BRACT's
VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE – 48
An Autonomous Institute Affiliated to Savitribai Phule Pune University, Pune

SD(LP-II) ASSIGNMENT (S.Y.B. Tech. – DIV: C)

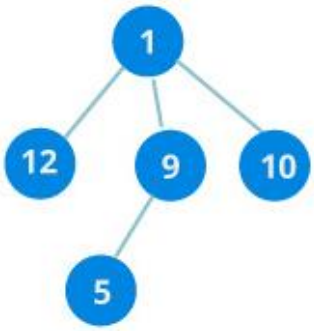
Name: Prajwal Dabhade;

Roll no.: 223012;

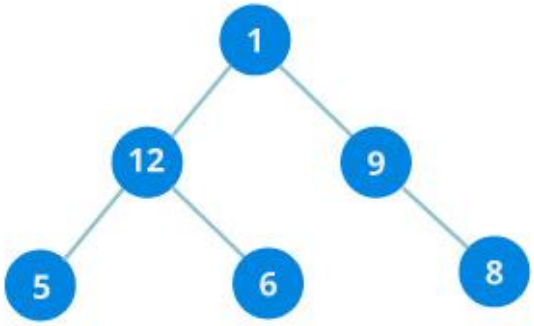
GR. No.: 17u385; Batch: C1

**********Assignment No.6**********

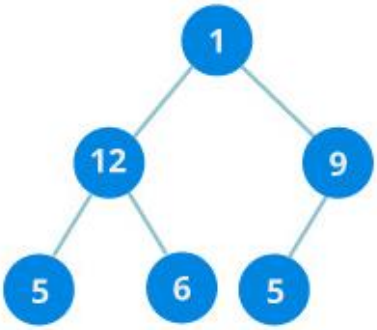
- **Aim:** Reads the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject using head data structure.
- **Objective:** To organise/arrange for finding the max and min marks using heap and its property.
- **Theory:** A heap is a complete binary tree except the bottom level.
  All nodes in the tree follow the property that they are greater than their children i.e. the largest element is at the root and both its children and smaller than the root and so on. Such a heap is called a max-heap. If instead all nodes are smaller than their children, it is called a min-heap. A heap can be used as a priority queue: the highest priority item is at the root and is trivially extracted. But if the root is deleted, we are left with two sub-trees and we must *efficiently* re-create a single tree with the heap property.



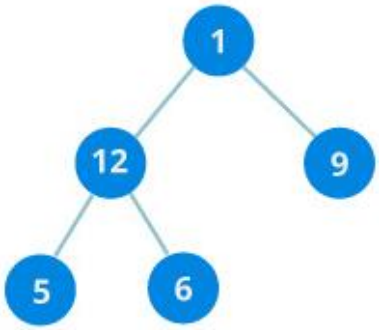
✖ Binary Tree
✖ Full Tree
✖ Complete Tree



✔ Binary Tree
✖ Full Tree
✖ Complete Tree

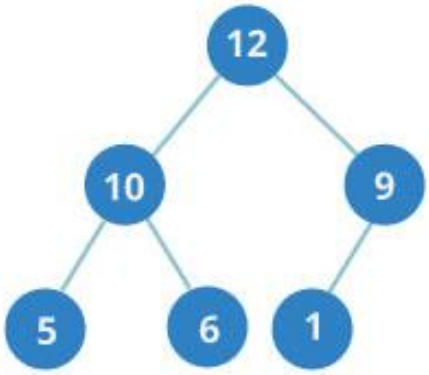


✔ Binary Tree
✖ Full Tree
✔ Complete Tree

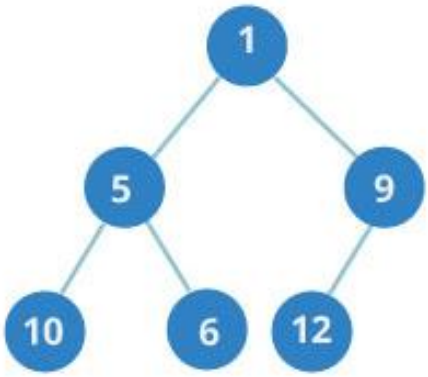


✔ Binary Tree
✔ Full Tree
✔ Complete Tree

Following example diagram shows Max-Heap and Min-Heap:



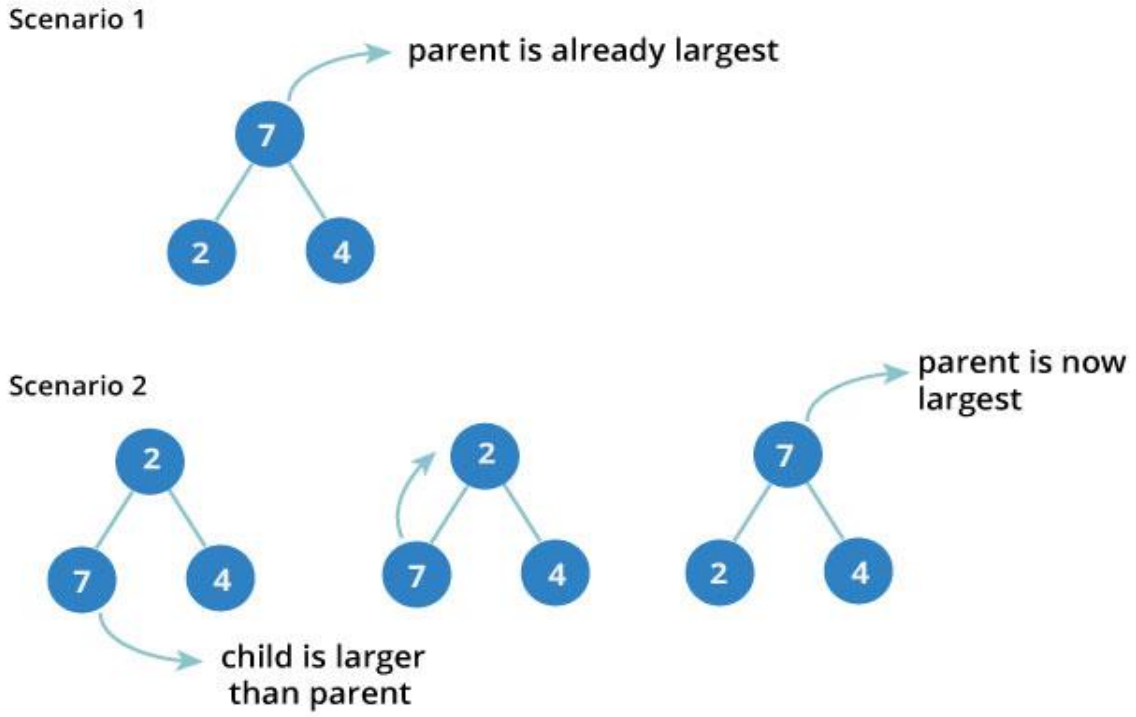
Max Heap



Min Heap

Starting from a complete binary tree, we can modify it to become a Max-Heap by running a function called **heapify** on all the non-leaf elements of the heap.
Since heapfiy uses recursion, it can be difficult to grasp. So let's first think about how you would heapify a tree with just three elements.

```
heapify(array)
   Root = array[0]
   Largest = largest( array[0] , array [2*0 + 1]. array[2*0+2])
   if(Root != Largest)
        Swap(Root, Largest)
```



- **Algorithm:**

1. Max-Heapify(numbers[], i)
2. leftchild := numbers[2i]
3. rightchild := numbers [2i + 1]
4. if leftchild ≤ numbers[].size and numbers[leftchild] > numbers[i]
   largest := leftchild
5. else
   largest := i
6. if rightchild ≤ numbers[].size and numbers[rightchild] > numbers[largest]
   largest := rightchild
7. if largest ≠ i
   swap numbers[i] with numbers[largest]
   Max-Heapify(numbers, largest)

- **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;
}
```

- **Output:**

```
Enter the no. of students: 5

Enter the marks: 30

Enter the marks: 28

Enter the marks: 19

Enter the marks: 22

Enter the marks: 12

Max marks: 30
Min marks: 12
Process returned 0 (0x0)   execution time : 17.663 s
Press any key to continue.
```

- **Conclusion:** We successfully found the marks by implementing heap data structure. The value of the heap structure is that we can both extract the highest priority item and insert a new one in **O(log*n*)** time.