```
#include<bits/stdc++.h>
using namespace std;

class heap{
   public:
    int arr[100];
    int size;
    heap(){
       arr[0]=-1;
       size=0;
    }

    void insert(int val){
       size=size+1;
       int index=size;
       arr[index]=val;

       while(index>1){
          int parent= index/2;
          if(arr[parent]< arr[index]){
             swap(arr[parent],arr[index]);
             index=parent;
          }
          else{
             return;
          }
       }
    }


    void print(){
       for(int i=1;i<=size;i++){
          cout<<arr[i]<<" ";
       }
       cout<<endl;
    }


    void deleteFromHeap(){
       if(size==0){
          cout<<"nothing to delete : "<<endl;
          return;
       }
       // last node k uthiya 1st node a boshate hoba, size decrement korte hoba
       arr[1]=arr[size];
       //remove lasrt element
       size--;
```

```
        //root node k tar correct position a boshate hoba
        int i=1;
        while(i<size){
            int leftIndex=2*i;
            int rightIndex=2*i +1;
            if(leftIndex<size && arr[i]<arr[leftIndex]){  // root ar man left value ar chaya choto tai swap korte hoba
               swap(arr[i],arr[leftIndex]);
               i=leftIndex;

            }
            else if(rightIndex< size && arr[i]<arr[rightIndex]){
                swap(arr[i],arr[rightIndex]);
                i=rightIndex;
            }
            else{
                return;
            }
        }

     }
};

void heapify(int arr[],int n,int indx){
    int largest=indx;
    int left=2*indx;
    int right=2*indx +1;
    // for 0 base left< n
    if(left<=n && arr[largest]<arr[left]){
        largest=left;
    }
    if(right<= n && arr[largest]<arr[right]){
        largest=right;
    }

    // if leargest is change thn swap
    if(largest !=indx){
        swap(arr[largest],arr[indx]);
        // ai process toh choltai thkbe tai recursion call korte hoba
        // largest k tar thik jaygay pouchay daw
        heapify(arr,n,largest);
    }
}

// min heap
void heapifyMin(int arr[],int n,int indx){
    int smallest=indx;
    // for 0 base indexing
    // int left=2*indx+1;
    // int right=2*indx +2;
       int left=2*indx;
    int right=2*indx +1;

    if(left<=n && arr[smallest]>arr[left]){
        smallest=left;
    }
    if(right<=n && arr[smallest]>arr[right]){
        smallest=right;
    }

    // if leargest is change thn swap
    if(smallest !=indx){
        swap(arr[smallest],arr[indx]);
        // ai process toh choltai thkbe tai recursion call korte hoba
        // largest k tar thik jaygay pouchay daw
        heapify(arr,n,smallest);
    }
}

void heapSort(int arr[],int n){
    int size=n;
    while(size>1){
        //step 1 :  swap 1 st and last value
        swap(arr[size],arr[1]);
        size--;
        //step 2
        heapify(arr,size,1);
    }
}



int main(){

   heap h;
     h.insert(50);
     h.insert(55);
     h.insert(53);
     h.insert(52);
     h.insert(54);

     h.print();
     h.deleteFromHeap();
     h.print();
```

```
// max heap creation
  int arr[6]={-1,54,53,55,52,50};
  int n=5;
  for(int i=n/2; i>0;i--){
     heapify(arr,n,i);
  }

  cout<<"Printing the max heap : "<<endl;
  for(int i=1;i<=n;i++){
     cout<<arr[i]<<" ";
  }cout<<endl;

 // // have some issue
 // int arr1[5]={54,53,55,52,50};
 //  n=5;
 //  for(int i=n/2; i>= 0;i--){
 //     heapifyMin(arr1,n,i);
 //  }

 //  cout<<"Printing the min heap: "<<endl;
 //  for(int i=1;i<=n;i++){
 //     cout<<arr1[i]<<" ";
 //  }cout<<endl;

 // heap sort
 heapSort(arr,n);
 cout<<"Printing the heap sort : "<<endl;
  for(int i=1;i<=n;i++){
     cout<<arr[i]<<" ";
  }cout<<endl;



  // STL in heap


  cout<<"using prayority queue here "<<endl;

  //max heap
  priority_queue<int> pq;
  pq.push(4);
  pq.push(2);
  pq.push(5);
  pq.push(3);

  cout<<"top element of max heap is : "<<pq.top()<<endl;
  pq.pop();
  cout<<" top element of max heap is  : "<<pq.top()<<endl;
```

```
    cout<<"Size is "<<pq.size()<<endl;
    if(pq.empty()){
      cout<<"it's empty "<<endl;
    }
    else{
      cout<<"It's not empty "<<endl;
    }



    //min heap

    priority_queue<int ,vector<int>,greater<int>> minHeap;

    minHeap.push(4);
    minHeap.push(2);
    minHeap.push(5);
    minHeap.push(3);

    cout<<"top element of min heap is : "<<minHeap.top()<<endl;
    minHeap.pop();
    cout<<"top element of min heap is  : "<<minHeap.top()<<endl;
    cout<<"Size is "<<minHeap.size()<<endl;
    if(minHeap.empty()){
      cout<<"it's empty "<<endl;
    }
    else{
      cout<<"It's not empty "<<endl;
    }

    return 0;
}
```

L2:

```
#include<bits/stdc++.h>
using namespace std;
// Q1. kth smallest element    gfg
// where l startign index ,r ending index,k the kth element
int kthSmallest(int arr[],int l,int r,int k){
    priority_queue<int> pq;
    // step 1 : make priority  q consistin  first k element
    for(int i=0;i<k;i++){
      pq.push(arr[i]);
    }
   // step 2
   for(int i=k;i<=r;i++){
        if(arr[i]<pq.top()){
            pq.pop();
            pq.push(arr[i]);
        }
    }

    int ans=pq.top();
    return ans;
}


//Q2. is binary tree heap gfg

int countNode(node* root){
    //base case
    if(root==NULL){
        return 0;
    }

    int ans=1+countNode(root->left)+countNode(root->right);
    return ans;
}

bool isCBT(node* root,int index,int cnt){
    // base case ,if it is in the leaft node then CBT
    if(root==NULL){
        return true;
    }
    // if it go out of renge
        //     6
        //    /  \
        //   5    4
        //  / \   \
        // 2   3    1

    if(index>=cnt){
        return false;
    }
    else{
        bool left=isCBT(root->left,2*index+1 ,cnt);
        bool right=isCBT(root->right,2*index+2,cnt);

        return (left && right);
    }
}

bool isMaxOrder(node* root){
    // leaf node
    }
    if(root->left ==NULL && root->right ==NULL){
        return true;
    }
    // just left exist kore
    if(root->right==NULL){
        return (root->left  >root->left->data)
    }
    else{
        // left and right non null
        bool left=isMaxOrder(root->left);
        bool right=isMaxOrder(root->right);

        return (left && right && (root->data > root->left->data   && root->data > root->right->data
))
    }
}

bool isHeap(node* root){
    int index=0;
    int totalCount=countNode(root);
    if(isCBT(root,index,totalCount)  && isMaxOrder(root)){
        return true;
    }
    else{
        return false;
    }
}



//Q3.marge two binary max heaps  gfg


void heapify(int arr[],int n,int indx){
    int largest=indx;
    int left=2*indx;
    int right=2*indx +1;

    if(left<=n && arr[largest]<arr[left]){
        largest=left;
    }
    if(right<= n && arr[largest]<arr[right]){
        largest=right;
    }

    // if leargest is change thn swap
    if(largest !=indx){
        swap(arr[largest],arr[indx]);
        // ai process toh choltai thkbe tai recursion call korte hoba
        // largest k tar thik jaygay pouchay daw
        heapify(arr,n,largest);
    }

}


vector<int> margeHeap(vector<int> &a,vector<int> &b,int n,int m){
   // marge two arrays into  one array
    vector<int> ans;
    ans.push_back(-1);
    for(auto i:a){
        ans.push_back(i);
    }
    for(auto i: b){
        ans.push_back(i);
    }

    // build heap using marged array
    int size =ans.size();
    for(int i=size/2 ;i>0;i--){
        heapify(ans,size,i);
    }

    return ans;
}


//Q4 .minnimum cost of ropes  gfg


 long long minCost(long long arr[],long long n){
    // creat  a min heap
    priority_queue<long long ,vector<long long> ,greater<long long>> pq;
    for(int i=0;i<n;i++){
        pq.push(arr[i]);
    }

    long long cost=0;

    while(pq.size()>1){
        long long a=pq.top();
        pq.pop();
        long long  b = pq.top();
        pq.pop();
        long long sum=a+b;
        cost+=sum;
        pq.push(sum);
    }

    return cost;
 }


//Q5. Convert bst to min heap  gfg




int main(){

     int arr[]={7,10,4,20,15};
     cout<<kthSmallest(arr,0,5,4)<<endl;

     // if k th largest element ber korte bola toba min heap use korte hoba logic samae

     return 0;
}



L3:
#include<bits/stdc++.h>
using namespace std;


//Q1. k'th largest sum subarray    code stuio

int getKthLargest(vector<int> &arr,int k){
    vector<int> sumStore;

    int n=arr.size();

    for(int i=0;i<n;i++){
        int sum=0;
        for(int j=i;i<n;j++){
            sum+=arr[j];
            sumStore.push_back(sum);
        }
    }

    sort(sumStore.begin(),sumStore.end());
    return sumStore[sumStore.size()-k];
```

```
}

//optimal ans

int getKthLargestOPT(vector<int> &arr,int k){

    priority_queue<int,vector<int> ,greater<int>> mini;
    int n=arr.size();

    for(int i=0;i<n;i++){
        int sum=0;
        for(int j=i;j<n;j++){
            sum+=arr[j];
            if(mini.size()<key){
                mini.push(sum);
            }
            else{
                if(sum>mini.top()){
                    mini.pop();
                    mini.push(sum);
                }
            }

        }
    }


    return  mini.top();
}


//Q2.marge k sorted arrays   code studio
class node{
    public:
    int data;
    int i;
    int j;
    node(int val,int row,int col){
        data=val;
        i=row;
        j=col;
    }
};

class compare{
 public:
 bool operator()(node* a,node* b){
    return a->data > b->data;
 }
}


vector<int> margeKSortedArrays(vector<vector<int>> &kArrays,int k){
    priority_queue<node* ,vector<node*>,compare> minHeap;


    // step 1: saara arrays k first element insert h
    for(int i=0;i<k;i++){
        node* tmp=new node(kArrays[i][0],i,0);
        minHeap.push(tmp);
    }
    // now the minheap store all array first element in lower to upper

    //strp 2: heap ar smallest element jahetu heap ar top a aca tai taka ans  array te store kori
o jai array thaka oi elelment k paici sai array ar index k 1 increment kora dai
    vector<int> ans;

    while(minHeap.size()>0){
        node* tmp=minHeap.top();
        ans.push_back(tmp->data);
        minHeap.pop();

        int i=tmp->i;
        int j=tmp->j;

        // now cheque if the next element array ar renge ar modha exist kora kina
        if(j+1 < kArrays[i].size()){
            node* next=new node(kArrays[i][j+1],i,j+1);
            minHeap.push(next);
        }

    }
    return ans;
}


//Q3.MARGR  k sorted linked list

class compare{
    public:
    bool operator()(node<int>* a,node<int> *b){
        return a->data > b->data;
    }
}
 node<int> * margeKList(vector<node<int>> & listArray){
    priority_queue<node<int>* ,vector<node<int>*>,compare> minHeap;
    int k=listArray.size();
    if(k==0){
        return NULL;
    }

    //step 1 :
    for(int i=0;i<k;i++){
        if(listArray[i]!=NULL){
            minHeap.push(listArray[i]);
        }
    }


    //step 2
    node<int>* head=NULL;
    node<int>* tail=NULL;

    while(minHeap.size()>0){
        node<int>* top=minHeap.top();
        minHeap.pop();


            if(top->next !=NULL){
                minHeap.push(top->next);
            }


        if(head==NULL){  //answer LL is empty
            head=top;
            tail=top;

        }
        else{  //insert at linnked list
            tail->next=top;
            tail=top;

        }
    }
    return head;
 }



int main(){



     return 0;
```

```
}
```

L4:

```
#include<bits/stdc++.h>
using namespace std;
// q1. smallest renge from K sorted list   code studio medium
// #include<limit.h>,<queue>

class node{
  public:
    int data;
    int row;
    int col;
    node(int d,int r,int c){
        data=d;
        row=r;
        col=c;
    }
};


class compare{
    public:
    bool operator()(node* a,node* b){
        return a->data > b->data;
    }
}

int kSorted(vector<vector<int>> &a,int k,int n){
   int mini=INT_MAX;
   int maxi=INT_MIN;
   priority_queue<node* ,vector<node*> compare> minHeap;

   // step 1 :startin  element gula k  queue a store kori,and tracking mini and maxi value
   for(int i=0;i<k;i++){
       int element= a[i][0];
       mini=min(mini,element);
       maxi=max(maxi,element);
       minHeap.push(element,i,0);
   }

   int start=mini,end=maxi;

 // process renge
   while(!minHeap.empty()){
    //find minimum
    node* tmp=minHeap.top();
    minHeap.pop();

    mini=tmp->data;

    if((maxi-mini) < (end-start)){
        start=mini;
        end=maxi;
    }
    // if min element exist then again select mini and maxi
    if(tmp->col +1  <n ){
        maxi=max(maxi,a[tmp->row][tmp->col +1]);
        minHeap.push(new node(a[tmp->row][tmp->col +1] ,tmp->row,tmp->col+1));
    }
    else{
        // next element does't exist
        break;
    }
   }


   return (end-start +1);
}


//Q2. median in a steram  code studio hard

int signum(int a,int b){
    if(a==b){
        return 0;
    }
    else if(a>b){
        return 1;
    }
    else{
        return -1;
    }
}


void callMedian(int element.vector<int> &arr,priority_queue<int> &maxi,
    priority_queue<int,vector<int> greater<int>> &mini,
    int median){
    switch(signum(maxi.size(),mini.size())){

        case 0:  if(element>median){
                  mini.push(element);
```

```
                 median=mini.top();
               }
               else{
                maxi.push(element);
                median=maxi.top();
               }
               break;


        case 1: if(element>median){
                   mini.push(element);
                   median=(mini.top()+maxi.top())/2;
               }
               else{
                   mini.push(maxi.top());
                   maxi.pop();
                   maxi.push(element);
                   median=(mini.top()+maxi.top())/2;
               }
               break;


        case -1: if(element>median){
            maxi.push(mini.top());
            mini.pop();
            mini.push(element);
            median=(mini.top()+maxi.top())/2;
            }
            else{
                maxi.push(element);
                median=(mini.top()+maxi.top())/2;
            }
            break;

    }
}

vector<int> findMedian(vector<int> &arr,int n){
    vector<int> ans;
    priority_queue<int> maxheap;
    priority_queue<int,vector<int> greater<int>> minheap;
    int median =0;

    for(int i=0;i<n;i++){
        callMedian(arr,maxheap,minheap,median);
        ans.push_back(median);
    }
    return ans;
}


int main(){


     return 0;
}
```

L5:

```
#include<bits/stdc++.h>
using namespace std;
//q1. maximum frequency number

int maxFrequency(vector<int> &arr,int n){
   unordered_map<int,int> count;
   int maxfreq=0;
   int maxAns=0;
   for(int i=0;i<arr.size();i++){
      count(arr[i])++;
   }

   for(int i=0;i<arr.size();i++){
      if(maxfreq==count[arr[i]]){
         maxAns =arr[i];
         break;
      }
   }

   return maxAns;
}


int main(){


//creation
unordered_map<string,int> m;

// insertion
 // way1
pair<string,int> p=make_pair("shafiul",3);
m.insert(p);


//way 2
pair<string,int> p2("islam",2);
m.insert(p2);

//way 3
m["mera"]=1;
// what will happed
m["mera"]=2;
// under one key there will be a single entery


//search

cout<<m["mera"]<<endl;
cout<<m.at("shafiul")<<endl;

// if we want to search an element that doest not exist in the map

//cout<<m.at("unknownKey")<<endl;


// solve above problem
 cout<<m["unknownKey"]<<endl;
 cout<<m.at("unknownKey")<<endl;

 //size
 cout<<m.size()<<endl;

 // to cheque presence
 cout<<m.count("sa")<<endl;
 cout<<m.count("shafiul")<<endl;

 // erase
 m.erase("shafiul");

  cout<<m.size()<<endl;



  // itreator
  unordered_map<string,int>:: iterator it=m.begin();
  // unorderer map print in random  order but map print in sequential order

  while(it!=m.end()){
   cout<<it->first<<" "<<it->second<<endl;
   it++;
  }

     return 0;
}
```