ALGORITHMS

*. Algorithm	Program
1. Design	Implementation
2. Domain knowledge	Programmer
3. Any lang	Programming lang
4. Hardware & OS independent	Hardware & OS dependent
5. Analyze	Testing

*. Priori Analysis	Posteriori Testing
1. Algorithm	Program
2. Independent of language	Language dependent
3. Hardware Independent	Hardware dependent
4. Time & Space Function	Watch time & Bytes

*. Characteristics of Algorithm :

1. Input - 0 or more.
2. output - atleast 1 generated.
3. Definiteness - should have single & exact meaning.
4. Finiteness - must terminate & return something.
5. Effectiveness - shouldnot include unnecessary statements.

*. How to write an Algorithm (Example)

```
Algorithm Swap (a, b)
begin
    temp ← a ;
    a ← b ;
    b ← temp ;
End
```

Analyzing:

1. Time
2. Space
3. Network Consumption (optional)
4. Power Consumption (optional)
5. CPU Registers (optional) (Hardware)

Analysis :

1) Time Analysis :

- every simple statement (assigning, declaration) takes 1 unit of time
- Swap (a, b) ⟶ Has constant time complexity. O(1)

2) Space Analysis :

- No. of variables used = 3 {a, b, temp}
- So, again Swap (a, b) takes constant space complexity O(1)

*. Frequency count method :

Ex1:

```
Algorithm Sum (A, n)        // A: Array, n: size of Array
{
    s = 0 ;                 ----> 1
    for (i = 0 ; i < n ; i++)   ----> n+1
    {
        s = s + A[i] ;      ----> n
    }
    return s ;              ----> 1
}
```

O(n)

- statement `s = s + A[i];` is repeating n times

  `for (i=0; i<n; i++)` → takes n+1 time

  total time function, $O(n) = 2n+3$

- For space complexity,

  A → takes n (size of array)

  n → 1

  s → 1

  i → 1

  $n+3 \Rightarrow \boxed{S(n) = n+3}$

  O(n) : time complexity
  O(n) : space complexity

Ex2 :

```
Algorithm Add (A, B, n)
{
    for (i=0; i<n; i++)          ——— n+1
    {
        for (j=0; j<n; j++)      ——— n × (n+1)
        {
            C[i,j] = A[i,j] + B[i,j];   ——— n × n
        }
    }
}
```

Time : $T(n) = 2n^2+2n$

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

Space complexity :

variables : A - n^2
B - n^2
C - n^2
n - 1
i - 1
j - 1

$S(n) = 3n^2+3 \Rightarrow \boxed{O(n^2)}$

Space : $\boxed{O(n^2)}$

Ex3:

```
Algorithm Multipy (A, B, n)
{
n+1 ——— for (i=0; i<n; i++)
        {
(n+1)×n ——— for (j=0; j<n; j++)
            {
n×n ——— C[i,j] = 0;
(n+1)×n×n ——— for (k=0; k<n; k++)
                {
n×n×n ——— C[i,j] = C[i,j] + A[i,k] * B[k,j];
                }
            }
        }
}
```

Variables	
n	1
i	1
j	1
k	1
A	n^2
B	n^2
C	n^2

Time complexity : $T(n) = 2n^3 + 3n^2 + 2n + 1 \longrightarrow \boxed{O(n^3)}$

Space complexity : $S(n) = 3n^2 + 4 \longrightarrow \boxed{O(n^2)}$

Ex4:

```
for (i=1; i<n; i=i+8)
{
    print( );  ——→ n/8 times
}
```

Time complexity = $O(n)$

Ex5:

```
for (i=0; i≤n; i++)
{
    for (j=0; j ≤ i; j++)
    {
        print( );
    }
}
```

i	j	times
0	0✗	0
1	0✓	1
	1✗	
2	0✓	2
	1✓	
	2✗	
⋮	⋮	
n	⋮	n times

Time Complexity, $1+2+3+\ldots+n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$

Time Complexity = $O(n^2)$

Ex6:

```
p = 0
for (i=1 ; p<=n ; i++)
{
    p = p+i ;
}
```

i	p
1	0+1 = 1
2	1+2 = 3
3	1+2+3 = 6
⋮	⋮
K	1+2+3+... +K

It stops if $p <= n$ not

$\Rightarrow p > n$ → stopping criteria

Here, $p = 1+2+3+\ldots+K = \frac{K(K+1)}{2}$

$\therefore \frac{K(K+1)}{2} > n \Rightarrow K^2+K > 2n$

$K^2+K-2n > 0 \Rightarrow K > \sqrt{n}$

Roughly, **Time complexity : $O(\sqrt{n})$**

Ex7:

```
for (i=1; i<n ; i=i*2)
{
    print();
}
```

i
1
$1\times2 = 2$
$2\times2 = 2^2$
$2^2\times2 = 2^3$
⋮
2^K

terminates if $i >= n$

Since, $i = 2^K \Rightarrow 2^K >= n$

$K = \log_2 n$

$\therefore$ **Time Complexity, $O(\log_2 n)$** Take $\lceil \log_2 n \rceil$

Ex8:

```
for (i=n ; i>=1 ; i= i/2)
{
    print();
}
```

i
n
$\frac{n}{2}$
$\left(\frac{n}{2^2}\right)$
$\left(\frac{n}{2^3}\right)$
⋮
$\frac{n}{2^k}$ → kth time

terminates when $i < 1$

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

$\Rightarrow \frac{n}{2^k} < 1 \Rightarrow 2^k > n$

$k > \log_2 n \Rightarrow$ **$O(\log_2 n)$** time complexity

Ex9:

```
for(i=0; i<n; i++)
{
    print();          ——— n
}
for(j=0; j<n; j++)
{
    print();          ——— n
}
```

independent loops

time Complexity $= O(n)$

Ex10:

```
p=0
for(i=1; i<n; i=i*2)
{
    p++;              ——— log2 n = p
}
for(j=1; j<p; j=j*2)
{
    print()           ——— log2 p
}
```

So, $p = \log_2 n$. So, $\log_2 p$ is the total time complexity.

$$\log_2 p = \log_2(\log_2 n) = \log(\log n)$$

$\therefore$ **Time Complexity = $O(\log(\log n))$**

Ex11:

```
for(i=0 ; i<n ; i++)          ——— n
{
    for(j=1 ; j<n ; j=j*2)    ——— n x logn
    {
        print();              ——— n x logn
    }
}
```

Time Complexity = $O(n \log n)$

---

*. Important pieces of code with Time Complexity :

- `for (i=0 ; i<n ; i++)` —— $O(n)$
- `for (i=0 ; i<n ; i=i+2)` —— $O(n)$ $\{\frac{n}{2}\}$
- `for (i=n ; i>1 ; i--)` —— $O(n)$
- `for (i=1 ; i<n ; i=i*2)` —— $O(\log_2 n)$
- `for (i=1 ; i<n ; i=i*k)` —— $O(\log_k n)$
- `for (i=n ; i>1 ; i=i/2)` —— $O(\log_2 n)$
- `for (i=n ; i>1 ; i=i/k)` —— $O(\log_k n)$

*. Analysis of if & while :

Ex1:

```
i=0                          ——— n+1
while (i<n)
{
    stmt;                    ——— n
    i++;                     ——— n
}
```

$F(n) = 3n+2$

Time complexity : $O(n)$

Ex2:

```
a=1;
while (a<b)
{
    stmt;
    a = a*2;
}
```

a
1
$1\times2=2$
$2\times2=2^2$
$2^2\times2=2^3$
⋮
2^k

Stopping criteria $\Rightarrow a \geq b$

Here $a = 2^k \Rightarrow 2^k \geq b$

$k = \log_2 b$

Time complexity : $O(\log_2 n)$

Ex3:

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

i	k
1	1
2	1+1=2
3	2+2=4
4	2+2+3=7
⋮	
m	2+2+3+4+...+m

Stopping Criteria $\rightarrow$ $K \geq n$

we get $K = 2+3+4+5+\cdots+m$

$K = 1+(1+2+3+\cdots+m) = 1+\frac{m(m+1)}{2}$

$1+\frac{m(m+1)}{2} \geq n \approx m = \sqrt{n}$

$\therefore$ **Time Complexity : $O(\sqrt{n})$**

Ex4 : GCD of two numbers m, n :

```
while (m != n)
{
    if (m > n)
    {
        m = m - n;
    }
    else
    {
        n = n - m;
    }
}
```

m = 16	n = 2
14	2
12	2
10	2
8	2
6	2
4	2
2	2

} 7 times

m = 4	n = 3
4	3
1	3

} 2 times

m = 2	n = 2

0 times

So, it is executing almost $\approx \frac{m \text{ value}}{2}$ times

$\therefore$ **Time Complexity : $O(n)$** $\leftarrow$ maximum

If $m == n \Rightarrow$ it executes for constant time

$\therefore$ **Time Complexity : $O(1)$** $\leftarrow$ minimum

Ex5:

```
Algorithm Test(n)
{
    if (n<5)
    {
        printf("%d", n);        ---> 1 time
    }
    else
    {
        for (i=0; i<n; i++)
        {
            printf("%d", i);    ---> n times
        }
    }
}
```

Best Case time : $O(1)$

Worst Case time : $O(n)$

Note: If there is conditional statements in code, there may be varied time complexity based on the condition.

*. **Types of Time functions :** (classes)

- $O(1)$ —— Constant
- $O(\log n)$ —— Logarithm
- $O(n)$ —— Linear
- $O(n^2)$ —— Quadratic
- $O(n^3)$ —— Cubic
- $O(2^n)$ ——> Exponential
- $O(3^n)$,
- $O(n^n)$

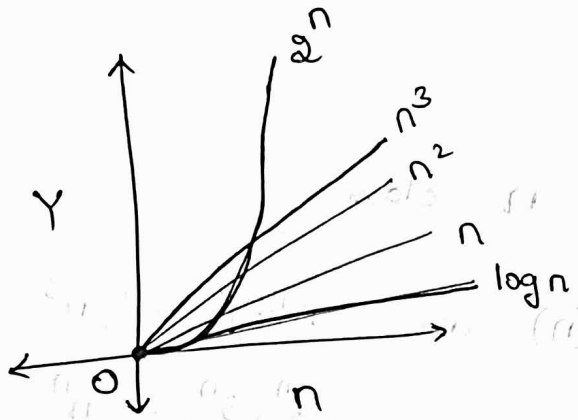
$f(n) = 2$
$f(n) = 10$ —— $O(1)$
$f(n) = 50000$

$f(n) = 2n+3$
$f(n) = \frac{n}{2} + 5n$ —— $O(n)$
$f(n) = \frac{n}{2300} + 3$

.
.
.

*. Comparing classes of Functions : (increasing order)

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \ldots < 2^n < 3^n < \ldots < n^n$



log n	n	n^2	2^n
0	1	1	2
1	2	4	4
2	4	16	16
3	8	64	256
≈ 3.1	9	81	512

*. Asymptotic Notations :

O	big-oh	upper bound
Ω	big-omega	lower bound
Θ	theta	average bound

• Big-oh :

→ The function $f(n) = O(g(n))$ iff ∃ +ve Constants c and n_0 such that $\boxed{f(n) \leq c * g(n)}$ ∀ $n \geq n_0$

Ex: $f(n) = O(2n+3)$

Ex: Suppose $f(n) = 2n+3$

$2n+3 \leq c \cdot n$

$2n - cn \leq -3$ (take +ve constants only)

$cn - 2n \geq 3$

$n(c-2) \geq 3$ → This inequality is valid only if $c > 2$

$\Rightarrow$ least possible value of $\boxed{c = 3}$

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

$2n+3 \leq 3 \cdot n \Rightarrow$ $\boxed{f(n) = 2n+3}$ $\boxed{g(n) = n}$

$\boxed{\text{time complexity : } O(n)}$

$f(n) = 2n+3 \in n$ class

$\therefore$ upper bounds of $f(n)$ are $n\log n, n^2, n^3, \ldots 2^n, 3^n, \ldots n^n$

lower bounds of $f(n)$ are $n, \sqrt{n}, \log n, 1$

Average bound of $f(n)$ is n

- Big-omega :

$\rightarrow$ The function $f(n) = \Omega(g(n))$ iff $\exists$ +ve constant c and n_0 such that $\boxed{f(n) \geq c * g(n)}$ $\forall\ n \geq n_0$

Ex : $f(n) = 2n+3$

$2n+3 \geq c \cdot n$

$2n - cn \geq -3$

$cn - 2n \leq 3$

$n(c-2) \leq 3$ (minimum when $n > 1 \Rightarrow c = 1$)

$2n+3 \geq 1 \cdot n$

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

$\therefore$ $\boxed{\text{Time complexity : } \Omega(n)}$

- **Theta - Notation :**

→ The function $f(n) = \Theta(g(n))$ iff $\exists$ +ve Constants c_1, c_2 and n_0 such that $\boxed{c_1 * g(n) \leq f(n) \leq c_2 * g(n)}$

Ex : $f(n) = 2n+3$

$2n+3$ lies between n & $3n$

lower bound = n

upper bound = $3n$

$\Rightarrow \quad 1 \cdot n \leq 2n+3 \leq 3 \cdot n$ $\quad \therefore \boxed{f(n) = \Theta(n)}$

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

$\therefore$ $\boxed{\text{Time Complexity : } \Theta(n)}$

NOTE : Donot mix the concept of Asymptotic notation with best case, worst case, Average case scenarios. We can use any Notation for Best Case or Worst (Notations $\times$ Case Analysis)

Examples : • $f(n) = 2n^2+3n+4$ (Given)

$2n^2+3n+4 \leq 2n^2+3n^2+4n^2$

$2n^2+3n+4 \leq 9n^2 \Rightarrow O(n^2)$

$2n^2+3n+4 \geq 1 \cdot n^2 \Rightarrow \Omega(n^2)$

$1 \cdot n^2 \leq 2n^2+3n+4 \leq 9n^2 \Rightarrow \Theta(n^2)$

- Suppose given, $f(n) = n^2 \log n + n$

$n^2 \log n + n \leq 10 n^2 \log n \rightarrow O(n^2 \log n)$

$n^2 \log n + n \geq n^2 \log n \rightarrow \Omega(n^2 \log n)$

$n^2 \log n \leq n^2 \log n + n \leq 10 n^2 \log n \rightarrow \Theta(n^2 \log n)$

- Suppose given, $f(n) = n!$

$f(n) = n! = n(n-1)(n-2) \ldots 1$

$1 \times 2 \times \ldots \times n-1 \times n \leq n \times n \times n \ldots n \text{ times}$

$n! \leq n^n \rightarrow O(n^n)$

$1 \times 2 \times \ldots n-1 \times n \geq 1 \times 1 \times \ldots 1$

$n! \geq 1 \rightarrow \Omega(1)$

$\Rightarrow \boxed{1 \leq n! \leq n^n}$

upper bound = n^n

lower bound = 1

Here both sides we are not obtaining same → Here we cannot find average Time Complexity for $n!$

↓

A particular place/range cannot be found in classes of functions ⇒ Upper bound & lower bound is useful.

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

$0 \leq \log(1 \times 2 \times \ldots \times n) \leq \log(n \times n \ldots \times n)$

$\boxed{1 \leq \log n! \leq n \log n}$

- For factorial functions, we cannot decide the time bound. So, we take upper bound = $O(n \log n)$ here
  lower bound = $\Omega(1)$

*. Properties of Asymptotic Notations :

- If $f(n)$ is $O(g(n))$ then $a * f(n)$ is $O(g(n))$.
- If $f(n)$ is $\Omega(g(n))$ then $a * f(n)$ is $\Omega(g(n))$.
- If $f(n)$ is $\Theta(g(n))$ then $a * f(n)$ is $\Theta(g(n))$.
- If $f(n)$ is given then $f(n)$ is $O(f(n))$. (A function is upper bound of itself.)
- If $f(n) = O(g(n))$ & $g(n) = O(h(n))$ then $f(n) = O(h(n))$
- If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$. (It is applicable to only Θ notation).
- If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$
- If $f(n) = \Omega(g(n))$ then $g(n) = O(f(n))$

  } true for O, Ω notations
- If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ then $f(n) = \Theta(g(n))$.

*. Comparision of functions :

Ex : $f(n) = n^2 \log n$ & $g(n) = n (\log n)^{10}$

which is upper bound, lower bound ?

Applying log on both sides,

$\log[n^2 \log n]$ $\qquad$ $\log[n(\log n)^{10}]$

$\log n^2 + \log(\log n)$ $\qquad$ $\log(n) + \log((\log n)^{10})$

$2\log n + \log(\log n)$ $\quad > \quad$ $\log(n) + 10\log(\log n)$

$\Rightarrow \quad n^2 \log n > n(\log n)^{10} \quad \{\because f(n) > g(n)\}$

Ex: $f(n) = 3n^{\sqrt{n}}$, $g(n) = 2^{\sqrt{n}\log n}$

Applying log on both sides

$\log(3n^{\sqrt{n}})$ $\qquad$ $\log(2^{\sqrt{n}\log n})$

$\log 3 + \sqrt{n} \cdot \log n$ $\qquad$ $\sqrt{n}\log n \cdot (\log 2)$

[OR] $3n^{\sqrt{n}}$ $\qquad$ $2^{\sqrt{n}\log n}$

$3n^{\sqrt{n}}$ $\qquad$ $2^{\log_2 n^{\sqrt{n}}}$

$\Rightarrow \quad 3n^{\sqrt{n}}$ $\qquad$ $(n^{\sqrt{n}})^{\log_2 2}$

$3n^{\sqrt{n}} \quad > \quad n^{\sqrt{n}}$ (Value wise)

$=$

$3n^{\sqrt{n}} \to O(n^{\sqrt{n}}) \quad = \quad O(n^{\sqrt{n}})$ (asymptotic equal)

*. Best, Worst & Average Case Analysis :

(I) Linear search

- Best case : Searching key element present at first index.
- Best case time = Constant time : $O(1)$
- worst case : Searching key element present at last index.
- worst case time = no. of elements present : $O(n)$
- Average case : $\frac{\text{All possible case times}}{\text{no. of cases}}$ (based on comparisions)

$$= \frac{1+2+\cdots+n-1+n \text{ Comparisions}}{n} = \frac{n(n+1)}{2} \approx O(n)$$

- Average case time = $\frac{n+1}{2}$: $O(n)$.

Best case	Worst case	Average Case
$B(n) = 1$	$w(n) = n$	$A(n) = \frac{n+1}{2}$
$B(n) = O(1)$	$w(n) = O(n)$	$A(n) = O(n)$
$B(n) = \Omega(1)$	$w(n) = \Omega(n)$	$A(n) = \Omega(n)$
$B(n) = \Theta(1)$	$w(n) = \Theta(n)$	$A(n) = \Theta(n)$

follows

(II) Binary Search Tree

Best case : Searching root element

Best case Time : $\boxed{B(n) = 1}$ $\approx O(1), \Omega(1), \Theta(1)$

worst case : Searching leaf element

worst case Time : (depends on height of B.T) - $\boxed{w(n) = \log n}$

- Minimum worst case time = $\log n$ (balanced B.S.T)
- maximum worst case time = n (left aligned B.S.T)

Note : Best case, worst case, Average case can be written using any notation like Big-oh, Big-omega, Big-theta

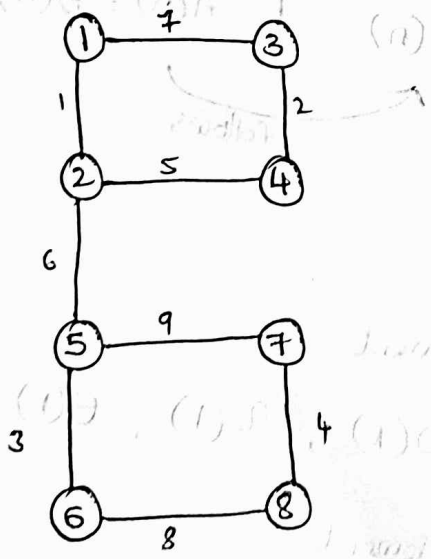
*. DISJOINT SETS :

- Two sets S_1, S_2 are disjoint $\Rightarrow S_1 \cap S_2 = \phi$
- Find cycle in a graph ?

→ If the vertices of an edge belong to same set, ⇒ there exists a cycle.

→ If the both vertices of an edge (any of them) doesn't belong to same set ⇒ perform union between two sets.

Ex : $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$



weight	Taking the edge based on edge weight / dist.
1	(1,2) → remove from U & form new set $S_1 = \{1,2\}$
2	(3,4) ⇒ $S_2 = \{3,4\}$
3	(5,6) ⇒ $S_3 = \{5,6\}$
4	(7,8) → $S_4 = \{7,8\}$

weight	edge
5	{2,4} → checking in S_1, S_2, S_3, S_4

vertex 2 is in S_1, vertex 4 in S_2

$S_1 = \{1,2\}$ ⇒ $S_1 \cup S_2 = \{1,2,3,4\}$ ((2,4) is found by union)

$S_2 = \{3,4\}$

$S_5 = \{1,2,3,4\}$

6 {2,5} → vertex 2 in S_5, vertex 5 in S_3 ⇒ different sets, perform union

$S_3 \cup S_5 \Rightarrow \{1,2,3,4,5,6\} = S_6$

7 {1,3} → both the vertex are in same set S_6 ⇒ there exists CYCLE

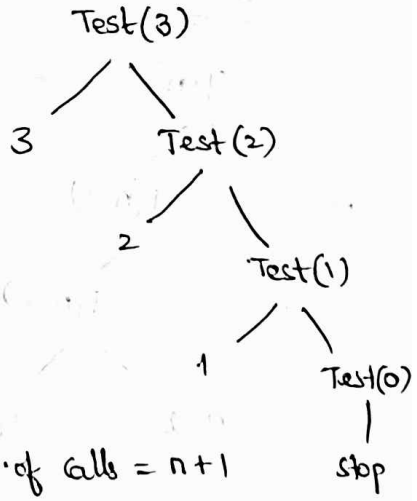
∴ The Given Graph has CYCLE. (using disjoint sets)

*. RECURRENCE RELATION :

Ex:

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

Take n = 3 (Tracing tree)



Time complexity : O(n) ← No. of Calls = n+1

No. of printing = n

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

Ex:

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

$T(n)$ ✓
1
$n+1$
n
$T(n-1)$

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

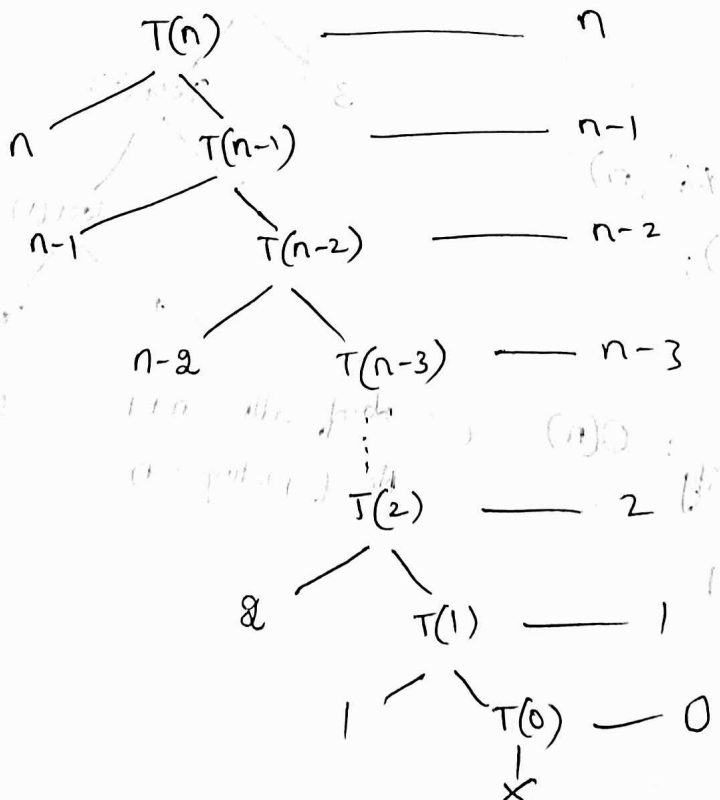
$\Rightarrow$ Recurrence Relation : $T(n) = T(n-1) + (2n+2)$

Asymptotic class

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

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

Tracing Tree



Total sum of time taken

$= n + n-1 + n-2 + n-3 + \cdots + 2 + 1$

$= 1 + 2 + \cdots + n-1 + n$

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

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

[Time Complexity : $O(n^2)$]

Ex: Solving Recurrence relation using Back Substitution:

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

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

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

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

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

$\vdots$

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

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

$T(n) = T(0) + 1 + 2 + \cdots + (n-2) + (n-1) + n$

$T(n) = T(0) + [1 + 2 + \cdots + (n-1) + n]$

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

$\boxed{T(n) = 1 + \left(\frac{n(n+1)}{2}\right)}$ → No. of Calls

$\boxed{T(n) = \frac{n(n+1)}{2}}$ → Time function

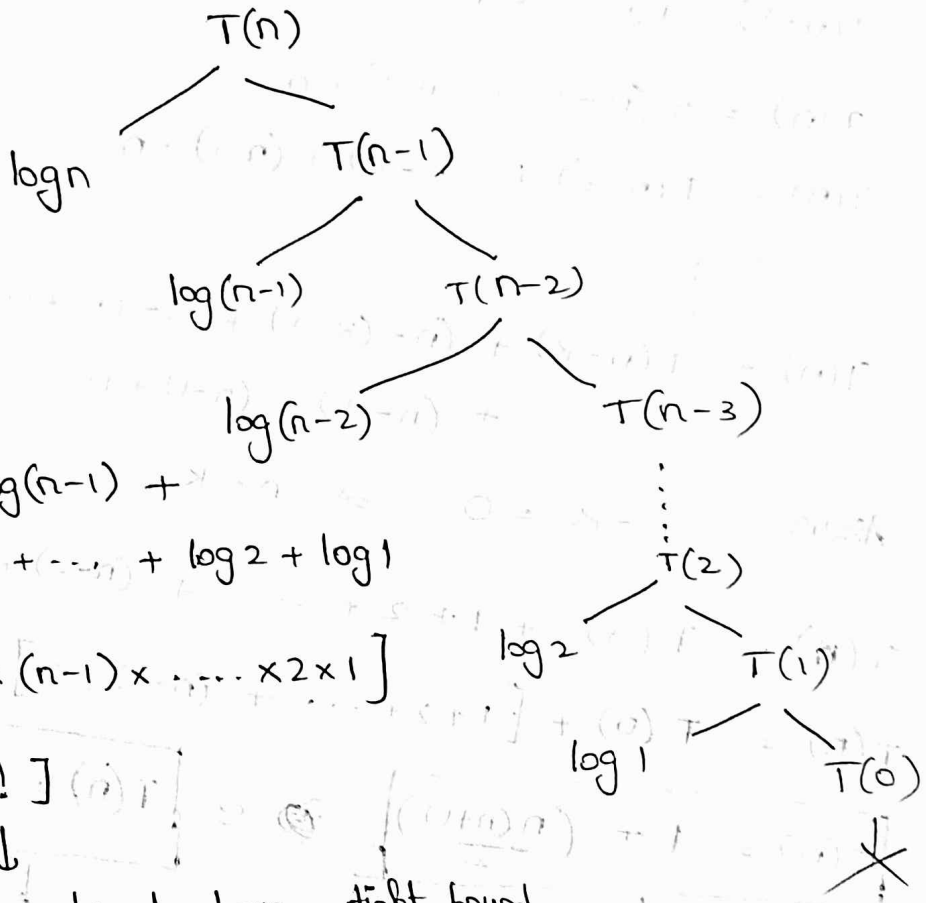
$\therefore$ Time Complexity $= O(n^2)$

Ex:

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

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

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



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

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

$$= \log [n!]$$

↓

this doesnot have tight bound for this. There is upper bound $\approx \log(n^n)$

$\Rightarrow$ Time taken $\simeq n \log n$

Time Complexity : $O(n \log n)$

Solving recurrence relation,

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

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

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

So on,

$$\Rightarrow T(n) = T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \log(n-(k-3)) + \dots + \log(n-1) + \log n$$

If $n = k$ (or) $n - k = 0$

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

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

$$= 1 + \log(n!) \rightarrow \text{no. of calls}$$

$\therefore$ Time Complexity $= O(n \log n)$

Logic :

$T(n) = T(n-1) + 1$ — $O(n)$

$T(n) = T(n-1) + n$ — $O(n^2)$

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

If the constant / addition term is : multiplied by n

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

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

$$T(n) = T(n-100) + n \approx \frac{n^2}{100} - O(n^2)$$

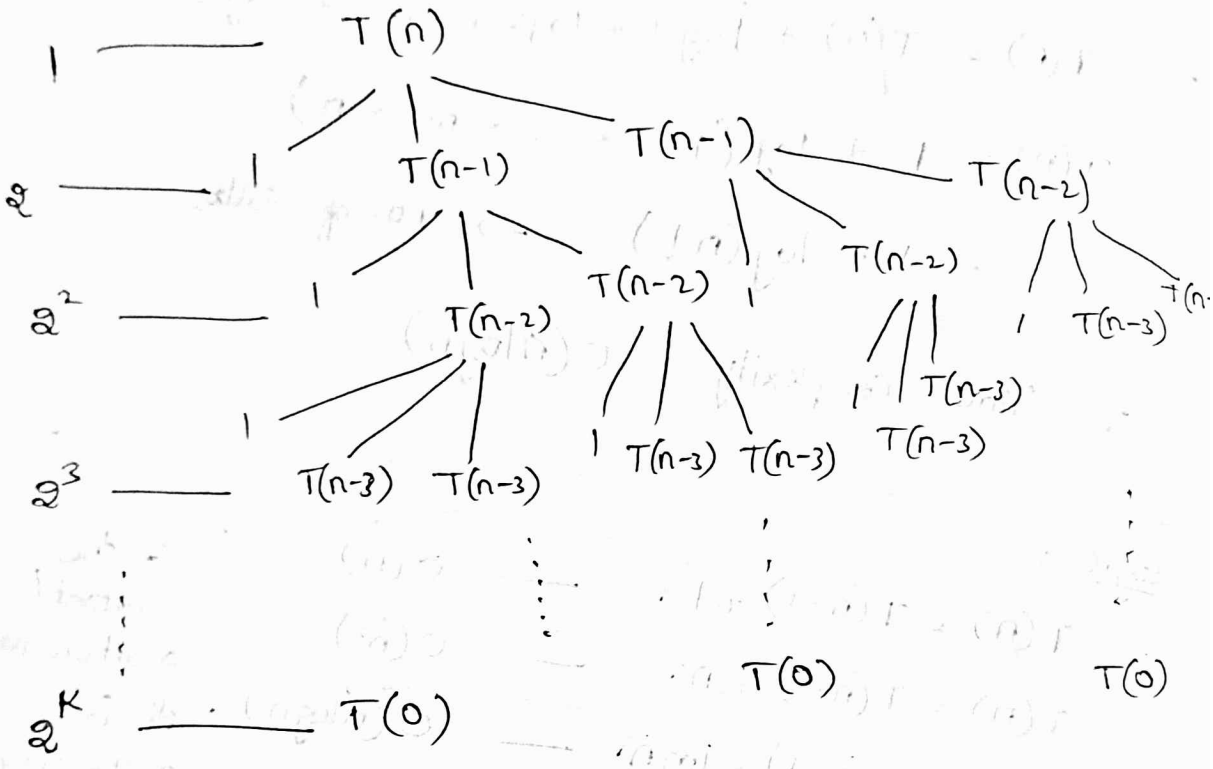
↓ no coefficients just 1 is the coefficient.

Ex :

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

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

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



Total time $= 1 + 2 + 2^2 + 2^3 + \cdots + 2^K$

(sum of terms in GP series)

$= 2^{K+1} - 1$ $\quad \left\{ a + ar + ar^2 + ar^3 + \cdots + ar^k = \dfrac{a(r^{k+1} - 1)}{r - 1} \right\}$

$\therefore$ Time complexity : $O(2^n)$

Solving using back substitution method,

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

$$T(n) = 2\left[2\,T(n-2) + 1\right] + 1$$

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

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

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

$$\vdots$$

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

Assume, $n-K=0$ (or) $n=K$

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

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

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

∴ Time Complexity : $O(2^n)$

---

*. MASTERS THEOREM FOR DECREASING FUNCTIONS :

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

General form of Recurrence relation,

* $T(n) = a \cdot T(n-b) + f(n)$ where

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

- if $a=1 \Rightarrow$ Time complexity $\approx O(n^{K+1}) \approx O(n * f(n))$
- if $a>1 \Rightarrow$ Time complexity $\approx O(n * f(n)) \approx O(n^K a^{n/b})$
- if $a<1 \Rightarrow$ Time $\approx O(n^K)$ (or) $O(f(n))$

This is Masters' Theorem.

In short,

if $a<1$ → Time : $O(f(n))$

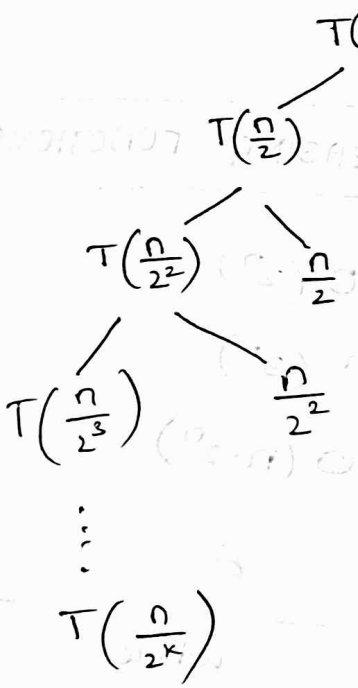
if $a=1$ → Time : $O(n f(n))$

if $a>1$ → Time : $O(f(n)\, a^{n/b})$

where $\boxed{T(n) = a \cdot T(n-b) + f(n)}$; $f(n) = O(n^k)$

*. Solve the dividing function, recurrence relation

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



total time =

$= n + \frac{n}{2} + \frac{n}{2^2} + \cdots + \frac{n}{2^k}$

$T(n) = n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k}\right]$

$= n \sum_{i=0}^{k} \frac{1}{2^i}$

$T(n) = n(1) \Rightarrow T(n) = n$

$\therefore$ [Time Complexity : $O(n)$]

Suppose, Given : $T(n) = \begin{cases} 1 & n=1 \\ 2T(\frac{n}{2}) + n & n>1 \end{cases}$

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

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

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{4}$

So, $T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$

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

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

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

$\vdots$

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

$T\left(\frac{n}{2^K}\right) = T(1)$ → smallest reduction

$\Rightarrow n = 2^K$ (or) $\boxed{K = \log n}$

$T(n) = 2^K \cdot T(1) + K \cdot n$

$= n \times 1 + n \cdot \log n = n\log n + n$

$\therefore$ [Time complexity : $O(n \log n)$]

---

*. MASTER'S THEOREM for DIVIDING FUNCTIONS :

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ → General form

where $a \geq 1$, $b > 1$, $f(n) = \Theta(n^k \log n^p)$

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

case 2 : if $\log_b a = k$ then : * [just $\Theta(f(n) \cdot \log n)$]

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

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

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

Case 3 : if $\log_b a < k$ then :

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

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

- **Examples** : {Case 1}

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

$T(n) = 4\,T\left(\frac{n}{2}\right) + 1$ —— $\Theta(n^2)$

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

$T(n) = 16\,T\left(\frac{n}{2}\right) + n^2$ —— $\Theta(n^4)$

- **Examples** : {Case 3}

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

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

$T(n) = 2\,T\left(\frac{n}{2}\right) + n^2 \log n$ —— $\Theta(n^2 \log n)$

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

$T(n) = 2\,T\left(\frac{n}{2}\right) + \frac{n^2}{\log n}$ —— $\Theta(n^2)$

*. Recurrence Relation of given Function :

```
void Test (int n)                          ---- T(n)
{
    if (n > 2)
    {
        Stmt;                              ---- 1
        Test(√n);                          ---- T(√n)
    }
}
```

∴ Recurrence relation $\Rightarrow T(n) = T(\sqrt{n}) + 1$

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

Solving $T(n) = T(\sqrt{n}) + 1$

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

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

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

⋮ K times

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

Assume that $n = 2^m$

$\Rightarrow T(2^m) = T\left(2^{\frac{m}{2^K}}\right) + K$

Assume $T\left(2^{\frac{m}{2^K}}\right) = T(2) \Rightarrow \boxed{K = \log_2 m}$

$n = 2^m \Rightarrow \log n = m \log 2 \Rightarrow m = \log_2 n$

$k = \log_2 m = \log(\log_2 n)$

$\therefore$ **Time Complexity = $\Theta(\log(\log_2 n))$** *Imp

*. BINARY SEARCH

- All the elements in the Array must be in Ascending or Descending order. (Sorted Array only)

```
int BinarySearch (A, n, key)
{
    l = 1, h = n;
    while (l ≤ h)
    {
        mid = (l+h)/2;
        if (key == A[mid])
            return mid;
        else if (key < A[mid])
            h = mid - 1;
        else
            l = mid + 1;
    }
    return 0;
}
```

height of Tree

- min Time = $O(1)$
- max Time = $\log_2 n$
- max T. Complexity = $O(\log_2 n)$

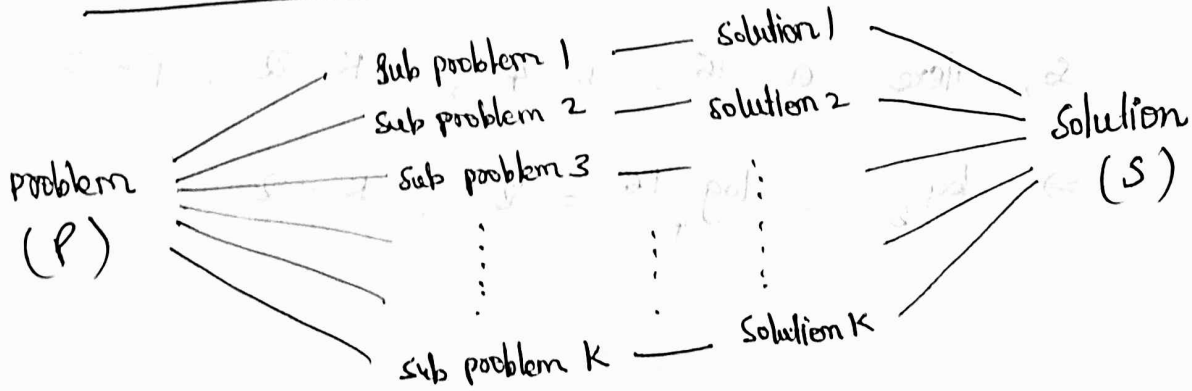
Recursive Algo :

```
Algorithm RecursiveBS (l, h, key)
{
    if (l == h)
    {
        if (A[l] == key)   return l;
        else               return 0;
    }
    else
    {
        mid = (l+h)/2;
        if (key == A[mid])   return mid;
        elseif (key < A[mid])
            return RecursiveBS (l, mid-1, key);
        else
            return RecursiveBS (mid+1, h, key);
    }
}
```

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

$\therefore$ **Time Complexity = $\Theta(\log n)$**

*. DIVIDE AND CONQUER :



- Strategy General method :

```
DAC(P)
{
    if ( small(P) )
    {
        S(P);
    }
    else
    {
        divide P into P1, P2, .... Pk ;
        Apply DAC(P1), DAC(P2) .... ;
        Combine ( DAC(P1), DAC(P2) , ... );
    }
}
```

- Problems under this topic : (P)

1) Binary Search
2) Finding Maximum & Minimum
3) Merge Sort
4) Quick sort
5) Strassen's Matrix Multiplica

---

* Practice Problems :

(a) $T(n) = 16\, T\left(\frac{n}{4}\right) + n^2$

General form : $T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta\left(n^k \log^p n\right)$

So, Here $a = 16$, $b = 4$, $K = 2$, $P = 0$

$\Rightarrow \log_b a = \log_4 16 = 2$, $K = 2$

As $\log_b a = K = 2 \rightarrow$ checking P

& $P = 0 > -1 \Rightarrow$ Time $= \Theta(n^K \log^{P+1} n)$

$\therefore$ Time Complexity $= \Theta(n^2 \log n)$

---

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

So, Here $a = 7$, $b = 3$, $K = 2$, $P = 0$

$\Rightarrow \log_b a = \log_3 7 \approx 1.771 < K = 2$

& $P = 0 \Rightarrow$ Time $= \Theta(n^K \log^P n)$

$\therefore$ Time Complexity $= \Theta(n^2)$

---

(9) $T(n) = 7T\left(\frac{n}{2}\right) + n^2$

So, Here $a = 7$, $b = 2$, $K = 2$, $P = 0$

$\log_b a = \log_2 7 \approx 2.807 > K = 2$

& $P = 0 \Rightarrow$ Time $= \Theta(n^{\log_b a})$

$\therefore$ Time Complexity $\simeq \Theta(n^3)$

---

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

So, Here $a = 2$, $b = 4$, $K = \frac{1}{2}$, $P = 0$

$\log_b a = \log_4 2 \approx 0.5 = K = \frac{1}{2}$

$\therefore$ Time Complexity $= \Theta(\sqrt{n} \log n)$

(6) $T(n) = T(\sqrt{n}) + 1$

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

Solving $T(\sqrt{n}) + 1 = T(n)$

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

$$T(n) = T\left(n^{\frac{1}{2^2}}\right) + 1 + 1 = T\left(n^{\frac{1}{2^2}}\right) + 2 \quad \text{--(2)}$$

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

⋮ K times

$$T(n) = T\left(n^{\frac{1}{2^K}}\right) + K$$ (Pattern by observing eq 1, 2, 3)

Assume that n is in powers of 2

i.e. $n = 2^m$

$$\Rightarrow T(2^m) = T\left(2^{\frac{m}{2^K}}\right) + K$$

Since we ~~reached~~ ~~reche~~ reached at final step i.e. at $n = 2 \Rightarrow T(n) = 1 = T(2)$

$$T\left(2^{\frac{m}{2^K}}\right) = T(2)$$

$$\Rightarrow K = \log_2 m \quad \& \quad m = \log_2 n$$

$$\Rightarrow \boxed{\text{Time Complexity} = \Theta(\log(\log n))}$$

(a) $T(n) = T(n-1) + n$ by iterative method

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

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

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

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

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

$$\vdots$$

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

Assuming, $n-K = 0 \Rightarrow n = K$

$$T(n) = T(0) + 1 + 2 + \ldots + (n-2) + (n-1) + n$$

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

$$T(n) \simeq \frac{n^2+n+2}{2} \Rightarrow \boxed{\text{Time Complexity} = O(n^2)}$$

MODULE - 2 :

* MERGE SORTING :

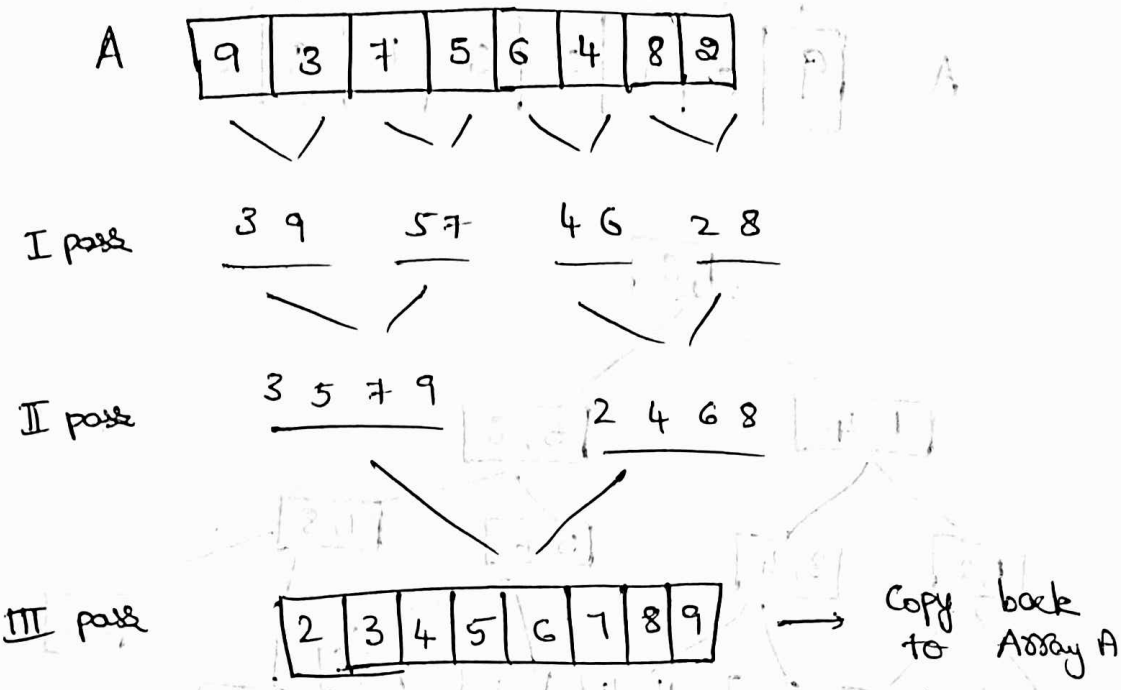
```
Algorithm Merge (A, B, m, n)
{
    i = 1 ; j = 1 ; k = 1 ;
    while ( i <= m && j <= m)
    {
        if (A[i] < B[j])
            c [k++] = A[i++] ;
        else
            c[k++] = B[j++] ;
    }
    for ( ; i <= m ; i++)
        c [k++] = A[i] ;
    for ( ; j <= n ; j++)
        c[k++] = B[j] ;
}
```

Time : $\Theta(m+n)$

- Merge Sort — recursive method
- 2-way Merge Sort — iterative method

} works same

Ex : Given an Array A = [9, 3, 7, 5, 6, 4, 8, 2]
Apply 2-way Merge Sort.



No. of passes = $\log n$ where no. of elements = n

$\therefore$ Time Complexity : $\Theta(n \log n)$

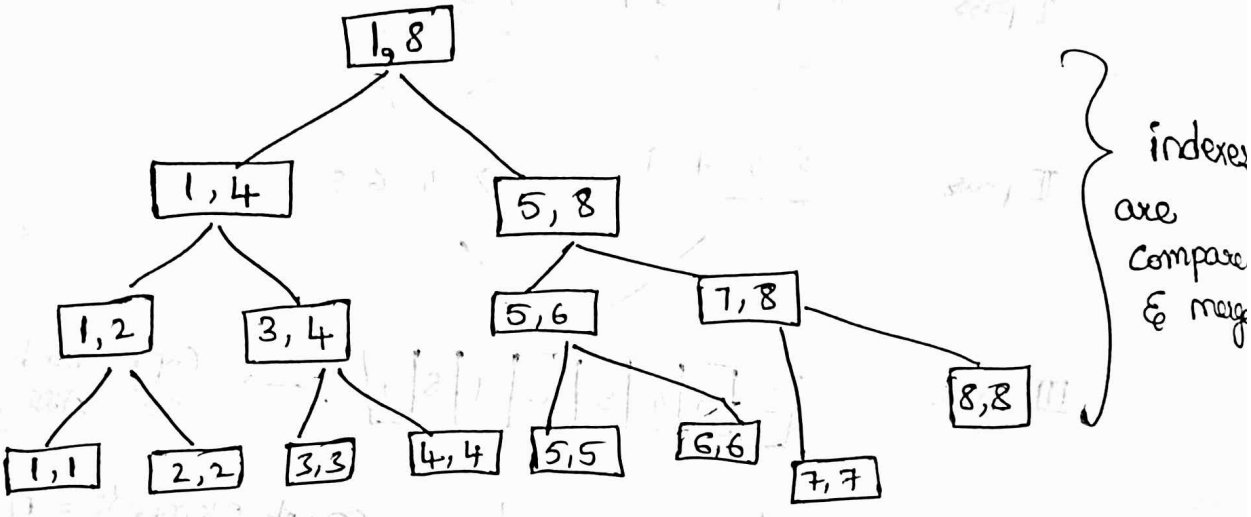
- Applying Merge sort for same Array A. (This is divide & Conquer)

```
Algorithm MergeSort (l, h)          —— T(n)
{
    if (l < h)                      —— 1
    {
        mid = (l+h)/2 ;             —— 1
        MergeSort (l, mid) ;        —— T(n/2)
        MergeSort (mid+1, h) ;      —— T(n/2)
        Merge (l, mid, h) ;         —— n
    }
}
```

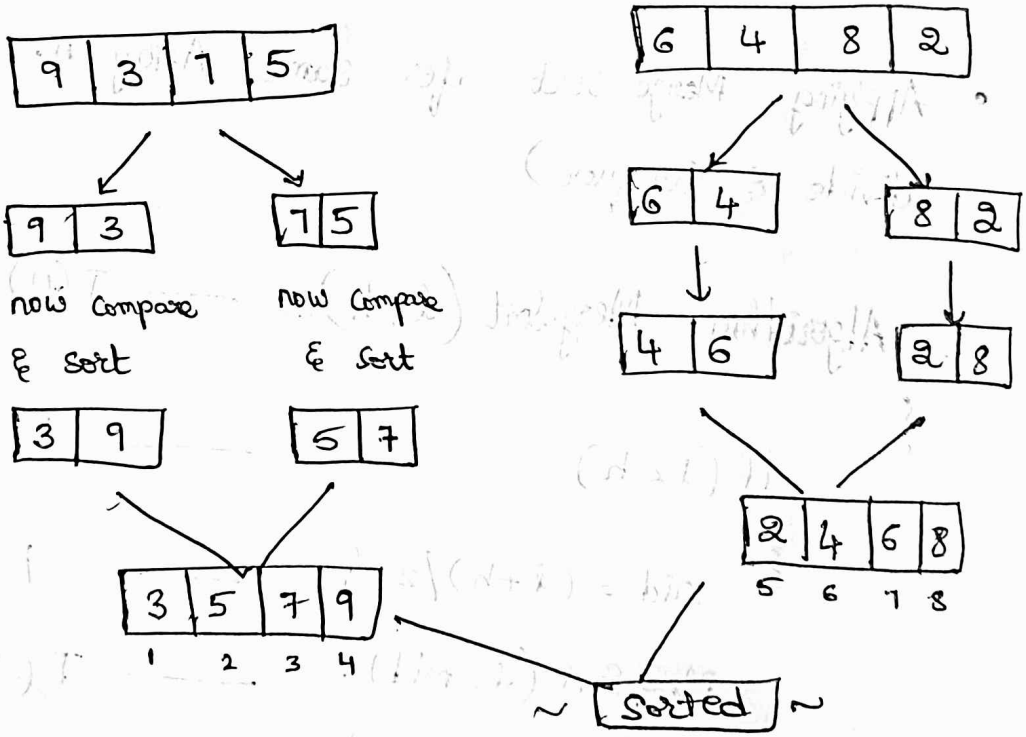
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

we know the time Complexity : $\Theta(n \log n)$

	1	2	3	4	5	6	7	8
A	9	3	7	5	6	4	8	2



Time taken = $\Theta(n \log n)$



Traversing is done - POST order Traversal

↓

left, right, root

Total space taken = $\Theta(n + \log n)$.

- Pros & Cons of Merge Sort :

Pros :

1. Large sized list.
2. suitable linked list.
3. Supports External sorting
4. It is stable (preserves order of duplicity).

Cons :

*1. Needs extra space
2. Not suitable for Arrays.
3. Use Insertion sort because its stable, suitable for merging.
4. It is Recursive (use memory's stack)

---

*. QUICK SORT : (It follows Divide - Conquer Algo).

```
Partition (l, h)
{
    pivot = A[l];
    i = l , j = h;
    while (i < j)
    {
        do
        {
            i++;
        } while (A[i] <= pivot)

        do
        {
            j++;
        } while (A[j] > pivot)

        if (i < j)  swap (A[i], A[j]);
    }
    swap (A[l], A[j]);
    return j;
}
```

```
QuickSort (l, h)
{
    if (l < h)
    {
        j = partition (l,h)
        QuickSort (l, j);
        QuickSort (j+1, h);
    }
}
```

Best Case

Time Complexity : $\Theta(n \log n)$

worst Case

$\Theta(n^2)$

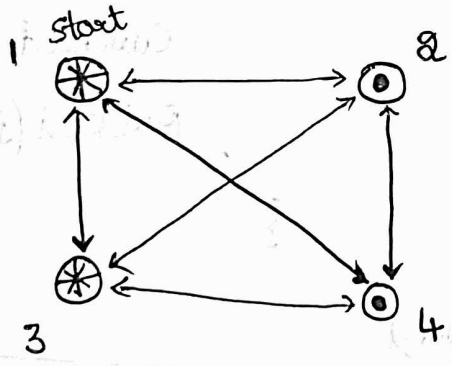
* BRUTE FORCE :

- General Problem Solving Technique that enumerating all possibilities for the solution and checking whether each candidate satisfies the problem statement.
- often, brute force algorithm required exponential time

Ex 1 : TRAVELING SALESMAN PROBLEM

⟹ Given a list of cities and the distances between each pair of cities, the shortest possible route that visits each city exactly once and returns to the origin city.

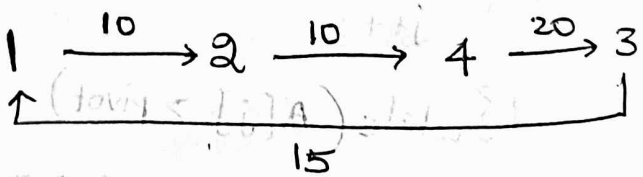
For Example,



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

choose a node that "minimizes" cost.

- By Greedy Approach :

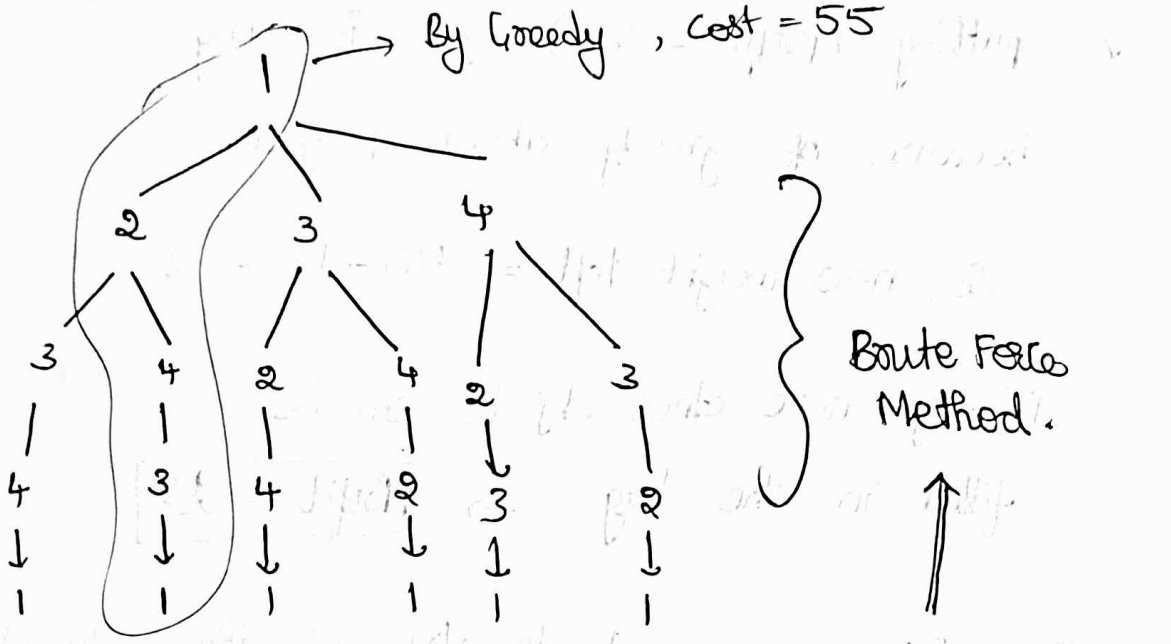


↓

total Cost = 10 + 10 + 20 + 15 = 55

- It is called Hamiltonian Cycle Problem.

Total Possibilities:



total possibilities = $O(n!) \approx O(n^n)$ → biggest

So, we apply Dynamic Programming to solve this.

Ex 2: KNAPSACK PROBLEM

⟹ Given a set of items, each with a weight and a value, determine the no. of each item to include in a collection so that the total weight is less than or equal to given limit and total value is as large as possible.

For Example,

objects	obj1	obj2	obj3
profit	25	24	15
weight	18	15	10

- Greedy about profit.

Sol: Given max weight = 20

✓ putting (profit = 25) obj 1 in Bag because of greedy about project.

So, new weight left = 20 − 18 = 2

Finally none other objects can be filled in the bag ⇒ **Profit = 25**

- Like this, we need to try all the different possibilities to get Maximum Profit ⟹ BRUTE FORCE

$$\text{Max profit} = 25 + \frac{2}{15} \times 24 \approx 28.2$$ (2 = 20 − 18)

Case 2: Greedy about weight (less weight)

Given max weight in Bag = 20

✓ putting (weight = 10) obj 3 in Bag

So, profit = 15

✓ putting (weight = $\frac{10}{15}$) obj 2 in Bag

So, profit = $15 + \frac{10}{15} \times 24 \approx 31$

optimal: Consider $\frac{\text{profit}}{\text{weight}}$ ratio.

Another Example,

- Given Knapsack $W = 10$, different products weights are $\{w_1, w_2, w_3, w_4\} = \{7, 3, 4, 5\}$ and corresponding profits are $\{v_1, v_2, v_3, v_4\} = \{42, 12, 40, 25\}$

Apply Brute Force method.

Sol:

Subset	Total Weight	Total Value
$\emptyset$	0	0
$\{1,2,3,4\}$	19	✗
$\{1\}$	7	42
$\{2\}$	3	12
$\{3\}$	4	40
$\{4\}$	5	25
$\{1,2\}$	10	54
$\{1,3\}$	11	✗
$\{1,4\}$	12	✗
$\{2,3\}$	7	52
$\{2,4\}$	8	37
$\{3,4\}$	9	65
$\{1,2,3\}$	14	✗
$\{1,2,4\}$	15	✗
$\{1,3,4\}$	16	✗
$\{2,3,4\}$	12	✗

- By spending $54, purchased 10 kg weight

So, Maximum value = $54 ⇔ weight = 1
minimum weight = 7 ⇔ Value = $

- Best case, ratio1: $\frac{54}{10} = 5.4$ ✓ suitable.
- another case, ratio2: $52/7 = 7.4$ ✗ not

- In Knapsack Problem, By using Brute Force approach

The no. of subsets of an n-element set is 2^n
the search leads to a $\Omega(2^n)$ algorithm, wh
is not based on the generation of individual subset

*. ASSIGNMENT PROBLEM :

Ex: There are n people who need to be assigned to n jobs, one person per job. The cost of assigning person i to job j is C[i,j]. Find an assignment that minimizes the total cost.

	job0	job1	job2	job3
person0	9	2	7	8
person1	6	4	3	7
person2	5	8	1	8
person3	7	6	9	4

So, $C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$

Assignment	Total Cost
1, 2, 3, 4	$9 + 4 + 1 + 4 = 18$
1, 2, 4, 3	$9 + 4 + 8 + 9 = 30$
1, 3, 2, 4	$9 + 3 + 8 + 4 = 24$
1, 3, 4, 2	$9 + 3 + 8 + 6 = 26$
1, 4, 2, 3	$9 + 7 + 8 + 9 = 33$
1, 4, 3, 2	$9 + 7 + 1 + 6 = 23$ etc ...
⋮	⋮

---

*. IMPORTANT EXAMPLES FOR CAT-1:

① Find Θ notation for function $f(n) = \frac{1}{3}n^3 + \frac{n^2}{2} + \frac{n}{6}$

Sol: The function $f(n) = \Theta(g(n))$ iff there exists a positive constant c_1, c_2 and n_0 such that

$$\boxed{0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)} \quad \forall\ n > n_0$$

Given $f(n) = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$

$$= n^3 \left[\frac{1}{3} + \frac{1}{2n} + \frac{1}{6n^2} \right]$$

$f(n)$ has highest exponent as n^3

$$\Rightarrow \left(\frac{1}{3} + \frac{1}{2n} + \frac{1}{6n^2}\right) n^3 \leq 1 \cdot n^3 \leq c_2 \cdot n^3$$

$$\left(\frac{1}{3} + \frac{1}{2n} + \frac{1}{6n^2}\right) n^3 \leq \left(\frac{1}{3} + \frac{1}{2} + \frac{1}{6}\right) n^3 \leq c_2 \cdot n^3$$

Assume $g(n) = n^3$

We know that, $f(n) \leq c_2 \cdot g(n)$

$$\Rightarrow c_2 \cdot n^3 \geq \left(\frac{1}{3} + \frac{1}{2} + \frac{1}{6}\right) n^3 \Rightarrow c_2 = 1$$

$$\therefore f(n) = \Theta(g(n)) = \Theta(n^3)$$

---

② Given, $f(n) = \frac{1}{2} n^2 - 3n$. Find Θ notation.

Sol: If $f(n)$ & $g(n)$ are two functions such that $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ then $f \in \Theta(g(n))$

Given, $f(n) = \frac{n^2}{2} - 3n$. $f(n)$ has highest exponent as n^2

So, Assume $g(n) = n^2$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{\frac{n^2}{2} - 3n}{n^2} = \frac{n^2\left(\frac{1}{2} - \frac{3}{n}\right)}{n^2}$$

$$= \lim_{n \to \infty} \left(\frac{1}{2} - \frac{3}{n}\right) \approx \frac{1}{2} - \frac{3}{(\infty)} = \frac{1}{2}$$

Here $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\frac{1}{2}n^2 - 3n}{n^2} = \frac{1}{2}$ is constants

So, we can say $f(n) \in \Theta(g(n))$

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

---

(3) Find Θ notation for given exponential function

$f(n) = 3 \times 2^n + 4n^2 + 5n + 3$

Sol: $\lim_{n\to\infty} \frac{3\times 2^n + 4n^2 + 5n + 3}{2^n}$ (Assume $g(n) = 2^n$)

$= \lim_{n\to\infty} \frac{2^n\left(3 + 4\frac{n^2}{2^n} + 5\frac{n}{2^n} + \frac{3}{2^n}\right)}{2^n} = \lim_{n\to\infty}\left(3 + \frac{4(2n)}{2^n \log_e 2} + \frac{5}{2^n \log_e} + \frac{0}{2^n \log_e^2}\right)$

$= \lim_{n\to\infty}\left(3 + \frac{8}{(2^n \log_e 2)^2} + \frac{0}{(2^n \log_e 2)^2}\right)$

$= \lim_{n\to\infty}\left(3 + \frac{0}{\cdots}\right) = 3$ (Constant)

So, we can say that $f(n) \in \Theta(g(n))$

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

---

(4) If $f(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_m n^m$ is any polynomial of degree m or less than, $f(n) = \Theta(n^m)$.

Sol: Consider the limit of $\frac{f(n)}{n^m}$ as $n \to \infty$.

Since this equals $a_m + \frac{a_{m-1}}{n} + \cdots + \frac{a_1}{n^{m-1}} + \frac{a_0}{n^m}$

$$\lim_{n\to\infty} \frac{f(n)}{n^m} = \lim_{n\to\infty} \left[a_m + \frac{a_{m-1}}{n} + \cdots + \frac{a_1}{n^{m-1}} + \frac{a_0}{n^m} \right]$$

Theorem states that,

$$\boxed{\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0, \text{ finite} \Rightarrow f(n) \in \Theta(g(n))}$$

Here, the $\lim_{n\to\infty} \frac{f(n)}{n^m} = a_m$ (constant)

$\therefore f(n) = \Theta(n^m)$ where n^m is $g(n)$

---

⑤ Show that for any real constant a, b where $b > 0$; $(n+a)^b = \Theta(n^b)$

Sol: By definition of Θ, we need to find three constants c_1, c_2 & n_0 such that $0 \leq c_1 \cdot n^b \leq (n+a)^b \leq c_2 \cdot n^b \quad \forall \quad n \geq n_0$

$$0 \leq c_1 \cdot n^b \leq (n+a)^b \leq c_2 \cdot n^b$$

Since $n \geq |a|$, we have

$$n+a \leq n+|a| \leq 2n$$

Since $n \geq 2|a|$ (i.e. $|a| \leq \frac{1}{2}n$), we have

$$n+a \geq n - |a| \geq \frac{1}{2} n$$

Thus, when $n \geq 2|a|$, we have

$$0 \leq \frac{1}{2} n \leq n+a \leq 2n$$

Since b is positive constant,

$$0 \leq \left(\frac{1}{2} n\right)^b \leq (n+a)^b \leq (2n)^b$$

$$0 \leq \left(\frac{1}{2}\right)^b \cdot n^b \leq (n+a)^b \leq (2^b)\, n^b$$

So, $c_1 = \left(\frac{1}{2}\right)^b$, $c_2 = 2^b$, $n_0 = 2|a|$. satisfies the above definition on comparision.

---

(6) Is $2^{n+1} = O(2^n)$?

Sol: Let $f(n) = 2^{n+1} = 2^n \cdot 2 = 2(2^n)$

Here, $f(n)$ is represented as $c \cdot g(n)$

i.e. $2^{n+1} = 2 \cdot 2^n \leq c \cdot 2^n$ where $c \geq 2$

So, $2^{n+1} = O(2^n)$ is TRUE.

Is $2^{2n} = O(2^n)$?

Sol: $2^{2n} = 2^n \cdot 2^n$. Suppose $2^{2n} = O(2^n)$. Then there is constant $c > 0$ such that $c > 2^n$.

Since 2^n is unbounded, no such c exist.

So, $2^{2n} \neq O(2^n)$ (false)

④ show that $n \log n \in \Theta(\log n!)$

Sol:

$$n! = (n-0)(n-1)(n-2) \cdots (n-(n-1))$$

$$= n\left(1-\frac{0}{n}\right) \cdot n\left(1-\frac{1}{n}\right) \cdot n\left(1-\frac{2}{n}\right) \cdots n\left(1-\frac{n-1}{n}\right)$$

$$= n^n \left(1-\frac{0}{n}\right) \cdot \left(1-\frac{1}{n}\right) \cdot \left(1-\frac{2}{n}\right) \cdots \left(1-\frac{n-1}{n}\right)$$

$$= n^n \prod_{k=0}^{n-1} \left(1-\frac{k}{n}\right)$$

$$\log(n!) = \log\left(n^n \prod_{k=0}^{n-1} \left(1-\frac{k}{n}\right)\right)$$

$$= \log(n^n) + \log\left(\prod_{k=0}^{n-1} \left(1-\frac{k}{n}\right)\right)$$

$$= n \log n + \log\left(\prod_{k=0}^{n-1} \left(1-\frac{k}{n}\right)\right)$$

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

We know that, If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$ *IMP

→ Here $f(n) = \log(n!)$, $g(n) = n \log n$

So, we obtained $f(n) = \Theta(g(n))$ as

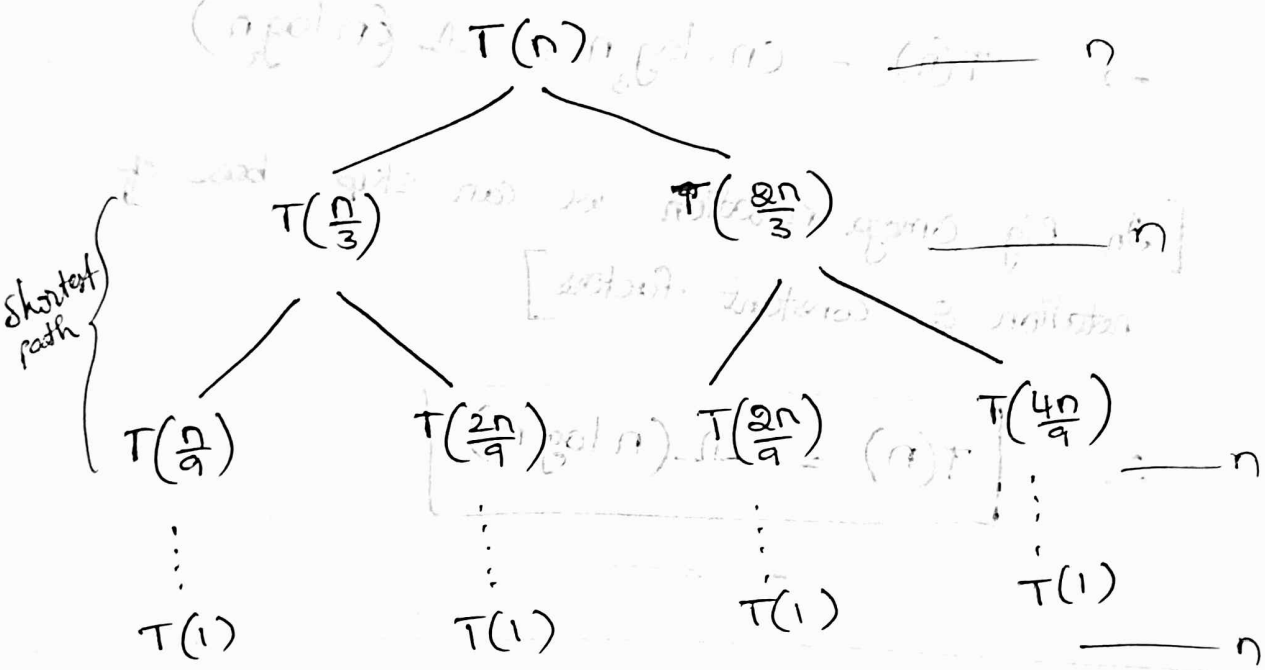
$$\log(n!) = \Theta(n \log n)$$

Also, $n \log n = \Theta(\log n!)$

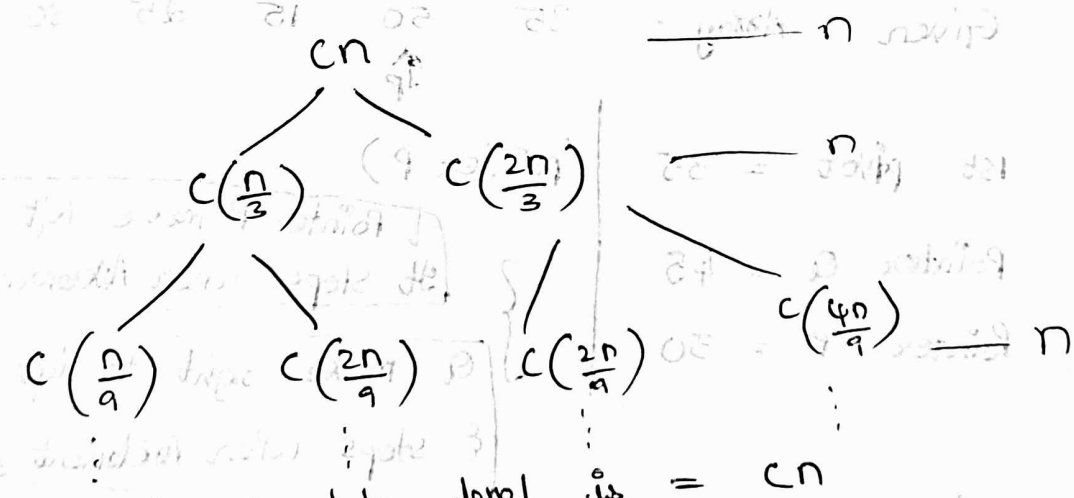
$$\Rightarrow n \log n \in \Theta(\log n!)$$

(10) $T(n) = T\left(\frac{n}{3}\right) + 2T\left(\frac{2n}{3}\right) + cn$ where c is a constant is $\Omega(n \log n)$ by appealing to a recursion Tree method.

Sol: Given, $T(n) = T\left(\frac{n}{3}\right) + 2T\left(\frac{2n}{3}\right) + c \cdot n$



And cn value on recursive tree



Sum of each complete level is = cn

- Elements from shortest path are being divided by 3. so length of this path will be equal to $\log_3 n$.

So, if number of complete levels of recursion tree of shortest path is equal to $\log_3 n$ that means cost of algorithm for this path is (let) $T(n)$.

$$\Rightarrow \quad T(n) = cn \cdot \log_3 n = \Omega(n \log_3 n)$$

[In Big Omega notation we can skip base of notation & constant factors]

$$\therefore \quad \boxed{T(n) = \Omega(n \log n)}$$

---

*. QUICK SORT Example :

Given Array : 35 50 15 85 80 20 90 45

(pointer P) (pointer Q)

1st pivot = 35
Pointer Q = 45
Pointer P = 50

↓ Pointer P moves left to right
& stops when A[element] ≥ pivot

Q moves right to left
& steps when A[element] ≤ pivot

P pivot
50 > 35 → stop

Q pivot
45 ≤ 35 → false so move towards left

Q pivot
90 ≤ 35 → false so move towards left

Q pivot

20 < 35 → True → stops

P & Q are stopped. So now swap P & Q

(35)	20	15	25	80	50	90	45	+∞
pivot	↑				↑			
	P				Q			

Again, P pivot

20 > 35 → No continue move right

P pivot

15 > 35 → No continue move right

P pivot

80 > 35 → Yes, so stop

Now, Q will move

Q pivot

50 < 35 → false so move towards left

Q pivot

80 < 35 → false so move towards left

Q pivot

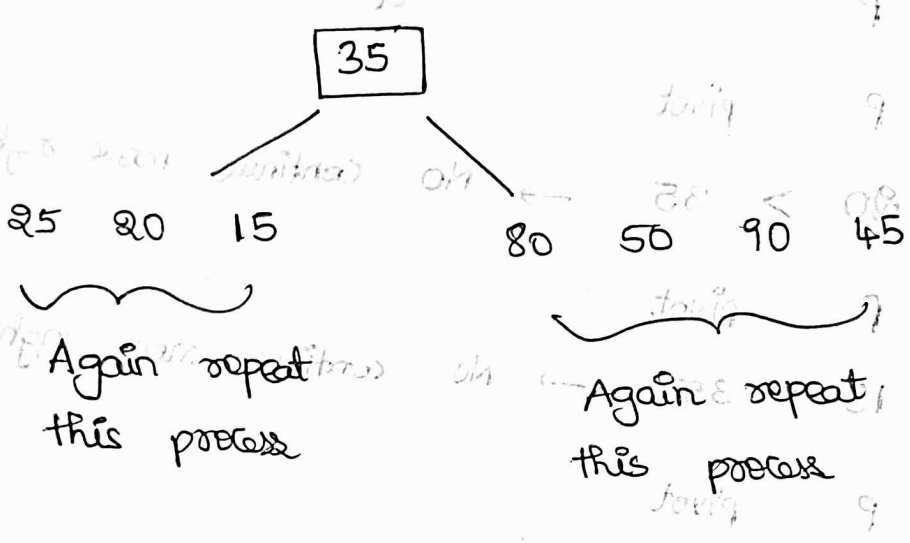
25 < 35 → True ; Q stops

Checking P & Q crossed each other

replace pivot element with Q

25 20 15 (35) 80 50 90 45

↑ Now pivot is at right place ⇒ elements greater than pivot are towards RHS and elements lesser than pivot are towards LHS.



(25) 20 15 +∞
pivot ↑ ↑
p q

p pivot
20 > 25 → No, continue

p pivot
15 > 25 → No, continue

p pivot
+∞ > 25 → stop

q pivot
15 < 25 → stop

If p & q Crossed

So, exchange pivot with q

15	20	25

LHS of [pivot 35] sorted

Now for RHS of 35 { (80) 50 90 45 }
pivot ↑ ↑
p q

P pivot

50 > 80 → No continue

P pivot

90 > 80 → Yes stop

Q pivot

45 < 80 → yes stop

P & Q didn't cross

So, interchange P & Q

⇒ (80) 50 45 90 +∞

pivot

P Q

P pivot

90 > 80 → yes stop

Q pivot

45 < 80 → yes stop

P & Q crossed each

So, interchange

pivot element & Q

⇒

45	50	80	90

Sorted array RHS

So, finally the sorted array is,

Array:

15	20	25	35	45	50	80	90

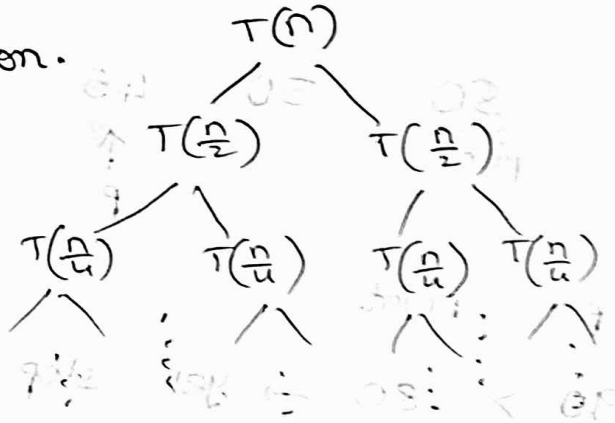
$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$ → 1st time to scan entire array

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

Average case Time complexity : $\Theta(n \log n)$

Analysis of Quicksort :

- Depends on partition of array.
- Best case is pivot is middle & the array is balanced partition.

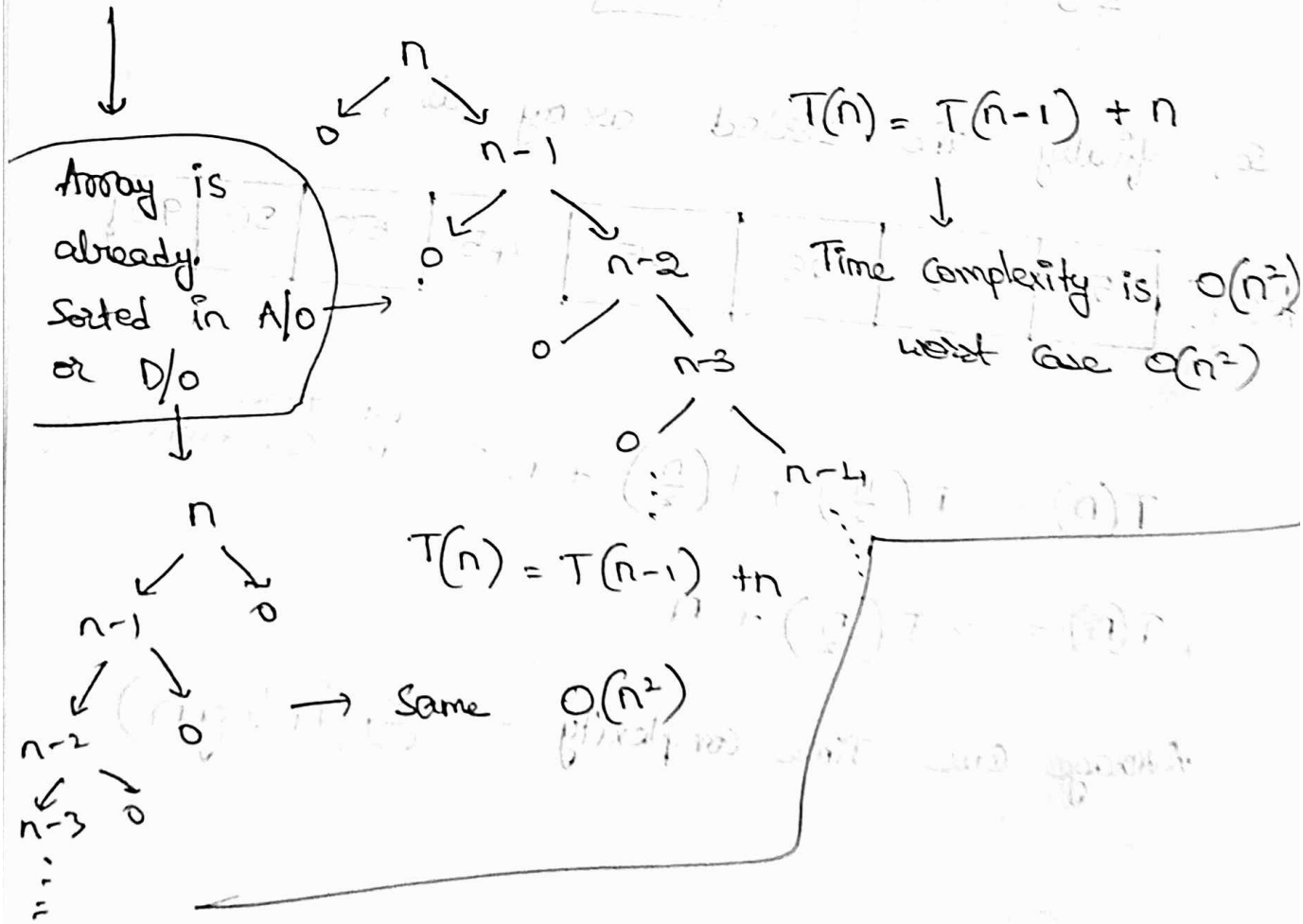


Best / Average

$O(n \log n)$

- Worst Case Time Complexity :

In worst case, there will be no elements to LHS of pivot because pivot is already at right place & its the smallest.



(*Q)

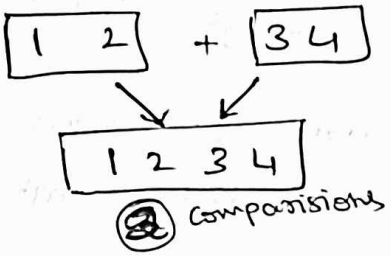
The average no. of comparisions performed by merge sort of algorithm, in merging 2 sorted lists of length 2 is,

$n \log_2 n$

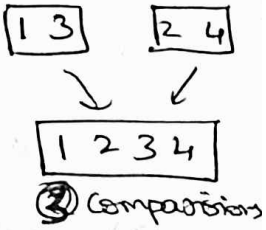
$4 \times \log_2 4$

$4 \times 2 = 8$

(not all cases)



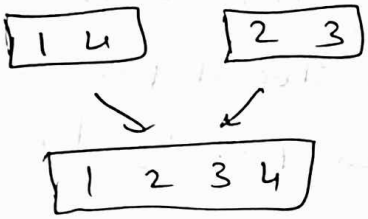
(or)



one list has = m elements

another list has = n elements

(or)



3 comparisions

Total comparisions = $m+n-1$

$2+3+3 = 8$

$\frac{8}{3} = 2.67$

(8 → all comparisions, 3 → all cases)

Pivot – first / last → general Quick sort — less elements

Pivot – middle-most → randomized Q.S — more elements

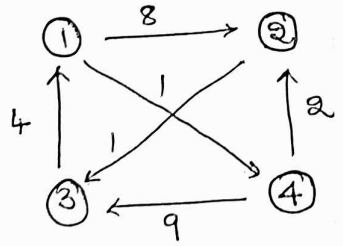
MODULE -3 : DYNAMIC PROGRAMMING

- Dynamic Programming approach is similar to divide & conquer in breaking down the problem into smaller & yet smaller possible sub-problems.
- But unlike, divide & conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are used for similar or overlapping sub-problems. (suitable for optimization Problem).
- Examples : Fibonacci number series, knapsack problem, Tower of Hanoi; All pair shortest path by Floyd-Warsha Project scheduling, shortest path by Dijkstra

*. FLOYD-WARSHALL ALGORITHM :

- It is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph.
- This algorithm follows the dynamic programming approach to find the shortest path.

Ex :



$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & (12) & 0 & (5) \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & (9) & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & (3) & 0 \end{bmatrix} \end{array}$$

$$D^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

∴ All pair shortest path is : $\begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$.

$$A^k[i,j] = \min \left\{ \begin{array}{l} A^{k-1}[i,j] \\ A^{k-1}[i,k] + A^{k-1}[k,j] \end{array} \right\}$$

Time Complexity :

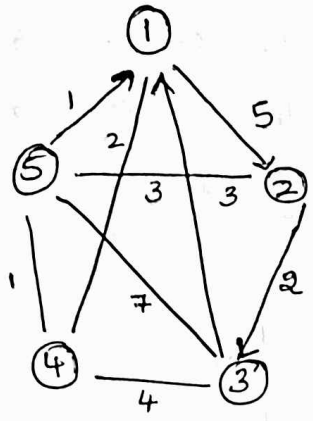
```
function Floyd (L[1...n, 1....n]): array[1...n, 1...n]
    array D[1....n; 1,....n]                          ——— O(1)
    D ← L                                             ——— O(n)
    for K ← 1 to n do                                 ——— O(n)
        for i ← 1 ton do
            for j ← 1 to n do                         ——— O(n)
                D[i,j] ← min (D[i,j], D[i,K] + D[K,j])
    return D
```

We apply this algorithm n times, each time choosing a different as the source.

So, the total Computation time is,

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

∴ Best, average, worst case Time Complexity = $O(n^3)$



$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & \infty & 0 & \infty & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & 8 & 0 & 5 & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & \infty & 3 & 0 \end{bmatrix} \end{array}$$

$$D_2 = \begin{bmatrix} 0 & 5 & 7 & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & 8 & 0 & 5 & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 5 & 7 & 2 & 14 \\ 5 & 0 & 2 & 7 & 9 \\ 3 & 8 & 0 & 5 & 7 \\ 7 & 12 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 5 & 6 & 2 & 3 \\ 5 & 0 & 2 & 7 & 8 \\ 3 & 8 & 0 & 5 & 6 \\ 7 & 12 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$

$$D_5 = \begin{bmatrix} 0 & 5 & 6 & 2 & 3 \\ 5 & 0 & 2 & 7 & 8 \\ 3 & 8 & 0 & 5 & 6 \\ 2 & 4 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$

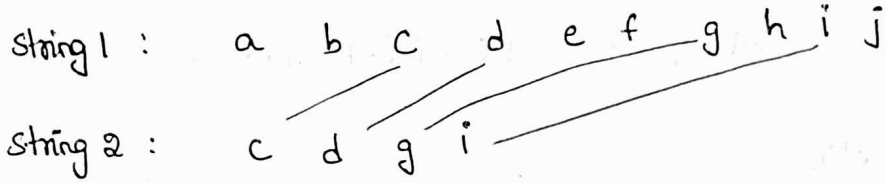
∴ Time Complexity : $O(n^3)$

Space Complexity : $O(n^2)$

*. LONGEST COMMON SUBSEQUENCE (LCS) :

- The Logest common Subsequence (LCS) is defined as the longest subsequence that is common to all the given sequences, provided that the elements of the subsequence are not required to occupy consecutive positions within the original sequences.

Ex :



longest subsequence is : c d g i

string 1 : a b c d e f g h i j

string 2 : e c d g i

starting with e : e g i
starting with c : c d g i } longest is : c d g i

string 1 : a b d a c e

string 2 : b a b c e

starting with b : b a c e
starting with a : a b c e } both are considered

next character should be checked only after the pointer/position of previous character going back is not considered/not allowed.

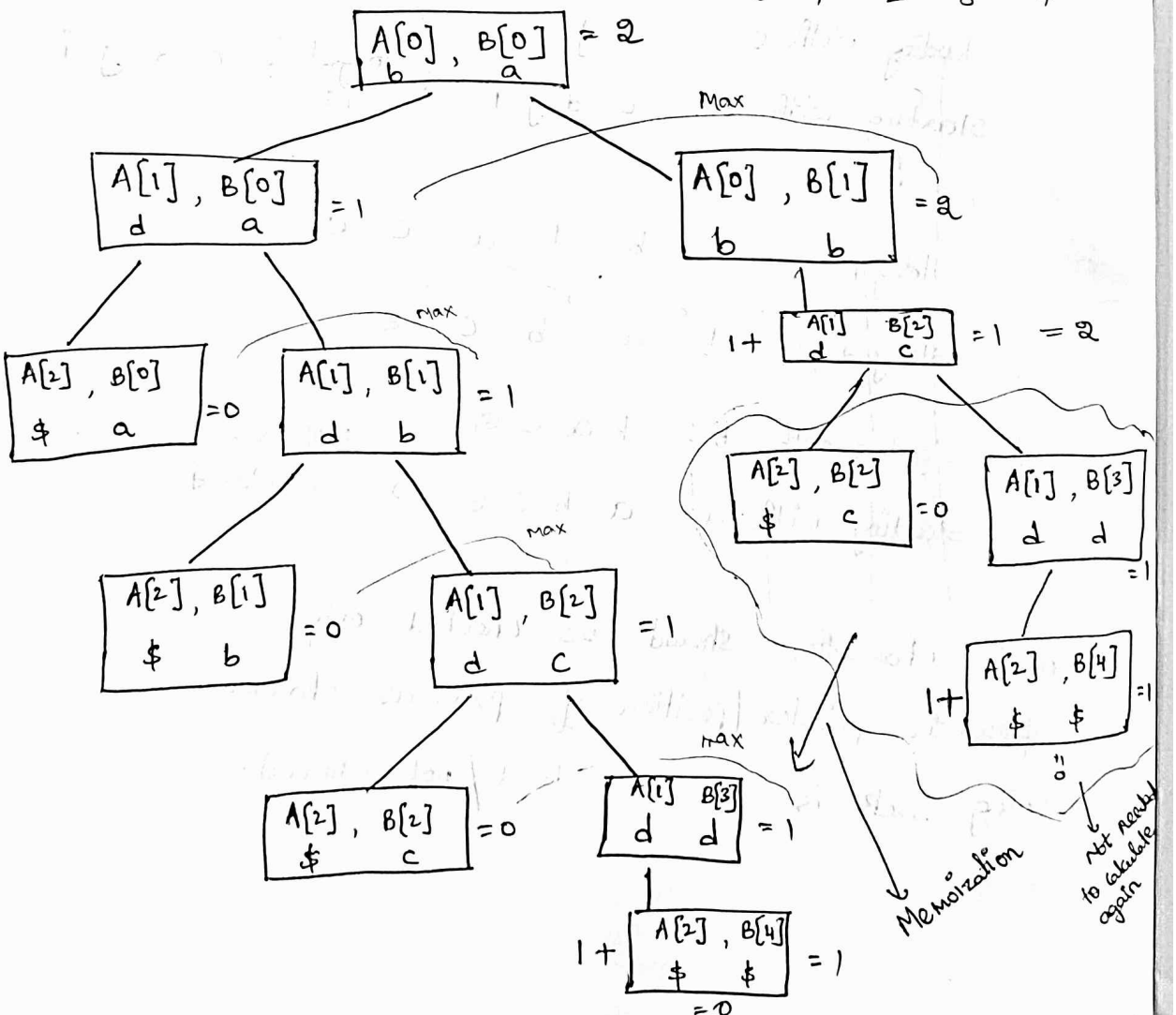
*- LCS using recursion :

```
int LCS(i, j)
{
    if (A[i] = '$' || B[j] = '$')
        return 0 ;
    else if ( A[i] == B[j] )
        return 1 + LCS (i+1, j+1) ;
    else
        return max ( LCS(i+1, j) , LCS (i, j+1)) ;
}
```

Tracing the algorithm with :

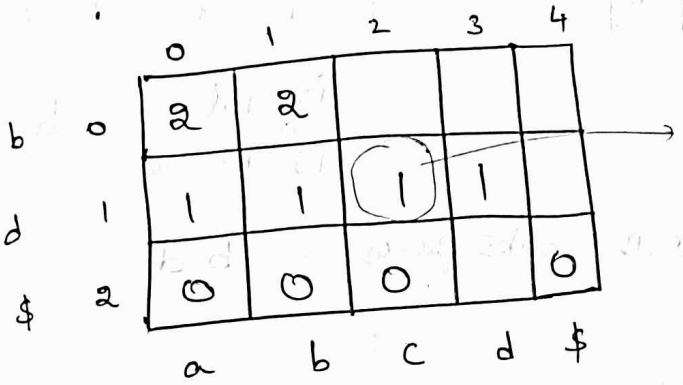
	0	1	2
A	b	d	$

	0	1	2	3	4
B	a	b	c	d	$



* Memoization : Here, the recursion may repeat the same that was already computed other side.

→ So, Dynamic Programming always stores the result of each recursive call through Top to bottom computation. This process of storing the values at each recursive call is "Memoization".



already present → Memoization

So, again no need to compute for the RHS sub tree

A[...] → m
B[...] → n

} time is O(m x n) *

- By recursion, time complexity : $O(2^n)$
- By memoization, time complexity : $O(m \times n)$ *IMP

*. LCS using Dynamic Programming : (IMP)

Same way derived

Time Complexity : m x n

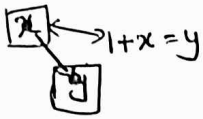
if (A[i] == B[j]

LCS[i,j] = 1 + LCS[i-1, j-1]

else

LCS[i,j] = max(LCS[i-1, j], LCS[i, j-1])

if matching → 1 + diagonal (above) element



not matching →  maximum of both = y

		1	2
A		b	d

		1	2	3	4
B		a	b	c	d

		a	b	c	d
	0	0	0	0	0	0
b	1	0	max(0,0)	1+0=1	max(1,1)=1	1
d	2	0	0	1	1	2

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	1	1	1
2	0	0	1	1	2

diagonal is present : b d

∴ Longest common subsequence = b d

Ex2 :

str1 : stone

str2 : longest

		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	0	1	1
t	2	0	0	0	0	0	0	1	2
o	3	0	0	1	1	1	1	1	2
n	4	0	0	1	2	2	2	2	2
e	5	0	0	1	2	2	3	3	3
			l	o	n	g	e	s	t

diagonal is : o n e

length = 3

∴ Longest common subsequence = o n e .

length of LCS = 3 .

Ex 3 :

A = < B D C A B A >

B = < A B C B D A B >

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

Longest Common Subsequence : B C B A ; Length of LCS = 4

---

*. KNAPSACK PROBLEM :

- You are given the following :
  1) A knapsack with limited weighted capacity.
  2) Few items each having some weight & value.
- The problem states that -
  which items should be placed into the knapsack such that:
  1) The value of / profit obtained is maximum.
  2) The weight limit of the knapsack doesnot exceed.

Knapsack problem has 2 variants :

- Fractional Knapsack problem.
- 0/1 knapsack problem.

• 0/1 KNAPSACK PROBLEM :

Consider -

Knapsack weight Capacity = w

No. of items each of Weight & value = n

Total solution = 2^n

Example :

Brute Force = ~~$O(2^n)$~~ $O(2^n)$

Dynamic Programming = $O(n \cdot m)$ — n: no. of objects, m: weight

Profit, $P = \{1, 2, 5, 6\}$ $\quad m = 8$

Weight, $W = \{2, 3, 4, 5\}$ $\quad n = 4$

P_i	W_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3 (max)	3	3	3
5	4	3	0	0	1	2	5	5 (max)	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

Formula *

$$V[i, w] = \max\{ V[i-1, w],\ V[i-1, w - w[i]] + P[i] \}$$

- Always write the weights in ascending order on the left of the table before solving.

	x_1	x_2	x_3	x_4	
Sequence of decision	0	1	0	1	$8-6=2$ $2-2=0$

$\therefore$ x_2, x_4 are put in knapsack

Example:

- A thief enters a house for robbing it. He can carry a maximal weight of 5 kg into his bag. There are 4 items in the house with the following weights and values. What items should thief take if he either takes the item completely or leaves it completely?

Item	Weight	Value
Mirror	2	3
Silver Nugget	3	4
Painting	4	5
Vase	5	6

Sol: Given Knapsack capacity (w) = 5 kg

No. of items (n) = 4

p_i	w_i		0	1	2	3	4	5	
0		0	0	0	0	0	0	0	$x_1\ x_2\ x_3\ x_4$
3	2	✓1	0	0	3	3	3	3	1 1 0 0
4	3	✓2	0	0	3	4	4	7	$\therefore x_1, x_2$
5	4	3	0	0	3	4	5	7	are put
6	5	4	0	0	3	4	5	(7)	in Knapsack

Program / Algorithm :

[0/1 Knapsack fails for Greedy approach]

```
main ()
{
    int  P[5] = { 0,1,2,5,6 } ;
    int  wt[5] = {0,2,3,4,5 } ;
    int  m = 8 ;  int  n = 4 ;
    int  K[5][9] ;

   for (int i = 0 ;  i <= n ;  i++)
   {
      for (int w = 0 ;  w <= m ;  w++)
      {
          if (i == 0  ||  w == 0)
              K[i][w] = 0;
          else if ( wt[i] <= w)
          K[i][w] =  max[ P[i] + K[i-1] [w - wt[i]],
                          K[i-1][w] ) ;
          else
              K[i][w] = K[i-1][w] ;
      }
   }
      print ( K[n][w] )
}
```

By Dynamic Programming approach

Time Complexity : $O(n \cdot w)$

Space Complexity : $O(w)$

Brute Force → Time Complexity : $O(2^n)$

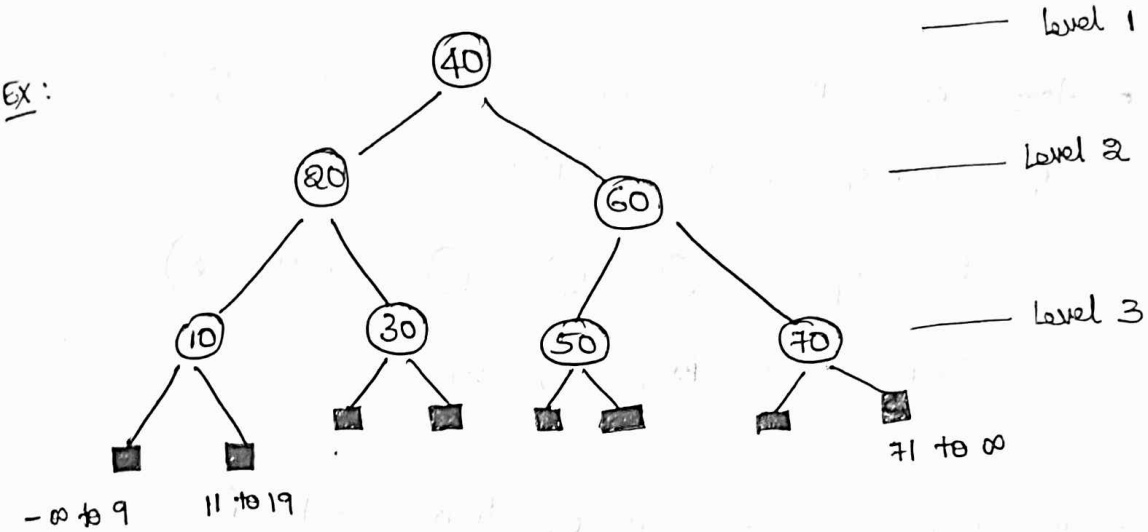
*. OPTIMAL BINARY SEARCH TREE :

- If element is found / not found while searching in BST, $\boxed{\text{no. of } \underset{max}{\text{Comparisions}} = \text{height of BST}}$

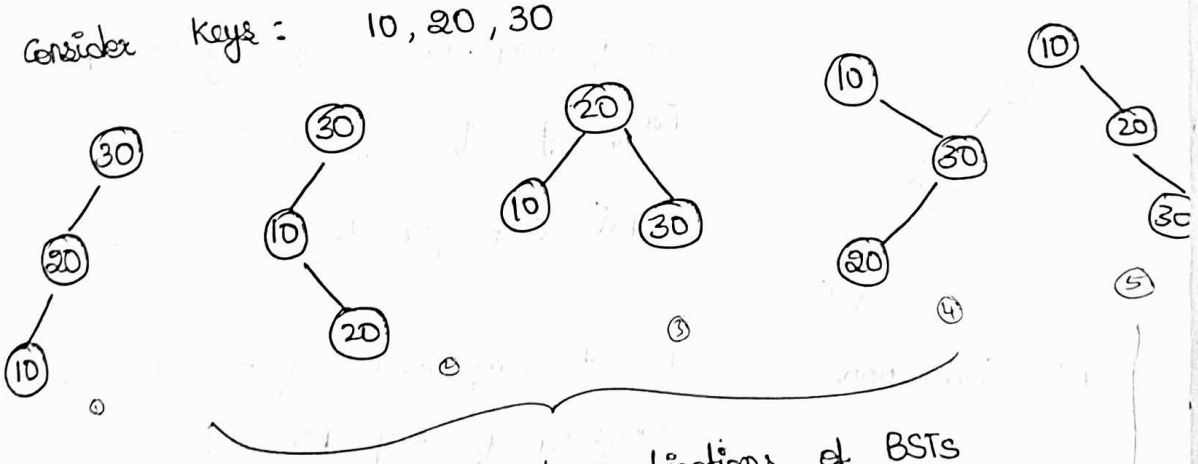
○ → Key of sucessful search

■ → represents all the unsuccesful search

Ex:



Consider Keys : 10, 20, 30



All the different combinations of BSTs having keys as 10, 20, 30

For n keys, no. of different combinations of BSTs possible are

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

Suppose $n = 3$ (10, 20, 30), $T(3) = \frac{{}^{2\times3}C_3}{4} = \frac{{}^{6}C_3}{4} = 5$

- n circular nodes O ⇒ n+1 square nodes ■

- Cost of searching each element : It depends on the no. of comparisions required for searching key elements.

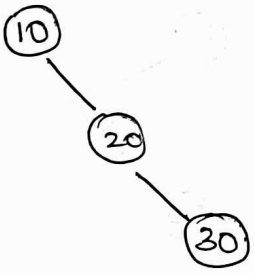
*. OPTIMAL BST :

- Here we also consider what is the frequency of searching of these keys.

frequency → ③ ② ⑤ (assume)

Key → 10, 20, 30

- If frequency is defined, then cost is :



1st Combination

no. of comparisions of key 10 = 1

Freqency of key 10 = 3

total cost for key 10 = 1 × 3 = 3

no. of comparisions of key 20 = 2

Freqency of key 20 = 2

Cost for key 20 = 2 × 2 = 4

no. of comparisions of key 30 = 3

Freqency of key 30 = 5

Cost for key 30 = 3 × 5 = 15

Total Cost of BST = Cost for Key 10 + Cost for Key 20 + Cost for Key 30

= 3 + 4 + 15 = 22

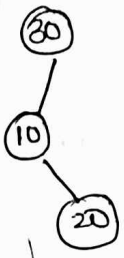
Similarly calculating Cost for other combinations

	30	20	10
Comparisions =	1	2	3
frequency =	5	2	3

Cost = $5\times1 + 2\times2 + 3\times3$

$= 5+4+9$

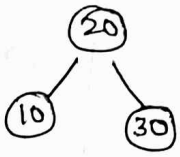
$= 18$



Cost = $1\times5 + 2\times3 + 3\times2$

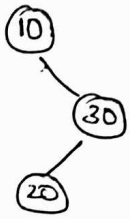
$= 5+6+6$

$= 17$



Cost = $1\times2 + 2\times3 + 2\times5$

$= 18$



Cost = $1\times3 + 2\times5 + 3\times2$

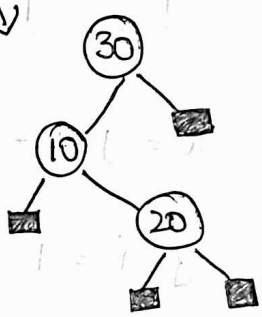
$= 3+10+6$

$= 19$

Cost = 18	Cost = 17	Cost = 18	Cost = 19

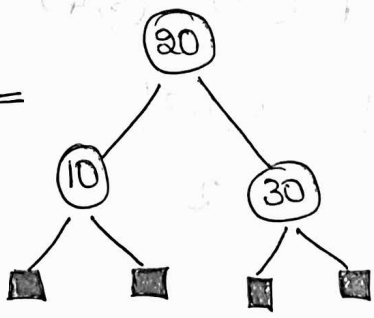
∴ Minimum Cost = 17 for

optimal Binary Search tree for the given Keys. (with frequency)



(Normal Brute Force approach)

optimal Binary Search tree for the given Keys. (without frequency)



(based on No. of minum Comparisions)

*. By Using Dynamic Programming :

Ex:

Key No.	1	2	3	4
Keys	10	20	30	40
Frequency	4	2	6	3

i \ j	0	1	2	3	4
0	0	4	$8_{(1)}$	$20_{(3)}$	$26_{(3)}$
1		0	2	$10_{(3)}$	$16_{(3)}$
2			0	6	$12_{(3)}$
3				0	3
4					0

initially : $l = j - i = 0$ [0-0=0, 1-1=0, ...]
diagonals are 0

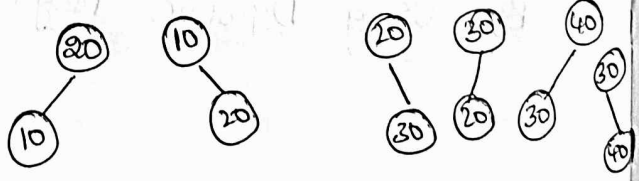
next : $l = j - i = 1$ [1-0=1, 2-1=1, 3-2=1, 4-3=1]

(0,1) (2,3)
(1,2) (3,4)

C[0,1]	C[1,2]	C[2,3]	C[3,4]
4	2	6	3

Step 2 : $l = j - i = 2$ C[0,2] C[1,3] C[2,4]

2 - 0 = (0,2)
3 - 1 = (1,3)
4 - 2 = (2,4)



Step 3 : $l = j - i = 3$

C[0,3] & C[1,4]

$C[0,3]$; $w[0,3] = 4 + 2 + 6 = 12$

10	20	30
4	2	6

Formula:

$$C[i,j] = \min_{i<k\leq j} \{ C[i,k-1] + C[k,j] + w[i,j] \}$$

$$C[0,3] = \min \{ C[0,0] + C[1,3] + 12,\ C[0,1] + C[2,3] + 12,\ C[0,2] + C[3,3] + 12 \}$$

$$= \min \{ 0 + 10 + 12,\ 4 + 6 + 12,\ 8 + 0 + 12 \}$$

$$= \min \{ 22, 22, 20 \} = 20$$

$C[0,3] = 20$ (by root 3)

$C[1,4] \Rightarrow \quad w[1,4] = 2 + 6 + 3 = 11$

2	3	4
20	30	40
2	6	3

$$C[1,4] = \min \{ C[1,1] + C[2,4] + 11,\ C[1,2] + C[3,4] + 11,\ C[1,3] + C[4,4] + 11, \}$$

$$= \min \{ 0 + 12 + 11,\ 2 + 3 + 11,\ 10 + 0 + 11 \}$$

$C[1,4] = 16$ (given by root 3)

Step 4: $l = j - i = 4$

$4 - 0 = 4$ $\Rightarrow$ $C[0,4]$

$C[0,4]$

$w[0,4] = 4+2+6+3$

$= 15$

10	20	30	40
4	2	6	3

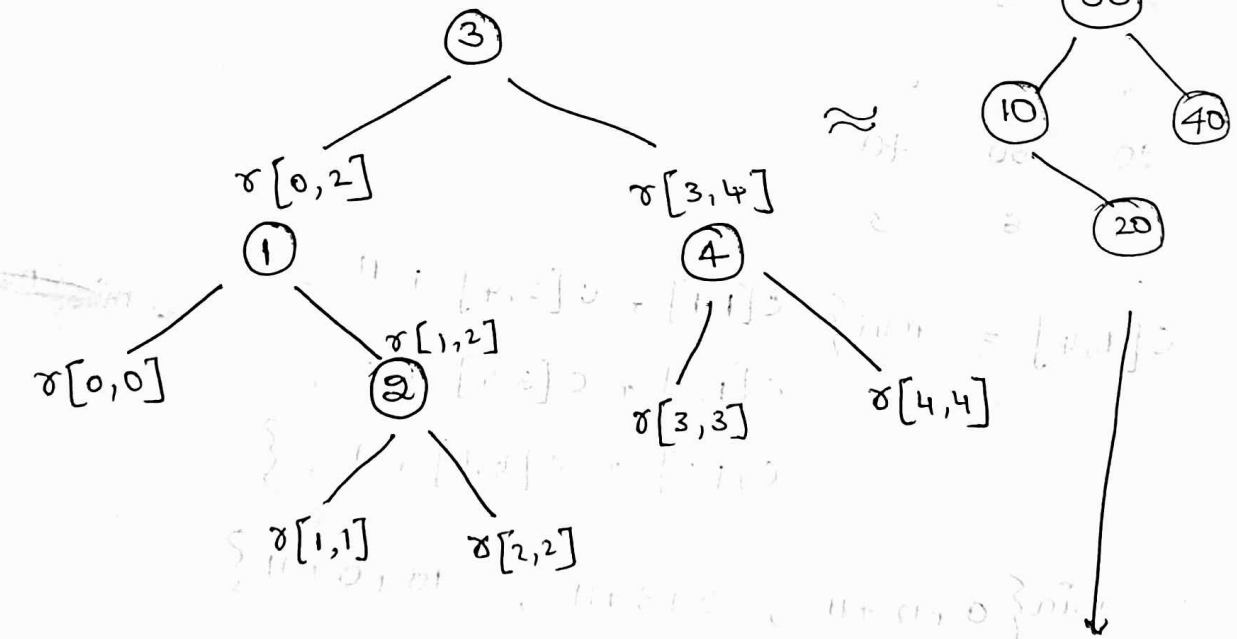
$$C[0,4] = \min \begin{cases} C[0,0] + C[1,4] + 15 \\ C[0,1] + C[2,4] + 15 \\ C[0,2] + C[3,4] + 15 \\ C[0,3] + C[4,4] + 15 \end{cases}$$

$= \min\{0+16+15,\ 4+12+15,\ 8+3+15,\ 20+0+15\}$

$= \min\{31, 31, 26, 35\} = 26$ $\longrightarrow$ end with minimum cost as 26

$C[0,4] = 26$ (given by root 3)

— Step —



OBST

∴ This is our optimal binary search tree.

with Minimum Cost = 26

- The search time of element in BST is $O(n)$
- The search time of element in Balanced BST is $O(\log n)$.

Algorithm to Compute minimum Cost :

```
function C(i,j)
  if C(i,j) already computed then
     return C(i,j)
  end if
  if i > j then return 0
  else if i = j then return Pi
  else
     return min_{i≤K<j} ( C(i,K-1) + C(K+1,j) ) + Σ_{l=i}^{j} Pl
  end if
end function
```

∴ Time Complexity : $O(n^3)$ $(O(n^{K+2}))$ K is Keys

∴ Space Complexity : $O(n^2)$ $(O(n^{K+1}))$ K is Keys

---

*. MATRIX CHAIN MULTIPLICATION :

- Given following matrices $(A_1, A_2, \ldots A_n)$ we have to Perform the matrix multiplication, which can be accomplished by a series of matrix multiplication.

$$A_1 \times A_2 \times \ldots \times A_{n-1} \times A_n$$

- Matrix multiplication is "associative" in nature.
- Matrix chain multiplication problem can be stated as "find the optimal parenthesization of a chain of matrices to be multiplied such that the no. of scalar multiplication is minimized".
- No. of ways for parenthesizing the matrices:

$(A_1) \times (A_2 \times A_3 \times \ldots \times A_{n-1} \times A_n)$

$(A_1 \times A_2) \times (A_3 \times A_4 \times \ldots \times A_n)$

$(A_1) \times (A_2 \times A_3 \times \ldots \times A_{k-1}) \times \ldots (A_k \times A_{k+1} \times \ldots \times A_n)$

$(A_1 \times A_2 \times A_3 \times A_4) \times (A_5 \times \ldots \times A_{n-1} \times A_n)$

$\vdots$

$(A_1 \times A_2 \times \ldots \times A_{n-2} \times A_{n-1}) \times (A_n)$

* Formula:

$$c[i,j] = \min_{i \le k < j} \left\{ c[i,k] + c[k+1,j] + (d_{i-1} \times d_k \times d_j) \right\}$$

* No. of parenthesizes possible = catalan number $= \frac{2n C_n}{n+1}$; where n = No. of Matrices $- 1$

Suppose $A_1 \times A_2 \times A_3 \times A_4$

$d_0\ d_1 \quad d_1\ d_2 \quad d_2\ d_3 \quad d_3\ d_4$

No. of matrices $= 4 \implies n = 4 - 1 = 3$

possibilities $= \frac{2 \times 3 C_3}{3+1} = \frac{6C_3}{4} = 5$

① $A_1 (A_2 (A_3 \cdot A_4))$ ② $A_1 ((A_2 \cdot A_3) \cdot A_4)$

③ $(A_1 \cdot A_2)(A_3 \cdot A_4)$ ④ $(A_1 (A_2 \cdot A_3))A_4$

⑤ $((A_1 \cdot A_2) \cdot A_3) \cdot A_4$ ——— 5 possibilities.

Applying formula : $c[i,j]$ (cost of Parenthesizing).

$$c[1,4] = \min_{1 \le k < 4} \begin{cases} k=1 & c[1,1] + c[2,4] + d_0 \times d_1 \times d_4 \\ k=2 & c[1,2] + c[3,4] + d_0 \times d_2 \times d_4 \\ k=3 & c[1,3] + c[4,4] + d_0 \times d_3 \times d_4 \end{cases}$$

Example :

A_1 (3×2) $\times$ A_2 (2×4) $\times$ A_3 (4×2) $\times$ A_4 (2×5)

$\Rightarrow d_0 = 3$, $d_1 = 2$, $d_2 = 4$, $d_3 = 2$, $d_4 = 5$

also, $c[1,1] = c[2,2] = c[3,3] = c[4,4] = 0$.

i \ j	①	②	③	④
①	0	k=1 24	k=1 28	k=3 58
②		0	k=2 16	k=3 36
③			0	k=3 40
④				0

k	1	2	3	4
1	0	1	1	3
2			2	3
3				3
4				

- $C[1,2] = \min_{i \le k < j} \begin{matrix} k=1 \\ i=1 \\ j=2 \end{matrix} \left\{ C[1,1] + C[2,2] + d_0 \times d_1 \times d_2 \right.$

$$= \min \left\{ 0 + 0 + (3 \times 2 \times 4) \right\} = 24$$

- $C[2,3] = \min_{i \le k < j} \begin{matrix} k=2 \\ i=2 \\ j=3 \end{matrix} \left\{ C[2,2] + C[3,3] + d_1 \times d_2 \times d_3 \right\}$

$$= \min \left\{ 0 + 0 + (2 \times 4 \times 2) \right\} = 16$$

- $C[3,4] = \min_{i \le k < j} \begin{matrix} k=3 \\ i=3 \\ j=4 \end{matrix} \left\{ C[3,3] + C[4,4] + d_2 \times d_3 \times d_4 \right\}$

$$= \min \left\{ 0 + 0 + (4 \times 2 \times 5) \right\} = 40$$

- $C[1,3] = \min_{1 \le k < 3} \begin{matrix} k=1 \\ k=2 \end{matrix} \begin{cases} C[1,1] + C[2,3] + d_0 \times d_1 \times d_3 \\ C[1,2] + C[3,3] + d_0 \times d_2 \times d_3 \end{cases}$

$$= \min \begin{cases} 0 + 16 + (3 \times 2 \times 2) \\ 24 + 0 + (3 \times 4 \times 5) \end{cases} = \min \begin{cases} 28, \\ 48 \end{cases} = 28$$

- $C[2,4] = \min_{2 \le k < 4} \begin{matrix} k=2 \\ k=3 \end{matrix} \begin{cases} C[2,2] + C[3,4] + (d_1 \times d_2 \times d_4) \\ C[2,3] + C[4,4] + d_1 \times d_3 \times d_4 \end{cases}$

$$= \min \begin{cases} 0 + 40 + (2 \times 4 \times 5) \\ 16 + 0 + (2 \times 2 \times 5) \end{cases} = \min \begin{cases} 80, \\ 36 \end{cases} = 36$$

- $c[1,4] = \min_{1 \le k < 4}$

$$\begin{cases} k=1 & c[1,1] + c[2,4] + d_0 \times d_1 \times d_4 \\ k=2 & c[1,2] + c[3,4] + d_0 \times d_2 \times d_4 \\ k=3 & c[1,3] + c[4,4] + d_0 \times d_3 \times d_4 \end{cases}$$

$$= \min \begin{cases} 0 + 36 + (3 \times 2 \times 5) \\ 24 + 40 + (3 \times 4 \times 5) \\ 28 + 0 + (3 \times 2 \times 5) \end{cases} = \min \begin{cases} 86, \\ 124, \\ 58 \end{cases} = 58$$

Minimum cost of Parenthesizing = 58

we get k-table (use it for parenthesizing)

k	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

$\Rightarrow$ based on where we got minimum cost like while finding $c[i,j]$ at which $k = ?$

at 3

$(1,4) \equiv (A_1 \cdot A_2 \cdot A_3) \cdot A_4$

k value = 3

so do parathesis at A_3

$(1,3) \quad ((A_1) \cdot A_2 \cdot A_3) \cdot A_4$

k value = 1

So parathesis at A_1

$\Rightarrow$ Result $= ((A_1) \cdot (A_2 \cdot A_3)) \cdot A_4$ (Minimum cost = 58)

Time Complexity = $T(n) \approx \frac{n(n+1)}{2} \times n$ (different k values upto n)

$\therefore$ $O(n^3)$

And, Space Complexity = $O(n^2)$

---

*. RELIABILITY DESIGN :

- The problem is to design a system that is composed of several devices connected in series.
- D_i – ith device r_i – reliability of ith device
  C_i – cost of ith device
- Total reliability of system = product of r_i
- "How many copies of each device to be bought within a particular cost such that reliability of whole system is maximized" — this is to be solved.

Ex :

D_i	C_i	r_i	U_i
D_1	30	0.9	2
D_2	15	0.8	3
D_3	20	0.5	3

$\sum C_i = C_1 + C_2 + C_3$

remaining Cost = $C - \sum C_i$

$= 105 - (30 + 15 + 20)$

$= 105 - 65 = 40$

Actual Cost, $C = 105$

Upper bound, $U_i = \left\lfloor \frac{C - \sum C_i}{C_i} \right\rfloor + 1$ (floor value)

$$U_1 = \left\lfloor \frac{C - \sum c_i}{c_1} \right\rfloor + 1 = \left\lfloor \frac{40}{30} \right\rfloor + 1 = 1+1 = 2$$

$$U_2 = \left\lfloor \frac{C - \sum c_i}{c_2} \right\rfloor + 1 = \left\lfloor \frac{40}{15} \right\rfloor + 1 = 2+1 = 3$$

$$U_3 = \left\lfloor \frac{C - \sum c_i}{c_3} \right\rfloor + 1 = \left\lfloor \frac{40}{20} \right\rfloor + 1 = 2+1 = 3$$

(R, C) → pair of Reliability & Cost

$S^0 = \{(1,0)\}$: initial (multiply, add)

Consider D_1 $\quad S_1^1 = \{(1\times0.9,\ 0+30)\}$ → because $U_1 = 2$, so S_1^1, S_2^1

$S_2^1 = \{(1\times0.99,\ 0+60)\}$

$= \{(0.99, 60)\}$

$= 1-(1-r_1)^2$

$= 1-(1-0.9)^2$

$= 0.99$

$S^1 = \{(0.9, 30), (0.99, 60)\}$

Consider D_2, $\quad S_1^2 = \{(0.8\times0.9,\ 30+15), (0.8\times0.99,\ 60+15)\}$

$S_1^2 = \{(0.72, 45), (0.792, 75)\}$

$S_2^2 = \{(0.96\times0.9,\ 30+30), (0.96\times0.99,\ 30+60)\} = \{(0.864, 60), (0.9504, 90)\}$

because cost = 90, remaing = 15, can't buy D_3

$= \{(0.864, 60)\}$

$S_3^2 = \{((0.864), (75)), (\cancel{105})\}$

because of cost

$S^2 = \{ (0.72, 45) , (0.792, 75) , (0.864, 60) , (0.8928, 75) \}$

↓ removed based on dominance

3rd device, $S_1^3 = \{ (0.36, 65) , (0.432, 80) , (0.4464, 95) \}$

$S_2^3 = \{ (0.75 \times 0.72 , 40+45) , (0.75 \times 0.864, 60+45) \}$

$= \{ (0.54, 85) , (0.648, 100) \}$

$S_3^3 = \{ (0.75 \times 0.875 , 60+45) , (, 60+60) (, 75+60) \}$

$= \{ (0.63 , 105) \}$

$S^3 = \{ (0.36, 65) , (0.432, 80) , (0.4464, 95) , (0.54, 85) , (0.648, 100) , (0.63, 105) \}$ → Dominance Rule

Arrange in ascending order based on reliability

$S^3 = \{ (0.36, 65) , (0.432, 80) , (0.54, 85) , (0.63, 105) , (0.648, 100) \}$

↓ Dominance Rule

check for Dominance Rule

$S^3 = \{ (0.36, 65) , (0.432, 80) , (0.54, 85) , (0.648, 100) \}$ ←

Maximum reliability of system = 0.648

Cost of entire system = 100

Copies of each device

check for (0.648, 100) pair in S_3^3 $\Rightarrow$ 2 $= D_3$

(0.648, 100) is obtained for (0.864, 60) $\Rightarrow$ 2 $= D_2$

(0.864, 60) is obtained from (0.9, 30) $\Rightarrow$ 1 $= D_1$

$\therefore$ D_1 D_2 D_3 $\longrightarrow$ $\Pi r_i = 0.648$

1 2 2 $\Sigma c_i = 100$

---

*. Comparision Dynamic Programming (vs) Greedy Algorithms :

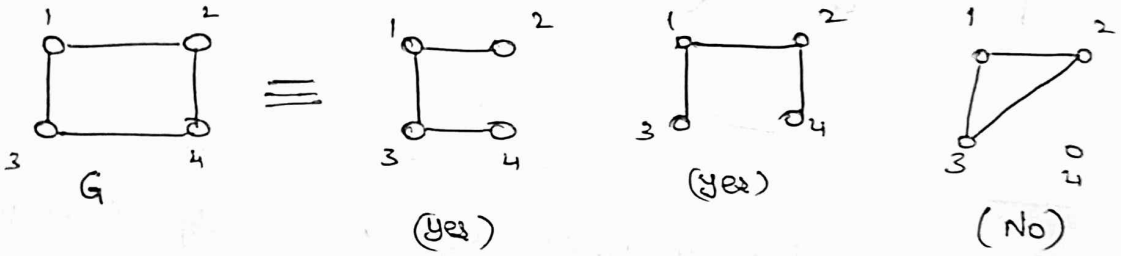
Dynamic Programming	Greedy Algorithms
• At each step, the choice is determined based on solutions of subproblems	• At each step, we quickly make a choice that looks best.
• sub problems are solved first.	• Greedy choice can be made first before solving further sub problems
• Bottom-up approach	• Top-down approach
• Can be slower, more complex	• Usually faster, simpler
• optimization problems	• Prim's, Kruskal's, Dijokshtra's Algorithms

*. Spanning Tree : A connected subgraph 's' of Graph G(V,E) is said to be spanning itself, iff,

1) 's' should contain all vertices of 'G'.

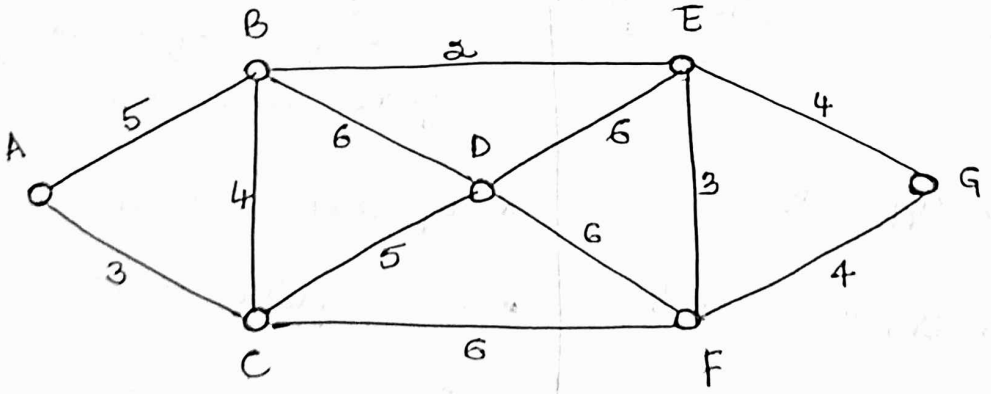
2) 's' should contain (|V| − 1) edges.

3) No cycles & all vertices should be reachable.



- No. of spanning trees of a complete graph with n vertices = n^{n-2}
- Minimum Cost Spanning Tree — Kruskal's, Prim's Algo.

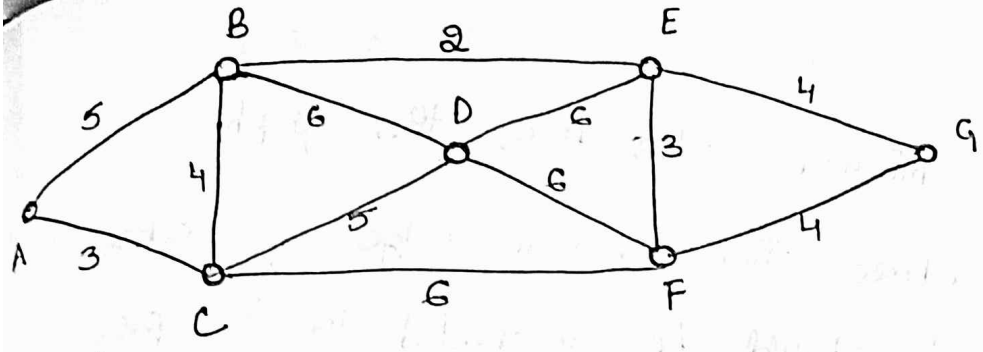
*. KRUSKAL ALGORITHM : (Best because of Sparse)



Arrange all the weights in A.O : 2, 3, 3, 4, 4, 4, 5, 5, 6, 6, 6, 6

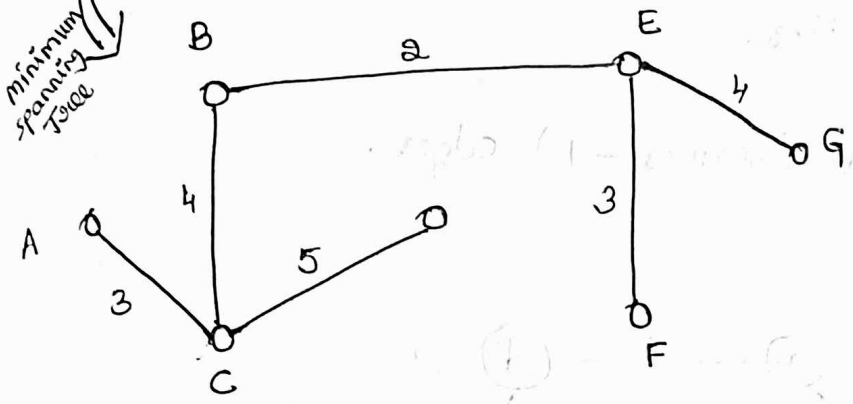
Keep connecting the edges in this order such that cycle doesn't form, upto (vertices − 1) edges.

↓ given in Question



vertices = 7
edges = 12

minimum spanning Tree



vertices = 7
edges = V-1
= 7-1
= 6

$\therefore$ Cost = 3+3+4+4+5+2 = 21

By using Greedy Approach :

1) Construct Min Heap with 'e' edges.
2) Take one by one edge and add in spanning tree. (cycle shouldnot be Created).

Best case : (n-1) edges. → Time : n log e

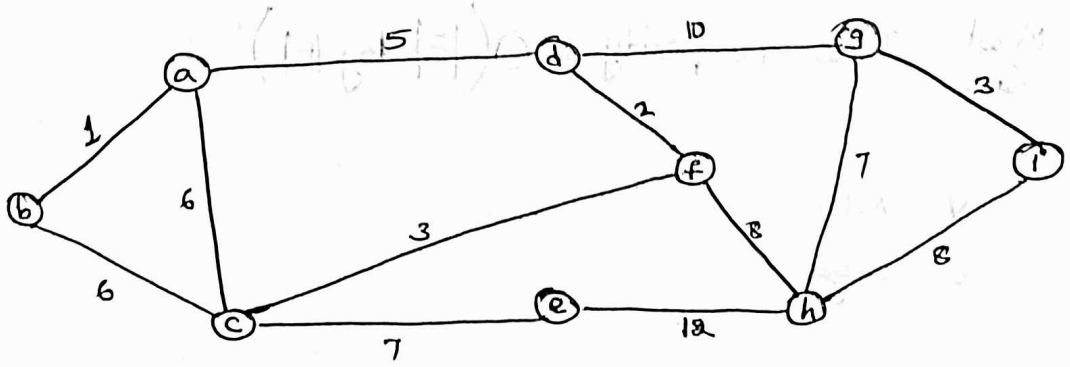
worst case : 'e' edges. → Time : e log e

n - vertices , e - edges

Space Complexity : $O(|E|+|V|)$

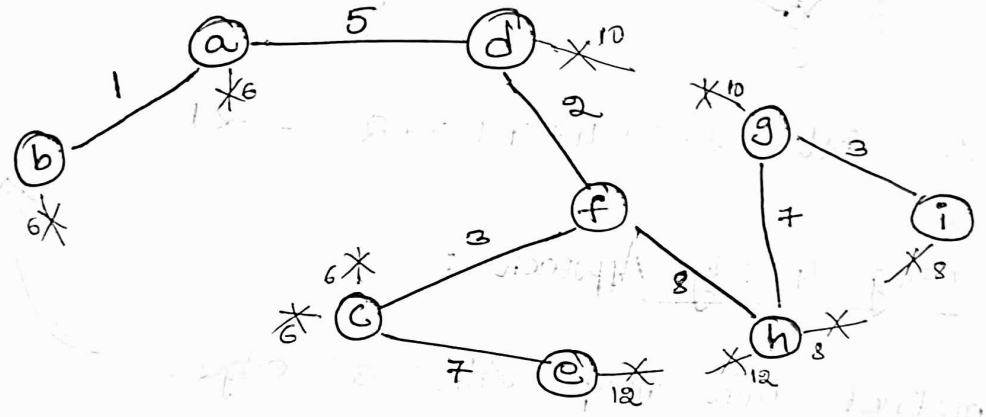
---

*. PRIM'S ALGORITHM :

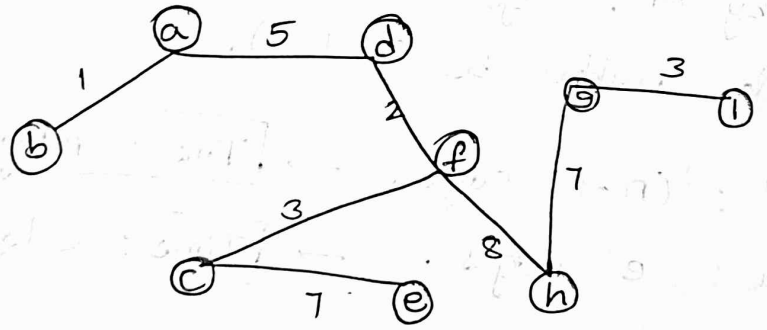


Steps :

1) Select minimum edge from the graph
2) Now choose the minimum edge of vertex such that it should be connected to the previous selected vertices.
3) Repeat upto (vertices − 1) edges.



⇒



cost = 36

- Best/Avg Time Complexity : $O((V+E)\log V)$
- Space Complexity : $O(V+E)$
- Worst Case Time Complexity : $O(|E| \log |E|)$

V - Vertices

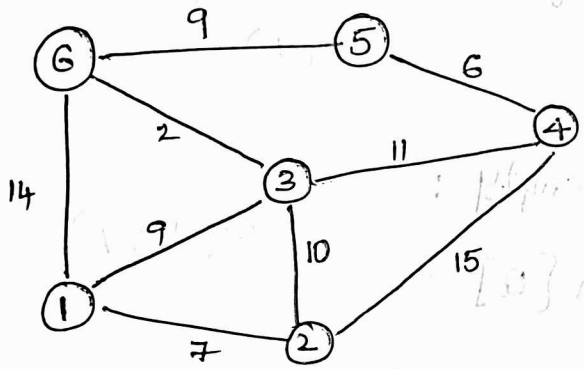
E - Edges

* DIJKSTRA'S ALGORITHM : (Single - source shortest Path)

- Effectively used in Social Networking, Google Maps
- **Relaxation** : if $d(u) + c(u,v) < d(v)$
  then $d(v) = d(u) + c(u,v)$

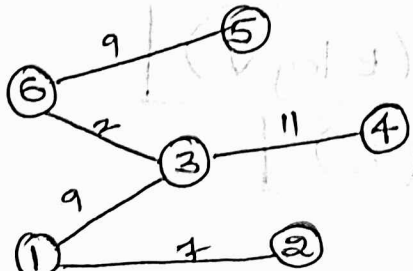
d - distance
c - cost
u - source vertex, v - destination vertex

EX:



Source	Destination				
1	2	3	4	5	6
	∞	∞	∞	∞	∞
	(7)	9	∞	∞	14
1,2	7	(9)	22	∞	14
1,2,3	7	9	20	∞	(11)
1,2,3,6	7	9	20	(20)	11
1,2,3,6,5	7	9	(20)	20	11
1,2,3,6,5,4	7	9	20	20	11

→ 1 → 2 → 3 → 6 → 5 → 4



Cost = 38

* Dijakstra's Analysis : [Use min-heap datastructure]

Dijkstra (Graph, Source) — GREEDY Approach

```
{
  Create vertex set Q ;

  for each vertex v in graph :
  {
      dist [v] = ∞          —— O(V)
  }   add v to Q            —— Build Heap
                                O(V)
  dist [source] = 0

  while Q is not empty :    —— O(log V)
  {
     u = Extract min [Q]

     for each neighbour v of u :
     {
        Relax (u,v) —— Relaxation formula —— O(log V)
     }
  }
}
```

Time: $O(V) + O(V) + V \log V + E \cdot \log V$ (Pick highest Asymptotic class)

worst Time Complexity : $O(V^2)$

Avg/Best case/ using min-priority Queue : $O((V+E)\log V)$

Space Complexity : $O(|V| + |E|)$

Here, $E > V-1 \Rightarrow |V| + |E| \approx |E|$

- Sometimes, **Time : $O(E \log V)$**

  Space : $O(E)$

E - Edges
V - vertices

* FORD - FULKERSON ALGORITHM :

- Problem :

Given a graph which represents a flow of network where every edge has a capacity. Also given two vertices Source S and sink T in the graph. Find out the maximum possible flow from S to T with following constraints:

a) Flow on an edge doesnot exceed the given capacity of the edge.

b) In flow is equal to out flow for every vertex except S and T.

- Algorithm : (Maximum Flow Problem)

The following is the simple idea of the algorithm.

1) Start with initial flow as 0.

2) while there is an augumeting path from source to sink, Add this path flow to flow.

3) Return flow.

- Terminologies :

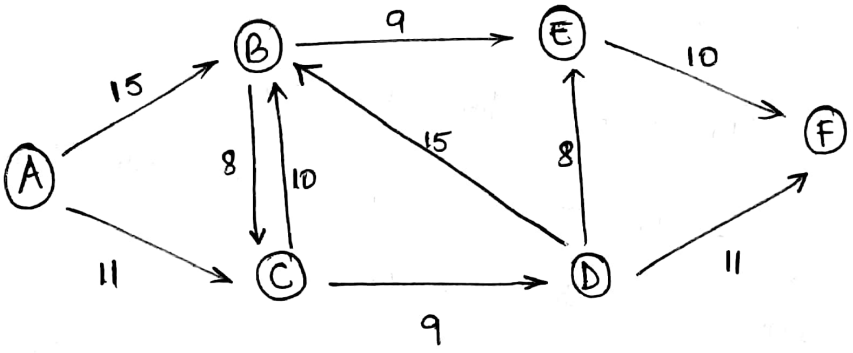
1. Residual Graph : It's a graph which indicates additional possible flow. If there is such path from source to sink then there is possibility to add flow.

2. Residual Capacity : It's original capacity of the edge minus flow.

3. Minimal Cut : Also known as bottle neck capacity, which decides maximum possible flow from source to sink through an augumented path.

4. Augumenting Path : Augumenting path can be done in two ways :- 1) Non-full forward Edges.
2) Non-empty backward Edges.

Example :



Source = A , Sink = F

Capacity of path A to C = 11
Capacity of path C to D = 9
Capacity of path D to F = 11

} Augumented Path
A → C → D → F

Maximum possible flow through
Augumented path = min (capacities of all sub paths)

Maximum possible flow
A → C → D → F = Minimal Cut = min(11, 9, 11)
= 9

∴ Bottle neck capacity of A to C, D to F = 9/11
Bottle neck capacity of C to D = 9/9

Residual Capacity of A to C = (Capacity of A to C) − (Minimal Cut)

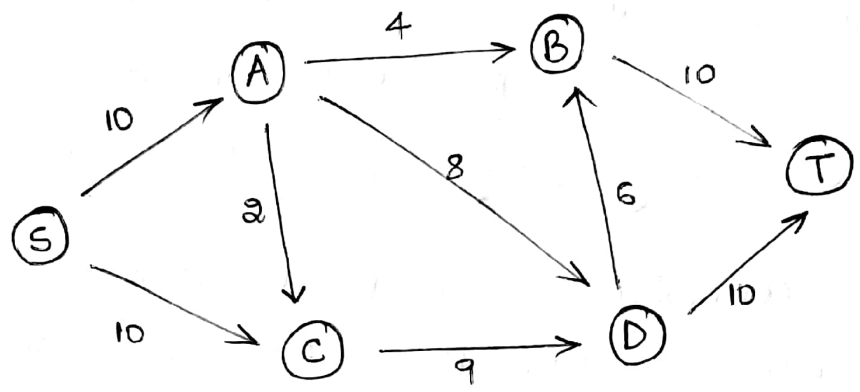
Residual Capacity of A to C = 11 − 9 = 2

Residual Capacity of C to D = 9 − 9 = 0

" " of D to F = 11 − 9 = 2

EXAMPLE :

Initially,

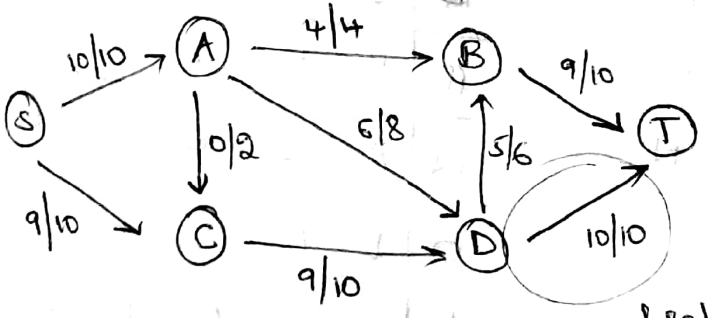


S.No.	Augumenting Paths	Bottle Neck capacity
①	S → A → D → T	8
②	S → C → D → T	2
③	S → C → D → A → B → T	4
④	S → A → D → B → T	2
⑤	S → C → D → B → T	3

(Non-empty Backward Edge)

total = 8 + 2 + 4 + 2 + 3 = 19

Finally,



Flow = 19

closed & Full

*. STABLE MARRIAGE PROBLEM :

- The Gale Shapley Algorithm is solution to stable Marriage Problem.
- Problem Statement : In the stable Marriage Problem, we want to create a set of n pairs using two sets of n people. With one person from each set appearing in each pair, each individual may only be paired once and such the result will be a direct one to one mapping between the two sets of people.
- The real complexity of this problem arises when from individual preferences each person has about who they would like to be paired with from the other set.
- The challenge / aim is to create a set of matchings for which there are no two people to be with who prefer to be with each other than their current matchings respectively.

Ex: We have 2 groups of people Males {A, B, C, D, E} and females {L, M, N, O, P} with their respective preferences are shown below.

	MALES				
A	O	M	N	L	P
B	P	N	M	L	O
C	M	P	L	O	N
D	P	M	O	N	L
E	O	L	M	N	P

	FEMALES				
L	D	B	E	C	A
M	B	A	D	C	E
N	A	C	E	D	B
O	D	A	C	B	E
P	B	E	A	C	D

Solution :

ALGORITHM :

1. All individuals have ranked members of the opposite set in order of preference.
2. one of the two sets is chosen to make proposals.
   - Either set will produce stable Matchings.
3. Choose a proposing group (either Male / Female). One individual from the proposing group who is not already engaged will propose to their most preferable option who has not already rejected them
4. The person being proposed to will :
   - Accept if this is their first offer.
   - Reject if this is worse than their current offer.
   - Accept if this is better than their current offer. In this case, they will reject their previous offer.
5. When all members of the proposing group are matched, the loop terminates. The currently engaged pairs are now stably matched.

Time Complexity : $O(n^2)$; where n is no. of persons in each group.

Proposing Group

	MALES				
A	O	M	N	L	P
B	P	N	M	L	O
C	M	P	L	O	N
D	P	M	O	N	L
E	O	L	M	N	P

	FEMALES				
L	D	B	E	C	A
M	B	A	D	C	E
N	A	C	E	D	B
O	D	A	C	B	E
P	B	E	A	C	D

- Now, Coming to Male D, P Cannot be mapped because P's best preference is B & B's best is P. Both are stable, So, when D proposes P, P rejects D

	MALES				
A	O	M	N	L	P
B	P	N	M	L	O
C	M	P	L	O	N
D	~~P~~	M	O	N	L
E	~~O~~	L	M	N	P

	FEMALES				
L	D	B	E	C	A
M	B	A	D	C	E
N	A	C	E	D	B
O	D	A	C	B	~~E~~
P	B	E	A	C	~~D~~

- Similarly Male E proposes O but O has already better preference A than E. So, O rejects E
- At this point, A, B, C are paired. D, E are unpaired
- So, Next D proposes M. For female M, D is best preference than C, So, it rejects C & accepts D
- After E proposes to L. L accepts the proposal because L didnot get any proposal upto now.

	MALES				
A	(O)	M	N	L	P
B	(P)	N	M	L	O
C	~~M~~	P	L	O	N
D	~~P~~	(M)	O	N	L
E	~~O~~	(L)	M	N	P

	FEMALES				
L	D	B	(E)	C	A
M	B	A	(D)	~~C~~	E
N	A	C	E	D	B
O	D	(A)	C	B	~~E~~
P	(B)	E	A	C	~~D~~

- Here C is the only male left unpaired, so C proposes P but preference of P (i.e. B) is more than C. Similarly it proposes L again preference of L (i.e. E) is more than C. Next it proposes O again preference of O (i.e. A) is more than C.
- Left with no choice, C proposes N. Luckily, N didnot have any proposals, N accepts C and C is paired with N.
- Final Table will be, (All Males are engaged)

	MALES				
A	(O)	M	N	L	P
B	(P)	N	M	L	O
C	~~M~~	~~P~~	~~L~~	~~O~~	(N)
D	~~P~~	(M)	O	N	L
E	~~O~~	(L)	M	N	P

	FEMALES				
L	D	B	(E)	~~C~~	A
M	B	A	(D)	~~C~~	E
N	A	(C)	E	D	B
O	D	(A)	~~C~~	B	~~E~~
P	(B)	E	A	~~C~~	~~D~~

Hence Stable Pairs are :

(If proposing group is Male)

$$\{(A,O),\ (B,P),\ (C,N),\ (D,M),\ (E,L).\}$$

- If any man prefers a woman to his current matching, then he must have already proposed to her. Because they are not matched, she must have already rejected him meaning that she cannot prefer him to her current matching.

Assuming Proposing group is Woman/Female :

FEMALES (proposing)

L	~~D~~	~~B~~	(E)	C	A
M	~~B~~	(A)	D	C	E
N	~~A~~	(C)	E	D	B
O	(D)	A	C	B	E
P	(B)	E	A	C	D

MALES

A	O	(M)	~~N~~	L	P
B	(P)	N	~~M~~	L	O
C	M	P	L	O	(N)
D	P	M	(O)	N	~~L~~
E	O	(L)	M	N	P

- (E, L) (A, M) (B, P) (C, N) (D, O) } If proposing group is Female

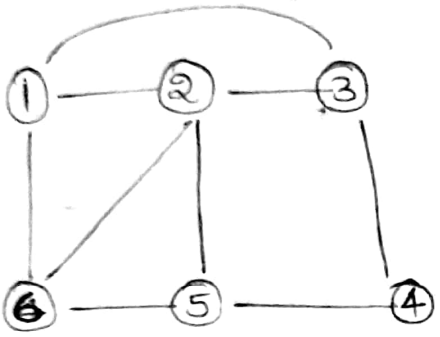
- (E, L) (A, O) (B, P) (C, N) (D, M) } If proposing group is Male

- Female M got more favourable than previous.
- Male A got less favourable than previous.

* HAMILTONIAN CYCLE :

- Hamiltonian cycle is a path that visits each vertex exactly once and forms a cycle without duplication.

Ex:



Possible paths are :

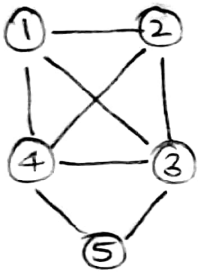
1, 2, 3, 4, 5, 6, 1

1, 2, 6, 5, 4, 3, 1

1, 6, 2, 5, 4, 3, 1

~~2, 3, 4, 5, 6, 1, 2~~ (duplicate of)

Ex:



1, 2, 3, 5, 4, 1

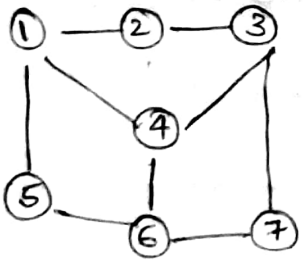
1, 2, 4, 5, 3, 1

1, 3, 5, 4, 2, 1

1, 4, 5, 3, 2, 1

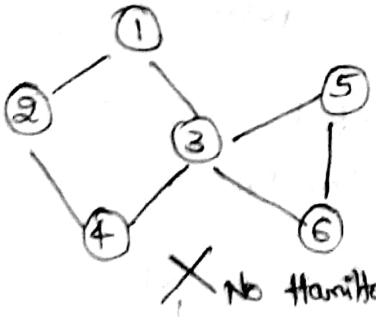
⋮

Ex:



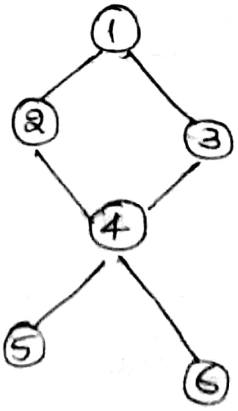
No hamiltonian cycle if we visit 4, then visiting 5 is not possible, if we visit 5, visting 4 is not possible.

Ex:



X No Hamiltonian cycle

vertex 3 is called "Articulation Point"



X No Hamiltonian cycle

vertex 4 is called "Pendant vertex"

```
Algorithm Hamiltonian (k)
{
    do
    {
        NextVertex(k);
        if (x[k] == 0)
            return;
        if (k == n)
            print (x[1:n]);
        else
            Hamiltonian (k+1);
    } while (true);
}
```

```
Algorithm NextVertex (k)
{
    do
    {
        x[k] = (x[k]+1) mod (n+1);
        if (x[k] == 0) return;
        if (G[x[k-1], x[k]] + 0)
        {
            for j=1 to k-1 do
                if (x[j] == x[k])
                    break;
            if (j == k)
                if (k<n or k==n &&
                    G[x[n], x[1]] ≠ 0)
                    return;
        } while (true);
```

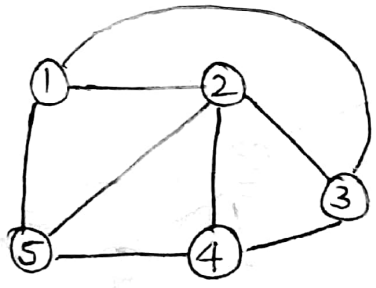
Time Complexity : $(n-1)!$ or $O(n^n)$ ← Exponential time

Space Complexity : $O(n)$

It is NP-Hard problem.

Example: Given Graph, G & Adjacency matrix G_M. Find all the possible Hamiltonian Paths / Cycles using Backtracking.

G:



$$G_m = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

State Space tree need to be constructed with root node as starting vertex.

x

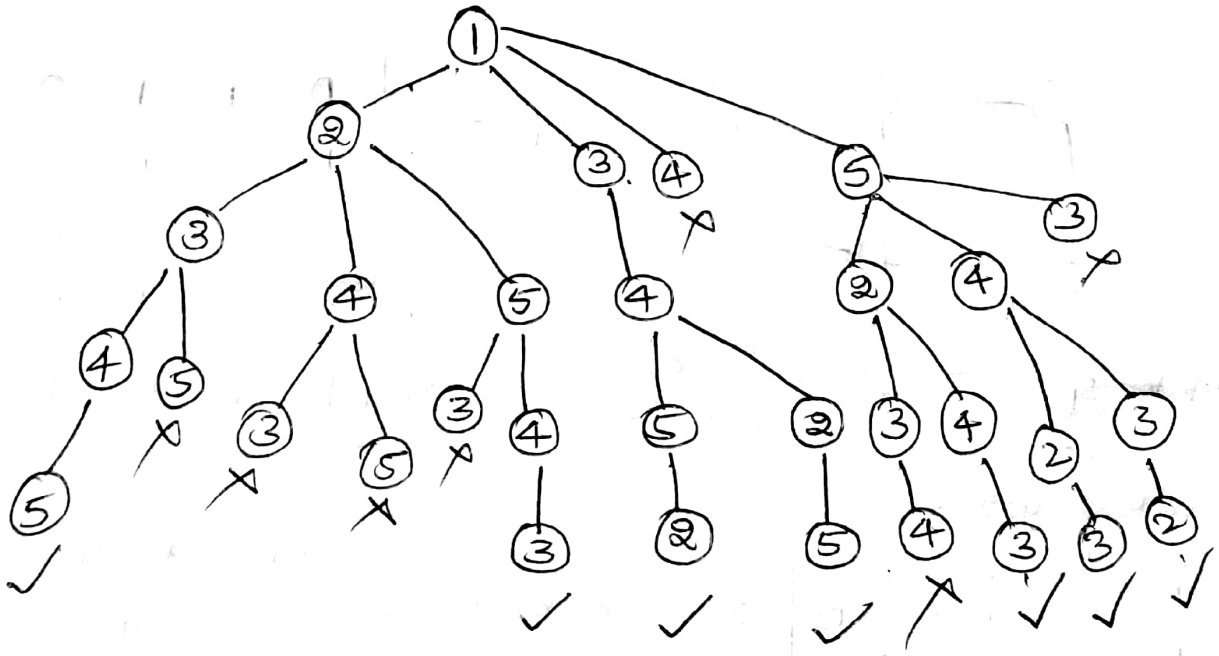
0	0	0	0	0
1	2	3	4	5

initial values. When all becomes 1, then solution is found & that is our Hamiltonian cycle.

Algorithm :

- Create an empty path Array
- Add vertex 0 to the array
- Start adding vertex 1 and other connected nodes and check if the current vertex can be included in the array or not.
- This can be done by using a visiting array and check if the vertex has already been visited or is adjacent to previously added vertex.
- If any such vertex is found, add it to the array and backtrack from the node.
- Try every possible combinations and if a path returns false, ignore the vertex and start iterating from the next ~~week~~ vertex till all nodes have been visited.

Backtracking state space tree :



All possible hamiltonian cycles are :

1, 2, 3, 4, 5, 1

1, 2, 5, 4, 3, 1

1, 3, 4, 5, 2, 1

1, 3, 4, 2, 5, 1

1, 5, 2, 4, 3, 1

1, 5, 4, 2, 3, 1

1, 5, 4, 3, 2, 1

---

*. GRAPH COLORING :

- Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.
- Coloring of graph means the assignment of colors to all the vertices.

Algorithm :

1. Create a recursive function that takes the graph, current index, no. of vertices and output color array.
2. If the current index is equal to the no. of vertices. Print the color configuration in output array.
3. Assign a color to a vertex (1 to m).
4. For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices donot have the same color) recursively call the function with next index and no. of vertices.

5. If any recursive function returns true, break the loop & return true.

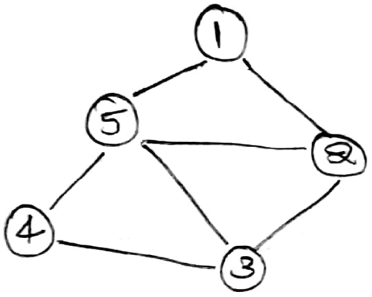
6. If no recursive function returns true, then return false.

Time Complexity : $O(m^V)$

Space Complexity : $O(V)$

m - no. of colors
V - no. of vertices of graph

Ex :



Colors	1	2	3	4	5
	Red	Green	Red	Green	Blue
	Green	Red	Green	Red	Blue
	Blue	Red	Blue	Red	Green
	Blue	Green	Blue	Green	Red

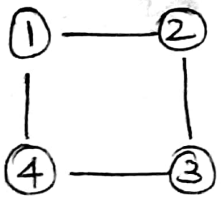
Find all possible solutions → BACKTRACKING can be used.

① m Coloring decision Problem — Can the graph be Colorable?

② m Coloring optimization Problem — minimum Colors required to color a graph ?

③ which color - which vertex - total Colors — Chromatic Problem.

Ex :



m : No. of Colors

$m = 3$

{ Red, Green, Blue }
R G B

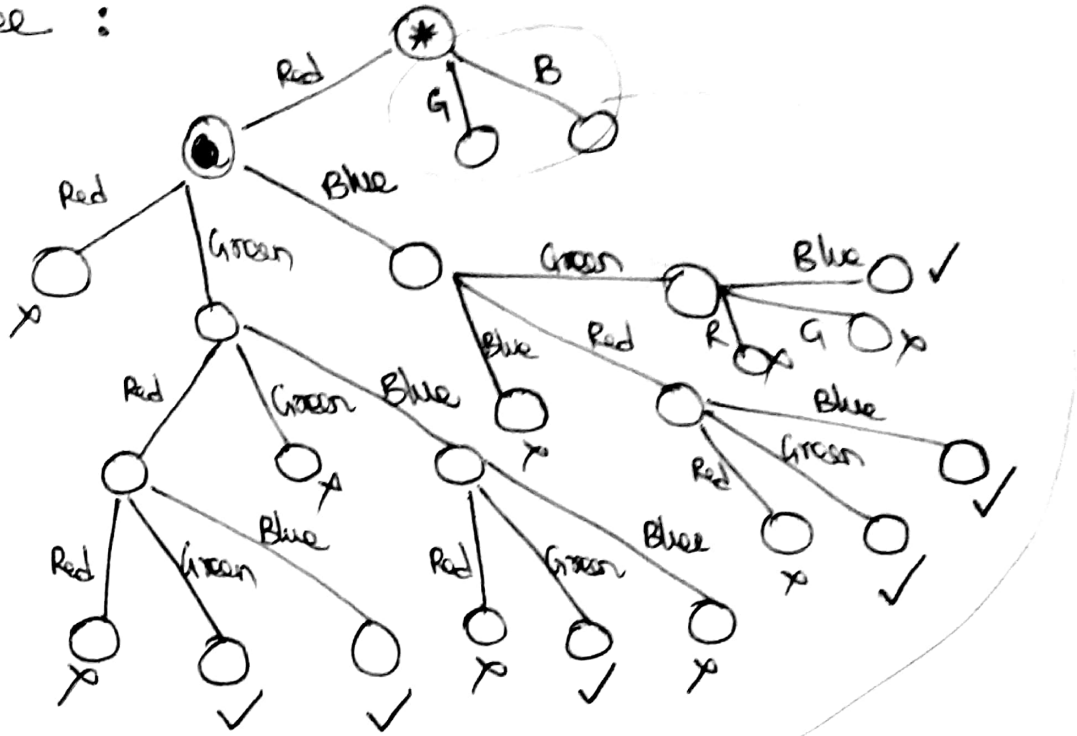
Total Nodes Without bounding $= 1 + 3 + 3\times3 + 3\times3\times3 + 3\times3\times3\times3$

$= 1 + 3 + 3^2 + 3^3 + 3^4$

4 vertices — $\frac{3^{4+1} - 1}{3 - 1}$

m vertices — $\frac{m^{n+1} - 1}{m - 1}$

State Space tree :



Red, Green, Red, Green

Red, Green, Red, Blue

Red, Green, Blue, Green

Red, Blue, Red, Green

Red, Blue, Red, Blue

Red, Blue, Green, Blue

Similarly for Green, Blue Cases.

*. MAXIMUM BIPARTITE MATCHING :

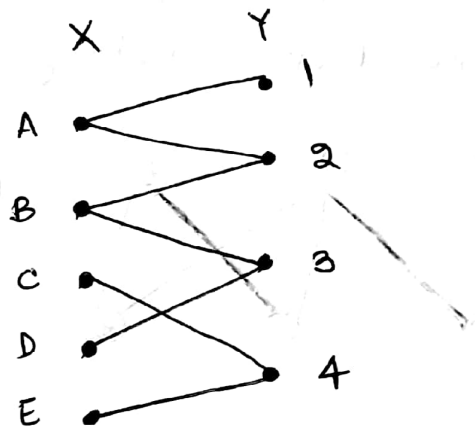
- Pairing 2 vertices from 2 sets.
- m applications to n jobs. (Example)

- **Bipartite Graph :**

- A Graph $G = (V, E)$ in which the vertex set V is divided into 2 disjoint sets X and Y.
- Every edge of the Graph has one end point in X and other end point in Y.

Example :

Graph



$X \cup Y = \{\text{vertex set}\} = V$

$X \cap Y = \emptyset$

$V = \{A, B, C, D, E, 1, 2, 3, 4\}$

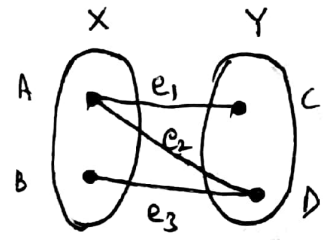
$X = \{A, B, C, D, E\}$

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

INP
- Vertices of the same set are not connected. X, Y are disjoint sets.
- Contains 2 disjoint / distinct set of vertices.

- **Matching : (M)**

- Matching in a graph is a subset of edges that no two edges share a vertex.



$V = \{A, B, C, D\}$

$X = \{A, B\}$ $\quad Y = \{C, D\}$

Edge set, $E = \{e_1, e_2, e_3\}$

* **Matching, $M = \{e_1, e_3\}$**

Here $\{e_1, e_2\}$ cannot be matching set because they share same vertex A.

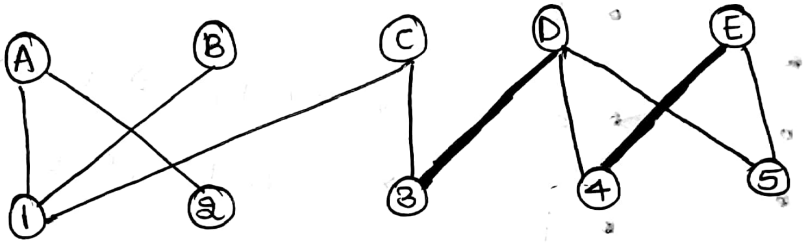
Similarly $\{e_2, e_3\}$ cannot be matching set because they share same vertex D.

- **Two colorable Graph :**

- A Graph that ~~cannot~~ can be colored only with 2 colors.
- No two edges connect the same color. (or) adjacent vertices are not colored same color.
- Bipartite Graph is a 2-color graph. one color is given to one distinct set & other to another distinct set.

- **Maximum Matching Problem :**

- Let M be the matching in bipartite graph G. Consider given Bipartite Graph. Find Maximum matching.



where
$M = \{(D,3), (E,4)\}$

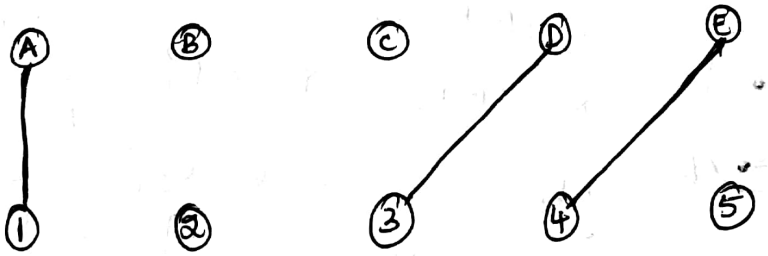
Sol :

Given
matching set = $\{(D,3), (E,4)\} = M$

Let $X = \{A, B, C, D, E\}$

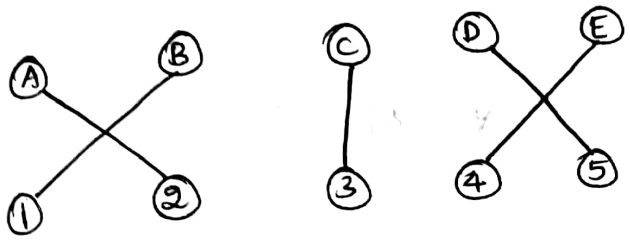
$Y = \{1, 2, 3, 4, 5\}$

Step 1 : Consider all free vertices (vertices not under matching)



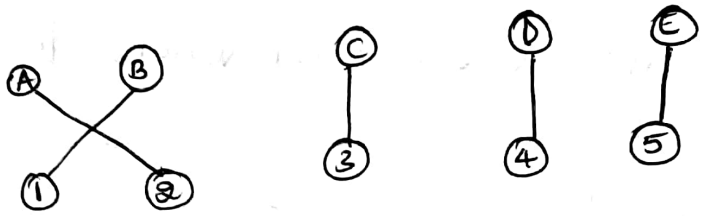
$M_1 = \{(A,1), (D,3), (E,4)\}$

$M_2 = \{(A,2), (B,1), (D,3), (E,4)\}$



$M_3 = \{(A,2), (B,1), (C,3), (D,5), (E,4)\}$

(or) $M_3 = \{(A,2), (B,1), (C,3), (D,4), (E,5)\}$



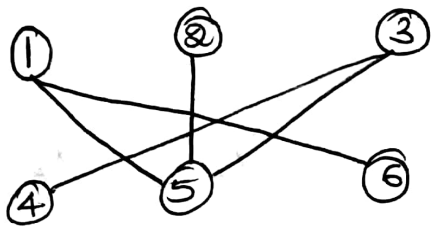
∴ Maximum matching set = $\{M_3\}$

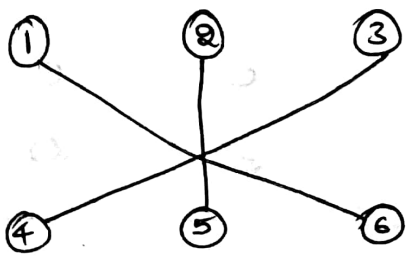
$M = \{(A,2), (B,1), (C,3), (D,5), (E,4)\}$

(or) $M = \{(A,2), (B,1), (C,3), (D,4), (E,5)\}$

Ex 2: Find maximum matching for given Bipartite Graph:

Sol:





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

* SIMPLEX METHOD :

$\leq$	Slack variable	+ variable
$\geq$	Surplus variable	− variable

Ex: Max $Z = 5x_1 + 3x_2$

$3x_1 + 5x_2 \leq 15$

$5x_1 + 2x_2 \leq 10$ $\quad \forall \; x_i \geq 0$

Sol:

Max $Z = 5x_1 + 3x_2 + 0x_3 + 0x_4$

$3x_1 + 5x_2 + x_3 + 0x_4 = 15$

$5x_1 + 2x_2 + 0x_3 + x_4 = 10$; $\forall \; x_i \geq 0$

Here we added an extra variable (slack variable), to balance both the equations.

		C_j	5	3	0	0
C_B	x_B	b	x_1	x_2	x_3	x_4
0	x_3	15	3	5	1	0
0	x_4	10	5	2	0	1

coffients of max Z

x_B costs coefficients of basic variables

by which variables unit vector matrix is formed

RHS of linear eq

coefficients of both equations

$$\boxed{Z_j = \Sigma C_{B_i} \times x_i}$$

	x_1	x_2	x_3	x_4
Z_j	$(3\times0 + 5\times0)$	$(5\times0 + 2\times0)$	$(1\times0 + 0\times0)$	$(0\times0 + 0\times1)$
Z_j	0	0	0	0
C_j	5	3	0	0
$Z_j - C_j$	−5	−3	0	0

C_B	x_B	b	x_1	x_2	x_3	x_4	Min Ratio
0	x_3	15	3	5	1	0	$\frac{15}{3} = 5$
0	x_4	10	5	2	0	1	$\frac{10}{5} = 2$ (min)
z_j			0	0	0	0	
c_j			5	3	0	0	
$z_j - c_j$			-5	-3	0	0	

5 → key element

2 → outgoing

↑ entering variable

= previous − 3 × new outgoing

C_B	x_B	b	x_1	x_2	x_3	x_4	min ratio
0	x_3	9	0	19/5	1	-3/5	$\frac{9}{19/5} = \frac{45}{19}$
5	x_1	2	1	2/5	0	1/5	$\frac{2}{2/5} = 10$

- Divide entire outgoing row by key element to make it 1
- Then do (previous row) − (3 * new outgoing row) as the row above current outgoing row.

z_j : $(0\times0 + 5\times1)$, $\left(0\times\frac{19}{5} + 5\times\frac{2}{5}\right)$, $(0\times1 + 5\times0)$, $\left(0\times\frac{-3}{5} + 5\times\frac{1}{5}\right)$

	x_1	x_2	x_3	x_4	min
z_j	5 (key element)	2	0	1	$\frac{45}{19}$ ← outgoing
c_j	5	3	0	0	10
$z_j - c_j$	0	-1	0	1	

↑ entering variable

C_B	x_B	b	x_1	x_2	x_3	x_4
3	x_2	$\frac{45}{19}$	0	1	$\frac{5}{19}$	$-\frac{3}{19}$
5	x_1	$\frac{20}{19}$	1	0	$\frac{-2}{19}$	$\frac{5}{19}$
Z_j			5	3	$\frac{5}{19}$	$\frac{16}{19}$
C_j			5	3	0	0
$Z_j - C_j$			0	0	$+\frac{5}{19}$	$\frac{16}{19}$

Step here since → $\boxed{Z_j - C_j \geq 0}$; we obtained optimal solution

And $x_1 = \frac{20}{19}$, $x_2 = \frac{45}{19}$

Max $Z = 5x_1 + 3x_2$

$= 5\left(\frac{20}{19}\right) + 3\left(\frac{45}{19}\right) = \frac{100}{19} + \frac{135}{19} = \frac{235}{19}$

$\therefore$ Maximum $Z = \frac{235}{19}$

Example :

$$\max Z = 3x_1 + 2x_2 + 5x_3$$

$$x_1 + x_2 + x_3 \leq 9$$

$$2x_1 + 3x_2 + 5x_3 \leq 30$$

$$2x_1 - x_2 - x_3 \leq 8 \quad ; \quad x_i \geq 0$$

Sol :

$$\max Z = 3x_1 + 2x_2 + 5x_3 + 0x_4 + 0x_5 + 0x_6$$

$$x_1 + x_2 + x_3 + x_4 + 0x_5 + 0x_6 = 9$$

$$2x_1 + 3x_2 + 5x_3 + 0x_4 + 0x_5 + 0x_6 = 30$$

$$2x_1 - x_2 - x_3 + 0 \cdot x_4 + 0 \cdot x_5 + x_6 = 8$$

Iteration 1 :

C_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	min ratio
0	x_4	9	1	1	1	1	0	0	$9/1 = 9$
0	x_5	30	2	3	5	0	1	0	$30/5 = 6$
0	x_6	8	2	-1	-1	0	0	1	$8/-1 = -8$
		z_j	0	0	0	0	0	0	
		c_j	3	2	5	0	0	0	
		$z_j - c_j$	-3	-2	-5	0	0	0	

↑ entering variable

Iteration 2 :

C_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	min ratio
0	x_4	3	$\frac{3}{5}$	$\frac{2}{5}$	0	1	$\frac{-1}{5}$	0	$\frac{3}{\frac{1}{5}} = 5$
5	x_3	6	$\frac{2}{5}$	$\frac{3}{5}$	1	0	$\frac{1}{5}$	0	$\frac{6}{\frac{2}{5}} = 15$
0	x_6	14	$\frac{12}{5}$	$\frac{-2}{5}$	0	0	$\frac{1}{5}$	1	$\frac{14}{\frac{12}{5}} = \frac{35}{6}$
		z_j	2	3	5	0	1	0	
		c_j	3	2	5	0	0	0	
		$z_j - c_j$	-1	1	0	0	0	0	

Iteration 3 :

		c_j	3	2	5	0	0	0	
C_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	min ratio
3	x_1	5	1	$\frac{2}{3}$	0	$\frac{5}{3}$	$\frac{-1}{3}$	0	
5	x_3	4	0	$\frac{1}{3}$	1	$\frac{-2}{3}$	$\frac{1}{3}$	0	
0	x_6	2	2	-2	0	-4	1	1	
		$z_j - c_j$	0	5/3	0	$\frac{5}{3}$	$\frac{2}{3}$	0	

all $z_j - c_j$ values are 0. So, step

And,

$x_1 = 5$, $x_2 = 0$, $x_3 = 4$.

x_2 ↓ didnot appear in the table

Max $z = 3x_1 + 2x_2 + 5x_3$

$= 3(5) + 2(0) + 5(4) = \underline{35}$

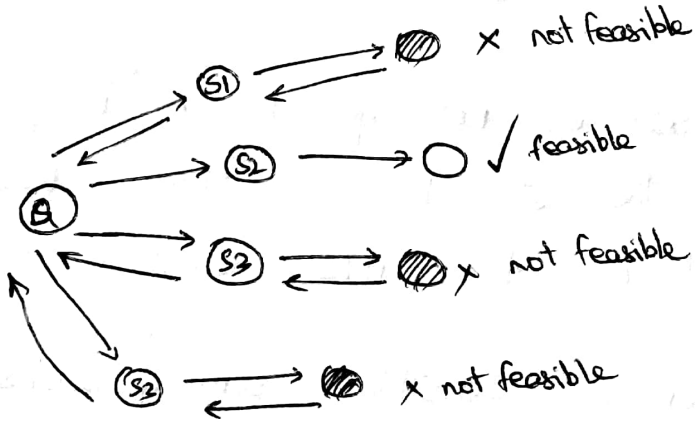
∴ Maximum z value = 35

MODULE-5 : BACKTRACKING

*. BACKTRACKING :

- Backtracking is a technique based on Algorithm to solve a bruteforce problem.
- It uses recursive calling to find the solution by building a solution step by step increasing values with time.
- It removes the solution that doesnot give rise to the solution of the problem based on the constraints given to solve the problem.
- Backtracking algorithm is applied to some specific types of problems,
  1) Decision problem used to find a feasible solution of problem
  2) optimisation problem used to find a feasible / best solution.
  3) Enumeration problem used to find a set of solutions.
- In backtracking, the algorithms tries to find a sequence path to the solution which has some small checkpoints from where the problem can backtrack it if no feasible, solution is found for the problem.

Example :



State Space Tree

*. Sum of Subsets Problem :

- To find subset of elements that are selected from a given set whose sum adds up to a given number m.
- Constraints : non-negative values, no duplicate values in set.
- Backtracking Algorithm :

Ex: Assume given set of 6 elements, say $w[1], \ldots w[6]$ as $w[1:6] = 4, 9, 11, 13, 14, 18$ with $n = 6$. Given target sum of the subset is $m = 27$. Find subset of elements of a given set $w[1:6]$ whose sum adds up to given m using Backtracking approach?

Sol: Algorithm :

1. Start with an empty set.
2. Add the next element from the list to the set.
3. If the subset is having sum m, then stop with that subset as solution.
4. If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find most suitable value.
5. If the subset is feasible (sum of subset $< M$) then go to step 2.
6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop solution. (No solution case)

Sol:

Assume, x

1/0	1/0	1/0	1/0	...	1/0	...
?	?	?	?	?	?	. . ? . .
1	2	3	4	5	6	... ..

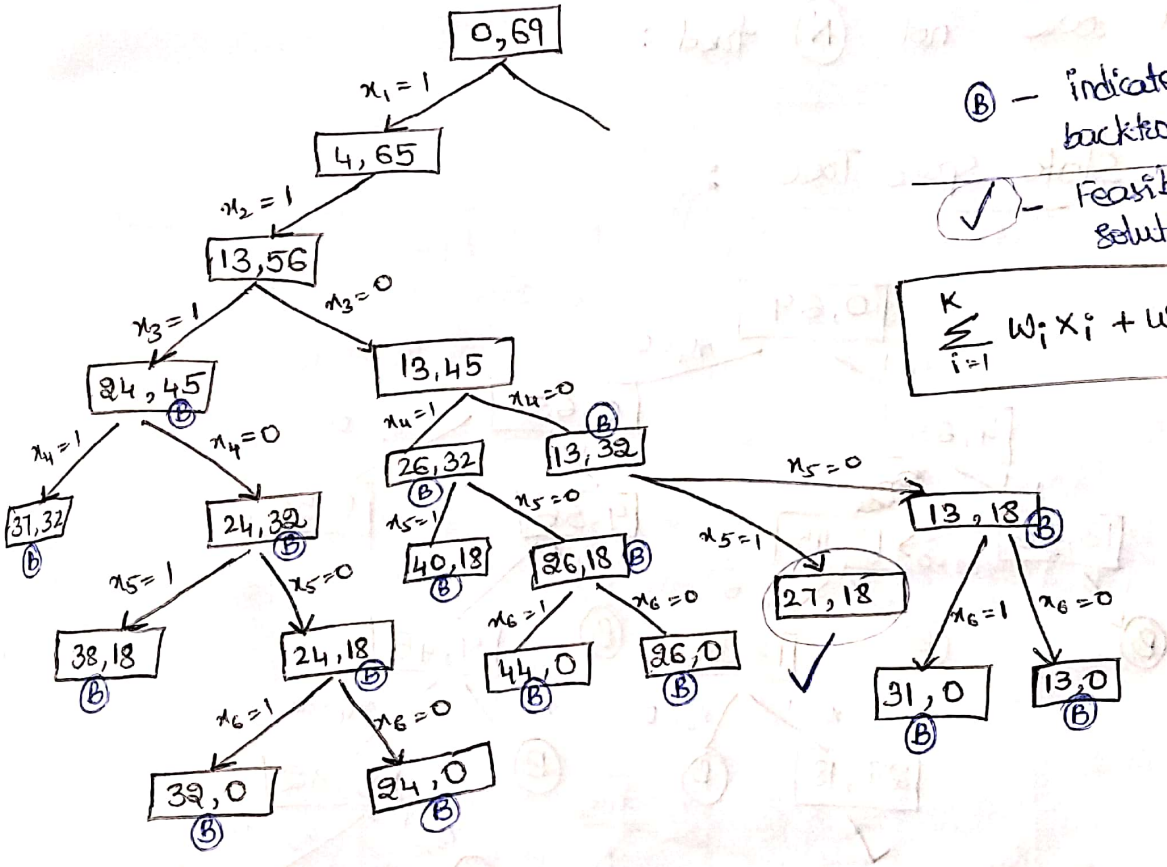
path traces / values considered

- Since we are following backtracking approach, we follow depth-first search by generating (included) object.

Total paths $= 2^n = 2^6 = 64$

$w[1:6] = \{4, 9, 11, 13, 14, 18\}$, total $= 4+9+11+13+14+18 = 69$

STATE SPACE TREE :



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

$\Rightarrow 4+9+14 = 27$

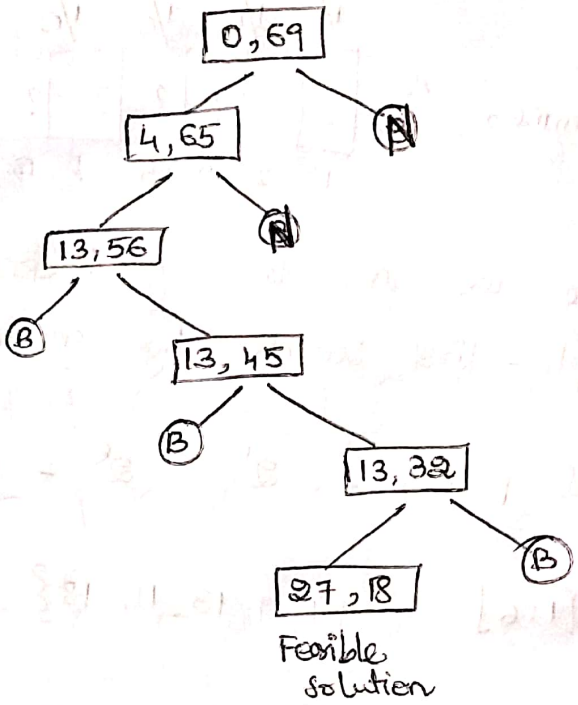
w	4	9	11	13	14	18

∴ One Feasible solution backtracked $= \{4, 9, 14\}$

$\equiv \{1, 1, 0, 0, 1, 0\}$

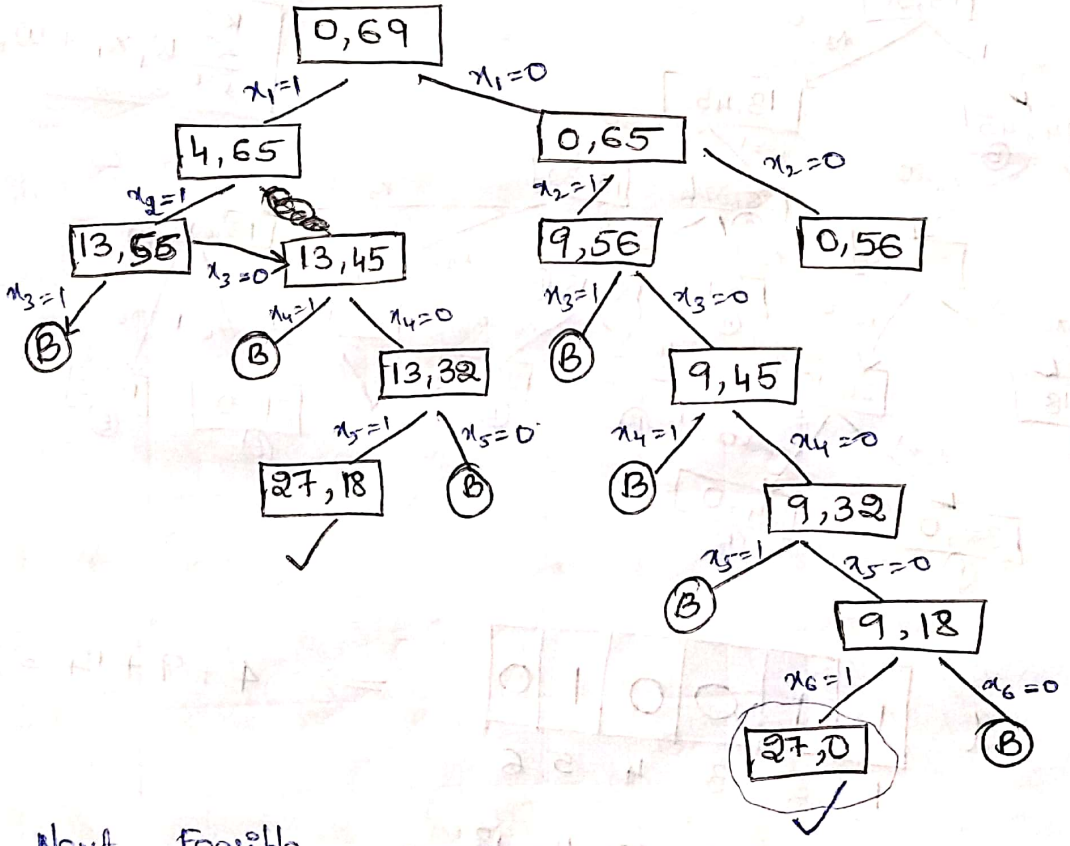
corresponding x value

updating space tree with Feasible path :



For next Feasible solution, we check for Backtracking paths that are not (N) tried :

State Space Tree :



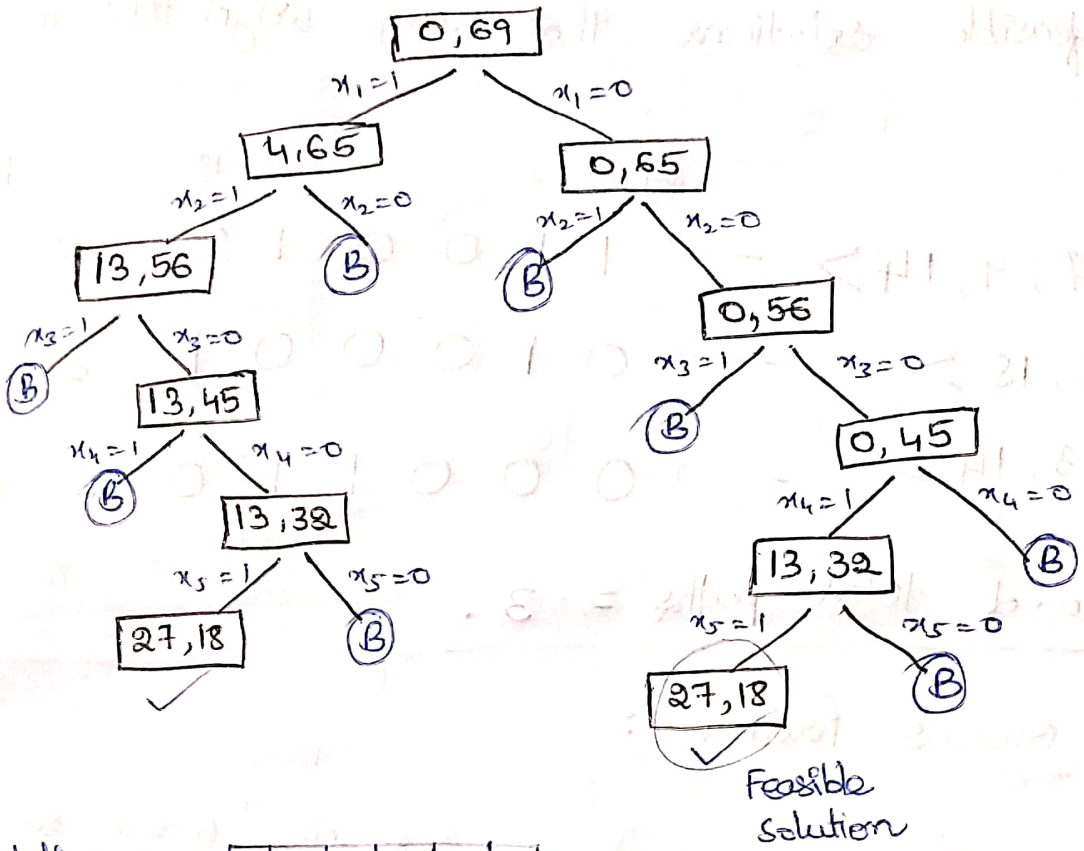
∴ Next Feasible solution :

x	0	1	0	0	0	1
	1	2	3	4	5	6
	4	9	11	13	14	18

$\Rightarrow 9+18=27$

$\{9, 18\}$

Another state space tree :



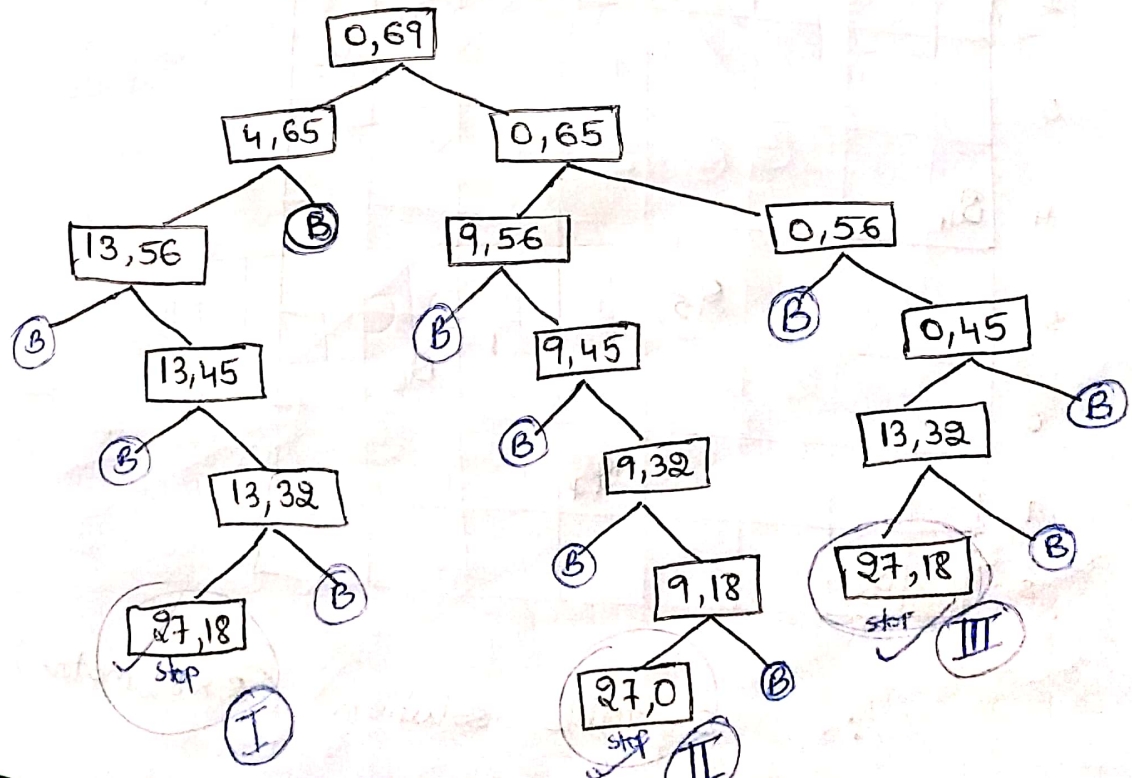
solution : x

0	0	0	1	1	0
1	2	3	4	5	6
4	9	11	13	14	18

→ 13 + 14 = 27

∴ path : {0, 0, 0, 1, 1, 0} & value = {13, 14}

Final state space tree with all Feasible paths :



After performing backtracking, finally, we obtained 3 feasible solutions that can weight total = m = 27 i.e.

	wi: 4	9	11	13	14	18	total
$\langle 4, 9, 14 \rangle \longrightarrow$	1	1	0	0	1	0	$\simeq$ 27
$\langle 9, 18 \rangle \longrightarrow$	0	1	0	0	0	1	$\simeq$ 27
$\langle 13, 14 \rangle \longrightarrow$	0	0	0	1	1	0	$\simeq$ 27

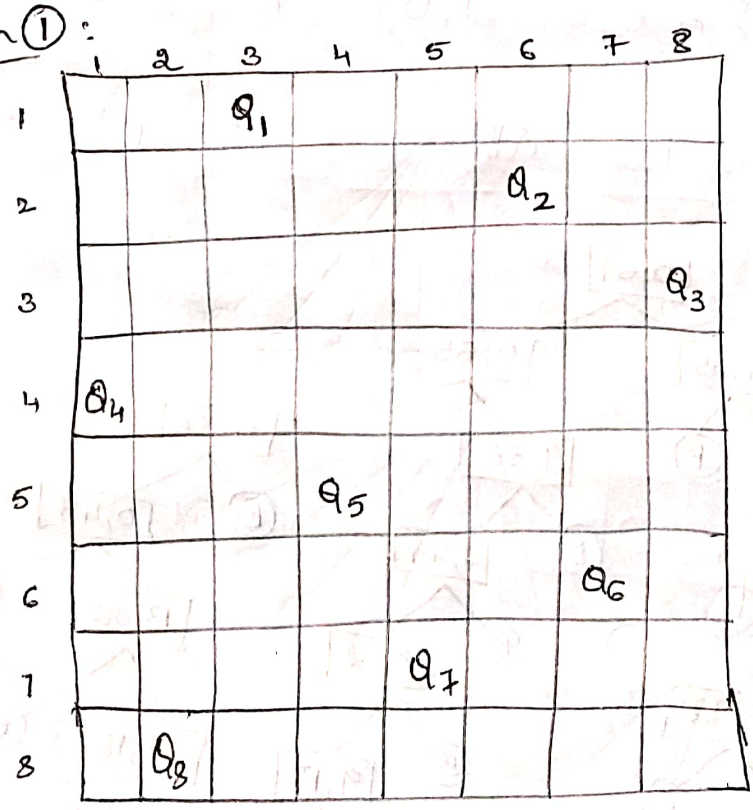
∴ No. of total paths = 3.

---

*. 8-Queen's Problem :

Problem : Place 8-queens in 8×8 chess board so that no two of them can attack i.e. no two of them are on same row, same column or diagonal.

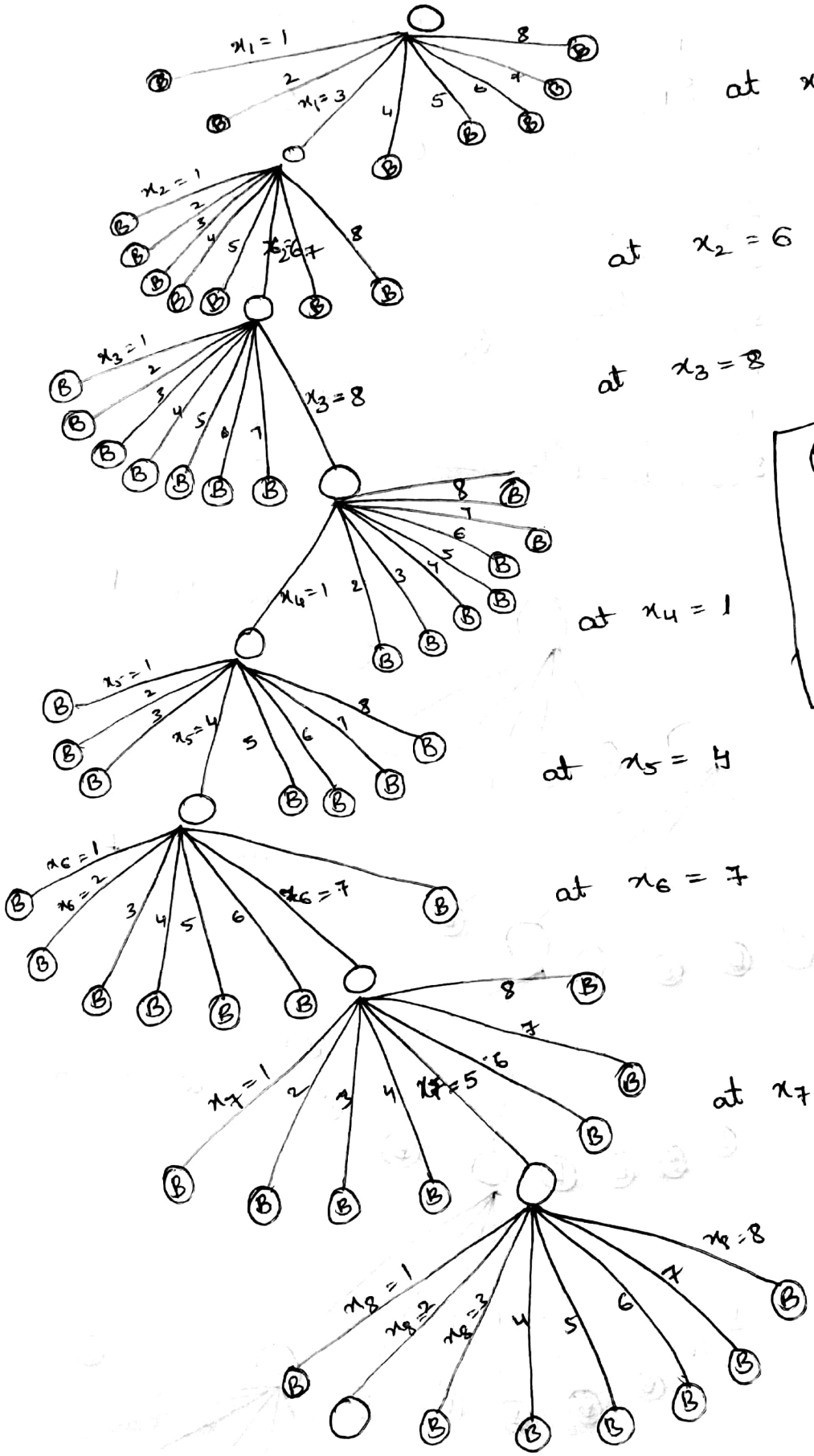
Solution ① :



Q_i - i^{th} Queen

∴ 8-tabb solution (8×8 matrix)

State Space Tree using Backtracking:



at $x_1 = 3$

at $x_2 = 6$

at $x_3 = 8$

at $x_4 = 1$

at $x_5 = 4$

at $x_6 = 7$

at $x_7 = 5$

at $x_8 = 2$

(B) - Solution not feasible so, backtracked

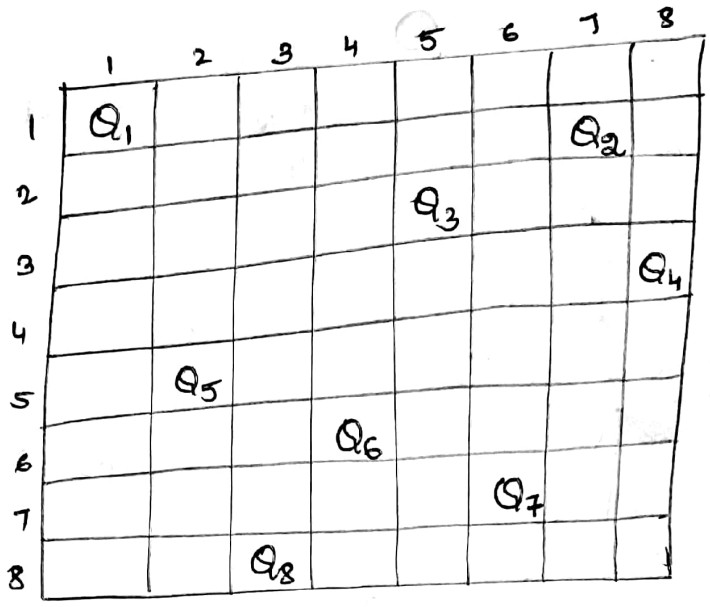
O - path found, solution feasible

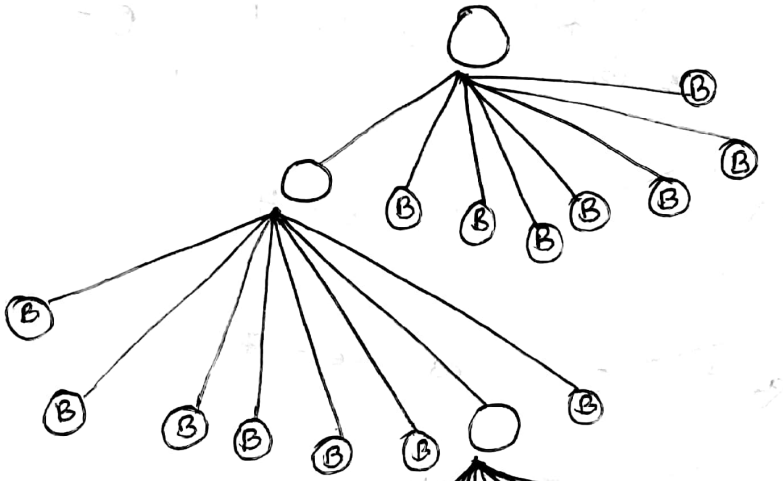
$\therefore$ X

1	2	3	4	5	6	7	8
3	6	8	1	4	7	5	2

is the 8-tuple solution for 8-queen problem.

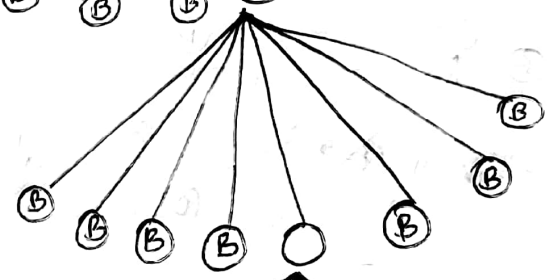
Solution 2 :



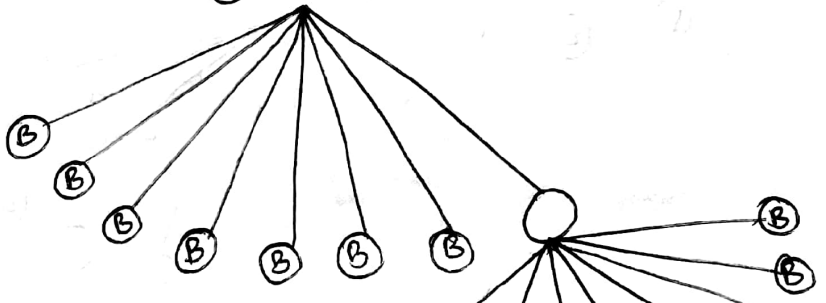


at $x_1 = 1$

at $x_2 = 7$

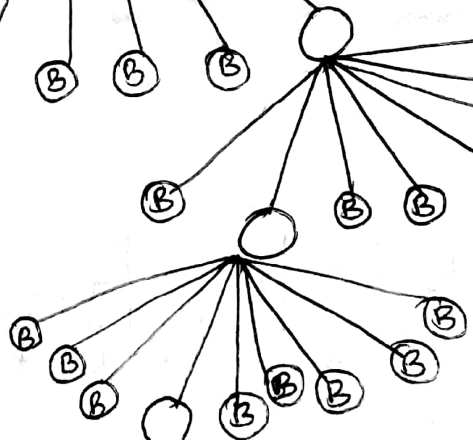


at $x_3 = 5$

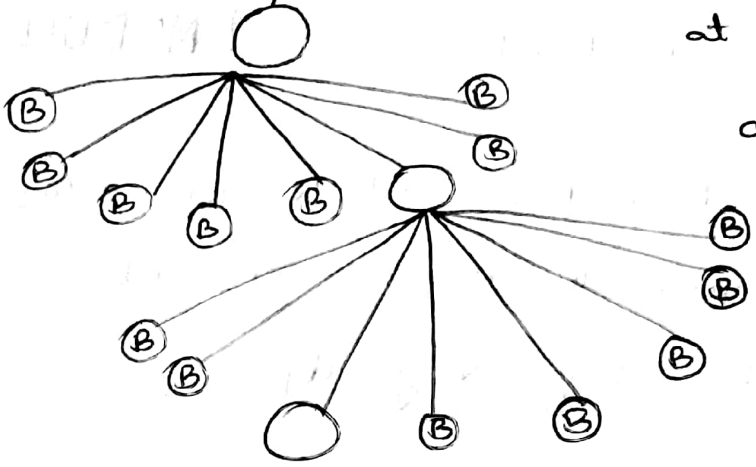


at $x_4 = 8$

at $x_5 = 2$



at $x_6 = 4$



at $x_6 = 4$

at $x_7 = 6$

at $x_8 = 3$

$\therefore$

1	7	5	8	2	4	6	3
1	2	3	4	5	6	7	8

is the 8-tuple solution.

following 3 conditions : (valid/non-attacking placement)

Queens at (i, j) (k, l)

1. same row $(i = k)$
2. same column $(j = l)$
3. $|i-k| = |j-l|$ (diagonal along)

} shouldnot be placed at these places.

- For a 8-queen problem, there are 92 possible ways to put a queen on 8×8 chess board. Out of them 12 unique solutions determine the Queen positions by placing them at valid positions.

*. ASSIGNMENT PROBLEM : [MODULE-6]

- Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment.
- It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.
- For each job, we choose a worker with lowest cost for that job, from list of unassigned workers (take minimum entry from each column).

Ex: Consider the given cost Matrix C. Solve this Job Assignment Problem using Branch & Bound approach?

$$\text{Cost Matrix, } C = \begin{bmatrix} 8 & 1 & 6 & 7 \\ 5 & 3 & 2 & 6 \\ 4 & 7 & 1 & 7 \\ 6 & 5 & 8 & 3 \end{bmatrix}$$

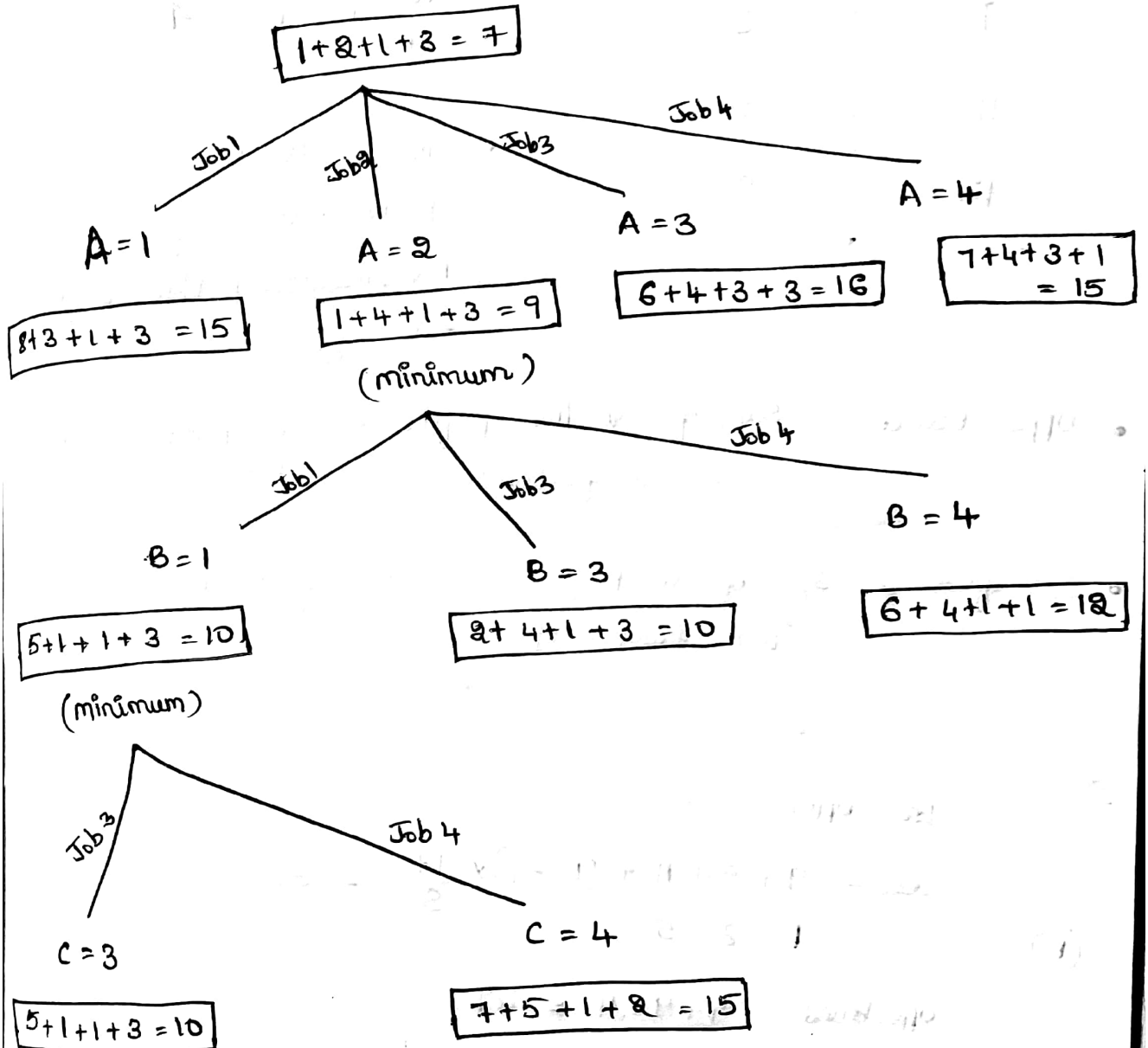
Sol:

$$C = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} \text{Job1} & \text{Job2} & \text{Job3} & \text{Job4} \end{array} \\ \begin{bmatrix} 8 & 1 & 6 & 7 \\ 5 & 3 & 2 & 6 \\ 4 & 7 & 1 & 7 \\ 6 & 5 & 8 & 3 \end{bmatrix} \end{array}$$

Lower Bound = Sum of minimum values of every row

= minimum value of row A + minimum value of row B + min C + min D

Lower Bound = 1 + 2 + 1 + 3 = 7 (root node)



∴ Upper Bound = 10

	A	B	C	D
assigned with	Job2	Job1	Job3	Job4
Costs :	1	5	1	3

Upper bound = 1 + 5 + 1 + 3 = 10

*. KNAPSACK PROBLEM (Branch & Bound approach) : {0/1}

Ex:

Profits	Weights
9	1
9	3
11	5
17	8

- Finding fixed size solution.

Knapsack weight, $m = 12$

No. of items, $n = 4$

- It is a maximization problem. Converting into minimization problem & putting -ve sign.

- Upper Bound = Sum of all the profits (without fraction) = $\sum_{i=1}^{n} p_i x_i \leq m$

- Cost factor = Sum of all profits (with fraction of weight) = $\sum_{i=1}^{n} p_i x_i$

Sol:

Let upper = ∞

(i) Cost = $9 + 9 + 11 + (12-9) \times \frac{17}{8} = 35.375$

1 3 5

Upper bound = $9+9+11 = 29$

Put -ve sign $\Rightarrow$ $C = -35.375$, $U = -29$ $\rightarrow$ upper = -29

(ii) Cost = $9 + 11 + (12-8) \times \frac{17}{8} = 28.5$

(1st object not included) 3 5

Upper Bound = ~~9~~ $9 + 11 = 20$ $\Rightarrow$ $C = -28.5$, $U = -20$

Consider Least cost - Branch Bound & continue.

(iii) Cost (2nd object not included) $= 9 + 11 + (12-6) \times \frac{17}{8} = 32.75$

$1 \quad 5$

Upper Bound $= 9 + 11 = 20$

$\Rightarrow C = -32.75$ $\quad \Rightarrow$ upper $= -29$

$U = -20$

(iv) Cost $= 9 + 9 + (12-4) \times \frac{17}{8} = 35$

(3rd object is not included) $\quad 1 \quad 3 \quad 8$

Upper Bound $= 9 + 9 = 18 + 17$ ✓ because no fraction it is considered $= 35$

$\Rightarrow C = -35$, $U = -35$ $\Rightarrow$ upper $= -35$

(updated because we got smaller) upper bound

(v) Cost $= 9 + 9 + (12-4) \times \frac{11}{5} = 35.6$

(4th object not included in bag) $\quad 1 \quad 3$

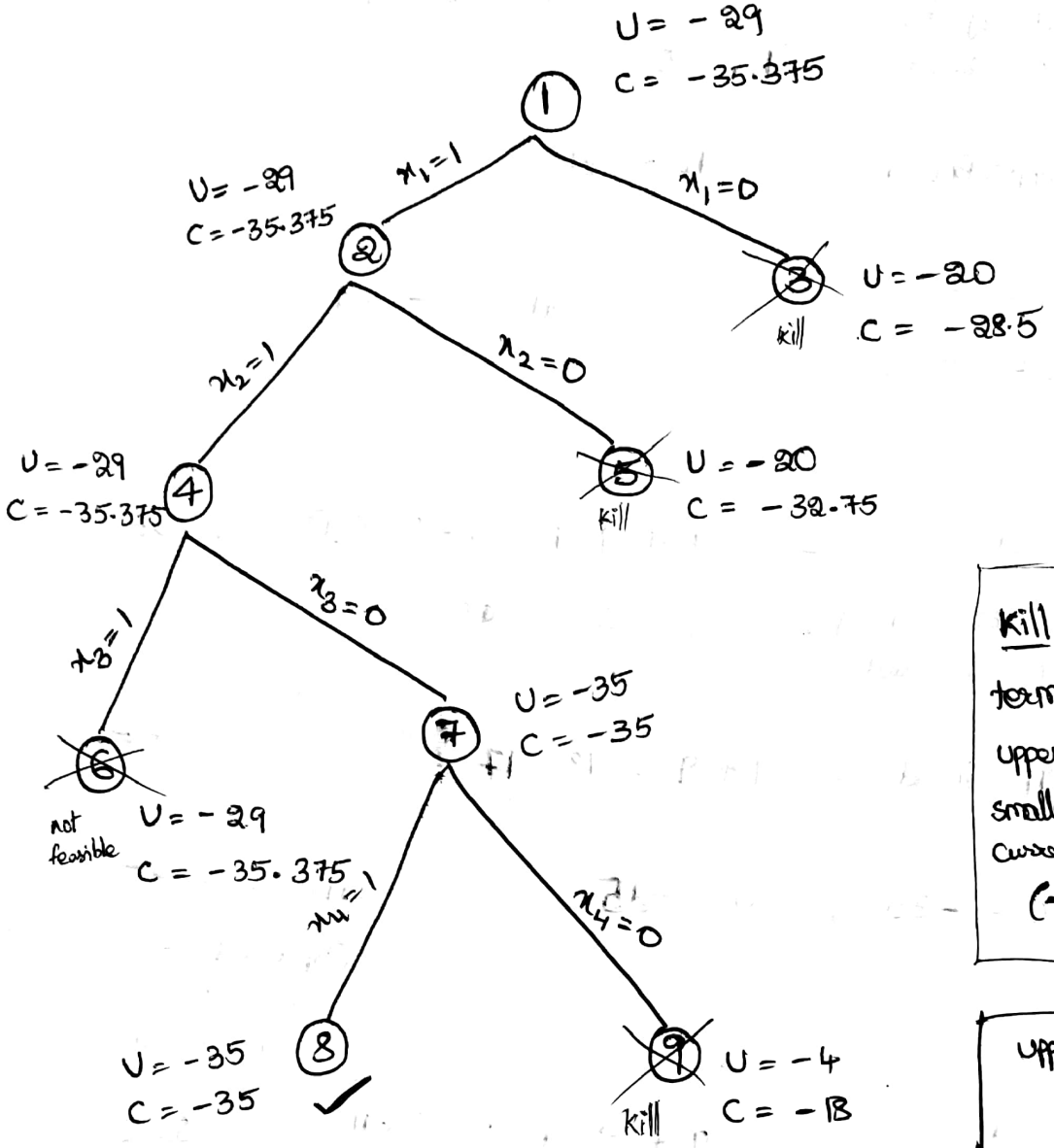
Upper Bound $= 9 + 9 = 18$

$\Rightarrow C = -35.6$, $U = -18$ $\Rightarrow$ upper =

(vi) Cost $= 9 + 9 = 18$ $\quad C = -18$

(3rd & 4th both are not included) $\quad$ Upper Bound $= 1 + 3 = 4$ $\Rightarrow U = -4$

State space Tree :



Kill - it is terminated because upper value is smaller than current value. (-ve value terms)

upper = ~~∞~~
= ~~-29~~
= -35

∴

x_1	x_2	x_3	x_4
{ 1	1	0	1 }

profits : 9 + 9 + 0 + 17 = 35

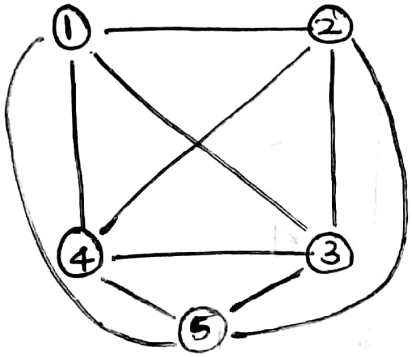
weights : 1 + 3 + 0 + 8 = 12

Therefore, objects 1, 2, 4 are included & object 3 not included with maximum profits are 35.

TRAVELLING SALESMEN PROBLEM : [Branch & Bound approach]

*

Ex:



	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Sol: Consider the Adjacency Matrix & reduce it.

- Write all minimum values from rows. Subtract all minimum values from respective rows.
- Write all minimum values from columns. Subtract all minimum values from respective columns.
- Cost effecitive = Sum of all minimum values of rows, columns.
- For x to y path, make all xth row ∞ & yth column values as infinity. & y to x value also ∞.
- Then minimize / reduce that matrix and calculate effective cost of that matrix.
- Repeat the same for entire traversal from initial node to destination node & construct a state space tree.
- Everytime at child nodes in each level follow least - cost Branch & Bound (Consider minimum cost) approach.
- Repeat the same. & update upper bound value when reached leaf node (last node).

	1	2	3	4	5	min row
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

⟹ subtract from

∞	10	20	0	1	
13	∞	14	2	0	
1	3	∞	0	2	
16	3	15	∞	0	
12	0	3	12	∞	

total = 21

∞	10	20	0	1	
13	∞	14	2	0	
1	3	∞	0	2	
16	3	15	∞	0	
12	0	3	12	∞	

min. col: 1 0 3 0 0 / total = 4

Effective Cost = 21 + 4 = 25

$C_r = 25$

Cost of root Node

initial

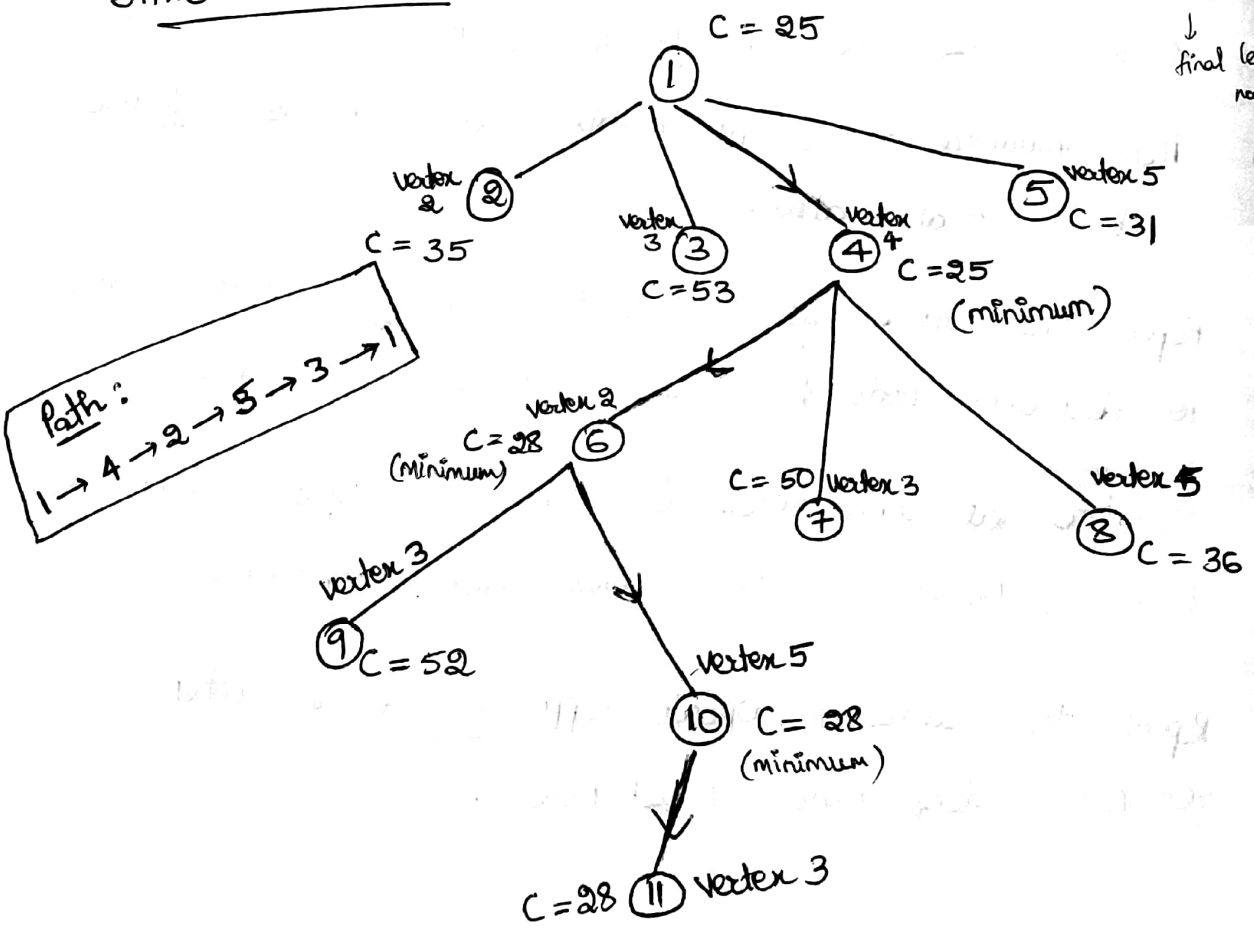
upper = ~~∞~~

28

↓

final leaf node

STATE SPACE TREE :



At Node 2 :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & 11 & 2 & 0 & 0 \\ 0 & \infty & \infty & 0 & 2 & 0 \\ 15 & \infty & 12 & \infty & 0 & 0 \\ 11 & \infty & 0 & 12 & \infty & 0 \\ \hline 0 & & 0 & 0 & 0 & 0 \end{bmatrix}$$

Effective cost = 0

$C_2 = 0$

Cost at Node $\hat{j}$ = Cost $(i, \hat{j})$ + Cost of next Node + Effective Cost at that node

Cost = $c(1,2) + C_r + C_j = 10 + 25 + 0 = 35$

At Node 3 :

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix} \longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & 0 \\ 12 & \infty & \infty & 2 & 0 & 0 \\ \infty & 3 & \infty & 0 & 2 & 0 \\ 15 & 3 & \infty & \infty & 0 & 0 \\ 11 & 0 & \infty & 12 & 0 & 0 \\ \hline 11 & 0 & & 0 & 0 & 0 \end{bmatrix}$$

subtracting $\Rightarrow$ in Col

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & 0 \end{bmatrix}$$

$C_3 = 0 + 11 = 11$

Cost = $c(1,3) + C_r + C_3$

$= 17 + 25 + 11$

Cost = 53

At Node 4 :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

$C_4 = 0$

Cost = $c(1,4) + C_r + C_4$

$= 0 + 25 + 0$

$= 25$

At Node 5 :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

$C_5 = 5$

Cost $= c(1,5) + C_8 + C_5$

$= 1 + 25 + 5$

$= 31$

At Node 6 : (minimum obtained is Node 4 (vertex 4))

our root node is 4 now, & matrix is

Cost of root $= C_4 = \boxed{25 = C_8}$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \\ 0 & & 0 & & 0 \end{bmatrix} \begin{matrix} 0 \\ 0 \\ \\ 0 \\ \hline 0 \end{matrix}$$

Cost $= c(4,2) + C_8 + C_6 = 3 + 25 + 0 = 28$

At Node 7 :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

$C_7 = 13$

Cost $= c(4,3) + C_8 + C_7$

$= 12 + 25 + 13$

$= 50$

At Node 8 :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

$C_8 = 11$

Cost $= c(4,5) + C_8 + C_8$

$= 0 + 25 + 11$

$= 36$

At Node 9 : Minimum obtained at Node 6 (vertex 2)

our root node is 6 now with cost, $C_6 = \boxed{C_8 = 28}$

and matrix is,

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \\ \\ 2 \\ \\ 11 \end{matrix} \longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

13 $\qquad$ 0 $\qquad$ 0

$C_9 = 13$

$$\text{Cost} = c(2,3) + C_8 + C_9 = 11 + 28 + 13 = 52$$

At Node 10 :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \\ \\ 0 \\ \\ + \end{matrix}$$

$0 \qquad 0 \qquad + \quad 0$

$C_{10} = 0$

$$\text{Cost} = c(2,5) + C_8 + C_{10}$$
$$= 0 + 28 + 0 = 28$$

At Node 11 :

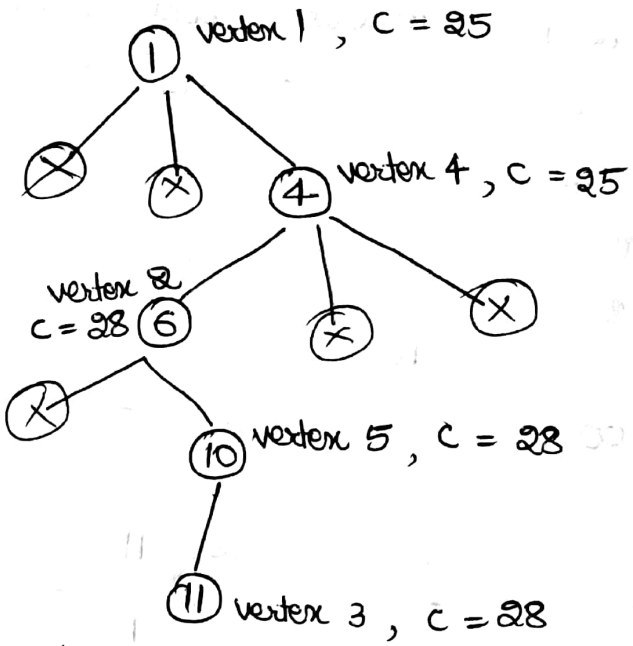
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix} \longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} C \\ D \\ C \\ 0 \\ 0 \\ + \end{matrix}$$

$0 \quad 0 \quad 0 \quad 0 \quad +\ 0$

$C_{11} = 0$

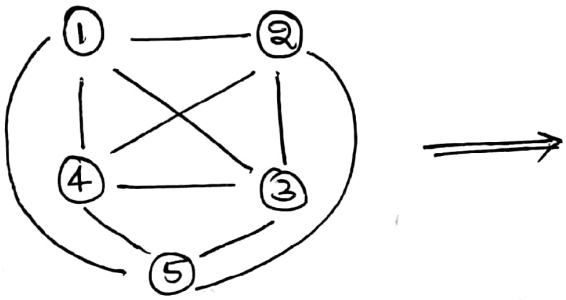
$$\text{Cost} = c(5,3) + C_8 + C_{11}$$
$$= 0 + 28 + 0 = 28$$

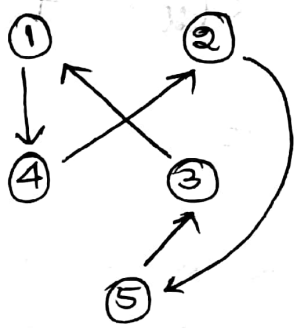
Final state space tree :



Path is :

$1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$





upper = ~~∞~~ = 28

with Cost = 28

---

*. STRING MATCHING : (Boyer Moore Algorithm)

- Same like various pattern matching algorithms, we match the characters of the pattern with the given text from right to left and then left to right.
- In this algorithm, we have a pre-processing step where we construct a bad match table.
- This algorithm works with the idea of a "Bad match rule" and a good shift rule.

- The Bad match rule is we calculate the values for each & every character using a formula in the pattern.
- The Good shift rule is using the values of those characters of the pattern we keep on shifting the pattern position to the right corresponding to the value of the table.

* Example :

Given Text, T = WELCOME TO SURANA COLLEGE

and pattern, P = SURANA

Sol : Length of Text = $n = |T| = 22$

Length of Pattern = $m = |P| = 6$

last character = A in pattern

Now, Bad Match Table :

positions	0	1	2	3	4	*
	S	U	R	A	N	*
	5	4	3	~~2~~ 6 ~~0~~	1	6

last character from pattern length is same as length of pattern.

while this algorithm, if we find any character other than given pattern, we are going to consider that character as * & value is length of pattern.

Value = length − index − 1 (length of pattern)

- If any characters comes twice, thrice, ... keep updating its value in Bad Match Table
- If it is last character, it is length of pattern

S = 6 − 0 − 1 = 5 , U = 6 − 1 − 1 = 4 , R = 6 − 2 − 1 = 3

N = 6 − 4 − 1 = 1 , A (1st time) = 6 − 3 − 1 = 2 , A (2nd time) = 6 − 5 − 1 = 0

S	U	R	A	N	A
5	4	3	6	1	6

Now, while matching the pattern with text, consider first m characters (m is length of pattern) & keep checking from right to left.

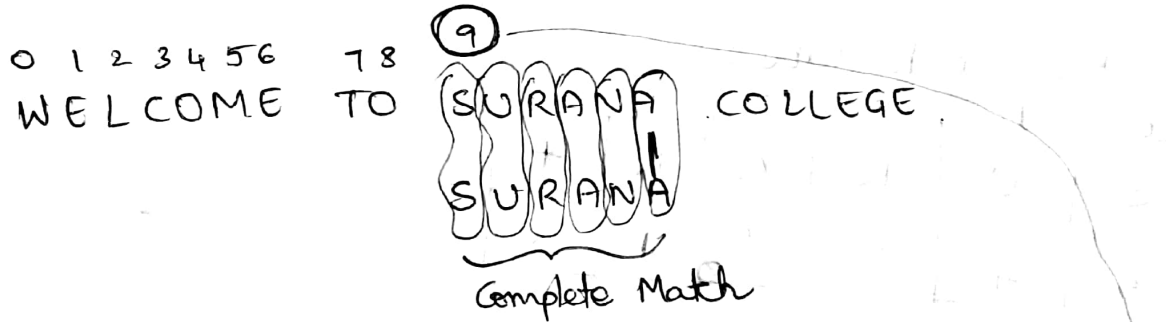
WELCOME TO SURANA COLLEGE

SURANA — Mismatch, So search in Bad Match Table
'M' is not found. So, move by 6 characters →

- If mismatch, then move the pattern towards right by length of * value characters. if that character is not found in table.
- If the character is found in table, then move towards right by its value in table.

WELCOME TO SURANA COLLEGE

SURANA — Mismatch, So search in Bad Match table,
'R' is found. So, shift by 3 values



So, given pattern matches with the text at position 9

∴ P present in T at 9th position

Ex: T = WELCOME TO VIT COLLEGE

P = COLLEGE
index 0 1 2 3 4 5 6

n = |T| = 19

m = |P| = 7

C	O	L	E	G	*	
6	5	~~4~~ 3	~~2~~ 07	1	7	

last character

value = length - index - 1

C = 7 - 0 - 1 = 6

O = 7 - 1 - 1 = 5

L = 7 - 2 - 1 = 4
1st

L = 7 - 3 - 1 = 3 (overwritten) update
2nd

E = 7 - 4 - 1 = 2
1st

G = 7 - 5 - 1 = 1

E = 7 - 6 - 1 = 0 (update)
2nd

* value = length of pattern
= 7

WELCOME TO VIT COLLEGE
COLLEGE
× ✓ shift by 7

WELCOME TO VIT COLLEGE
COLLEGE
× shift by 5

0 1 2 3 4 5 6 7 8 9 10 11 (12)
WELCOME TO VIT COLLEGE
COLLEGE
Complete Match

∴ P present in T at 12th position.

---

*. NP - Completeness :

Polynomial Time

Linear Search — n
Binary search — log n
Insertion sort — n^2
Merge sort — n log n
Matrix Multiplication — n^3

Exponential Time

0/1 knapsack — 2^n
Travelling Salesmen — 2^n
Sum of subset — 2^n
Graph Coloring — 2^n
Hamiltonian Cycle — 2^n