Algorithms and Problem-Solving Lab(15B17CI471) EVEN 2025

Week–2

**Q1.Consider a sorted array A of n elements. The array A may haverepetitive/duplicateelements.Designandimplementanefficientalgorithmfora given target element T,to find T’s first and last occurrence in the array A. Also,printthemessageifanelementisNOTPRESENTinthearray.**

|  |  |
| --- | --- |
| **Example1:**  **Input:arr=[2,5,5,5,6,6,8,9,9,9]**  **target=5**  **Output:**  **Thefirstoccurrenceofelement5islocatedat index 1**  **Thelastoccurrenceofelement5islocatedat index 3** | **Example2:**  **Input:arr=[2,5,5,5,6,6,8,9,9,9]**  **target=4Output:**  **Elementnotfoundinthearray** |

#include <iostream>

#include <vector>

using namespace std;

int binarySearchFirst(const vector<int>&arr,inttarget,intlow,int high) {

while (low<=high)

{

int mid=low+(high-low)/2;

if (arr[mid]==target)

{

if (mid==0||arr[mid-1]!=target)

return mid;

else

high=mid-1;

}

else if (arr[mid]<target)

low=mid+1;

else

high=mid-1;

}

return -1;

}

int binarySearchLast(const vector<int>&arr,inttarget,intlow,int high) {

while (low<=high)

{

int mid=low+(high-low)/2;

if (arr[mid]==target)

{

if (mid==arr.size()-1||arr[mid+1]!=target)

return mid;

else

low=mid+1;

}

else if (arr[mid]<target)

low=mid+1;

else

high=mid-1;

}

return -1;

}

void findFirstAndLast(const vector<int>&arr,int target) {

int first=binarySearchFirst(arr,target,0,arr.size()-1);

if (first==-1)

{

cout<<"The element "<<target<<" is not present in the array."<<endl;

return;

}

int last=binarySearchLast(arr,target,0,arr.size()-1);

cout<<"The first occurrence of element "<<target<<" is located at index "<<first<<endl;

cout<<"The last occurrence of element "<<target<<" is located at index "<<last<<endl;

}

int main()

{

int n,target;

cout<<"Input the number of elements : ";

cin>>n;

vector<int>arr(n);

cout<<"Input the elements : ";

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

cin>>arr[i];

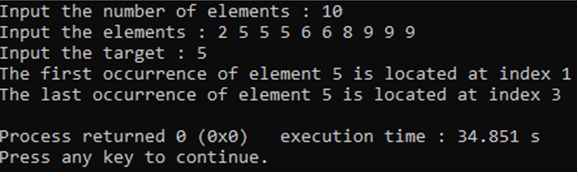
cout<<"Input the target : ";

cin>>target;

findFirstAndLast(arr,target);

}

**Output :**

****

**Q2.Givenalistofpointsona2DPlane.Calculate the ‘*k*’closestpoints to the origin (0,0)assuming that the valueof*k* is given by the useratruntime.Consider Euclidean distance to find the distance between two points.Use D&C approach.**

* 1. **WriteaC++codetoreturntheanswerinanyorder.Analyzeapace&timecomplexity.**
  2. **Sortallthe‘*k*’closestpointsindescendingorder.**

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;

struct Point

{

int x,y;

Point(int x,int y) : x(x),y(y) {}

};

double euclideanDistance(Point p) {

return sqrt(p.x\*p.x+p.y\*p.y);

}

int partition(vector<Point>&points,intlow,int high)

{

double pivot=euclideanDistance(points[high]);

int i=low-1;

for (int j=low;j<high;j++)

if (euclideanDistance(points[j])<=pivot)

{

i++;

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

}

swap(points[i+1],points[high]);

return i+1;

}

void quickSelect(vector<Point>&points,intlow,inthigh,int k)

{

if (low<high)

{

int pivotIndex=partition(points,low,high);

if (pivotIndex==k) return;

else if (pivotIndex<k) quickSelect(points,pivotIndex+1,high,k);

else quickSelect(points,low,pivotIndex-1,k);

}

}

vector<Point>kClosestPoints(vector<Point>&points,int k)

{

quickSelect(points,0,points.size()-1,k);

vector<Point> result(points.begin(),points.begin()+k);

return result;

}

int main()

{

int n,k,x,y;

cout<<"Input the number of points : ";

cin>>n;

vector<Point> points;

cout<<"Input the x then y coordinates of the point : ";

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

cin>>x>>y;

points.push\_back(Point(x,y));

}

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

cin>>k;

vector<Point> closest=kClosestPoints(points,k);

sort(closest.begin(),closest.end(),[](Point a,Point b) {

return euclideanDistance(a)>euclideanDistance(b);

});

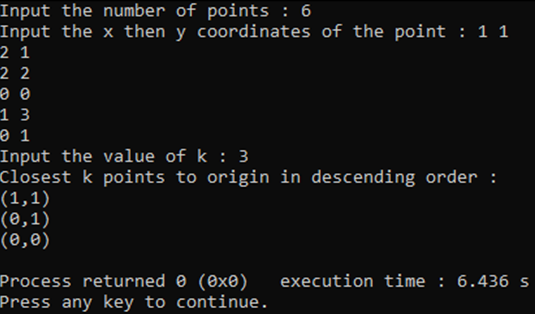
cout<<"Closest k points to origin in descending order : "<<endl;

for (auto& p : closest)

cout<<"("<<p.x<<","<<p.y<<")"<<endl;

}

**Output :**

****

**Q3. Cubic integer root *x* of *n* is the largest number *x* such that *x*3<=*n*. Find the value of *x* given *n* using the D&C approach. Also,analyse the complexity.**

#include <iostream>

using namespace std;

int cubicRoot(int n)

{

int low=0,high=n,ans=0;

while (low<=high)

{

int mid=low+(high-low)/2;

long long cube=1LL\*mid\*mid\*mid;

if (cube<=n)

ans=mid;

low=mid+1;

else

high=mid-1;

}

return ans;

}

int main()

{

int n;

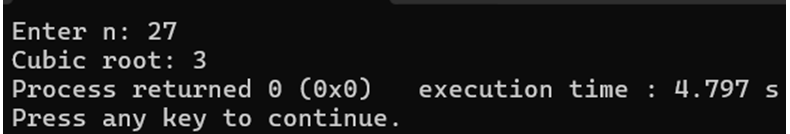
cout<<"Enter n: ";

cin>>n;

cout<<"Cubic root: "<<cubicRoot(n);

}

**Output :**

****

**Q5.CreateamodifiedversionofSelectionSortusingthedivideandconquer paradigm. Write its recurrence equation as well.**

#include <iostream>

using namespace std;

void conquer(int arr[],int left,intmid,int right)

{

for (int i=left;i<=mid;i++)

{

int minIndex=i;

for (int j=i+1;j<=right;j++)

if (arr[j]<arr[minIndex])

minIndex=j;

if (minIndex!=i)

swap(arr[i],arr[minIndex]);

}

}

void divideAndSort(int arr[],int left,int right)

{

if (left<right)

{

int mid=left+(right-left)/2;

divideAndSort(arr,left,mid);

divideAndSort(arr,mid+1,right);

conquer(arr,left,mid,right);

}

}

void selectionSortDNC(int arr[],int n)

{

divideAndSort(arr,0,n-1);

}

int main()

{

int n;

cout<<"Input the size of the array : ";

cin>>n;

int arr[n];

cout<<"Input the elements : ";

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

cin>>arr[i];

selectionSortDNC(arr,n);

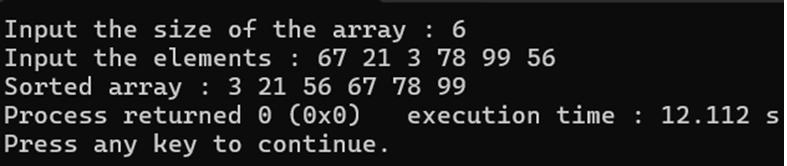
cout<<"Sorted array : ";

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

cout<<arr[i]<<" ";

}

**Output :**

****

**Recurrence Equation :** T(n)=2T(2n​)+O(n2)

**Q6. Given an array,count the number of inversions using a D&C approach. Example**

**Input:arr[]={8,4,2,1}**

**Output:6inversions:(8,4),(4,2),(8,2),(8,1),(4,1),(2,1).**

#include <iostream>

using namespace std;

int mergeAndCount(int arr[],int temp[],int left,intmid,int right)

{

int i=left,j=mid+1,k=left,count=0;

while (i<=mid&&j<=right)

{

if (arr[i]<=arr[j])

temp[k++]=arr[i++];

else

{

for (int p=i;p<=mid;p++)

cout<<"("<<arr[p]<<","<<arr[j]<<")"<<endl;

temp[k++]=arr[j++];

count+=(mid-i+1);

}

}

while (i<=mid)

temp[k++]=arr[i++];

while (j<=right)

temp[k++]=arr[j++];

for (i=left;i<=right;i++)

arr[i]=temp[i];

return count;

}

int mergeSortAndCount(int arr[],int temp[],int left,int right)

{

int count=0;

if (left<right)

{

int mid=left+(right-left)/2;

count+=mergeSortAndCount(arr,temp,left,mid);

count+=mergeSortAndCount(arr,temp,mid+1,right);

count+=mergeAndCount(arr,temp,left,mid,right);

}

return count;

}

int countInversions(int arr[],int n)

{

int\* temp=new int[n];

int result=mergeSortAndCount(arr,temp,0,n-1);

delete[] temp;

cout<<"Total Inversions : ";

return result;

}

int main()

{

int n;

cout<<"Input the size of the array : ";

cin>>n;

int arr[n];

cout<<"Input the elements : ";

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

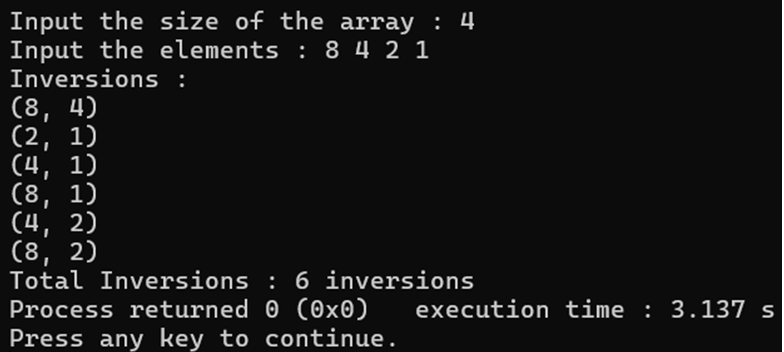
cin>>arr[i];

cout<<"Inversions :"<<endl;

cout<<countInversions(arr,n)<<" inversions";

}

**Output :**

****

**Q7.Givenasortedarray,whereinall,exceptone,elementsappeartwice(oneafter one)andoneelementappearsonlyonce.FindthatelementinO(logn)complexity.**

#include <iostream>

using namespace std;

int findSingleElement(int arr[],int n)

{

int low=0,high=n-1;

while (low<high)

{

int mid=low+(high-low)/2;

if (mid%2==1)

mid--;

if (arr[mid]==arr[mid+1])

low=mid+2;

else

high=mid;

}

return arr[low];

}

int main()

{

int n;

cout<<"Input the size of the array : ";

cin>>n;

int arr[n];

cout<<"Input the elements : ";

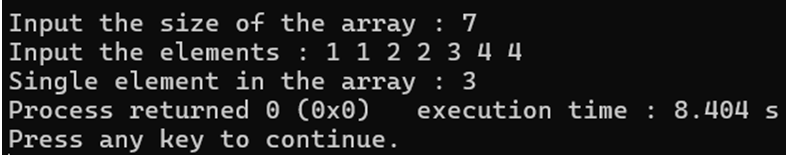
for(int i=0;i<n;i++)

cin>>arr[i];

cout<<"Single element in the array : "<<findSingleElement(arr,n);

}

**Output :**



**Q8.Givenanarray,findthemajorityelement(anelementthatappearsmorethan**

***n/2*times)usingaD&Capproach.**

|  |  |
| --- | --- |
| **Example1**  **Input:A[]={3,3,4,2,4,4,2,4,4}**  **Output:4**  **Explanation: The frequency of 4 is 5 which is greater than half of the size of the array size.** | **Example2**  **Input:A[]={3,3,4,2,4,4,2,4}**  **Output:NoMajority Element**  **Explanation:Thereisnoelementwhosefrequencyisgreaterthanhalfofthesize of the array size.** |

#include <iostream>

#include <vector>

using namespace std;

int findCandidate(const vector<int>&arr,intleft,int right)

{

if (left==right)

return arr[left];

int mid=left+(right-left)/2;

int leftCandidate=findCandidate(arr,left,mid);

int rightCandidate=findCandidate(arr,mid+1,right);

if (leftCandidate==rightCandidate)

return leftCandidate;

int leftCount=0,rightCount=0;

for (int i=left;i<=right;i++)

{

if (arr[i]==leftCandidate)

leftCount++;

else if (arr[i]==rightCandidate)

rightCount++;

}

return leftCount>rightCount ? leftCandidate : rightCandidate;

}

bool isMajority(const vector<int>&arr,int candidate)

{

int count=0;

for (int num : arr)

if (num==candidate)

count++;

return count>arr.size()/2;

}

void findMajorityElement(const vector<int>&arr)

{

int candidate=findCandidate(arr,0,arr.size()-1);

cout<<"Majority Element : ";

if (isMajority(arr,candidate))

cout<<candidate;

else

cout<<"No Majority Element";

}

int main()

{

int n;

cout<<"Input the size of the array : ";

cin>>n;

vector<int>arr(n);

cout<<"Input the elements : ";

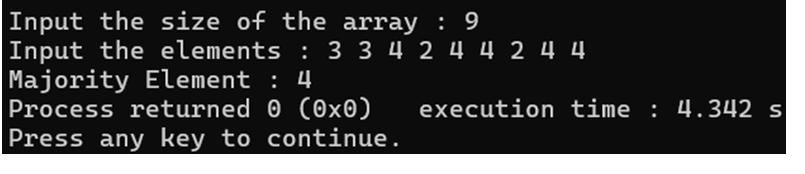
for(int i=0;i<n;i++)

cin>>arr[i];

findMajorityElement(arr);

}

**Output :**

****