UNIT – V

GRAPHS

Objective:

- To gain knowledge of heap trees and graph data structure.

Syllabus:

Heap Trees and Graphs

Heap Trees: Basic concept, operations, application-heap sort.

Graphs- Basic concept, representations of graphs, graph traversals-breadth first search and depth first search techniques.

Learning Outcomes:

At the end of the unit student will be able to

- differentiate between Min-heap and Max-heap
- construct Binary heap for the given set of keys
- apply Insertion and Deletion operations on Binary Heap
- Sort the given set of keys using Heap sort
- know various graph representation techniques.
- implement graph traversal techniques.

Learning Material

Binary Heap / Heap Tree

- ***Binary Heap/Heap Tree:*** A Heap is a Complete Binary tree with elements from partially ordered set which satisfies Heap Ordering property. The ordering can be of 2 types:
  1. ***Min Heap:*** For each node N in a complete binary tree, the value of N is less than or equal to the value of its children's, such a heap tree is called a Min Heap.

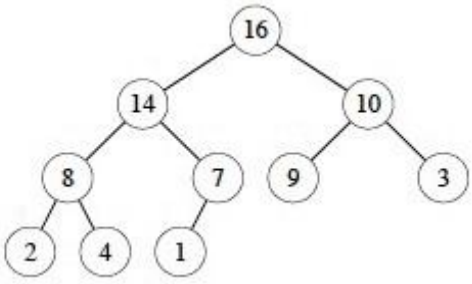


Fig: Min heap

2. ***Max Heap:*** For each node N in a complete binary tree, the value of N is greater than or equal to the value of its children's, such a heap tree is called a Max Heap.

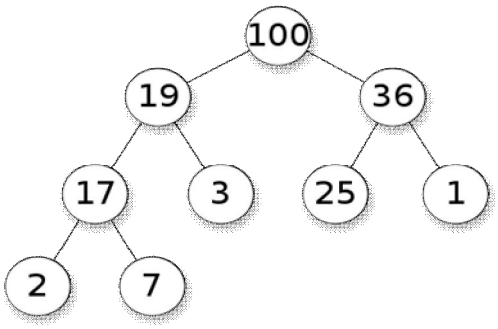
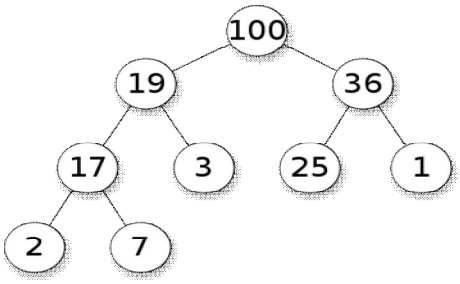


Fig: Max Heap

- ***Properties of Binary Heap:***
  1. *Structure Property:* A Heap is a binary tree that is also a Complete Binary tree.
  2. *Heap Ordering property:*
     - Min Heap property
     - Max Heap property

- **Array Representation of Heap Tree/Binary Heap:**
  - A heap tree can be represented using array and linked list.
  - For an element in array position i, the left child is in the position 2i, the right child is at position (2i+1) i.e in the cell after the left child.
  - The parent is in position floor(i/2).



Array -

100	19	36	17	3	25	1	2	7	-	-	-	-	-	-
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**8**	**10**	**11**	**12**	**13**	**14**	**15**

Heap ADT:

1. Inserting a new element
2. Deleting an element

INSERTION OPERATION ON BINARY HEAP:

- To insert an element say x, into the heap with n elements, we first create a hole in position (n+1) and see if the heap property is violated by putting x into the hole.

- If the heap property is not violated, then we have found the correct position for x. Otherwise, we "Reheap –up"' or "percolate-up"' x until the heap property is restored.
- *Percolate-up / Reheap-up:* we slide the element that is in the hole's parent node into the hole, thus bubbling the hole up toward the root. We continue this process until x can be placed in the whole.
- *Algorithm:*

```
Algorithm Insert_Maxheap( A[1…N], N, X )
{
        N=N+1;
        A[N]=X;
        Reheap_up(N);  /* rebuild heap tree if the heap ordering property is violated */
}
Algorithm Reheap_up(node M)
{
        while(M>1)
        {
                parent=M/2;
                if(A[parent] > A[M])        /*  if(A[parent < A[M]) in case of Min-heap */
                {
                        temp=A[M];
                        A[M]=A[parent];
                        A[parent]=temp;
                        M=parent;
                }
        }
}
```

Example:

- In the below example, 4 is inserted at the last position, which satisfies complete binary tree property.
- Now assume newly inserted node as current node and x is the parent of the current node.
- Now compare current node value with its parent(x). If current node is less than parent(x) node value, then interchange the current node value and x value.

Continue the re-heap up to the current node is less than the child node value.

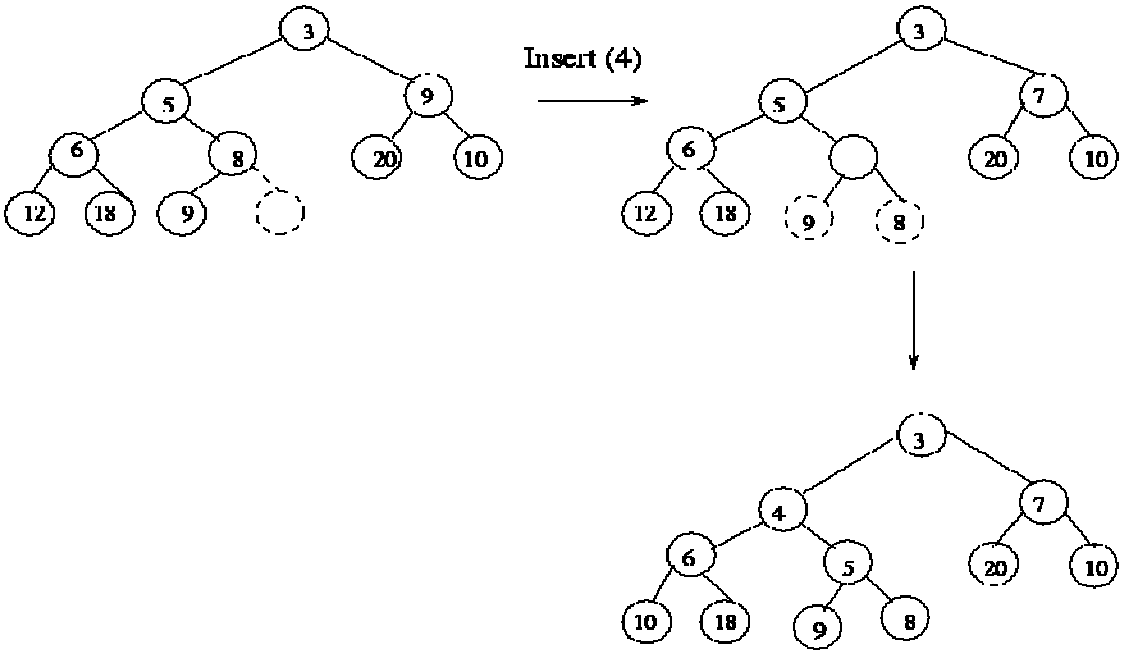


Fig: Insertion of node **4** into Heap tree

- Worst-case complexity of insert is O (h) where h is the height of the heap.
- Thus insertions are O (log n) where n is the number of elements in the heap.

DELETION OPERATION ON BINARY HEAP:

- When the minimum is deleted, a hole is created at the root level. Since the heap now has one less element and the heap is a complete binary tree, the element in the least position is to be relocated.
- First place the last element in the hole created at the root.
- This will leave the heap property possibly violated at the root level. We now “Reheap-down” or ``percolate-down'' the hole at the root until the violation of heap property is stopped.
- *Percolate-down/Reheap-down:* we slide the smaller of the hole’s children into the hole, thus pushing the hole down one level.

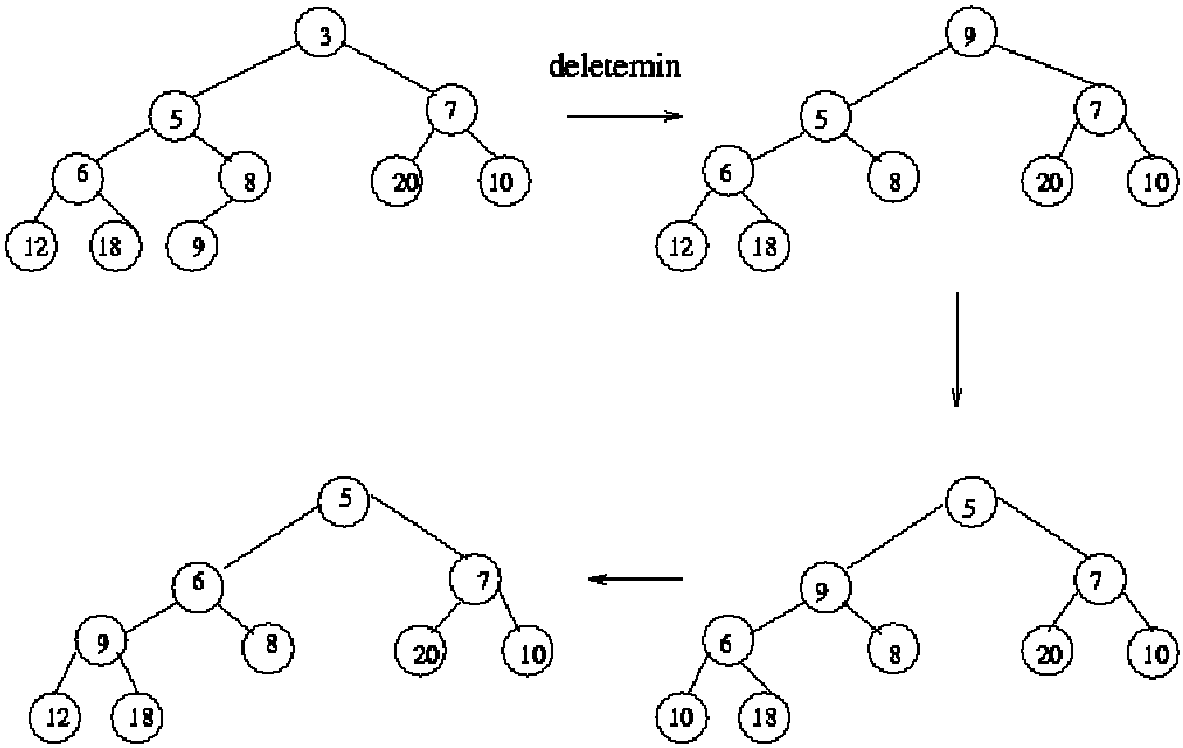
Algorithm:

Algorithm DeleteMin(A[1…N], N, X)
{

```
if ( N == 0)
        print(“Heap tree is empty, deletion not possible”)
else
{
        A[1]=A[N];
        A[N]=0;
        N--;
```

```
        Reheap_down(1);          /* rebuild heap tree if the heap ordering property is violated */
        }
}
Algorithm Reheap_down(node M)
{
        while(2*M <= N)
        {
                left=2*M;
                right=2*M+1;
                if(right <=N and A[right] < A[left])
                        target=right;
                else
                        target=left;
                if(A[target] < A[M])      /* if (A[target] > A[M]) in case of  Max-heap */
                {
                        temp=A[M];
                        A[M]=A[target];
                        A[target]=temp;
                        M=target;
                }
        }
}
```

Example:



- The worst-case time complexity of deletemin/deletemax is O (log n) where n is the number of elements in the heap

Heap Using Linked List

It doesn't make any sense at all to implement a heap as a linked list. Heaps are inherently binary trees. You can store a heap in an array because it's easy to compute the array index of a node's children: the children of the node at position K live at positions 2K and 2K+1. It's massively more efficient to find the K^{th} element of an array than the K^{th} element of a linked list.

Advantages of storing a heap as an array rather than a pointer-based binary tree include the following.

- Lower memory usage (no need to store three pointers for every element of the heap).
- Easier memory management (just one object allocated, rather than N).
- Better locality of reference (the items in the heap are relatively close together in memory rather than scattered wherever the allocator put them).

HEAP SORT:

- It is a Comparison based sorting algorithm.
- It is an In place sorting method, because it does not require any extra storage space other than the input storage list.
- Basic steps in the Heap sort are:
  1. *Create max-heap:* Create a max-heap with n elements stored in the array A

     for i = n down to 2 do
  2. *Remove max:* select the value in the root node and swap it with the value at the i^{th} location in A
  3. *Rebuild heap:* rebuild the heap tree for elements A[1,2,……., i-1]

Algorithm:

Input: An array A[1,2,…..n] where n is the number of elements to be sorted
Output: n elements are arranged in A in ascending order

```
Algorithm Heapsort(A, n)
{
        CreateMaxHeap(A, n)
        for i=n down to 2
        {
                temp=A[1]               //swap(A[1], A[n])
                A[1]=A[n]
                A[n]=temp
                Reheap_down(1,i-1)
        }
}
Algorithm CreateMaxHeap(A, n)
{
        for i=n/2 down to 1
```

Reheap_down(i, n)

}

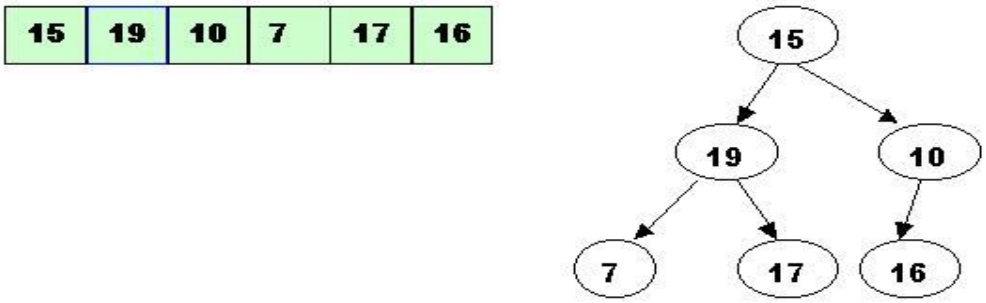
```
Algorithm Reheap_down(M, N)
{
        while(2*M <= N)
        {
                left=2*M;
                right=2*M+1;
                if(right <=N and A[right] < A[left])
                        target=right;
                else
                        target=left;
                if(A[target] < A[M])
                {
                        temp=A[M];
                        A[M]=A[target];
                        A[target]=temp;
                        M=target;
                }
        }
}
```

Example:

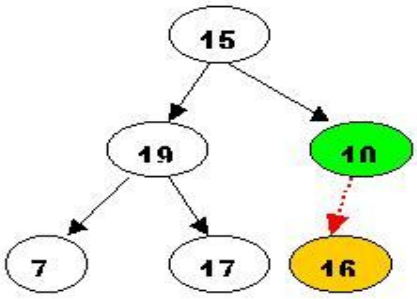
Given an array of 6 elements: 15, 19, 10, 7, 17, 16, sort it in ascending order using heap sort.

A. Building the heap tree

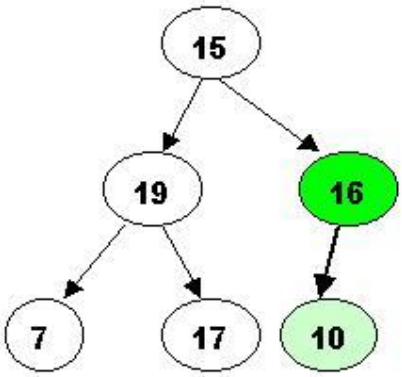
The array represented as a tree, complete but not ordered:



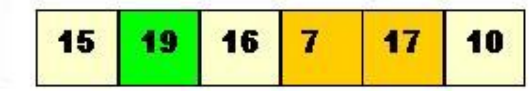
Start with the rightmost node at height 1 - the node at position 3 = Size/2.
It has one greater child and has to be percolated down:

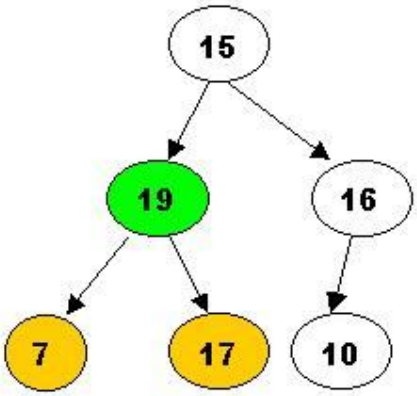


After processing array[3] the situation is:

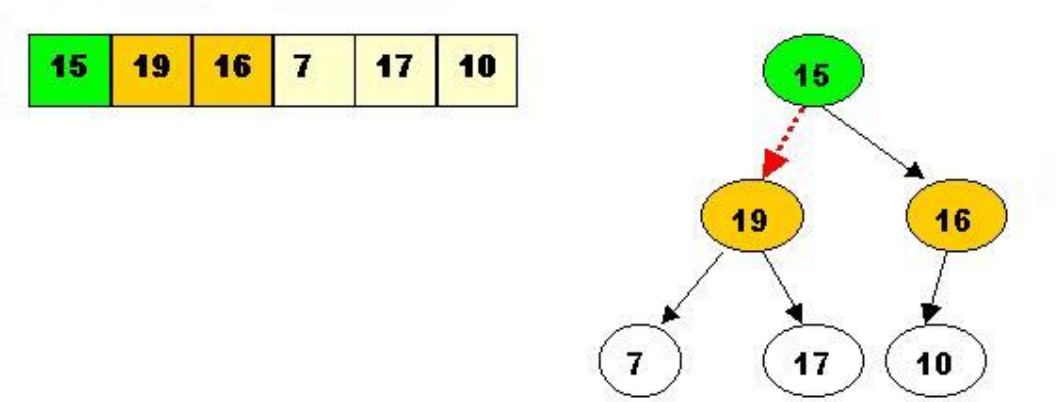


Next comes array[2]. Its children are smaller, so no percolation is needed.

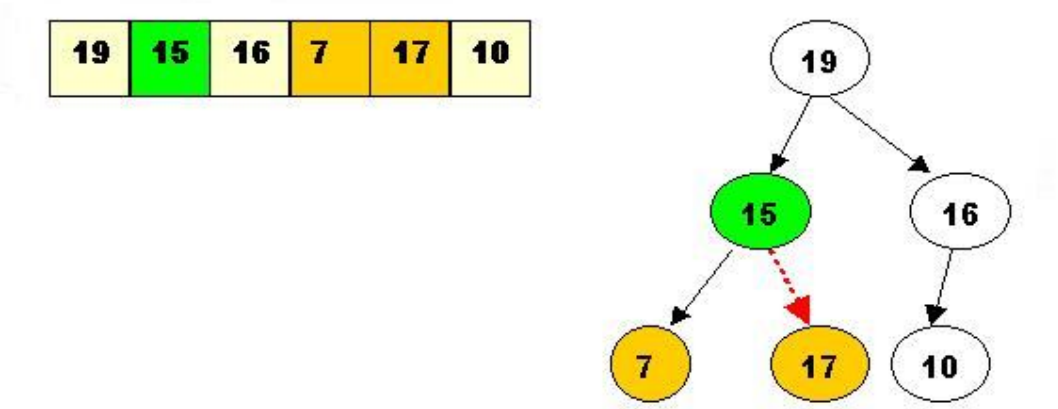




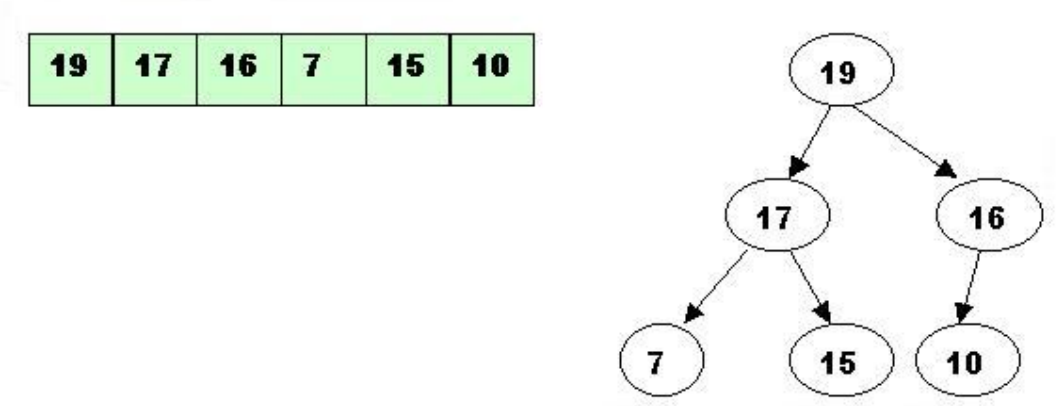
The last node to be processed is array[1]. Its left child is the greater of the children.
The item at array[1] has to be percolated down to the left, swapped with array[2].



As a result the situation is:



The children of array[2] are greater, and item 15 has to be moved down further, swapped with array[5].

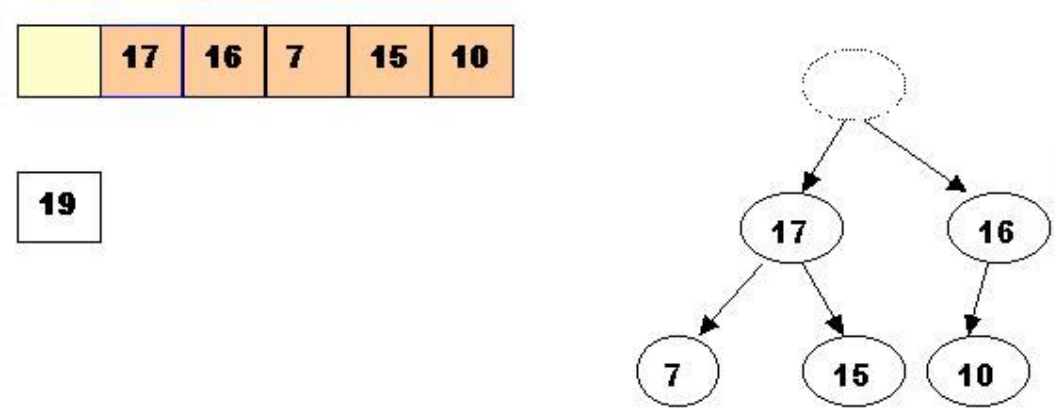


Now the tree is ordered, and the binary heap is built.

B. Sorting - performing deleteMax operations:

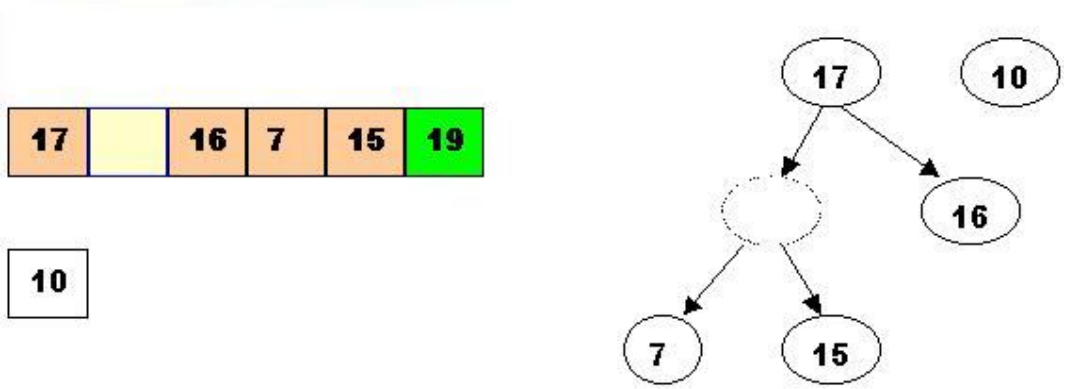
1. Delete the top element 19.

1.1. Store 19 in a temporary place. A hole is created at the top

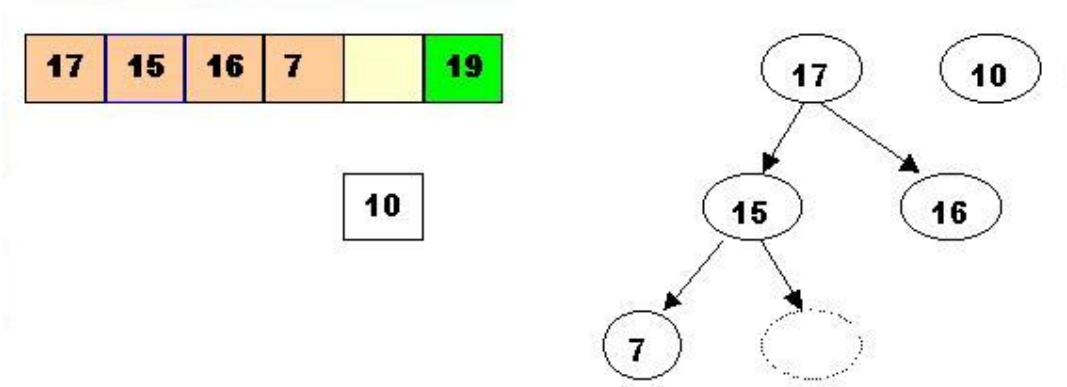


1.2. Swap 19 with the last element of the heap.
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a cell from the sorted array

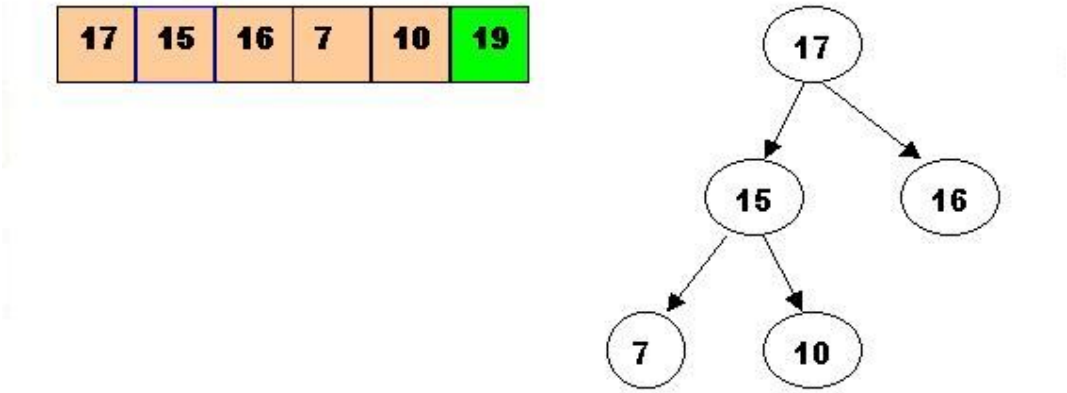
1.3. Percolate down the hole



1.4. Percolate once more (10 is less that 15, so it cannot be inserted in the previous hole)

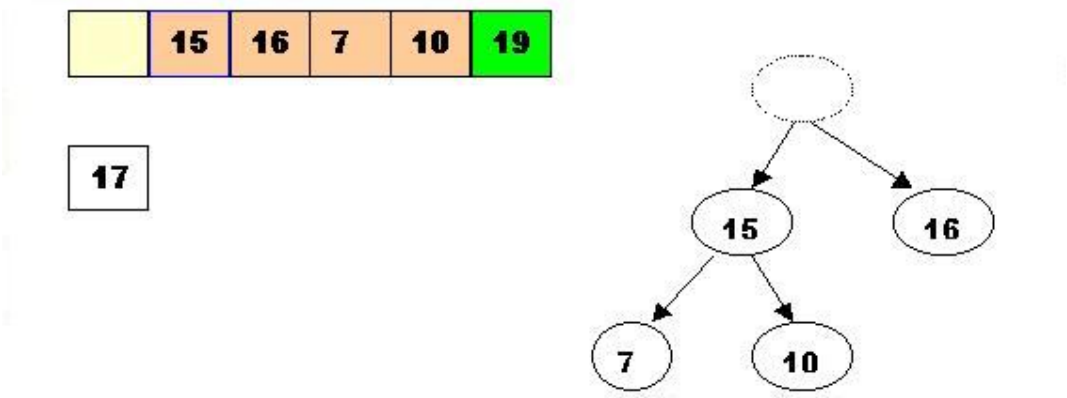


Now 10 can be inserted in the hole



2. DeleteMax the top element 17

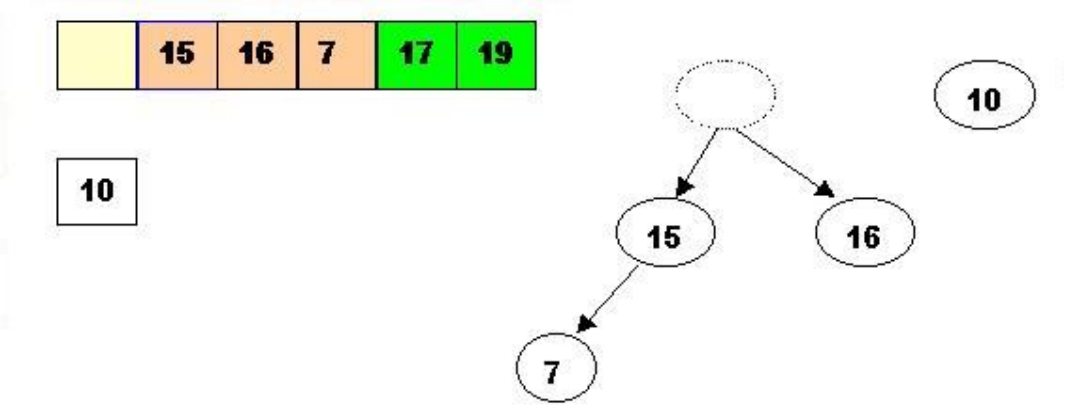
2.1. Store 17 in a temporary place. A hole is created at the top



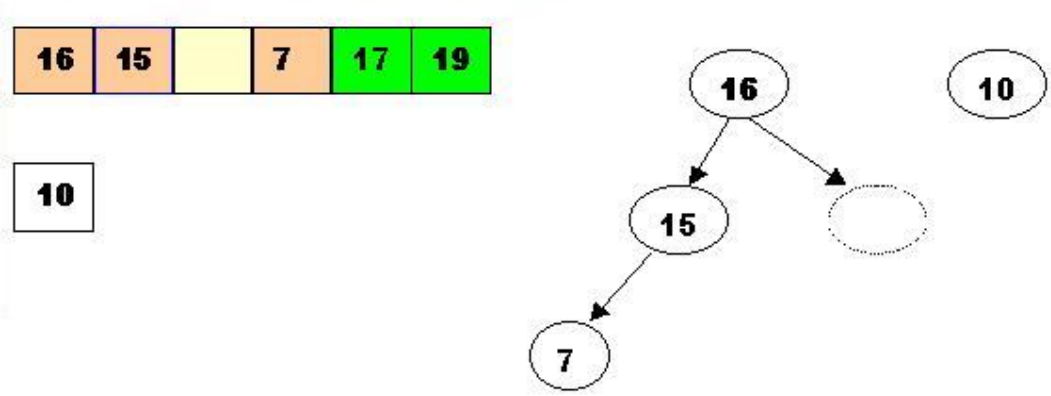
2.2. Swap 17 with the last element of the heap.

As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.

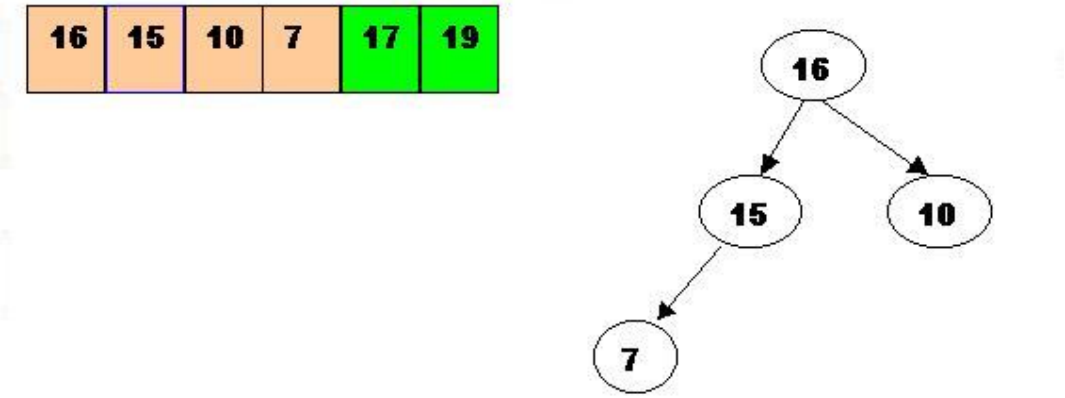
Instead it becomes a cell from the sorted array



2.3. The element 10 is less than the children of the hole, and we percolate the hole down:

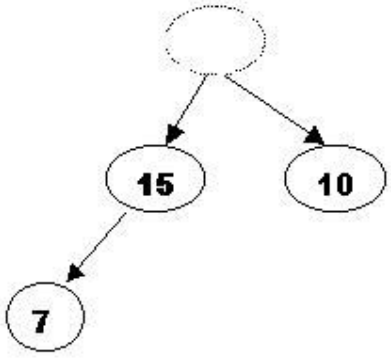


2.4. Insert 10 in the hole

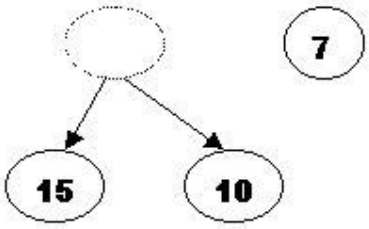


3. DeleteMax 16

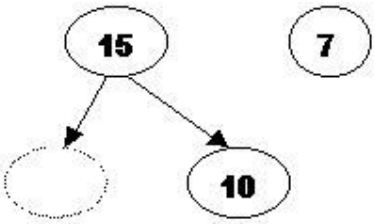
3.1. Store 16 in a temporary place. A hole is created at the top



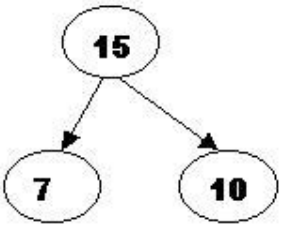
3.2. Swap 16 with the last element of the heap.
As 7 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a cell from the sorted array



3.3. Percolate the hole down (7 cannot be inserted there - it is less than the children of the hole)



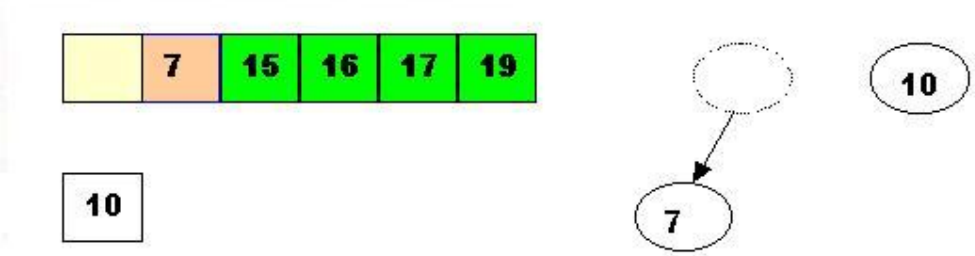
3.4. Insert 7 in the hole



4. DeleteMax the top element 15

4.1. Store 15 in a temporary location. A hole is created.

4.2. Swap 15 with the last element of the heap.
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a position from the sorted array



4.3. Store 10 in the hole (10 is greater than the children of the hole)

5. DeleteMax the top element 10.
5.1. Remove 10 from the heap and store it into a temporary location.

5.2. Swap 10 with the last element of the heap.
As 7 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a cell from the sorted array

5.3. Store 7 in the hole (as the only remaining element in the heap

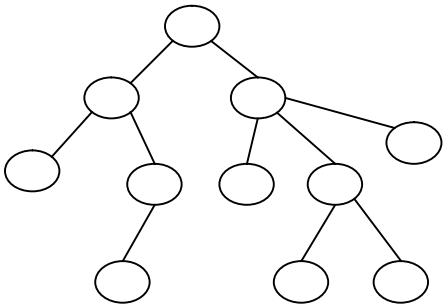
7 is the last element from the heap, so now the array is sorted

- The Time complexity of Heap sort is O(nlogn), where n is the number of elements in the heap

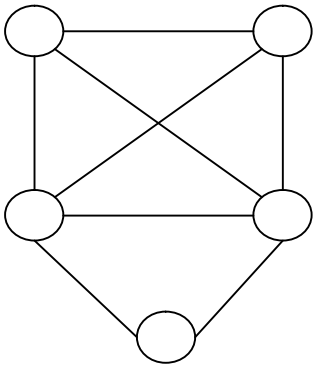
Graphs

Basic Concept:

- **Graph** is another important non-linear data structure.
- The tree structure is a special kind of graph structure.
- In tree structure there is a hierarchical relationship between parent and children, i.e. one parent and many children.
- In the graph relationship is from many parents to many children.



TREE



GRAPH

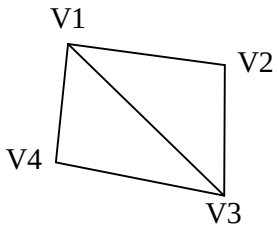
- **Graph Terminology**

  Graph: A graph consists of two sets.

  (i) A set V(G) called as set of all vertices.

  (ii) A set E(G) called as set of all edges (arcs). This set of edges is pair of elements from V(G).

  Eg:

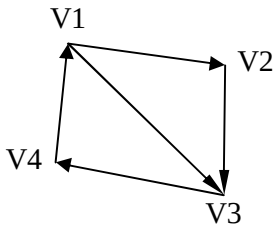


For the above graph V(G)={V1, V2, V3, V4}

E(G) = {(V1,V2), (V2,V3), (V1,V3), (V1,V4), (V3,V4)}

- **Digraph:** A digraph is also called as directed graph. It is a graph G, such that G=<V,E>, where V is set of all the vertices and E is set of ordered pair of elements from V.

  Eg:

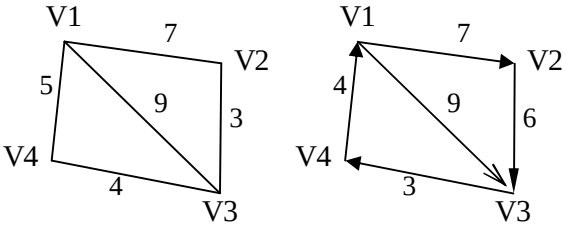


V(G)={V1, V2, V3, V4}

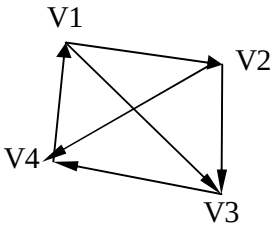
E(G) = {(V1,V2), (V2,V3), (V3,V4), (V4,V1), (V1,V3)}

- **Weighted Graph:** A graph (or digraph) is termed as weighted graph, if all the edges in the graph are labeled with some weights.

  Eg:



- **Adjacent Vertex:** A vertex V_i is adjacent (neighbor of) of another vertex say V_j, if there is an edge from V_i to V_j.



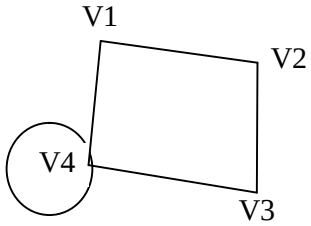
Eg:

V2 is adjacent to V3 and V4.

V1 is not adjacent to V4.

- **Self-loop:** if there is an edge whose starting and ending vertices are same, i.e. (Vi, Vi), then that edge is called as self-loop (loop).

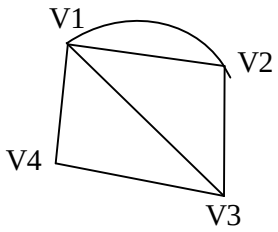
  Eg:



In the above graph Vertex V4 has self-loop.

- **Parallel edges:** if there is more than one edge between the same pair of vertices, then they are known as parallel edges.

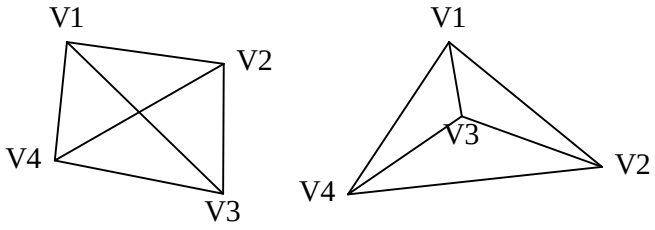
  Eg:



In the above graph two parallel edges between vertices V1 and V2.

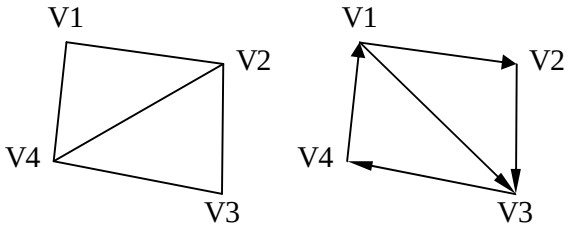
- **Multi graph:** A graph which has either self-looped (or) parallel edges (or) both, that type of graph is called as multi graph.
- **Simple Graph (digraph):** A graph (digraph), if it doesn't have any parallel edges or self-loops, such types of graphs are called as Simple Graph (digraph).
- **Complete Graph:** A graph G is said to be complete graph, if there are edges from any vertex to all other vertices present in Graph.

  Eg:

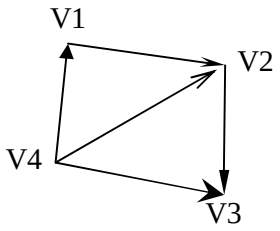


For **n** number of vertices present in complete graph, the total no.of edges are $\frac{n(n-1)}{2}$.

- **Cycle:** If there is a path containing one or more edges, which start from vertex V_i and terminates with same vertex V_i, then that path is known as Cyclic path (or) cycle.

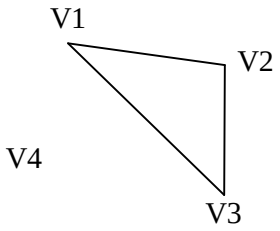


- **Cyclic Graph**: A graph (digraph) that have cycle(s) is called Cyclic Graph (digraph).
- **Acyclic Graph:** A graph (digraph) that does not have cycle(s) is called Acyclic Graph (digraph).



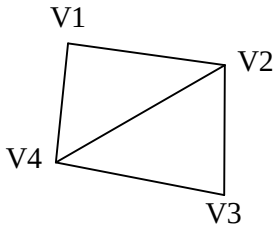
- **Isolated Vertex:** In a graph, a vertex is isolated, if there is no edge is connected from any vertex to the other vertex.

  Eg:



In the above graph V4 is a isolated vertex.

- **Degree of Vertex:** The no.of edges are connected to a vertex is called as degree of a vertex and is denoted by degree(V_i).

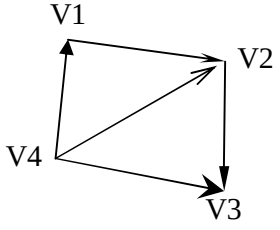


degree(V1) = 2

degree(V2) = 3

  - For **digraph**, there are two edges. i.e. indegree and outdegree.
  - Indegree of V_i denoted as indegree(V_i) = no.of edges coming towards vertex V_i.
  - Outdegree of Vi denoted as outdegree(V_i) = no.of edges coming out from vertex V_i.

  Eg:

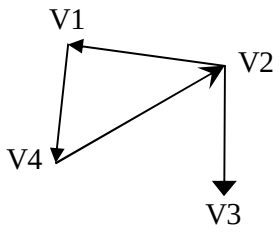


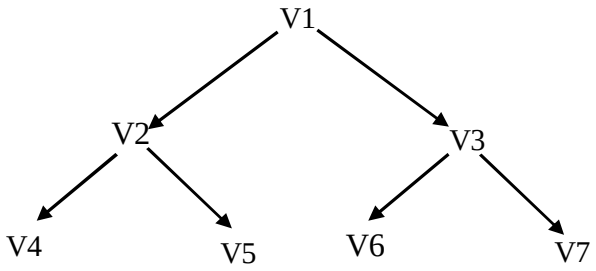
Indegree(V1) = 1 Indegree(V2) = 2

Indegree(V4) = 0 Outdegree(V4) = 3

- **Pendent Vertex:** A vertex V_i is pendent, if its indegree(V_i) = 1and Outdegree(V_i) = 0.

Eg:



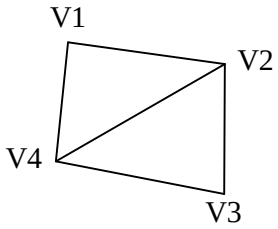


Here V3 is Pendent Vertex. Here V4, V5, V6 and V7 are Pendent Vertices.

- **Connected Graph:** In a graph (not digraph) G, two vertices V_i and V_j are said to be connected, if there is a path in G from V_i to V_j (or) V_j to V_i.

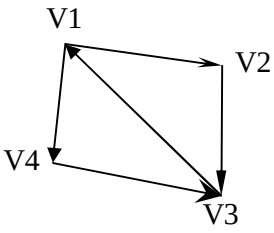
A Graph G is said to be the connected, if for every pair of distinct vertices V_i, V_j in graph, there is a Path.

Eg:

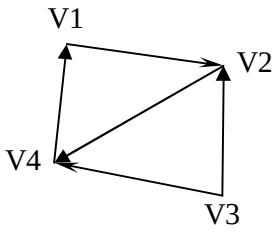


A **digraph** with the above property is called as **Strongly Connected graph**.i.e. a digraph G is said to be Strongly Connected, if for every pair of distinct vertices V_i, V_j in G, there is a Direct path from V_i to V_j and also fromV_j to V_i.

Eg:



Strongly Connected Graph

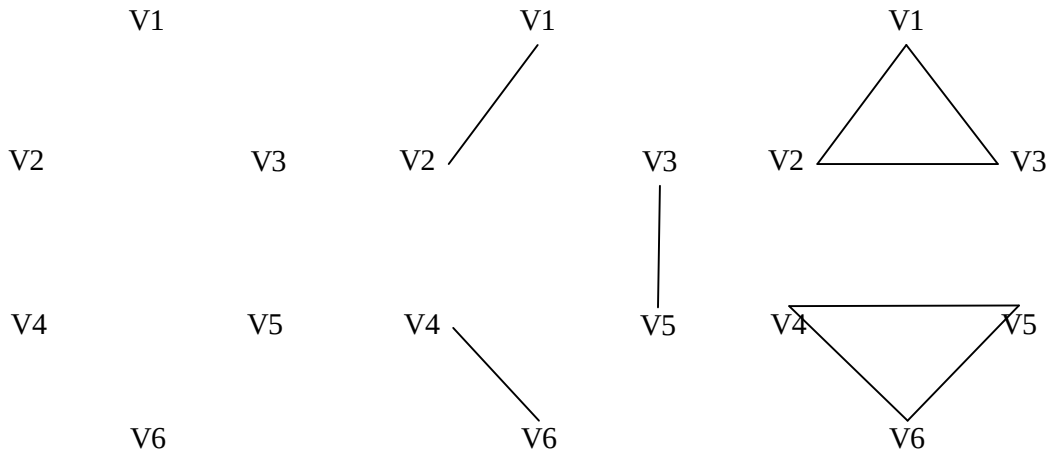


Not Strongly Connected Graph

- **Regular Graph:** In a graph, where every vertex has same degree, such type of graph is called as Regular Graph.

A Regular Graph with vertices of degree K is called as **K-Regular Graph**.

Eg:



0- Regular Graph **1- Regular Graph** **2- Regular Graph**

Representation of a Graph

A graph can be represented in the following ways.

1. Set Representation
2. Linked List Representation (or) Adjacency List Representation
3. Matrix (or) Adjacency Matrix Representation

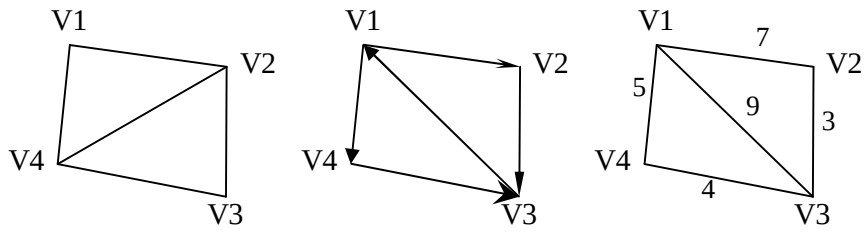
1. Set Representation:

- This is straight forward method of representing graph.
- In this method, 2 sets are maintained V(G) and E(G).

V(G) is set of Vertices.

E(G) is set of Edges.

Eg: G1 G2 G3



V(G1)={V1, V2, V3, V4}

E(G1) = {(V1,V2), (V1,V4), (V2,V4), (V2,V3), (V3,V4)}

V(G2)={V1, V2, V3, V4}

E(G2)={(V1,V2), (V1,V4), (V2,V3), (V3,V1), (V4,V3)}

For representation of weighted graphs, the edge set consist of 3 tuples. i.e. E= W × V × V, where W is the set of edge weights.

V(G3)={V1, V2, V3, V4}

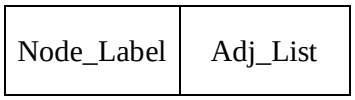
E(G3)={(7,V1,V2), (5,V1,V4), (9,V1,V3), (3,V2,V3), (4,V4,V3)}

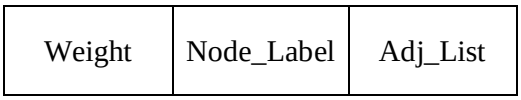
* Multi graphs (undirected) can't be able to represent with the help of Set representation.

2. Linked List Representation:

Linked List Representation is another space saving way of graph representation.

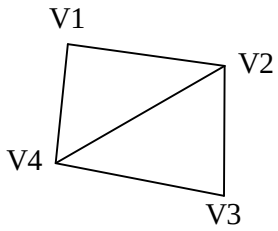
In this graph representation, two types of node structures are assumed.

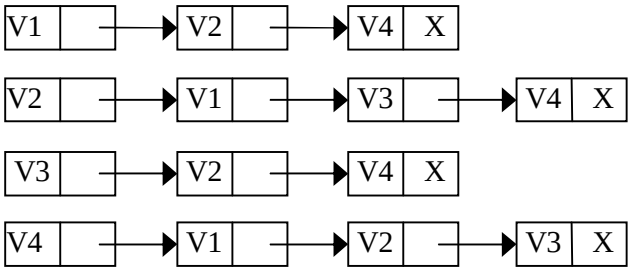




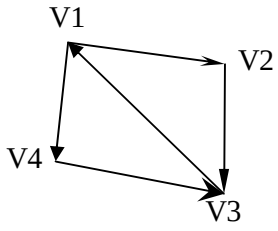
Node Structure for Un-weighted Graph Node Structure for weighted Graph

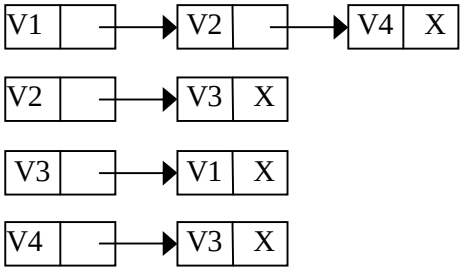
For Graph G1:



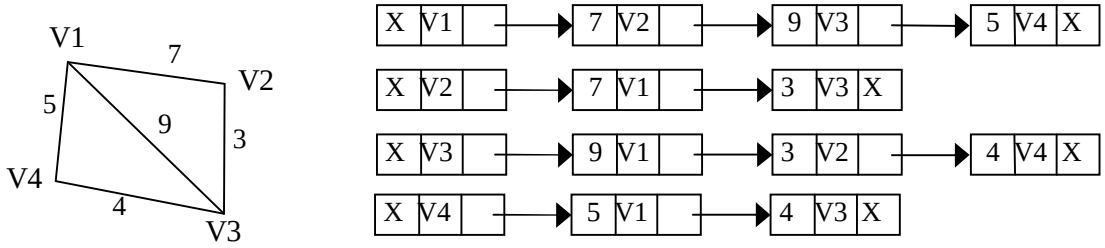


For Graph G2:





For Graph G3:

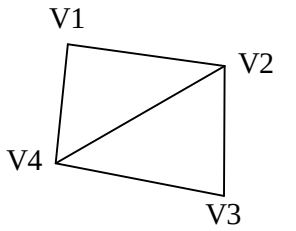


3. Adjacency Matrix Representation:

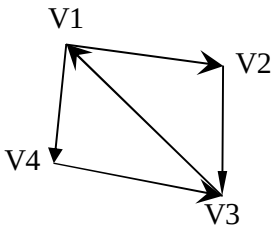
This representation uses a square matrix of order n × n, where n is no.of vertices in graph.

Adjacency Matrix is also termed as Bit matrix (or) Boolean Matrix as the entries are 0 (or) 1.

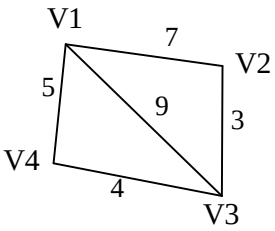
Eg: G1 G2 G3



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



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



	1	2	3	4
1	0	7	9	5
2	7	0	3	0
3	9	3	0	4
4	5	0	4	0

ADT / Operations on Graphs:

1. Insertion
    a. Inserting a new vertex
    b. Inserting a new edge
2. Deletion
    a. Deleting a existing vertex
    b. Deleting a existing edge
3. Traversal: To visit all the vertices present in the graph.

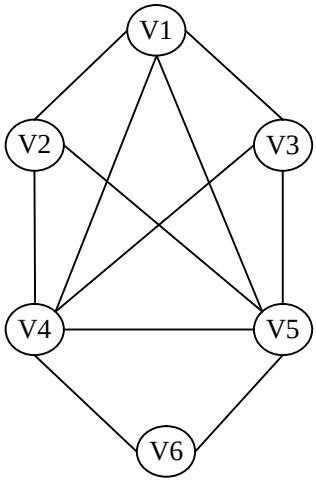
Graph Traversals:

Two techniques are available.

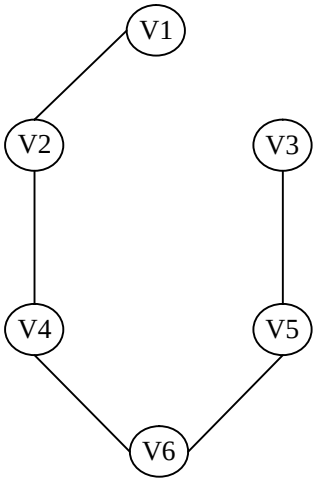
1. Depth First Search (DFS)
2. Breadth First Search (BFS)

Depth First Search (DFS):

- It is similar to the inorder traversal of a binary tree.
- Here, starting from the given node, DFS traversal visit all the nodes upto the deepest level and so on.
- While traversing the vertices, Cyclic path (Closed path) can't be occur. i.e. we can visit a vertex only once.



Graph



DFS of Graph

The sequence of visiting of vertices for the above as follows.

V1-V2-V4-V6-V5-V3

- To traverse a graph in DFS, a stack and one single linked kist is required.
- A stack can be used to maintain the track of all paths from a vertex. Let the name of the stack is OPEN.
- A Single Linked List, VISIT can be maintained to store the vertices already visited.
- Here OPEN is the name of the stack and VISIT is the name of the Single Linked List.

Process:

- Initially the starting vertex will be pushed on to the stack OPEN.
- To visit a vertex, POP a vertex from stack OPEN, and then PUSH all the adjacent vertices into stack OPEN.
- Whenever a vertex s popped, check whether it is already visited or not by searching in Single Linked List VISIT.

- If the vertex is already visited, then we will simply ignore it and we will POP the stack OPEN for the next vertex to be visited. This procedure is continued till the stack is not empty.

Informal description for DFS Traversal of a graph using *array representation* as follows:

Algorithm DFS( )

Input: Adjacent matrix representation of a graph.

Output: DFS traversal of graph.

1. PUSH the starting vertex into stack
2. While stack is not empty
   a) POP a vertex v from stack
   b) if vertex v is not visited
      (i) visit the vertex v
      (ii) PUSH all the adjacent vertices of v into stack
   c) end if
3. end loop

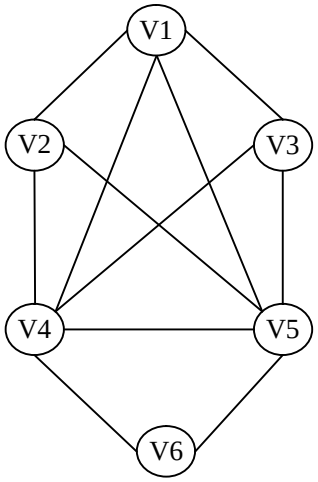
End DFS

Breadth First Search (BFS)

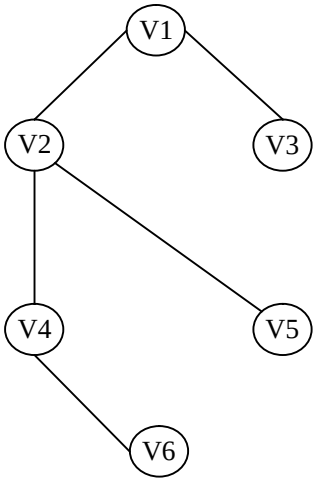
Here any vertex in level i will be visited only after the visiting of all the vertices present in the preceding level. i.e. level i-1.

Simply BFS can be call as Level-by-Level traversal.

Eg:



Graph



BFS of Graph

The order of visiting of vertices in the above BFS traversal of a graph as follows.

V1-V2-V3-V4-V5-V6

- The implement idea of BFS traversal is almost same as DFS traversal except that, BFS uses Queue Data Structure instead of Stack Data Structure in DFS.
- Let us assume the name of the Queue is OPENQ to use it in BFS and Single Linked List is VISIT, to store the order of vertices visited during the BFS traversal.

Informal description for BFS Traversal of a graph using array representation as follows:

Algorithm BFS( )

Input: Adjacent matrix representation of a graph.

Output: BFS traversal of graph.

1. ENQUEUE the starting vertex into queue
2. While queue is not empty
   a) DEQUEUE a vertex v from queue
   b) if vertex v is not visited
      (i) visit the vertex v
      (ii) ENQUEUE all the adjacent vertices of v into queue
   c) end if
3. end loop

End BFS

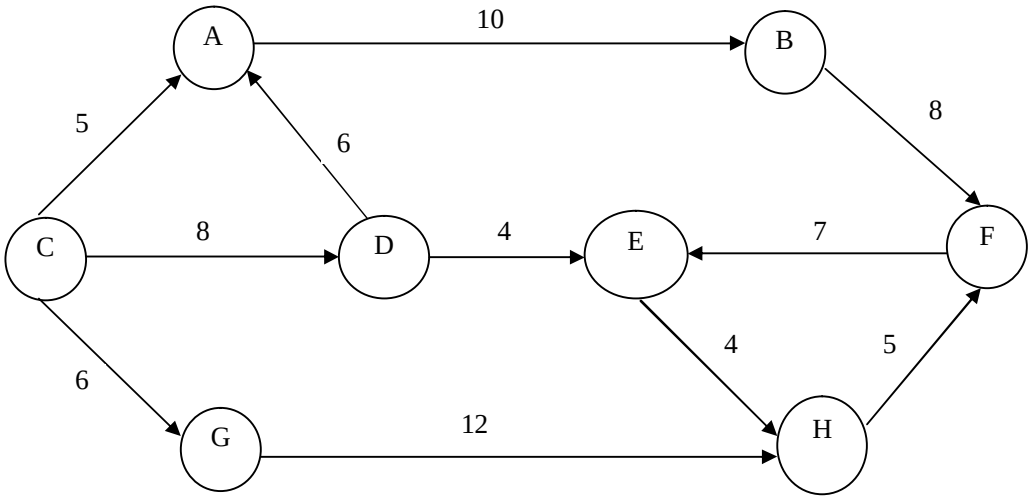
UNIT-V
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. A _____ is a heap where the value of each parent is less than or equal to the values of its children.
2. Consider any array representation of an n element binary heap where the elements are stored from index 1 to index n of the array. For the element stored at index i of the array (i <= n), the index of the left child and right child are _____. [ ]

   A) 2i+1, 2i B) 2i+1, floor(i/2) C) 2i, floor(i/2) D) 2i, 2i+1

Consider the following graph and answer to the questions 3 to 9



3. The above graph is _____ [ ]

   A) Complete Graph B) Weighted Graph
   C) Multi Graph D) None of the above

4. In the above graph which of the following is a pendant vertex? [ ]

   A) vertex B B) vertex D C) vertex E D) None of the above

5. In the above graph indegree and outdegree of vertex H is ___ [ ]

   A) indegree - 2 outdegree – 0 B) indegree - 3 outdegree - 0
   C) indegree - 3 outdegree – 1 D) indegree - 2 outdegree - 1

6. The above graph is a _____ [ ]

   A) Connected Graph B) Simple Graph
   C) Cyclic graph D) None of the above

7. The node A is adjacent to _____ node. [ ]

A) B B) C C) D D) None

8. In the above graph there is a self loop with vertex _____ [ ]

A) E B) G C) H D)None

9. In a graph if e=(u,v) means ..... [ ]

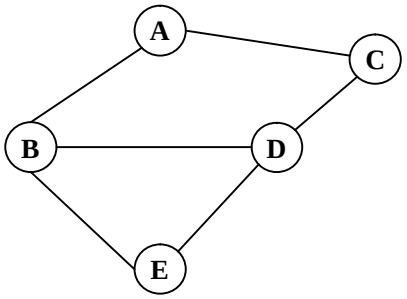
A) u is adjacent to v but v is not adjacent to u.

B) e begins at u and ends at v

C) u is node and v is an edge.

D) both u and v are edges.

Consider the following graph to answer the questions 10 to 10



10. The above graph is a _____ [ ]

A) Weighted graph B) Simple graph C) Acyclic Graph D) None

11. The adjacent vertices of node A are _____ [ ]

A) B, D, E B) B, D, C C) E, D D) None

12. The above graph is a _____ [ ]

A) Connected graph B) Complete graph C) Both A&B D) None

13. For an undirected graph with n vertices and e edges, the sum of the degree of each vertex is equal to [ ]

A) 2n B) (2n-1)/2 C) 2e D) $e^2/2$

14. A graph with n vertices will definitely have a parallel edge or self loop, if the total number of edges are [ ]

(A) more than n (B) more than n+1

(C) more than (n+1)/2 (D) more than n(n-1)/2

15. The maximum degree of any vertex in a simple graph with n vertices is_____ [ ]

16. An adjacency matrix representation of a graph cannot contain information of _____

(A) Nodes (B) edges [ ]

(C) Direction of edges (D) parallel edges

17. How many undirected graphs (not necessarily connected) can be constructed out of a given set V= {V 1, V 2,…V n} of n vertices ? [ ]

(A) n(n-l)/2 (B) 2^n (C) n! (D) 2^(n(n-1)/2)

18. The data structure required for Breadth First Traversal on a graph is _____

(A) Queue (B) Stack (C) Array (D) Tree

19. Which of the following statements is/are TRUE for an undirected graph? [ ]

P: Number of odd degree vertices is even

Q: Sum of degrees of all vertices is even

A) P Only B) Q Only

C) Both P and Q D) Neither P nor Q.

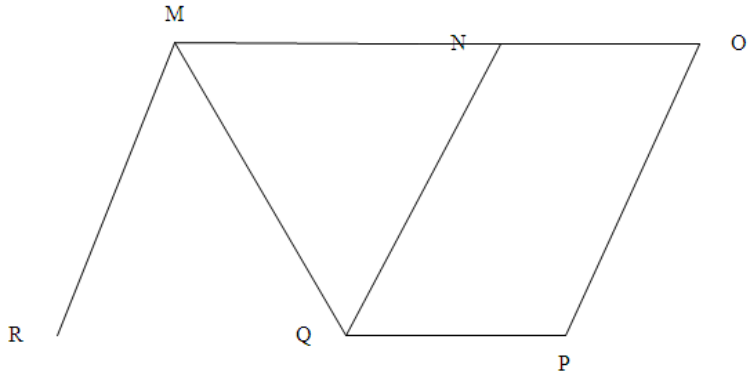
20. The Minimum no.of edges in a connected cyclic graph on n vertices is___? [ ]

(A) n-1 (B) n (C) n+1 (D) None of the above

21. The no.of simple graphs on **n** labled vertices is___ [ ]

(A) n (B) n(n-1)/2 (C) $2^{n(n-1)/2}$ (D) n(n+1)/2

22. The BFS Algorithm has been implemented using Queue Data Structure. One possible order of visiting nodes in the following graph is [ ]



(A) MNOPQR (B) NQMPOR (C) QMNPRO (D) QMNPOR

SECTION-B

Descriptive Questions

1. Show the result of inserting the keys: 14, 5, 12, 6, 4, 8, 9, 13, 11, 2, 18, 30 one at a time into an initially empty Max heap with neat diagrams.
2. Show the result of inserting the keys: 10, 12, 8, 14, 6, 5, 1, 3 one at a time into an initially empty Min heap. Apply deleteMin operation on the resulting min heap
3. Construct a Max heap for the following keys: 4, 67, 23, 89, 12, 8, 7, 44, 78, 64, 70. Apply deleteMax operation on the resulting max heap

4. Sort the following keys using Heap sort: 5, 8, 11, 3, 9, 2, 10, 1, 45, 32.
5. Write adjacency matrix representation of graph with an example.
6. Write linked representation of graph with an example.
7. Write set representation of graph with an example.
8. Write DFS Algorithm& Write BFS Algorithm.
9. Consider the graph given below
    a) Write the adjacency matrix of G1.
    b) Give Linked list representation of G1.
    c) Give Set representation of G1.
    d) Is the graph complete?
    e) Is the graph strongly connected?
    f) Find out the degree of each node.
    g) Is the graph regular?

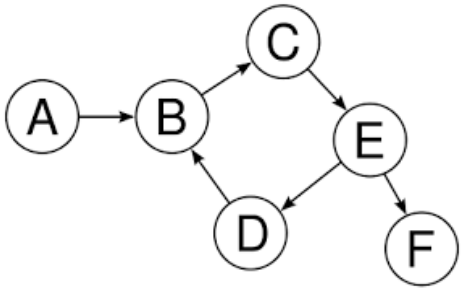
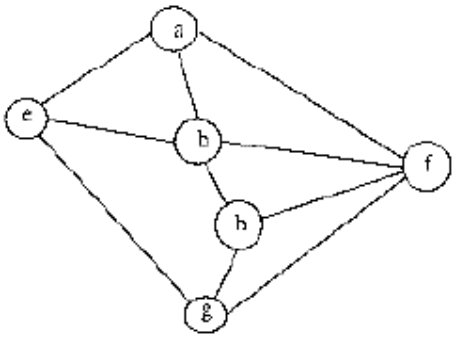


Fig. Graph G1

10. Consider the following adjacency matrix, draw the weighted graph.

$$\begin{pmatrix} 0 & 4 & 0 & 2 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

11. Consider the following graph

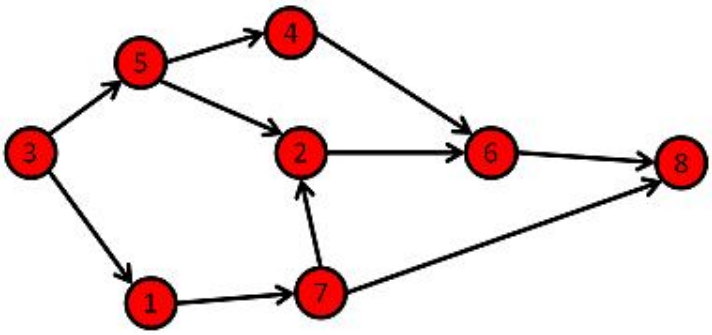


Among the following sequences

i) a b e g h f ii) a b f e h g iii) a b f h g e iv) a f g h b e

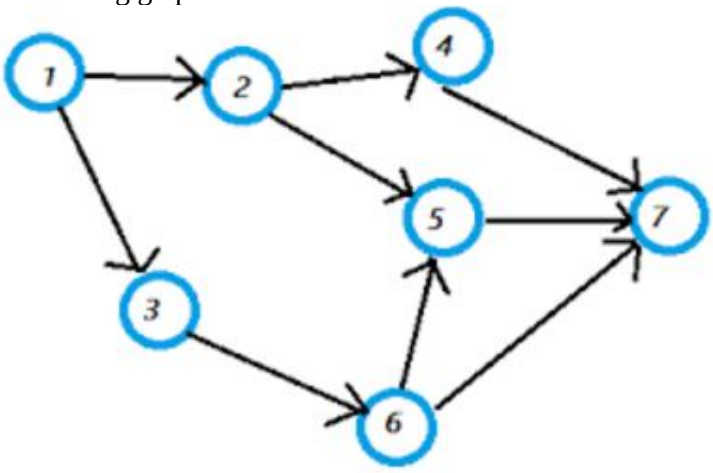
Which are depth first traversals of the above graph?

12. Consider the following graph



What is breadth first traversal of the above graph if starting vertex is 3?

13. Consider the following graph



What is the depth first traversal of the above graph if starting vertex is 1?

Section C

Questions asked in GATE

1. Consider any array representation of an n element binary heap where the elements are stored from index 1 to index n of the array. For the element stored at index i of the array (i <= n), the index of the parent is_____ **(GATE-CS-2001)** [ ]

   A) i-1 B) floor(i/2) C) ceiling(i/2) D)(i+1)/2

2. In a Binary max heap containing n numbers, the smallest element can be found in time **(GATE 2006)** **[ ]**

   A) O(n) B) O(logn) C) O(loglogn) D) O(1)

3. Which of the following sequences of array elements forms a heap? [ ]

   A) {23, 17, 14, 6, 13, 10, 1, 12, 7, 5} **(GATE IT 2006)**

   B) {23, 17, 14, 6, 13, 10, 1, 5, 7, 12}

   C) {23, 17, 14, 7, 13, 10, 1, 12, 5, 7}

   D) {23, 17, 14, 7, 13, 10, 1, 5, 6, 12}

4. Consider a binary max-heap implemented using an array. Which one of the following array represents a binary max-heap? (**GATE CS 2009**) **[ ]**

   A) 25,12,16,13,10,8,14 B) 25,14,16,13,10,8,12

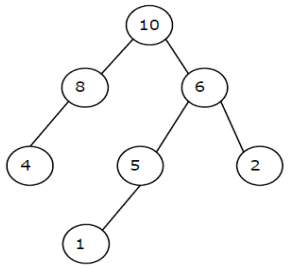
   C) 25,14,12,13,10,8,16 D) 25,16,12,13,10,8,12

5. What is the content of the array after two delete operations on the correct answer to the previous question? **(GATE CS 2009**) [ ]

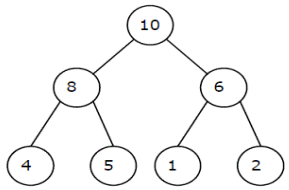
   A) 14,13,12,10,8 B) 14,12,13,8,10 C) 14,13,8,12,10 D) 14,13,12,8,10

6. A max-heap is a heap where the value of each parent is greater than or equal to the values of its children. Which of the following is a max-heap? (**GATE CS 2011**) **[ ]**

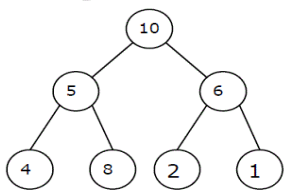
(A)



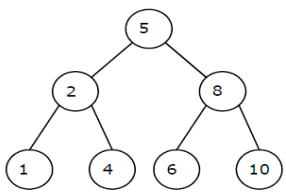
(B)



(C)



(D)



7. A priority queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is: 10, 8, 5, 3, 2. Two new elements 1 and 7 are inserted into the heap in that order. The level-order traversal of the heap after the insertion of the elements is: **(GATE-CS-2014)** [ ]

(A) 10, 8, 7, 3, 2, 1, 5 (B) 10, 8, 7, 2, 3, 1, 5

(C) 10, 8, 7, 1, 2, 3, 5 (D) 10, 8, 7, 5, 3, 2, 1

8. Consider a max heap, represented by the array: 40, 30, 20, 10, 15, 16, 17, 8, 4. Now consider that a value 35 is inserted into this heap. After insertion, the new heap is **(GATE-CS-2015) [ ]**

A) 40, 30, 20, 10, 15, 16, 17, 8, 4, 35 B) 40, 35, 20, 10, 30, 16, 17, 8, 4, 15

C) 40, 30, 20, 10, 35, 16, 17, 8, 4, 15 D) 40, 35, 20, 10, 15, 16, 17, 8, 4, 30

9. A 3-ary max heap is like a binary max heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows: The root is stored in the first location, a[0], nodes in the next level, from left to right, is stored from a[1] to a[3]. The nodes from the second level of the tree from left to right are stored from a[4] location onward. An item x can be inserted into a 3-ary heap containing n items by placing x in the location a[n] and pushing it up the tree to satisfy the heap property. Which one of the following is a valid sequence of elements in an array representing 3-ary max heap? **(GATE 2006)** [ ]

A) 1, 3, 5, 6, 8, 9 B) 9, 6, 3, 1, 8, 5 C) 9, 3, 6, 8, 5, 1 D) 9, 5, 6, 8, 3, 1

10. Suppose the elements 7, 2, 10 and 4 are inserted, in that order, into the valid 3- ary max heap found in the above question, which one of the following is the sequence of items in the array representing the resultant heap? (**GATE CS 2006)** [ ]

A) 10, 7, 9, 8, 3, 1, 5, 2, 6, 4 B) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

C) 10, 9, 4, 5, 7, 6, 8, 2, 1, 3 D) 10, 8, 6, 9, 7, 2, 3, 4, 1, 5

11. Consider the following array of elements. ⟨89, 19, 50, 17, 12, 15, 2, 5, 7, 11, 6, 9, 100⟩. The minimum number of interchanges needed to convert it into a max-heap is **(GATE-CS-2015)** [ ]

A) 4 B) 5 C) 2 D)3

12. An operator delete(i) for a binary heap data structure is to be designed to delete the item in the i-th node. Assume that the heap is implemented in an array and i refers to the i-th index of the array. If the heap tree has depth d (number of edges on the path from the root

to the farthest leaf), then what is the time complexity to re-fix the heap efficiently after the removal of the element? **(GATE 2016)** [ ]

A) O(1) B) O(d) but not O(1) C) $O(2^d)$ but not O(d) D) $O(d\ 2^d)$ but not $O(2^d)$

13. A complete binary min-heap is made by including each integer in [1, 1023] exactly once. The depth of a node in the heap is the length of the path from the root of the heap to that node. Thus, the root is at depth 0. **(GATE 2016)** **[ ]**

A) 6 B) 7 C) 8 D) 9

14. Which of the following statements is/are TRUE for undirected graphs?

P: Number of odd degree vertices is even.

Q: Sum of degrees of all vertices is even. **(GATE 2013)** **[ ]**

A) P only B) Q only C) Both P and Q D) Neither P nor Q

15. Let G be a simple undirected planar graph on 10 vertices with 15 edges. If G is a connected graph, then the number of bounded faces in any embedding of G on the plane is equal to_____ **(GATE 2012)** **[ ]**

A) 3 B) 4 C) 5 D) 6

16. Which one of the following is TRUE for any simple connected undirected graph with more than 2 vertices? **(GATE 2009)** **[ ]**

A) No two vertices have the same degree. B) At least two vertices have the same degree.

C At least three vertices have the same degree. D All vertices have the same degree.