SURYA Gold

Date Page

① Explain Asymptotic notations with example.

A Big oh (O)

It is a measure of the longest amount of time that it could possibly take for an algorithm to complete execution. It gives the upper bound of an algorithm's running time or growth rate of function. (worst case)

Eg:- given $f(n) = 5^n + 2$ prove that $f(n) = O(n)$

Here $g(n)$ is n

$|f(n)| \leq c|g(n)| \quad \forall n \geq no$

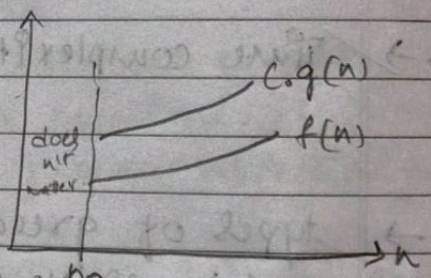
$|5n+2| \leq c|n| \quad \forall n \geq no$

to find value of c & no

$|5n+2| \leq 7|n|$

$c = 7$ & $no = 1$

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



Big Omega (Ω)

" " lowest amount

" lower bound

(best case)

Eg:- given $f(n) = 5n+2$ Prove that $f(n) = \Omega(n)$

Here $g(n)$ is n

omega constraint is $|f(n)| \geq c|g(n)| \quad \forall n \geq no$

$|5n+2| \geq c|n| \quad \forall n \geq no$

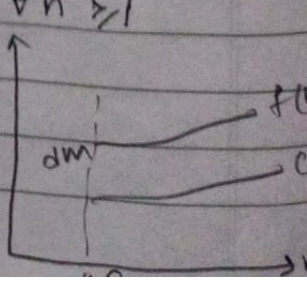
Since $5n+2$ is always greater than $5n$ we can choose $c = 5$ & $no = 1$

$5n+2 \geq 5n$

$c = 5 \quad no = 1$

$\therefore f(n) = \Omega(n)$ is proved

$\forall n \geq 1$



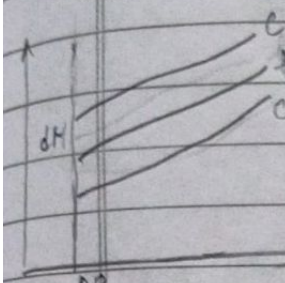
Big Theta (Θ)

It gives the upper bound & lower bound of an algorithm's running time or growth rate of function (average case)

Eg :-

$f(n) = 3n+2$ prove that $f(n) = \Theta(n)$

Constraint is $C_1|g(n)| \leq |f(n)| \leq C_2|g(n)|$



$C_1 \cdot n \leq 3n+2 \leq C_2 \cdot n$

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

$C_1 = 3$, $C_2 = 4$ & $n_0 = 2$

∴ $f(n) = \Theta(n)$ is proved

(2) Mergesort algorithm & Eg.

Algorithm

Step (1) :- If (low < high) then
(2) :- mid = (low + high) / 2
(3) :- call mergesort (a, low, mid)
(4) :- call mergesort (a, mid+1, high)
(5) :- call mergesort (a, low, mid, high)
End if
(6) :- Exit

Merge (a, low, mid, high)

Step (1) :- i = low
(2) :- j = mid+1
(3) :- k = low
(4) :- while (i <= mid) && (j <= high) do
(5) :- if a[i] < a[j]
c[k] = a[i]
k = k+1

```
        i = i+1
    else
        c[k] = a[j]
        k = k+1
        j = j+1
    End while
```

⑥ :- while(i<=mid)
```
    c[k] = a[i]
    k = k+1
    i = i+1
end while
```
⑦ :- while (j<=high)
```
    c[k] = a[j]
    k = k+1
    j = j+1
end while
```
⑧ :- for i=low to k+1
```
    a[i] = c[i]
end for
```
⑨ :- return

41, 32, 11, 92, 66, 74, 87, 38

0	1	2	3	4	5	6	7
41	32	11	92	66	74	87	38

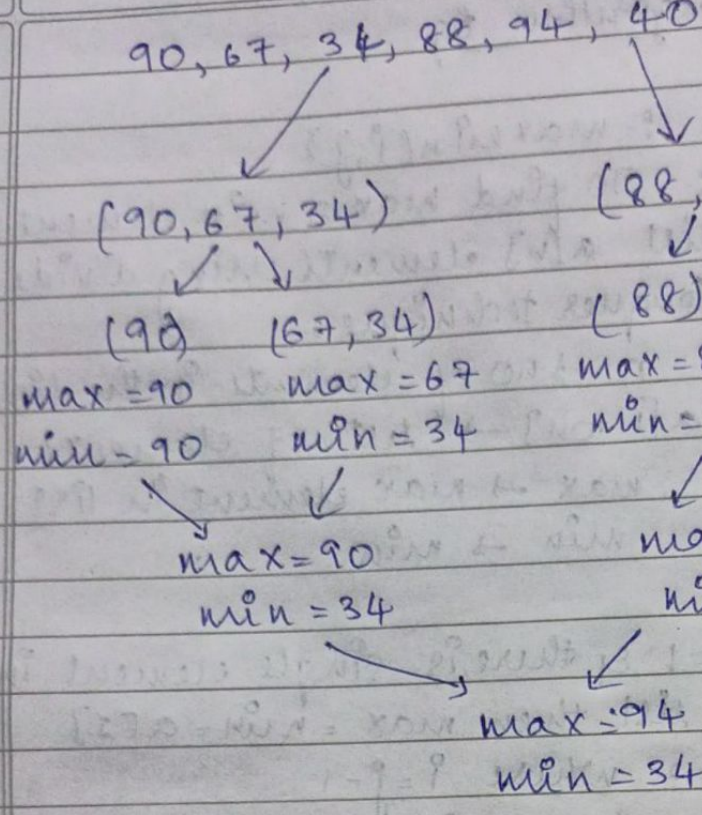
low = 0 high = 7 mid = $\frac{low+high}{2}$

$= \frac{0+7}{2} = \frac{7}{2} = 3.5$

(3) Minmax algorithm Eg.

Algorithm :- maxmin(i,j)
Purpose :- To find max & min element of the list a[n] elements using divide & conquer technique.
Input :- n → no of elements in the list
a[i to n] → list of elements
output :- max → max element in list
min → min.

Logic
① If n=1 & there is single element in the list then max = min = a[1]
② If n=2 where i=j-1
If the element is smaller then min = a[i] max = a[j] and
If the first element is greater then min = a[j], max = a[i]
③ if n>2 then divide & conquer technique is used.
(a) List is divided into 2 halves where mid = (i+j)/2
(b) recursive call to fun for first half
maxmin(i, mid);
max1 = max, min1 = min;
(c) recursive call to function for second half
maxmin(mid+1, j);
(d) two solutions are combined
if (max < max1)
max = max1;
if (min > min1)
min = min1;



④ Recursive binary search (A, first, last, key)

Input : Given an array A of n element is sorted order
first is index of first element
last is index of last element
key is element to be searched

```
if ( first = last)                    // If there is only one element
{
  if ( key = = A[first])
    return first ;
  else
    return -1;                        // Unsuccessful
}
else
{
  mid = ( first + last) / 2 ;
  if ( key = = A[mid])
```

```
else
  if (key < A[mid])
return R Binarysearch(A, first, mid-1) key)
else
return R Binarysearch (A, mid+1, last, key)
}
}
```

Eg:- Let us select the 14 elements as shown below

0	1	2	3	4	5	6	7	8	9	10	11	12	13
05	08	10	12	15	18	20	22	30	35	40	45	50	60

Key = 35

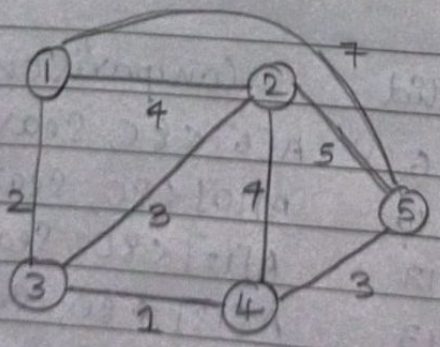
First	Last	Mid	Comparisons
0	13	6	A[6] < 35, Search right Part
7	13	10	A[10] > 35, Search left Part
7	9	8	A[8] < 35, Search left Part
9	9	9	A[9] == 35, Search ends Element found

Key = 80

First	last	Mid	Comparisons
0	13	6	A[6] < 80, Search right Part
7	13	10	A[10] < 80, Search right Part
11	13	12	A[12] < 80, Search right Part
13	13	13	A[13] < 80, Search right Part
14	13	-	Index 14 is out of range & hence element not found

(5) Prim's Algorithm

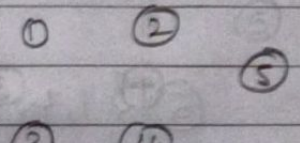
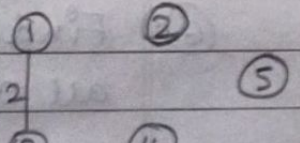
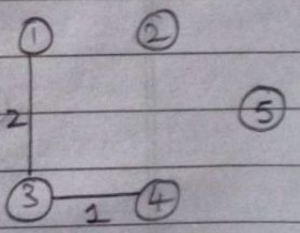
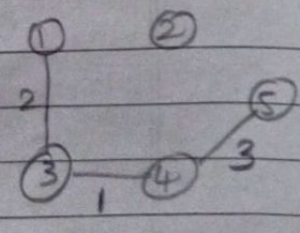
```
Prims(c, n)

// Assume that G is connected, Undirected & weighted graph
// Input :- The cost adjacency matrix c & no of vertices n.
// output :- Minimum weight Spanning tree T
{
 for(i=1; i<=n; i++)
  visited[i] = 0;        // Initia all vertices as unvisited
  u=1;                   // consider vertex 1 as starting vertex
 visited[u] = 1;
 while (there is still unchoosen vertex) do
 {
  Let (u,v) be the lightest edge b/w any chosen u to any chosen vertex v
  visited[v] = 1;
 T = union(T, <u, v>);   // add edges (u,v) to Spanning tree
 }
  return T;
}
```



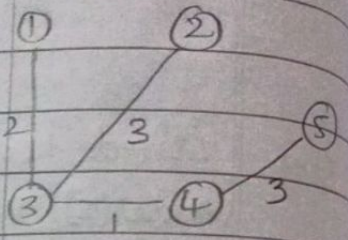
SURYA Gold

Date_____ Page_____

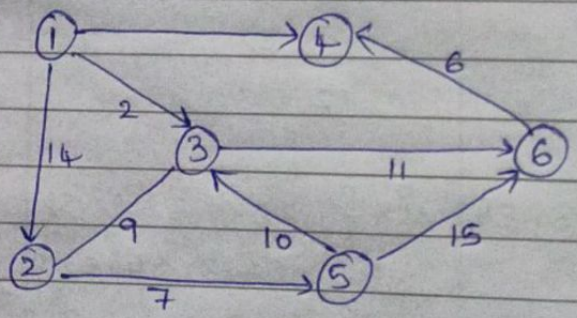
C	1	2	3	4	5
1	-	4	2	-	7
2	4	-	3	4	5
3	2	3	-	1	-
4	-	4	1	-	3
5	7	5	-	3	-

Step	visited vertices[S]	unvisited vertices[G]	Minimum edge from S to G	Illustrations
①	-	{1,2,3,4,5}	-	
②	1 (Assumption)	{2,3,4,5}	min{<1,2>, <1,3>, <1,4>, <1,5>} = min(4, 2, 7) = 2 = <1, 3> = 2	
③	{1,4}	{2,3,5}	min{<1,2> <1,3> <1,5> <4,2> <4,3> <4,5>} = min(4, 2, 7, 4, 1, 3) = 1 = <4, 3> = 1	
④	{1,4,3}	{2,5}	min{<1,2> <1,5> <4,2> <4,5> <3,2 X 3,5>} = min(4, 7, 4, 3, 3} = 3 = <4, 5> = 3	

(5)	{1, 4, 3, 5}	{2}	{<1, 2> <4, 2> <3, 2> <5, 2>} min(4, 4, 3, 5) = 3 = <3, 2> = 3
(6)	{1, 4, 3, 5, 2}	-	-	Total Cost = 1 + 2 + 3 + 3 = 9



(6) Find the shortest distance from node 1 to all other nodes using Dijkstra's algorithm.



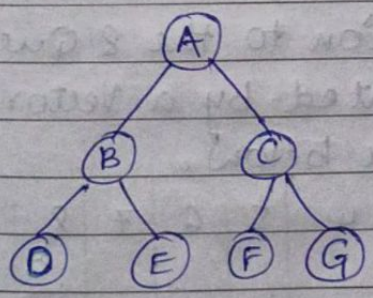
(7) Write recursion algorithm for preorder traversal & apply it to the following complete binary tree.

Purpose :- To perform preorder traversal of binary tree t

Input :- Binary tree t

Output :- The node elements in preorder. Each node of t has three fields lchild, data & rchild

```
Void Preorder (struct treenode *t)
{
  if(t)
  {
  visit(t);
  Preorder (t→lchild);
  Preorder (t→rchild);
  }
}
```



Value in root	Action
A	-
-	Print(A)
C	-
B	-
-	Print (B)
E	-
B	-
-	Print(B)
-	Print (E)
G	-
F	-
C	-
-	Print(C)
-	Print(F)
-	Print(G)
-	

Preorder traversal is

A B C D E C F G

(8) Note on graph coloring

(a)

→ In graph theorel, graph coloring is a Special Cale of graph labelling.

→ It is an assignment of labels called 'colors' to elements of a graph subject to certain constraints.

→ In its simplest form it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same colour this is called vertex coloring.

(b) 8 Queens Problem

The Problem is to place 8 Queens on 8 x 8 chessboard so that no two Queens attack each other by being in the same row or in the same column or on the same diagonal.

Eg :-

One of the Solution to the 8 Queens Problem is represented by a Vector(1, 7, 4, 6, 8, 2, 5, 3) is shown below.

	1	2	3	4	5	6	7	8
1	Q_1							
2							Q_2	
3				Q_3				
4						Q_4		
5								Q_5
6		Q_6						
7					Q_7			
8			Q_8					

→ General backtracking Method

```
Procedure backtrack(n)

// This is a program scheme which delivribes
     the back tracking process
// All solutions are generated in x(1:n)
& printed as soon as they are determined.
{
  k = 1
  while (k != 0) do
{
 if ( there remains an untried x[k] ∈ T
      ( (x[1], x[2] ---- x[k-1])
 and Bk ( x[1] - x[k]) = true then
{
 if (x[1] - x[k]) is a path to an answer node
    then print (x[1], ---- x[k])
  k → k+1                    // Consider the next set
}
 else
   k → k-1                   // back track to previous set
}
}
```

→ Sum of Subset

given a set S = {s1 ---- sn} of n positive integers & a positive integer M. The Problem is to find all subsets of the given set whose sum is equal to M.

Eg:- If S = (11, 13, 24, 7) & M = 31 then desired subsets are

Subset 1 = (11, 13, 7)
Subset 2 = (24, 7)

Explicit Constraint :

$x_i \in \{j \mid j$ is an integer & $1 \le j \le n\}$

Implicit constraint :

(i) $\sum_{i=1}^{k} w_i = m$ (ii) $x_i < x_{i+1}$, $1 \le i \le k$

→ For the knapsack Problem $n = 3, m = 20$, $(P_1, P_2, P_3) = (25, 24, 15)$ & $(w_1, w_2, w_3) = (18, 15, 10)$ find the feasible & optimal solution.

Sol Step ① : Arrange the objects in increasing order of weights

objects	3	2	1
weights	10	15	18
Profit	15	24	25

remaining capacity(rc)	object selected	weight	Fraction of x added to knapsack
20	3	10	1 full unit
20-10=10	2	15	10/15 = 2/3 unit
0	1	18	0 unit

$(x_1, x_2, x_3) = (0, 2/3, 1)$

Profit earned $= x_1P_1 + x_2P_2 + x_3P_3$

$= 0 * 25 + 2/3 * 24 + 1 * 15$

$= 16 + 15$

$= 31$

Step ② :- Arrange the objects in decreasing order of Profits

objects	1	2	3	
weights	18	15	10	
Profit	25	24	15	

RC	object Selected	weight	Fraction of x added to knapsack
20	1	18	1 full Unit
20-18=2	2	15	2/15 Unit
0	3	10	0 Unit

$(x_1, x_2, x_3) = (1, 2/15, 0)$

Profit earned $= x_1p_1 + x_2p_2 + x_3p_3$

$= 1 * 25 + 2/15 * 24 + 0 * 15$

$= 25 + 3.2$

$= 28.2$

Step ③ :- Arrange the object in decreasing order of P_i/W_i

object 1 : $25/18 = 1.39$
object 2 : $24/15 = 1.6$
object 3 : $15/10 = 1.5$

RC	object Selected	weight	Fraction of x added to knapsack
20	2	15	1 full Unit
20-15=5	3	10	5/10 = 1/2 Unit
0	1	18	0 Unit

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

Profit earned $= x_1p_1 + x_2p_2 + x_3p_3$

$= 0 * 25 + 1 * 24 + 1/2 * 15$

$= 24 + 7.5$

$= 31.5$

Step 4 :- Arrange the objects in increasing order of p_i/w_i

Object 1 : $25/18 = 1.39$

Object 2 : $24/15 = 1.6$

object 3 : $15/10 = 1.5$

RC	object Selected	weight	Fraction of x
20	1	18	1 full Unit
20-18=2	3	10	2/10 = 1/5 Unit
0	2	15	0 Unit

$(x_1, x_2, x_3) = (1, 0, 1/5)$

Profit earned $= x_1p_1 + x_2p_2 + x_3p_3$

$= 1 * 25 + 0 * 24 + 1/5 * 15$

$= 25 + 3$

$= 28$

feasible solutions are :-

31

28.2

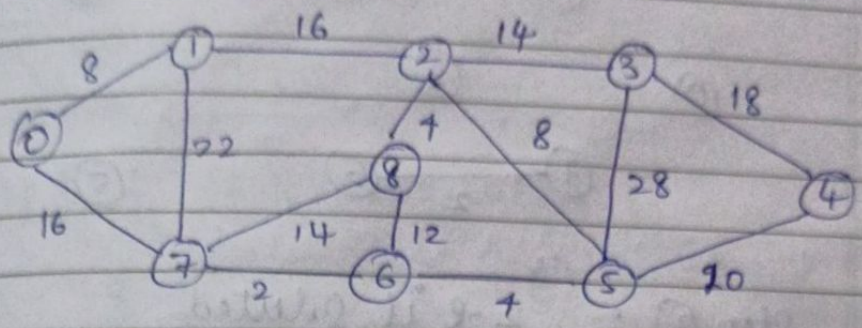
31.5

28

optimal solution :- 31.5

SURYA Gold
Date Page

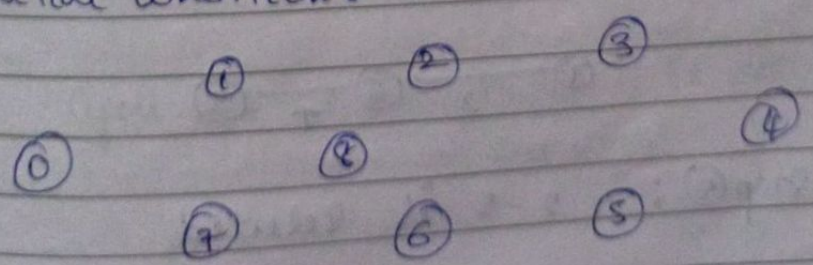
→ Minimum spanning tree using kruskal algorithm



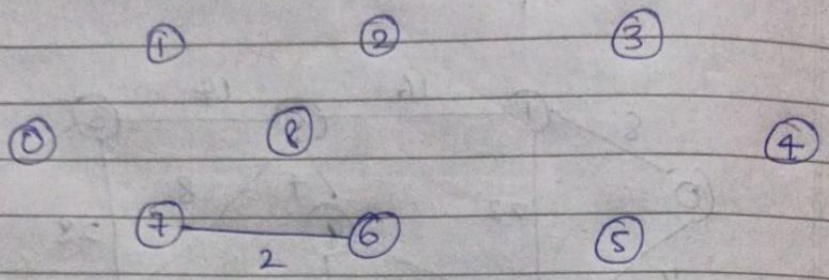
Sol Arranging the edges in increasing order of the weights.

Edge	cost
6-7	2
2-8	4
5-6	4
0-1	8
2-5	8
6-8	12
2-3	14
7-8	14
0-7	16
1-2	16
3-4	18
4-5	20
1-7	22
3-5	28

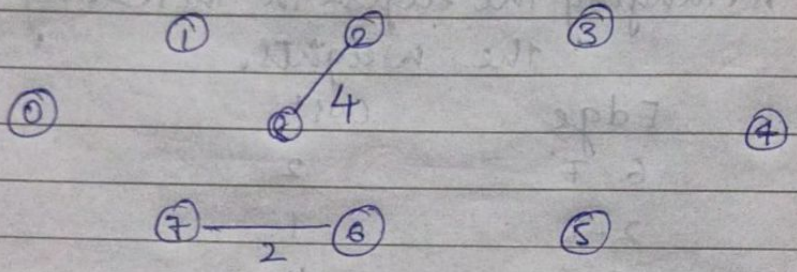
Initial condition:-



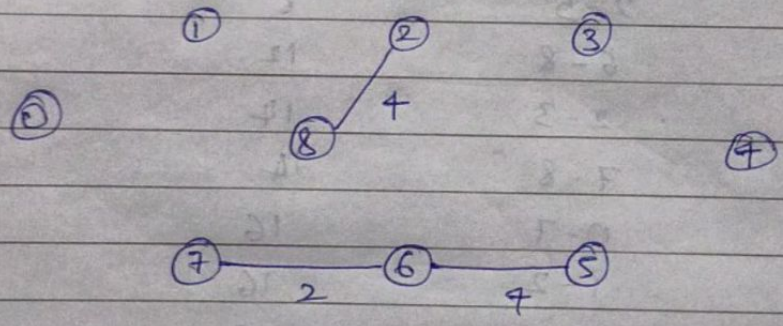
Step ① :- The cost of edge 6-7 is 2 so it is selected



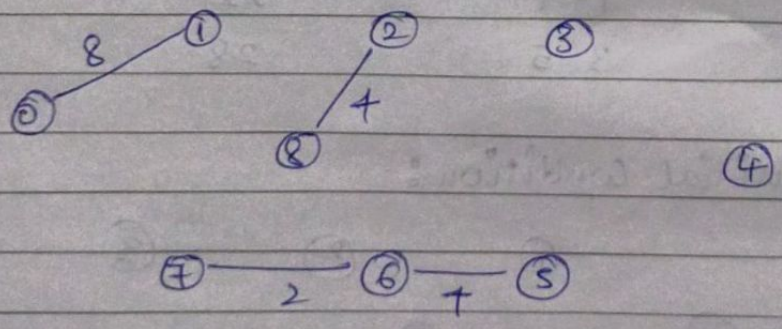
Step ② :- 2-8 is selected.



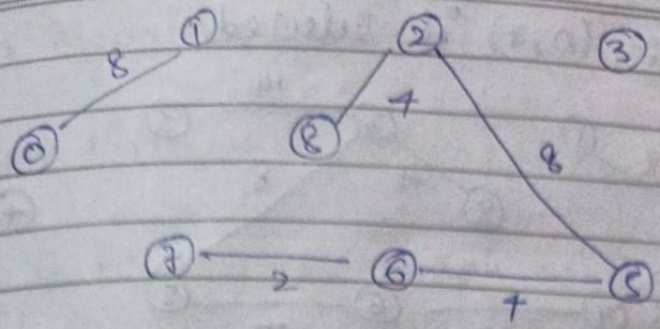
Step 3 :- 5-6 is selected



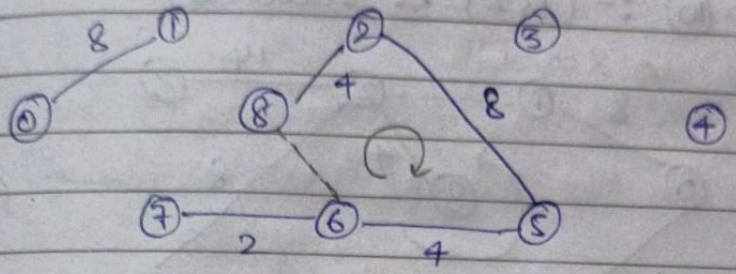
Step ④ :- 0-1 is selected



Step ⑤ :- 2-5 is selected

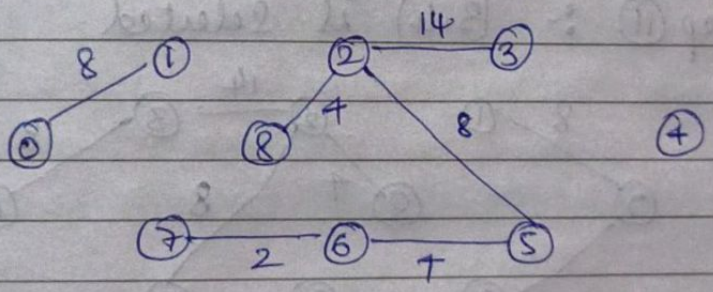


Step ⑥ :- (6,8) is selected

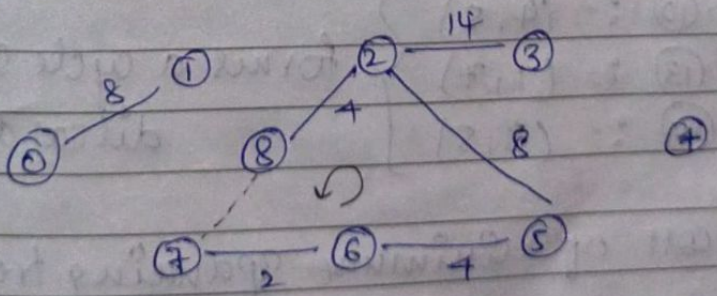


This will form a cycle so it is discarded

Step ⑦ :- (2,3) is selected

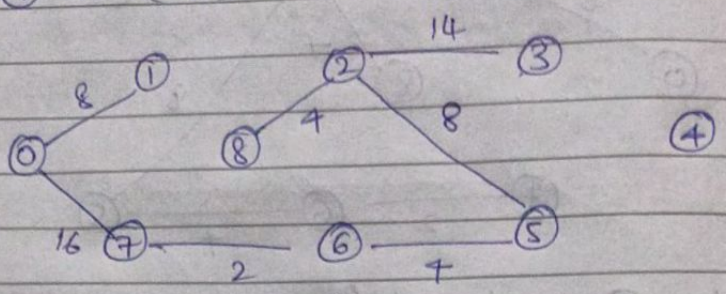


Step ⑧ :- (7,8) is selected

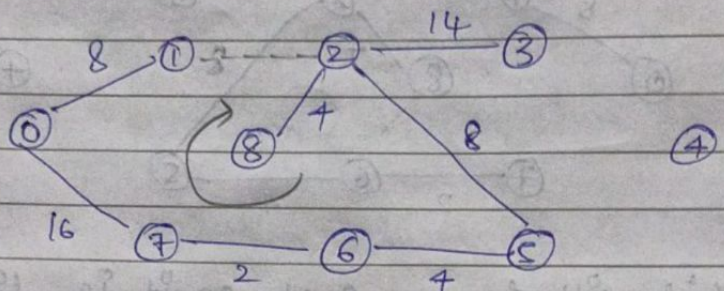


Cycle will be formed so it is dilcarded.

Step (9) :- (0,7) is selected

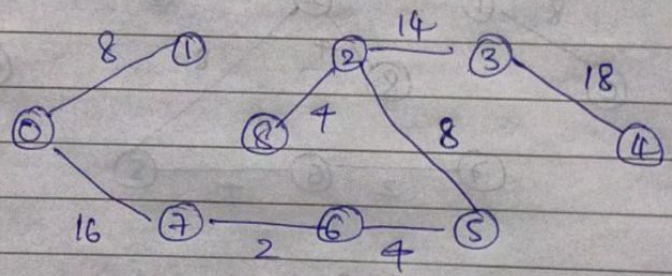


Step (10) :- (1,2) is selected



forms a cycle so it is discarded.

step (11) :- (3,4) is selected



Step (12) :- (4,5)
(13) :- (1,7)
(14) :- (3,5)
} forms a cycle so it is discarded.

The cost of minimum spanning tree is given by sum of weights of selected edges

$8+16+2+4+4+14+8+18 = 74$

→ DFS Algorithm

Depth First Search (G, n)

Let G be a given graph with n vertices
This algorithm traversed the G in Depth first search manner

```
{
 for i=1 to n do        // mark all vertices unvisited
 visited [i] = false
   if ( ! visited [i])
   Dfs(i);
}
Dfs(Vertex v)
 visited [v] = true
 for each vertex w adjacent to v do
   if (! visited [w])
 Dfs (w)
```

→ Recursive algorithm for post order traversal

Algorithm

Purpose :- To purpose postorder traversal of binary tree t

Input :- Binary tree t

Output :- The node elements in postorder. each node of 't' has tree fields lchild, data & rchild.

```
void postorder (struct tree node * t)
{
  if (t)
  {
```

```
        Postorder(t->lchild);
        Postorder(t->rchild);
        visit(t);
    }
}
```

→ Kruskal's Algorithm

```
kruskal's(E,n)
// Let E is the list of edges
// n is the no of vertices in a given
                    graph G
sort E in increasing order of their
        edge weights;
initially T = 0;
while(T does not contain n-1 edges)
{
find the minimum cost edge not yet
  considered in E & call it as (u,v)
if (u,v) does not form a cycle
   T = T+(u,v)                    //add (u,v) to T
  else
   delete (u,v)
}
 return T
```

→ Floyd's Algorithm

```
// Input : The cost/ weight matrix of a
          given graph G=(V,E)
// Output : The distance matrix D of the
          shortest paths lengths.
```

```
D ← C
for k ← 1 to n do
for i ← 1 to n do
for j ← 1 to n do
D[i,j] = min{ D[i,j], D[i,k] + D[k,j] }
end for
end for
end for
return D
```

DFS	BFS
① Data structure used is Stack.	① Data structure used is Queue
② Two vertex ordering is used.	② One vertex ordering is used.
③ Tree & back edges are present	③ Tree & cross edges are present
④ Used to check connectivity acyclicity & articulation points.	④ Used to check connectivity acyclicity & minimum edge paths.
⑤ Time efficiency is $O(\|V\|^2)$	⑤ Time efficiency is $\Theta(\|V\|^2)$

→ Multistage graph $G=(V,E)$ is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint set says $V_1, V_2 \cdots V_k$.

→ N Queens Problem.

Problem Statement :- N Queens should be placed in N x N matrix where in the following conditions should apply.

Implicit Constraints

(i) No two Queens should be placed in the same row
(ii) " " same column
(iii) " " same diagonal

Explicit Constraints

$S_i = \{1, 2, 3, 4\}$ $1 \le i \le n$

$1 \le x_i \le 4$

$i = 1, 2, 3, 4$

Solution Space = $8!$