ASSIGNMENT : 6.

Aim :-

Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject using heap data structure.

Objective:- To study the heap data structure.

Theory:-

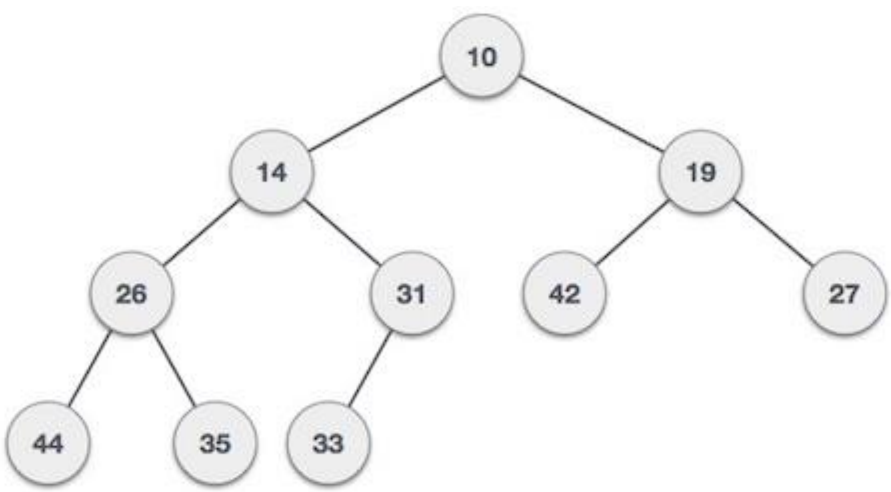
Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then −

$$\mathbf{key(\alpha) \geq key(\beta)}$$

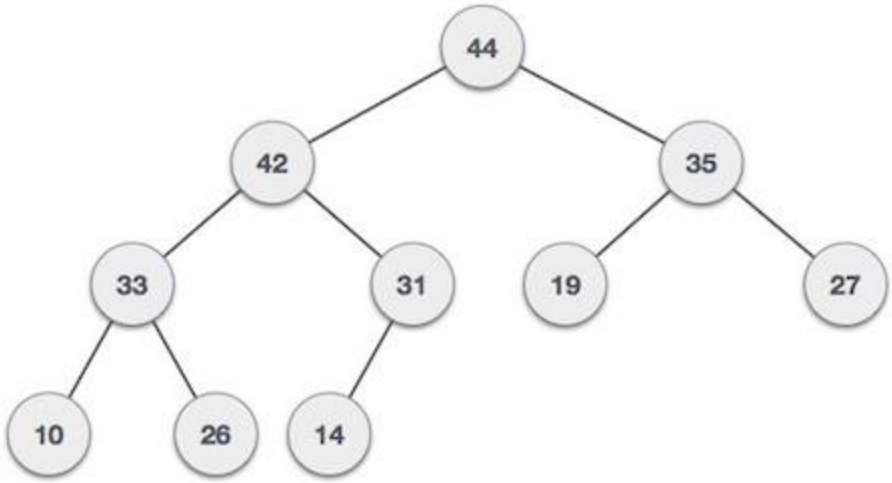
As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types −

For Input → 35 33 42 10 14 19 27 44 26 31

Min-Heap − Where the value of the root node is less than or equal to either of its children.



Max-Heap − Where the value of the root node is greater than or equal to either of its children.



Both trees are constructed using the same input and order of arrival.

Applications:-

1) Heap Sort: Heap Sort uses Binary Heap to sort an array in O(nLogn) time.

2) Priority Queue: 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 efficiently.
3) Graph Algorithms: The priority queues are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.
4) Many problems can be efficiently solved using Heaps. See following for example.
a) K'th Largest Element in an array.
b) Sort an almost sorted array/
c) Merge K Sorted Arrays.

Algorithm:-

1.max heap:-

Step 1 − Create a new node at the end of heap.
Step 2 − Assign new value to the node.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.

Program Code:-

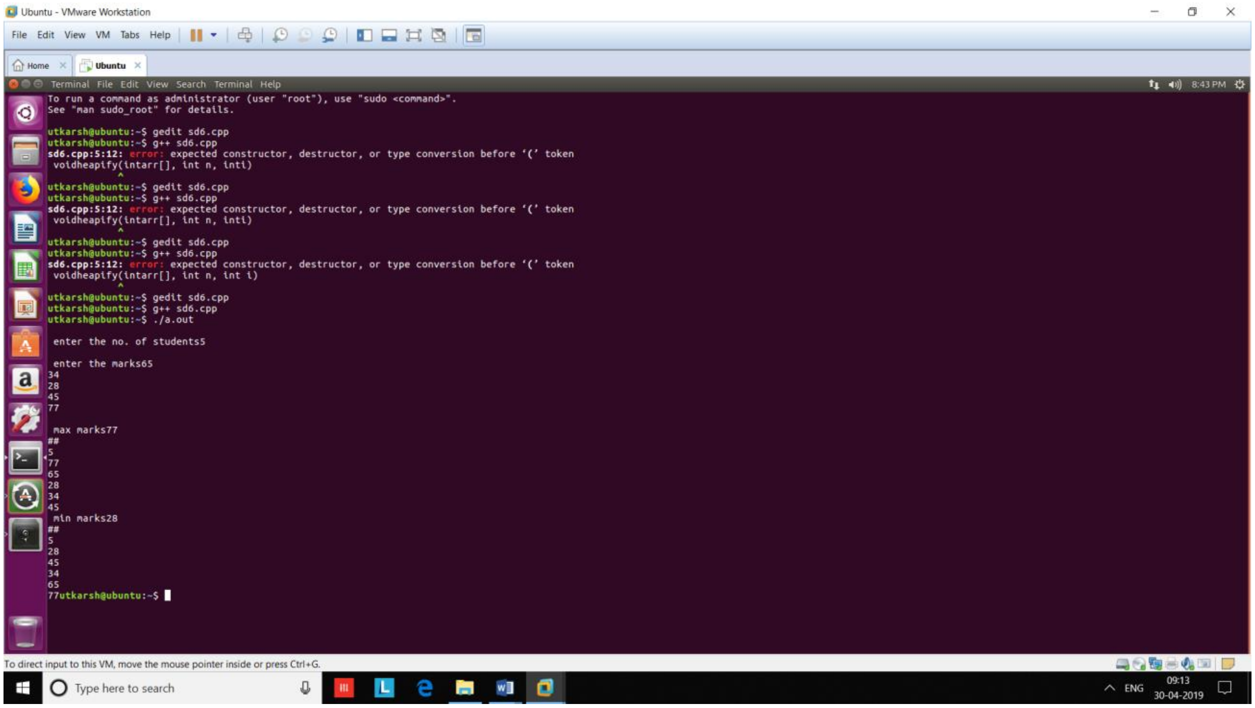
```
#include<iostream>
using namespace std;

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<<"\n enter the no. of students";
   cin>>n1;
   cout<<"\n enter the marks";
   for(i=0;i<n1;i++)
```

```
   {   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<<"\n max marks"<<heap[1];
```

```
  cout<<"\n##";
  for(i=0;i<=n1;i++)
  {   cout<<"\n"<<heap[i];  }
  cout<<"\n min marks"<<heap1[1];
  cout<<"\n##";
  for(i=0;i<=n1;i++)
  {   cout<<"\n"<<heap1[i];  }
}
int main()
{
 hp h;
 h.getdata();
 h.minmax();
 return 0;
}
```

Output Screenshots:-



Conclusion:- Thus, we have studied heap data structure,