ASSIGNMENT 6

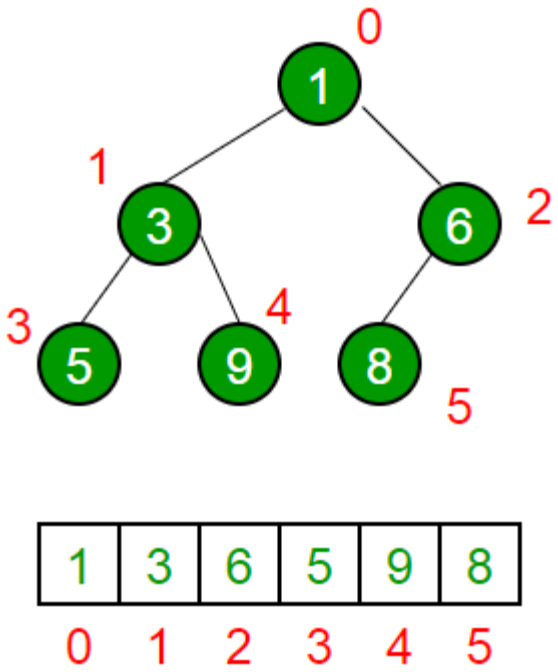
1. **Aim** – Read the marks obtained by the students of second year in an online eaxamination of particular subject.Find out maximum and minimum marks obtained in the subject using heap data structure.

2. **Objective** – To implement max-heap and misn-heap to obtain minimum and maximum marks.

3. Theory –

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

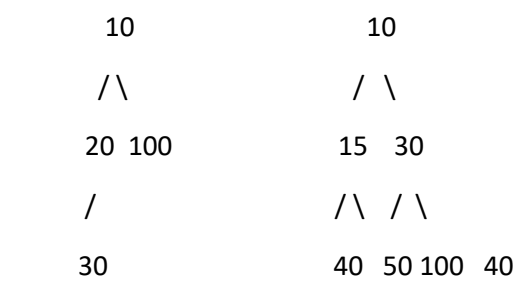
1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



A Binary Heap is a Binary Tree with following properties.
1) It's a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

Examples of Min Heap:



How is Binary Heap represented?

A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.

- The root element will be at Arr[0].
- Below table shows indexes of other nodes for the ith node, i.e., Arr[i]:

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node

Operations on Min Heap:

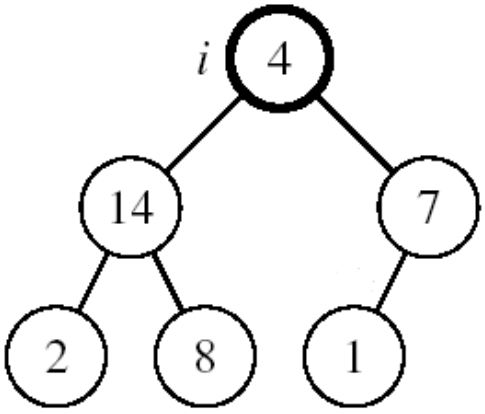
1) getMini(): It returns the root element of Min Heap. Time Complexity of this operation is O(1).
2) extractMin(): Removes the minimum element from MinHeap. Time Complexity of this Operation is O(Logn) as this operation needs to maintain the heap property (by calling heapify()) after removing root.
3) decreaseKey(): Decreases value of key. The time complexity of this operation is O(Logn). If the decreases key value of a node is greater than the parent of the node, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.
4) insert(): Inserting a new key takes O(Logn) time. We add a new key at the end of the tree. IF new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.
5) delete(): Deleting a key also takes O(Logn) time. We replace the key to be deleted with minum infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call extractMin() to remove the key.

3. **Algorithm** –

Assumptions:

a. Left and Right subtrees of i are max-heaps

b. A[i] may be smaller than its children



Alg: MAX-HEAPIFY(A, i, n)

1. l ← LEFT(i)
2. r ← RIGHT(i)
3. **if** l ≤ n and A[l] > A[i]
4. **then** largest ←l
5. **else** largest ←i
6. **if** r ≤ n and A[r] > A[largest]
7. **then** largest ←r
8. **if** largest ≠ i
9. **then** exchange A[i] ↔ A[largest]
10. MAX-HEAPIFY(A, largest, n)

5. **Program code –**

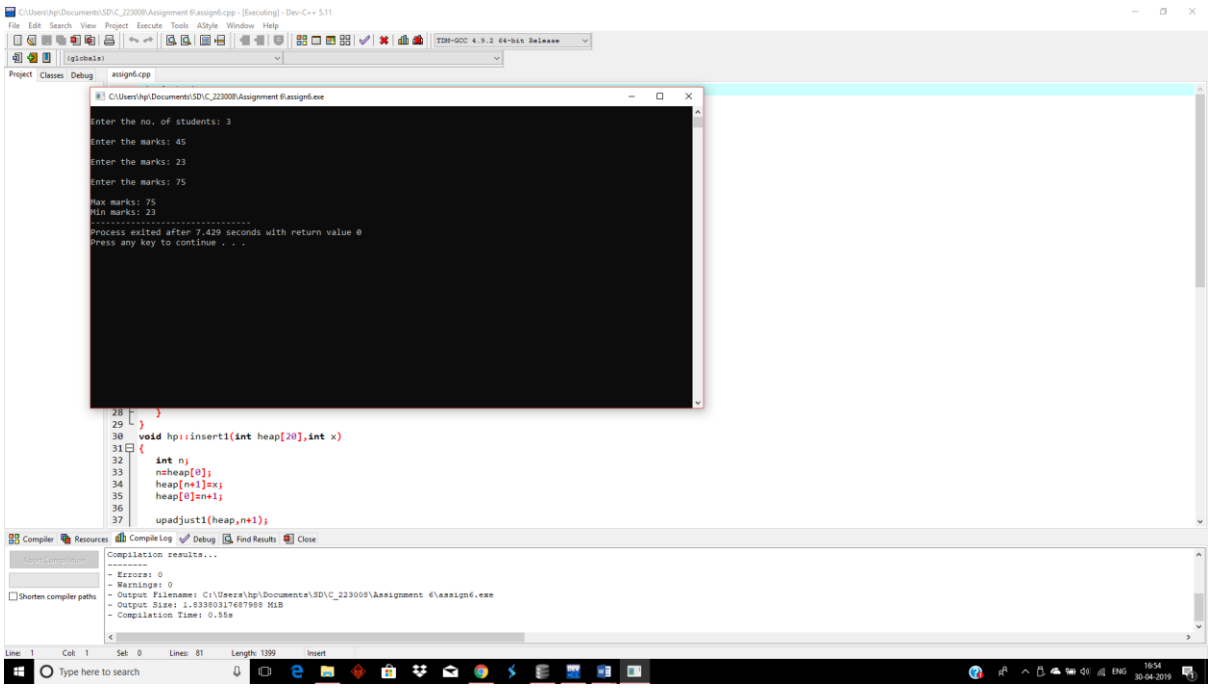
```
#include<iostream>
using namespace std;

class hp
{
  int heap[20],heap1[20],x,n1,i;
  public:
  hp()
  { heap[0]=0;  heap1[0]=0;
  }
  void getdata();
  void insert1(int heap[],int);
  void upadjust1(int heap[],int);
  void insert2(int heap1[],int);
  void upadjust2(int heap1[],int);
  void minmax();
};
void hp::getdata()
{
  cout<<"\nEnter the no. of students: ";
  cin>>n1;
  for(i=0;i<n1;i++)
  {
      cout<<"\nEnter the marks: ";
       cin>>x;
```

```
      insert1(heap,x);
      insert2(heap1,x);
   }
}
void hp::insert1(int heap[20],int x)
{
   int n;
   n=heap[0];
   heap[n+1]=x;
   heap[0]=n+1;

   upadjust1(heap,n+1);
}
void hp::upadjust1(int heap[20],int i)
{
    int temp;
    while(i>1&&heap[i]>heap[i/2])
    {
      temp=heap[i];
      heap[i]=heap[i/2];
      heap[i/2]=temp;
      i=i/2;
    }
}
void hp::insert2(int heap1[20],int x)
{
```

```
   int n;
   n=heap1[0];
   heap1[n+1]=x;
   heap1[0]=n+1;

   upadjust2(heap1,n+1);
}
void hp::upadjust2(int heap1[20],int i)
{
    int temp1;
    while(i>1&&heap1[i]<heap1[i/2])
    {
      temp1=heap1[i];
      heap1[i]=heap1[i/2];
      heap1[i/2]=temp1;
      i=i/2;
    }
}
void hp::minmax()
{
   cout<<"\nMax marks: "<<heap[1];
   cout<<"\nMin marks: "<<heap1[1];
}
int main()
{
  hp h;
```

```
  h.getdata();
  h.minmax();
  return 0;
}
```

6. **Output screen shots/Copy Past From Terminal –**



7. **Conclusion** –

Heap Data Structure is generally taught with Heapsort. Heapsort algorithm has limited uses because Quicksort is better in practice. Nevertheless, the Heap data structure itself is enormously used. Following are some uses other than Heapsort.

Priority Queues: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time. Binomoial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also in O(logn) time which is a O(n) operation in Binary Heap. Heap Implemented priority queues are used in Graph algorithms like Prim's Algorithm and Dijkstra's algorithm.