Date:24.08.2021

**Third Year B. Tech., Sem V 2021-22**

**Design and Analysis of Algorithm Lab**

**Assignment / Journal submission**

**PRN/ Roll No: 2019BTECS00064**

**Full name: Kunal Santosh Kadam**

**Batch: T2**

**Assignment: Week 2**

**Title of assignment: Sorting Algorithm**

1. Given an array A[0…n-1] of n numbers containing repetition of some number. Given an algorithm for checking whether there is repeated element or not. Assume that we are not allowed to use additional space (i.e., we can use a few temporary variable, O(1) storage).

Ans:

1. **Algorithm: (Pseudocode)**

* Traverse the array from start
* For every element take its absolute value and if the abs(arr[i])th element is positive, change it to negative as it has not been encountered before.
* Else if it is negative the element has been encountered before and so we increment the counter

1. **Code snapshots of implementation**

#include<bits/stdc++.h>

using namespace std;

int repeatingNumbers(int arr[], int size)

{

int cnt=0;

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

{

if(arr[abs(arr[i])]>=0)

arr[abs(arr[i])]=-arr[abs(arr[i])];

else

cnt++;

}

return cnt;

}

int main()

{

int n;

cout<<"Enter size of array: ";

cin>>n;

int arr[n];

cout<<"Enter array elements\n";

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

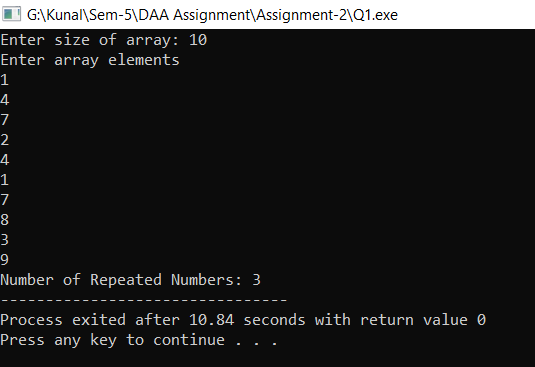
cin>>arr[i];

cout<<"Number of Repeated Numbers: "<<repeatingNumbers(arr,n);

return 0;

}

**Output:**



1. **Complexity of proposed algorithm (Time & Space)**

* Time Complexity: O(n)
* Space Complexity: O(1)

1. **Your comment (How your solution is optimal?)**

* The proposed algorithm is space efficient.
* If we sort algorithm and the find the repeating array then we would need O(nlogn) time complexity.

1. Given an array A[0…n-1] , where each element of the array represents a vote in the election. Assume that each vote is given as an integer representing the ID of the chosen candidate. Given an algorithm for determining who wins the election.

Ans:

1. **Algorithm: (Pseudocode)**

* Traverse the array from start
* Take a counter array of same size having initial values 0, As we travers the array increments the counter of the element of the array in counter array.
* Now traverse the counter array and find which element has the highest count and print it.

1. **Code snapshots of implementation**

#include<bits/stdc++.h>

using namespace std;

int election\_won(int arr[],int n)

{

int count[n]={0};

int pos=0,max=0;

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

{

count[arr[i]]++;

if(count[i]>max)

{

max=count[i];

pos=arr[i];

}

}

return pos;

}

int main()

{

int n;

cout<<"Enter size of array";

cin>>n;

int arr[n];

cout<<"Enter array elements\n";

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

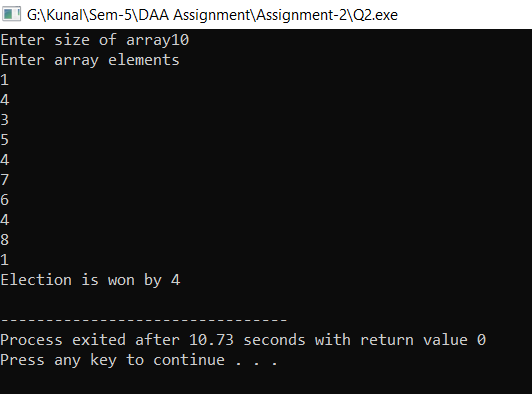
cin>>arr[i];

cout<<"Election is won by "<<election\_won(arr,n)<<endl;

return 0;

}

**Output:**



1. **Complexity of proposed algorithm (Time & Space)**

* Time Complexity: O(n)
* Space Complexity: O(n)

1. **Your comment (How your solution is optimal?)**

* The proposed algorithm is time efficient O(n)

1. Given an array A of n elements, each of which is an integer in the range [1, n2 ]. How do we sort the array in O(n) time?

Ans:

1. **Algorithm: (Pseudocode)**

* Take a map for counting the recursive elements of the array.
* Traverse the may from the start and print the elements that have come equal to or more than one.

1. **Code snapshots of implementation**

#include <bits/stdc++.h>

#include <iostream>

#include <map>

#include <string>

#include <iterator>

#include <algorithm>

using namespace std;

int main()

{

int length;

cout<<"Enter the length of array: ";

cin >> length;

vector<int>array(length);

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

cin >> array[i];

map<int, int>count;

cout << "Before Sorting array" << endl;

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

{

count[array[i]]++;

cout << array[i] << " ";

}

cout << endl << "After Sorting array" << endl;

map<int, int>::iterator it;

for (it=count.begin();it!=count.end();it++)

{

for (int i = 0; i < it->second; i++)

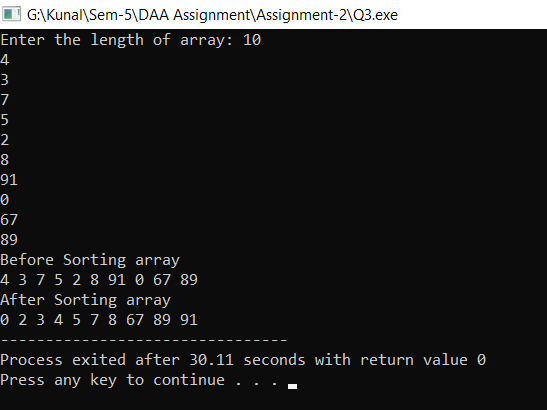
cout << it->first << " ";

}

return 0;

}

**Output:**



1. **Complexity of proposed algorithm (Time & Space)**

* Time Complexity: O(n)
* Space Complexity: O(n)

1. **Your comment (How your solution is optimal?)**

* Solution works on O(n) time complexity

1. Let A and B two arrays of n elements, each. Given a number K, give an O (nlogn) time algorithm for determining whether there exists a ϵ A and b ϵ B such that a+b =K.

Ans:

1. **Algorithm: (Pseudocode)**

* Create two HashMap and store frequency of each element in respective array
* Traverse 1st HashMap and whether there is an element (k key) if present then print YES otherwise NO

1. **Code snapshots of implementation**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int length, k;

cout<<"Enter the length of array: ";

cin >> length;

cout<<"Enter value of K: ";

cin>>k;

vector<int>array(length), array2(length); map<int, int>mp1, mp2;

cout<<"Enter elements of array A:\n";

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

cin >> array[i];

cout<<"Enter elements of array B:\n";

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

cin >> array2[i];

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

{

mp1[array[i]]++;

mp2[array2[i]]++;

}

bool check = false;

map<int, int>::iterator it;

for (it=mp1.begin();it!=mp1.end();it++)

{

if (mp2[k - it->first] != 0)

{

check = true;

break;

}

}

if (check)

cout << "YES";

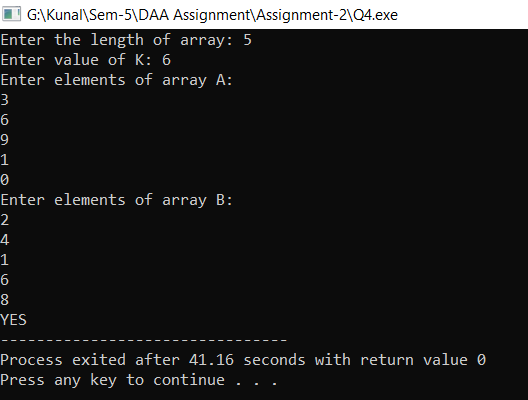
else

cout << "NO";

return 0;

}

**Output:**



1. **Complexity of proposed algorithm (Time & Space)**

* Time Complexity: O(nlogn)
* Space Complexity: O(n)

1. **Your comment (How your solution is optimal?)**

* My solution is time efficient as it takes O(n) time complexity