Batch: B1 **Roll No.: 1711072**

Experiment No. 10

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Priority Queue

Objective: Implementation of priority queue using heap data structure.

Expected Outcome of Experiment:

CO	Outcome
CO3	Use linear and non-linear data structure in domain like compiler construction, DBMS etc.

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:

NODE STRUCTURE:

1. int arr[50] //to store the priorities.
2. int max=50 //maximum length of priority queue.
3. int count=0 //to find number of elements in queue at any given time.

OPERATIONS:

1. **int parent(int i)**

   Used for returning the parent of i^{th} element.

2. **void swap(int *x, int *y)**

   Used for swapping memory addresses of x and y. (Pass by reference)

3. **void insert(int element)**

   Used for inserting an element into the priority queue in its proper place. This method internally performs heap sort.

4. **void display()**

   Used for displaying the array of elements in priority queue.

5. **int left(int i)**

   Used for getting the left child index of element at index i.

6. **int right(int i)**

   Used for getting the right child index of element at index i.

7. **int delete()**

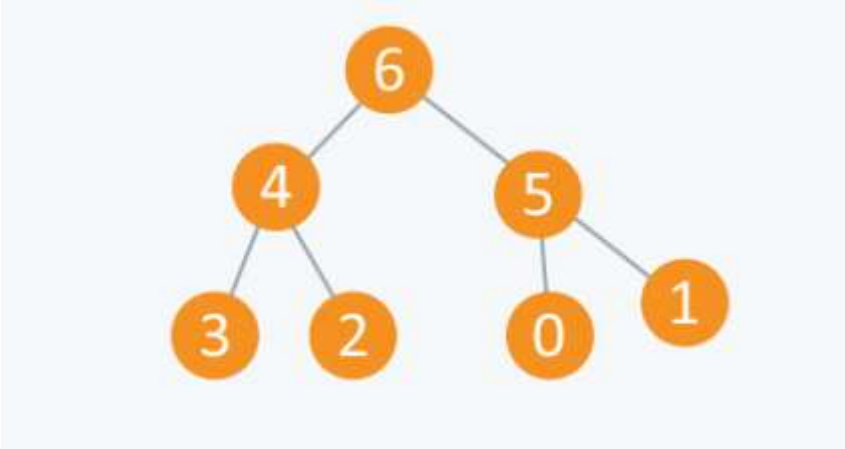
   Used for deleting the element with top priority and rearranging the heap to maintain heap property.

Related Theory: -

Heaps:
A heap is a specific tree based data structure in which all the nodes of tree are in a specific order. Let's say if X is a parent node of Y, then the value of X follows some specific order with respect to value of Y and the same order will be followed across the tree.

The maximum number of children of a node in the heap depends on the type of heap. However in the more commonly used heap type, there are at most 2 children of a note and it's known as a Binary heap.

In binary heap, if the heap is a complete binary tree with N nodes, then it has the smallest possible height which is $\mathbf{\log_2 N}$.

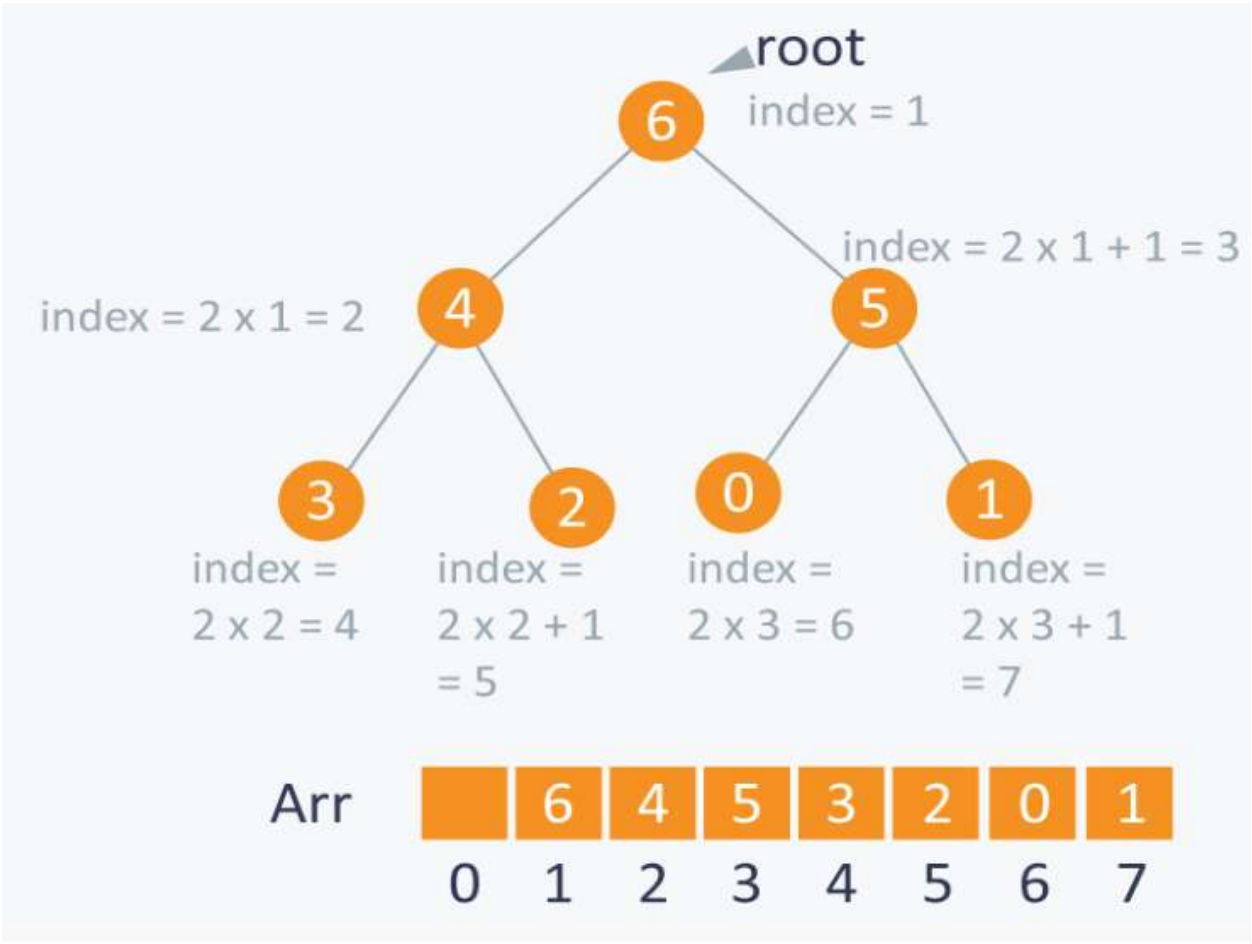


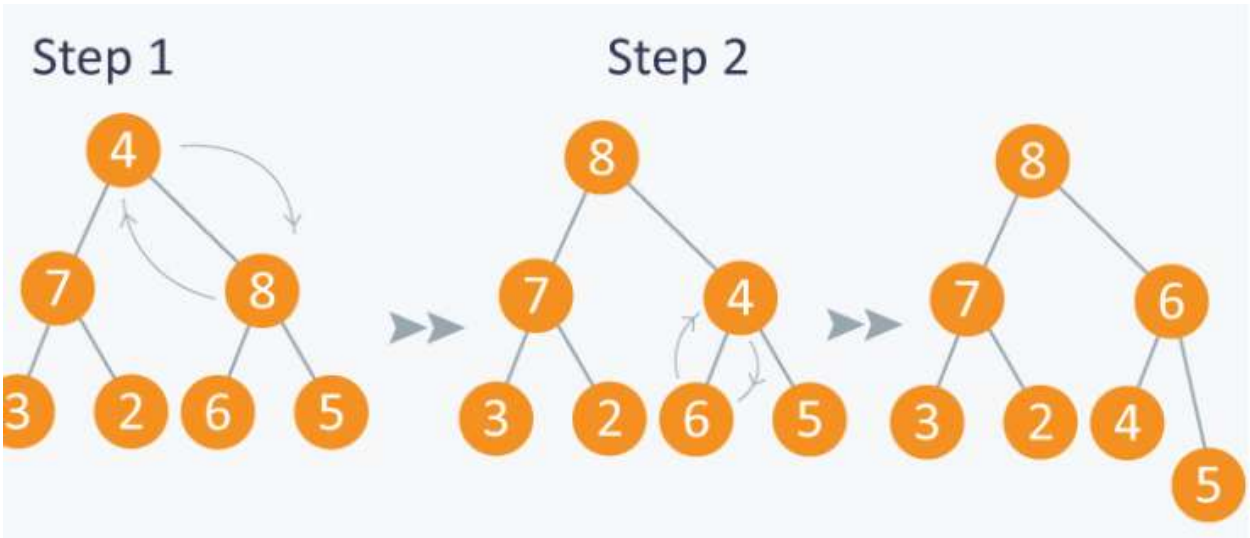
Suppose there are N Jobs in a queue to be done, and each job has its own priority. The job with maximum priority will get completed first than others. At each instant we are completing a job with maximum priority and at the same time we are also interested in inserting a new job in the queue with its own priority.

So at each instant we have to check for the job with maximum priority to complete it and also insert if there is a new job. This task can be very easily executed using a heap by considering N jobs as N nodes of the tree.

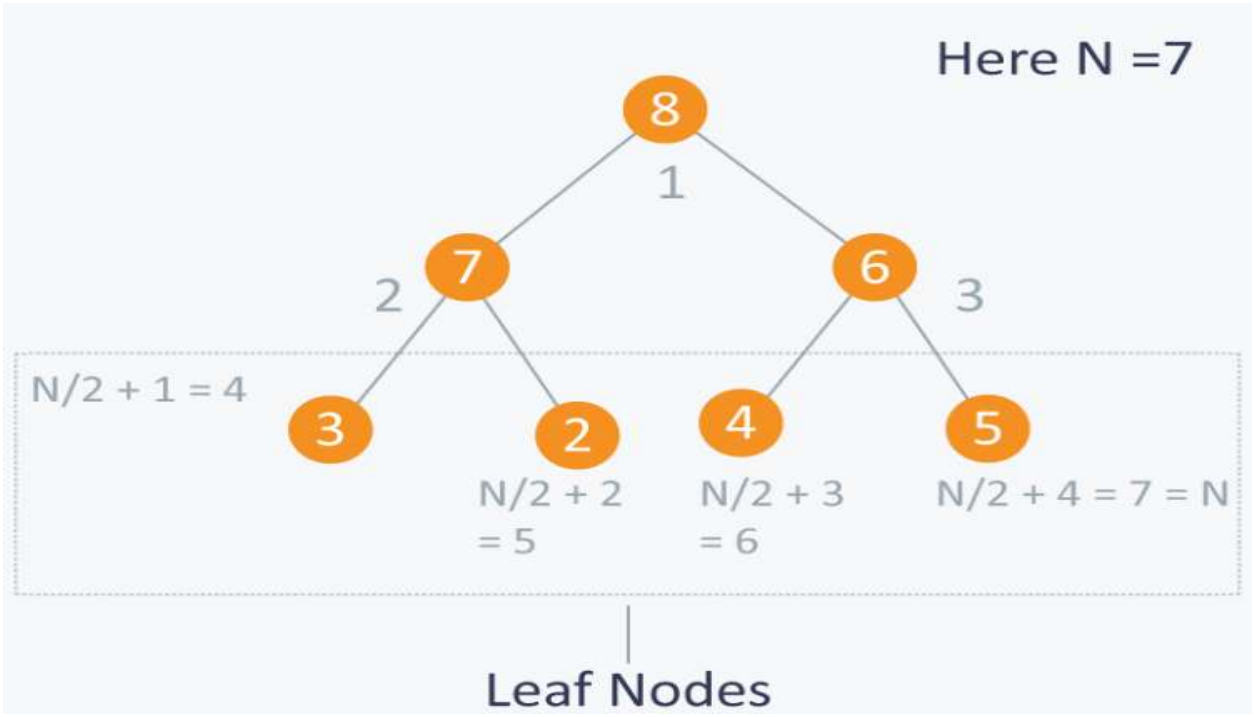
As you can see in the diagram below, we can use an array to store the nodes of the tree. Let's say we have 7 elements with values {6, 4, 5, 3, 2, 0, 1}.

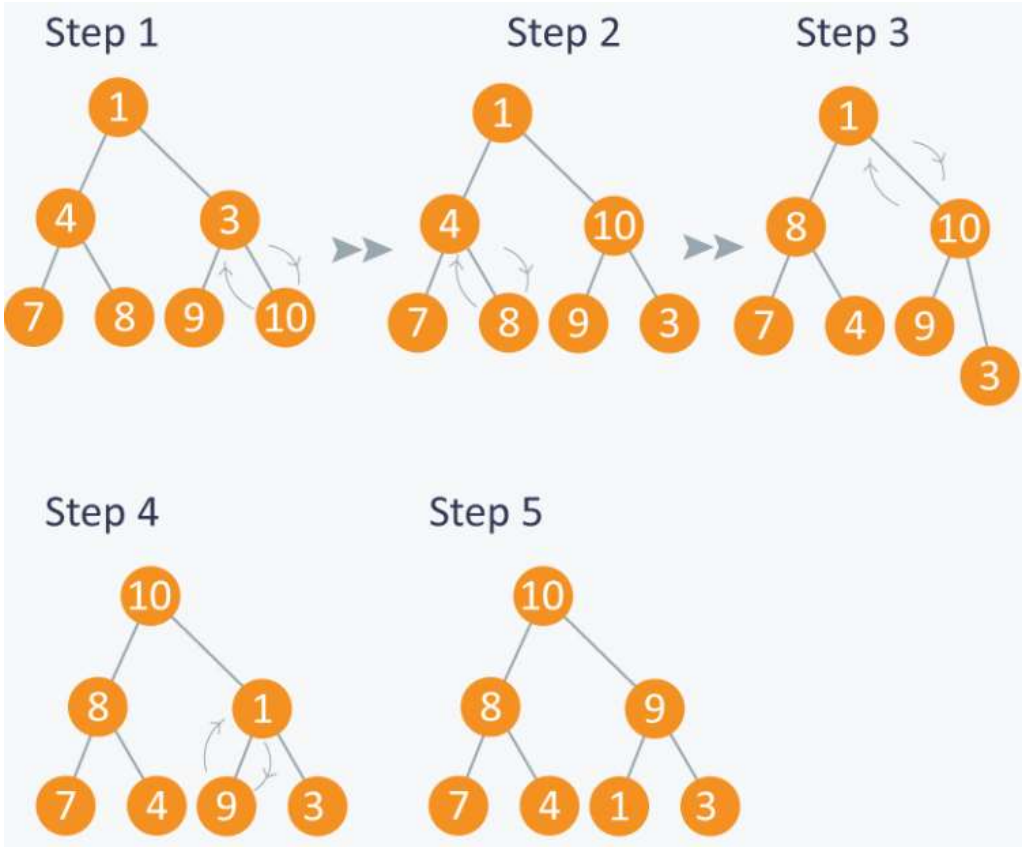
Note: An array can be used to simulate a tree in the following way. If we are storing one element at index i in array Ar, then its parent will be stored at index i/2 (unless it's a root, as root has no parent) and can be access by Ar[i/2], and its left child can be accessed by Ar[2*i] and its right child can be accessed by Ar[2*i+1]. Index of root will be 1 in an array.





As 8 is greater than 4, then 8 is swapped with 4 and max_heapify is performed again on 4, but on different position. Now in step 2, 6 is greater than 4, so 4 is swapped with 6 and we will get a max heap, as now 4 is a leaf node, so further call to max_heapify will not create any effect on heap.





Arr={NULL, 10,8,9,7,4,1,3} This can be verified using heap properties.

Implementation Details:

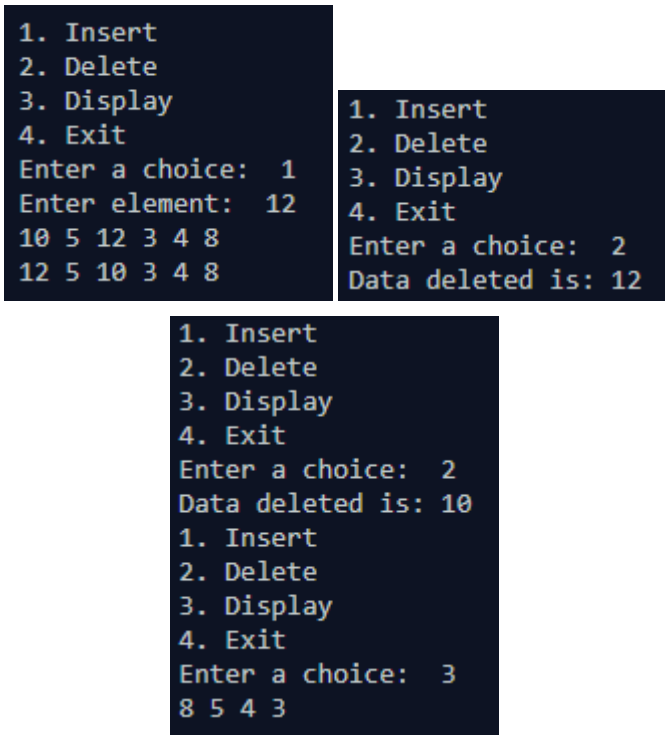
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int arr[50];
int max=50;
int count=0;
int parent(int i)
{
  float p;
  p= (i-1)/2.0;
  return floor(p);
}

void swap(int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}

void insert (int element)
{
  int i = count;
  int p = parent(i);
  if (count == max)
  {
    printf("Sorry but queue is full!\n");
    return;
  }
  arr[i] = element;
  while (i > 0 && arr[p] < arr[i])
  {
    swap(arr+p, arr+i);
    i = p;
    p = parent(i);
    for (int j = 0; j < count+1; j++)
      printf("%d ", arr[j]);
    printf("\n");
  }
  count++;
```

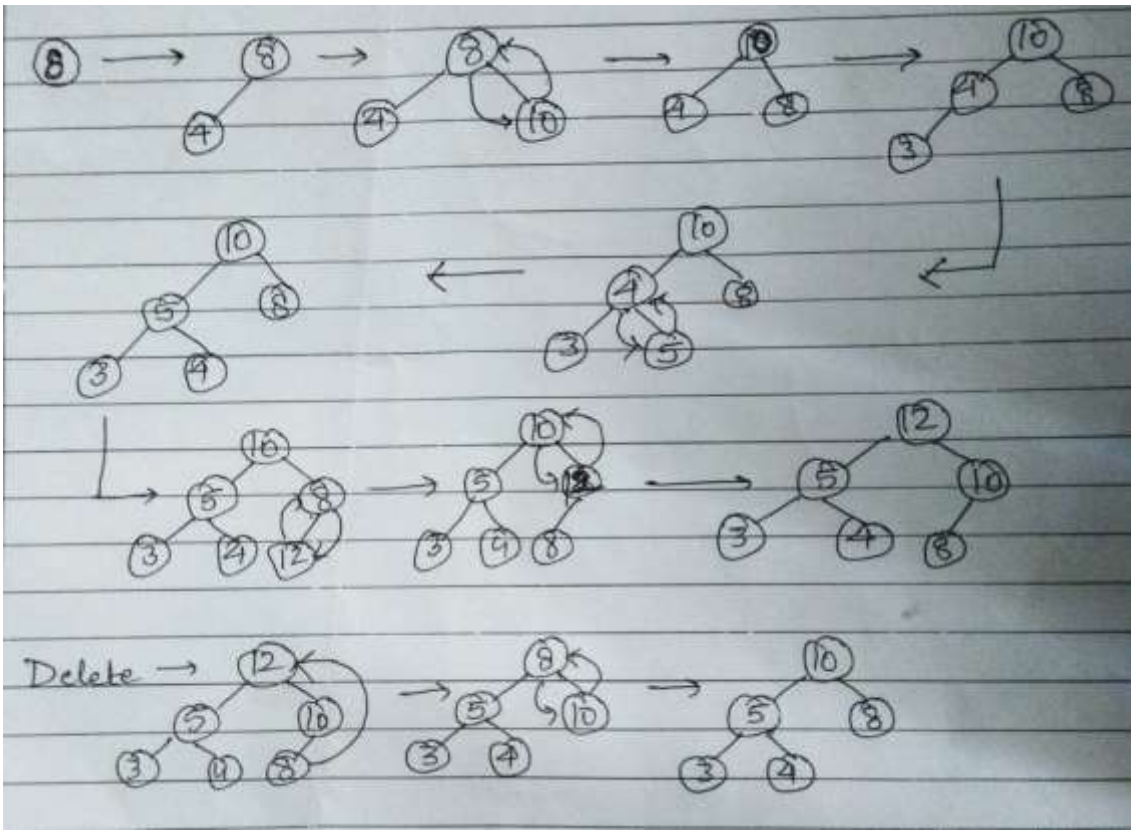
```
}

void display()
{
  for (int i = 0; i < count; i++)
    printf("%d ", arr[i]);
  printf("\n");
}

int left(int j){
  return 2*j+1;
}
int right(int j){
  return 2*(j+1);
}

int delete(){
  int l,r,max,i=0,del=arr[0];
  if(count==0){
    printf("Sorry but queue is empty!\n");
    return -1;
  }
  swap(&arr[0], &arr[--count]);
  while(1){
    if((l=left(i))>=count)
      break;
    if((r=right(i))>=count)
      max=l;
    else
      max=(arr[l]>arr[r])?l:r;
    if(arr[i]>arr[max])
      break;
    swap(&arr[i], &arr[max]);
    i=max;
  }
  return del;
}

int main(void) {
  int choice, element;
  do
  {
    printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter a
choice: ");
```

```
    scanf(" %d", &choice);
    switch(choice)
    {
      case 1: printf("Enter element: ");
              scanf(" %d", &element);
              insert(element);
              break;
      case 2: printf("Data deleted is: %d", delete());
break;
      case 3: display();
              break;
    }
  }while (choice < 4);
  return 0;
}
```

For verification, my code is available on:
https://repl.it/@ARGHYADEEPDAS/PriorityQueueMaxHeap

OUTPUT SCREEN:





(Test Case)

CONCLUSION:

The program ran successfully as we were able to implement menu driven program for priority queue and internally perform heap sort and all test cases handled successfully.