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 Find out maximum and minimum marks obtained in that subject using heap data structure.

Theory:-

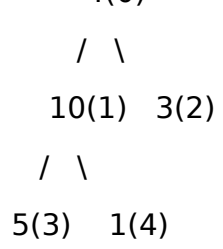
n computer science, a **heap** is a specialized tree-based data structure which is essentially an almost complete[1] tree that satisfies the *heap property:* in a *max heap*, for any given node C, if P is a parent node of C, then the *key* (the *value*) of P is greater than or equal to the key of C. In a *min heap*, the key of P is less than or equal to the key of C.[2] The node at the "top" of the heap (with no parents) is called the *root* node.

The heap is one maximally efficient implementation of an abstract data type called a priority queue, and in fact priority queues are often referred to as "heaps", regardless of how they may be implemented. In a heap, the highest (or lowest) priority element is always stored at the root. However, a heap is not a sorted structure; it can be regarded as being partially ordered. A heap is a useful data structure when it is necessary to repeatedly remove the object with the highest (or lowest) priority.

A common implementation of a heap is the binary heap, in which the tree is a binary tree (see figure). The heap data structure, specifically the binary heap, was introduced by J. W. J. Williams in 1964, as a data structure for the heapsort sorting algorithm.[3] Heaps are also crucial in several efficient graph algorithms such as Dijkstra's algorithm. When a heap is a complete binary tree, it has a smallest possible height—a heap with N nodes and for each node a branches always has $\log_a N$height.

Algorithm:-

Input data: 4, 10, 3, 5, 1



The numbers in bracket represent the indices in the array

representation of data.

Applying heapify procedure to index 1:

```
        4(0)
       /   \
    10(1)    3(2)
   /   \
5(3)    1(4)
```

Applying heapify procedure to index 0:

```
       10(0)
       /  \
     5(1)  3(2)
   /   \
 4(3)    1(4)
```

The heapify procedure calls itself recursively to build heap in top down manner.

Code:-

```
#include <iostream>

using namespace std;

class Heap{
private :
    int * max_heap;
    int * min_heap;
    int capacity, currEmptyPos;
public :
    Heap(int capacity){
        currEmptyPos = 0;
        this->capacity = capacity;
        max_heap = new int[capacity];
        min_heap = new int[capacity];
```

```
    for(int i=0; i<capacity; i++){
        max_heap[i] = 0;
        min_heap[i] =9999;
    }
}
void max_heapify(int pos){

    cout<<"called."<<pos<<endl;
    int left = (pos*2)+1;
    int right = (pos*2)+2;
    int largest = pos;
    if( pos < ((currEmptyPos+1)/2)){
        if(max_heap[left]>max_heap[largest]){
        largest = left;
        }
        if(max_heap[right]>max_heap[largest]){
            largest = right;
        }
    }
    if(largest == pos){
        return;
    }
    else{
        int temp = max_heap[largest];
        max_heap[largest] = max_heap[pos];
        max_heap[pos] = temp;
        max_heapify((pos-1)/2);
    }
}
```

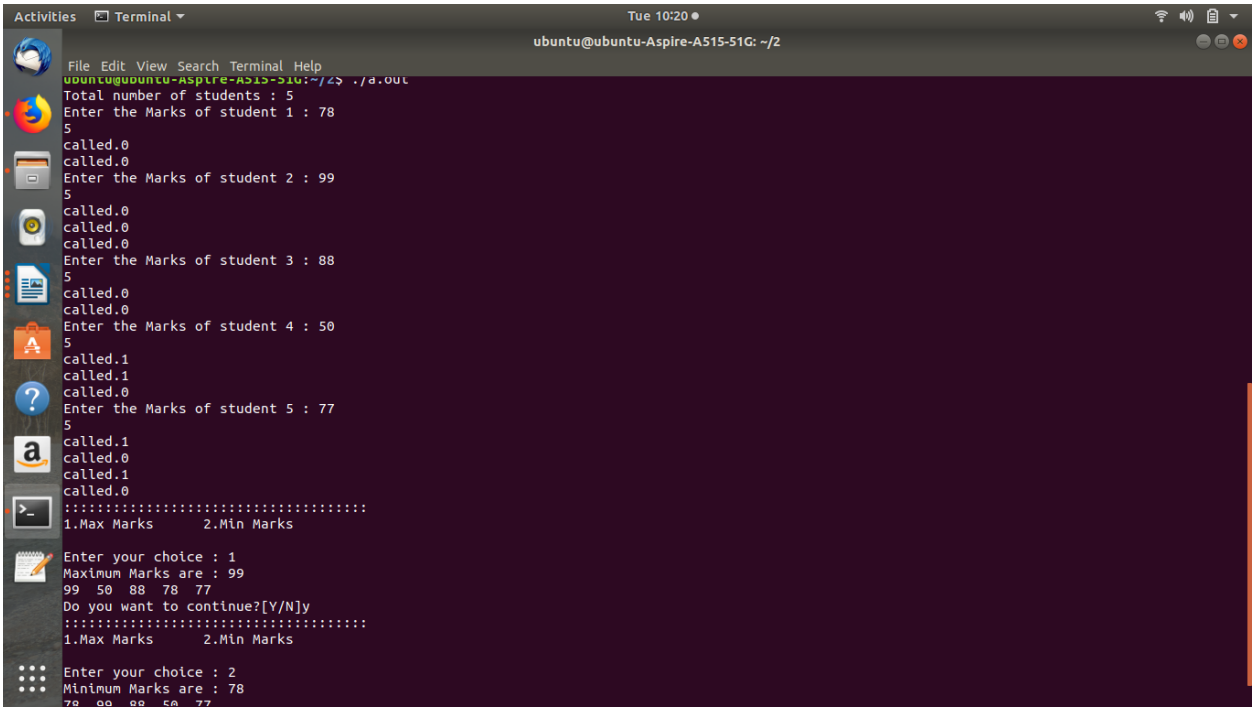
```
void min_heapify(int pos){

    cout<<"called."<<pos<<endl;
    int left = (pos*2)+1;
    int right = (pos*2)+2;
    int largest = pos;
    if( pos < ((currEmptyPos+1)/2)){
        if(min_heap[left]<min_heap[largest]){
            largest = left;
        }
        if(min_heap[right]<min_heap[largest]){
            largest = right;
        }
    }
    if(largest == pos){
        return;
    }
    else{
        int temp = max_heap[largest];
        max_heap[largest] = max_heap[pos];
        max_heap[pos] = temp;
        max_heapify((pos-1)/2);
    }
}

void insertIntoHeap(int val){
    cout<<capacity<<endl;
    if(currEmptyPos< capacity){
        max_heap[currEmptyPos] = val;
        max_heapify((currEmptyPos-1)/2);
```

```
            min_heap[currEmptyPos] = val;
            min_heapify((currEmptyPos-1)/2);
            currEmptyPos++;
        }
    }

    int getMaxTop(){return max_heap[0];}
    int getMinTop(){return min_heap[0];}

    void print_max_heap(){
        for(int i = 0;i<capacity;i++){
            cout<<max_heap[i]<<"  ";
        }
        cout<<endl;
    }
    void print_min_heap(){
        for(int i = 0;i<capacity;i++){
            cout<<min_heap[i]<<"  ";
        }
        cout<<endl;
    }
};

void setStudentsMarks(){
    cout<<"Total number of students : ";
    int noOfStu;
    cin>>noOfStu;
    Heap stuMarks(noOfStu);
    for(int i = 0; i < noOfStu ; i++){
        cout<<"Enter the Marks of student "<<i+1<<" : ";
```

```
        int marks;
        cin>>marks;
        stuMarks.insertIntoHeap(marks);
    }
    char ch;
    do{
        cout<<"::::::::::::::::::::::::::::::::::::"<<endl;
        cout<<"1.Max Marks"<<"      "<<"2.Min Marks"<<endl;
        cout<<endl<<"Enter your choice : ";
        int choice;
        cin>>choice;
        if(choice == 1){
            cout<<"Maximum Marks are : "<<stuMarks.getMaxTop()<<endl;
            stuMarks.print_max_heap();

        }
        else{
            cout<<"Minimum Marks are : "<<stuMarks.getMinTop()<<endl;
            stuMarks.print_min_heap();
        }


        cout<<"Do you want to continue?[Y/N]";
        cin>>ch;
    }while(ch=='y' || ch=='Y');

}

int main()
{
```

```
    setStudentsMarks();
    return 0;
}
```

Output Screen Shots:-



Conclusion:-

We Conclude That We can Find The Maximum And Minimum Marks Stored By A Student In Particular Test Can be Found Out By Using Heap Datastructure.