

TABLE OF CONTENTS

TASK 1: SCENARIO.....	3
TASK 1: SOLUTION	4
a) Adjacency Matrix & Adjacency List	4
ADJACENCY MATRIX	4
ADJACENCY LIST	5
b) Pseudo-codes for the BFS, DFS and Dijkstra's Algorithm	6
Pseudo code for Breadth First Search (BFS) ALGORITHM	6
Pseudo code for Depth First Search (DFS) ALGORITHM	7
Pseudo code for Dijkstra's ALGORITHM	8
c) Step by step solution for the BFS, DFS and Dijkstra's of the give n graph.	9
Step by step solution of BFS ALGORITHM (<i>unweighted</i>)	9
Step by step solution of DFS ALGORITHM (<i>unweighted</i>)	16
Step by step solution of Dijkstra's Algorithm	26
d) Efficiencies for the BFS, DFS and Dijkstra's Algorithm	36
Time Complexity of BFS Algorithm	36
Time Complexity of DFS Algorithm	36
Time Complexity of DIJKSTRA Algorithm	36
TASK 2: SCENARIO.....	37
TASK 2: SOLUTION	38
a) Adjacency Matrix & Adjacency List	38
ADJACENCY MATRIX	38
ADJACENCY LIST	39
b) Pseudo-codes for the Prim's and Kruskal's Algorithm	40
Pseudo code for Prim's ALGORITHM	40
Pseudo code for Kruskal's ALGORITHM	41
c) Step by step solution for the Prim's and Kruskal's of the given graph.	42
Step by step solution of PRIM's ALGORITHM	42
Step by step solution of KRUSKAL's ALGORITHM	62
d) Efficiency of the Prim's and Kruskal's algorithms in terms of big-oh.	67
TIME COMPLEXITY OF PRIM'S ALGORITHM:	67
TIME COMPLEXITY OF KRUSKAL'S ALGORITHM:	68
TASK 3: SCENARIO.....	69

TASK 3: SOLUTION	69
a) All phases of insert operations	69
b) All phases of deleteMin operations	71
c) Complexity of Heap Sort in terms of Big-oh Notation	84
REFERENCES	85
Websites:	85
Books:	85

TASK 1: SCENARIO

Graph theory has many practical applications in the real world. One such application is to determine how a network of interconnected places can be optimized in order to ensure that commuters can travel as efficiently as possible.

For this specific assignment, the first graph problem is represented hereby:

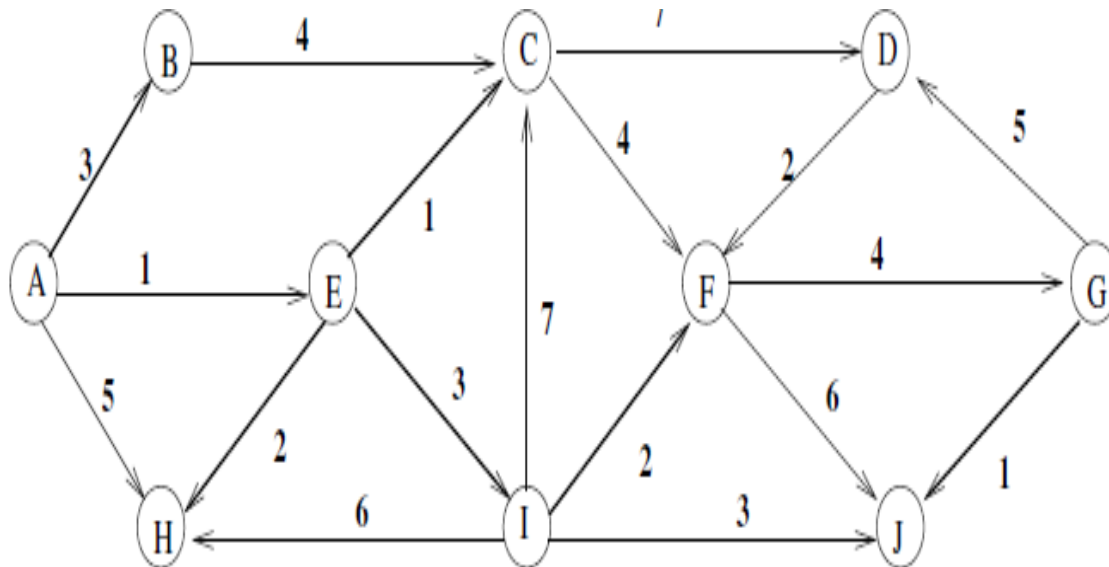


Fig. 1 Graph Theory Problem (for BFS, DFS & Dijkstra's SSSP)

The nodes in the above diagram represent the various places scattered across a country, while the edges represent the roads running between them. The weight of each edge represents the distance between the two places along the edge.

You have to **deliver** a solution for the problem mentioned above by completing the following tasks:

- Draw adjacency list and adjacency matrix of the graph given.
- Write the pseudo-codes for the BFS, DFS and Dijkstra's algorithm along with their explanation.
- Provide step by step solution for the BFS, DFS and Dijkstra's of the given graph. (For BFS and DFS algorithm consider the graph as unweighted one)
- Compute the efficiencies of the above algorithms in terms of big-Oh.

TASK 1: SOLUTION

a) Adjacency Matrix & Adjacency List

ADJACENCY MATRIX

ADJACENCY MATRIX										
	A	B	C	D	E	F	G	H	I	J
A	0	3	0	0	1	0	0	5	0	0
B	0	0	4	0	0	0	0	0	0	0
C	0	0	0	1	0	4	0	0	0	0
D	0	0	0	0	0	2	0	0	0	0
E	0	0	1	0	0	0	0	2	3	0
F	0	0	0	0	0	0	4	0	0	6
G	0	0	0	5	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0	0	0
I	0	0	7	0	0	2	0	6	0	3
J	0	0	0	0	0	0	0	0	0	0

ADJACENCY LIST

ADJACENCY LIST

A 0 → B 3 → E 1 → H 5 ⊗

B 0 → C 4 ⊗

C 0 → D 1 → F 4 ⊗

D 0 → F 2 ⊗

E 0 → C 1 → H 2 → I 3 ⊗

F 0 → G 4 → J 6 ⊗

G 0 → D 5 → J 1 ⊗

H 0 ⊗

I 0 → C 7 → F 2 → H 6 → J 3 ⊗

J 0 ⊗

b) Pseudo-codes for the BFS, DFS and Dijkstra's Algorithm

Pseudo code for Breadth First Search (BFS) ALGORITHM

S. No.	BFS Algorithm	Explanation
	BFS(G,s) /* G = (V,E) */	Graph G is represented by adjacency list G
1.	For each vertex u in V - {s}	Here u is the vertices of the graph G and V is the set of vertices.
2.	u.color = WHITE	Color of all the vertices will be white in color.
3.	u.distance = ∞	The distance of all the vertices is initialized with ∞ , except the root (source) vertex.
4.	u. predecessor = NIL	The parents of all the vertices are set as NIL.
5.	s.color = GRAY	The root (source) vertex will be gray in color.
6.	s.distance = 0	The distance of root vertex has been initialized with 0 as it is the first vertex to be processed.
7.	s. predecessor = NIL	The parent of the root vertex is set as NIL.
8.	While Q is equal to empty ($= \emptyset$)	Initially the queue Q is empty.
9.	ENQUEUE(Q,s)	Inserting the source vertex s in Queue.
10.	While Q is not empty ($\neq \emptyset$)	While loop has been put with the condition of non – empty queue (Q) until the queue is empty.
11.	u = DEQUEUE	The DEQUEUE elements are stored in u.
12.	For each v adjacent to u	For loop has been put for each vertex v which are adjacent to vertex u.
13.	if v.color = WHITE	In explored state, vertex v is in white color
14.	v.color = GRAY	Color of vertex v is now changed to gray color
15.	v.distance = u.distance + 1	The distance of vertex v is the sum of distance of parent and 1.
16.	u = DEQUEUE	The DEQUEUE elements are stored in u.
17.	v. predecessor = u	The parent of the vertex v is set as u.
18.	ENQUEUE(Q,v)	The vertex v is inserted in the queue (Q).
19.	u.color = BLACK	The color of the explored node is changed to black color.

Pseudo code for Depth First Search (DFS) ALGORITHM

S. No.	DFS Algorithm	Explanation
	DFS(G,s) /*G = (V,E)*/	Graph G is represented by adjacency list G
1.	For each vertex u in V	Here u is the vertices of the graph G and V is the set of vertices.
2.	do u.color = WHITE	Initialize color of each vertex is white in color.
3.	u. predecessor = NIL	Initially the parent of each vertex is NIL.
4.	time = 0	The starting time of the source node is zero
5.	For each vertex v adjacent to u	The for loop consider each vertex v in the adjacency list of u.
6.	do if u.color = WHITE	The condition if the color of the vertex is white
7.	then DFS - VISIT(u)	If the color is white, then DFS VISIT is applied on u.

S. No.	DFS - VISIT(u)	Explanation
8.	u.color = GRAY	When adjacent node u is visited or explored then its color changes from white to gray.
9.	time = time + 1	The starting time is increased by 1.
10.	For each vertex v adjacent to u	Each vertex v adjacent to u and visits v if it is white.
11.	do if v.color = WHITE	It examines whether the color of vertex is white or not and then it proceeds.
12.	then v.pred = u	The parent of visited node is updated.
13.	DFS-VISIT(u)	Again DFS-VISIT is applied on the vertex which was explored recently.
14.	u.color = BLACK	Finally the color of the vertex which is fully explored is set as black.

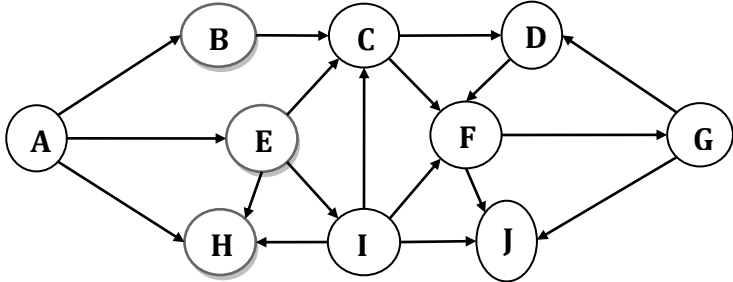
Pseudo code for Dijkstra's ALGORITHM

S. No.	Dijkstra Algorithm	Explanation
	DFS(G,s) INITIALIZE SINGLE-SOURCE	Graph G is represented by adjacency list G and the source vertex is s
1.	For each vertex v in V	Here for all the vertex, we set the vertex in graph
2.	v.distance = ∞	Initialize the distance of each vertex be infinity
3.	v.predecessor = NIL	Initially the parent of each vertex is NIL.
4.	s.distance = 0	Initially the distance of source vertex is set to 0.
5.	S = \emptyset	Initially the set S is set as null or empty.
6.	$Q \leftarrow V(G)$	Initially the min-priority queue Q contains all the vertices in V which is in ascending order by weight
7.	While $Q \neq \emptyset$	Initialize the min-priority queue Q
8.	u = EXTRACT-MIN(Q)	Each time when while loop is called a vertex u is extracted from queue Q
9.	S = S \cup {u}	Perform relaxation for each vertex v adjacent to u
10.	For each vertex v adjacent to u	While loop has been put with the condition of non – empty queue (Q) until the queue is empty.
11.	u = DEQUEUE	The DEQUEUE elements are stored in u.
12.	For each vertex v adjacent to u	Examines each vertex v adjacent to u and also visits v
13.	//Relax (u,v,w)	
14.	if v.distance > (u.distance + edge-weight(u,v))	Check the minimum distance of vertex
15.	then v.distance = u.distance + edge-weight(u,v)	Update the distance of vertex
16.	v.predecessor = u	Update the predecessor of the vertex

c) Step by step solution for the BFS, DFS and Dijkstra's of the given graph.

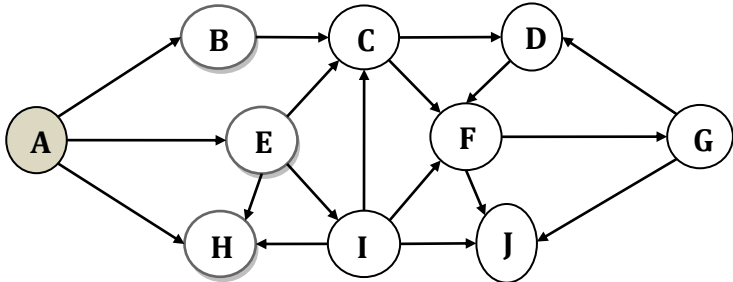
Step by step solution of BFS ALGORITHM (*unweighted*)

Here it has been mentioned in the scenario that we have to assume that the graph is unweighted. Initially the color of the entire vertex is white and its distance is initialized to ∞ . The parent or predecessor of all the vertices is initially NIL. The BFS uses First in First Out queue (FIFO) to store vertices which become gray after traversal.

GRAPH	DESCRIPTION
	<p>All the vertices will be in WHITE in color.</p> <p>Distance of all the vertices will be ∞</p> <p>Parent of all the vertices will be = NIL</p>

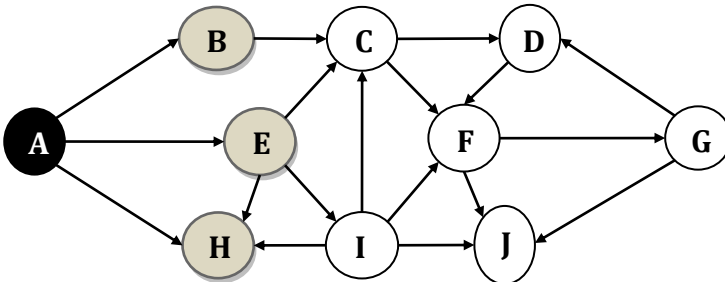
Step 1:

Let's assume the source vertex is node A. So first discover the source vertex A and add into queue Q. Also change the color of vertex A to gray and its distance and predecessor to 0. All other vertices will be white in color.

GRAPH	DESCRIPTION
	<p>Source node = A [Let] A.Color = GRAY</p> <p>Q A</p> <p>Distance = 0 Predecessor = NIL</p>

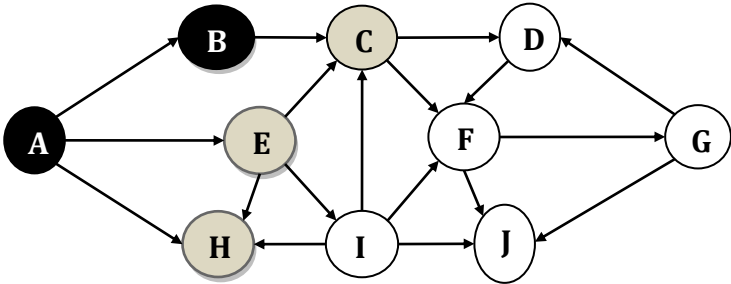
Step 2:

All the adjacent white vertices (B, E, H) of gray vertex A must be explored and added in the queue. Now change the color of adjacent vertices of parent A to gray. As all the white adjacent vertices of A are explored, there is no need to check vertex A further so change the color of parent vertex A to black and delete it from the queue Q.

GRAPH	DESCRIPTION			
	A.Color = BLACK B.Color = GRAY E.Color = GRAY H.Color = GRAY			
	Q	B	E	H
	Distance	1	1	1
	Predecessor	A	A	A

Step 3:

As in queue three vertices are present so we will first explore the white adjacent vertex of the vertex which enter first in the queue i.e. B whose white adjacent vertex is only C. We have to add that vertex in queue and update the queue. There is no further need to check vertex B so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION			
	B.Color = BLACK E.Color = GRAY H.Color = GRAY C.Color = GRAY			
	Q	E	H	C
	Distance	1	1	2
	Predecessor	A	A	B

Step 4:

As there are three vertices in the queue Q following FIFO, we will explore the adjacent vertex of vertex E as it enters into the queue first. E is having only one white adjacent vertex i.e. I. We have to add that vertex in queue and update the queue Q. There is no further need to check the vertex I, as all the adjacent vertex of E is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION			
	E.Color = BLACK H.Color = GRAY C.Color = GRAY I.Color = GRAY			
	Q	H	C	I
	Distance	1	2	2
	Predecessor	A	B	E

Step 5:

As there are three vertices in the FIFO queue Q, we will explore the white adjacent vertex of vertex H as it enters into the queue first. There are no white adjacent vertices of H. There is no further need to check the vertex H, as all the adjacent vertex of H is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION		
	H.Color = BLACK C.Color = GRAY I.Color = GRAY		
	Q	C	I
	Distance	2	2
	Predecessor	B	E

Step 6:

As there are two vertices in the FIFO queue Q, we will explore the white adjacent vertex of vertex C as it enters into the queue first. There are two white adjacent vertex of C i.e. D and F. We will add the vertex D into the queue Q and update the queue Q. There is no further need to check the vertex C, as all the adjacent vertex of C is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION			
	C.Color = BLACK I.Color = GRAY D.Color = GRAY F.Color = GRAY			
	Q	I	D	F
	Distance	2	3	3
	Predecessor	E	C	C

Step 7:

As there are three vertices in the FIFO queue Q, we will explore the adjacent vertex of vertex I as it enters into the queue first. The white adjacent vertex of I is only vertex J. We will add the vertex J into the queue Q and update the queue. There is no further need to check the vertex I, as all the adjacent vertex of I is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION			
	I.Color = BLACK D.Color = GRAY F.Color = GRAY J.Color = GRAY			
	Q	D	F	J
	Distance	3	3	3
	Predecessor	C	C	I

Step 8:

As there are three vertices in the FIFO queue Q, we will explore the adjacent vertex of vertex D as it enters into the queue first among the three vertices. There is no further need to check the vertex D, as all the adjacent vertex of D is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION		
	D.Color = BLACK F.Color = GRAY J.Color = GRAY		
	Q	F	J
	Distance	2	3
	Predecessor	D	I

Step 9:

We will explore the adjacent vertex F as it enters into the queue first. The white adjacent vertex of F is only vertex G. We will add the vertex G into the queue Q and update the queue. There is no further need to check the vertex F, as all the adjacent vertex of F is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION		
	F.Color = BLACK J.Color = GRAY G.Color = GRAY		
	Q	J	G
	Distance	3	4
	Predecessor	I	F

Step 10:

We will explore the vertex J as it enters into the queue first. There is no white adjacent vertex left to vertex J. There is no further need to check the vertex J, as all the adjacent vertex of J is explored so change the color of parent vertex to black and delete it from the queue Q.

GRAPH	DESCRIPTION	
	J.Color = BLACK G.Color = GRAY	
	Q	G
	Distance	4
	Predecessor	F

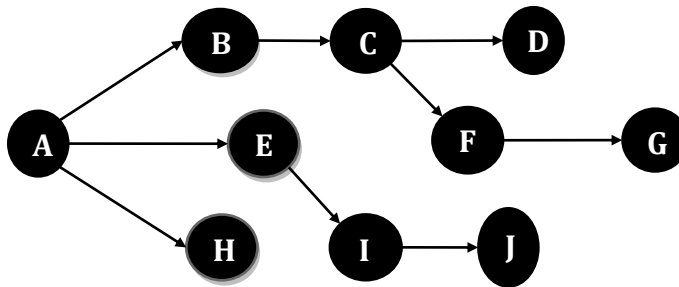
Step 11:

There is no white adjacent vertex left to vertex G. All the adjacent vertices of G are already deleted and since vertex G is the only vertex left in the queue, so there is no further need to check the vertex G, as all the adjacent vertex of G is explored so change the color of parent vertex to black and delete it from the queue Q.

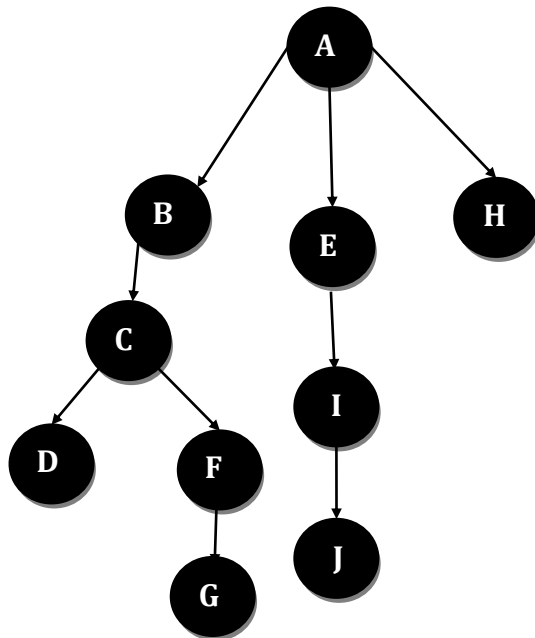
GRAPH	DESCRIPTION	
	G.Color = BLACK	
	Q	ϕ

Now the queue becomes empty so no more nodes are left to check, that means Traversal is complete.

After Breadth First traversal, the final graph is:



Tree after Breadth First Traversal



Path of traversal of graph according to Breadth First Traversal are: **A, B, E, H, C, I, D, F, J, G**

Step by step solution of DFS ALGORITHM (unweighted)

Here it has been mentioned in the scenario that we have to assume that the graph is unweighted. Initially the color of the entire vertex is white and its distance is initialized to ∞ . The parent or predecessor of all the vertices is initially NIL.

GRAPH	DESCRIPTION
<p>Initial state of the graph for DFS:</p> <ul style="list-style-type: none"> A: pred=NIL, color=WHITE B: pred=NIL, color=WHITE C: pred=NIL, color=WHITE D: pred=NIL, color=WHITE E: pred=NIL, color=WHITE F: pred=NIL, color=WHITE G: pred=NIL, color=WHITE H: pred=NIL, color=WHITE I: pred=NIL, color=WHITE J: pred=NIL, color=WHITE 	<p>All the vertices will be WHITE in color.</p> <p>The distance of all the vertices will be ∞.</p> <p>Predecessor of all the vertices will be NIL.</p>

Step 1:

GRAPH	DESCRIPTION
<p>State of the graph after Step 1:</p> <ul style="list-style-type: none"> A: pred=NIL, color=GRAY (marked with 1) B: pred=NIL, color=WHITE C: pred=NIL, color=WHITE D: pred=NIL, color=WHITE E: pred=NIL, color=WHITE F: pred=NIL, color=WHITE G: pred=NIL, color=WHITE H: pred=NIL, color=WHITE I: pred=NIL, color=WHITE J: pred=NIL, color=WHITE 	<p>Let's assume the source vertex is node A. So first discover the source vertex A and make its color to GRAY. As this is the first node to be traversed so mark it as 1 and predecessor to NIL.</p>

Step 2:

GRAPH	DESCRIPTION
<p>Graph diagram showing vertices A through J. Vertex A is the source (pred=NIL, color=GRAY) and is marked with '1'. Its adjacent vertices B, E, and H are marked with '2'. Vertex B has pred=A and color=GRAY. Vertex E has pred=NIL and color=WHITE. Vertex H has pred=NIL and color=WHITE. Vertex C has pred=NIL and color=WHITE. Vertex D has pred=NIL and color=WHITE. Vertex F has pred=NIL and color=WHITE. Vertex G has pred=NIL and color=WHITE. Vertex I has pred=NIL and color=WHITE. Vertex J has pred=NIL and color=WHITE.</p>	<p>There are three adjacent vertices of A i.e. B, E & H. Now explore any one of the adjacent vertex of A which I chose is B. Color the vertex B as GRAY and predecessor of B is A. As we visit the node on second number, so we mark it as 2.</p>

Step 3:

GRAPH	DESCRIPTION
<p>Graph diagram showing vertices A through J. Vertex A is the source (pred=NIL, color=GRAY) and is marked with '1'. Its adjacent vertices B, E, and H are marked with '2'. Vertex B has pred=A and color=GRAY. Vertex E has pred=NIL and color=WHITE. Vertex H has pred=NIL and color=WHITE. Vertex C has pred=B and color=GRAY and is marked with '3'. Vertex D has pred=NIL and color=WHITE. Vertex F has pred=NIL and color=WHITE. Vertex G has pred=NIL and color=WHITE. Vertex I has pred=NIL and color=WHITE. Vertex J has pred=NIL and color=WHITE.</p>	<p>We will explore the white adjacent vertex of B which is C. The color of the vertex C will be GRAY and predecessor of C is B. As we visit the node on third number, so we mark it as 3.</p>

Step 4:

GRAPH	DESCRIPTION
<p>Graph structure and node states for Step 4:</p> <ul style="list-style-type: none"> Node A: 1, A.pred=NIL, A.color=GRAY Node B: 2, B.pred=A, B.color=GRAY Node C: 3, C.pred=B, C.color=GRAY Node D: 4, D.pred=C, D.color=GRAY Node E: E.pred=NIL, E.color=WHITE Node F: F.pred=NIL, F.color=WHITE Node G: G.pred=NIL, G.color=WHITE Node H: H.pred=NIL, H.color=WHITE Node I: I.pred=NIL, I.color=WHITE Node J: J.pred=NIL, J.color=WHITE 	<p>After exploring C, we will explore one of the white adjacent nodes of C which is D. Change the color of vertex D to GRAY and predecessor of D is C. As we visit the node on forth number, so we mark it as 4.</p>

Step 5:

GRAPH	DESCRIPTION
<p>Graph structure and node states for Step 5:</p> <ul style="list-style-type: none"> Node A: 1, A.pred=NIL, A.color=GRAY Node B: 2, B.pred=A, B.color=GRAY Node C: 3, C.pred=B, C.color=GRAY Node D: 4, D.pred=C, D.color=GRAY Node E: E.pred=NIL, E.color=WHITE Node F: 5, F.pred=D, F.color=GRAY Node G: G.pred=NIL, G.color=WHITE Node H: H.pred=NIL, H.color=WHITE Node I: I.pred=NIL, I.color=WHITE Node J: J.pred=NIL, J.color=WHITE 	<p>After exploring D, we will explore one of the white adjacent nodes of D which is F. Change the color of vertex F to GRAY and predecessor of F is D. As we visit the node on fifth number, so we mark it as 5.</p>

Step 6:

GRAPH	DESCRIPTION
<p>Graph structure and labels for Step 6:</p> <ul style="list-style-type: none"> A (1): A.pred=NIL, A.color=GRAY B (2): B.pred=A, B.color=GRAY C (3): C.pred=B, C.color=GRAY D (4): D.pred=C, D.color=GRAY E: E.pred=NIL, E.color=WHITE F (5): F.pred=D, F.color=GRAY G: G.pred=NIL, G.color=WHITE H: H.pred=NIL, H.color=WHITE I: I.pred=NIL, I.color=WHITE J (6): J.pred=F, J.color=GRAY 	<p>After exploring F, we will explore one of the white adjacent nodes of F which is J. Change the color of vertex J to GRAY and predecessor of J is F. As we visit the node on sixth number, so we mark it as 6.</p>

Step 7:

GRAPH	DESCRIPTION
<p>Graph structure and labels for Step 7:</p> <ul style="list-style-type: none"> A (1): A.pred=NIL, A.color=GRAY B (2): B.pred=A, B.color=GRAY C (3): C.pred=B, C.color=GRAY D (4): D.pred=C, D.color=GRAY E: E.pred=NIL, E.color=WHITE F (5): F.pred=D, F.color=GRAY G: G.pred=NIL, G.color=WHITE H: H.pred=NIL, H.color=WHITE I: I.pred=NIL, I.color=WHITE J (6/7): J.pred=F, J.color=BLACK 	<p>After visiting vertex J, we found that there are no adjacent vertices which can be explored further. We will change the color of vertex J to BLACK and predecessor of J will remain same i.e. F. We will mark it as 6/7 and move further.</p>

Step 8:

GRAPH	DESCRIPTION
<p>Graph state after visiting vertex G:</p> <ul style="list-style-type: none"> A: pred=NIL, color=GRAY B: pred=A, color=GRAY C: pred=B, color=GRAY D: pred=C, color=GRAY E: pred=NIL, color=WHITE F: pred=D, color=GRAY G: pred=F, color=BLACK H: pred=NIL, color=WHITE I: pred=E, color=WHITE J: pred=I, color=BLACK 	<p>After visiting vertex G, we found that there are no white adjacent vertices which can be explored. So we will change the color of vertex G to BLACK and predecessor of G is J. As we are moving backward, so we will mark G as 8/9.</p>

Step 9:

GRAPH	DESCRIPTION
<p>Graph state after moving back to F:</p> <ul style="list-style-type: none"> A: pred=NIL, color=GRAY B: pred=A, color=GRAY C: pred=B, color=GRAY D: pred=C, color=GRAY E: pred=NIL, color=WHITE F: pred=D, color=BLACK G: pred=F, color=BLACK H: pred=NIL, color=WHITE I: pred=E, color=WHITE J: pred=I, color=BLACK 	<p>As moving backtrack, we reach to the node F and as no node is left which is white. It means exploration of F is complete so change the color of vertex F to BLACK and predecessor of F will be same i.e. D. Mark it 5/10 and move to the parent node D.</p>

Step 10:

GRAPH	DESCRIPTION
<p>Graph structure for Step 10:</p> <ul style="list-style-type: none"> Node A: 1, A.pred=NIL, A.color=GRAY Node B: 2, B.pred=A, B.color=GRAY Node C: 3, C.pred=B, C.color=GRAY Node D: 4/11, D.pred=C, D.color=BLACK Node E: E.pred=NIL, E.color=WHITE Node F: 5/10, F.pred=D, F.color=BLACK Node G: 8/9, G.pred=J, G.color=BLACK Node H: H.pred=NIL, H.color=WHITE Node I: I.pred=NIL, I.color=WHITE Node J: 6/7, J.pred=F, J.color=BLACK 	<p>As moving backtrack, we reach to the node D and as no node is left which is white. It means exploration of D is complete so change the color of vertex D to BLACK and predecessor of D will be same i.e. C. Mark it 4/11 and move to the parent node C.</p>

Step 11:

GRAPH	DESCRIPTION
<p>Graph structure for Step 11:</p> <ul style="list-style-type: none"> Node A: 1, A.pred=NIL, A.color=GRAY Node B: 2, B.pred=A, B.color=GRAY Node C: 3/12, C.pred=B, C.color=BLACK Node D: 4/11, D.pred=C, D.color=BLACK Node E: E.pred=NIL, E.color=WHITE Node F: 5/10, F.pred=D, F.color=BLACK Node G: 8/9, G.pred=J, G.color=BLACK Node H: H.pred=NIL, H.color=WHITE Node I: I.pred=NIL, I.color=WHITE Node J: 6/7, J.pred=F, J.color=BLACK 	<p>As moving backtrack, we reach to the node C and as no node is left which is white. It means exploration of C is complete so change the color of vertex C to BLACK and predecessor of C will be same i.e. B. Mark it 3/12 and move to the parent node B.</p>

Step 12:

GRAPH	DESCRIPTION
<p>Graph structure and node states for Step 12:</p> <ul style="list-style-type: none"> A: A.pred=NIL, A.color=GRAY, 1 B: B.pred=A, B.color=BLACK, 2/13 C: C.pred=B, C.color=BLACK, 3/12 D: D.pred=C, D.color=BLACK, 4/11 E: E.pred=NIL, E.color=WHITE F: F.pred=D, F.color=BLACK, 5/10 G: G.pred=J, G.color=BLACK, 8/9 H: H.pred=NIL, H.color=WHITE I: I.pred=NIL, I.color=WHITE J: J.pred=F, J.color=BLACK, 6/7 	<p>As moving backtrack, we reach to the node B and as no node is left which is white. It means exploration of B is complete so change the color of vertex B to BLACK and predecessor of B will be same i.e. A. Mark it 2/13 and move to the parent node A.</p>

Step 13:

GRAPH	DESCRIPTION
<p>Graph structure and node states for Step 13:</p> <ul style="list-style-type: none"> A: A.pred=NIL, A.color=GRAY, 1 B: B.pred=A, B.color=BLACK, 2/13 C: C.pred=B, C.color=BLACK, 3/12 D: D.pred=C, D.color=BLACK, 4/11 E: E.pred=A, E.color=GRAY, 14 F: F.pred=D, F.color=BLACK, 5/10 G: G.pred=J, G.color=BLACK, 8/9 H: H.pred=NIL, H.color=WHITE I: I.pred=NIL, I.color=WHITE J: J.pred=F, J.color=BLACK, 6/7 	<p>As moving backtrack, we reach to the node A but node A is having white adjacent vertex i.e. E & H, so I chose E and move further. We will change the color of vertex E to GRAY as it was not visited before and the predecessor of E will be A. Mark it as 14.</p>

Step 14:

GRAPH	DESCRIPTION
<p>Graph diagram for Step 14. Nodes A through J are shown with their predecessors, colors, and visit counts. Node A is the root with A.pred=NIL and A.color=GRAY. Node B is a child of A with B.pred=A, B.color=BLACK, and visit count 2/13. Node C is a child of B with C.pred=B, C.color=BLACK, and visit count 3/12. Node D is a child of C with D.pred=C, D.color=BLACK, and visit count 4/11. Node E is a child of A with E.pred=A, E.color=GRAY, and visit count 14. Node F is a child of C with F.pred=C, F.color=BLACK, and visit count 5/10. Node G is a child of F with G.pred=F, G.color=BLACK, and visit count 8/9. Node H is a child of A with H.pred=A, H.color=WHITE, and visit count 1. Node I is a child of E with I.pred=E, I.color=GRAY, and visit count 15. Node J is a child of F with J.pred=F, J.color=BLACK, and visit count 6/7.</p>	<p>As moving forward, we reach to the node I which was white in color and was also not visited before, so we will change the color of vertex I to GRAY and the predecessor of I will be E. Mark it as 15.</p>

Step 15:

GRAPH	DESCRIPTION
<p>Graph diagram for Step 15. Nodes A through J are shown with their predecessors, colors, and visit counts. Node A is the root with A.pred=NIL and A.color=GRAY. Node B is a child of A with B.pred=A, B.color=BLACK, and visit count 2/13. Node C is a child of B with C.pred=B, C.color=BLACK, and visit count 3/12. Node D is a child of C with D.pred=C, D.color=BLACK, and visit count 4/11. Node E is a child of A with E.pred=A, E.color=GRAY, and visit count 14. Node F is a child of C with F.pred=C, F.color=BLACK, and visit count 5/10. Node G is a child of F with G.pred=F, G.color=BLACK, and visit count 8/9. Node H is a child of A with H.pred=A, H.color=WHITE, and visit count 1. Node I is a child of E with I.pred=E, I.color=GRAY, and visit count 15. Node J is a child of F with J.pred=F, J.color=BLACK, and visit count 6/7.</p>	<p>As moving forward, we reach to the node H which was white in color and was also not visited before, so we will change the color of vertex H to GRAY and the predecessor of H will be I. Mark it as 16.</p>

Step 16:

GRAPH	DESCRIPTION
<p>Graph structure and node information:</p> <ul style="list-style-type: none"> Node A: A.pred=NIL, A.color=GRAY Node B: B.pred=A, B.color=BLACK, Value: 2/13 Node C: C.pred=B, C.color=BLACK, Value: 3/12 Node D: D.pred=C, D.color=BLACK, Value: 4/11 Node E: E.pred=A, E.color=GRAY, Value: 14 Node F: F.pred=D, F.color=BLACK, Value: 5/10 Node G: G.pred=J, G.color=BLACK, Value: 8/9 Node H: H.pred=I, H.color=BLACK, Value: 16/17 Node I: I.pred=E, I.color=GRAY, Value: 15 Node J: J.pred=F, J.color=BLACK, Value: 6/7 	<p>As moving further, we found that all nodes are visited. So we will move back to the parent node and node H will be marked as 16/17. The color of node H will be changed to BLACK and predecessor of H will be same i.e. I.</p>

Step 17:

GRAPH	DESCRIPTION
<p>The graph shows the following nodes and their attributes:</p> <ul style="list-style-type: none"> A: A.pred=NIL, A.color=GRAY, 1 B: B.pred=A, B.color=BLACK, 2/13 C: C.pred=B, C.color=BLACK, 3/12 D: D.pred=C, D.color=BLACK, 4/11 E: E.pred=A, E.color=GRAY, 14 F: F.pred=D, F.color=BLACK, 5/10 G: G.pred=J, G.color=BLACK, 8/9 H: H.pred=I, H.color=BLACK, 16/17 I: I.pred=E, I.color=BLACK, 15/18 J: J.pred=F, J.color=BLACK, 6/7 <p>The edges represent the parent-child relationships: A→B, A→E, A→H, B→C, C→D, C→F, D→G, E→I, F→J, G→I, H→I, I→H, I→J.</p>	<p>As moving backtrack, we reach to the node I and no node of I is left which is white. It means exploration of I is complete, so change the color of node I to BLACK and mark it as 15/18 and will move back to the parent node i.e. E.</p>

Step 18:

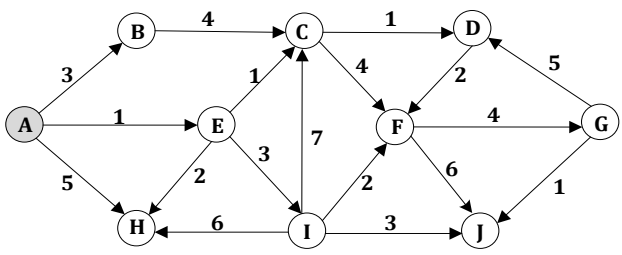
GRAPH	DESCRIPTION
<p> B.pred=A B.color=BLACK 2/13 C.pred=B C.color=BLACK 3/12 D.pred=C D.color=BLACK 4/11 E.pred=A E.color=BLACK 14/19 F.pred=D F.color=BLACK 5/10 G.pred=J G.color=BLACK 8/9 H.pred=I H.color=BLACK 16/17 I.pred=E I.color=BLACK 15/18 J.pred=F J.color=BLACK 6/7 A.pred=NIL A.color=GRAY 1 </p>	<p>As moving backtrack, we reach to the node E and no node of E is left which is white. It means exploration of E is complete, so change the color of node E to BLACK and mark it as 14/19 and will move back to the parent node i.e. A.</p>

Step 19:

GRAPH	DESCRIPTION
<p> B.pred=A B.color=BLACK 2/13 C.pred=B C.color=BLACK 3/12 D.pred=C D.color=BLACK 4/11 E.pred=A E.color=BLACK 14/19 F.pred=D F.color=BLACK 5/10 G.pred=J G.color=BLACK 8/9 H.pred=I H.color=BLACK 16/17 I.pred=E I.color=BLACK 15/18 J.pred=F J.color=BLACK 6/7 A.pred=NIL A.color=BLACK 1/20 </p>	<p>As moving backtrack, we reach to the last node A and no node of A is left which is white. It means exploration of A is complete, so change the color of node A to BLACK and mark it as 1/20. Predecessor of node A will remain as NIL.</p>

Step by step solution of Dijkstra's Algorithm

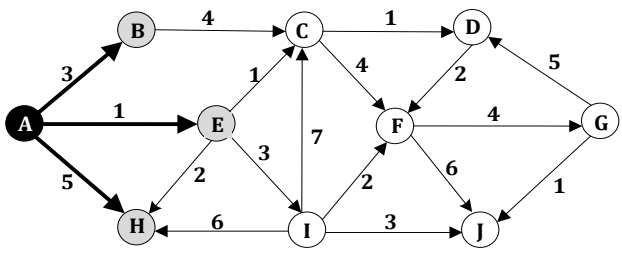
Here it has been mentioned in the scenario that we have to assume that the graph is weighted. Initially the color of the entire vertex is white and its distance is initialized to ∞ except the source vertex A whose distance will be 0. The parent or predecessor of all the vertices is initially NIL.

GRAPH	DESCRIPTION
	<p>Source vertex = A [Let]</p> <p>The distance of all the vertices will be ∞ except the source vertex A whose distance will be 0.</p> <p>Predecessor of all the vertices will be NIL.</p>

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
Predecessor	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Step 1:

As we have assumed the source node as vertex A, A is having the minimum distance as 0. Add it to S and relax all the nodes adjacent to source node A. The adjacent nodes of A are B, E & H. Update all the predecessor of all the adjacent nodes and we will dequeue the source node A.

GRAPH	DESCRIPTION
	<p>Adjacent nodes of A = B, E, H</p> <p>Predecessor of B, E and H = A and rest all the vertices will be NIL.</p> <p>Distance of all the adjacent nodes will be updated.</p>

For vertex B: if $B.distance > A.distance + wt.(A,B) \Rightarrow \infty > 0+3$ (True), then $B.distance = 3$,
 $B.predecessor = A$

For vertex E: if $E.distance > A.distance + wt.(A,E) \Rightarrow \infty > 0+1$ (True), then $E.distance = 1$,
 $E.predecessor = A$

For vertex H: if $H.distance > A.distance + wt.(A,H) \Rightarrow \infty > 0+5$ (True), then $H.distance = 5$,
 $H.predecessor = A$

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	∞	∞	1	∞	∞	5	∞	∞
Predecessor	NIL	A	NIL	NIL	A	NIL	NIL	A	NIL	NIL

Graph: $S = \{A\}$



Step 2:

We have to choose the minimum distance node which is at priority in the queue Q i.e. vertex E.
 We will add the vertex E into the set S and relax all the adjacent nodes of E which are C, H, I.
 The predecessor of all the adjacent nodes will get updated and node E will be dequeued.

GRAPH	DESCRIPTION
	<p>Adjacent nodes of E = C, H, I</p> <p>Predecessor of C, H and I = E and rest all the vertices will be NIL.</p> <p>Distance of all the adjacent nodes will be updated and E will be dequeued.</p>

For vertex C: if $C.distance > E.distance + wt.(E,C) \Rightarrow \infty > 1+1$ (True), then $C.distance = 2$,
 $C.predecessor = E$

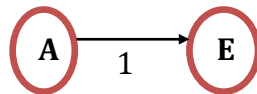
For vertex H: if $H.distance > E.distance + wt.(E,H) \Rightarrow 5 > 1+2$ (True), then $H.distance = 3$,
 $H.predecessor = E$

For vertex I: if $I.distance > E.distance + wt.(E,I) \Rightarrow \infty > 1+3$ (True), then $I.distance = 4$,
 $I.predecessor = E$

H distance will be replaced from 5 to 3 as recent distance of H was greater than parent distance + edge length from parent node.

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	∞	1	∞	∞	3	4	∞
Predecessor	NIL	A	E	NIL	A	NIL	NIL	E	E	NIL

Graph: $S = \{A, E\}$



Step 3:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex C and then add the vertex C into the set S by relaxing all the adjacent nodes of C which are D and F. Now the predecessor of all the adjacent nodes will get updated and node C will be dequeued.

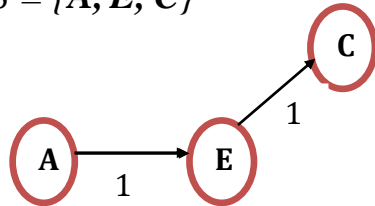
GRAPH	DESCRIPTION
	<p>Adjacent nodes of C = D, F</p> <p>Predecessor of D and F = C</p> <p>Distance of all the adjacent nodes will be updated.</p>

For vertex D: if $D.distance > C.distance + wt.(C,D) \Rightarrow \infty > 2+1$ (True), then $D.distance = 3$,
 $D.predecessor = C$

For vertex F: if $F.distance > C.distance + wt.(C,F) \Rightarrow \infty > 2+4$ (True), then $F.distance = 6$, $F.predecessor = C$

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	6	∞	3	4	∞
Predecessor	NIL	A	E	C	A	C	NIL	E	E	NIL

Graph: $S = \{A, E, C\}$



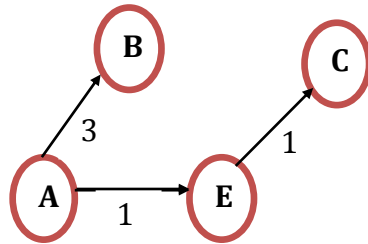
Step 4:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex B and then add the vertex B into the set S by relaxing all the adjacent nodes of B which is C. But C is already deleted from the queue so no need to relax it and node B will be dequeued.

GRAPH	DESCRIPTION
	<p>Adjacent nodes of B = C</p> <p>Predecessor of B will remain same i.e. A</p> <p>No node will be updated as B does not have any adjacent node left.</p>

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	6	∞	3	4	∞
Predecessor	NIL	A	E	C	A	C	NIL	E	E	NIL

Graph: $S = \{A, E, C, B\}$

Step 5:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex D and then add the vertex D into the set S by relaxing all the adjacent nodes of D which is F. Now the predecessor of all the adjacent nodes will get updated and node D will be dequeued.

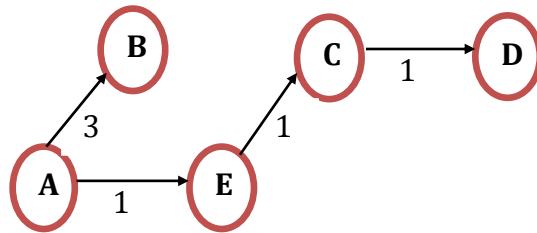
GRAPH	DESCRIPTION
	<p>Adjacent nodes of D = F</p> <p>Predecessor of F will be D</p> <p>Distance of all the adjacent nodes will be updated.</p>

For vertex F: if $F.distance > D.distance + wt.(D,F) \Rightarrow 6 > 3+2$ (True), then $F.distance = 5$, $D.predecessor = C$

F distance will be replaced from 6 to 5 as recent distance of F was greater than parent distance + edge length from parent node.

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	5	∞	3	4	∞
Predecessor	NIL	A	E	C	A	D	NIL	E	E	NIL

Graph: $S = \{A, E, C, B, D\}$



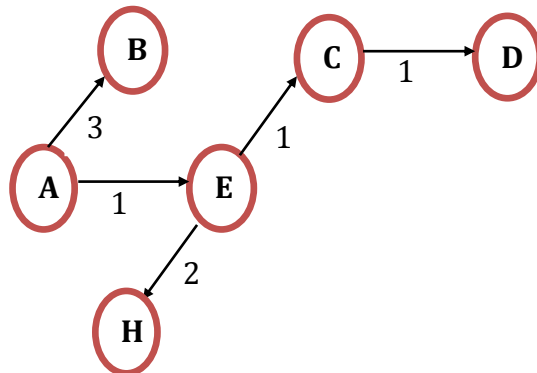
Step 6:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex H and then add the vertex H into the set S. Since there is no adjacent node of vertex H so delete the vertex H from the queue.

GRAPH	DESCRIPTION
	There is no adjacent node of H so no update in the queue.

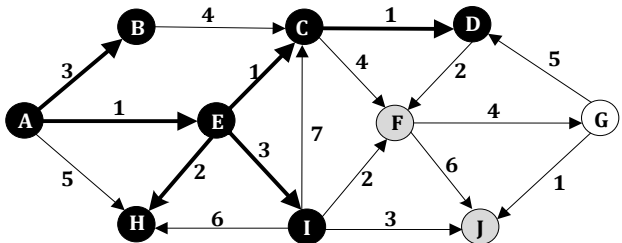
Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	5	∞	3	4	∞
Predecessor	NIL	A	E	C	A	D	NIL	E	E	NIL

Graph: $S = \{A, E, C, B, D, H\}$



Step 7:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex I and then add the vertex I into the set S by relaxing all the adjacent nodes of I which is C, H, F and J. But C and H is already deleted from the queue so no need to relax it. Now the predecessor of all the adjacent nodes F and J will get updated and node I will be dequeued.

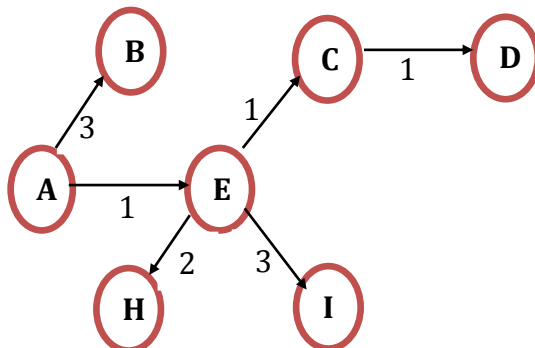
GRAPH	DESCRIPTION
	<p>Adjacent node of I = C, H, F and J</p> <p>Predecessor of F and J = I</p> <p>Node I will be dequeued.</p>

For vertex F: if $F.distance > I.distance + wt(I,F) \Rightarrow 5 > 4+2$ (False), then $F.distance = 5$, $F.predecessor = D$

For vertex J: if $J.distance > I.distance + wt(I,J) \Rightarrow \infty > 4+3$ (True), then $J.distance = 7$, $J.predecessor = D$

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	5	∞	3	4	7
Predecessor	NIL	A	E	C	A	D	NIL	E	E	I

Graph: $S = \{A, E, C, B, D, H, I\}$



Step 8:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex F and then add the vertex F into the set S by relaxing all the adjacent nodes of F which is G and J. Now the predecessor of all the adjacent nodes G and J will get updated and node F will be dequeued.

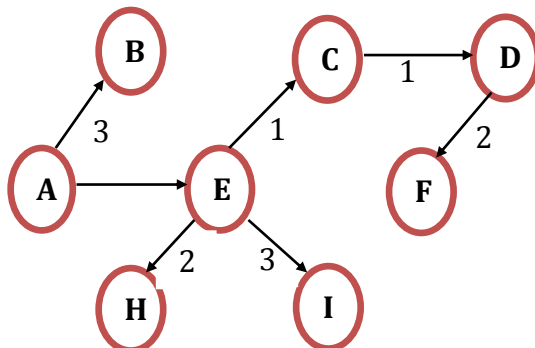
GRAPH	DESCRIPTION
	<p>Adjacent nodes of F = G and J</p> <p>Predecessor of G will be F and J will be I</p> <p>Distance of all the adjacent nodes will be updated.</p>

For vertex G: if $G.distance > F.distance + wt.(F,G) \Rightarrow \infty > 5+4$ (True), then $F.distance = 5$, $F.predecessor = D$

For vertex J: if $J.distance > F.distance + wt.(F,J) \Rightarrow 7 > 5+4$ (False), then $J.distance = 7$, $J.predecessor = D$

Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	5	9	3	4	7
Predecessor	NIL	A	E	C	A	D	F	E	E	I

Graph: $S = \{A, E, C, B, D, H, I, F\}$



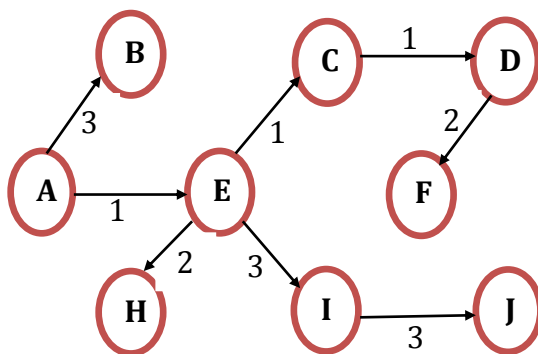
Step 9:

We have to delete the minimum distance node which is at priority in the queue Q i.e. vertex J and then add the vertex J into the set S. As there are no adjacent nodes of vertex J, so no need to check further and therefore node J will be dequeued.

GRAPH	DESCRIPTION
	<p>No adjacent nodes of J</p> <p>The distance will not be updated.</p>

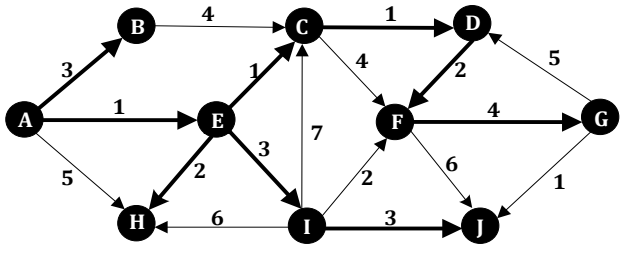
Queue	A	B	C	D	E	F	G	H	I	J
Distance	0	3	2	3	1	5	9	3	4	7
Predecessor	NIL	A	E	C	A	D	F	E	E	I

Graph: $S = \{A, E, C, B, D, H, I, F, J\}$



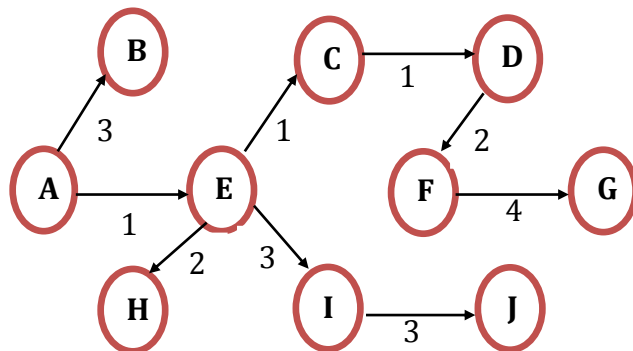
Step 10:

Now we are left with only one vertex i.e. G. So we have to delete the vertex G from the queue and then add the vertex G into the set S. As there are no adjacent nodes of vertex G, so no need to check further and therefore node G will be dequeued and the queue will be empty.

GRAPH	DESCRIPTION
	<p>No adjacent nodes of G</p> <p>The distance will not be updated.</p>

<i>Q</i>	A	B	C	D	E	F	G	H	I	J
<i>Distance</i>	0	3	2	3	1	5	9	3	4	7
<i>Predecessor</i>	NIL	A	E	C	A	D	F	E	E	I

Graph: $S = \{A, E, C, B, D, H, I, F, J, G\}$



As all the nodes are added in set S and the minimum priority Q is empty so it means stop there.

$$\text{Queue} = \{\emptyset\}$$

d) Efficiencies for the BFS, DFS and Dijkstra's Algorithm

Time Complexity of BFS Algorithm

The space complexity of BFS is expressed as $O|V|$. Every vertex is added only once in the queue thus occupies $O(1)$ space and vertex V occupies $O(V)$. As every vertex is initialized, added in the queue and dequeued from the queue at most once. The operation of enqueueing takes $O(1)$ time so the total time taken by V vertex is $O(V)$. Because adjacency vertex of each vertex is scanned only when vertex is de-queued, so each adjacent vertex is discovered at least once when directed and two times when undirected. The for-loop inside the while loop is executed at most $|E|$ times if G is a directed graph or $2|E|$ times if G is undirected.

Time Complexity of DFS Algorithm

In the case of DFS, each vertex is initialized once and initialization of source node takes $O(V)$ times and DFS-VISIT takes place only once for each vertex in which check white vertex and make it GRAY in color by traversing all its adjacent node which take at most $O(E)$ time and thus complexity is $O(V+E)$. The complexity of DFS is $O(V+E)$.

Time Complexity of DIJKSTRA Algorithm

EXTRACT-MIN is invoked once per vertex i.e., V times and each EXTRACT-MIN operation takes $O(V)$ time since we have to search through the entire array. Each edge in the adjacency list $Adj[v]$ is examined exactly once during the course of the algorithm. Since the total number of edges in all the adjacency lists is $|E|$, there a total of $|E|$ iterations of this for loop. These operations are known as DECREASE-KEY operations. Other operations inside the for loop take $O(1)$ time.

$$\begin{aligned}\text{Time} &= |V| O(V) + |E| O(1) \\ &= O(V^2 + E) \\ &= O(V^2)\end{aligned}$$

The above running time resulted from assuming that Q is implemented as an array. The time depends on different Q implementations. The efficiency of Dijkstra's algorithm can be increased when using binary heap or Fibonacci heap. Complexity: depends on the complexity of the min-heap implementation.

TASK 2: SCENARIO

Graph theory has many practical applications in the real world. One such application is to determine how a network of interconnected oil stations can be optimized in order to ensure that pipe lines used for connecting the stations can be reduced to minimum possible.

For this specific assignment, the second graph problem is represented hereby:

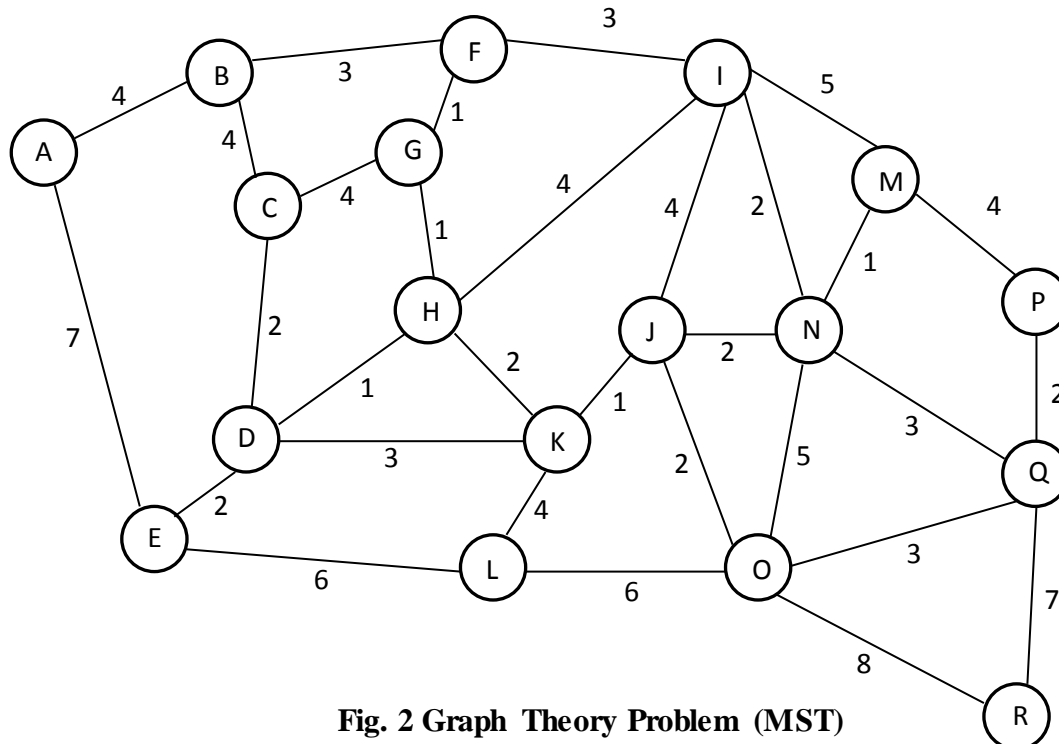


Fig. 2 Graph Theory Problem (MST)

The nodes in the above diagram represent the various oil stations scattered across a country, while the edges represent the oil pipelines running between them. The weight of each edge represents the distance between two stations along the edge.

You have to **deliver** a solution for the problem mentioned above by completing the following tasks:

- Draw adjacency list and adjacency matrix of the graph given.
- Write the pseudo-codes for the Prim's and Kruskal's algorithm along with their explanation.
- Provide step by step solution for the Prim's and Kruskal's algorithm of the given graph.
- Compute the efficiencies of the above the algorithms in terms of big-Oh.

TASK 2: SOLUTION

a) Adjacency Matrix & Adjacency List

ADJACENCY MATRIX

ADJACENCY MATRIX																		
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
A	0	4	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0
B	4	0	4	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
C	0	4	0	2	0	0	4	0	0	0	0	0	0	0	0	0	0	0
D	0	0	2	0	2	0	0	1	0	0	3	0	0	0	0	0	0	0
E	7	0	0	2	0	0	0	0	0	0	0	6	0	0	0	0	0	0
F	0	3	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	0
G	0	0	4	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	1	0	4	0	2	0	0	0	0	0	0	0
I	0	0	0	0	0	3	0	4	0	4	0	0	5	2	0	0	0	0
J	0	0	0	0	0	0	0	0	4	0	1	0	0	2	2	0	0	0
K	0	0	0	3	0	0	0	2	0	1	0	4	0	0	0	0	0	0
L	0	0	0	0	6	0	0	0	0	0	4	0	0	0	6	0	0	0
M	0	0	0	0	0	0	0	0	5	0	0	0	0	1	0	4	0	0
N	0	0	0	0	0	0	0	0	2	2	0	0	1	0	5	0	3	0
O	0	0	0	0	0	0	0	0	0	2	0	6	0	5	0	0	3	8
P	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	2	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	2	0	7
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	7	0

ADJACENCY LIST

ADJACENCY LIST

A 0 → B 4 → E 7 ⊗

B 0 → A 4 → C 4 → F 3 ⊗

C 0 → B 4 → D 2 → G 4 ⊗

D 0 → C 2 → E 2 → H 1 → K 3 ⊗

E 0 → A 7 → D 2 → L 6 ⊗

F 0 → B 3 → G 1 → I 3 ⊗

G 0 → C 4 → F 1 → H 1 ⊗

H 0 → D 1 → G 1 → I 4 → K 2 ⊗

I 0 → F 3 → H 4 → J 4 → M 5 → N 2 ⊗

J 0 → I 4 → K 1 → N 2 → O 2 ⊗

K 0 → D 3 → H 2 → J 1 → L 4 ⊗

L 0 → E 6 → K 4 → O 6 ⊗

M 0 → I 5 → N 1 → P 4 → K 2 ⊗

N 0 → I 2 → J 2 → M 1 → O 5 → Q 3 ⊗

O 0 → J 2 → L 6 → N 5 → Q 3 → R 8 ⊗

P 0 → M 4 → Q 2 ⊗

Q 0 → N 3 → O 3 → P 2 → R 7 ⊗

R 0 → O 8 → Q 7 ⊗

b) Pseudo-codes for the Prim's and Kruskal's Algorithm

Pseudo code for Prim's ALGORITHM

S. No.	PRIM's Algorithm	Explanation
	MST- PRIM(G,s) /* G = (V,E) */	V denotes the set of each vertex and E is the set of each edge of the graph. One is Graph G and the other is source vertex s.
1.	For each vertex u in V	For each vertex in set of vertex V
2.	u.distance = ∞	Initialize the distance of every vertex to ∞ .
3.	u.predecessor = NIL	Initialize the predecessor of all other vertex in the graph to be equal to NIL.
4.	s.distance = 0	Initializing the distance of the source vertex to be equal to 0.
5.	$Q \leftarrow V\{G\}$	Create a minimum-priority queue Q for V according to values of distance.
6.	While Q is not empty ($\neq \emptyset$)	Assume queue Q is not empty it contains all the vertices of graph. Iteration of While loop is done.
7.	u = EXTRACT-MIN (Q)	Extract minimum vertex from queue Q.
8.	For each v adjacent to u	The for loop checks and updates the distance and predecessor of u of every vertex v adjacent to u.
9.	if v exists in Q and weight of (u, v) < v.distance	Checks the distance of the vertex v and then update it.
10.	v.predecessor = u	If the condition in step 9 results to be true, the predecessor of the vertex v is changed and u is updated as new predecessor.
11.	v.distance = weight of (u,v)	The distance of vertex v is updated and the weight of edge u-v is made the new distance of v.

Pseudo code for Kruskal's ALGORITHM

S. No.	KRUSKAL's Algorithm	Explanation
	MST- KRUSKAL(G,s) /* G = (V,E) */	V denotes the set of each vertex and E is the set of each edge of the graph. One is Graph G and the other is source vertex s.
1.	A is empty set(= \emptyset)	Initialize the set A to the empty set
2.	For each vertex v in V	Randomly vertex v is chosen out of the set of all the vertices V of the graph G.
3.	MAKE-SET (v)	MAKE-SET (v) creates a new set whose only member is pointed to by v and for this v must be in a set.
4.	Sort the edges of E into non-decreasing order by the weight	The edges are being sorted into increasing order of their weights.
5.	For each edge (u, v) in E, taken in non-decreasing order by weight	It takes all the edges in increasing order. Edge (u, v) forms the edge of the minimum weight.
6.	If FIND-SET (u) \neq FIND-SET (v)	Checks for each edge (u, v) whether the endpoint u and v belongs to the same set or tree. If they belong to the same set it means they can create cycle and cannot be added to the tree, and the edge is discarded.
7.	A = A U {(u, v)}	If the arbitrary vertices belong to the same tree, they get discarded. If not, then they get added to the set A.
8.	UNION (u,v)	The vertices of two trees are merged in the same set by UNION function.
9.	return A	When all the vertices come into same set A then algorithm terminate.

c) Step by step solution for the Prim's and Kruskal's of the given graph.

Step by step solution of PRIM's ALGORITHM

Step 1:

GRAPH	DESCRIPTION
	<p>Initialize all the nodes with predecessor as NIL and distance as infinity (∞) except the source node whose distance is zero (0).</p> <p>Consider A as the source node and start moving one by one to rest of the nodes. The source node is made Gray in color to indicate that we are considering the adjacent vertex of A.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Step 2:

GRAPH	DESCRIPTION
	<p>A has been deleted from the priority queue and the adjacent vertices are updated. A will be the predecessor of each adjacent vertex.</p> <p>Distance of (A, B) = 4 and distance of (A, E) = 7, so now distance is minimum as (A, B) = 4 so it extracted from the minimum priority queue and after that we will move to the next node B and check its neighbours. So the vertex B is made Gray in color and color of A will be Black.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	∞	∞	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	N	N	A	N	N	N	N	N	N	N	N	N	N	N	N	N

Step 3:

GRAPH	DESCRIPTION
	<p>B has been deleted from the priority queue and the vertices adjacent to B have been updated with the distance given in the graph.</p> <p>F is identified as the next vertex having minimum priority. Hence F is changed to Gray color and B as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	4	∞	7	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	B	N	A	B	N	N	N	N	N	N	N	N	N	N	N	N

Step 4:

GRAPH	DESCRIPTION
	<p>F has been deleted from the priority queue and the vertices adjacent to F have been updated with the distance given in the graph.</p> <p>G is identified as the next vertex having minimum priority. Hence G is changed to Gray color and F as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	4	∞	7	3	1	∞	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	B	N	A	B	F	N	F	N	N	N	N	N	N	N	N	N

GRAPH	DESCRIPTION
	<p>G has been deleted from the priority queue and the vertices adjacent to G have been updated with the distance given in the graph.</p> <p>H is identified as the next vertex having minimum priority. Hence H is changed to Gray color and G as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	4	∞	7	3	1	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	B	N	A	B	F	G	F	N	N	N	N	N	N	N	N	N

Step 6:

GRAPH	DESCRIPTION
	<p>H has been deleted from the priority queue and the vertices adjacent to H have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>D is identified as the next vertex having minimum priority. Hence D is changed to Gray color and H as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	4	1	7	3	1	1	4	∞	2	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	B	H	A	B	F	G	H	N	H	N	N	N	N	N	N	N

Step 7:

GRAPH	DESCRIPTION
	<p>D has been deleted from the priority queue and the vertices adjacent to D have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>C is identified as the next vertex having minimum priority. Hence C is changed to Gray color and D as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	4	∞	2	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	D	H	D	B	F	G	H	N	H	N	N	N	N	N	N	N

Step 8:

GRAPH	DESCRIPTION
	<p>C has been deleted from the priority queue but C does not have any adjacent vertex so we will check other adjacent node that comes first in queue with minimum priority.</p> <p>E is identified as the next vertex having minimum priority. Hence E is changed to Gray color and C as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	4	∞	2	∞	∞	∞	∞	∞	∞	∞
Predecessor	N	A	D	H	D	B	F	G	H	N	H	N	N	N	N	N	N	N

Step 9:

GRAPH	DESCRIPTION
	<p>E has been deleted from the priority queue and the vertices adjacent to E have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>K is identified as the next vertex having minimum priority. Hence K is changed to Gray color and E as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	4	∞	2	6	∞	∞	∞	∞	∞	∞
Predecessor	N	A	D	H	D	B	F	G	H	N	H	E	N	N	N	N	N	N

Step 10:

GRAPH	DESCRIPTION
	<p>K has been deleted from the priority queue and the vertices adjacent to K have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>J is identified as the next vertex having minimum priority. Hence J is changed to Gray color and K as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	4	1	3	4	∞	∞	∞	∞	∞	∞
Predecessor	N	A	D	H	D	B	F	G	H	K	D	K	N	N	N	N	N	N

Step 11:

GRAPH	DESCRIPTION
	<p>J has been deleted from the priority queue and the vertices adjacent to J have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>O is identified as the next vertex having minimum priority. Hence O is changed to Gray color and N as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	4	1	3	4	∞	2	2	∞	∞	∞
Predecessor	N	A	D	H	D	B	F	G	H	K	D	K	N	J	J	N	N	N

Step 12:

GRAPH	DESCRIPTION
	<p>O has been deleted from the priority queue and the vertices adjacent to O have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>N is identified as the next vertex having minimum priority. Hence N is changed to Gray color and O as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	4	1	3	4	∞	2	2	∞	3	8
Predecessor	N	A	D	H	D	B	F	G	H	K	D	K	N	J	J	N	O	O

Step 13:

GRAPH	DESCRIPTION
	<p>N has been deleted from the priority queue and the vertices adjacent to N have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>M is identified as the next vertex having minimum priority. Hence M is changed to Gray color and N as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	2	1	3	4	1	2	2	∞	3	8
Predecessor	N	A	D	H	D	B	F	G	N	K	D	K	N	J	J	N	O	O

Step 14:

GRAPH	DESCRIPTION
	<p>M has been deleted from the priority queue and the vertices adjacent to M have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>I is identified as the next vertex having minimum priority. Hence I is changed to Gray color and M as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	2	1	3	4	1	2	2	4	3	8
Predecessor	N	A	D	H	D	B	F	G	N	K	D	K	N	J	J	M	O	O

[illegible]

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	2	1	3	4	1	2	2	4	3	8
Predecessor	N	A	D	H	D	B	F	G	N	K	D	K	N	J	J	M	O	O

Step 16:

GRAPH	DESCRIPTION
	<p>Q has been deleted from the priority queue and the vertices adjacent to Q have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>P is identified as the next vertex having minimum priority. Hence P is changed to Gray color and Q as Black in color.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	2	1	3	4	1	2	2	2	3	7
Predecessor	N	A	D	H	D	B	F	G	N	K	D	K	N	J	J	Q	O	Q

GRAPH	DESCRIPTION
	<p>P has been deleted from the priority queue and the vertices adjacent to P have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>L is identified as the next vertex having minimum priority. Hence L is changed to Gray color and P as Black in color.</p>

Page 58 of 85

Step 18:

GRAPH	DESCRIPTION
	<p>L has been deleted from the priority queue and the vertices adjacent to L have been updated with the distance given in the graph. The predecessor of the respective vertex will also get updated likewise.</p> <p>R is identified as the next vertex having minimum priority. Hence R is changed to Gray color and L as Black in color.</p>

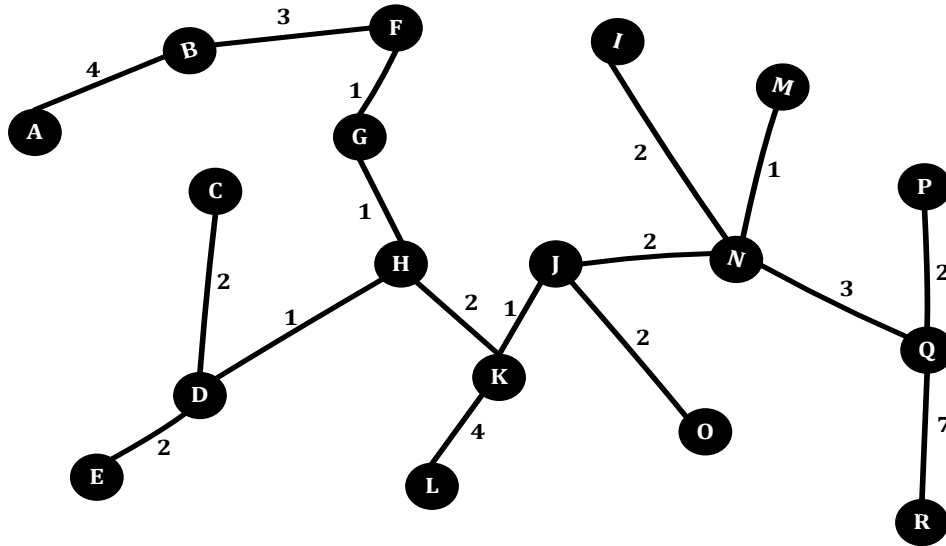
Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	2	1	3	4	1	2	2	2	3	7
Predecessor	N	A	D	H	D	B	F	G	N	K	D	K	N	J	J	Q	O	Q

Step 19:

GRAPH	DESCRIPTION
	<p>R has been deleted from the priority queue and it is colored as Black.</p> <p>As we know all the edges are deleted from the queue and the queue becomes empty. So no further traversal and finding of minimum path is complete.</p>

Queue	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Distance	0	4	2	1	2	3	1	1	2	1	3	4	1	2	2	2	3	7
Predecessor	N	A	D	H	D	B	F	G	N	K	D	K	N	J	J	Q	O	Q

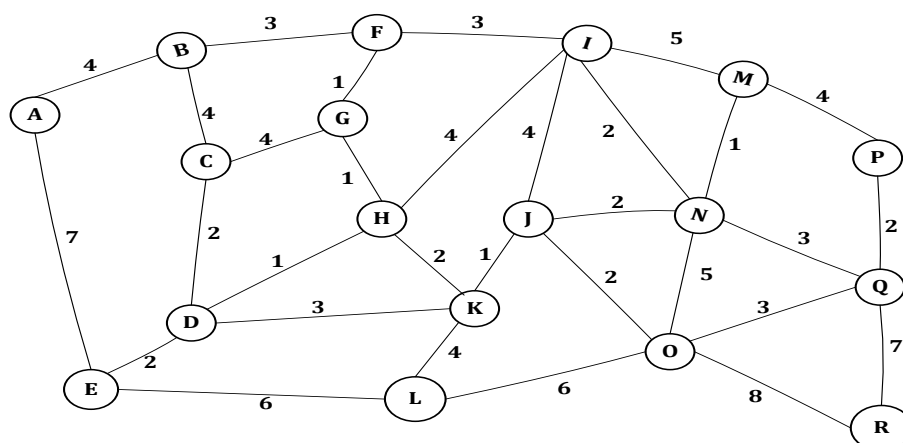
The complete minimum spanning tree is



Weight of minimum-spanning tree (MST) =

$$\begin{aligned}
 &AB + CD + DH + ED + BF + FG + GH + HK + KJ + KL + JO + JN + NI + NM + NQ + QP + QR \\
 &= 4 + 2 + 1 + 2 + 3 + 1 + 1 + 2 + 1 + 4 + 2 + 2 + 2 + 1 + 3 + 2 + 7 \\
 &= 40
 \end{aligned}$$

Step by step solution of KRUSKAL's ALGORITHM



All the sets are =

{A},{B},{C},{D},{E},{F},{G},{H},{I},{J},{K},{L},{M},{N},{O},{P},{Q},{R}

Distance of the edge	Edges Considered in Ascending Order
1	DH, FG, GH, JK, MN
2	CD, DE, HK, IN, JN, JO, PQ
3	BF, DK, FI, NQ, OQ
4	AB, BC, CG, HI, IJ, KL, MP
5	IM, NO
6	EL, LO
7	AE, QR
8	OR

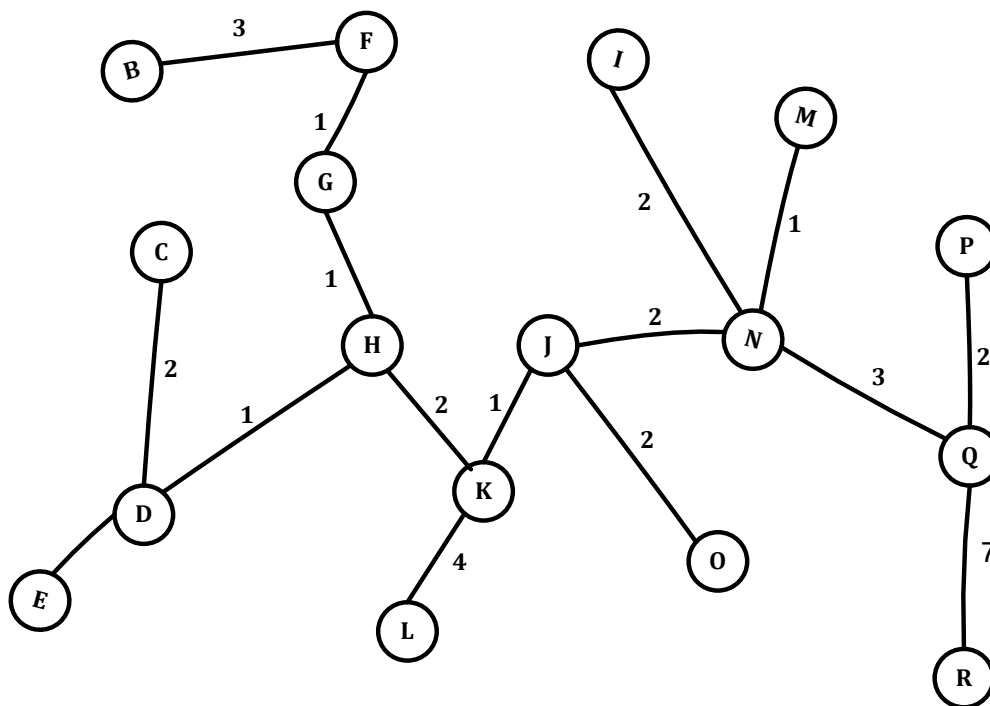
<u>Edge Considered</u>	<u>Sets</u>	<u>Status</u>	<u>MST Edges</u>
<u>Step 1:</u> {D,H} Separate Set	{A}, {B}, {C}, {D,H}, {E}, {F}, {G}, {I}, {J}, {K}, {L}, {M}, {N}, {O}, {P}, {Q}, {R}	Selected	{D,H}
<u>Step 2:</u> {F,G} Separate Set	{A}, {B}, {C}, {D,H}, {E}, {F,G}, {I}, {J}, {K}, {L}, {M}, {N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}
<u>Step 3:</u> {G,H} Separate Set	{A}, {B}, {C}, {D,F,G,H}, {E}, {I}, {J}, {K}, {L}, {M}, {N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}
<u>Step 4:</u> {J,K} Separate Set	{A}, {B}, {C}, {D,F,G,H}, {E}, {I}, {J,K}, {L}, {M}, {N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}
<u>Step 5:</u> {M,N} Separate Set	{A}, {B}, {C}, {D,F,G,H}, {E}, {I}, {J,K}, {L}, {M,N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}
<u>Step 6:</u> {C,D} Separate Set	{A}, {B}, {C,D,F,G,H}, {E}, {I}, {J,K}, {L}, {M,N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}
<u>Step 7:</u> {D,E} Separate Set	{A}, {B}, {C,D,E,F,G,H}, {I}, {J,K}, {L}, {M,N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}
<u>Step 8:</u> {H,K} Separate Set	{A}, {B}, {C,D,E,F,G,H,J,K}, {I}, {L}, {M,N}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}
<u>Step 9:</u> {I,N} Same Set	{A}, {B}, {C,D,E,F,G,H,J,K}, {I,M,N}, {L}, {O}, {P}, {Q}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}
<u>Step 10:</u> {J,N} Separate Set	{A}, {B}, {C,D,E,F,G,H,I,J,K,M,N}, {L}, {O}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}
<u>Step 11:</u> {J,O} Separate Set	{A}, {B}, {C,D,E,F,G,H,I,J,K,M,N,O}, {L}, {P}, {Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}
<u>Step 12:</u> {P,Q} Separate Set	{A}, {B}, {C,D,E,F,G,H,I,J,K,M,N,O}, {L}, {P,Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}
<u>Step 13:</u> {B,F} Separate Set	{A}, {B,C,D,E,F,G,H,I,J,K,M,N,O}, {L}, {P,Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}
<u>Step 14:</u> {D,K} Same Set	{A}, {B,C,D,E,F,G,H,I,J,K,M,N,O}, {L}, {P,Q}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {D,K}

<u>Step 15:</u> {F,I} Same Set	{A}, {B,C,D,E,F,G,H,I,J,K,M,N,O}, {L}, {P,Q}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {D,K}, {F,I}
<u>Step 16:</u> {N,Q} Separate Set	{A}, {B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}
<u>Step 17:</u> {O,Q} Same Set	{A}, {B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {O,Q}
<u>Step 18:</u> {A,B} Separate Set	{A,B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}
<u>Step 19:</u> {B,C} Same Set	{A,B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {B,C}
<u>Step 20:</u> {C,G} Same Set	{A,B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {B,C}, {C,G}
<u>Step 21:</u> {H,I} Same Set	{A,B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {B,C}, {C,G}, {H,I}
<u>Step 22:</u> {I,J} Same Set	{A,B,C,D,E,F,G,H,I,J,K,M,N,O,P,Q}, {L}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {B,C}, {C,G}, {H,I}, {I,J}
<u>Step 23:</u> {K,L} Separate Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {K,L}
<u>Step 24:</u> {M,P} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {K,L}, {M,P}

<u>Step 25:</u> {I,M} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Rejected	{D,H},{F,G},{G,H},{J,K},{M,N}, {C,D},{D,E},{H,K},{I,N},{J,N}, {J,O},{P,Q},{B,F},{N,Q},{A,B}, {K,L}, {M,P},{I,M}
<u>Step 26:</u> {N,O} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Rejected	{D,H},{F,G},{G,H},{J,K},{M,N}, {C,D},{D,E},{H,K},{I,N},{J,N}, {J,O},{P,Q},{B,F},{N,Q},{A,B}, {K,L}, {M,P},{I,M},{N,O}
<u>Step 27:</u> {E,L} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Rejected	{D,H},{F,G},{G,H},{J,K},{M,N}, {C,D},{D,E},{H,K},{I,N},{J,N}, {J,O},{P,Q},{B,F},{N,Q},{A,B}, {K,L},{M,P},{I,M},{N,O},{E,L}
<u>Step 28:</u> {L,O} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Rejected	{D,H},{F,G},{G,H},{J,K},{M,N}, {C,D},{D,E},{H,K},{I,N},{J,N}, {J,O},{P,Q},{B,F},{N,Q},{A,B}, {K,L},{M,P},{I,M},{N,O},{E,L}, {L,O}
<u>Step 29:</u> {A,E} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, {R}	Rejected	{D,H},{F,G},{G,H},{J,K},{M,N}, {C,D},{D,E},{H,K},{I,N},{J,N}, {J,O},{P,Q},{B,F},{N,Q},{A,B}, {K,L},{M,P},{I,M},{N,O},{E,L}, {L,O},{A,E}
<u>Step 30:</u> {Q,R} Separate Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, R	Selected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {K,L}, {Q,R}
<u>Step 31:</u> {O,R} Same Set	{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q}, R	Rejected	{D,H}, {F,G}, {G,H}, {J,K}, {M,N}, {C,D}, {D,E}, {H,K}, {I,N}, {J,N}, {J,O}, {P,Q}, {B,F}, {N,Q}, {A,B}, {K,L},{Q,R},{O,R}

Minimum Spanning tree =

$$\begin{aligned}
 &\mathbf{DH+ HG+ GF+ KJ+ NM+ ED+ DC+ OJ + NI+ JN+ HK+ QP+ BF+ NQ+ AB+ LK+ QR} \\
 &= 1+ 1+ 1+ 1+ 1+ 2+ 2+ 2+ 2+ 2+ 2+ 2+ 3+ 3+ 4+ 4+ 7 \\
 &= 40
 \end{aligned}$$



d) Efficiency of the Prim's and Kruskal's algorithms in terms of big-oh.

Note: The time complexity of Prim's & Kruskal's algorithm is explained in the pseudo code in the above Task 2(b).

Time Complexity of Kruskal's Algorithm: $O(n \log n)$ or $O(n \log V)$

Justification:

- Sorting of edges: $O(n \log n)$ time
- After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take at most: $O(\log V)$ time.
- So overall complexity = $O(n \log n + n \log V)$ time.
- Since, the value of E can be = V^2 . So $O(\log V)$ are $O(\log n)$ same. Therefore, overall time complexity for the Kruskal's Algorithm is = $O(n \log n)$ or $O(n \log V)$.

TIME COMPLEXITY OF PRIM'S ALGORITHM:

- Step 1 to 3 will be having the equal running time – $O(V)$. Run time will be maximum depending on the total number of vertices.
- Run time for step 4 and 5 = $O(1)$. These statements will be executed only once.
- Run time for step 6 and 7 = $O(V \log V)$. The loop in step 6 is iterating V times, hence the combined runtime = $O(V \log V)$.
- The number of edges will affect the runtime of step 9. We can compare maximum up to E times. Thus, the complexity of this step will be $O(E)$. The further steps have dependency over the edges E , and since they are still present in the loop, their complexity will be $O(E \log V)$.

Total run time = $O(V) + O(1) + O(V \log V) + O(E) + O(E \log V)$

$O(E \log V)$ is the maximum complexity involved in this algorithm and hence, this will affect the runtime to the greater extent as compared to the complexities of the other steps. So, total running time complexity of Prim's Algorithm = $O(E \log V)$.

TIME COMPLEXITY OF KRUSKAL'S ALGORITHM:

- The first step will have a runtime of $O(1)$ and this will remain constant as it does not depend on any other factor.
- Step 2 and 3 have their runtime $= O(V)$. This loop depends on V .
- Step 3 and 4 are storing the edges and they will have equal runtime $= O(E \log E)$
- Step 6, step 7 and step 8 depend on the number of vertices V and are running inside the loop. Thus, their run time $= O(E \log V)$
- Step 9 will have complexity $= O(1)$ as it is running only once.

Sum of total running times $= O(1) + O(V) + O(E \log E) + O(E \log V) + O(1)$. $O(E \log V)$ is the maximum running time that affects the running time of the Kruskal's Algorithm.

Thus, the total running time for Kruskal's algorithm $= O(E \log V)$.

TASK 3: SCENARIO

Given the following elements: 12, 9, 3, 4, 8, 2, 11, 5 apply the Heap Sort algorithm to sort them. Show all phases during the sort.

You have to deliver a solution for the problem mentioned above by completing the following tasks

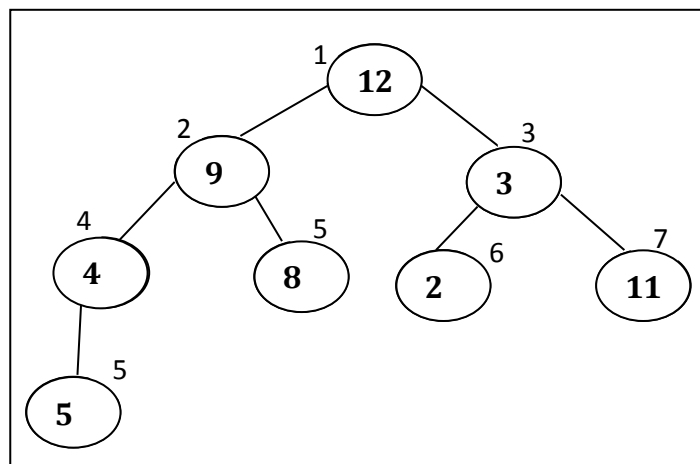
- Provide all phases of insert operations when building a heap out of these elements.
- Provide all phases of deleteMin operations when removing the minimal elements and storing them into the sorted array
- Discuss the complexity of the Heapsort in terms of the Big-Oh notation.

TASK 3: SOLUTION

a) All phases of insert operations

Heap Sort = 12, 9, 3, 4, 8, 2, 11, 5

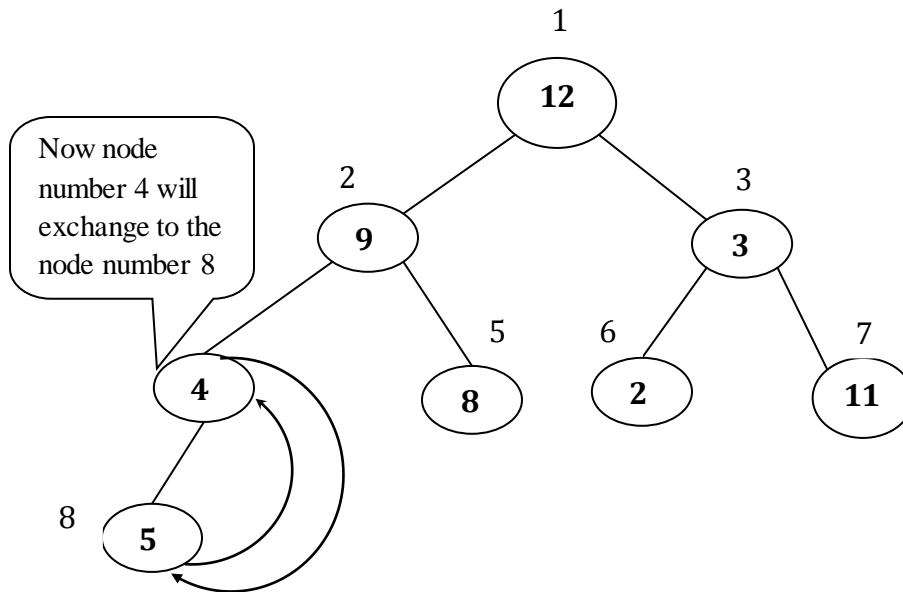
INDEX NO.	1	2	3	4	5	6	7	8
ELEMENTS	12	9	3	4	8	2	11	5



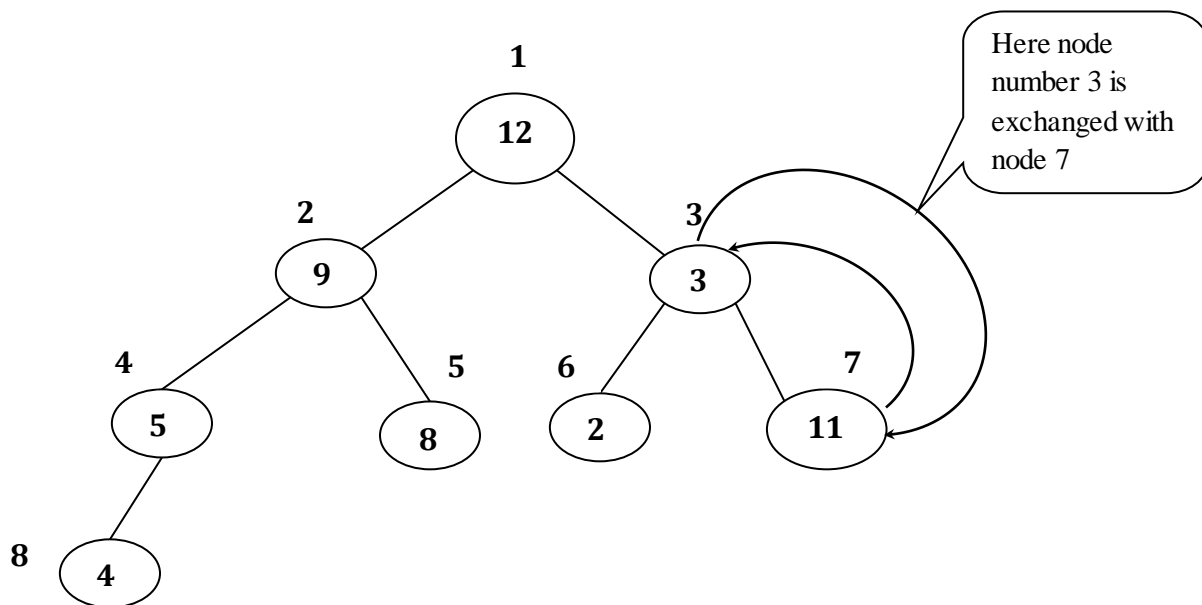
Total number of node = $n/2 + 1 = 8/2 + 1 = 4 + 1 = 5$

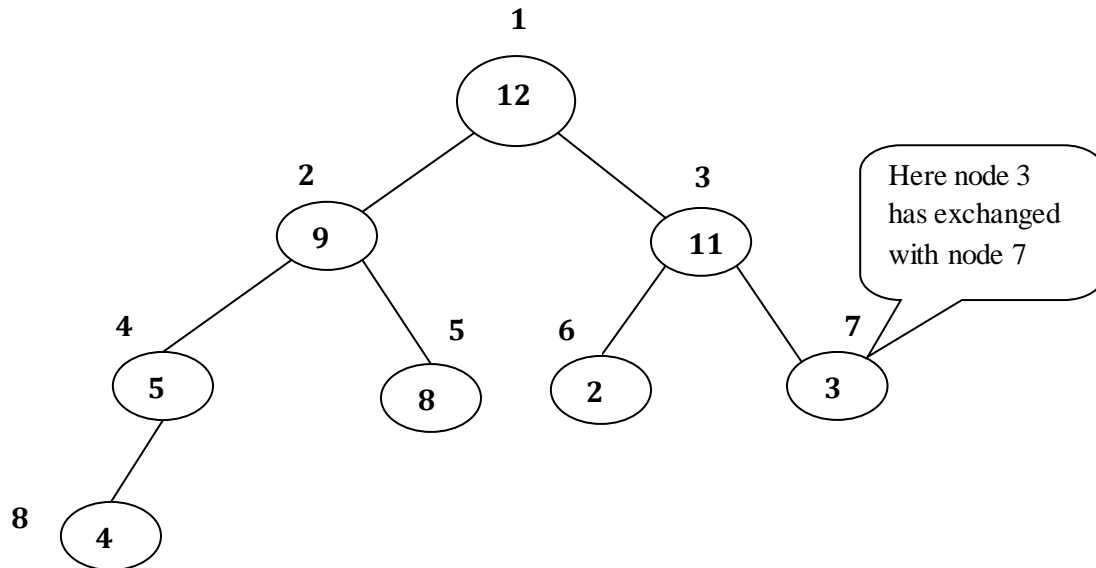
Now apply Max-Heap,

Step 1:



Step 2:



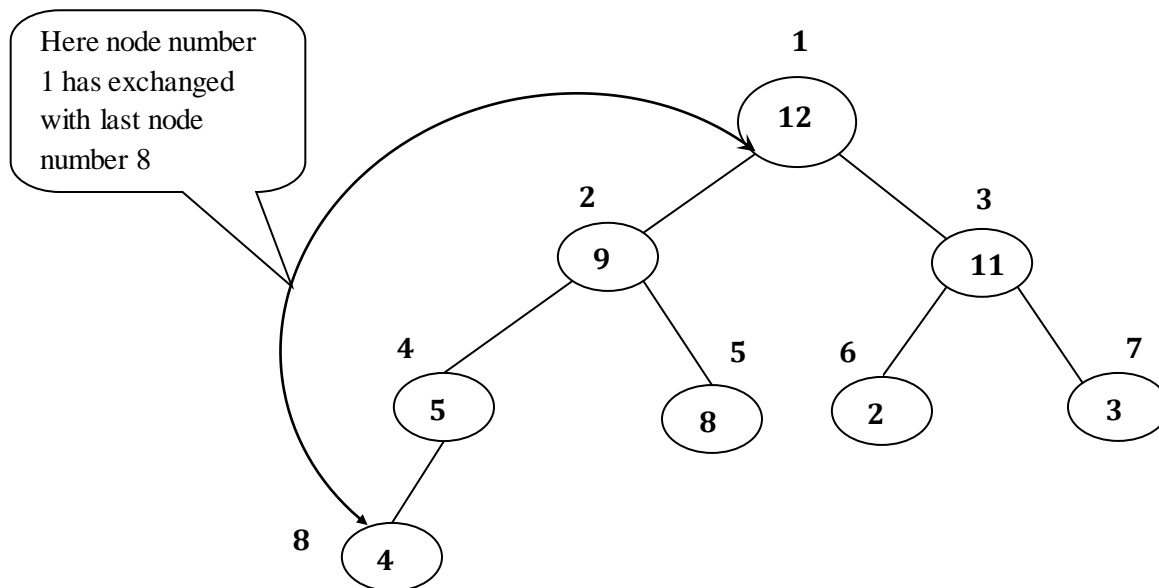
Step 3:Step 4:

Now the elements are:

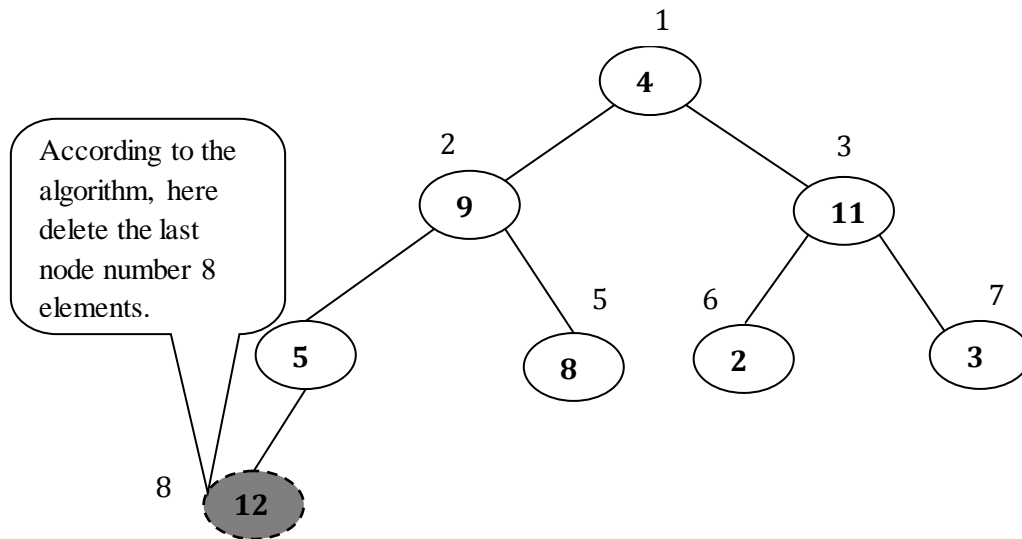
ELEMENTS	12	9	11	5	8	2	3	4
----------	----	---	----	---	---	---	---	---

b) All phases of deleteMin operations

Now apply Heap-sort on the elements given in the above table:



Step 5:

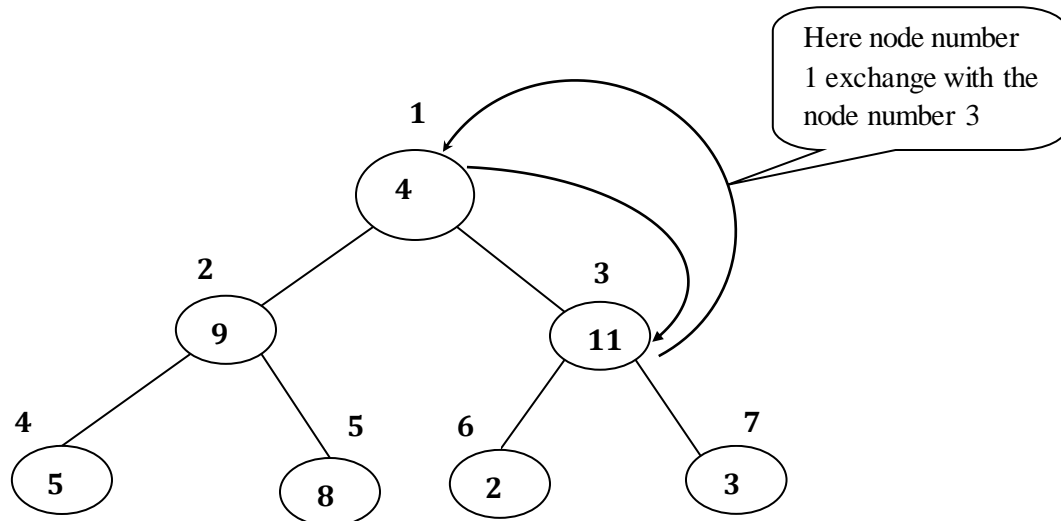


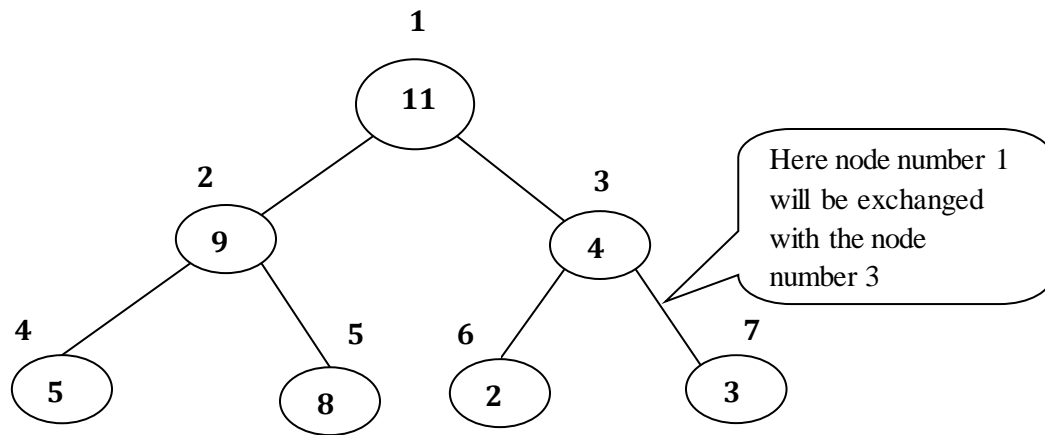
The elements are:

ELEMENTS	4	9	11	5	8	2	3	12
----------	---	---	----	---	---	---	---	----

Step 6:

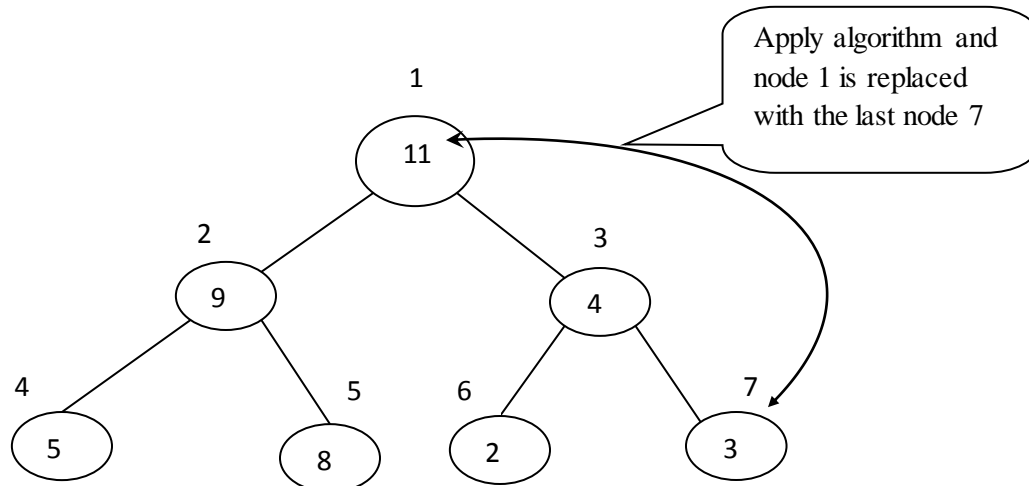
Now apply Heapify,



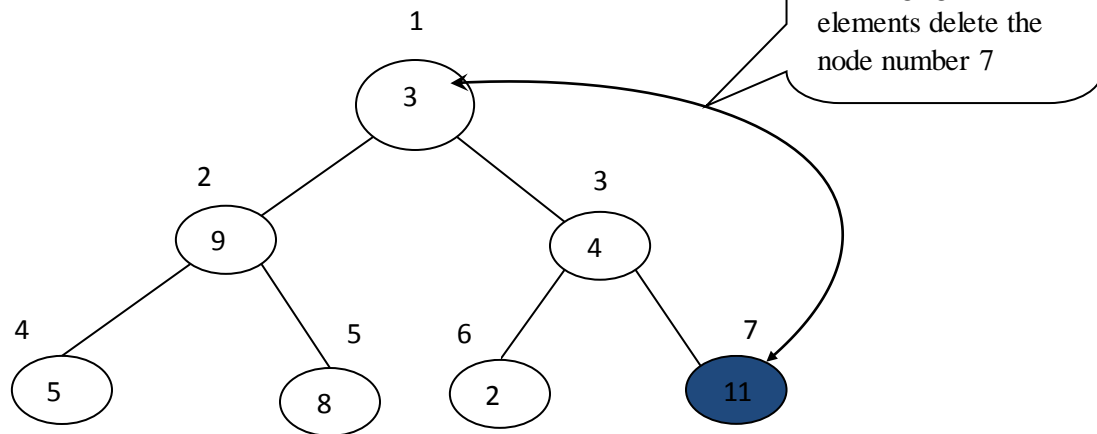
Step 7:

The elements are:

ELEMENTS	11	9	4	5	8	2	3
-----------------	-----------	----------	----------	----------	----------	----------	----------

Step 8:

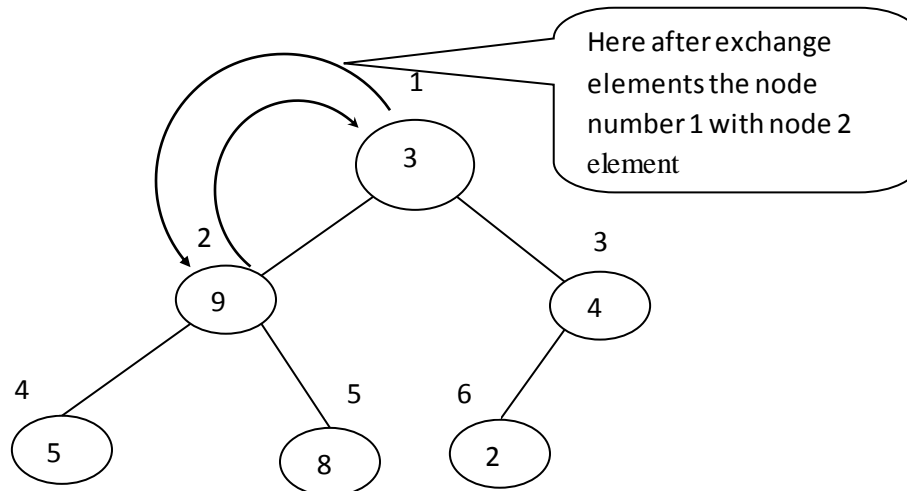
Step 9:

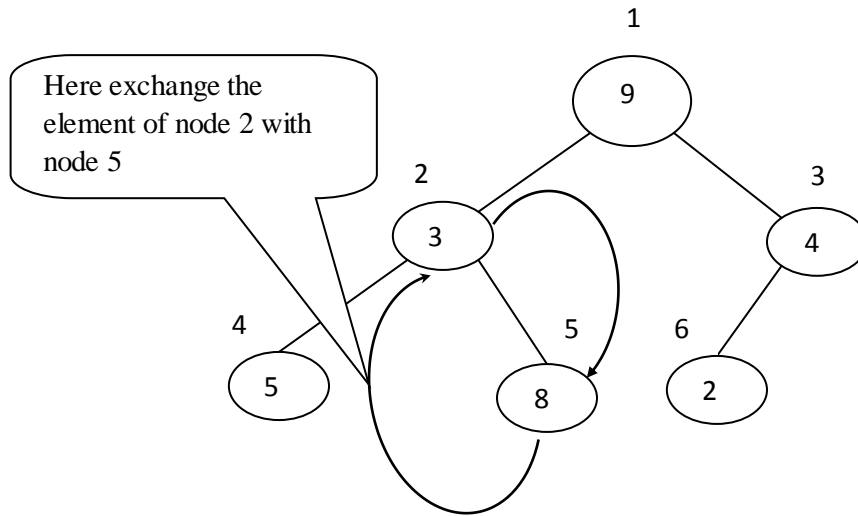
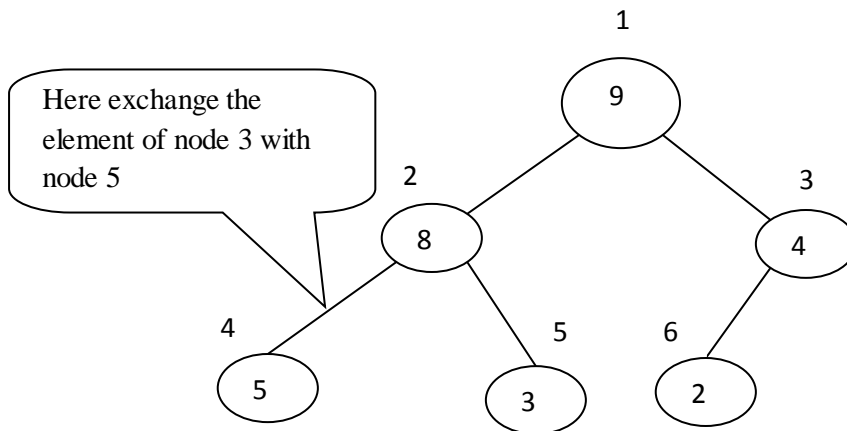


The elements are:

ELEMENTS	3	9	4	5	8	2	11	12
----------	---	---	---	---	---	---	----	----

Apply Heapify,

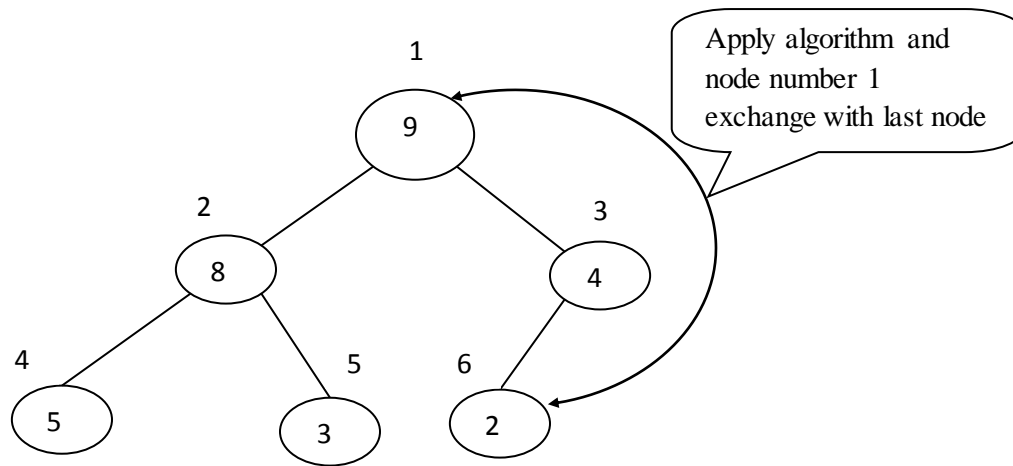


Step 10:Step 11:

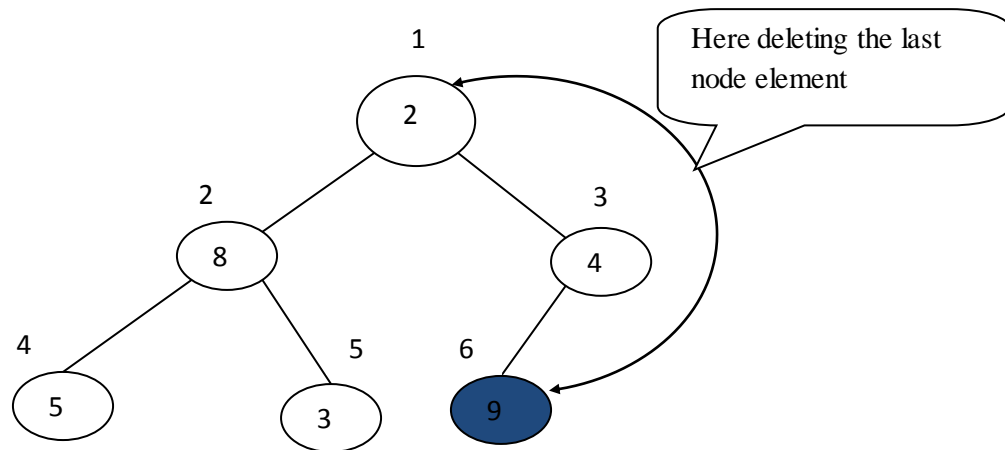
The Elements are:

ELEMENTS	9	8	4	5	3	2
----------	---	---	---	---	---	---

Step 12:



Step 13:



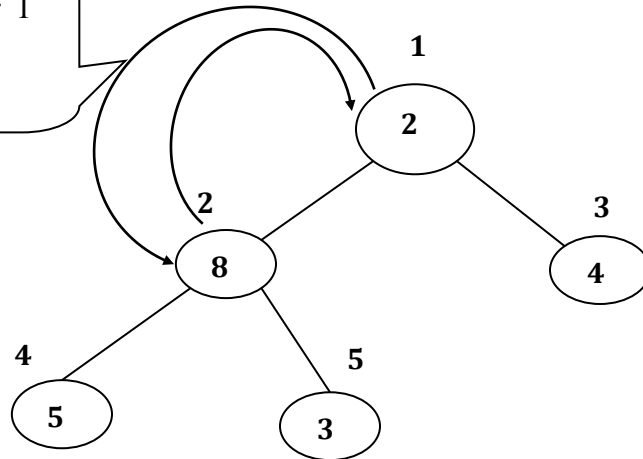
The elements are:

ELEMENTS	2	8	4	5	3	9	11	12
----------	---	---	---	---	---	---	----	----

Step 14:

Now apply Heapify,

Here node number 1
exchange with the
node number 2

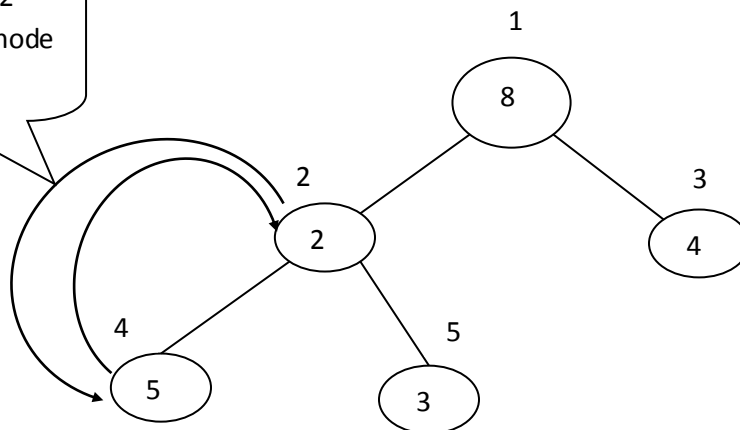


The elements are:

Elements	2	8	4	5	3
----------	---	---	---	---	---

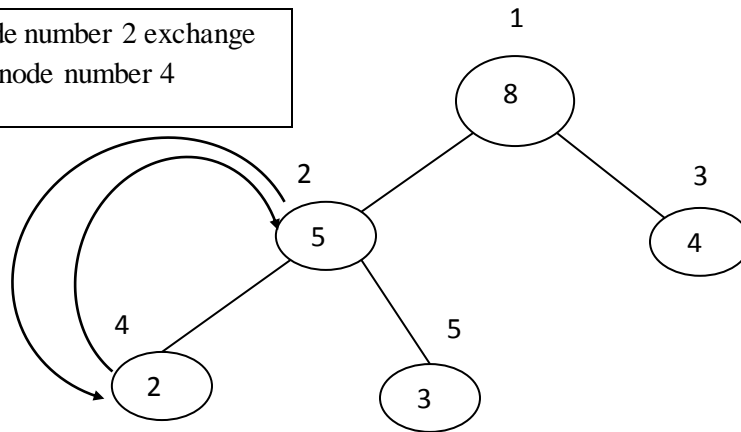
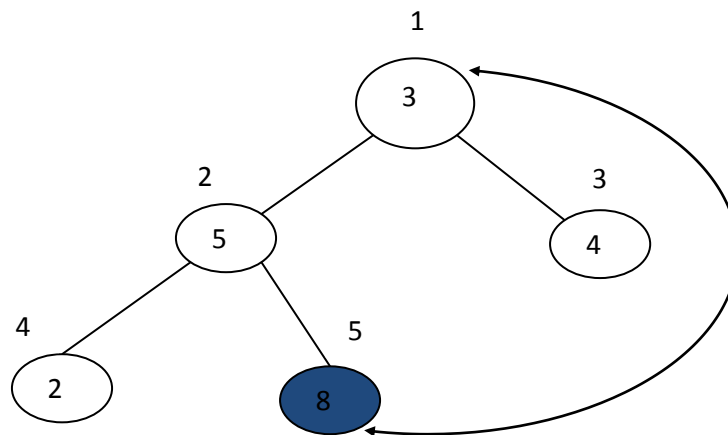
Step 15:

Here node number 2
exchange with the node
number 4



Step 16:

Here node number 2 exchange with the node number 4

Step 17:

Now apply algorithm and node 1 element exchange with last node element 5

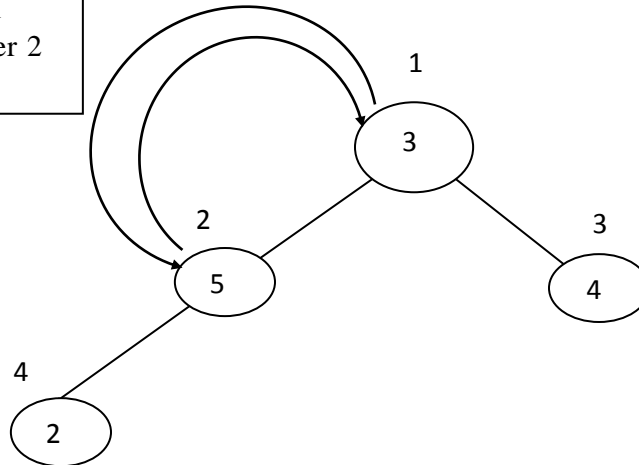
The elements are:

Elements	3	5	4	2	8	9	11	12
----------	---	---	---	---	---	---	----	----

Step 18:

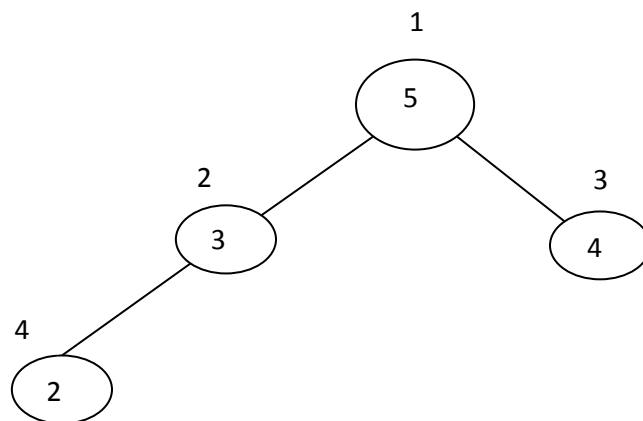
Apply Heapify,

Here exchange node no 1
element with node number 2
element



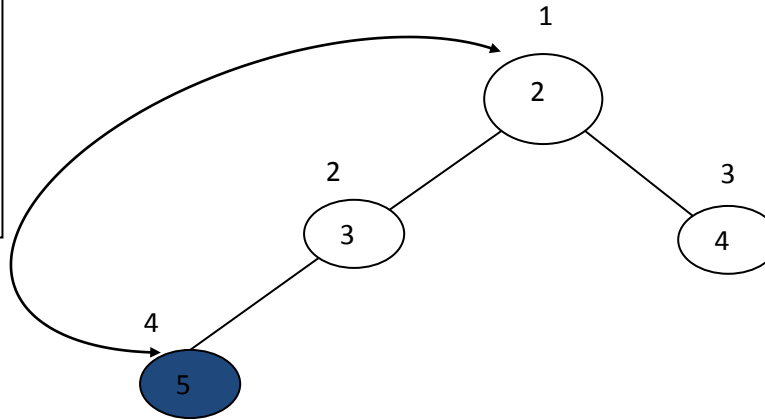
The elements are:

Elements	3	5	4	2
----------	---	---	---	---



Step 19:

Here Apply
algorithm and
exchange node no 1
element with node
no 4 and deleting
the node no 4

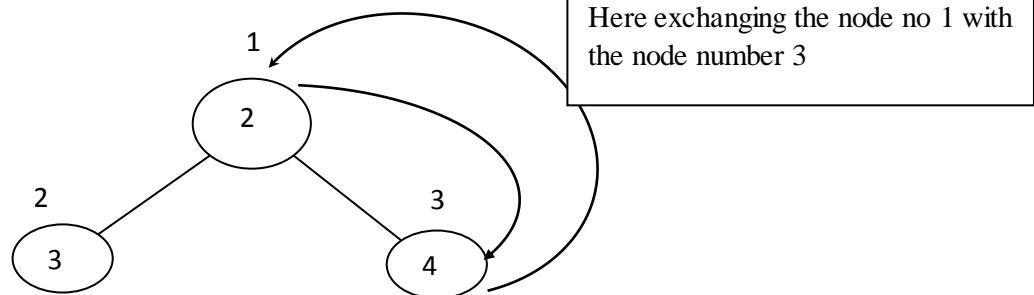


The elements are:

Elements	2	3	4	5	8	9	11	12
----------	---	---	---	---	---	---	----	----

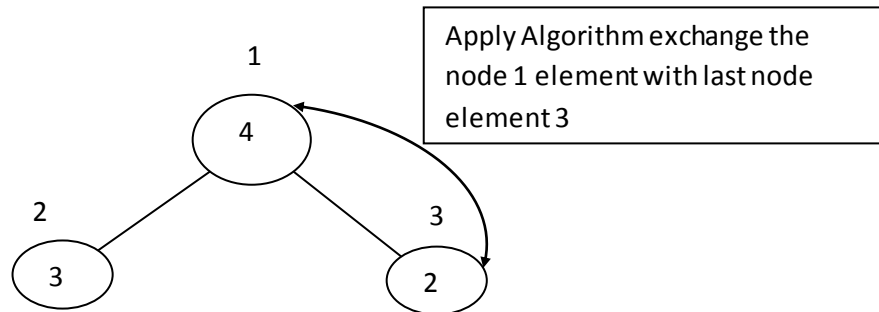
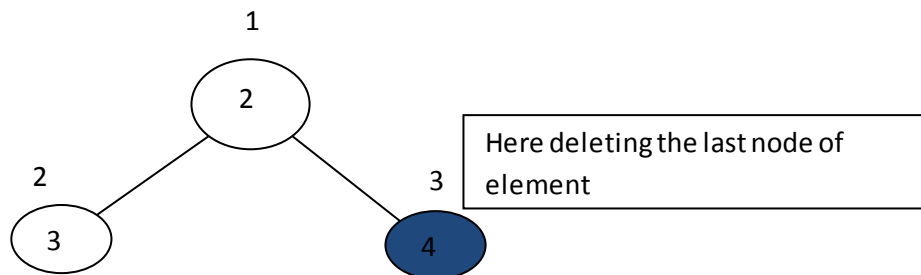
Step 20:

Now apply Heapify,



The elements are:

Elements	2	3	4
----------	---	---	---

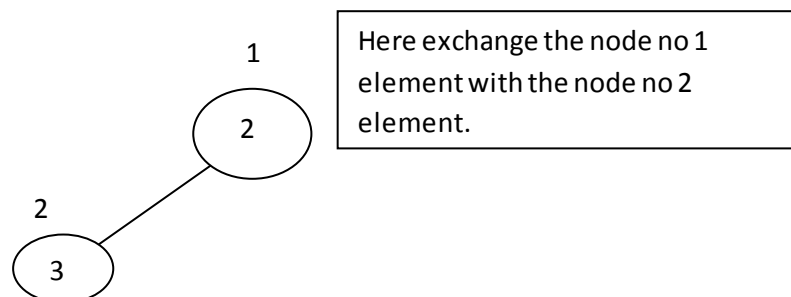
Step 21:Step 22:

The Elements are:

Elements	2	3	4	5	8	9	11	12
----------	---	---	---	---	---	---	----	----

Step 23:

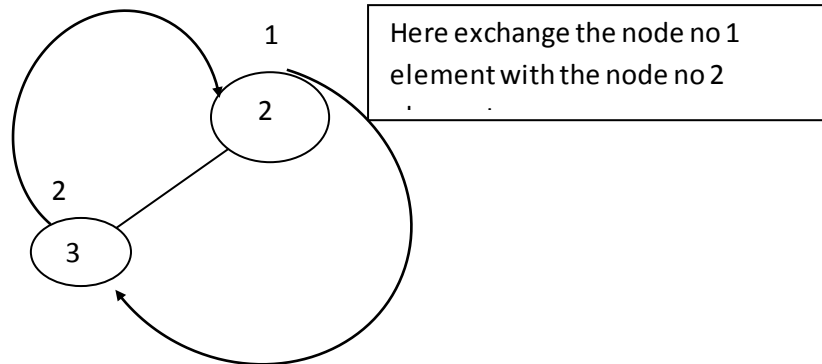
Now Apply Heapify,



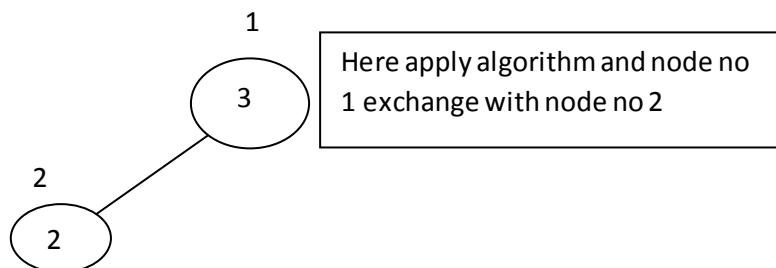
The elements are:

Elements	2	3
-----------------	----------	----------

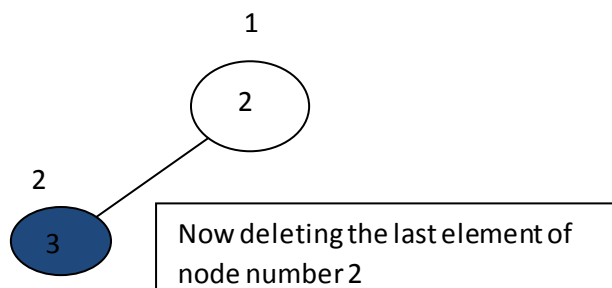
Step 24:



Step 25:



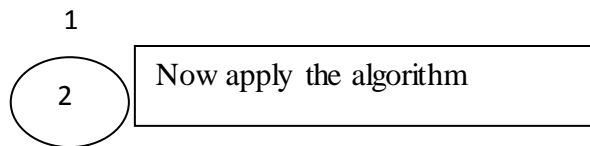
Step 26:



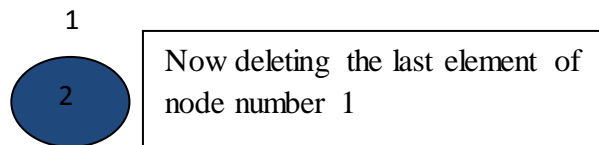
The elements are:

Elements	2	3	4	5	8	9	11	12
-----------------	----------	----------	----------	----------	----------	----------	-----------	-----------

Step 27:



Step 28:



Now the final elements in the sorted form are:

Elements	2	3	4	5	8	9	11	12
-----------------	----------	----------	----------	----------	----------	----------	-----------	-----------

c) Complexity of Heap Sort in terms of Big-oh Notation

MAX-HEAPIFY(A, i, n)

1. $l \leftarrow \text{LEFT}(i)$
2. $r \leftarrow \text{RIGHT}(i)$
3. **if** $l \leq n$ and $A[l] > A[i]$
4. **then** $\text{largest} \leftarrow l$
5. **else** $\text{largest} \leftarrow i$
6. **if** $r \leq n$ and $A[r] > A[\text{largest}]$
7. **then** $\text{largest} \leftarrow r$
8. **if** $\text{largest} \neq i$
9. **then** exchange $A[i] \leftrightarrow A[\text{largest}]$
10. MAX-HEAPIFY(A, largest, n)

TASK	DESCRIPTION
$T(n) = \sum_{i=0}^h n_i h_i$	Cost of HEAPIFY at level i * number of nodes at that level
$= \sum_{i=0}^h 2^i (h - i)$	Replace the values of n_i and h_i computed before
$= \sum_{i=0}^h \frac{h-i}{2^{h-i}} 2^h$	Multiply by 2^h both at the nominator and denominator and write 2^i as $\frac{1}{2^{-i}}$
$= 2^h \sum_{k=0}^h \frac{k}{2^k}$	Change variables: $k = h - i$
$\leq n \sum_{k=0}^{\infty} \frac{k}{2^k}$	The sum above is smaller than the sum of all elements to ∞ and $h = \lg n$
$= O(n)$	The sum above is smaller than 2
Running time of BUILD-MAX-HEAP: $T(n) = O(n)$	

REFERENCES

Websites:

1. *Breadth First Search*

PersonalKent, (2010). Breadth First Search – PersonalKent. [online] Available at: <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm> [Accessed 21 Oct. 2014]

2. *Depth First Search*

Brpreiss, (2012). Depth First Traversal – Brpreiss, [online] Available at: <http://www.brpreiss.com/books/opus4/html/page551.html> Depth First Traversal by Bruno R. Preiss, P.Eng. [Accessed 23 Oct. 2014]

3. *Dijkstra Algorithm*

Ryan Mark (2004) *Dijkstra's Algorithm* [Online]. Available from : <http://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/DijkstraAlgo.html> [Accessed From: 28 Oct. 2014]

4. *Time Complexity of Algorithm*

GeeksforGeeks, (2012). Greedy Algorithms | Set 2 (Kruskal's Minimum Spanning Tree Algorithm) - GeeksforGeeks. [online] Available at: <http://www.geeksforgeeks.org/greedy-algorithms-set-2-kruskals-minimum-spanning-tree-mst/> [Accessed 30 Oct. 2014].

5. *Prim's Algorithm*

Papaioannou Panagiotis (1992) *Prims Algorithm* [Online] Available From: <http://students.ce.id.upatras.gr/~papagel/project/prim.htm> [Accessed 1 Nov. 2014]

6. *Kruskal's Algorithm*

Kruskal J.B.(1989) *Kruskal's Algorithm* [Online] Available from: <http://lcm.csa.iisc.ernet.in/dsa/node184.html> [Accessed 3 Nov. 2014]

Books:

7. *Data Structure and algorithm analysis*

Weiss, M. (1995). Data structures and algorithm analysis. Redwood City, Calif.: Benjamin/Cummings Pub. Co.

8. *Fundamentals of computer algorithm*

Satraj sahani (2008). Fundamental of computer algorithm. 2nd ed. New delhi: Universities Press India pvt.