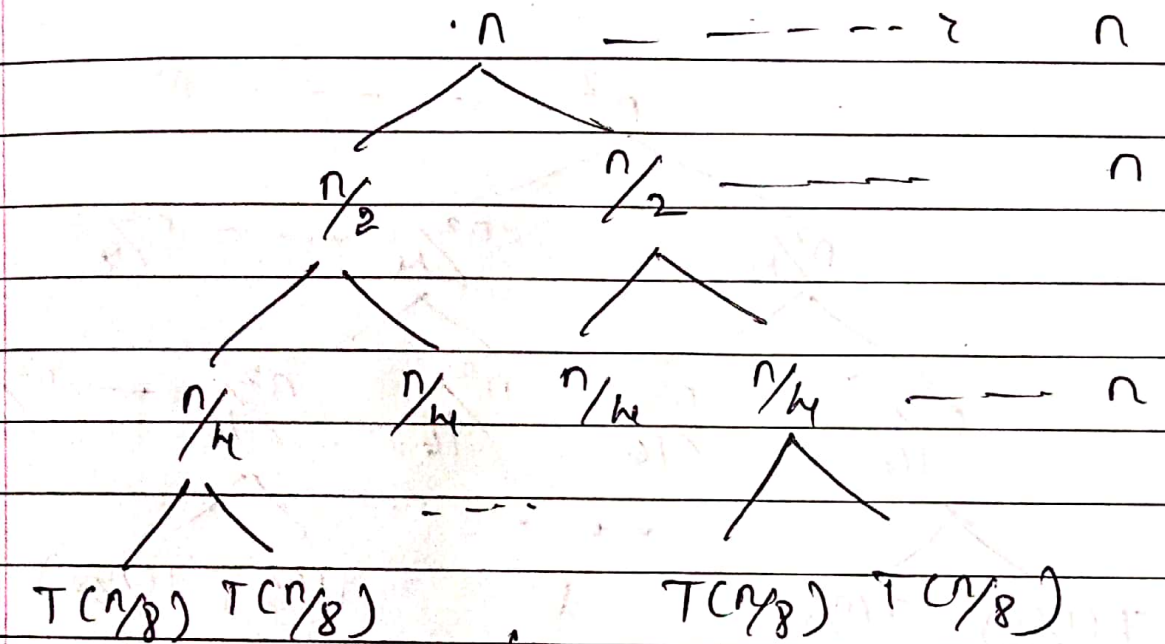
step 3: no of leaves : $L_c = 3^i = 3^{\log_4 n} = n^{\log_4 3}$

Total cost = $L_c + I_c$

$= n^{\log_4 3} + \frac{16}{13} n^2$

$\therefore T(n) \in \theta(n^2)$

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



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

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

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

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

$T(n/2^k) = T(1)$

$n = 2^k \Rightarrow k = \lg n$

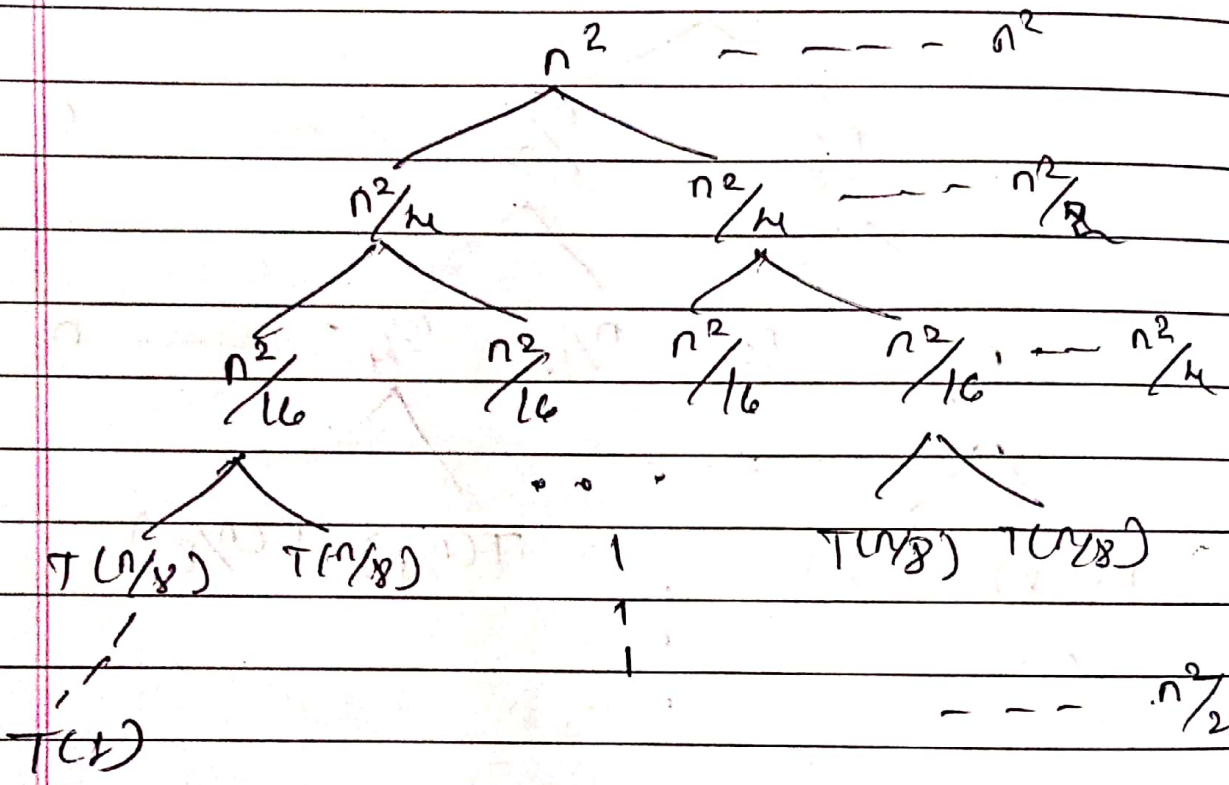
$L_c = 2^k \Rightarrow 2^{\log n} \Rightarrow n^{\log 2} = n \qquad I_c = k.n$

The total cost $= L_c + I_c = n + k.n$

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

$T(n) \in O(n \log n) \qquad = n + n \log n$

3 $T(n) = \begin{cases} 1 & : n = 1 \\ 2T(n/2) + n^2 & n > 1 \end{cases}$



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

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

$T(n/2^2) = 2T(n/8) + n^2/4^2$

$T(n/2^k) = T(1)$

$n = 2^k \Rightarrow k = \lg n$

$L_c = 2^k = 2^{\lg n} = n^{\lg 2} = n$

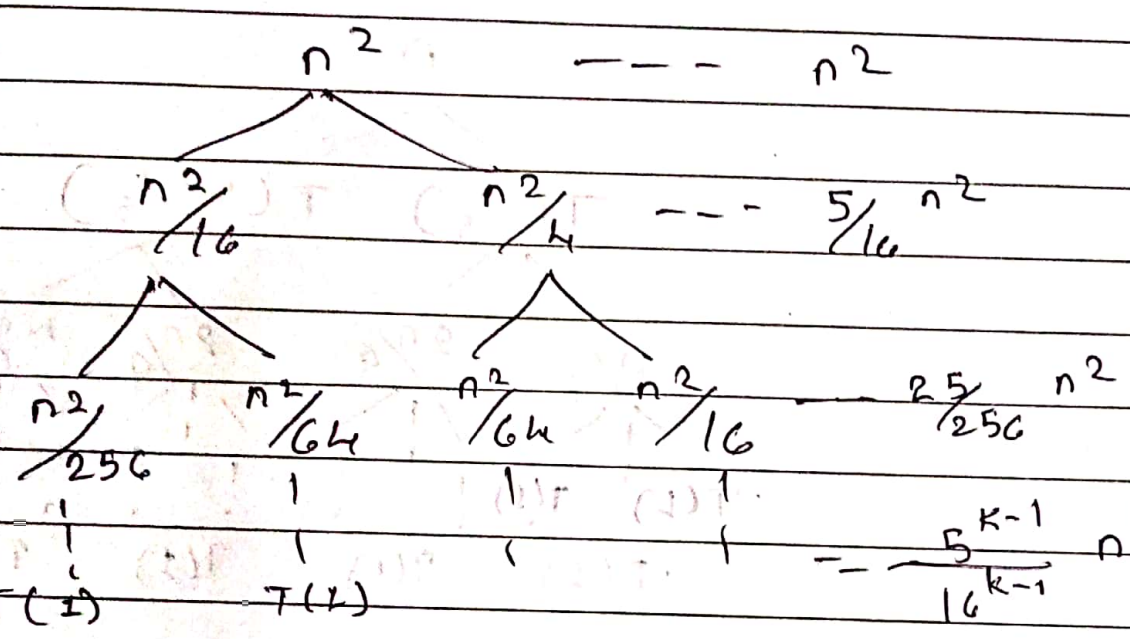
$I_c = n^2 \left[(1/2)^0 + (1/2)^1 + (1/2)^2 + \cdots + (1/2)^{k-1} \right]$

$I_c = n^2 \left[\frac{1}{1 - 1/2} \right] = 2n^2$

Total cost $= L_c + I_c = n + 2n^2$

$T(n) \in O(n^2)$

4 $T(n) = \begin{cases} 1 & n = 1 \\ T(n/4) + T(n/2) + n^2 & n > 1 \end{cases}$



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

$\begin{cases} T(n/4) = T(n/16) + T(n/8) + n^2/16 \\ T(n/2) = T(n/8) + T(n/4) + n^2/4 \end{cases}$

$\begin{cases} T(n/16) = T(n/64) + T(n/32) + n^2/256 \\ T(n/8) = T(n/32) + T(n/16) + n^2/64 \\ T(n/4) = T(n/16) + T(n/8) + n^2/16 \end{cases}$

$T(n/2^k) = T(1)$

$n = 2^k \Rightarrow k = \lg n \qquad k = 2^k \Rightarrow 2^{\lg n} \Rightarrow n^{\lg 2} \Rightarrow n$

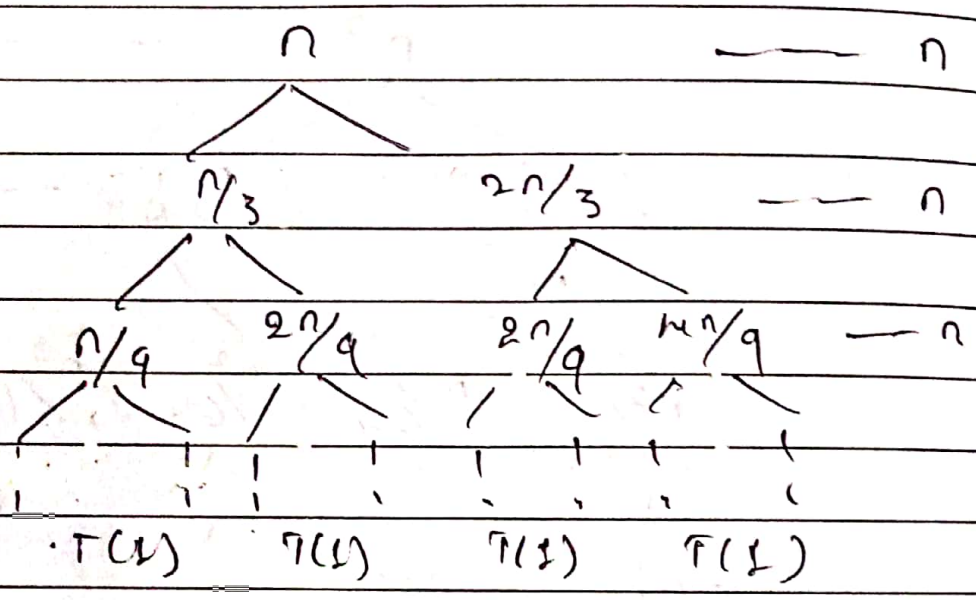
$I_c = n^2 \cdot [(5/16)^0 + (5/16)^1 + \cdots + (5/16)^{k-1}]$

$I_c = 1/(1 - 5/16) = 16/11\ n^2$

Total cost $= h_c + I_c = n + 16/11\ n^2 = n^2$

$T(n) \in O(n^2)$

5 $T(n) = \begin{cases} 1 & n = 1 \\ T(n/3) + T(2n/3) + n & n > 1 \end{cases}$



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

$T(n/3) = T(n/9) + T(2n/9) + n/3$

$T(2n/3) = T(2n/9) + T(4n/9) + 2n/3$

$T(n/9) = T(n/27) + T(2n/27) + n/9$

$T(2n/9) = T(2n/27) + T(4n/27) + 2n/9$

$T(4n/9) = T(4n/27) + T(8n/27) + 4n/9$

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

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

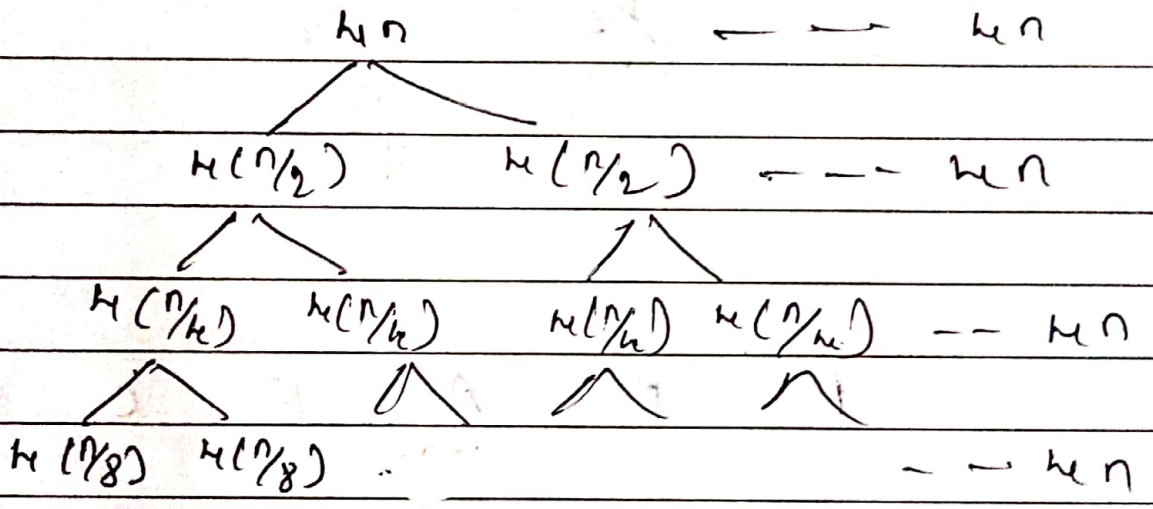
$I_c = kn \Rightarrow \log_{3/2} n = n \log_{3/2} n$

Total cost $= L_c + I_c = n^{\log_{3/2} 2} + n \log_{3/2} n$

$T(n) \in O(n \log n)$??

6 $T(n) = 2T(n/2) + 4n \qquad T(1) = 4$

$T(n) = 2T(n/2) + 4n$
$T(n/2) = 2T(n/4) + 4n/2$
$T(n/4) = 2T(n/8) + 4n/8$
$T(n/8) = 2T(n/16) + 4n/16$



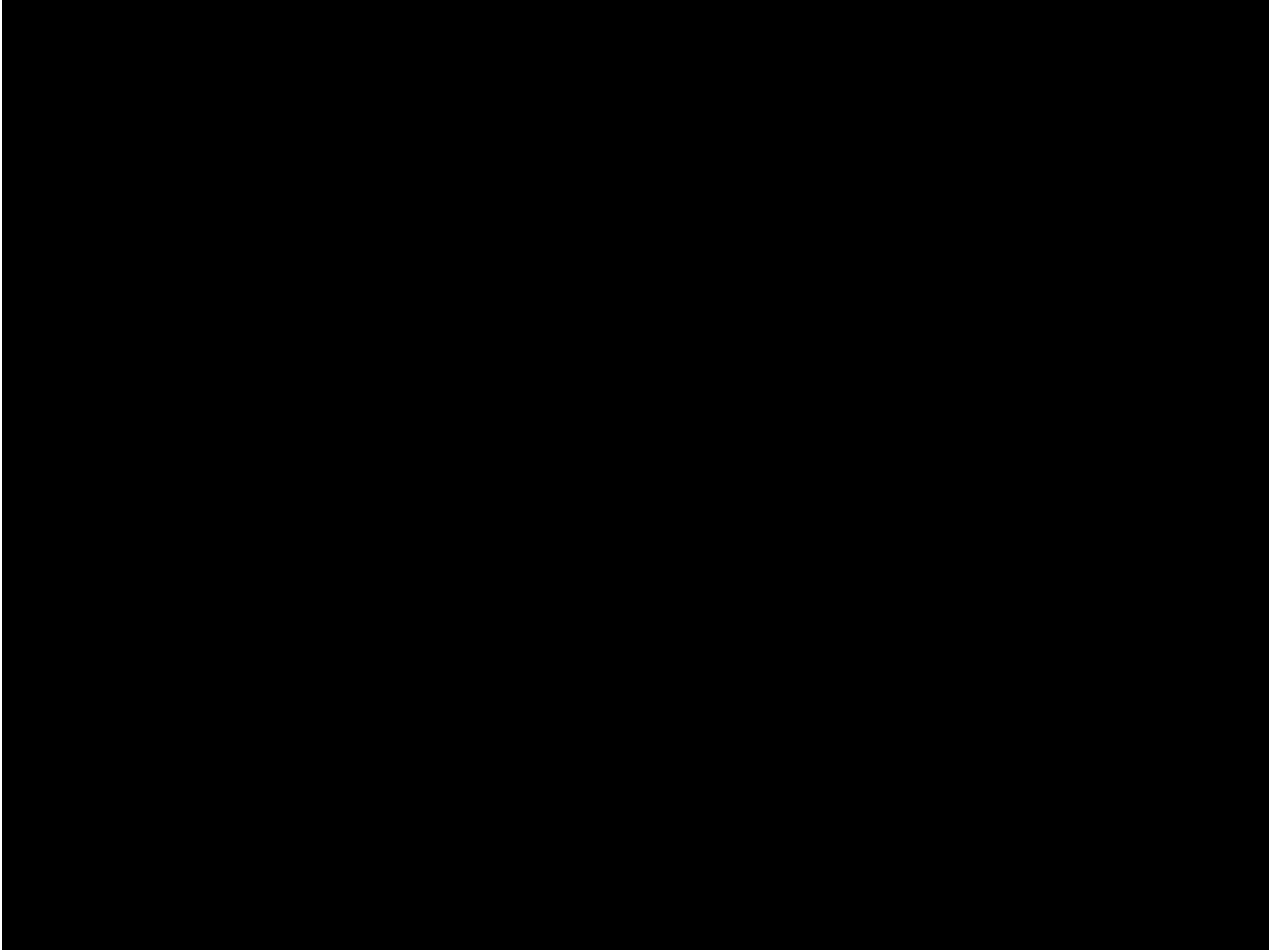
$T(n/2^k) = T(1) = 4$

$n/2^k = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$\sum_{k=0}^{\log_2 n} 4n \Rightarrow 4n(\log_2 n + 1)$

$L_c = 2^k = 2^{\log_2 n} = n^{\log_2 2} = n$
$I_c = 4n \cdot k = 4n(\log_2 n + 1)$
Total cost $= L_c + I_c = n + 4n(\log_2 n + 1)$
$= n + 4n\log_2 n + 4n$
$= 5n + 4n\log_2 n$

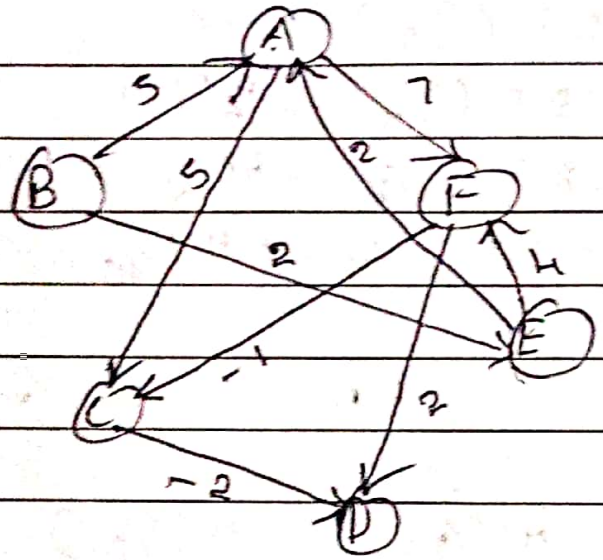
$T(n) \in O(n \log_2 n)$



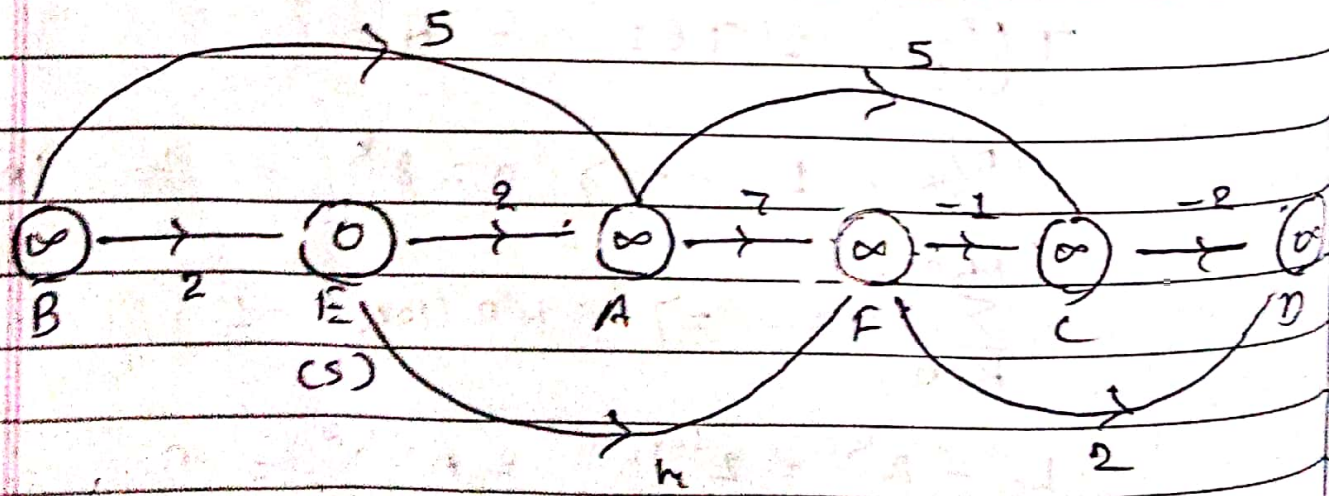
Unit - II

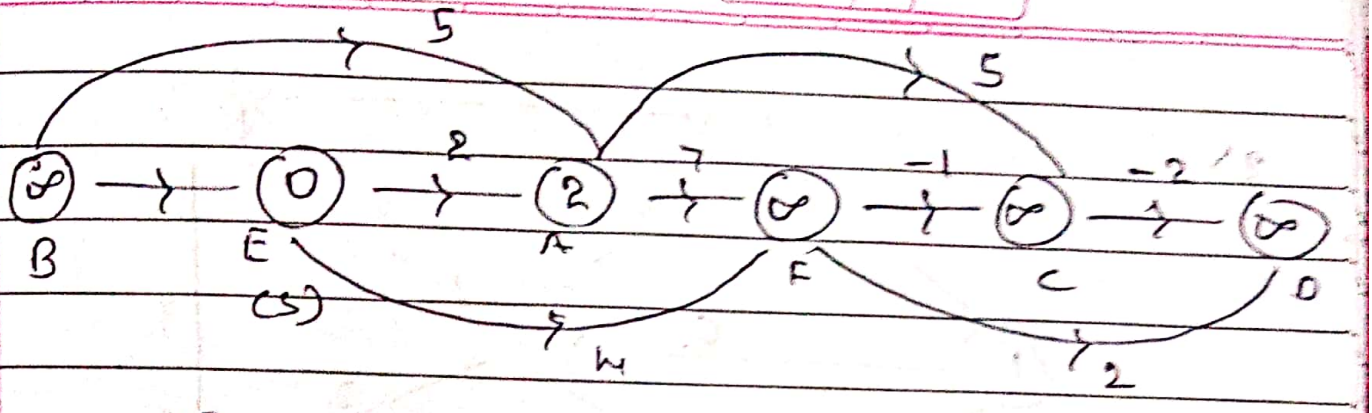
Shortest path in DAG

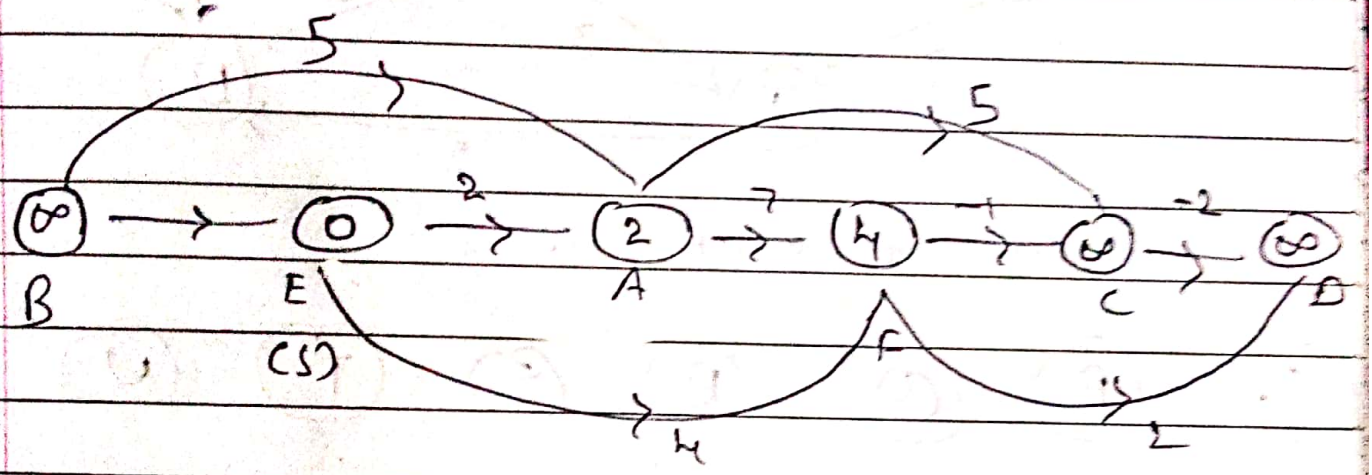
1) Topological sorting

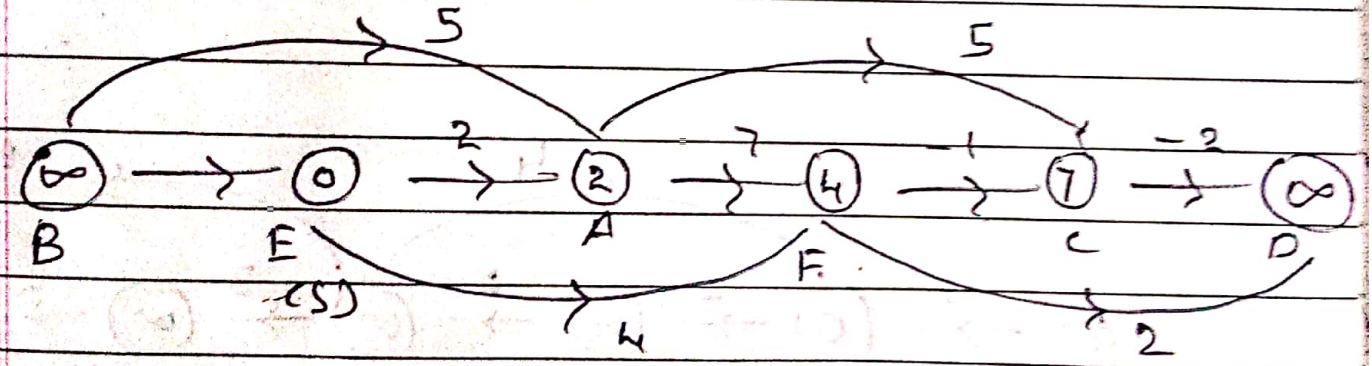


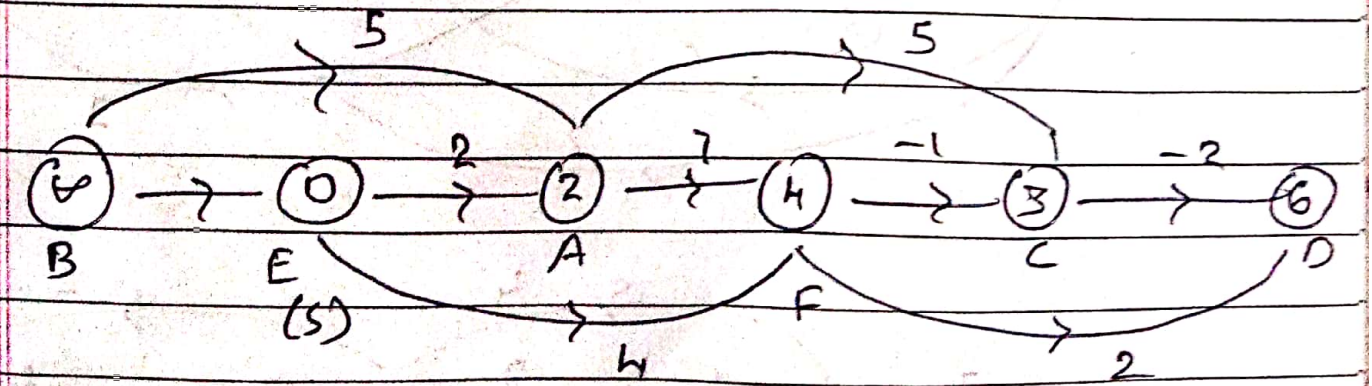
B E A F C D

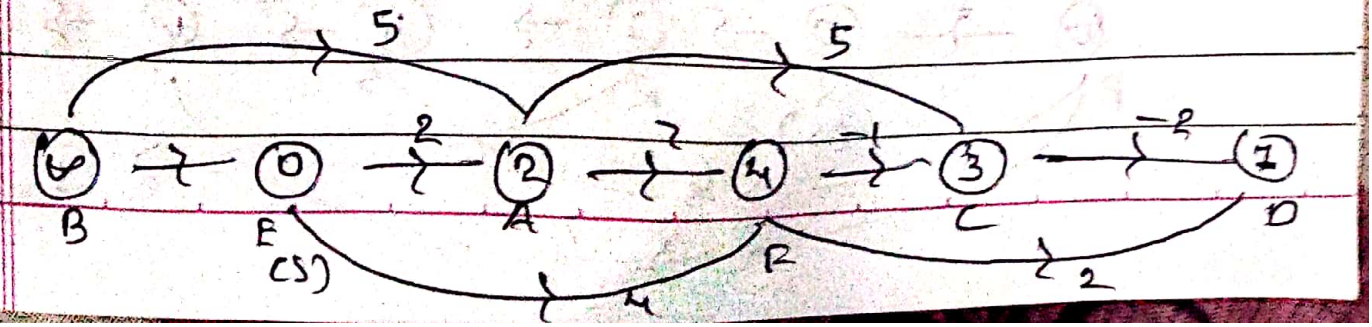




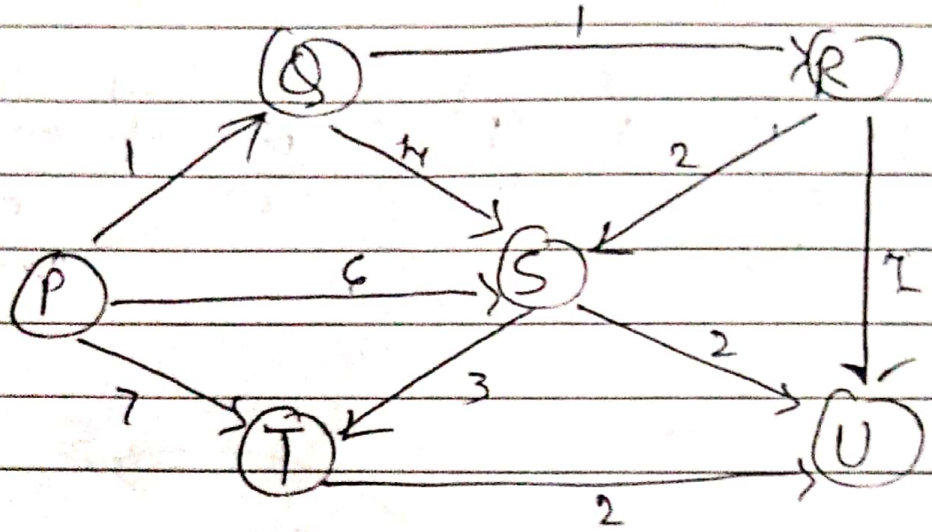




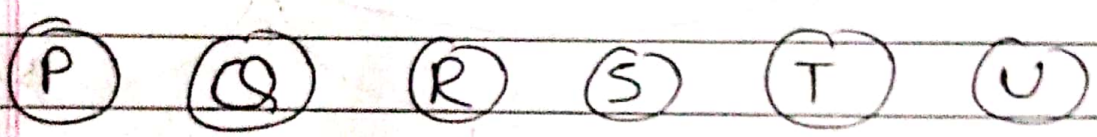


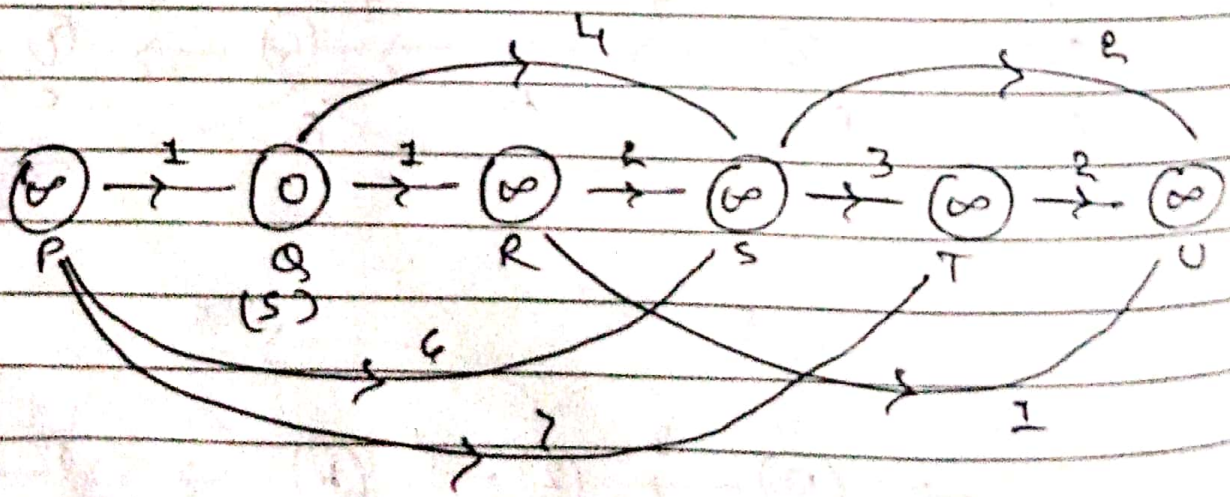


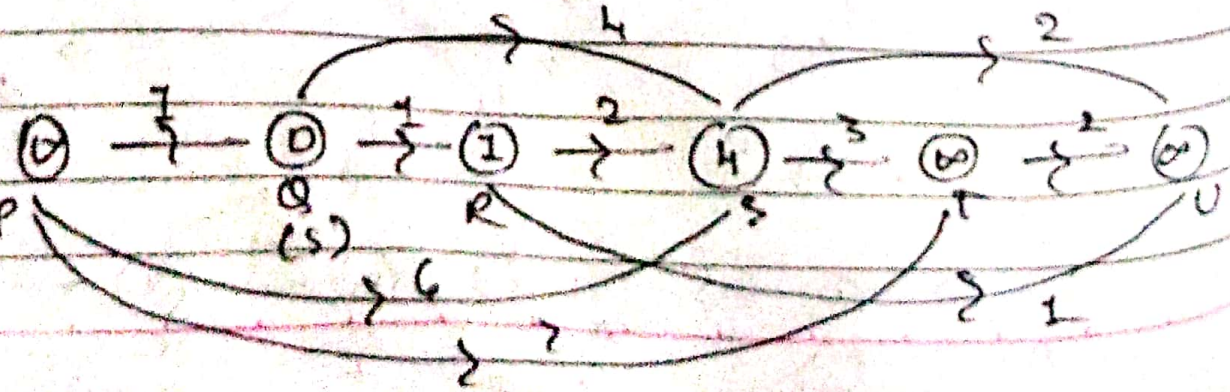
2)

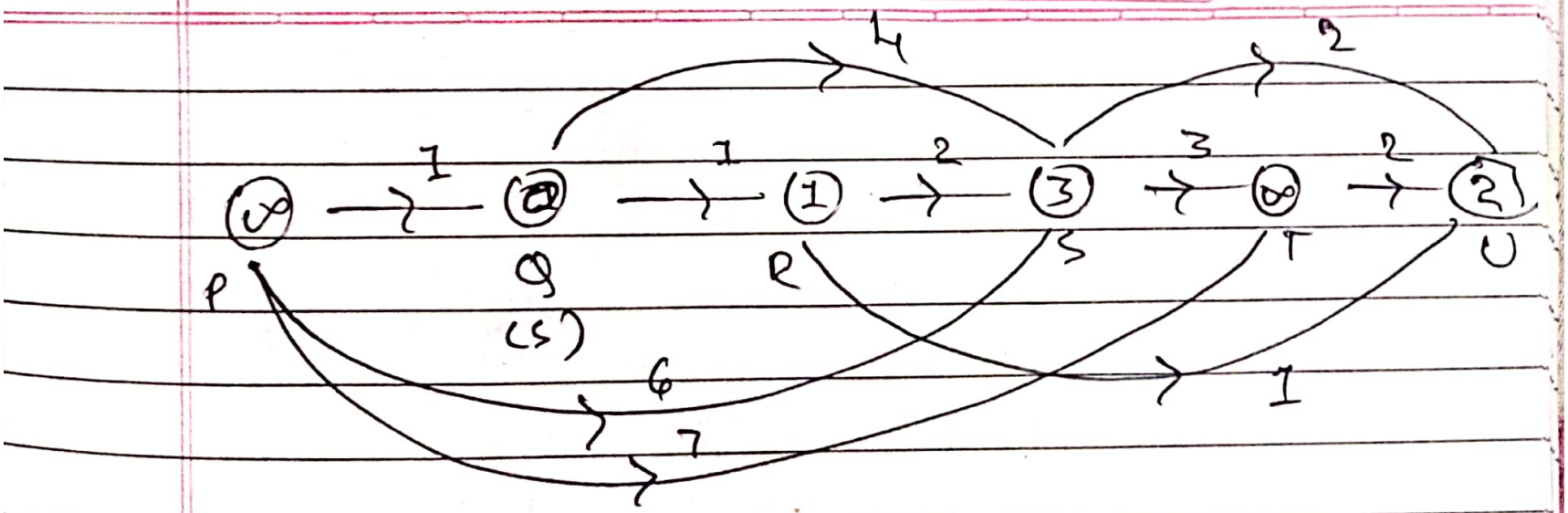


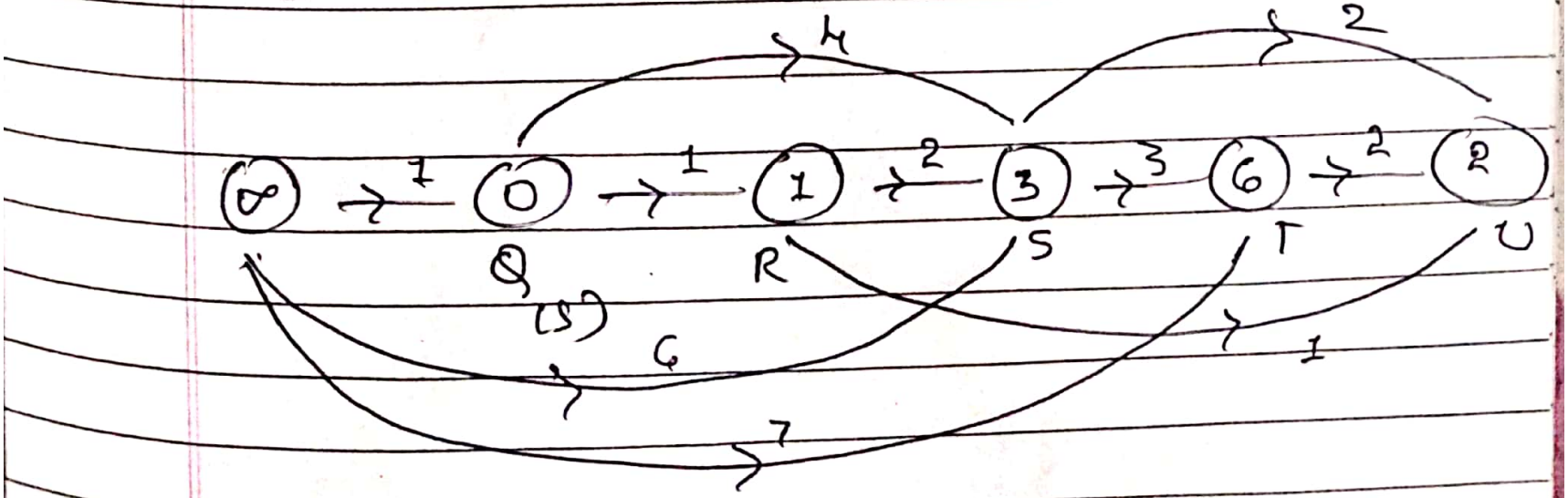
Topological sort









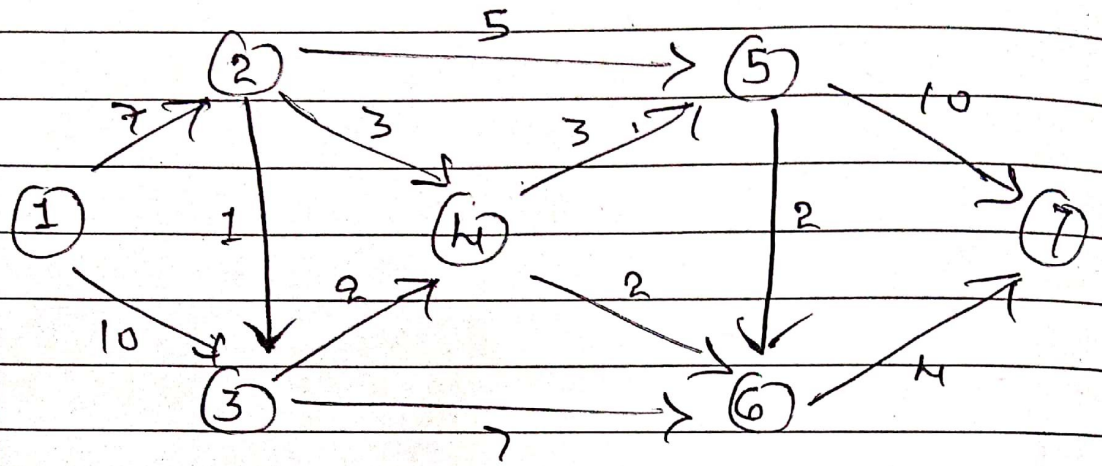


B Ramesh

Page No. :

Date

Maximal Flow Problem (Ford Fulkerson Rule)



The Ford Fulkerson method is used for solving maximum flow problem

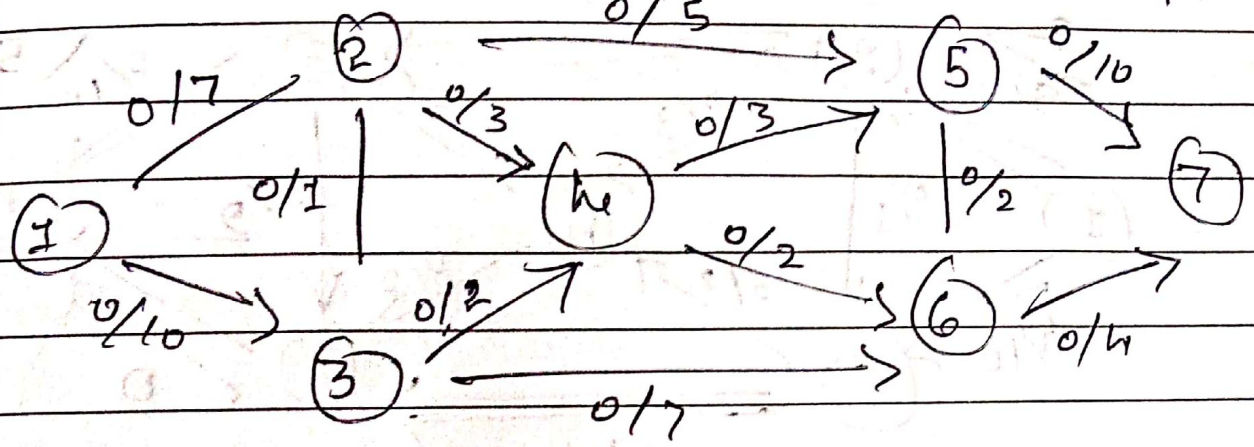
Source : The source vertex has all outward edge no inward edge

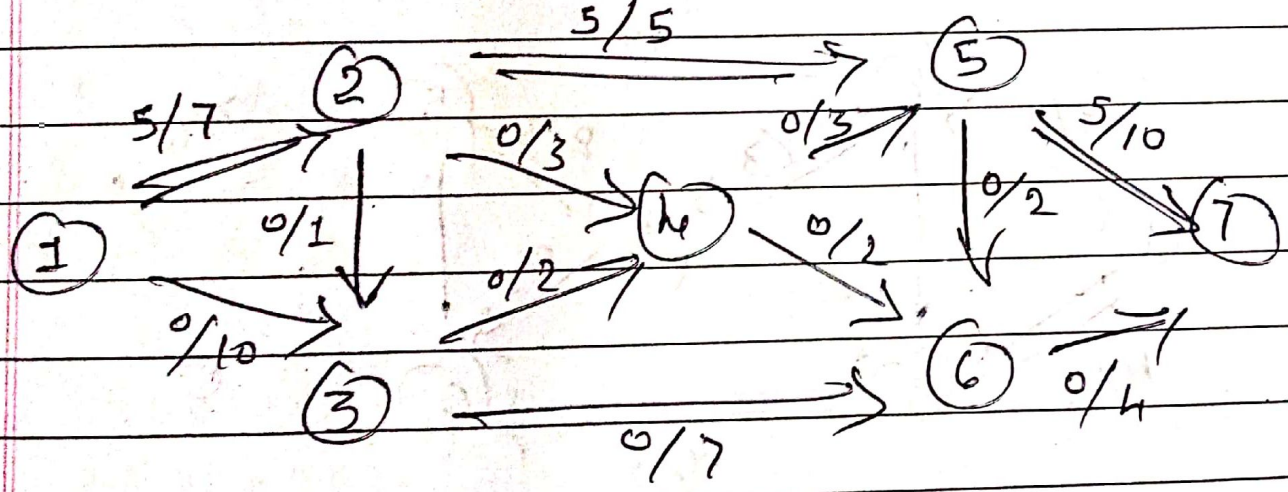
Sink : Sink will have all inward edge, no outward edge

Bottleneck capacity : Bottle neck capacity of the path is the minimum capacity of any edge on the path

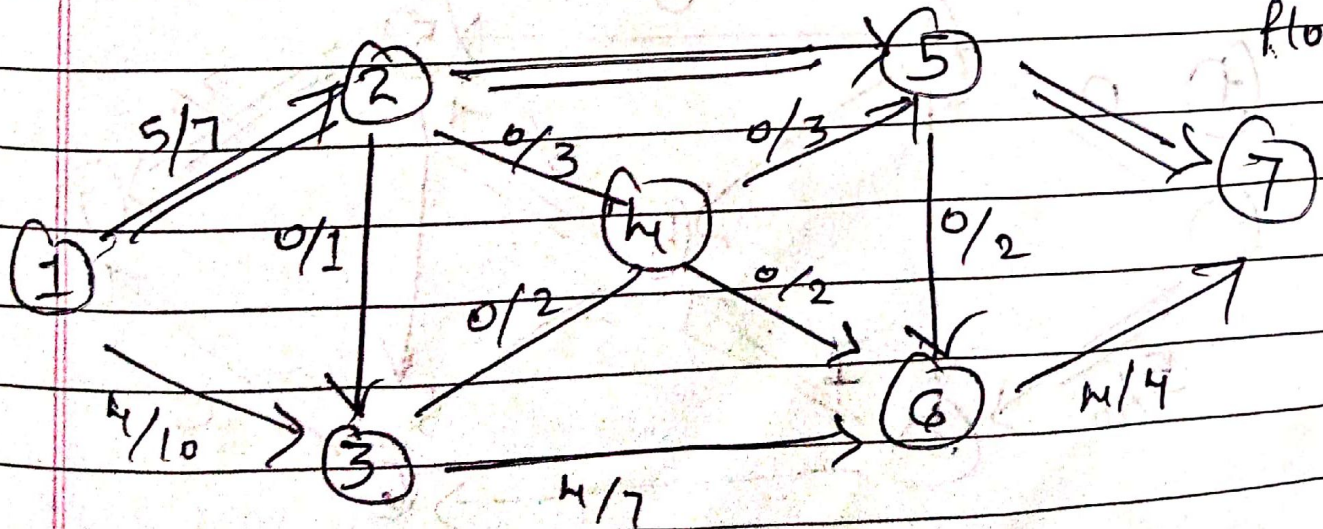
Residual capacity : Every edge of a residual graph has a value called capacity which is equal to orginal capacity minus current flow.

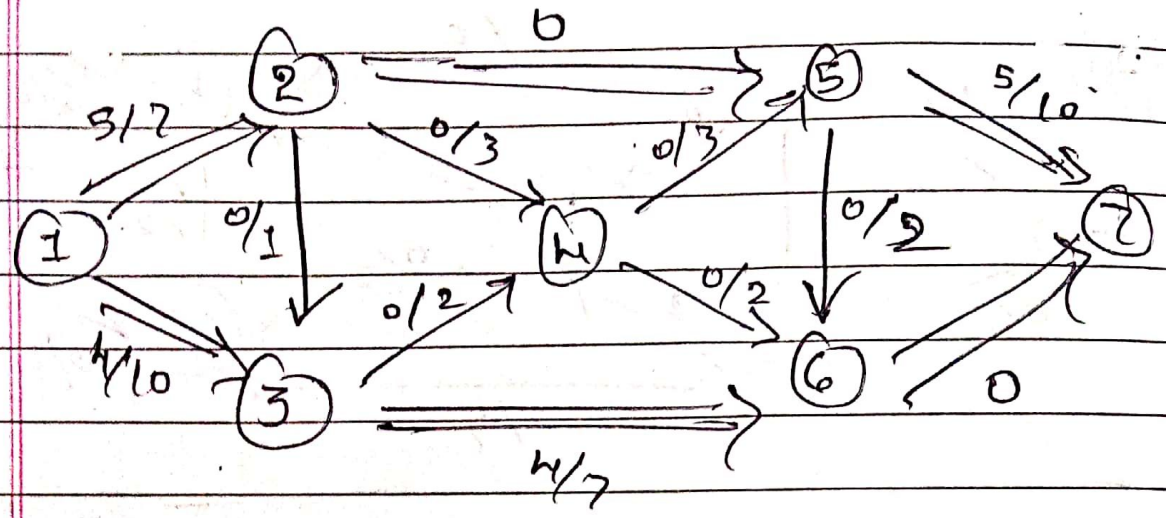
Flow=5

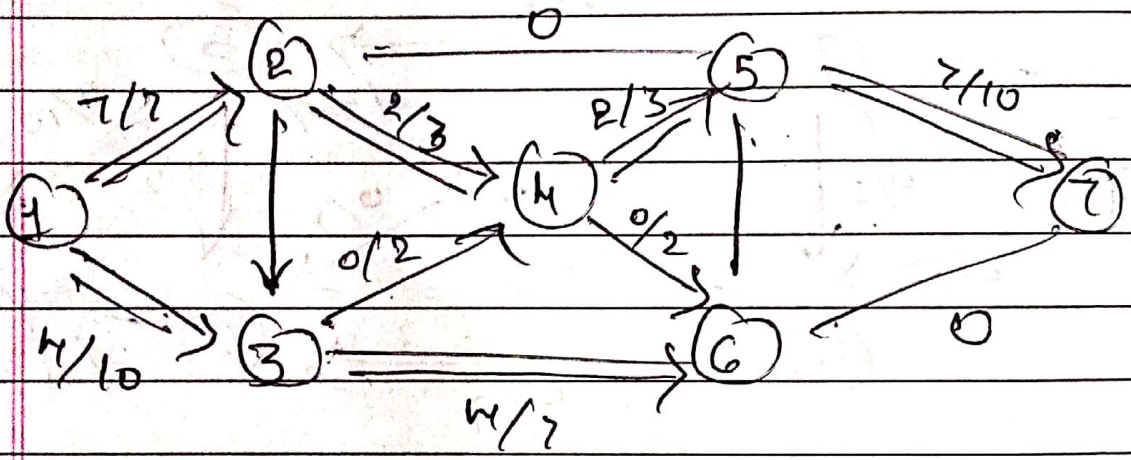


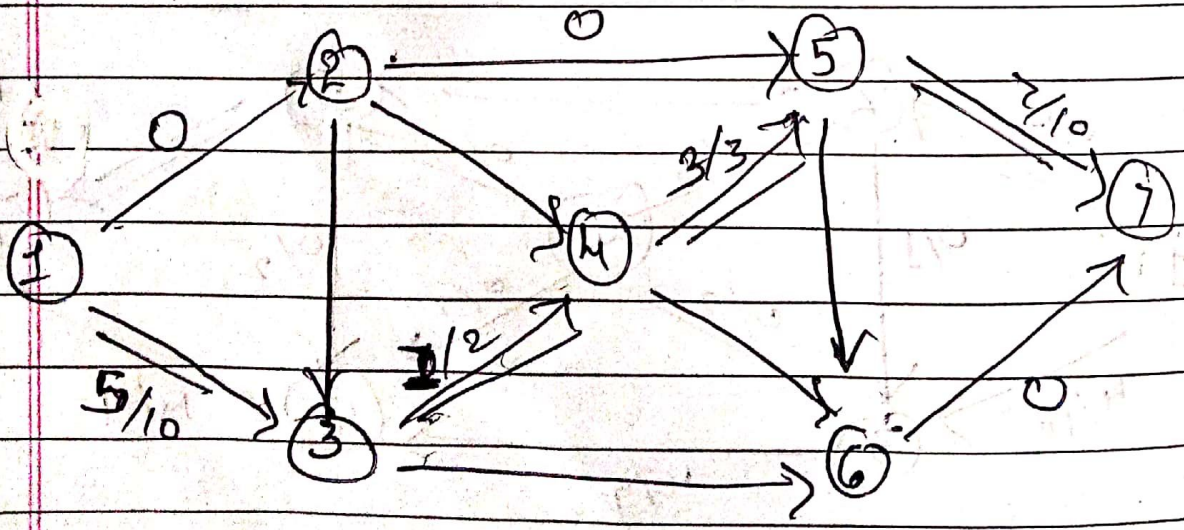


Flow=4

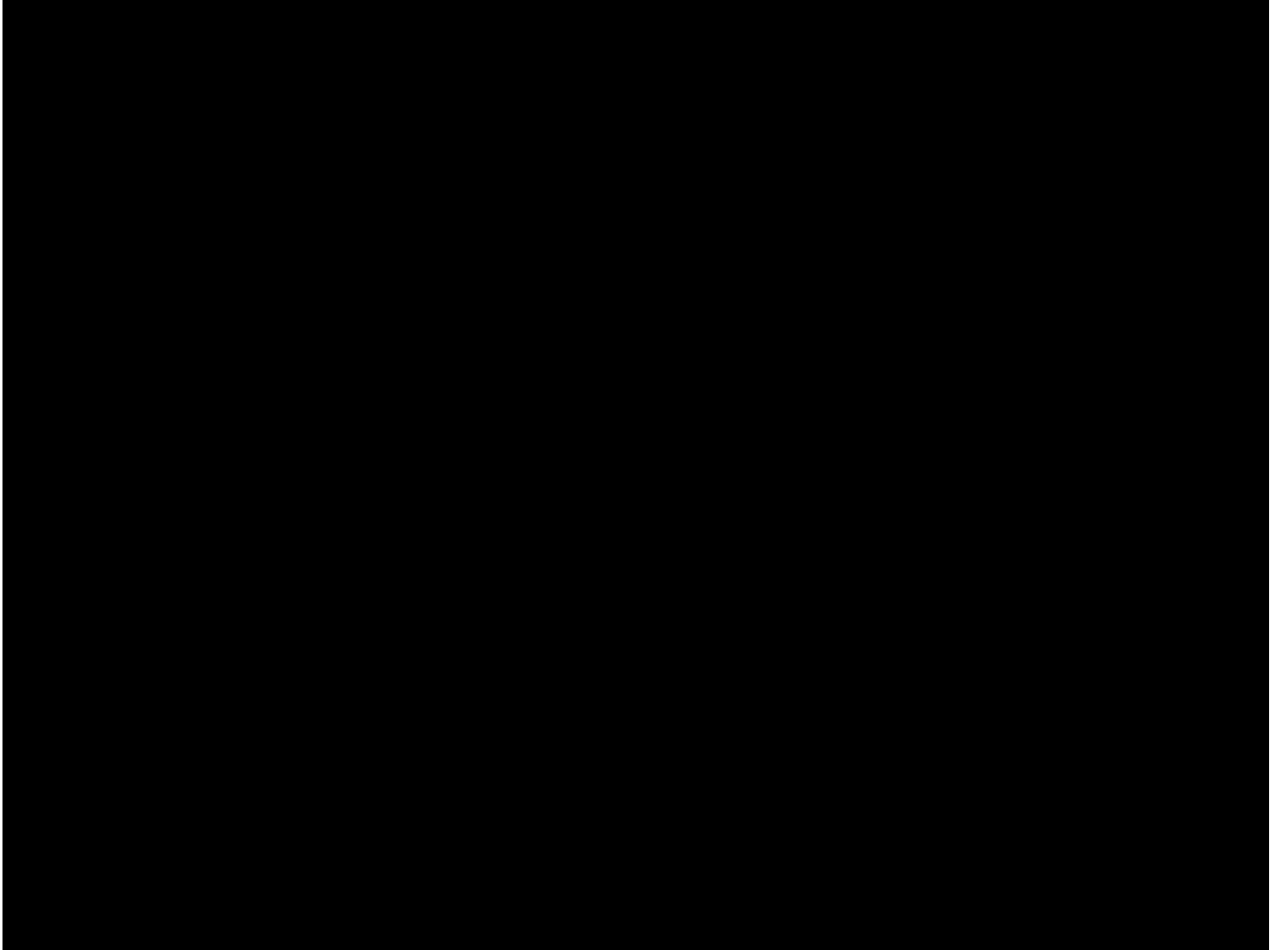








Maximum flow = 5+4+2+1 = 12



Unit - III

Extended Euclidian Algorithm [Tabular Method]

Find the value of gcd (99, 78) & corresponding x, y values using Extended Euclidian algorithm

a	b	a/b	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	(3)	2	3	0	1
3	0	-	3	1	0

GCD

$y^1 = x - (y * a/b)$
$= 1 - (0 * 2)$
$= 1$

$y^1 = x - (y * a/b)$
$= -2 - (3 * 3)$
$= -11$

$y^1 = x - (y * a/b)$
$= 0 - (1 * 2)$
$= -2$

$y^1 = x - (y * a/b)$
$= 3 - (-11 * 1)$
$= 14$

$y^1 = x - (y * a/b)$
$= 1 - (-2 * 1)$
$= 3$

$d = ax + by$
$= 99(-11) + 78(14)$
$= 3$ //

B Ramesh
Page No. :
Date : / /

Find the value of gcd (120, 23)

a	b	a/b	d	x	y
120	23	5	1	-9	47
23	5	4	1	2	-9
5	3	1	1	-1	2
3	2	1	1	1	-1
2	(1)	2	1	0	1
1	0	-	1	1	0

↓ gcd

$y' = x - (y * a/b)$
$= 1 - (0 * 2)$
$= 1$

$y' = x - (y * a/b)$
$= -1 - (2 * 47)$
$= -1 - 8 = -9$

$y' = x - (y * a/b)$
$= 0 - (1 * 1)$
$= -1$

$y' = 2 - (-9 * 5)$
$= 47$

$y' = x - (y * a/b)$
$= 1 - (-1 * 1)$
$= 2$

$d = ax + by$
$= 120(-9) + 23(47)$
$= -1080 + 1081$
$d = 1$

Euclid's Algorithm

B.Ramesh
Page No. :
Date : / /

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

Ex Find the value of gcd(30, 21) using Euclid's algorithm.

$$\begin{aligned}\gcd(30, 21) &= \gcd(21, 9) \\ &= \gcd(9, 3) \\ &= \gcd(3, 0) \\ &= 3 \ //\end{aligned}$$

Ex Find the value of gcd(68, 119) using Euclid's algorithm

$$\begin{aligned}\gcd(68, 119) &= \gcd(68, 51) \\ &= \gcd(51, 17) \\ &= \gcd(17, 0) \\ &= 17\end{aligned}$$

B Ramesh
Page No. :
Date :

Extended Euclidian Algorithm [Backward Substitution]

Find the value of the gcd (54, 41) & corresponding x, y values using Extended Euclidian Algorithm by backward substitution method

gcd (54, 41)

$54 = 1.41 + 13$

$41 = 3.13 + 2$

$13 = 6.2 + 1$

$2 = 2.1 + 0$

gcd (54, 41) = 1

$1 = 13 - 6.2$

$1 = 13 - 6[41 - 3.13]$

$1 = 13 - 6.41 + 18.13$

$1 = 19.13 - 6.41$

$1 = 19[54 - 1.41] - 6.41$

$1 = 19.54 - 19.41 - 6.41$

$1 = 19.54 - 25.41$

$1 = 19.54 + (-25).41$

$d = ax + by$

$= 54(19) + 41(-25)$

$d = 1026 - 1025$

$d = 1$ //

$x = 19 \quad y = -25$

B.Ramesh
Page No. :
Date : / /

Find the value of the gcd (17,43) & corresponding x, y values using Extended Euclidian Algorithm by backward substitution method

gcd (17,43)

$43 = 17.2 + 9$
$17 = 9.1 + 8$
$9 = 8.1 + 1$
$8 = 8.1 + 0$

gcd (17,43) = 1

$d = ax + by$
$= 43(2) + 17(-5)$
$= 86 - 85$
$d = 1$

$1 = 9 - 8.1$
$1 = 9 - 1[17 - 9.1]$
$1 = 9 - 17 + 9$
$1 = 2.9 - 17$
$1 = 2[43 - 17.2] - 17$
$1 = 2.43 - 17.4 - 17$
$1 = 2.43 - 5.17$
$1 = 2.43 + (-5).17$

$x = 2 \quad y = -5$

Page No.:
Date: / /

Find the value of the gcd (143, 7) & corresponding x, y values using Extended Euclidean Algorithm by backward substitution method

gcd (143, 7)

$143 = 20 \cdot 7 + 3$

$7 = 2 \cdot 3 + 1$

$3 = 3 \cdot 1 + 0$

gcd (143, 7) = 1

$1 = 7 - 2 \cdot 3$

$1 = 7 - 2(143 - 20 \cdot 7)$

$1 = 7 - 2 \cdot 143 + 40 \cdot 7$

$1 = 41 \cdot 7 - 2 \cdot 143$

$d = ax + by$

$= 143(-2) + 41 \cdot 7$

$d = 1$

$x = -2$

$y = 7$

B.Ramesh
Page No. :
Date : / /

Find the value of the gcd (77, 13) & corresponding x, y values using Extended Euclidian Algorithm by backward substitution method.

gcd (77, 13)

$77 = 5.13 + 13$
$13 = 1.12 + 1$
$12 = 12.1 + 0$

gcd (77, 13) = 1

$1 = 13 - 1.12$
$1 = 13 - 1[77 - 5.13]$
$1 = 13 - 77 + 5.13$
$1 = 6.13 + (-1)77$

$x = -1 \quad y = 6$

$d = ax + by$
$= 77(-1) + 13(6)$

$d = 1$

Multiplicative Inverse

$a \not\equiv 0 \mod n$ (n is prime number)

$a \cdot b \equiv 1 \mod n$ ['y' is multiplicative inverse of 'x']

$b \equiv a^{-1} \mod n$

Example:

$3 \not\equiv 0 \mod 5$

$3 \times 2 \equiv 1 \mod 5$

$2 \equiv 3^{-1} \mod 5$

∴ '2' is the multiplicative inverse of '3'

If n is not prime than the gcd of a, n should be 1 (coprime)

Example:

$5 \not\equiv 0 \mod 9$

$5 \cdot 2 \equiv 1 \mod 9$ $\quad \gcd(5,9) = 1$

In modular arithmetic we do not have a division operation, however, we do have modular inverses.

Find the multiplicative inverse of gcd (77, 13)

gcd (77, 13)

$77 = 5 \cdot 13 + 12$

$13 = 1 \cdot 12 + 1$

$12 = 12 \cdot 1 + 0$

gcd (77, 13) = 1

$1 = 13 - 1 \cdot 12$

$1 = 13 - 1 [77 - 5 \cdot 13]$

$1 = 13 - 77(1) + 5 \cdot 13$

$1 = 6 \cdot 13 + (-1) 77$

$x = -1 \quad y = 6$

$d = ax + by$

$= 77(-1) + 13(6)$

$d = 1$

Multiplicative inverse

MI = 13 - 1 = 12 //

$a \cdot b = 1 \bmod n$

$77 \cdot 12 = 1 \bmod 13$

$12 = 77^{-1} \bmod 13$ //

B Ramesh
Page No. :
Date : / /

Find the multiplicative inverse of gcd (17, 43)

gcd (17, 43)

$43 = 2.17 + 9$
$17 = 1.9 + 8$
$9 = 1.8 + 1$
$8 = 8.1$

$d = ax + by$
$= 43.2 + 17(-5)$

$d = 1$

gcd (17, 43) = 1

Multiplicative inverse

$1 = 9 - 1.8$
$1 = 9 - 1[17 - 1.9]$
$1 = 9 - 1.17 + 1.9$
$1 = 2.9 - 1.17$
$1 = 2[43 - 2.17] - 1.17$
$1 = 2.43 - 4.17 - 1.17$
$1 = 2.43 - 5.17$

$x = 2 \quad y = -5$

MI = 43 - 5 = 38

$a.b = 1 \bmod n$
$17.38 = 1 \bmod 43$
$38 = 17^{-1} \bmod 43$

Chinese Reminder Theorem

B.Ramesh
Page No. :
Date : / /

$$\left.\begin{aligned} x &\equiv a_1 \bmod m_1 \\ x &\equiv a_2 \bmod m_2 \\ x &\equiv a_3 \bmod m_3 \end{aligned}\right\} \text{ find } x ?$$

Steps to find x

1. Find our common modulus M

$$M = m_1 \times m_2 \times m_3 \cdots \times m_n$$

2. Find $M_1 = \frac{M}{m_1}, M_2 = \frac{M}{m_2} \cdots M_n = \frac{M}{m_n}$

3. Find our the inverses of $M_1^{-1}, M_2^{-1} \cdots M_n^{-1}$ with respect to $m_1, m_2 \cdots m_n$

4. $x = [(a_1 \times M_1 \times M_1^{-1}) + (a_2 \times M_2 \times M_2^{-1}) \cdots (a_n \times M_n \times M_n^{-1})] \bmod M$

The CRT is a theorem which gives a unique solution to simultaneous linear congruences with coprime moduli.

Ex $x \equiv 3 \bmod 5$
$x \equiv 1 \bmod 7$
$x \equiv 6 \bmod 8$

$a_1 = 3 \quad m_1 = 5$
$a_2 = 1 \quad m_2 = 7$
$a_3 = 6 \quad m_3 = 8$

Steps

1) $M = 5 \times 7 \times 8 = 280$

2) $M_1 = \frac{M}{m_1} = \frac{280}{5} = 56$

$M_2 = \frac{M}{m_2} = \frac{280}{7} = 40$

$M_3 = \frac{M}{m_3} = \frac{280}{8} = 35$

4) $x = (3 \times 56 \times 1) + (1 \times 40 \times 3) + (6 \times 35 \times 3) \bmod 280$
$x = 918 \bmod 280$
$x = 78$ //

3) i) 56 mod 5

$56 = 11 \times 5 + 1$

$5 = 5 \times 1 + 0$

$\gcd(56, 5) = 1$

$M_1^{-1} = 1$

$1 = 1.56 - 11 \times 5$

$1 = 1.56 + (-11) \times 5$

$x = 1 \quad y = -11$ //

$MI = 1$ //

ii) 40 mod 7

$40 = 5 \times 7 + 5$

$7 = 1 \times 5 + 2$

$5 = 2 \times 2 + 1$

$2 = 2 \times 1 + 0$

$\gcd(40, 7) = 1$

$M_2^{-1} = 3$

$1 = 5 - 2.2$

$1 = 5 - 2[7 - 1.5]$

$1 = 5 - 2.7 + 2.5$

$1 = 3.5 - 2.7$

$1 = 3[40 - 5 \times 7] - 2.7$

$1 = 3.40 - 15.7 - 2.7$

$1 = 3.40 + (-17).7$

$MI = 3$

iii) 35 mod 8

$35 = 4 \times 8 + 3$

$8 = 2 \times 3 + 2$

$3 = 1 \times 2 + 1$

$2 = 2 \times 1 + 0$

$\gcd(35, 8) = 1$

$MI = 3$

$M_3^{-1} = 3$

$1 = 3 - 1.2$

$1 = 3 - 1.[8 - 2.3]$

$1 = 3 - 1.8 + 2.3$

$1 = 3.3 - 1.8$

$1 = 3.[35 - 4.8] - 1.8$

$1 = 3.35 - 12.8 - 1.8$

$1 = 3.35 - 13.8$

Divisor) Dividend (Quotient

Reminder

B.Ramesh

Page No. :

Date : / /

Ex $x \equiv 1 \bmod 9$

$x \equiv 2 \bmod 8$

$x \equiv 3 \bmod 7$

$a_1 = 1 \qquad m_1 = 9$

$a_2 = 2 \qquad m_2 = 8$

$a_3 = 3 \qquad m_3 = 7$

Steps

1) $M = 9 \times 8 \times 7 = 504$

2) $M_1 = \frac{504}{9} = 56 \qquad M_2 = \frac{504}{8} = 63 \qquad M_3 = \frac{504}{7} = 72$

3) i) $56 \bmod 9 \qquad M_1^{-1} = 5$

$56 = 6 \times 9 + 2$

$9 = 4 \times 2 + 1$

$2 = 2 \times 1 + 0$

$gcd(56, 9) = 1$

$1 = 9 - 4.2$

$1 = 9 - 4.[56 - 6 \times 9]$

$1 = 9 - 4.56 + 24 \times 9$

$1 = 25.9 + (-4).56$

$MI = 9 - 4 = 5$

B.Ramesh
Page No.:
Date: / /

Dividend Divisor
↑ ↑

ii) 63 mod 8 $M_2^{-1} = 7$

$63 = 7 \times 8 + 7$
$8 = 1 \cdot 7 + 1$
$7 = 7 \cdot 1 + 0$

gcd (63, 8) = 1

$1 = 8 - 1 \cdot 7$
$1 = 8 - 1 \cdot [63 - 7 \cdot 8]$
$1 = 8 - 1 \cdot 63 + 7 \cdot 8$
$1 = 8 \cdot 8 + (-1) \cdot 63$

M.I = 8 - 1 = 7

iii) 72 mod 7 $M_3^{-1} = 4$

$72 = 10 \times 7 + 2$
$7 = 3 \times 2 + 1$
$2 = 2 \times 1 + 0$

gcd (72, 7) = 1

$1 = 7 - 3 \cdot 2$
$1 = 7 - 3 [72 - 10 \cdot 7]$
$1 = 7 - 3 \cdot 72 + 30 \cdot 7$
$1 = 31 \cdot 7 - 3 \cdot 72$

MI = 7 - 3 = 4

4) $x = (1 \times 56 \times 5) + (2 \times 63 \times 7) + (3 \times 72 \times 4) \mod 504$

$x = 280 + 882 + 864 \mod 504$
$x = 2026 \mod 504$
$x = 10$ //

B Ramesh
Page No. :

Solving Modular Linear Equations

$ax \equiv b \pmod{n}$ $\quad a>0,\ n>0$

a, b and n are given

x is unknown

Modular-Linear-Equation-Solver (a, b, n)

1. $(d, x', y') = \text{Extended-Euclid}(a, n)$
2. if $d \mid b$
3. $\quad x_0 = x'(b/d) \bmod n$
4. $\quad$ for $i = 0$ to $d-1$
5. $\quad\quad$ print $(x_0 + i(n/d)) \bmod n$
6. else print "no solutions"

Corollary - 1

The equation $ax \equiv b \pmod{n}$ is solvable for the unknown x if and only if $d \mid b$ where $d = \gcd(a, n)$

Corollary - 2

The equation $ax \equiv b \pmod{n}$ either has d distinct solution modulo n, where $d = \gcd(a, n)$, or it has no solutions.

Ex 1 $14x \equiv 30 \bmod 100$

$a = 14 \quad b = 30 \quad n = 100$

$100 = 7 \times 14 + 2$
$14 = 7 \times 2 + 0$

$\gcd(14, 100) = 2$
$d = 2$

$R = 1.100 - 7.14$
$2 = 1.100 + (-7).14$

$x' = -7 \quad y' = 1$

if $R \mid 30$

$x_0 = x'(b/d) \bmod n$
$\equiv -7(30/2) \bmod 100$
$\equiv -7 \times 15 \bmod 100$
$\equiv -105 \bmod 100$
$\equiv -105 + 2 \times 100 \bmod 100$
$\equiv 95 \bmod 100$

$x_1 \equiv x_0 + 1(n/d) \bmod 100$
$\equiv 95 + 1(100/2) \bmod 100$
$= 95 + 50 \bmod 100$
$= 145 \bmod 100 \equiv 45$ //

$\therefore x = 95, 45$

B.Ramesh
Page No. :
Date : / /

Ex 2) $35x \equiv 10 \bmod 50$

$a = 35 \quad b = 10 \quad n = 50$

$50 = 1 \times 35 + 15$
$35 = 2 \times 15 + 5$
$15 = 3 \times 5 + 0$

$\gcd(35, 50) = 5$
$d = 5$

$5 = 35 - 2.15$
$5 = 35 - 2[50 - 1.35]$
$5 = 35 - 2.50 - 2.35$
$5 = 3.35 - 2.50$
$5 = 3.35 + (-2)50$

$x' = 3$
$y = -2$

if $5 | 10$

$x_0 = x'(b/d) \bmod n$
$= 3(10/5) \bmod 50$
$= 6 \bmod 50$

for i = 0 to 4

i = 1 $(x_0 + i(50/5)) \bmod n$

$(6 + 1(50/5)) \bmod 50$

$(6 + 10) \bmod 50$

$16 \bmod 50$

i = 2 $(6 + 2(50/5)) \bmod 50$

$(6 + 20) \bmod 50$

$26 \bmod 50$

i = 3 $(6 + 3(50/5)) \bmod 50$

$(6 + 30) \bmod 50$

$36 \bmod 50$

i = 4 $(6 + 4(50/5)) \bmod 50$

$(6 + (40)) \bmod 50$

$46 \bmod 50$

Values of x are 6, 16, 26, 36, 46

Ex 3) $17x \equiv 3 \pmod{29}$

$a = 17 \quad b = 3 \quad n = 29$

$29 = 1 \times 17 + 12$
$17 = 1 \times 12 + 5$
$12 = 2 \times 5 + 2$
$5 = 2 \times 2 + 1$
$2 = 2 \times 1 + 0$

$\gcd(17, 29) = 1$
$d = 1$

$1 = 5 - 2 \times 2$
$1 = 5 - 2 \times [12 - 2 \times 5]$
$1 = 5 - 2 \times 12 + 4 \times 5$
$1 = 5.5 - 2 \times 12$
$1 = 5[17 - 1 \times 12] - 2 \times 12$
$1 = 5.17 - 5 \times 12 - 2 \times 12$
$1 = 5.17 - 7 \times 12$
$1 = 5.17 - 7 \times [29 - 1 \times 17]$
$1 = 5.17 - 7.29 + 7 \times 17$
$1 = 12.17 - 7.29$

$x' = 12 \quad y' = -7$

if $1 \mid 3$

$x_0 = a^1 (3/d) \bmod n$

$= 12 (3/1) \bmod 29$

$= 36 \bmod 29$

$x_0 = 7$ //

B.Ramesh
Page No.:
Date: / /

Fermat's Theorem

Check wheather the given number is prime or not

a and p → +ve Integers

condition: a does not divides p } co prime
p does not divides a }

$$a^{p-1} \equiv 1 \bmod p$$

Ex 1) $p = 7 \quad a = 2$

$a^{p-1} \equiv 1 \bmod p$

$2^{7-1} \equiv 1 \bmod 7$

$2^6 \equiv 1 \bmod 7$

$64 \equiv 1 \bmod 7$

$64 \bmod 7 \equiv 1$ condition satisfied

∴ It satisfies Fermat's theorem

∴ 7 is a prime number

Page No.:
Date: / /

Ex 2 $p = 27 ; a = 2$

$a^{p-1} \equiv 1 \mod p$

$2^{27-1} \equiv 1 \mod 27$

$2^{26} \equiv 1 \mod 27$

$2^{26} \mod 27 = 13$

$\therefore 2^{26} \not\equiv 1 \mod 27$

$\therefore$ Fermat's theorem fails

$\therefore$ 27 is not a prime number

Ex 3 $p = 7 \quad a = 3$

$a^{p-1} \equiv 1 \mod p$

$3^{7-1} \equiv 1 \mod 7$

$3^6 \equiv 1 \mod 7$

$729 \mod 7 \equiv 1$ condition satisfied

$\therefore$ It satisfies Fermat's theorem

$\therefore$ 7 is a prime number

B.Ramesh

Page No. :

Date : / /

Carmichael Numbers

* A Carmichael number is a composite number N such that,
* $\forall\ a \in \{1, \ldots N-1\}$
if $\gcd(a, N) = 1$ then $a^{N-1} \equiv 1 \pmod{N}$

i.e every possible 'a' passes the Fermat test

Miller-Rabin Algorithm for Primality Test

Miller Rabin (n, a)

find m & k such that $n-1 = m \times 2^k$

$T = a^m \bmod n$

if ($T = \pm 1$) return n is a prime

for ($i = 1$ to $k-1$)

{

$T = T^2 \bmod n$

if ($T = +1$) return n is composite

if ($T = -1$) return n is prime

}

return n is composite

1. Perform $n = 1$ such that $n-1 = 2^k m$
2. If $k \leq 1$ calculate $T = a^m \bmod n$
   If $T = \pm 1$ number is prime else composite
3. If $k > 1$ calculate $T = T^2 \bmod n$
   if $T = 1$ number is composite
   if $T = -1$ number is prime
   else composete.

B.Ramesh
Page No. :
Date : / /

Ex 1) $n = 27$, $a = 2$

$n - 1 = m \times 2^k$

$27 - 1 = m \times 2^k$

$26 = 13 \times 2^1$

$k \leq 1$

$T = 2^{13} \bmod 27$

$T = 11$ //

∴ It is a composite number

Ex 2) $n = 61$, $a = 2$

$n - 1 = m \times 2^k$

$61 - 1 = m \times 2^k$

$60 = 15 \times 2^2$

$k > 1$

$T = T^2 \bmod n$

$T = 2^{15} \bmod 61$

$T = 11$

$T = (11)^2 \bmod 61$

$= 121 \bmod 61$

$= 60$

$T = T - n$

$= 60 - 61$

$T = -1$

$\therefore$ It is a prime number

The Miller-Rabin primality test is a probabilistic primality test. An algorithm which determines whether a given number is likely to be prime, similar to the Fermat primality test.

It is of historical significance for the research of polynomial-time deterministic primality test. Its probabilistic variants remain widely used in practice as one of the simplest and fastest test known.

Ex 3) $n = 53$, $a = 2$

$n - 1 = 2^k \cdot m$

$53 - 1 = 2^k \cdot m$

$52 = 2^2 \cdot 13$

$k > 1$ $\quad m = 13$

$T = T^2 \bmod n$

$T = a^m \bmod n$

$= 2^{13} \bmod 53$

$T = 30$

$T = T^2 \bmod n$

$= (30)^2 \bmod 53$

$T = 52$

$T = T - n$

$= 52 - 53$

$T = -1$

∴ It is a prime number

RSA Algorithm

Ex 1) $p = 7 \quad q = 11$

$n = p \times q = 7 \times 11 = 77$

$\phi = (p-1)(q-1) = (7-1)(11-1)$

$= 6 \times 10 = 60$ //

$1 < e < \phi \Rightarrow e = 13$

$d = ?$

$d \times e \bmod \phi = 1$

$d \times 13 \bmod 60 = 1$

$ax + by = \gcd(a, b)$

$a = \phi = 60 \quad b = e = 13$

$60x + 13y = \gcd(60, 13)$

$60 = 4 \times 13 + 8$	$8 = 60 - 4 \times 13$
$13 = 1 \times 8 + 5$	$5 = 13 - 1 \times 8$
$8 = 1 \times 5 + 3$	$3 = 8 - 1 \times 5$
$5 = 1 \times 3 + 2$	$2 = 5 - 1 \times 3$
$3 = 1 \times 2 + 1$	$1 = 3 - 1 \times 2$
$2 = 2 \times 1 + 0$	

$\gcd(60, 13) = 1$

B.Ramesh
Page No. :
Date : / /

$1 = 3 - 1 \times 2$

$1 = 3 - 1 \times (5 - 1 \times 3)$

$1 = 3 - 5 + 1 \times 3$

$1 = 2 \times 3 - 5$

$1 = 2(8 - 1 \times 5) - 5$

$1 = 2.8 - 2 \times 5 - 5$

$1 = 2.8 - 3 \times 5$

$1 = 2.8 - 3 \times (13 - 1 \times 8)$

$1 = 2.8 - 3 \times 13 + 3 \times 8$

$1 = 5.8 - 3 \times 13$

$1 = 5(60 - 4 \times 13) - 3 \times 13$

$1 = 5.60 - 20 \times 13 - 3 \times 13$

$1 = 5.60 - 23 \times 13$

$x = 5 \quad y = -23$

$d = -23$

if $d > \phi \Rightarrow d = d \bmod \phi$

if d is -ve $\Rightarrow d = d + \phi$

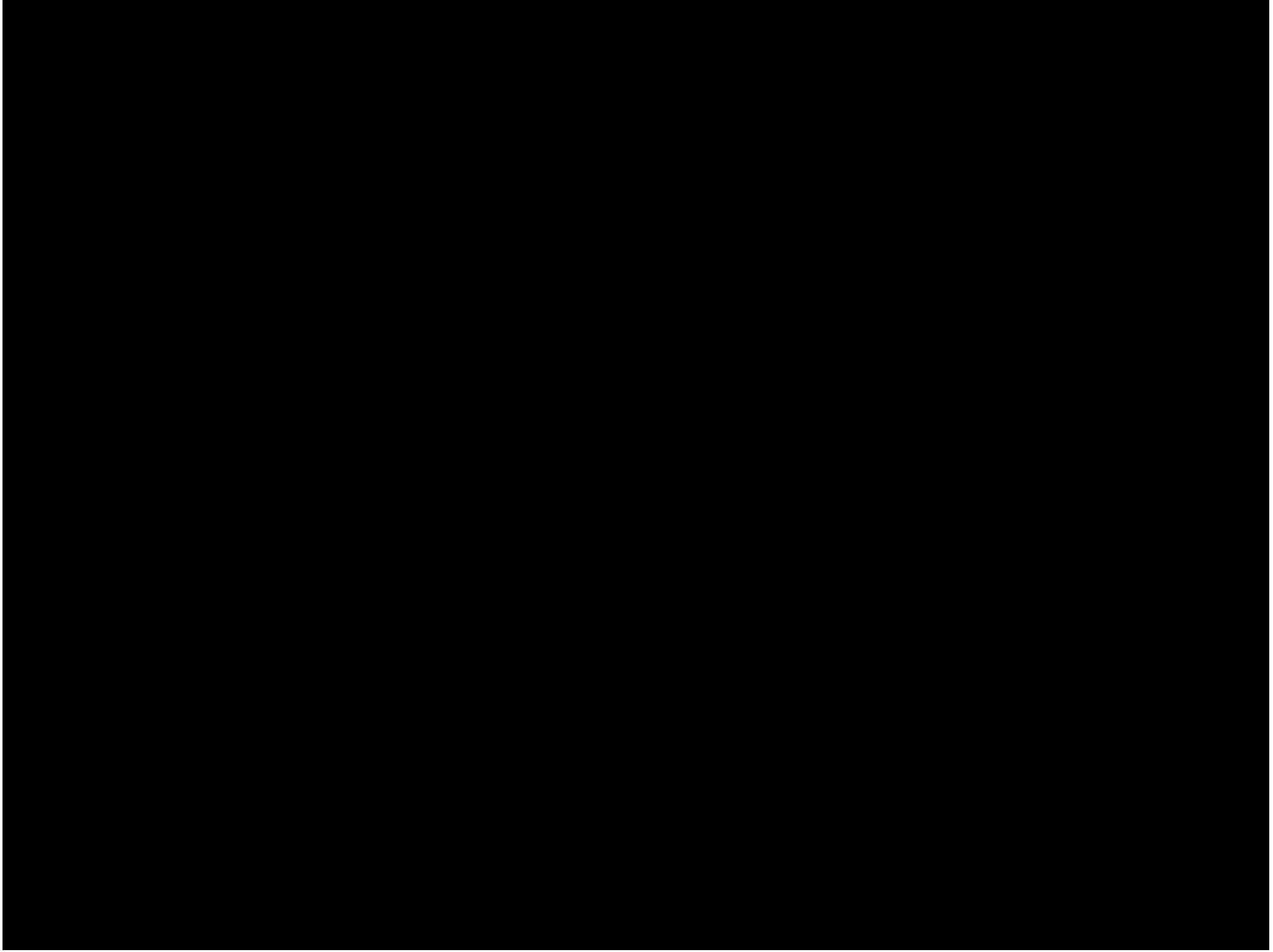
$= -23 + 60$

$= 37$

$e = 13$, $d = 37$

$c = p^e \bmod n \Rightarrow c = 7^{13} \bmod 77$

$p = c^d \bmod n \Rightarrow p = (35)^{37} \bmod 77$



Unit - V

B Ramesh
Page No. :

Probabilistic and Randomized Algorithms

Deterministic and Non-deterministic algorithms

Deterministic Algorithm –

Definition

The algorithms in which the result of every algorithm is uniquely defined are known as the Deterministic algorithm.

In other words we can say that the deterministic algorithm is the algorithm that performs fixed number of steps and always get finished with accept state with the same result

Execution

Deterministic Algorithms execution, the target machine executes the same instruction and results same outcome which is not dependent on process in which instruction get executed.

Type

On the basis of execution and outcome in case of Deterministic algorithm, they are also classified as reliable algorithms as for a particular input instructions the machine will give always the same output

Execution time
Deterministic algorithms takes polynomial time for their execution

Execution path
In deterministic algorithm the path of execution for algorithm is same in every execution.

Non-deterministic Algorithm –
Definition
The algorithms in which the result of every algorithm is not uniquely defined and result could be random are known as Non-deterministic algorithm.

Execution
Non-deterministic Algorithms, the machine executing each operation is allowed to choose any one of these outcomes, subject to a determination condition to be defined later.

B.Ramesh
Page No. :
Date : / /

Type

Non-deterministic algorithm are classified as non-reliable algorithms for a particular input the machine will give different output on different executions.

Execution time

Non-deterministic algorithm could not get executed in polynomial time

Execution path

Non-deterministic algorithm the path of execution is not same for algorithm in every execution and could take any random path for its execution

Concept of NP hard and NP complete —

NP Problem :

The NP problems set of problems whose solutions are hard to find but easy to verify and are solved by Non-Deterministic Machine in polynomial time

NP-Hard Problem :

Any decision problem Pi is called NP-Hard if and only if every problem of NP (say P<subj) is reducible to Pi in polynomial time.

NP-Complete Problem :

Any problem is NP complete if it is a part of both NP and NP-Hard Problem.

Difference between NP-Hard and NP-complete

NP - hard	NP - Complete
* NP-Hard problems (say X) can be solved if and only if there is a NP-complete problem (say Y) can be reducible into X in polynomial time.	NP - Complete problems can be solved by deterministic algorithm in polynomial time
* To solve this problem it must be a NP problem	To solve this problem, it must be both NP and NP-hard problem
* It is not a decision problem	It is exclusively decision problem
* Example - Halting problem Vertex cover problem Circuit satisfiability problem etc.	Example - Determine whether a graph has a Hamiltonian cycle, Determine whether a Boolean formula is satisfiable or not etc.

Traveling salesman problem Example -

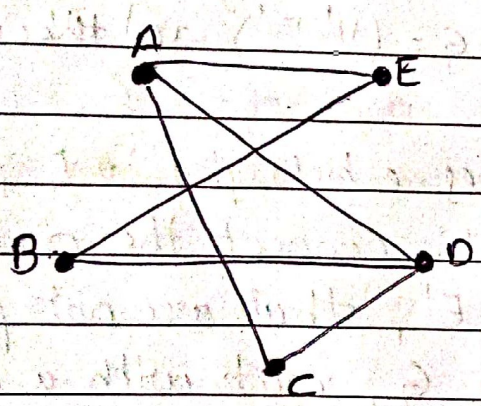
In order to prove the Travelling Salesman Problem is NP-Hard, we will have to reduce a known NP-Hard problem to this problem. We will carry out a reduction from Hamiltonian Cycle problem to the Travelling Salesman problem.

Every instance of the Hamiltonian Cycle problem consist of a graph $G = (V, E)$ as the input, can be converted to a Travelling salesman problem consisting of Graph $G = (V', E')$ and the maximum cost K.

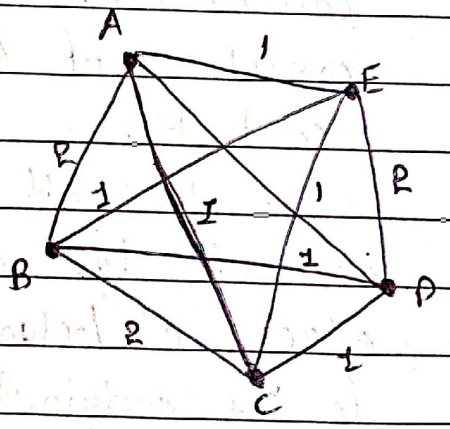
For all the edges e belonging to E, add the cost of edge $c(e) = 1$. Connect the remaining edges, e' belonging to E', that are not present in the original graph G, each with a cost $c(e') = 2$. And set $K = N$

Let us assume that the graph G contains a Hamiltonian Cycle, traversing all the vertices V of the graph. Now, these Vertices from TSP with cost = N since it uses all the edges of the original graph having cost $c(e) = 1$. And since it is a cycle, therefore it returns back to the original vertex.

We assume that the graph G' contains a TSP with cost, K = N. The TSP traverses all the vertices of the graph returning to the original vertex. Now since none of the vertices are excluded from the graph and cost sums to n, therefore, necessarily it uses all the edges of the graph present in E, with cost 1, hence forming a hamiltonian cycle with graph G



G = Hamiltonian cycle (EACDBE)



G' = TSP (EACDBE)
cost = 5 (= n)

Thus we can say that the graph G' contains TSP if graph G contains Hamiltonian cycle. Therefore any instance of Travelling Salesman problem can be reduced to an instance of the hamiltonian cycle problem. Thus, the TSP is NP-Hard.

Probabilistic algorithms -

Most of the probabilistic algorithms can make a random choice from a set of elements. For example, given integers m and n $m \leq n$, we assume that a random number generator function Random(m,n) returns a random Integer between m and n with each Integer equally likely to be chosen.

The performance of a probabilistic algorithm depends not only on the input to the algorithm but also on the result of randomized choices. Because running the algorithm twice with the same input I can result in a different number of basic operation being performed.

If the algorithm performs many random choices, than even for a single run we can expect the number of basic operation performed to be very close to Exp(I). Analogous to the functions B(n), W(n) and A(n) for the best case, worst case and average complexities of a deterministic algorithm.

We now have $B_{exp}(n)$, $W_{exp}(n)$, $A_{exp}(n)$ for the expected best-case, worst-case and average complexities of a probabilistic algorithm. These are defined by

B.Ramesh
Page No. :
Date : / /

$$B_{exp}(n) = \min\{T_{exp}(I) \mid I \in \phi_n\}$$

$$W_{exp}(n) = \max\{T_{exp}(I) \mid I \in \phi_n\}$$

$$A_{exp}(n) = E(I_{exp})$$

where ϕ_n denotes the set of all inputs of size n to the algorithm.

Randomizing Deterministic Algorithms –

Randomised algorithm arise from the deterministic algorithms by introducing randomness in some of the steps of algorithm. Randomized versions of deterministic algorithms tend to cause the expected behaviour of each input to approach average behaviour of the deterministic algorithm so that $A_{exp}(n)$ is approximately equal to $A(n)$.

Thus, a randomized version of a deterministic algorithm has the most potential of being useful when the average complexity $A(n)$ of the deterministic algorithm is significantly smaller than the worst-case complexity $W(n)$. Then a suitable randomization usually results in $W_{exp}(n)$. That is also significantly smaller than $W(n)$.

The procedure Quicksort is good example of when randomization is useful. In particular, the randomized version of Quicksort satisfy the Ultimate homogenization condition;

$$B_{exp}(n) = A_{exp}(n) = W_{exp}(n)$$

Randomizing Linear Search –

A simple randomization yields $B_{exp}(n) = A_{exp}(n) = W_{exp}(n) = (n+1)/2$. We merely flip a fair coin once at the beginning to decide whether we can scan left to right or from right to left. To verify that $T_{exp}(I) = (n+1)/2$ for any input $I \in \mathcal{I}_n$, Let $L[0:n-1]$ be any input list and let k be the index in L of the occurrence of X. With probability $1/2$, the scan will be from left to right, resulting in k comparisons; and with probability $1/2$ the scan will be from right to left, resulting $n-k+1$ comparisons. Thus, the expected number of comparisons performed by the algorithm to find X is given by

$$1/2\,(k) + 1/2\,(n-k+1) = (n+1)/2$$

since L and X were any input I, we have

$$T_{exp}(I) = \frac{n+1}{2}, \quad I \in \mathcal{I}_n$$

Randomizing PROBLINSRCH

ProbeLINSRCH for searching a list $L[0:n-1]$ that was maintained in sorted order using an auxiliary array $Link[0:n-1]$. Thus, the sorted order of the list L is given by $L[Head], L[Link[Head]], L[Link[Link[Head]]], \ldots, L[Link^{n-1}[Head]]$, where $Link^m$ denotes the m-fold composition of Link. $Link[i] = -1$ indicates the end of the list.

Given a search element X, Procedure ProbeLinSrch begins by determining the best place to start a link ordered linear search from the first $\sqrt{n}$ elements in $L[0:n-1]$. ProbeLinSRCH has $\theta(n)$ worst-case complexity but $\theta(\sqrt{n})$ average complexity, so it is good candidate for randomization.

The average behavior of ProbeLinSrch also establishes that the randomized versions has $Texp(I) \in \theta(\sqrt{n})$ for each

$I = (L[0:n-1], X)$ $X = L(0), L[1], \ldots L[n-1]$

RANDOMIZING QUICKSORT -

One way to randomize Quicksort is to make a random choice of a pivot element for the call to Partition. The latter method of randomization is an example of the general randomization technique known as stochastic preconditioning. Both methods of randomizing Quicksort accomplish the same thi that is the randomized version of Quicksort perform the same expected number of comparisons for any input list of size n.

We now design an algorithm Randomized Quicksort based on stochastic preconditioning Given an input list I = L[0 : n-1].

```
procedure Permute (L[0:n-1])
Input : L[0:n-1] (an array of list elements)
Output : L[0:n-1] (an array of list elements
                   randomly permuted)
    for i ← 0 to n-2 do
        j ← Random (i, n-1)
        interchange (L[i], L[j])
    endfor
end Permute
```

MONTE CARLO AND LAS VEGAS ALGORITHMS

A Monte carlo algorithm is a probabilistic algorithm that hay a certain probability of returning the correct answer whatever input is considered. On the other hand, a Las Vegas algorithm never returns an incorrect answer, but it might not return an answer at all.

The most useful class of Monte carlo algorithms are those that have a probability of returning the correct answer greter than some fixed positive constant for any input.

Specifically, for a fixed real numbers p, $0 < p < 1$, a p-correct Monte carlo algorithm is a probabilistic algorithm that returns the correct answer with probability not less than p, no matter what input is considered.

Unfortunetly, many times there is no efficient method available to test whether an answer returned by a Monte carlo algorithm for a given input is correct

BIASED MONTE CARLO ALGORITHM

A Monte Carlo algorithm for a decision problem is false biased if it is always correct when it returns the value false, and only has some probability of making a mistake when returning the value true. A similar definition holds for the true-biased-Monte Carlo algorithms. Fundamental to the applicability of Monte Carlo algorithm is the fact that the probability of returning the correct output increases with repeated trials. For Example given a false biased Monte Carlo algorithm MC, consider the following algorithm MCRepeat.

```
function MCRepeat(k)
Input: k (a positive integer)
Output: false if MC returns false for any invocation
        true otherwise
for i ← 1 to k do
    if MC returns false then
        return (false)
    end if
endfor
return (true)
end MCrepeat
```

A MONTE CARLO ALGORITHM OF TESTING POLYNOMIAL EQUALITY

Given symbolic polynomials $f(x)$, $g(x)$ and $h(x)$ of degree $2n$, n, and n, respectively. We consider the problem of testing whether $f(x)$ is identically equal to $g(x)h(x)$, written $f(x) \equiv g(x)h(x)$. Thus two distinct polynomials of degree $2n$ could not agree at more than $2n$ integers drawn from $[1, 2, \ldots, 4n]$. The following algorithm is therefore a false biased $(1/2)$-correct Monte Carlo algorithm for testing whether $f(x) \equiv g(x)h(x)$

```
function TestPolyEqual (f(x), g(x), h(x))
Input : f(x), g(x), h(x)
Output : true or false (always correct when false
         is output, and correct at least 50 percent
         of the time when output is true)
j ← Random (1, 4*n)
if f(j) = g(j) * h(j) then
    return (true)
else
    return (false)
endif
end TestPolyEqual
```

UNIT - 1

B.Ramesh
Page No. :
Date : / /

1 Explain different methods of solving recurrence relation.

→ There are mainly 4 ways for solving recurrence

i) Substitution Method :
We make a guess for the solution and then we use mathematical induction to prove the guess is correct or incorrect.

ii) Iteration Methods :
It means to expand the recurrences and express it as a summation of terms of n and initial condition.

iii) Recurrence Tree Method :
In this Method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally sum the work done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

iv Master Method:

Master method is a direct way to get the solution. The master method works only for following type of recurrence or for recurrence that can be transformed to following type.

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

8 Discuss the Potential approach of amortized analysis method.

→ The Potential Method:

* It is same as accounting method: something prepaid is used later.
* The prepaid work not as credit, but as "potential energy".
* The potential is associated with data structures as a whole rather than specific object within the data structure.
* Amortized analysis is tool for analyzing algorithm that perform a sequence of similar operations.
* An amortized analysis can be used to provide a bound on the actual cost of the entire sequence.

* Initial data structure D_0
* n operations, resulting in $D_0, D_1, \ldots, D_n$ with costs $c_1, c_2, \ldots, c_n$
* A potential function $\phi : \{D_i\} \forall \phi(D_i)$ is called the potential of D_i
* Amortized cost $\hat{c}_i$ of the i^{th} operation is $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$.

3 Discuss the aggregate analysis for method for amortized analysis

→ The aggregate analysis Method:

* In aggregate analysis, we show that for all n, a sequence of n operations takes worst-case time $T(n)$ in total.
* In the worst case, the amortized cost, per operation is $T(n)/n$
* Note that this amortized cost applies to each operation. Even when there are several types of operations in the sequence
* The amortized cost of an operation is $O(n)/n = O(1)$
* Amortized cost in aggregate analysis is defined to average cost.

4 Write a note on amortized analysis.

→ In an amortized analysis the time required to perform a sequence of data-structure operations is averaged over all the operations performed. It can be used to show that the average cost of an operation is small, even though a single operation within the sequence might be expensive.

Amortized analysis differes from average case-analysis in that probability is not involved. It guranteeg the average performance of each operation in the worst case.

The three most common technique used in amortized analysis starts with aggregate analysis. We determine an upper bound $T(n)$ on the total cost of a sequence of n operations.

5 Explain the three basic asymtotic notations with examples for each.

O - notation —

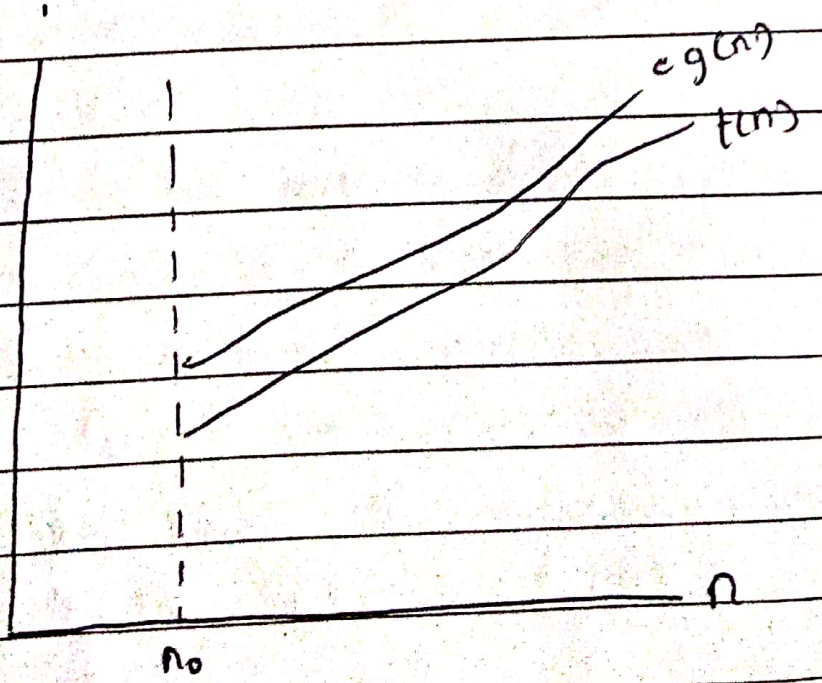
A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n, i.e if there exist some positive constant c and some non negative integer n_0 such that

$$t(n) \leq cg(n) \text{ for all } n \geq n_0$$

Example — $100n + 5 \in O(n^2)$

$100n + 5 \leq 100n + n$ (for all $n \geq 5$)

$101n \leq 101n^2$



Big-oh notation $t(n) \in O(g(n))$

Ω - notation –

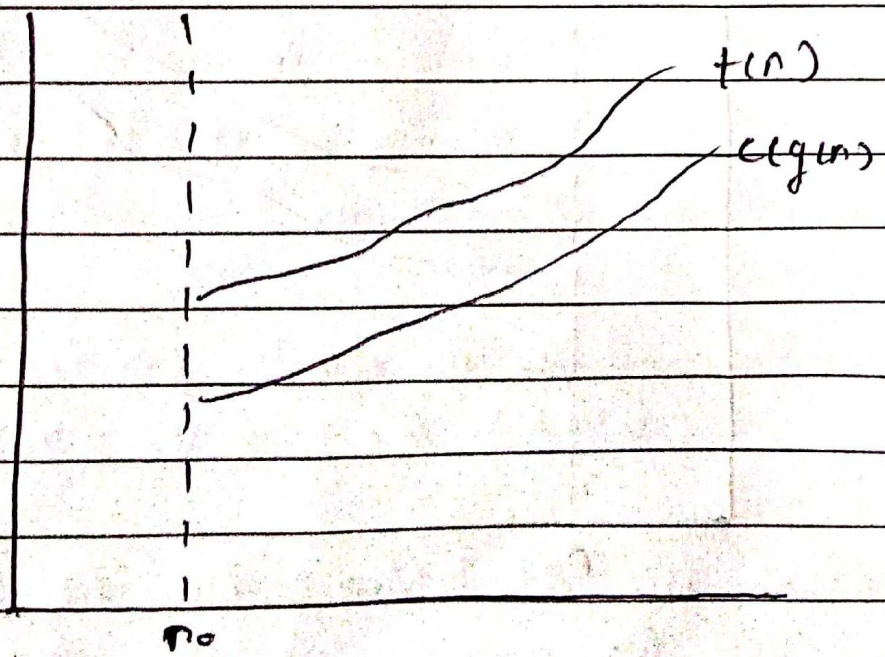
A function $f(n)$ is said to be $\Omega(g(n))$, denoted $f(n) \in \Omega(g(n))$, if $f(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n, i.e, if there exist some positive constant c and some non negative integer n_0 such that

$$f(n) \geq cg(n) \text{ for all } n \geq n_0$$

Example – $n^3 \in \Omega(n^2)$

$n^3 \geq n^2$ for all $n \geq 0$

i.e we can select $c = 1$ and $n_0 = 0$



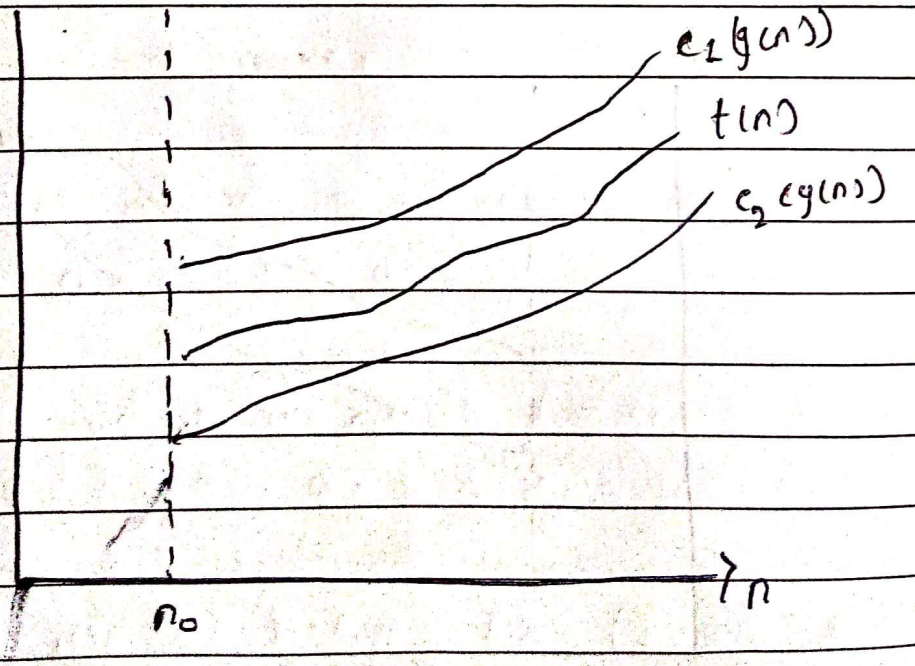
Big-Omega notation : $f(n) \in \Omega(g(n))$

Θ - notation

A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiple of $g(n)$ for all large n, i.e if there exist some positive constant c_1 and c_2 and some nonnegative integer n_0 such that

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \text{ for all } n \geq n_0$$

Example - $\frac{1}{2} n(n-1) \in \Theta(n^2)$



Big - theta notation : $t(n) \in \Theta(g(n))$

Unit-3

B.Ramesh
Page No. :
Date : / /

GCD

GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common factors

$36 = 2 \times 2 \times 3 \times 3$

$60 = 2 \times 2 \times 3 \times 5$

GCD = Multiplication of common factors

$= 2 \times 2 \times 3$

$= 12$

Basic Euclidean Algorithm for GCD

* If we subtract smaller number from larger, GCD doesn't change. So if we keep subtracting repeatedly the larger two, we end up with GCD

* Now instead of subtraction, if we divide smaler number, the algorithm stops when we find reminder 0.

$$gcd(a,b) = gcd(b, a \bmod b)$$

Extended Euclidean Algorithm :

It also finds integer coefficients x and y such that :

$$ax + by = gcd(a, b)$$

The extended Euclidean algorithm updates results of $gcd(a,b)$ using results calculated by recursive call $gcd(b\%a, a)$. Let values of x and y calculated by the recursive call be x_1 and y_1. x and y are updated using below expressions.

$$x = y_1 - \lfloor b/a \rfloor * x_1$$
$$y = x_1$$

As seen above, x and y results for inputs a and b

$$ax + by = gcd$$

And x_1 and y_1 are results for inputs $b\%a$ and a

$$(b\%a) \cdot x_1 + a \cdot y_1 = gcd$$

When we put $b\%a = (b - (\lfloor b/a \rfloor) \cdot a)$ in above we get following.

$$(b - (\lfloor b/a \rfloor) \cdot a) \cdot x_1 + a \cdot y_1 = gcd$$

After comparing coefficients of 'a' and 'b' in (1) and (2), we get following

$$x = y_1 - \lfloor b/a \rfloor * x_1 \quad \Rightarrow \quad y = x_1$$

Fermat's little theorem:

Fermat's little theorem states that if p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p

Here p is a prime number

$$a^p \equiv a \pmod{p}$$

Special case: If a is not divisible by p, Fermat's little theorem is equivalent to the statement that $a^{p-1} - 1$ is an integer multiple of p $\quad a^{p-1} \equiv 1 \pmod{p}$

Example: p = an integer prime number

a = an integer which is not multiple of p

Let a = 2 and p = 17

According to Fermat's little theorem

$$2^{17-1} \equiv 1 \bmod (17)$$

we get $65536 \,\%\, 17 \equiv 1$

that mean $(65534 - 1)$ is an multiple of 17

B.Ramesh
Page No.:
Date: / /

If n is a prime number, then for every a, $1 < a < n-1$,

$$a^{n-1} \equiv 1 \pmod{n}$$

Example:

Since 5 is prime, $2^4 \equiv 1 \pmod{5}$

$3^4 \equiv 1 \pmod{5}$

$4^4 \equiv 1 \pmod{5}$

Since 7 is prime, $2^6 \equiv 1 \pmod{7}$

$3^6 \equiv 1 \pmod{7}$

$4^6 \equiv 1 \pmod{7}$

$5^6 \equiv 1 \pmod{7}$

$6^6 \equiv 1 \pmod{7}$

Procedural steps for RSA algorithm

Step-1 : Choose two prime numbers p and q

Step-2 : Compute the value of n and ϕ

$n = p \times q$

$\phi = (p-1) \times (q-1)$

Step-3 : Find the value of e (public key)

Step-4 : Compute the value of d (private key)

Step-5 : Do the encryption and decryption

Encryption given as $c = t^e \bmod n$

Decryption given as $t = c^d \bmod n$