***National Institute of Technology, Kurukshetra***



**Advance Data Structure**

**and Algorithm**

**Submitted to:**

**Mamtesh Mam**

**Submitted by:**

**Shubham**

**Roll No: 11710556**

**1. Write a program to implement Linear Search and Binary Search. Also perform its complexity analysis.**

Code :

#include <iostream>

using namespace std;

int linear\_search(int arr[], int n , int num){

for(int i = 0; i < n ;i++){

if(arr[i]==num){

return i;

}

}

return -1;

}

int binary\_search(int arr[] , int n ,int num){

int l = 0, r = n;

while(l<=r){

int mid = (l+r)/2;

if(arr[mid] == num){

return mid;

}

if(arr[mid] > num){

r = mid-1;

}

if(arr[mid] < num){

l = mid+1;

}

}

return -1;

}

int main()

{

int n;

cout<<"Enter the total number of elements: ";

cin>>n;

int arr[n];

cout<<"Enter the element: "<<endl;

for(int i=0;i<n;i++){

int x ;

cin>>x;

arr[i] = x;

}

int num;

cout<<"Enter the number to be search: ";

cin>>num;

//int index = linear\_search(arr , n , num );

int index = binary\_search(arr , n , num);

cout<<"Your element is at "<<index + 1<<" position"<<endl;

return 0;

}

/\*

Linear Search

Best Case : O(1)

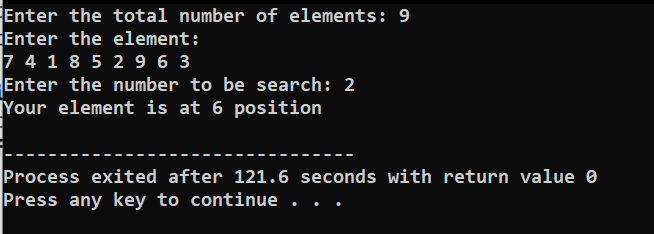
Worst Case : O(n)

Binary Search

Best Case : O(1)

Worst Case : O(logn)

\*/



**2. Finding kth minimum and maximum element in Heap**

Code :

#include <bits/stdc++.h>

using namespace std;

class Minheap

{

vector<int> v;

void heapify(int i)

{

int mini = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[mini] > v[left])

{

mini = left;

}

if (right < v.size() && v[mini] > v[right])

{

mini = right;

}

if (mini != i)

{

swap(v[mini], v[i]);

heapify(mini);

}

}

public:

Minheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] > v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int min()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMin()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

class Maxheap

{

vector<int> v;

void heapify(int i)

{

int maxi = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[maxi] < v[left])

{

maxi = left;

}

if (right < v.size() && v[maxi] < v[right])

{

maxi = right;

}

if (maxi != i)

{

swap(v[maxi], v[i]);

heapify(maxi);

}

}

public:

Maxheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] < v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int max()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMax()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

void get\_kth\_smaller(int k ,Minheap heap){

for(int i = 0 ; i < k-1 ;i++){

heap.removeMin();

}

cout<<"kth smaller Element ==> "<< heap.removeMin()<<endl;

return;

}

void get\_kth\_larger(int k, Maxheap heap)

{

for (int i = 0; i < k - 1; i++)

{

heap.removeMax();

}

cout <<"kth larger Element ==> " << heap.removeMax() << endl;

return;

}

int main(){

int n;

cout << "Enter the total number of elements: ";

cin >> n;

Minheap miniheap;

Maxheap maxaheap;

cout << "Enter the element: " << endl;

for (int i = 0; i < n; i++)

{

//int data;

//cin >> data;

miniheap.insert(i);

maxaheap.insert(i);

}

int k;

cout << "Enter the value of k: ";

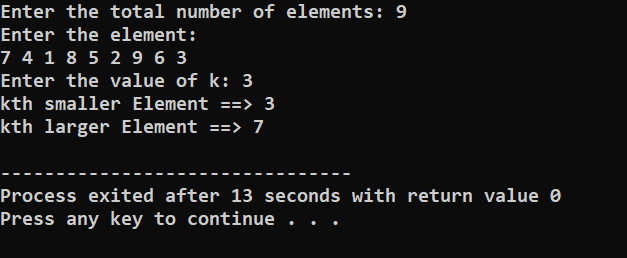
cin >> k;

get\_kth\_smaller(k , miniheap);

get\_kth\_larger(k , maxaheap );

return 0;

}



**3. Implement Quick Sort with duplicate numbers in the given array/elements**

**Code :**

#include <iostream>

using namespace std;

int partitionQuick(int a[], int l, int h){

int pivot=a[l];

int i=l;

int j=h;

while(i<j && i<=h && j>=l){

while(a[i]<=pivot && i<h){i++;}

while(a[j]>=pivot && j>l){j--;}

if(i<j)

swap(a[i],a[j]);

}

swap(a[l],a[j]);

return j;

}

void quickSort(int a[], int l, int h){

if(l<h){

int j=partitionQuick(a,l,h);

quickSort(a,l,j);

quickSort(a,j+1,h);

}

}

int main()

{

int n;

cout<<"Enter the total number of elements: ";

cin>>n;

int a[n];

cout<<"Enter the element: "<<endl;

for(int i=0;i<n;i++) cin>>a[i];

quickSort(a,0,n-1);

for(int i=0;i<n;i++){

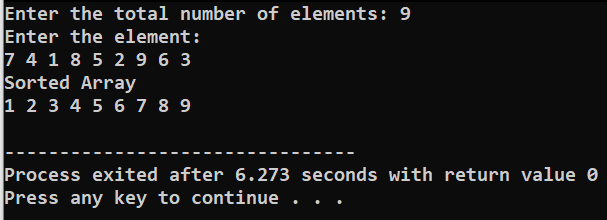
cout<<a[i]<<" ";

}

cout<<endl;

return 0;

}



**4. Build Min heap, Max heap and sort the given elements**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Minheap

{

vector<int> v;

void heapify(int i)

{

int mini = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[mini] > v[left])

{

mini = left;

}

if (right < v.size() && v[mini] > v[right])

{

mini = right;

}

if (mini != i)

{

swap(v[mini], v[i]);

heapify(mini);

}

}

public:

Minheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] > v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int min()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMin()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

class Maxheap

{

vector<int> v;

void heapify(int i)

{

int maxi = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[maxi] < v[left])

{

maxi = left;

}

if (right < v.size() && v[maxi] < v[right])

{

maxi = right;

}

if (maxi != i)

{

swap(v[maxi], v[i]);

heapify(maxi);

}

}

public:

Maxheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] < v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int max()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMax()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

void max\_heap\_sort( Maxheap heap )

{

cout<<"Sorting Element in descending order"<<endl;

while(!heap.isEmpty()){

cout<<heap.removeMax()<<" ";

}

cout<<endl;

return;

}

void min\_heap\_sort(Minheap heap)

{

cout << "Sorting Element in ascending order" << endl;

while (!heap.isEmpty())

{

cout << heap.removeMin() << " ";

}

cout << endl;

return;

}

int main()

{

int n;

cout << "Enter the total number of elements: ";

cin >> n;

Minheap miniheap;

Maxheap maxaheap;

cout << "Enter the element: " << endl;

for (int i = 0; i < n; i++)

{

int data;

cin >> data;

miniheap.insert(data);

maxaheap.insert(data);

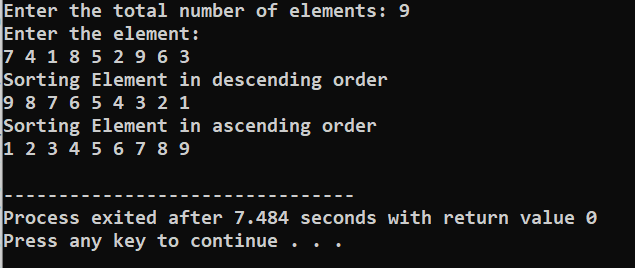
}

max\_heap\_sort(maxaheap);

min\_heap\_sort(miniheap);

return 0;

}



**5. Read the marks obtained by students of third year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm**

**Code :**

#include <bits/stdc++.h>

using namespace std;

class Minheap

{

vector<int> v;

void heapify(int i)

{

int mini = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[mini] > v[left])

{

mini = left;

}

if (right < v.size() && v[mini] > v[right])

{

mini = right;

}

if (mini != i)

{

swap(v[mini], v[i]);

heapify(mini);

}

}

public:

Minheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] > v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int min()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMin()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

class Maxheap

{

vector<int> v;

void heapify(int i)

{

int maxi = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[maxi] < v[left])

{

maxi = left;

}

if (right < v.size() && v[maxi] < v[right])

{

maxi = right;

}

if (maxi != i)

{

swap(v[maxi], v[i]);

heapify(maxi);

}

}

public:

Maxheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] < v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int max()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMax()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

int main()

{

int n;

cout << "Enter the total number of elements: ";

cin >> n;

Minheap miniheap;

Maxheap maxaheap;

cout << "Enter the element: " << endl;

for (int i = 0; i < n; i++)

{

int data;

cin >> data;

miniheap.insert(data);

maxaheap.insert(data);

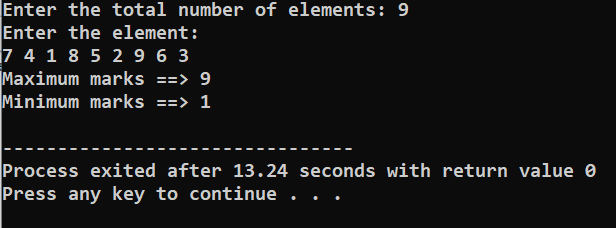
}

cout << "Maximum marks ==> " << maxaheap.removeMax() << endl;

cout << "Minimum marks ==> " << miniheap.removeMin() << endl;

return 0;

}



**6. Delete kth indexed element in Min heap and Max heap.**

**Code :**

#include <bits/stdc++.h>

using namespace std;

class Minheap

{

vector<int> v;

void heapify(int i)

{

int mini = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[mini] > v[left])

{

mini = left;

}

if (right < v.size() && v[mini] > v[right])

{

mini = right;

}

if (mini != i)

{

swap(v[mini], v[i]);

heapify(mini);

}

}

public:

Minheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] > v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int min()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMin()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

int remove\_ith(int i ){

int top = v[i];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(i);

return top;

}

void print()

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

class Maxheap

{

vector<int> v;

void heapify(int i)

{

int maxi = i;

int left = 2 \* i;

int right = 2 \* i + 1;

if (left < v.size() && v[maxi] < v[left])

{

maxi = left;

}

if (right < v.size() && v[maxi] < v[right])

{

maxi = right;

}

if (maxi != i)

{

swap(v[maxi], v[i]);

heapify(maxi);

}

}

public:

Maxheap()

{

v.push\_back(-1);

}

int size()

{

return v.size() - 1;

}

bool isEmpty()

{

return v.size() == 1;

}

void insert(int data)

{

v.push\_back(data);

int index = v.size() - 1;

while (index > 1 && v[index / 2] < v[index])

{

swap(v[index], v[index / 2]);

index = index / 2;

}

return;

}

int max()

{

if (isEmpty())

{

return -1;

}

return v[1];

}

int removeMax()

{

int top = v[1];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(1);

return top;

}

int remove\_ith(int i)

{

int top = v[i];

v[1] = v[v.size() - 1];

v.pop\_back();

heapify(i);

return top;

}

void print()

{

for (int i = 1; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

return;

}

};

int main()

{

int n;

cout << "Enter the total number of elements: ";

cin >> n;

Minheap miniheap;

Maxheap maxaheap;

cout << "Enter the element: " << endl;

for (int i = 0; i < n; i++)

{

int data;

cin >> data;

miniheap.insert(data);

maxaheap.insert(data);

}

int k;

cout<<"Enter value of k"<<endl;

cin>>k;

cout<<"All Elements of Max heap Before"<<endl;

maxaheap.print();

cout << "kth ==> " << maxaheap.remove\_ith(k) << endl;

cout << "All Elements of Max After Before" << endl;

maxaheap.print();

cout << "All Elements of Min heap Before" << endl;

miniheap.print();

cout << "kth ==> " << miniheap.remove\_ith(k) << endl;

cout << "All Elements of Min heap Before" << endl;

miniheap.print();

return 0;

}

