# Design and Analysis of Algorithms

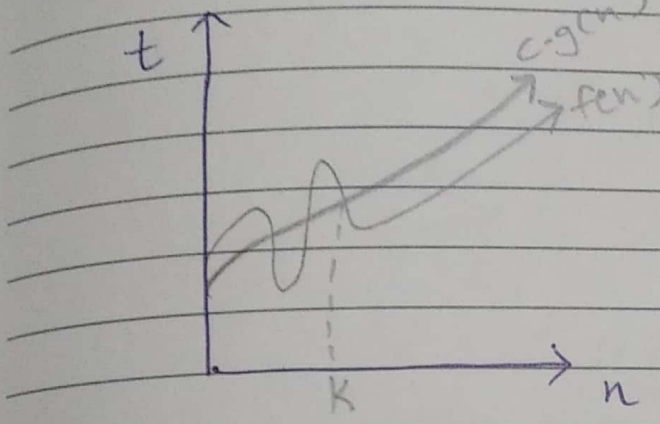
WEDNESDAY
AUGUST 2020
12

## * Algorithm :-

→ Finite set of steps to solve a problem is called Algorithm.

## * Asymptotic Notations :-

### 1) Big - Oh ( O )




→ Worst case

→ Upper Bound (Least)

→ $f(n) = O(g(n))$

$f(n) \leq c \cdot g(n)$ ; $c > 0$, $n \geq k$, $k \geq 0$

→ Ex: $f(n) = 2n^2 + n$

$\therefore f(n) = O(n^2)$

$\therefore 2n^2 + n \leq c \cdot g(n^2)$

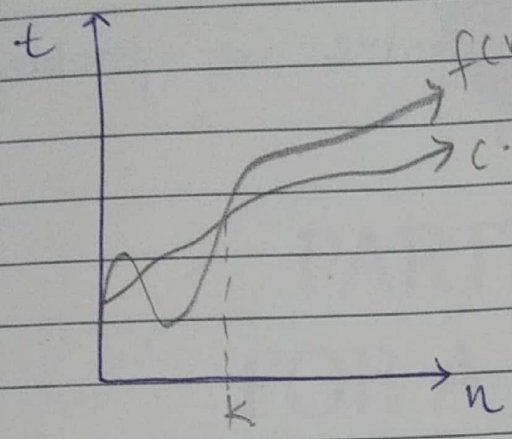
$\therefore 2n^2 + n \leq 3n^2$ ; $c = 3$ → Least Upper Bound.

$\therefore n \leq n^2$

$\therefore 1 \leq n$

$\therefore \boxed{n \geq 1}$

## 2) Big - Omega ( Ω )




→ Best Case

→ Greatest / Closest lower bound.

→ $f(n) = \Omega(g(n))$

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

→ Ex:. $f(n) = 2n^2 + n$.

→ $f(n) = \Omega(g(n))$

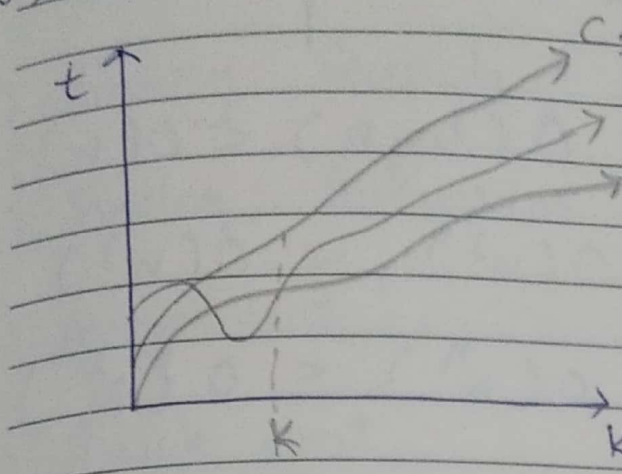
$\therefore f(n) \geq c \cdot g(n)$

$\therefore 2n^2 + n \geq c \cdot n^2$. ← closest value from f(n).

$\therefore 2n^2 + n \geq 2 \cdot n^2 \; ; c = 2$

$\therefore \boxed{n \geq 0}$

3) Theta ( $\theta$ )




→ Average case

→ Exact time.

→ $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

→ Ex:- $f(n) = 2n^2 + n$

$\therefore c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$\therefore 2 \cdot n^2 \leq 2n^2 + n \leq 3 \cdot n^2$

* Properties :-

| | Reflexive | Symmetric | Transitive |
|---|---|---|---|
| Big Oh (O) : $f(n) \leq c \cdot g(n)$ | ✓ | X | ✓ |
| Big Omega ($\Omega$) : $f(n) \geq c \cdot g(n)$ | ✓ | X | ✓ |
| Theta ($\theta$) : $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ | ✓ | ✓ | ✓ |
| Small oh (o) : $f(n) < c \cdot g(n)$ | X | X | ✓ |
| Small Omega ($\omega$) : $f(n) > c \cdot g(n)$ | X | X | ✓ |

15 SATURDAY AUGUST 2020

* Comparison of various Time Complexities :-

$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2})$
$< O(n) < O(n \cdot \log n) < O(n^2) < O(n^3)$
$< O(n^K) < O(2^n) < O(n^n) < O(2^{2^n})$

* Formulas for analyzing algorithms :-

for ( i=0 ; i<n ; i++ ) n → O(n)

for ( i=0 ; i<n ; i = i+2 ) n/2 → O(n)

for ( i = n ; i>1 ; i-- ) n → O(n)

for ( i = 1 ; i<n ; i = i*2 ) → $O(\log_2 n)$

for ( i=1 ; i<n ; i = i*3 ) → $O(\log_3 n)$

16 SUNDAY for ( i=n ; i>1 ; i = i/2 ) → $O(\log_2 n)$

* Divide and Conquer :-

1) Binary Search

2) Finding Minimum and Maximum

3) Merge sort

4) Quick sort

5) Strassen's Matrix Multiplication.

# * Recurrence Relation :- (Decreasing functions)

(1) $T(n)$ ⟶

```
void Test (int n)
{
    if (n > 0)
    {
        printf ("%d", n);      // 1 ⟶
        Test (n-1);            // T(n-1) ⟶
    }
}
```

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

∴ From the condition $n > 0$,

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

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

∴ Substituting $T(n-1) = T(n-2) + 1$

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

$T(n-3) = T(n-4) + 1$

~~$T(n-k) = T$~~

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

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

$\therefore T(n) = [T(n-4) + 1] + 3$

Assume that it will continue for k times,

$\therefore T(n) = T(n-k) + k$ ——— (★)

→ We know that, if $n = 0 \Rightarrow T(n) = 1$.

∴ Assume $n - k = 0$ (∵ From ★)

$\therefore n = k$.

∴ Now, from ★,

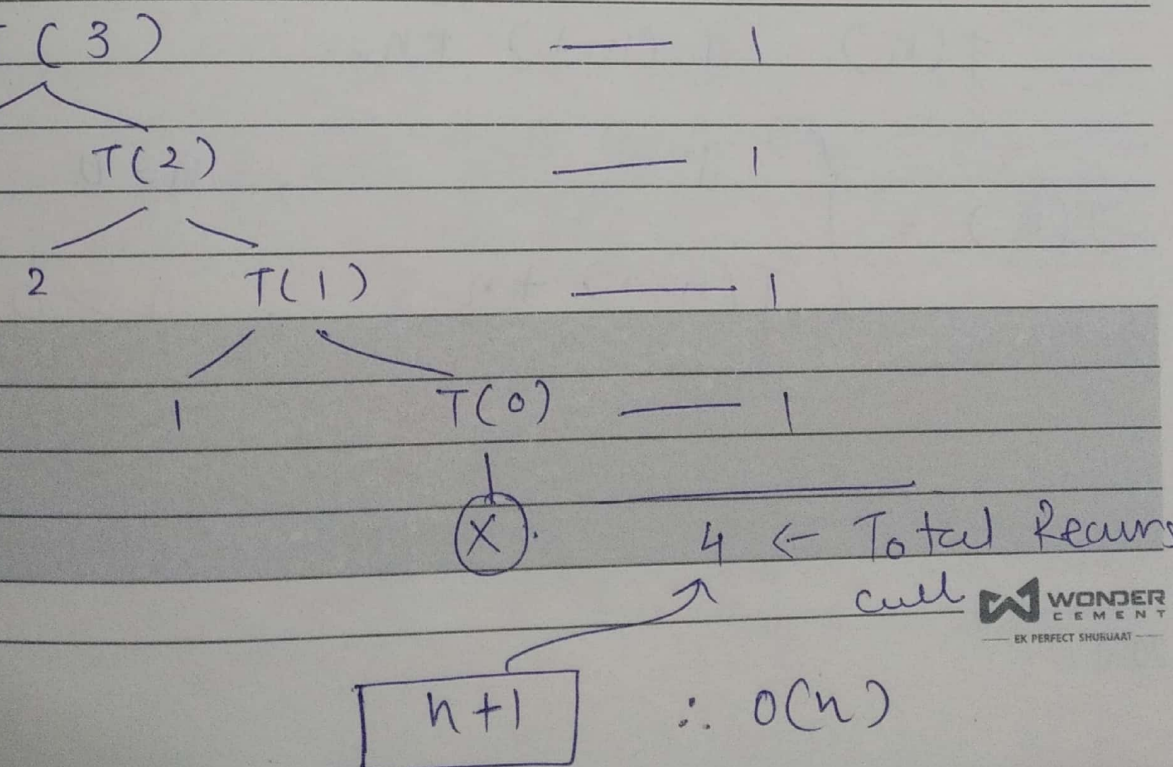
$T(n) = T(n-n) + n$

$\therefore T(n) = T(0) + n$

$\therefore \boxed{T(n) = 1 + n}$ → Answer of Recurrence Relation.

$\therefore O(n)$.

→ By creating recursive tree, $(n = 3)$




$\boxed{n+1}$ $\therefore O(n)$

(2)

| | |
|---|---|
| $T(n)$ | ⟶ |
| $1$ | ⟶ |
| $n+1$ | ⟶ |
| $n$ | ⟶ |
| $T(n-1)$ | ⟶ |

```
void Test( int n)
{
    if (n>0)
    {
        for(int i=0; i<n; i++)
        {
            printf ("%d", n);
        }
        Test (n-1);
    }
}
```

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

To make it easy to solve,

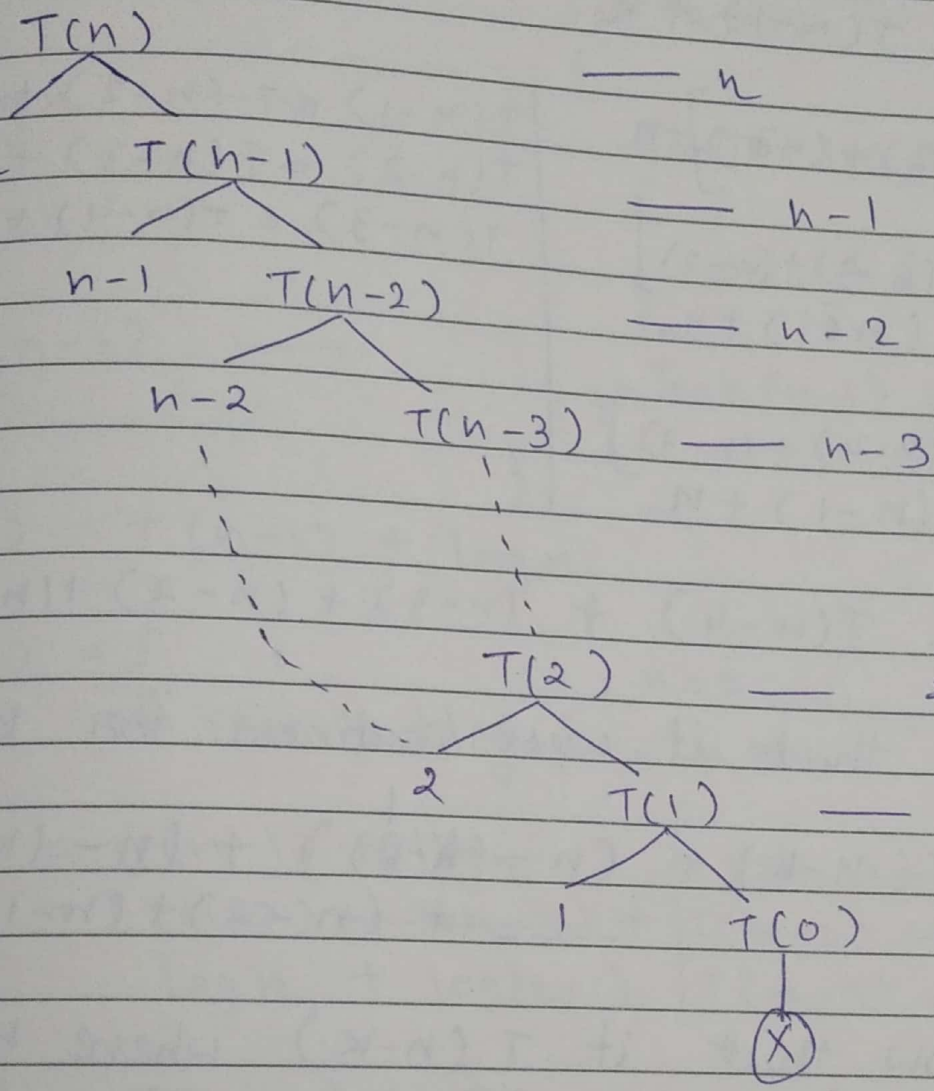
$$T(n) = T(n-1) + \underbrace{2n+2}$$

we take least upper bound of this.

$\therefore T(n) = T(n-1) + n.$

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

* Solving using recursive tree :-




$0 + 1 + 2 + \dots + (n-1) + n$

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

$\therefore O(n^2)$

** Solving using back-substitution method.

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

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

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

$\therefore T(n) = [T(n-4) + (n-3)] + (n-2) + (n-1) + n$

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

$\therefore T(n) = T(n-4) + (n-3) + (n-2) + (n-1) + n$

Assuming that it will continue for k times,

$T(n) = T(n-k) + (n-(k-1)) + (n-(k-2)) + \dots + (n-2) + (n-1) + n$.

We know that if $T(n-k)$ where $n-k=0$ then $a_1 = 1$

$\therefore$ Assume $n-k = 0$

23 SUNDAY $\therefore n = k$.

$\therefore T(n) = T(n-n) + (n-(n-1)) + (n-(n-2)) \dots + (n-1) + n$.

$\therefore T(n) = T(0) + 1 + 2 + \dots + (n-1) + n$.

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

$\therefore O(n^2)$

Here, we are getting 1 extra as this method gives no. of calls.

(3)

$T(n) \longrightarrow$

$\log n \longrightarrow$

$T(n-1) \longrightarrow$

```
void Test (int n)
{
    if (n>0)
    {
        for(int i=0; i<n; i=i*2)
        {
            printf ("%d", n);
        }
        Test (n-1);
    }
}
```

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

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

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

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

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

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

$T(n-3) = T(n-4) + \log(n-3)$

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

Assume that it will continue for k times,

$\therefore T(n) = T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \dots + \log n.$

Assume $n - k = 0$

$\therefore n = k.$

$\therefore T(n) = T(0) + \log(n-n+1) + \log(n-n+2) + \dots + \log n.$

$= 1 + \log 1 + \log 2 + \dots + \log n$

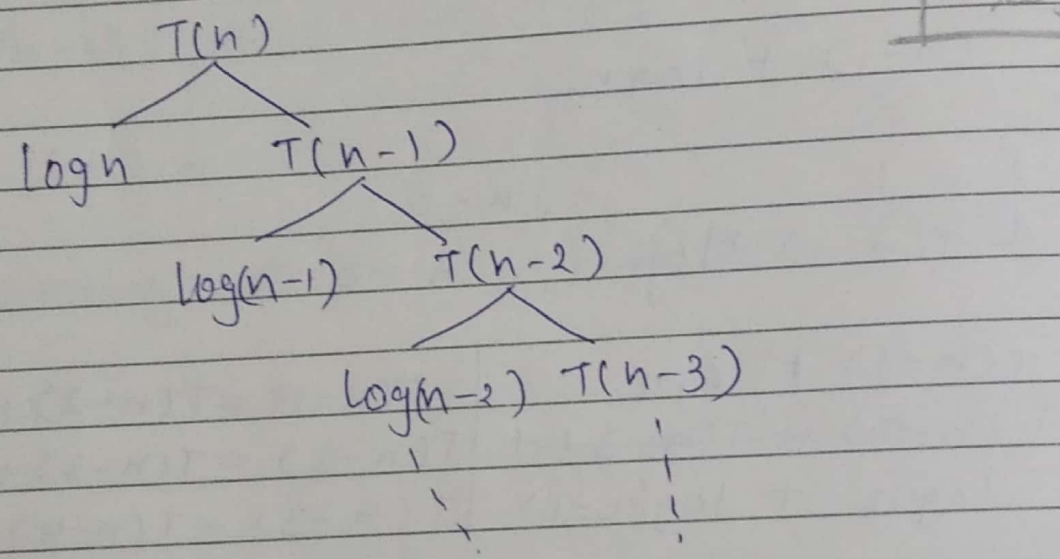
$= 1 + \log n!$

$\therefore O(n \cdot \log n)$

For $n!$ upper bound is $n^n = n \cdot \log n$

$\therefore$ For $\log n!$ upper bound is $\boxed{n \cdot \log n!}$

** Recursive tree :-




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

$= \log[n \times (n-1) \times (n-2) \times \dots \times 1]$

$= \log n!$ $\therefore O(n \cdot \log n)$

* Direct method to get answer of decreasing recurrence relation :- JUST MULTIPLY REGARDLESS OF CONSTANTS

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

$T(n) = T(n-1) + n \Rightarrow O(n^2)$

$T(n) = T(n-1) + \log n \Rightarrow O(n \cdot \log n)$

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

$T(n) = T(n-2) + 1 \Rightarrow O(n)$

$T(n) = T(n-100) + n \Rightarrow O(n^2)$

There should not be any co-efficient such as $T(n) = 2 \cdot T(n-100) + n$.

$T(n) = 2 \cdot T(n-1) + 1 \Rightarrow O(2^n)$

$T(n) = 3 \cdot T(n-1) + 1 \Rightarrow O(3^n)$

$T(n) = 2 \cdot T(n-1) + n \Rightarrow O(n \cdot 2^n)$

$T(n) = 3 \cdot T(n-2) + n \Rightarrow O(n \cdot 3^{n/2})$

(4)

```
Algorithm Test(int n)
{
  if(n>0)
  {
    printf("%d", n);
    Test(n-1);
    Test(n-1);
  }
}
```

$T(n) \rightarrow$

$1 \rightarrow$

$T(n-1) \rightarrow$

$T(n-1) \rightarrow$

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

$\therefore T(n) = 2 \cdot T(n-1) + 1$

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

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

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

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

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

$= 2^3 \cdot T(n-3) + 4 + 2 + 1$

$= 2^3[2 \cdot T(n-4)+1] + 4 + 2 + 1$

$\therefore T(n) = 2^4 T(n-4) + 8 + 4 + 2 + 1$

$\therefore T(n) = 2^4 \cdot T(n-4) + 2^3 + 2^2 + 2^1 + 2^0$

Assuming it will continue for k times,

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

$= 2^k \cdot T(n-k) + 2^{k-1} + 2^{k-2} + 2^{k-3} + \ldots + 2^0$

Assume $n-k=0 \quad \therefore n=k$

$\therefore T(n) = 2^n \cdot T(0) + 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2^1 + 2^0$

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

$= 1 + 2 + 4 + \ldots + 2^{n-2} + 2^{n-1} + 2^n$

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

$\therefore O(2^n)$

* GP Series :-

$a + ar + ar^2 + ar^3 + \ldots + ar^k = \frac{a \cdot (r^{k+1} - 1)}{r-1}$

(4) * Master Theorem for Decreasing Functions:

→ $T(n) = a \cdot T(n-b) + f(n)$

$a > 0$, $b > 0$ and $f(n) = O(n^k)$; $k \geq 0$

if $a = 1$ → $O(n \cdot f(n)) = O(n \cdot n^k)$

$= \boxed{O(n^{k+1})}$

if $a > 1$ → $O(f(n) \cdot a^{n/b})$

$= \boxed{O(n^k \cdot a^{n/b})}$

if $a < 1$ → $O(f(n))$

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

# * Recurrence Relation for Dividing Functions :-

① $T(n) \longrightarrow$

```
Algorithm Test(int n)
{
    if(n>1)
    {
        printf("%d", n);
        Test(n/2);
    }
}
```

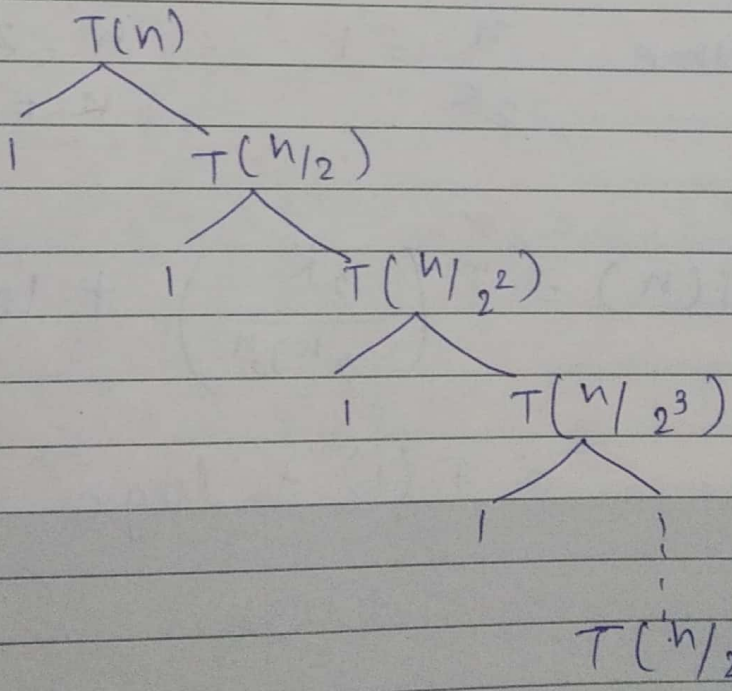
$1 \longrightarrow$

$T(n/2) \longrightarrow$

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

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

* Recurrence Tree :-




→ $n/2^K = 1$

$\therefore n = 2^K$

$\therefore K = \log_2 n$

$\therefore O(\log n)$

## Substitution method:

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

$\therefore T(n) = T(n/2) + 1$

$\therefore T(n) = T(n/4) + 1 + 1$

$\therefore T(n) = T(n/8) + 1 + 1 + 1$

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

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

$T(n/4) = T(n/8) + 1$

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

Assume $\frac{n}{2^k} = 1$ $\quad \therefore n = 2^k$ —— (1)

$\therefore k = \log_2 n$ —— (2)

$\therefore T(n) = T\left(\frac{2^k}{2^{\log_2 n}}\right) + \log_2 n$

$= T(1) + \log n$

$= 1 + \log n$

$\therefore O(\log n)$

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

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

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

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

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

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

$\therefore T(n) = T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} + \dots + \frac{n}{2} + \frac{n}{2^0}$

Assume, $\frac{n}{2^k} = 1 \quad \therefore n = 2^k$

$\therefore k = \log_2 n$

$\therefore T(n) = T(1) + n\left[\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \dots + \frac{1}{2^1} + \frac{1}{2^0}\right]$

$= 1 + n\left[\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \dots + \frac{1}{2} + 1\right]$

$= 1 + n[1+1]$ ← approximate value is 1

$= 1 + 2n$

$\therefore O(n)$

(4) (3) $T(n) \longrightarrow$

```
void Test(int n)
{
  if (n > 1)
  {
    for(int i=0; i<n; i++)
    {
      statement;
    }
    Test(n/2);
    Test(n/2);
  }
}
```

n →

$T(n/2) \longrightarrow$

$T(n/2) \longrightarrow$

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

$$\therefore T(n) = \begin{cases} 1 & , n = 1 \\ 2T\left(\frac{n}{2}\right) + n & , n > 1 \end{cases}$$

$\Rightarrow T(n) = 2 \cdot T(n/2) + n$

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

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

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

$$\therefore T(n) = 2 \cdot \left[ 2 \cdot T\left(\frac{n}{2^2}\right) + \frac{n}{2} \right] + n.$$

$$= 2^2 \cdot T\left(n/_{2^2}\right) + n + n$$

$$= 2^2 \cdot \left[ 2 \cdot T\left(n/_{2^3}\right) + n/_{2^2} \right] + 2n.$$

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

$$= 2^3 \left[ 2 \cdot T\left(n/_{2^4}\right) + n/_{2^3} \right] + 3n.$$

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

$$= 2^K \cdot T\left(n/_{2^K}\right) + Kn.$$

Assume, $n/_{2^K} = 1 \quad \therefore n = 2^K$

$\therefore K = \log \frac{n}{2}.$

$$\therefore T(n) = 2^{\log_2 n} \cdot T(1) + n \cdot \log_2 n$$

$$= n\,(1) + n \cdot \log_2 n$$

$$\therefore T(n) = n(1 + \log_2 n) = n + n \log n$$

$$\therefore O(n \cdot \log n)$$

# * Master Theorem for Dividing Functions :-

→ $T(n) = a \cdot T(n/b) + f(n)$ ; $a \geq 1$, $b > 1$

$f(n) = \Theta(n^k \cdot \log^p n)$

① $\log_b a$ } Find

② $k$ }

Case 1 : $\log_b a > k$ then $\Theta(n^{\log_b a})$

Case 2 : $\log_b a = k$

if $p > -1 \Rightarrow \Theta(n^k \cdot \log^{p+1} n)$

if $p = -1 \Rightarrow \Theta(n^k \log\log n)$

if $p < -1 \Rightarrow \Theta(n^k)$

06 SUNDAY Case 3 : $\log_b a < k$

if $p \geq 0 \Rightarrow \Theta(n^k \cdot \log^p n)$

if $p < 0 \Rightarrow \Theta(n^k)$

* Examples :-

(1) $T(n) = 2 \cdot T(n/2) + 1$

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

and $= (n^0 \cdot \log^0 n)$

$\log_b a = \log_2 2 = 1 \quad \therefore k = 0$

$\therefore \log_b a > k$

$\therefore \Theta(n^1)$

$\therefore \Theta(n)$

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

$a = 4, \quad b = 2, \quad f(n) = 1$

and $= (n^0 \cdot \log^0 n)$

$\log_b a = \log_2 4 = 2 \quad \therefore k = 0.$

$\therefore \log_b a > k.$

$\therefore \Theta(n^2)$

(3) $T(n) = 8 \cdot T(n/2) + n$.

$a = 8$, $b = 2$, $f(n) = n$

$\therefore \log_b a = \log_2 8$ $\quad \therefore k = 1$, $p = 0$.

$= 3$

$\therefore \log_b a > k$.

$\therefore \theta(n^3)$

(4) $T(n) = 9 \cdot T(n/3) + n^2$.

$a = 9$, $b = 3$ $\quad f(n) = n^2$

$\therefore k = 2$. and

$\log_b a = \log_3 9 = 2$ $\quad p = 0$.

$\therefore \log_b a = k$.

Here, $p = 0$ which is greater than $-1$

$\therefore p > -1$

$\therefore \theta(n^k \cdot \log^{p+1} n)$

$\therefore \theta(n^2 \cdot \log n)$

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

$a = 8, \quad b = 2 \quad , \quad k = 1 \quad , p = 1.$

$\therefore \log_b a = \log_2 8 = 3$

$\therefore \log_b a > k.$

$\therefore \Theta(n^{\log_b a}) = \Theta(n^3)$

(6) $T(n) = 4 \cdot T(n/2) + n^2.$

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

$\therefore \log_b a = \log_4 2 = 2 \qquad k = 2$

$\therefore \log_b a = k.$

$\therefore \Theta(n^2 \cdot \log n)$

when $\log_b a = k$. take $f(n)$ as it is and multiply it with $\log n$. while $p > -1$

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

$\therefore a = 4, \quad b = 2 \qquad$ and $k = 2 \quad , p = 2$

$\log_b a = \log_4 2 = 2 \qquad \therefore \log_b a = k$

$\therefore \Theta(n^2 \cdot \log^3 n)$

(8) $T(n) = 2 \cdot T(n/2) + \frac{n}{\log n}$

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

$\log_b a = \log_2 2 = 1$

$\log_b a = \log_2 2 = 1 = k.$

but $p = -1 \qquad \therefore \theta(n^k \cdot \log\log n)$

$\therefore \theta(n \log\log n)$

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

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

$\log_b a = \log_2 2 = 1$

$\therefore \log_b a = k$

but $p = -2 < -1 \qquad \therefore \theta(n^k)$

$\therefore \theta(n)$

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

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

$\therefore \log_b a = \log_2 1 = 0 < k$.

but $p = 0$ $\quad \therefore \Theta(n^k \cdot \log^p n)$

$\therefore \Theta(n^2 \cdot \log^0 n)$

$\therefore \Theta(n^2)$.

(11) $T(n) = 4 \cdot T(n/2) + \frac{n^3}{\log n}$.

$a = 4$, $b = 2$ $\quad k = 3$, $p = -1$

$\log_b a = \log_2 4 = 2 < k$.

but $p = -1 < 0$. $\quad \therefore \Theta(n^k)$

$\therefore \Theta(n^3)$.

* Root Function :-

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

→ $T(n) = T(\sqrt{n}) + 1$

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

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

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

⋮ ⋮

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

$\therefore$ Assume $n = 2^m$

$\therefore T(2^m) = T(2^{\frac{m}{2^k}}) + k$

Assume $T(2^{\frac{m}{2^k}}) = T(2^1)$

$\therefore \frac{m}{2^k} = 1$

$\therefore m = 2^k$

$\therefore k = \log_2 m$

$\therefore k = \log \log_2 n$

$\therefore \Theta(\log \log_2 n)$

$n = 2^m$

$\therefore m = \log_2 n$

13 SUNDAY

# * Strassen's Matrix Multiplication :-

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Algorithm MM(A, B, n)

{

if (n ≤ 2)

{

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$

$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$

$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$

$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

}

else

{

mid ⇒ n/2

MM($A_{11}$, $B_{11}$, n/2) + MM($A_{12}$, $B_{21}$, n/2)

MM($A_{11}$, $B_{12}$, n/2) + MM($A_{12}$, $B_{22}$, n/2)

MM($A_{21}$, $B_{11}$, n/2) + MM($A_{22}$, $B_{21}$, n/2)

MM($A_{21}$, $B_{12}$, n/2) + MM($A_{22}$, $B_{22}$, n/2)

}

}

→ $T(n) = \begin{cases} 1 & , n \leq 2 \\ 8\,T(n/2) + n^2 & , n > 2 \end{cases}$

$\therefore a = 8, \quad b = 2 \qquad k = 2, \quad p = 0$

$\therefore \log_b a = \log_2 8$

$= 3 > k$

$\therefore \Theta(n^3)$

→ In this algorithm, we are performing 8 multiplications,

Strassen found an algorithm which multiplies two matrice with 7 multiplication. which decreases time complexity a little bit.

→ Strassen's Algorithm :-

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22})\,B_{11}$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12}) \cdot B_{22}$

$U = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$

$V = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$

$\therefore C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

## * Greedy Algorithm :-

→ A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment.

→ It doesn't worry whether the current best result will bring the overall optimal result.

## * Dynamic Programming :-

→ Dynamic programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping sub problems and optimal substructure property.

→ If any problem can be divided into sub problems, which in turn are divided into smaller sub problems and if there are overlapping among these sub problems, then the solutions to these sub problems can be saved for future reference. In this way, efficiency of the CPU can be enhanced.

This method of solving a solution is referred to as dynamic programming.

* Memorization follows Top-down approach while Tabulation follows bottom-up approach.

* ~~All Pairs Shortest Path :-~~

* Greedy Techniques :-

1) knapsack problem
2) Job sequencing
3) Minimum Spanning Tree
4) Optimal Merge Pattern
5) Huffman Coding
6) Dijkstra's algorithm

→ Min cost
→ Max profit
→ Min Risk

* Knapsack Problem :- (Fractional)

→ Given positive integers $P_1, P_2, \ldots, P_n$ and $W_1, W_2, \ldots, W_n$ and $M$

where, P denotes Profit and
W denotes weights,
M denotes maximum capacity of sack.

Find $X_1, X_2, \ldots, X_n$ ; $0 \leq x_i \leq 1$

such that $\sum_{i=1}^{n} P_i X_i$ is maximized (Objective function)

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

20 SUNDAY

## Algorithm :-

for i = 1 up to n
    Calculate profit/weight

Sort ~~weight~~ objects in decreasing order of P/W ratio.

for i = 1 to n

if ( M > 0 and $W_i \leq M$ )

$M = M - W_i$;
$P = P + P_i$;

else
    break

if ( M > 0 )

$P = P + P_i ( M / W_i )$

Ex:- $M = 20$ $(P_1, P_2, P_3) = (25, 24, 15)$
$(W_1, W_2, W_3) = (18, 15, 10)$

→ Profit/weight $\Rightarrow \frac{P_1}{W_1} = \frac{25}{18} = 1.38$

$\frac{P_2}{W_2} = \frac{24}{15} = \boxed{1.6}$ maximum profit per weight.

$\frac{P_3}{W_3} = \frac{15}{10} = 1.5$

→ Soln :-

1) $x_1 = 0$, $x_2 = 1$, $x_3 = 1/2$

$\sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$

$= (18)(0) + 15(1) + 10(1/2)$

$= 0 + 15 + 5$

$= 20 \leq M$

2) $x_1 = 1/3$, $x_2 = 4/15$, $x_3 = 1$

$\sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$

$= (18)(1/3) + 15\left(\frac{4}{15}\right) + 10(1)$

$= 6 + 4 + 10.$

$= 20 \leq M$

3) $x_1 = 5/18$

$x_2 = 1$

$x_3 = 0$

$$\sum_{i=1}^{4} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$= 18\left(\frac{5}{18}\right) + 15(1) + 10(0)$$

$$= 20 \leq M$$

4) $x_1 = 0$

$x_2 = 2/3$

$x_3 = 1$

$$\sum_{i=1}^{4} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$= 18(0) + 15\left(\frac{2}{3}\right) + 10(1)$$

$$= 20 \leq M.$$

→ Therefore, we need to select those items which are providing maximum profit per weight.

→ In this example, we should prefer $x_2$, $x_3$ and $x_1$ respectively to get maximum profit.

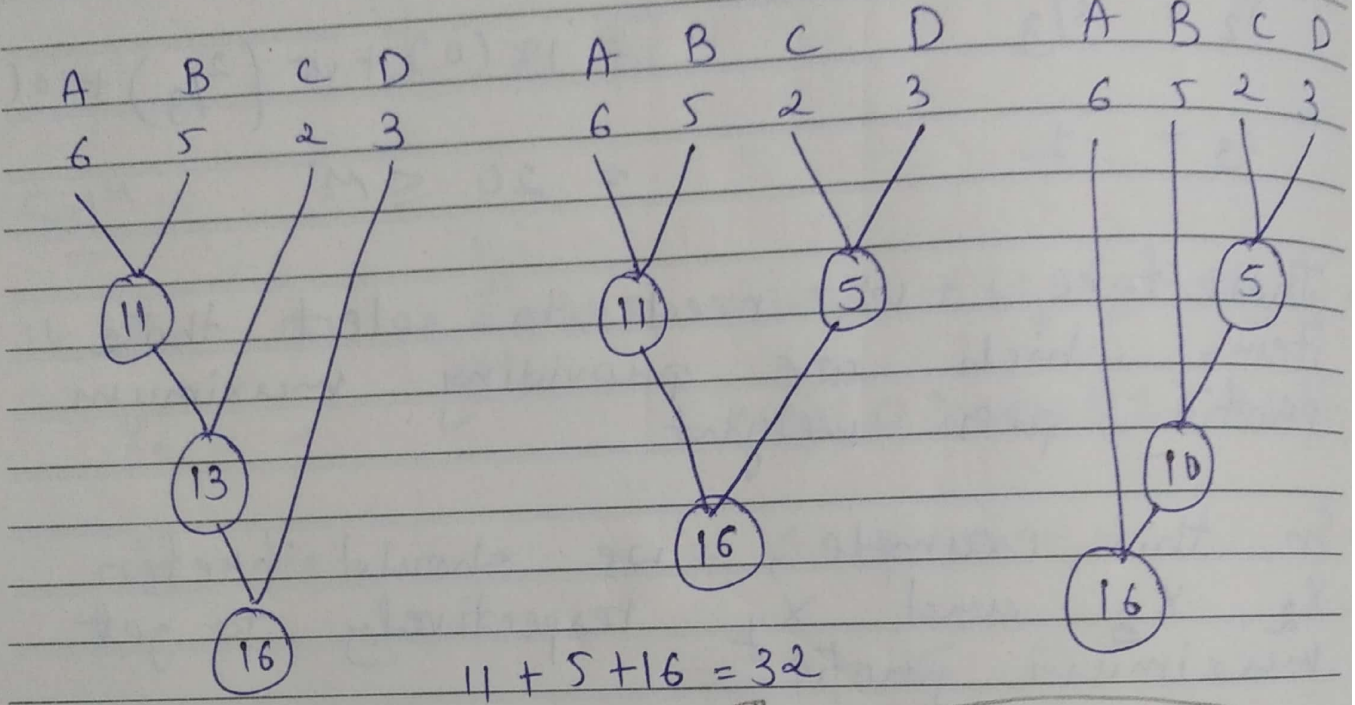
→ Among all the solutions, sol$^n$ 1 is providing maximum profit.

∴ Sol$^n$ 1 is optimal solution.

# * Huffman Coding :-

## * Optimal Merge Pattern :-

→ List : A B C D
Size : 6 5 2 3




11 + 5 + 16 = 32

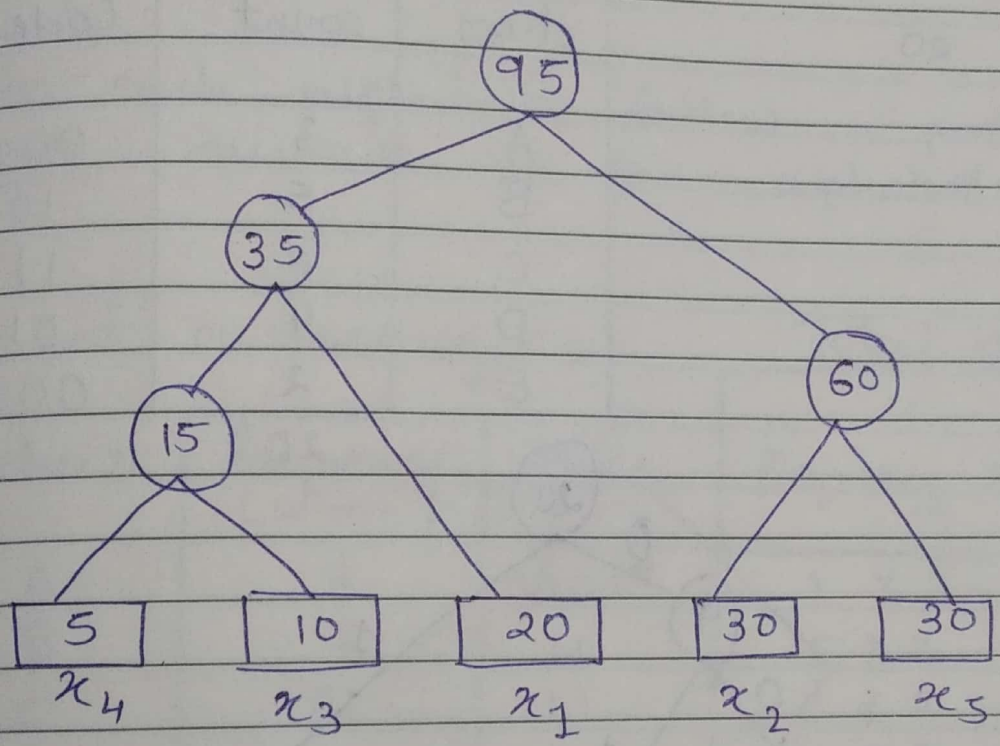
11 + 13 + 16 = 40 ← Time

5 + 10 + 16 = 31

→ In third case we are always selecting two smaller lists for merging.

→ Greedy method says always select two small lists to merge to get better results.

| Lists : | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| Size : | 20 | 30 | 10 | 5 | 30 |




Total time : 15 + 35 + 95 + 60 = 205

$= 3 \times 5 + 3 \times 10 + 2 \times 20 + 2 \times 30 + 2 \times 30$

$= 15 + 30 + 40 + 60 + 60.$

$= 205$

$\therefore \sum_{i=1}^{n} d_i * x_i$

To reach to 2020 95

(15, 35, 95) ⇒ 3

distance

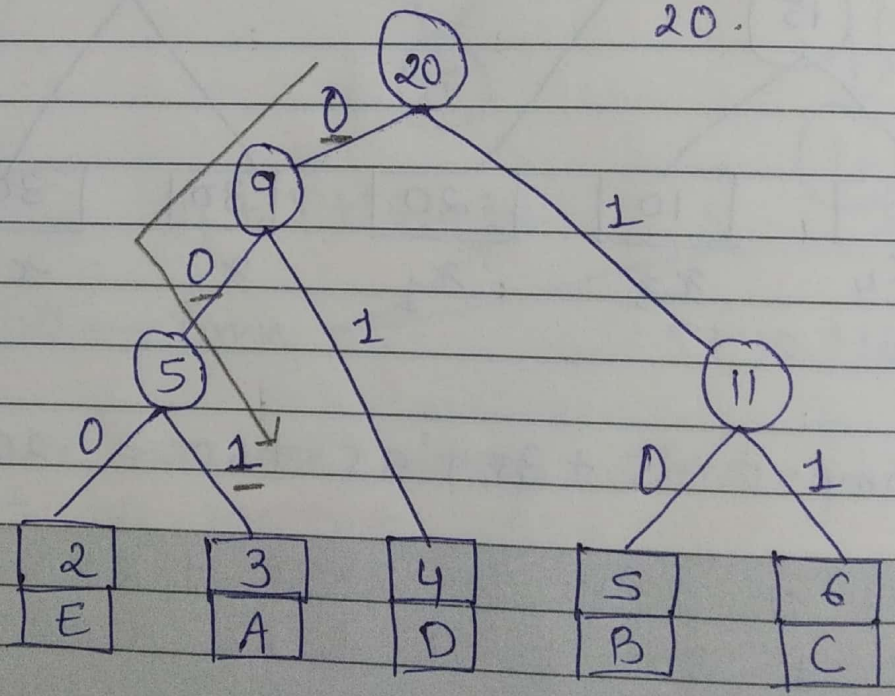
* Huffman Coding :-

→ It's a compression technique used for reducing size of data / message.

Message → B C C A B B D D A E C C B B A E D D C C

length → 20.

| chars | count | Code |
|---|---|---|
| A | 3 | 001 |
| B | 5 | 10 |
| C | 6 | 11 |
| D | 4 | 01 |
| E | 2. | 000 |
| | 20. | |




Optimal Merge Pattern Tree

→ Steps :-

1) Count the number of characters
2) arrange them in increasing order
3) select two smaller number each time and calculate the time.
4) Write left edges 0's and right edges 1's.

5) For each alphabet, follow a path from root onwards to the alphabet.

→ Size of a message :

Total bits = frequency x length of code (bits)

| char | frequency (count) | code | frequency x length of code. (bits) |
|---|---|---|---|
| A | 3 | 001 | 3 X 3 = 9 |
| B | 5 | 10 | 5 X 2 = 10 |
| C | 6 | 11 | 6 X 2 = 12 |
| D | 4 | 01 | 4 X 2 = 8 |
| E | 2 | 000 | 2 X 3 = 6 |

8 X 5 bits = 40 bits

12 bits.

45 bits.

Size of msg.

∴ Total size along with the table/chart

= 40 + 12 + 45

= 97 bits.

# * Job Sequencing with Deadline :-

→ There are n jobs to be processed on a machine.
→ Each job i has a deadline $d_i \geq 1$ and profit $p_i \geq 0$.

→ Each job takes 1 unit of time.
(simplification).

→ $P_i$ (profit) is earned if and only if the job is completed by its deadline.

→ Only one machine is available for processing jobs.

→ Only one job is processed at a time on the machine.

→ We want to find the subset of jobs that maximizes our profit.

→ A feasible solution is a subset of jobs J such that each job is completed by its deadline.

Ex: $n = 4$ ← total no. of jobs.

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$
$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Sol$^n$: Total unit of time available = 2.
(max deadline of job)

| Feasible sol$^n$ (Job number) | Processing sequence | Profit value. |
|---|---|---|
| 1) (1) | (1) | 100 |
| 2) (2) | (2) | 10 |
| 3) (3) | (3) | 15 |
| 4) (4) | (4) | 27. |
| 5) (1, 2) | (2, 1) | 110 |
| 6) (1, 3) | (1,3) or (3,1) | 115 |
| 7) (1, 4) | (4, 1) | 127. |
| 8) (2, 3) | (2, 3) | 25 |
| 9) (3, 4) | (4, 3) | 42. |

/* Here, (2,4) is not possible as both have deadline 1 and only one job is possible at a time. */

→ Solution 7 is the optimal solution having 1 and 4 jobs in (4,1) sequence with the highest profit of 127.

Ex:- no of jobs = 7.

| Jobs | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ | $J_6$ | $J_7$ |
|---|---|---|---|---|---|---|---|
| Profit | 35 | 30 | 25 | 20 | 15 | 12 | 5 |
| Deadline | 3 | 4 | 4 | 2 | 3 | 1 | 2. |

→ in decreasing order

Soln:- Total units of time available = 4
(∵ max deadline)
∴ 4 jobs can be done.

| 0 | $J_4$ | 1 | $J_3$ | 2 | $J_1$ | 3 | $J_2$ | 4 |
|---|---|---|---|---|---|---|---|---|

Profit: 20 + 25 + 35 + 30 = 110

→ Steps:-

1) Arrange all the jobs in profit decreasing order.

2) Select the job and put it in an appropriate place with regards of its deadlines.

Traversal method →

M table

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | \ | \ | ↘ |
| 2 | | 0 | \ | ↘ |
| 3 | | | 0 | ↘ |
| 4 | | | | 0 |

FRIDAY 02 OCTOBER 2020

# * Matrix Chain Multiplication :- (DP)

$$c[i,j] = \min\{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

$$i \le k < j$$

Ex:- $A_1$ $A_2$ $A_3$ $A_4$

$5\times4$ $4\times6$ $6\times2$ $2\times7$

M table

$M_{11} = 0$ $M_{33} = 0$

$M_{22} = 0$ $M_{44} = 0$

| i ↓ j → | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 120 | 88 | 158 |
| 2 | | 0 | 48 | 104 |
| 3 | | | 0 | 84 |
| 4 | | | | 0 |

→ M[1, 2]

$A_1 \cdot A_2$

$5\times4 \quad 4\times6$

$= 5\times4\times6$

$= 120$ $(K = 1)$

K on S table

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 1 | 1 | 3 |
| 2 | | | 2 | 3 |
| 3 | | | | 3 |
| 4 | | | | |

→ M[2, 3]

$A_2 \cdot A_3$

$4\times6 \quad 6\times2$

$= 4\times6\times2 = 48$ $(K = 2)$

WONDER CEMENT — EK PERFECT SHURUAAT

2020

→ M[3,4] : $A_3 \cdot A_4$

6X2 2X7 = 6 X 2 X 7

= 84 (K = 3)

→ M[1,3] : Multiplication from matrix 1 to 3

$A_1 \cdot (A_2 \cdot A_3)$

5X4 4X6 6X2

M[1,1] + M[2,3] + 5X4X2

= 0 + 48 + 40

= 88

$(A_1 \cdot A_2) \cdot A_3$

5X4 4X6 6X2

M[1,2] + M[3,3] + 5X6X2

= 120 + 0 + 60

= 180

→ K = 2.

Min(88, 180) = 88

→ K = 1

→ M[2,4] : $A_2 \cdot (A_3 \cdot A_4)$

04 SUNDAY

4X6 6X2 2X7

M[2,2] + M[3,4] + 4X6X7

= 0 + 84 + 168

= 252

→ K = 2

$(A_2 \cdot A_3) \cdot A_4$

$4 \times 6 \quad 6 \times 2 \quad 2 \times 7$

$M[2,3] + M[4,4] + 4 \times 2 \times 7$

$= 48 + 0 + 56$

$= 104$

$\hookrightarrow K = 3$

$\min(252, 104) = 104 \quad (K = 3)$

$M[1,4]$ $\quad$ * $\quad$ $A_1 \cdot A_2 \cdot A_3 \cdot A_4$

$5 \times 4 \quad 4 \times 6 \quad 6 \times 2 \quad 2 \times 7$

$M[1,4] =$

$\min \{ M[1,\underset{K}{1}] + M[2,4] + 5 \times 4 \times 7 , \quad A_1 \cdot (A_2 \cdot (A_3 A_4))$

$M[1,\underset{K}{2}] + M[3,4] + 5 \times 6 \times 7 , \quad (A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$

$M[1,\underset{K}{3}] + M[4,4] + 5 \times 2 \times 7 , \quad ((A_1 \cdot A_2) A_3) A_4 \}$

~~$M[1,4] + M[2,3] \times$~~

$= \min \{ 0 + 104 + 140 , \quad 120 + 84 + 210 ,$

$88 + 0 + 70 \}$

$= \min \{ 244 , \quad 414 , \quad 158 \}$

$= 158 \longrightarrow K = 3.$

→ The k or s table we got,

Select this first.

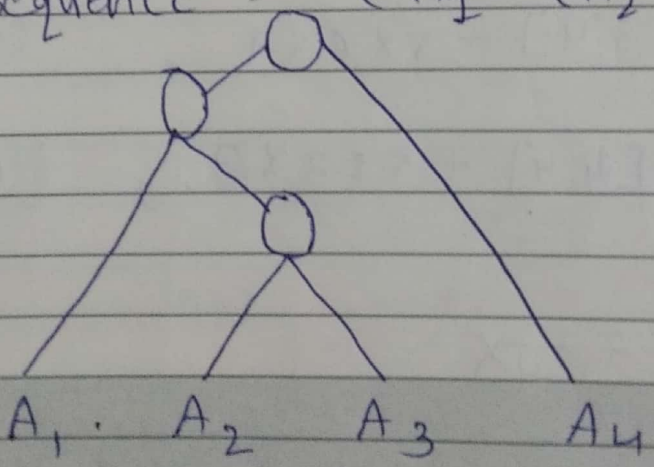
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 1 | 1 | 3 |
| 2 | | | 2 | 3 |
| 3 | | | | 3 |
| 4 | | | | |

$A_1 \quad A_2 \quad A_3 \quad A_4$

$k = 3 \quad \therefore \quad (A_1 \; A_2 \; A_3) \cdot A_4$

$k = 1 \quad \therefore \quad ((A_1) \cdot (A_2 \cdot A_3)) \cdot A_4$

∴ We will get optimal result by following the sequence : $(A_1 \cdot (A_2 \cdot A_3)) \cdot A_4$.




→ Time complexity = $O(n^3)$

# * 0-1 Knapsack Problem :- (DP)

$M = 8$ Profit $P = \{1, 2, 5, 6\}$
↳ Max. weight.
$n = 4$ Weight $W = \{2, 3, 4, 5\}$
↳ no. of objects

Profit Weight Object

Max. weight →

| | | | weights → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 4 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 6 | 5 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| | | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | (8) |

↑ Max Profit

→ Algo :- $V[i, W] = \max\{ V[i-1, W],\ V[i-1, W-W[i]] + P[i] \}$

→

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | $8 - 6 = 2$ |
| | | | | $2 - 2 = 0$ |

∴ $x_2$ and $x_4$ are included in a bag.

08 THURSDAY
OCTOBER 2020

# * Longest Common Subsequence (LCS) :- (DP)

* Subsequence :-

1) String 1 : a b c d e f g h i

String 2 : c d g i

→ Subsequence :- c d g i , d g i , ---

↑ longest is considered.

2) String 1 : a b c d e f g h i

String 2 : e c d g i

→ Subsequence :- e g i , c d g i , d g i , ---

↑ longest common subsequence.

* Algorithm :-

```
if ( A[i] == B[j] )
    LCS[i,j] = 1 + LCS[i-1, j-1]

else
    LCS[i,j] = max{ LCS[i-1, j],
                    LCS[i, j-1] } ;
```

→ Ex:- X = abcd, Y = bd

| | | a | b | c | d |
|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 |
| | 0 | 0 | 0 | 0 | 0 | 0 |
| b | 1 | 0 | 0 | 1 | 1 | 1 |
| d | 2 | 0 | 0 | 1 | 1 | 2 |

max

→ Steps to fill up the table :-

1) b is not matching with a
∴ max of (0,0) as shown in the table will be the answer of (1,1)

2) if b is matching with b
then ADD #1 to the answer of its left upper diagonal LCS.

→ Now, Finding the longest common subsequence from the table :-

| | | a | b | c | d |
|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 |
| | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 1 | 1 | 1 |
| | 2 | 0 | 0 | 1 | 1 | 2 |

b d

→ From the position where we go to diagonally upper side will be the character of a LCS.

∴ bd

→ Ex:- X : A B C B
= Y : B D C A B

| | | | B | D | C | A | B |
|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| B | 2 | 0 | 1 | 1 | 1 | 1 | 2 |
| C | 3 | 0 | 1 | 1 | 2 | 2 | 2 |
| B | 4 | 0 | 1 | 1 | 1 | 2 | 3 |

B C B

∴ longest common subsequence = BCB

→ Time complexity = $O((m+1).(n+1))$
$= O(mn)$.

11 SUNDAY

# * Binomial Co-efficient (DP) :-

→ $C(n,k) = 1$ , if $k=0$ or $n=k$

$C(n-1, k-1) + C(n-1, k)$ for $n > k > 0$.

| n/k | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | (k) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | | | |
| 1 | 1 | 1 | | | | | | | |
| 2 | 1 | 2 | 1 | | | | | | |
| 3 | 1 | 3 | 3 | 1 | | | | | |
| 4 | 1 | 4 | 6 | 4 | 1 | | | | |
| 5 | 1 | 5 | 10 | 10 | 5 | 1 | | | |
| 6 | 1 | 6 | 15 | 20 | 15 | 6 | 1 | | |
| 7 | 1 | 7 | 21 | 35 | 35 | 21 | 7 | 1 | |

→ for i = 0 → n

for j = 0 → k

if (j == 0 OR i == j)

C[i][j] = 1.

else

C[i][j] = C[i-1, j-1] + C[i-1, j];

return C[n, k];

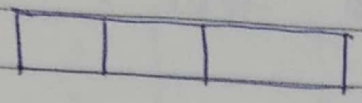
$C(n,k)$ $= {}^{n}C_{k}$

# * Back Tracking :- (DFS)

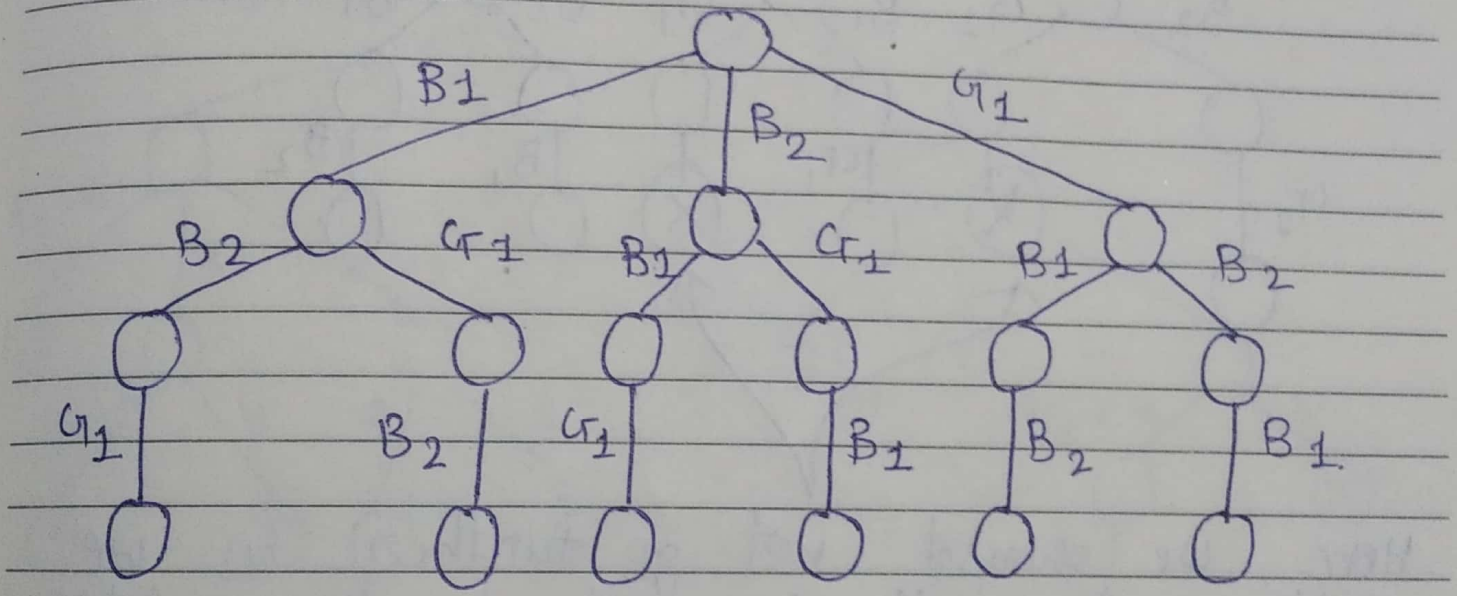
(Brute force approach)

1. $B_1$ $B_2$ $G_1$.

$n! = 3! = 6$ → Possible sol$^n$.



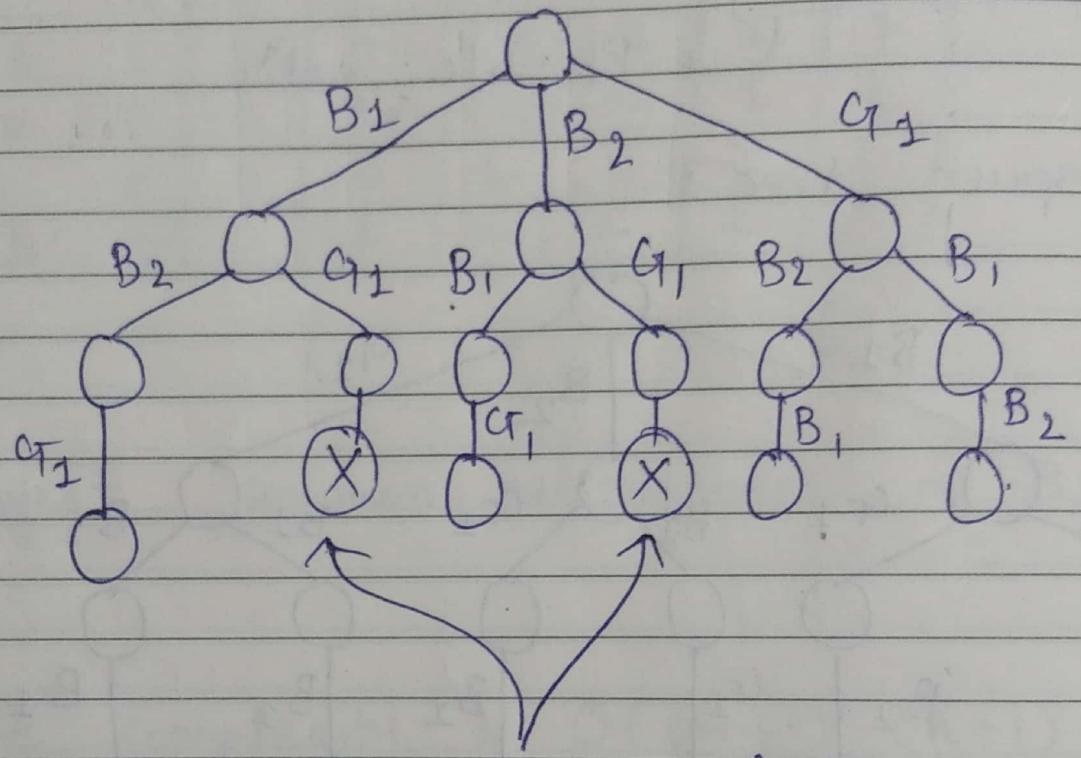
2. State space tree :-




Total 6 possible sol$^n$ we get.

→ Now applying condition that girl should not sit in the middle.

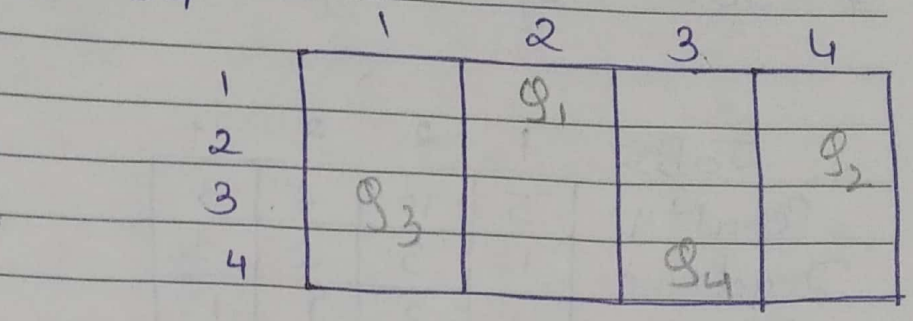
∴ State space tree :-

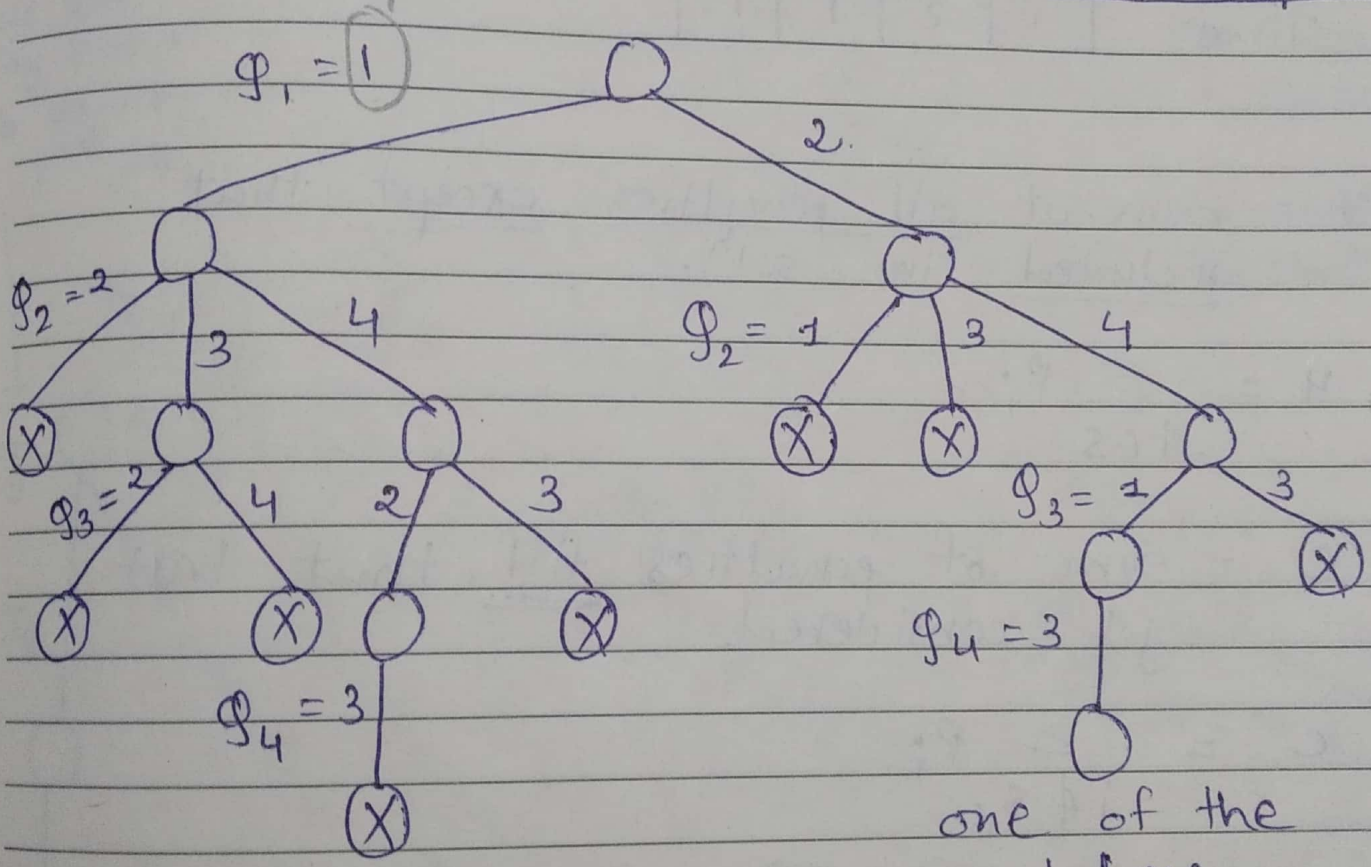



Here, we should not go further as we don't want girl to sit in the middle.

# * N - Queens Problem :-

$Q_1, Q_2, Q_3, Q_4$




one of the solutions.

∴ Queens can be placed at columns 2, 4, 1, 3 and its mirror image 3, 1, 4, 2.

# * Branch and Bound :- ( BFS ) and ( DFS )

## * Job sequencing with deadlines :-

| Jobs | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Penalty | 5 | 10 | 6 | 3 |
| Deadline | 1 | 3 | 2 | 1 |
| Time | 1 | 2 | 1 | 1 |

u = sum of all penalties except that included in soln.

$$u = \sum_{i \notin S} p_i$$

c = sum of penalties till that last job considered.

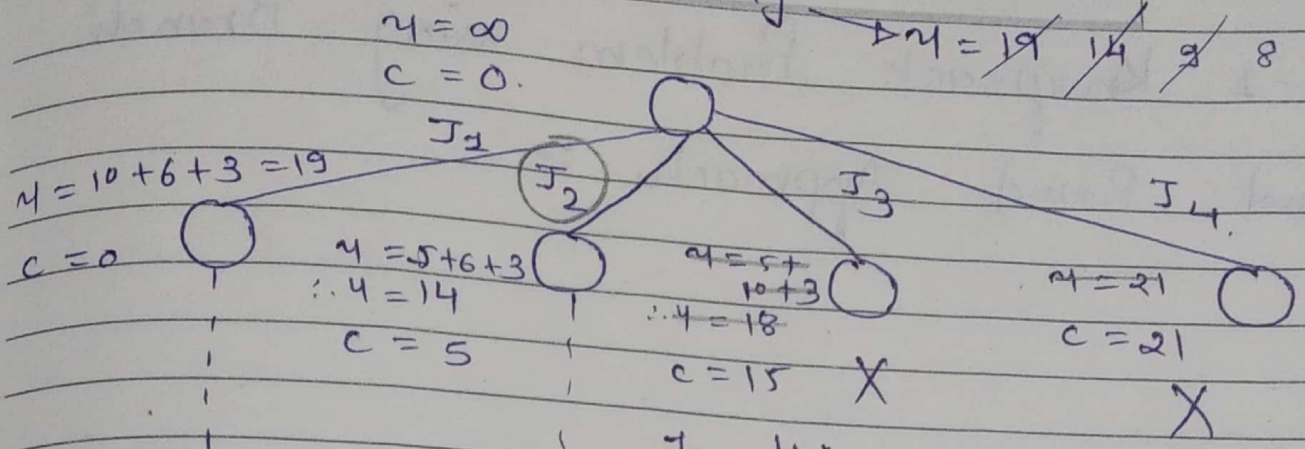
$$c = \sum_{i \notin S_k} p_i$$

∴ Minimum cost is 8.

∴ When we complete job 2 and 3 then we get minimum penalty.

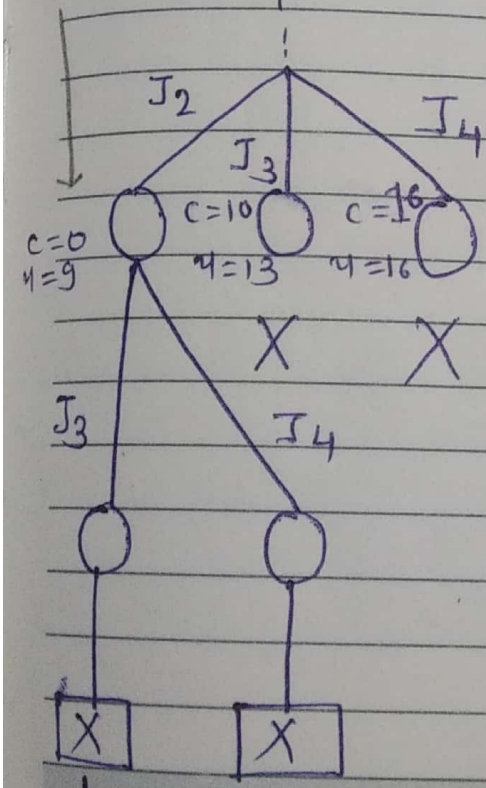
∴ Penalty that we will be paying is 5+3 = 8.

Upper Bound of Penalty → u = ~~19~~ ~~14~~ ~~9~~ 8



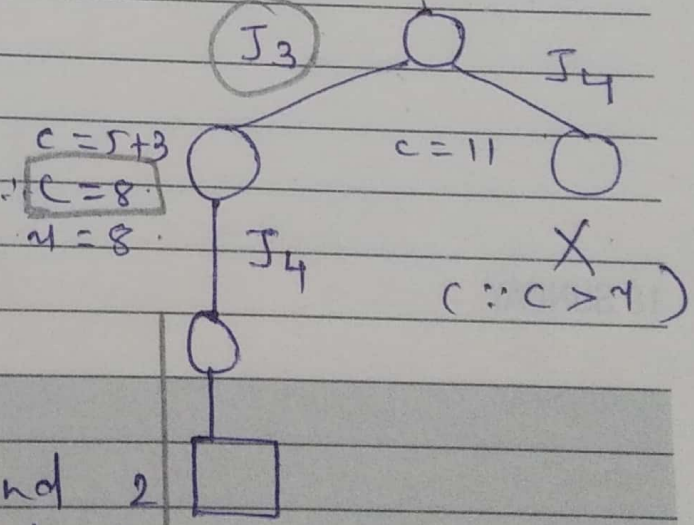

In this cases, cost is greater than the sum of all penalties (upper bound)
∴ We need not to consider those nodes further.

(∵ we are doing first job as well as 2nd job)




(∵ costs are greater than 9.)




We have completed job 1 and 2 having deadlines 1 and 3 which consumes time: 1 and 2 respectively.
∴ we have reach to deadline 3
by doing first two jobs
∴ We can NOT consider Job 3 further.

* Job assignment with deadlines and profit :-

| Tasks | Deadlines | Profit |
|---|---|---|
| $T_1$ | 7 | 15 |
| $T_2$ | 2 | 20 |
| $T_3$ | 5 | 30 |
| $T_4$ | 3 | 18 |
| $T_5$ | 4 | 18 |
| $T_6$ | 5 | 10 |
| $T_7$ | 2 | 23 |
| $T_8$ | 7 | 16 |
| $T_9$ | 3 | 25 |

→ Highest deadline = 7.

∴ Create 7 no. of boxes to assign tasks

deadline 2 ↓ 7 ↓

| $T_2$ | $T_7$ | $T_9$ | $T_5$ | $T_3$ | $T_8$ | $T_1$ |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

→ Now, we are required to find the sequence of tasks which should be followed to get maximum profit.

→ Steps to be followed :-

1) Find the max profit task and assign it in the box with respect to it's deadline.

2) Now, find the deadlines which are matching with the highest profit task. Here, we are searching for highest profit therefore just assign the task in appropriate box. */

3). When we include task 3 in our sequence, task 6 will not be included as it has lower profit than that of task 3.

→ if deadline is 2 that means we can do this in 1 as each task occupies 1 unit of time.

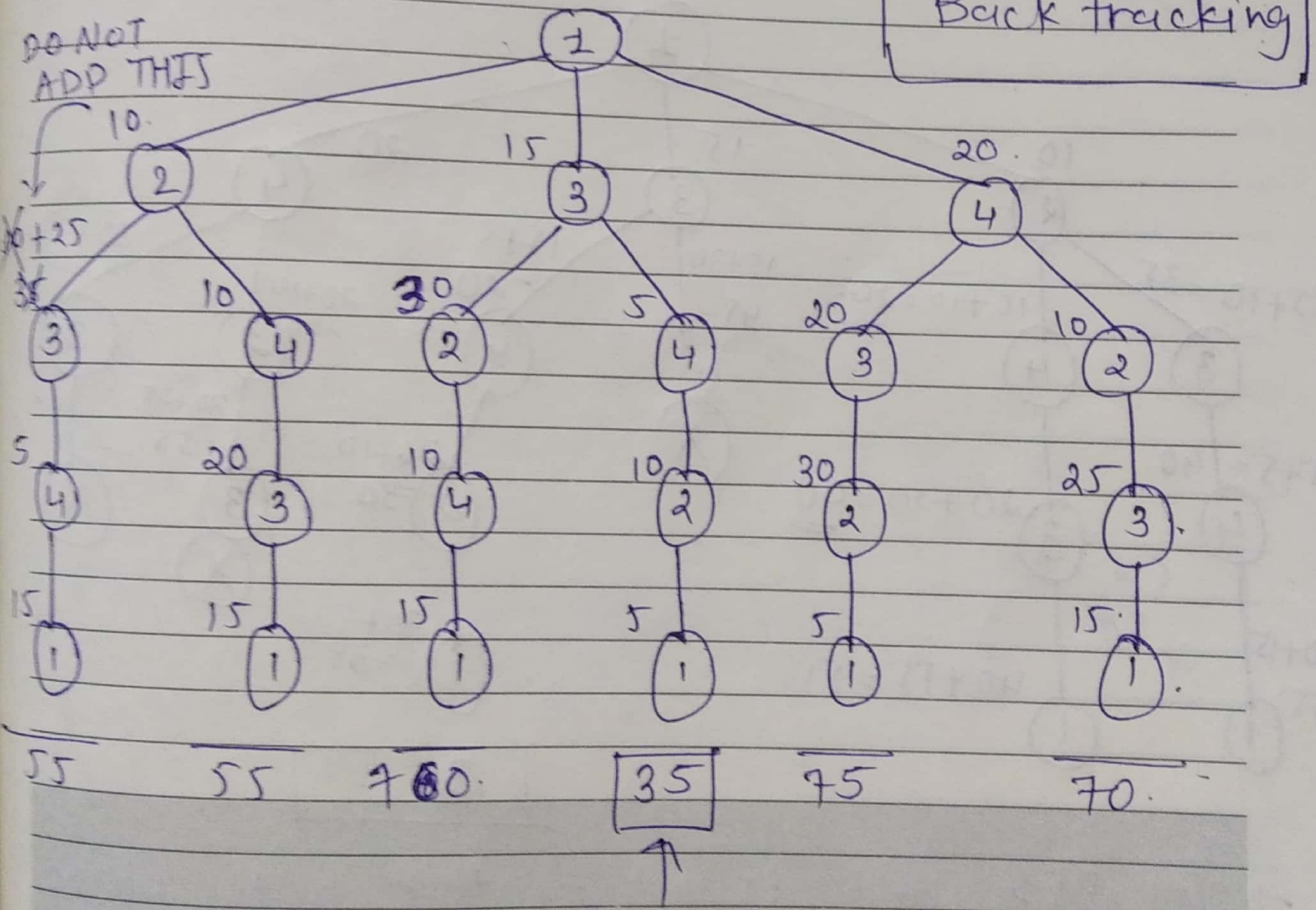
→ Here, task 8 is having deadline of 7 but we have vacancy at deadline 6. Additionally, At that moment task 8 is the only one which is providing higher profit.

∴ The sequence = $T_2, T_7, T_9, T_5, T_3, T_8, T_1$

# * Traveling Sales Person :-

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 25 | 10 |
| 3 | 15 | 30 | 0 | 5 |
| 4 | 15 | 10 | 20 | 0 |

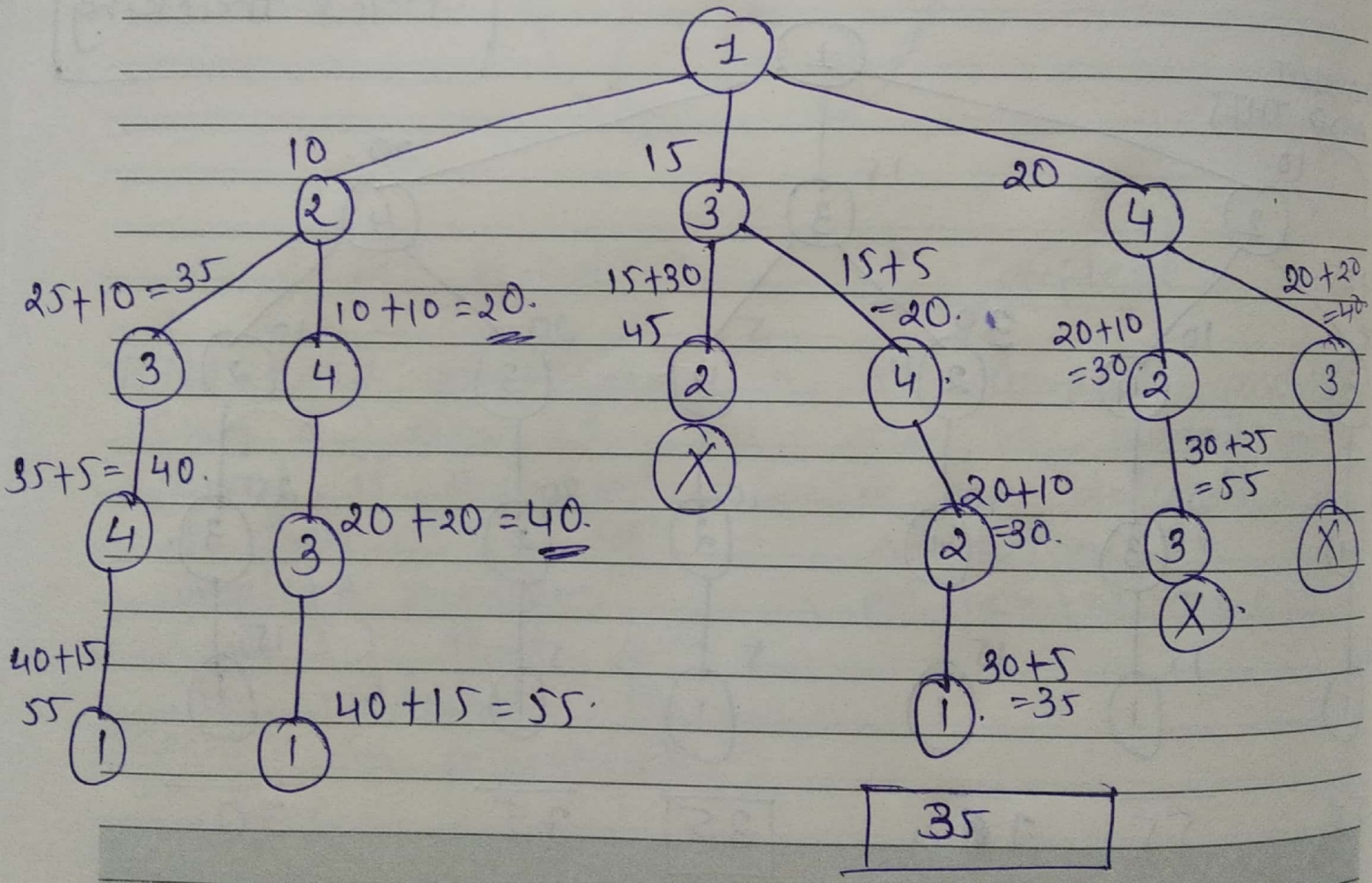
Using Back tracking




Optimal Path : 1, 3, 4, 2, 1
(∵ we are getting least cost.)

→ **Using Branch and Bound**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 25 | 10 |
| 3 | 15 | 30 | 0 | 5 |
| 4 | 15 | 10 | 20 | 0 |




→ Go BFS and extend further nodes whose cost is least in/among all the nodes.

# * NP Hard and NP Complete :-

## * Polynomical time :-

| | | |
|---|---|---|
| Linear search | → | $n$ |
| Binary search | → | $\log n$ |
| Insertion sort | → | $n^2$ |
| Merge sort | → | $n \cdot \log n$ |
| Matrix Multiplication | → | $n^3$ |

## * Exponential time :-

| | | |
|---|---|---|
| 0/1 knapsack | → | $2^n$ |
| Traveling SP | → | $2^n$ |
| Sum of subsets | → | $2^n$ |
| Graph colouring | → | $2^n$ |
| Hamiltonian cycle | → | $2^n$ |

* Non-deterministic :-

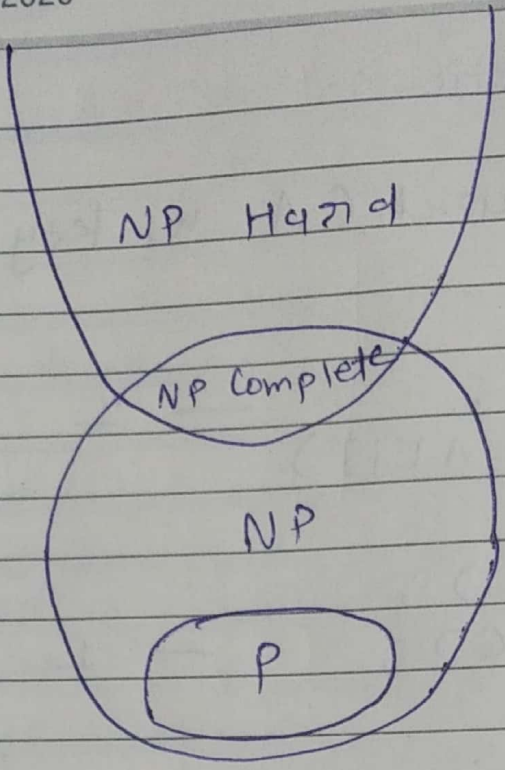
```
Algorithm NSearch(A, n, key)
{
    j = choice();          — 1
    if (key = A[j])
    {
        write(j);
        success();         — 1
    }
    write();
    failure();             — 1.
}
```

Non deterministic

O(1)

* P problems : Problems which can be solved in polynomial time.

* NP problems : Problems which can be solved in non-deterministic polynomial time.




## NP Hard

→ NP Hard problems (say X) can be solved if and only if there is a NP complete problem (say Y) that can be reducible into X in polynomial time.

→ Ex:- Halting problem, Vertex cover problem.

## NP Complete

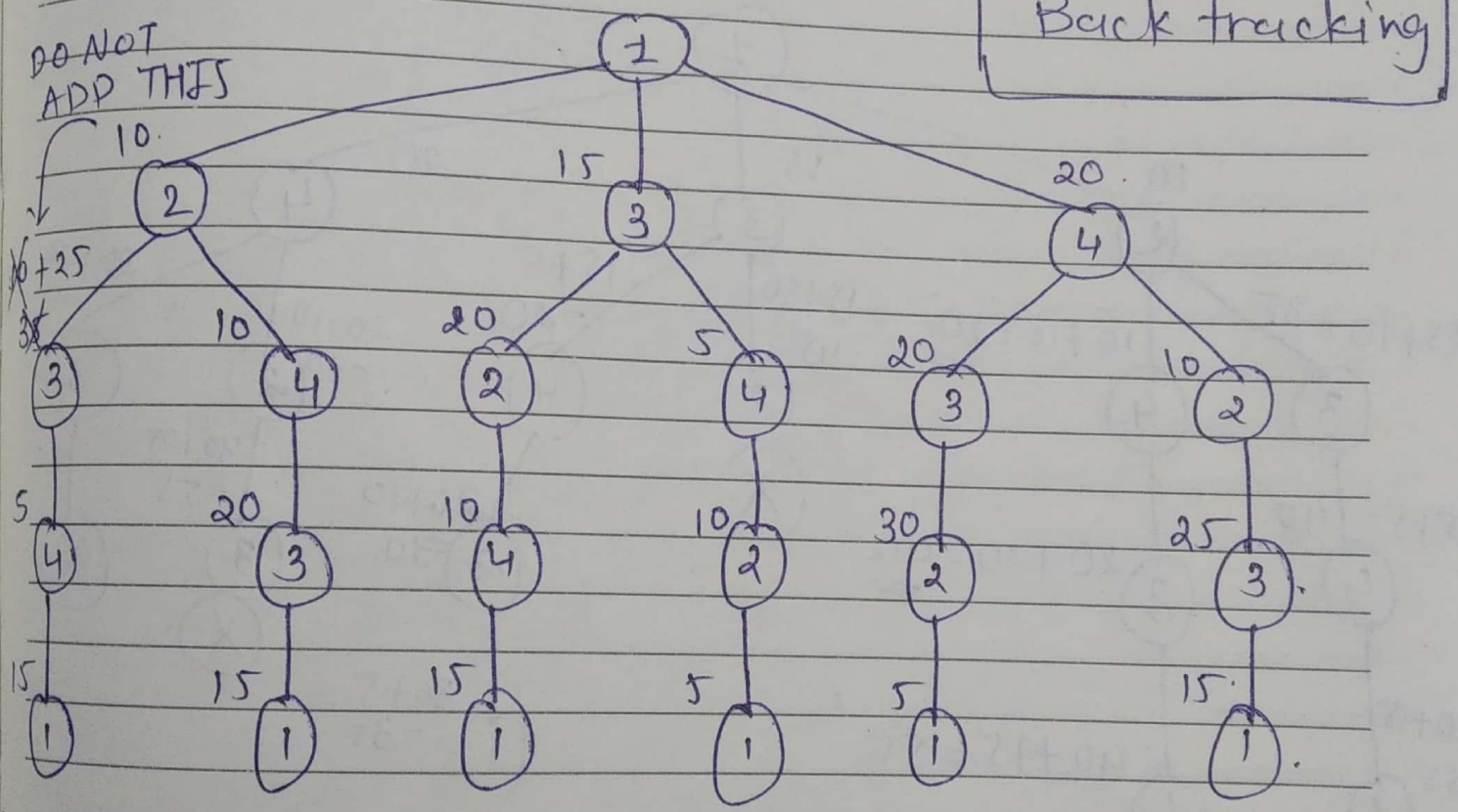
→ NP Complete problems can be solved by a non-deterministic algorithm in polynomial time.

→ Ex:- Determine whether a graph has a Hamiltonian cycle or not.

# * Traveling Sales Person :-

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 25 | 10 |
| 3 | 15 | 30 | 0 | 5 |
| 4 | 15 | 10 | 20 | 0 |

Using Back tracking




55 55 60 35 75 70

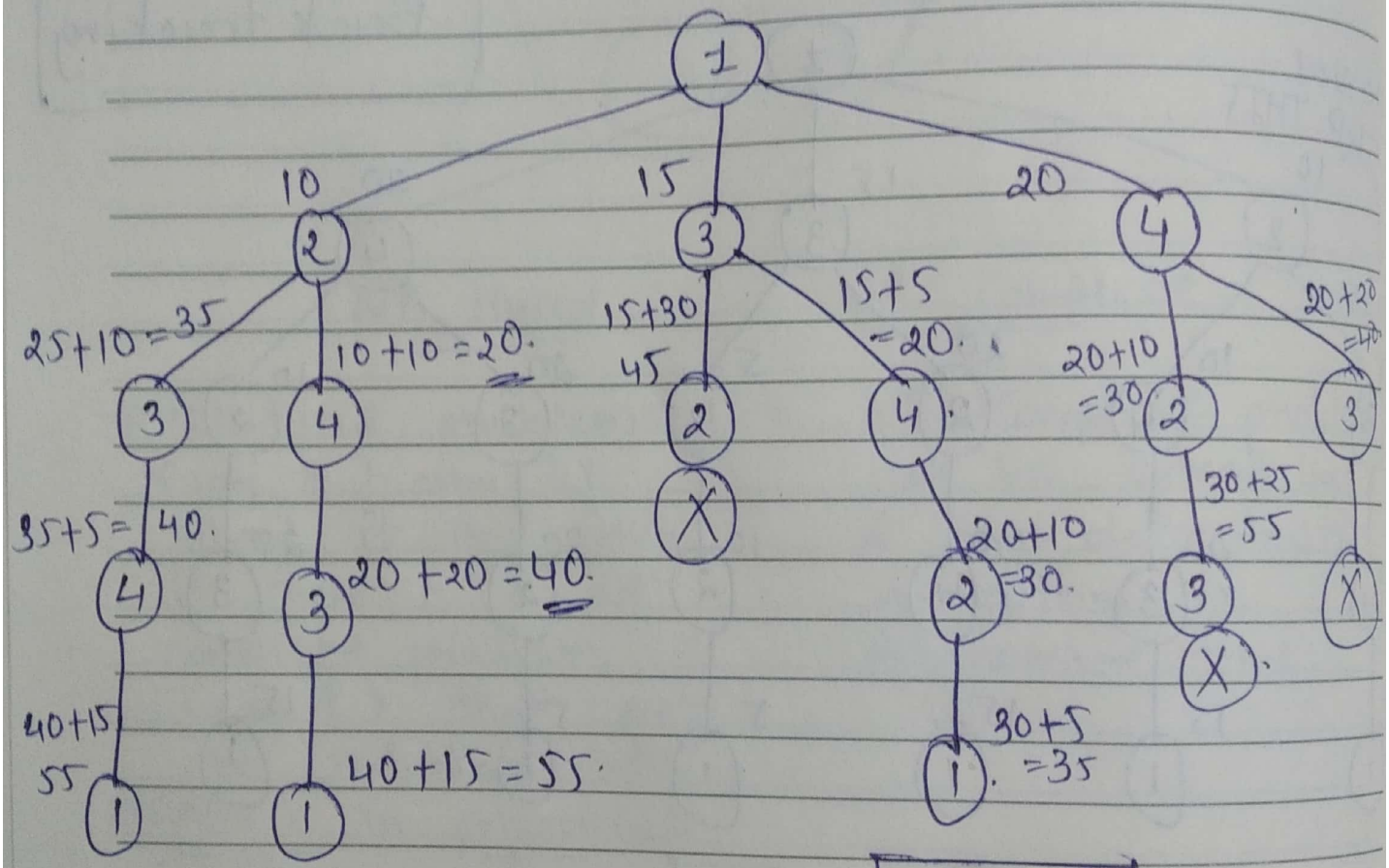
Optimal Path : 1, 3, 4, 2, 1
(∵ we are getting least cost.)

27 TUESDAY

OCTOBER 2020

→ Using Branch and Bound

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 25 | 10 |
| 3 | 15 | 30 | 0 | 5 |
| 4 | 15 | 10 | 20 | 0 |



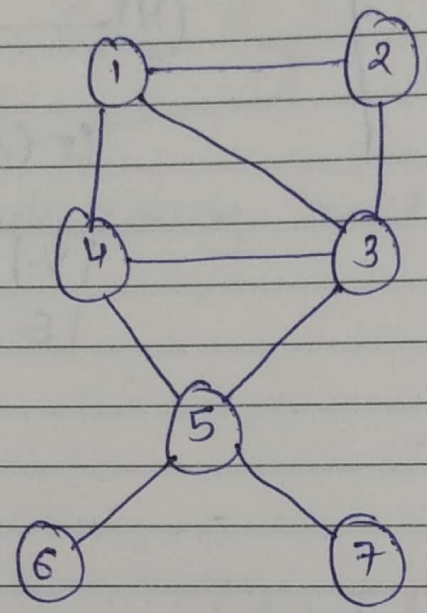

35

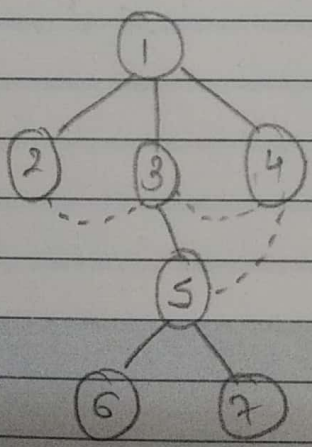
→ Go BFS and extend further nodes whose cost is least in all the nodes among.

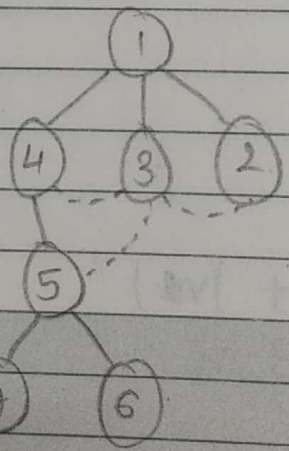
# * Breadth First Search BFS :-

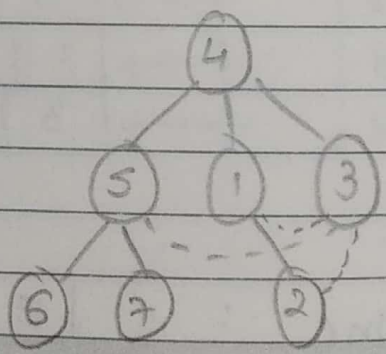
→ Can start traversal from any starting vertex.
→ When a vertex is visited, it should be visiting
↳ explored completely.
→ Neighbouring vertices can be visited in any order.

Implemented using queue.



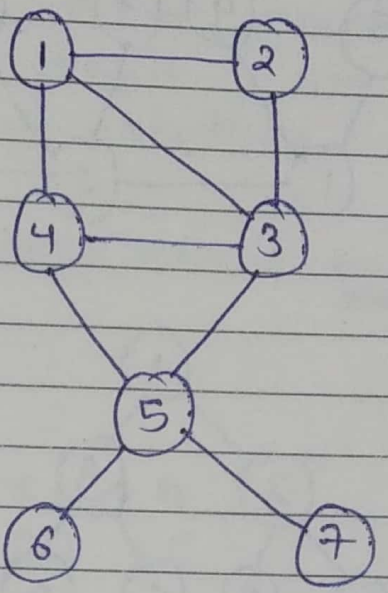




4, 5, 1, 3, 6, 7, 2.

BFS: 1, 2, 3, 4, 5, 6, 7 1, 4, 3, 2, 5, 7, 6

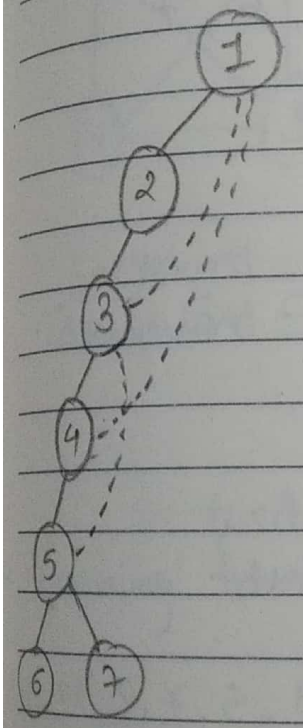
# * Depth First Search DFS :-

DFS : 1, 2, 3, 4, 5, 6, 7

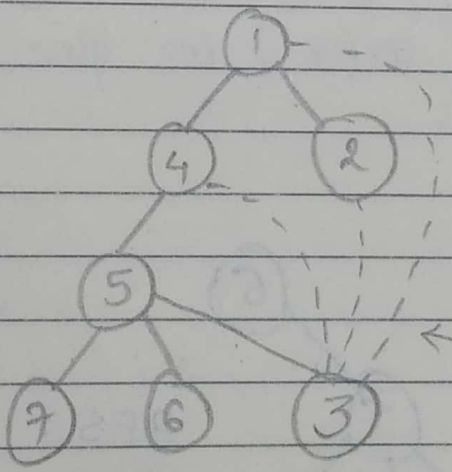



Implemented using stack

| |
|---|
| ~~8~~ 5 |
| ~~4~~ |
| ~~3~~ |
| ~~2~~ |
| ~~1~~ |




DFS : 1, 4, 5, 7, 6, 3, 2




Back edges (- - - →)

| |
|---|
| ~~8~~ |
| ~~4~~ |
| ~~1~~ ~~1~~ |

DFS spanning trees

→ Time : O(n)

BFS : Visit one, explore it

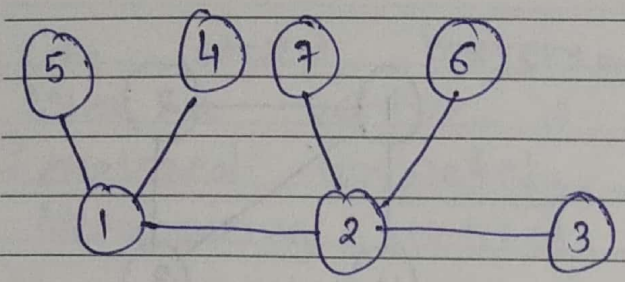
DFS : Visit all, explore it

27 SATURDAY JUNE 2020



* Examples :-

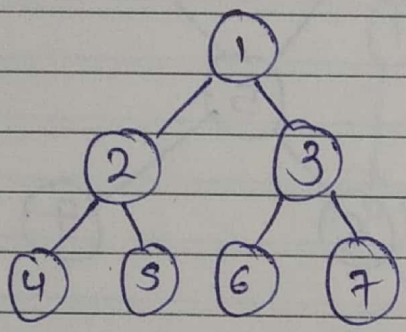
1)




BFS : 1, 5, 4, 2, 7, 6, 3

DFS : 1, 2, 3, 6, 7, 4, 5

2)

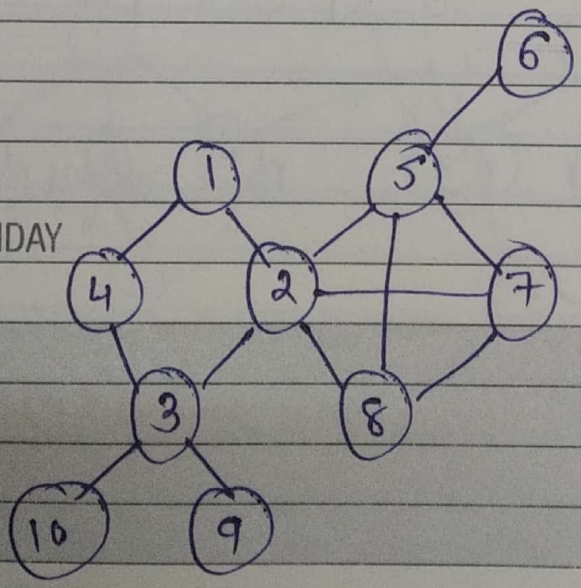



BFS : 1, 2, 3, 4, 5, 6, 7.

DFS : 1, 2, 4, 5, 3, 6, 7

→ In binary tree, BFS is level order traversal and DFS is pre order traversal.

3)




Explore first
visited first

BFS : 1, 2, 4, 3, 5, 7, 8, 10, 9, 6

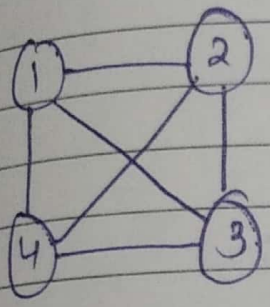
DFS : 1, 2, 8, 7, 5, 6, 3, 10, 9, 4

28 SUNDAY

MONDAY
JUNE 2020
29

## * Spanning Tree :-

- Spanning tree is a subgraph of a graph having all vertices of a graph and $(n-1)$ edges. where $n$ = no. of vertices.
- Graph must be connected and there must be no cycle.




$G_T = (V, E)$

$|V| = n$

$|E| = e.$

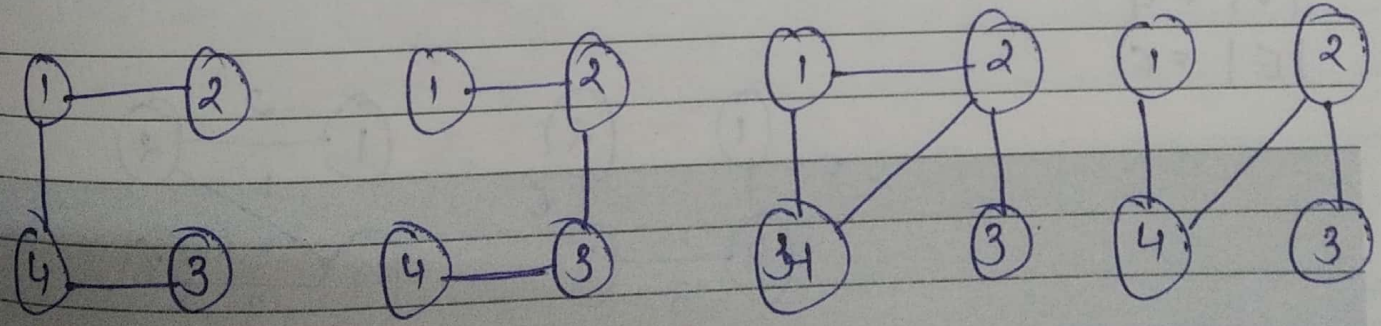
For $S \subseteq G$, $S = (V', E')$

$\therefore |V'| = |V|$

$|E| = |V| - 1$

$= n - 1$

- Spanning trees :-




✓ ✓ ✗ ✓

(∵ Cycle is created) and n vertices.



2020

30 TUESDAY JUNE 2020

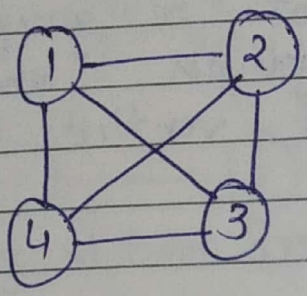
→ Number of spanning tree = $^{|E|}C_{|V|-1}$ - Cycles

$= {}^{6}C_{3} - 4$

$= \frac{6 \times 5 \times 4}{3 \times 2 \times 1} - 4$

$= 20 - 4$

$= 16$

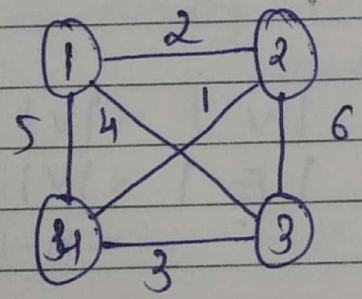


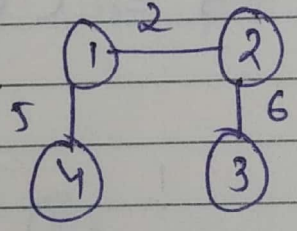

$|V| = 4$

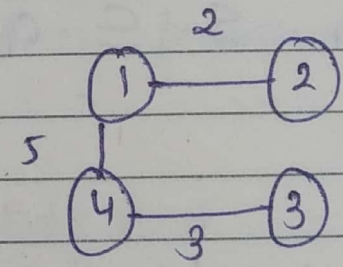
$|E| = 6$

* Minimum Cost Spanning Tree :-

$|E| = |V| - 1 = 3$



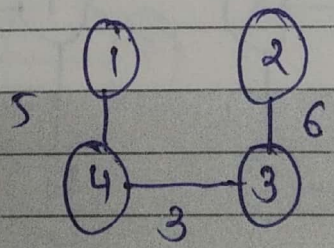



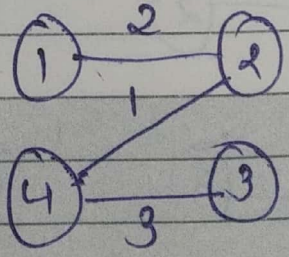

Cost : 13

Cost : 10

$|V| = 4$

$|E| = 6$




Cost : 14

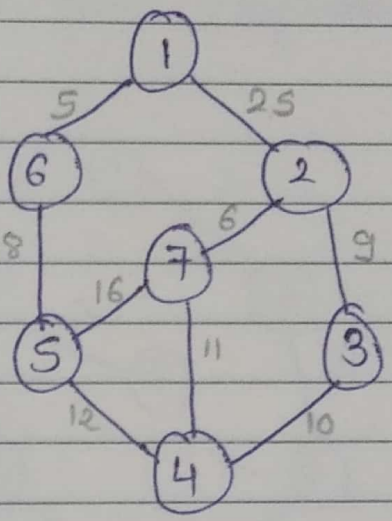
Cost : 6

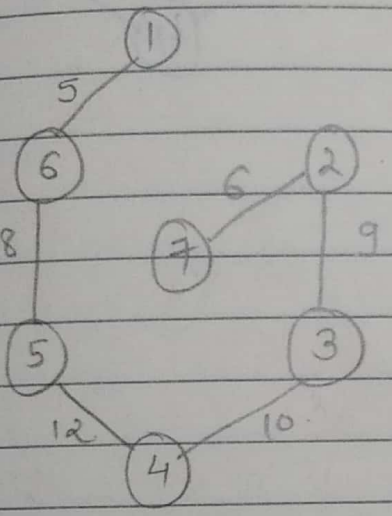
Minimum Cost Spanning Tree.

# * Prim's algorith for Minimum Cost Spanning Tree

1) Select the minimum weighted edge.
2) Select the minimum connected edge.

* Minimum cost spanning tree using prim's algorithm




Cost : 5 + 18 + 12 + 10 + 9 + 6 = 60.

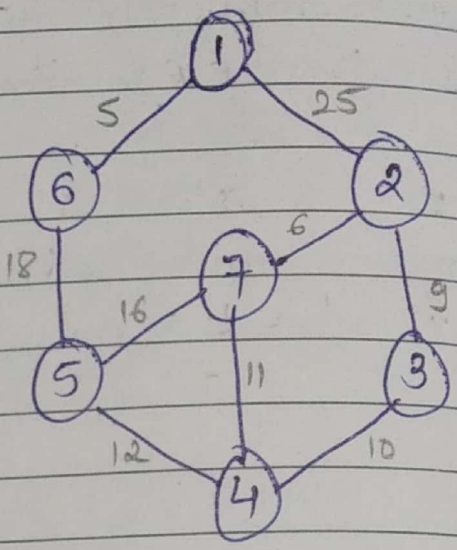
→ Time : $(|V| - 1)(|E|$
$= (n-1)\,e$
$= ne - e$ ≈ $O(n^2)$.

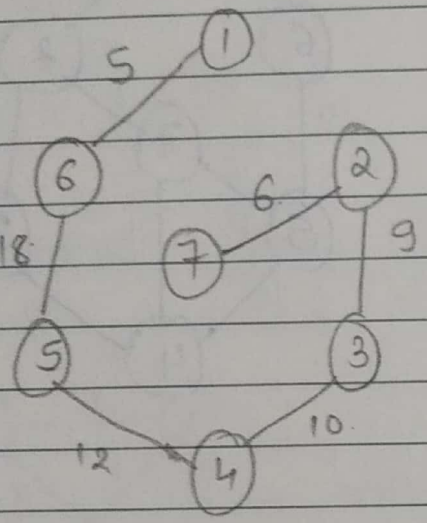
→ Time using heap : $(|V|-1)\log(|E|)$
≈ $O(n.\log n)$

# * Kruskal's algorithm for Minimum Cost Spanning Tree:

1) Always select a minimum cost edge but make sure it should not form a cycle.

→ If it is forming a cycle then just skip that.




Cost : 5 + 18 + 12 + 10 + 9 + 6 = 60.

→ Time : ( |V| − 1 ) (|E|)

= ve − v ≈ O(n²)

→ Time using heap : (|V| − 1) log(|E|)

≈ O(n.logn)

If two graphs are not connected,

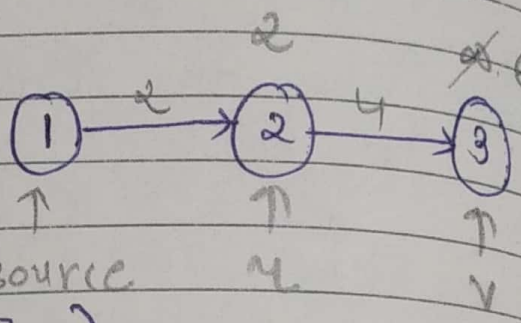
Prim's algorithm will find minimum cost spanning tree for one of the two components.

while kruskal's algorithm will find minimum cost spanning tree for both the components but individually.

04 SATURDAY JULY 2020

# * Dijkstra Algorithm :-

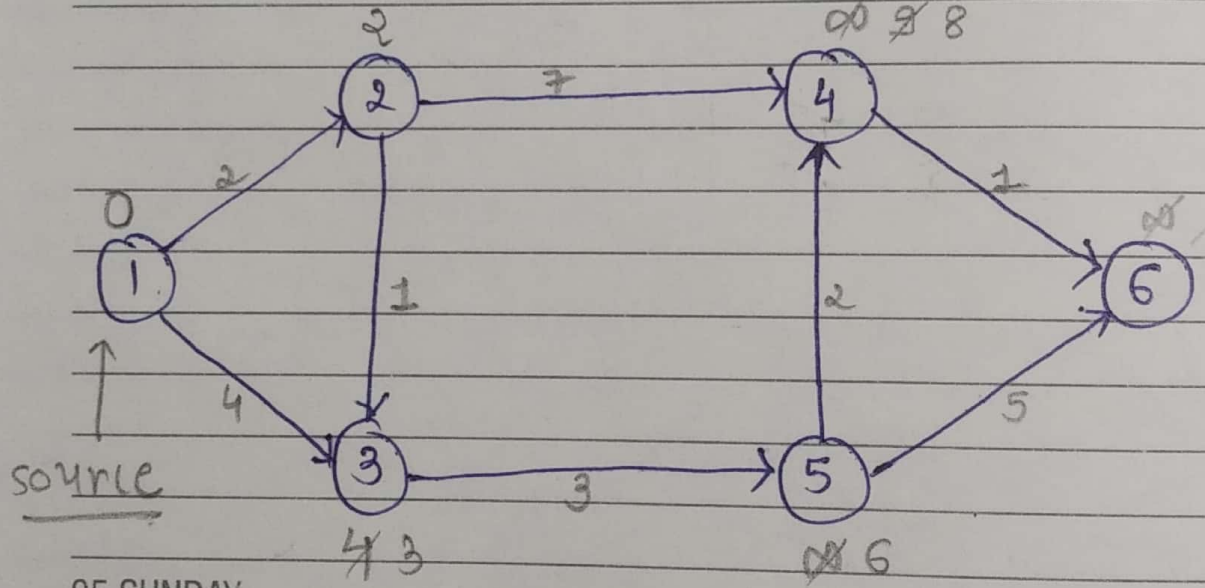
## Relaxation




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

d = distance
c = cost

→ Example :-

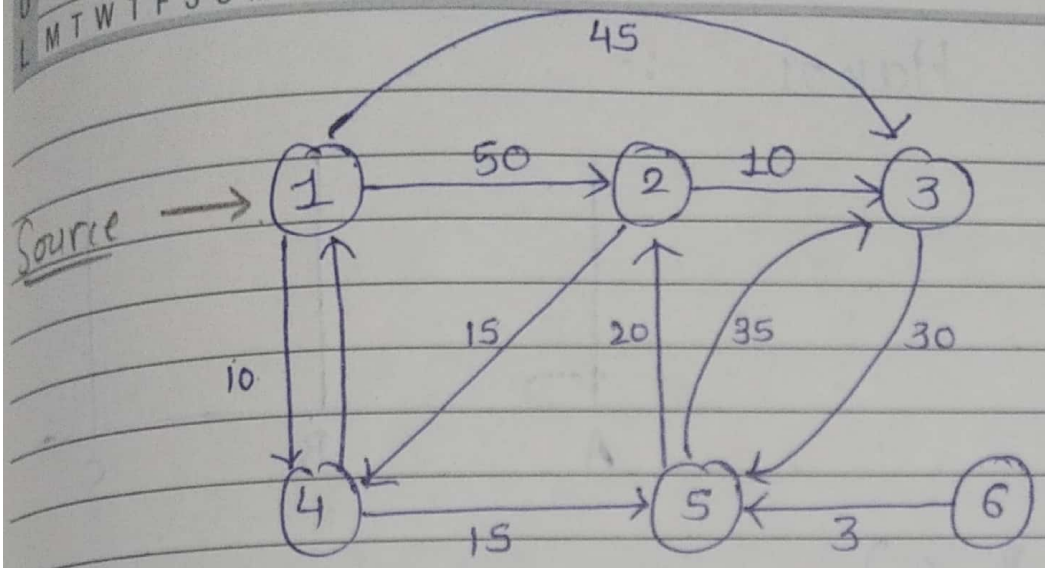



05 SUNDAY

| v | d(v) |
|---|---|
| 2 | 2 |
| 3 | 3 |
| 4 | 8 |
| 5 | 6 |
| 6 | 9 |

→ Time : |v| |v|
n . n
$O(n^2)$ (Worst case)

2020



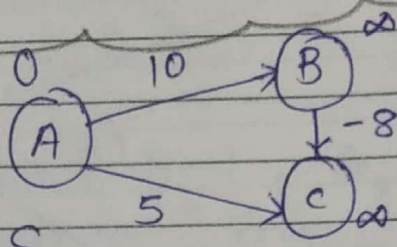

| Selected vertex | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 4 | 50 | 45 | 10 | ∞ | ∞ |
| 5 | 50 | 45 | 10 | 25 | ∞ |
| 2 | 45 | 45 | 10 | 25 | ∞ |
| 3 | 45 | 45 | 10 | 25 | ∞ |
| 6 | 45 | 45 | 10 | 25 | ∞ |

# * Bellman Ford Algorithm :-

→ In this algorithm, we have to relax each node for (v-1) times.

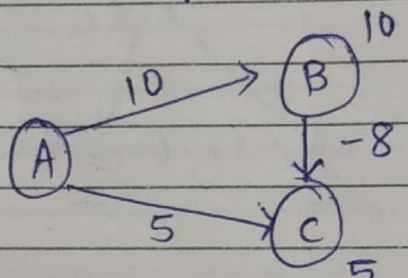
→ Single Source Shortest Path.

→ Ex:-

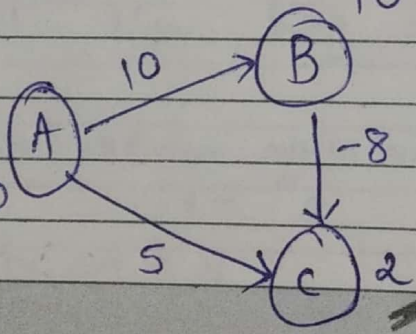



Source → S

↓ $R_1$ ← Relaxation




↓ $R_2$.




→ Bellman ford algorithm will run for v times just to check that whether -negative edge cycle exists or not.

→ Here in the third image, there is no changes from the previous result. Therefore, there does not exist -negative edge cycle.

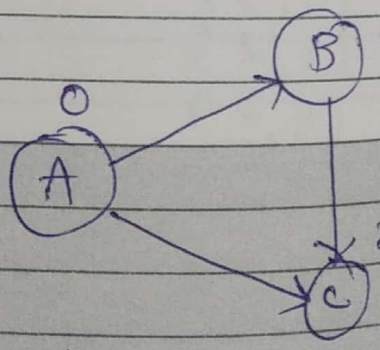
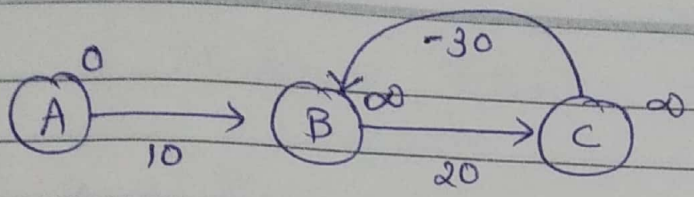



Image:-3

Ex:-

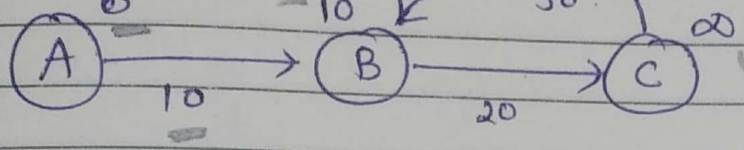



## Drawback of Bellman ford Algorithm

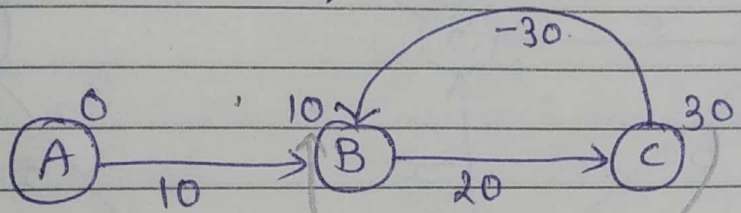
Source → S

R1

0+10 = 10




R2.



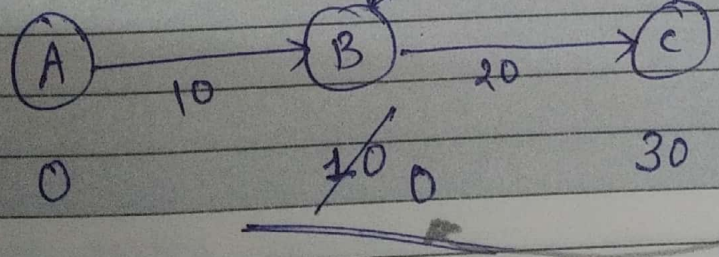

10 + 20 < ∞

Relaxing for 3rd time to check negative edge cycle.

R3

-30 + 30 = 0 < 10.

∴ 0 is there.

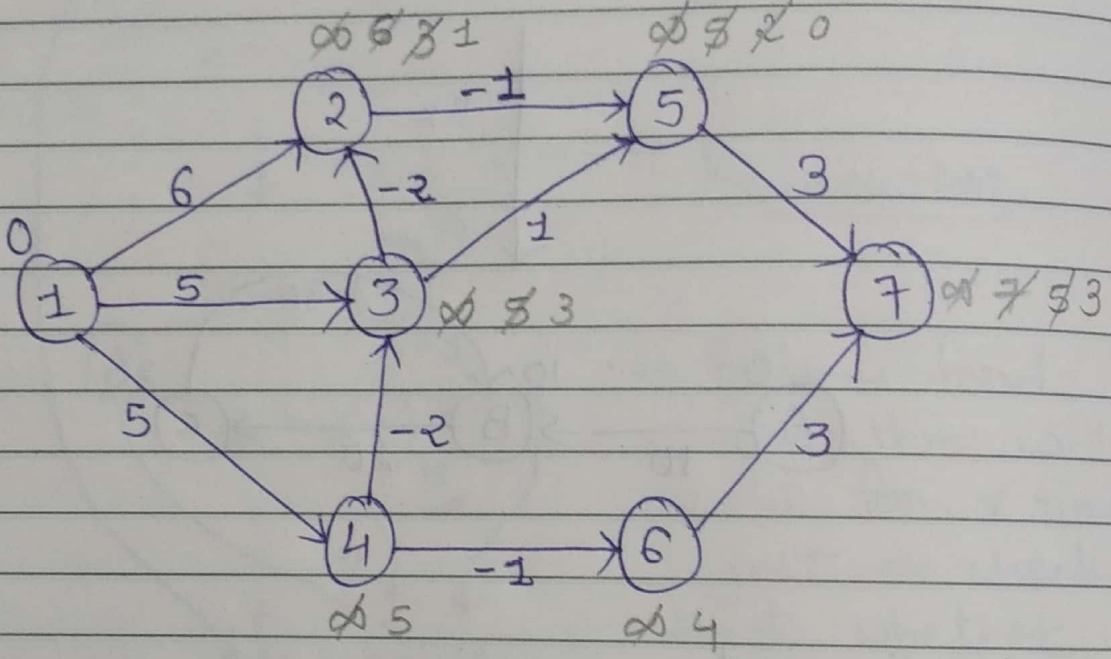



This will decrease each time we relax each edges. Hence there won't be any useful result.

** Relaxation :-

```
if ( d[u] + c(u,v) < d[v] )
{
    d[v] = d[u] + c(u,v);
}
```

→ Ex :-




→ edgelist : (1,2) (2,5) (3,2) (4,3)
(1,3) (4,6) (6,7) (5,7)
(1,4) (3,5)

→ Relaxation : 1, 2, 3, 4

↳ Here we stop relaxing as we got the same result as we got after relaxing for 3rd time.

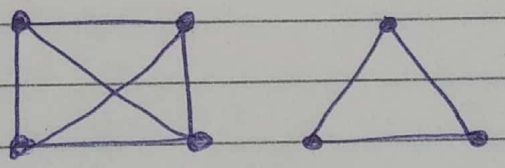
→ vertices - distance from source (1)

| | |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 3 | 3 |
| 4 | 5 |
| 5 | 0 |
| 6 | 4 |
| 7 | 3 |

→ Time complexity = $O(|E||V| - 1)$

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

→ In case of complete graph (where each vertex is connected with other vertices.)



∴ Total no. of edges = $\frac{n(n-1)}{2}$

and total vertices = $n-1$

∴ Time complexity = $\frac{n.(n-1)}{2} \times (n-1)$

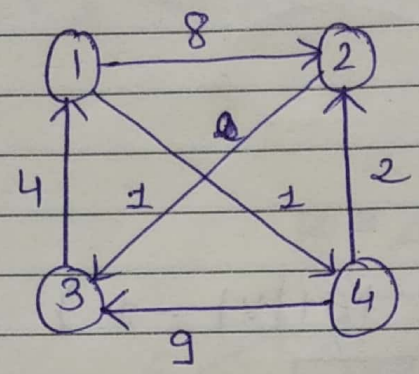
$= \boxed{O(n^3)}$.

16 THURSDAY JULY 2020

# * All Pairs Shortest Path :-

## Floyd Warshall Algorithm

→ Ex :-




→ $D^0$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 8 | ∞ | 1 |
| 2 | ∞ | 0 | 1 | ∞ |
| 3 | 4 | ∞ | 0 | ∞ |
| 4 | ∞ | 2 | 9 | 0 |

$D^1$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 8 | ∞ | 1 |
| 2 | ∞ | 0 | 1 | ∞ |
| 3 | 4 | 12 | 0 | 5 |
| 4 | ∞ | 2 | 9 | 0 |

vertex 1 is included in a path

→ 2 → 3 = 1

2 → 1 and 4 = ∞

∴ min = 1

→ 3 → 2 = ∞

3 → 1, 1 → 2 = 12

∴ min = 12

and

~~3 → 1 → 4, 4~~

9

~~11~~

→ 3 → 4 = ∞

3 → 1, 1 → 4 = 5

~~→ 4 → 2 = 2~~

~~4 → 3, 3 → 1 → 1 → 2~~

~~∴ min = 2~~

~~→ 4 → 3 = 9~~

~~4 → 2, 2 → 3 = 3~~

~~3~~

$D^2$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 8 | 9 | 1 |
| 2 | ∞ | 0 | 1 | ∞ |
| 3 | 4 | 12 | 0 | 5 |
| 4 | ∞ | 2 | 3 | 0 |

vertex 1 and 2 are included in a path

→ 4 → 3 = 9
4 → 2, 2 → 3 = 3

→ 1 → 3 = ∞.
1 → 2, 2 → 3 = 9.
min = 9.

→ 1 → 4 = 1
1 → 2, 2 → 4 = ∞
min = 1

→ 3 → 1 = 4.

→ 3 → 4 = ∞
3 → 1, 1 → 4 = 5.
min = 5.

→ 4 → 1 = ∞.
4 → 2, 2 → 1 = ∞.

$D^3$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 8 | 9 | 1 |
| 2 | 5 | 0 | 1 | 6 |
| 3 | 4 | 12 | 0 | 5 |
| 4 | 7 | 2 | 3 | 0 |

vertices 1, 2 and 3 are included.

→ 1 → 2 = 8
1 → 3, 3 → 2 = ∞
min = 8

→ 1 → 4 = 1
1 → 3, 3 → 4 = ∞
∴ min = 1

(*) → 2 → 4 = ∞
2 → 3, 3 → 1, 1 → 4 = 1 + 4 + 1
= 6
∴ min = 6

1 and 2 can be included.

→ 2 → 1 = ∞
2 → 3, 3 → 1 = 5
∴ min = 5

~~→ 2 → 4 = ∞~~
~~2 → 3~~

It is not necessary to have direct put between 3 → 4.

* → 4 → 1 = ∞.

4 → 3 , 3 → 1 = 9 + 4 = 13

4 → 2 , 2 → 3 , 3 → 1 = 2 + 1 + 4 = 7

∴ min = 7.

We can use 1 and 2.

→ 4 → 2 = 2

4 → 3 , 3 → 1 , 1 → 2 = 9 + 4 + 8 = 21

∴ min = 2

$D^4$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 4 | 1 |
| 2 | 5 | 0 | 1 | 6 |
| 3 | 4 | 7 | 0 | 5 |
| 4 | 7 | 2 | 3 | 0 |

vertices 1, 2, 3 and 4 are included in a path.

→ 1 → 2 = 8

1 → 4 , 4 → 2 = 1 + 2 = 3

19 SUNDAY ∴ min = 3.

→ 1 → 3 = 9

1 → 4 , 4 → 3 = 10.

1 → 4 , 4 → 2 , 2 → 3 = 1 + 2 + 1 = 4

∴ min = 4

→. 2 → 1 = 5

2 → 3 , 3 → 1 = 1 + 4 = 5. ∴ min = 5.

→ 2 → 3 = 1 ∴ min = 1

→ 3 → 1 = 4 ∴ min = 4.

B

→ 3 → 2 = 12

3 → 1 , 1 → 4, 4 → 2 = 4 + 1 + 2 = 7.

∴ min = 7.