HEAP
DATA
STRUCTURE

Content

7.1. Heap Sort

```
void buildHeap(int arr[], int n)
{
      //to build heap
        for(int i=n/2-1;i>=0;i--)
                heapify(arr,n,i);

        //for sortint
        for(int i=n-1;i>=0;i--){
                int temp = arr[i];
                arr[i] = arr[0];
                arr[0] = temp;
                heapify(arr,i,0);
        }
}

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
        int largest = i;
        int l = 2*i+1;
        int r = 2*i+2;

        if(l<n && arr[l]>arr[largest]){
                largest = l;
        }
        if(r<n && arr[r]>arr[largest]){
                largest = r;
        }
        if(i!=largest){
                int temp = arr[largest];
                arr[largest] = arr[i];
                arr[i] = temp;
                heapify(arr,n,largest);
        }
}
```

7.2 Binary Heap Operations

```
int extractMin()
{
        int m = -1;
        if(heap_size>0){
                m = arr[0];
                        if(heap_size==1){
                                heap_size=0;
                        }else{
                                arr[0] = arr[heap_size-1];
                                heap_size--;
                                MinHeapify(0);
                        }
        }
        return m;
}

void insertKey(int k)
{
        arr[heap_size] = k;
        heap_size++;
        if(heap_size>1){
                int i = heap_size-1;
                int p = (i-1)/2;
                while(p>=0 && arr[p]>arr[i]){
                        int temp = arr[p];
                        arr[p] = arr[i];
                        arr[i] = temp;
                        i = p;
                        p = (i-1)/2;
                }
        }
}
void deleteKey(int i)
{
        if(i<heap_size){
                if(heap_size==1 && i==0){
                        heap_size=0;
                }else{
                        int temp = arr[i];
```

```
            arr[i] = arr[heap_size-1];
            arr[heap_size-1] = temp;
            heap_size--;
            MinHeapify(i);
        }
    }
}
```

7.3. Rearrange characters

```
class GFG {
    class KeyComparator implements Comparator<Key>{
        //override
        public int compare(Key k1,Key k2){
            if(k1.freq==k2.freq)
                return 0;
            else if(k1.freq<k2.freq)
                return 1;
            return -1;
        }
    }
    class Key{
        int freq;
        char c;
        Key(int freq,char c){
            this.freq = freq;
            this.c = c;
        }
    }
    int rearrange(String str){
        int n = str.length();
        int count[] = new int[26];

        for(int i = 0 ; i < n ; i ++){
            char c = str.charAt(i);
            int val = c - 'a';
            count[val]++;
        }

        PriorityQueue<Key> pq = new PriorityQueue<>(new
                                                KeyComparator());

        for ( char c = ' a ' ; c < = ' z ' ; c + +){
            int val = c - 'a';
            if(count[val]>0){
                pq.add(new Key(count[val],c));
            }
        }
        Key prev = new Key(-1,'#');
        int strCount = 0;

        while ( ! pq.isEmpty ( ) ) {
            Key k = pq.peek();
            pq.poll();
            strCount++;

            if(prev.freq>0){
                pq.add(prev);
            }

            (k.freq)--;
            prev = k;
        }

        if(n==strCount)
            return 1;

        return 0;
    }
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        GFG gfg = new GFG();
        for(int i=0;i<test;i++){
            String str = sc.next();
            System.out.println(gfg.rearrange(str));
        }
    }
}
```

7.4. Find median in a stream

Statement : Given an input stream of **N** integers. The task is to insert these numbers into a new stream and find the median of the stream formed by each insertion of **X** to the new stream.

```
class MinHeap{
    int size;
    int arr[];
    MinHeap(int n){
        size = 0;
        arr = new int[n];
    }
    void insert(int e){
        arr[size] = e;
        size++;
        if(size>1){
            int i = size-1;
            int p = (i-1)/2;
            while(p>=0 && arr[p]>arr[i]){
                int temp = arr[p];
                arr[p] = arr[i];
                arr[i] = temp;
                i = p;
                p = (i-1)/2;
            }
        }
    }
    int getCount(){
        return size;
    }
    int getTop(){
        if(size>0){
            return arr[0];
        }
        return -1;
    }
    int extractTop(){
        int m = -1;
        if(size>0){
            m = arr[0];
            if(size==1){
                size=0;
            }else{
                int temp = arr[0];
                arr[0] = arr[size-1];
                arr[size-1] = temp;
                size--;
                heapify(0);
            }
        }
        return m;
    }
    void heapify(int i){
        int l = (2*i)+1;
        int r = (2*i)+2;
        int smallest = i;
        if(l<size && arr[l]<arr[smallest]){
            smallest = l;
        }
        if(r<size && arr[r]<arr[smallest]){
            smallest = r;
        }
        if(i!=smallest){
            int temp = arr[i];
            arr[i] = arr[smallest];
            arr[smallest] = temp;
            heapify(smallest);
        }
    }
}
class MaxHeap{
    int size;
    int arr[];
    MaxHeap(int n){
        size = 0;
        arr = new int[n];
    }
    void insert(int e){
        arr[size] = e;
        size++;
        if(size>1){
            int i = size-1;
        int p = (i-1)/2;
        while(p>=0 && arr[p]<arr[i]){
            int temp = arr[p];
            arr[p] = arr[i];
            arr[i] = temp;
            i = p;
            p = (i-1)/2;
        }
    }
}
int getCount(){
    return size;
}
int getTop(){
    if(size>0){
        return arr[0];
    }
    return -1;
}
int extractTop(){
    int m = -1;
    if(size>0){
        m = arr[0];
        if(size==1){
            size=0;
        }else{
            int temp = arr[0];
            arr[0] = arr[size-1];
            arr[size-1] = temp;
            size--;
            heapify(0);
        }
    }
    return m;
}
void heapify(int i){
    int l = (2*i)+1;
    int r = (2*i)+2;
    int largest = i;
    if(l<size && arr[l]>arr[largest]){
        largest = l;
    }
```

```
        if(r<size && arr[r]>arr[largest]){
            largest = r;
        }
        if(i!=largest){
            int temp = arr[i];
            arr[i] = arr[largest];
            arr[largest] = temp;
            heapify(largest);
        }
    }
}
class GFG {
    int signum(int a,int b){
        if(a==b)
            return 0;
        return (a<b)?1:-1;
    }
    public int median(int e,int m,MaxHeap left,MinHeap right){
        int a = left.getCount();
        int b = right.getCount();
        int sig = signum(a,b);
        switch(sig){
            case 1://right heap has more no of element
                if(e<m){
                    left.insert(e);
                }else{
                    int t = right.extractTop();
                    left.insert(t);
                    right.insert(e);
                }
                m = (left.getTop()+right.getTop())/2;
                break;
            case 0://both has same size
                if(e<m){
                    left.insert(e);
                        m = left.getTop();
                }else{
                    right.insert(e);
                    m = right.getTop();
                }
                break;
            case -1://left heap has more no of element
```

```
                if(e<m){
                    int t = left.extractTop();
                    right.insert(t);
                    left.insert(e);
                }else{
                    right.insert(e);
                }
                m = (left.getTop()+right.getTop())/2;
                break;
        }
        return m;
    }
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        MaxHeap left = new MaxHeap(n);
        MinHeap right = new MinHeap(n);
        GFG gfg = new GFG();
        int m = 0;
        for(int i=0;i<n;i++){
            int e = sc.nextInt();
            m = gfg.median(e,m,left,right);
            System.out.println(m);
        }
    }
}
```

7.5. Kth largest element in a stream

Statement : Given an input stream of **n** integers, find the kth largest element for each element in the stream.

Input : 4 6 **output** : -1 -1 -1 1 2 3
1 2 3 4 5 6

```
class MinHeap{
    int arr[];
    int size;
    public MinHeap(int k){
        size = k;
        arr = new int[k];
    }
    void buildHeap(){
        int i = (size-1)/2;
        while( i > = 0 ){
            heapify(i);
            i--;
        }
    }
    void replaceWithRoot(int n){
        arr[0] = n;
        heapify(0);
    }
    void heapify(int i){
        int l = i*2+1;
        int r = i*2+2;
        int smallest = i;
        if(l<size && arr[l]<arr[smallest]){
            smallest = l;
        }
        if(r<size && arr[r]<arr[smallest]){
            smallest = r;
        }
        if(i!=smallest){
            int temp = arr[i];
            arr[i] = arr[smallest];
            arr[smallest] = temp;
            heapify(smallest);
        }
    }
    int min(){
        if(size>0){
            return arr[0];
        }
        return -1;
    }
}
class GFG {
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        GFG gfg = new GFG();
        for(int i=0;i<test;i++){
            int k = sc.nextInt();
            int n = sc.nextInt();
            MinHeap minHeap = new MinHeap(k);
            for(int j=0;j<n;j++){
                int m = sc.nextInt();
                int res = -1;
                if(j<k-1){
                    minHeap.arr[j] = m;
                }else{
                    if(j==k-1){
                        minHeap.arr[j] = m;
                        minHeap.buildHeap();
                    }else{
                        if(m>minHeap.min()){
                            minHeap.replaceWithRoot(m);
                        }
                    }
                    res = minHeap.min();
                }
                System.out.print(res+" ");
            }
            System.out.println();
        }
    }
}
```

7.6. Merge k Sorted Arrays

```
class GfG
{
    class Key{
        int data;
        int row,column;
        public Key(int data,int row,int column){
            this.data = data;
            this.row = row;
            this.column = column;
        }
    }
    void minHeapify(Key arr[],int size,int i){
        int l = (2*i)+1;
        int r = (2*i)+2;
        int smallest = i;
        if(l<size && arr[l].data<arr[smallest].data){
            smallest = l;
        }
        if(r<size && arr[r].data<arr[smallest].data){
            smallest = r;
        }
        if(smallest!=i){
            Key temp = arr[i];
            arr[i] = arr[smallest];
            arr[smallest] = temp;
            minHeapify(arr,size,smallest);
        }
    }
    void replaceRoot(Key arr[],int k,Key s){
        arr[0] = s;
        minHeapify(arr,k,0);
    }
    void buildHeap(Key arr[],int size){
        for(int i=(size-1)/2;i>=0;i--){
            minHeapify(arr,size,i);
        }
    }
```

```
        public ArrayList<Integer> mergeKArrays(int[][] arrays,int k){

                ArrayList<Integer> list = new ArrayList<>();

                //minHeap
                Key heap[] = new Key[k];
                int n = arrays[0].length;
                for(int i=0;i<k;i++){
                        heap[i] = new Key(arrays[i][0],i,1);
                }

                buildHeap(heap,k);

                for(int i=0;i<n*k;i++){
                        Key kk = heap[0];
                        list.add(kk.data);
                        int row = kk.row;
                        int column = kk.column;
                        Key s;
                        if(column<n){
                                s = new Key(arrays[row][column],row,column+1);
                        }else{
                                s = new Key(Integer.MAX_VALUE,row,column);
                        }
                        replaceRoot(heap,k,s);
                }
                return list;
        }
}
```

7.7. Merge K sorted linked list

```
class Merge
{
    Node mergeTwoList(Node a,Node b){
        Node result = null;
        if(a==null)
            return b;
        else if(b==null)
            return a;
        if(a.data<=b.data){
            result = a;
            result.next = mergeTwoList(a.next,b);
        }else{
            result = b;
            result.next = mergeTwoList(a,b.next);
        }
        return result;
    }
    Node mergeKList(Node[]a,int N)
    {
        Node head = a[0];
        for(int i=1;i<a.length;i++){
            Node temp = a[i];
            head = mergeTwoList(head,temp);
        }
        return head;
    }
}
```