DATA STRUCTURES AND ALGORITHMS

Assignment - 1 (Github link)

Name : Kondury Rishabh

Branch : DSAI

Roll No. : 201020425

*used GCC 10.2 , sublime text 4 for this assignment

Q1.

Code :

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


void solve(){
     //taking inputs :
     int n ; cin >> n ;
     string name[n], number[n];
     for(int i = 0 ; i < n ; i++){
     cin >> name[i] >> number[i];
     }

     //sorting in the order of names using bubble sort :
     for(int i = 0 ; i < n ; i++){
     for(int j = i+1 ; j < n ; j++){
          if(name[i] > name[j]){
               swap(name[i], name[j]);
               swap(number[i], number[j]);
          }
     }
     }

     for(int i = 0 ; i < n ; i++){
     cout << name[i] << " " << number[i] << endl ;
     }

     //for finding the name of the person with the given phone
number :
     string phone_number; cin >> phone_number ;
     for(int i = 0 ; i < n ; i++){
     if(number[i] == phone_number){
```

```
          cout << "Name Telephone Number \n";
          cout << name[i] << " " << phone_number << endl ;
          return ;
     }
     }

     cout << "Name Telephone Number \n";
     cout << "The Entered Number is not in the directory";
}

int main()
{
     ios_base::sync_with_stdio(false);
     cin.tie(NULL);

     int t = 1 ;
     while(t--){
     solve();
     cout << endl ;
     }
}
```

Output :

Main.cpp | Main.java | Main.py

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


void solve(){
    //taking inputs :
    int n ; cin >> n ;
    string name[n], number[n];
    for(int i = 0 ; i < n ; i++){
        cin >> name[i] >> number[i];
    }

    //sorting in the order of names using bubble sort :
    for(int i = 0 ; i < n ; i++){
        for(int j = i+1 ; j < n ; j++){
            if(name[i] > name[j]){
                swap(name[i], name[j]);
                swap(number[i], number[j]);
            }
        }
    }

    cout << "Ordered List \n";
    for(int i = 0 ; i < n ; i++){
        cout << name[i] << " " << number[i] << endl ;
    }cout << endl ;

    //for finding the name of the person with the given phone number :
    string phone_number; cin >> phone_number ;
```

Input

```
10
Rahul 9598454222
Ashwin 7501202255
saleem 8545222522
Rithwik 7853266523
Anu 8832266636
Jancy 7852366336
Atul 7515555655
Jibin 9852453662
Jithin 7855656332
Stebin 8762556625
9598454222
```

stdout | stderr | compile output | scribble

```
Ordered List
Anu 8832266636
Ashwin 7501202255
Atul 7515555655
Jancy 7852366336
Jibin 9852453662
Jithin 7855656332
Rahul 9598454222
Rithwik 7853266523
Stebin 8762556625
saleem 8545222522

Name Telephone Number
Rahul 9598454222
```

Successful, 0.00s, 3152KB

Explanation :

Algorithm : The input is sorted according to the lexicographical order of the names. If the name corresponds to the name given in the input, break the loop and return the name found

Time complexity :

Worst Case : O(n^2) , the sorting is done using bubble sort, which results in the time complexity of O(n^2), the sorting can also be done using merge sort, which decreases the complexity to O(nlogn)

Best Case : O(n) , if the array is already sorted , the best case complexity turns out to be O(n)

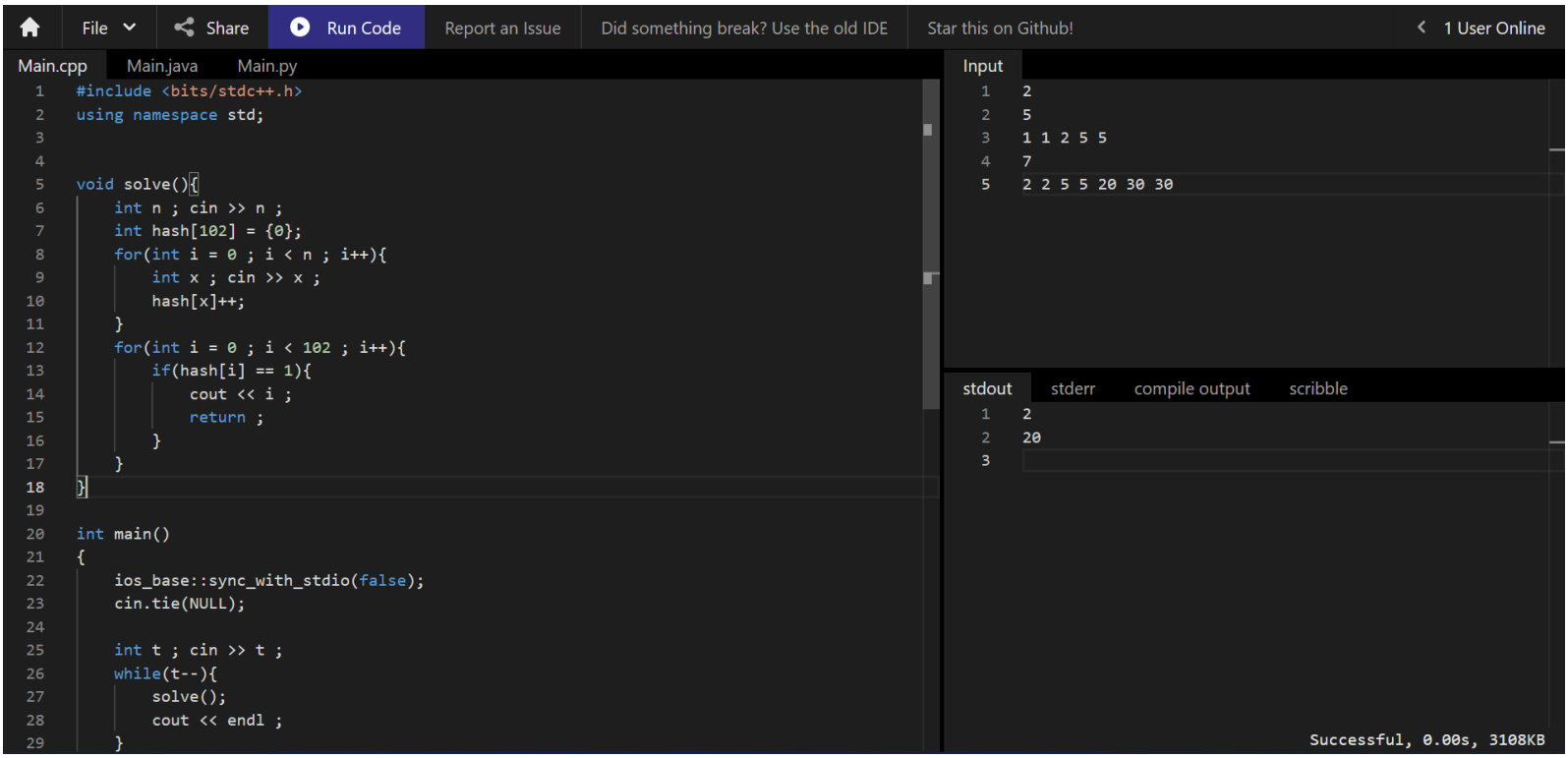
Q2.

Code :

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


void solve(){
     //taking inputs :
     int n ; cin >> n ;

     //making a hash array to store the frequency of all the
numbers till 100 :
     int hash[102] = {0};
     for(int i = 0 ; i < n ; i++){
     int x ; cin >> x ;
     hash[x]++;
     }

     //for the unique element , frequency is 1 , so breaking
loop when freq 1 is found :
     for(int i = 0 ; i < 102 ; i++){
     if(hash[i] == 1){
          cout << i ;
          return ;
     }
     }
}

int main()
{
     ios_base::sync_with_stdio(false);
```

```
    cin.tie(NULL);

    int t ; cin >> t ;
    while(t--){
    solve();
    cout << endl ;
    }
}
```

Output :



Explanation :

Algorithm : The frequency of all the elements are stored in the hash array and the value which has frequency 1 is returned by traversing the frequency array.

Improvement : The time complexity can be improved using Xor , the Xor of two same numbers ie. a^a = 0 , So after taking the xor of all the elements , the value corresponds to the ans required.

Time complexity :

Worst Case : O(n) , The array is iterated once to make the hash array, resulting in linear complexity.

Best Case : O(n) , The array needs to be traversed at least once, so in the best case also , time complexity is O(n). However , the space complexity can be decreased by using xor method , which only requires constant extra space.

Optimised code :

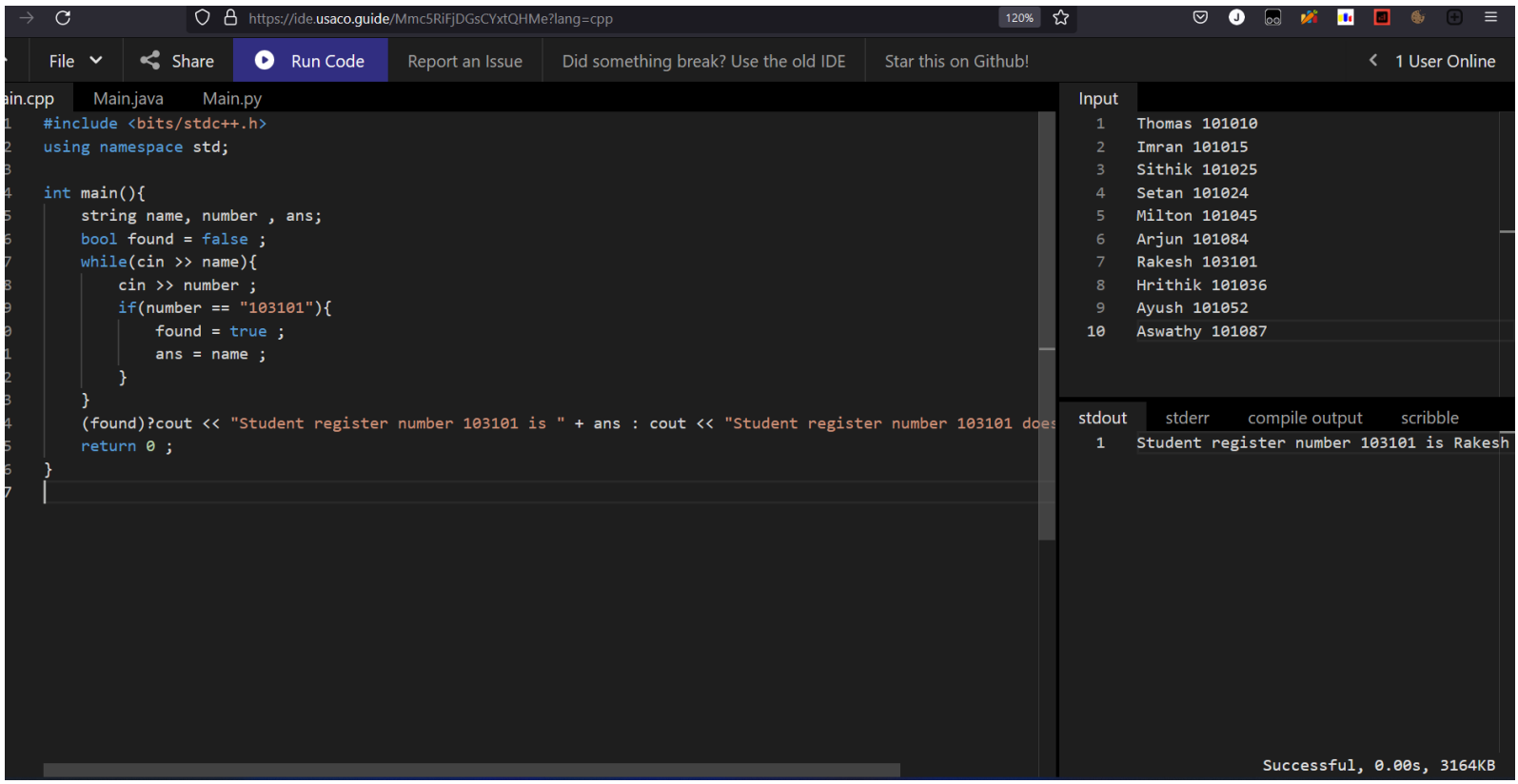
```cpp
#include <bits/stdc++.h>
using namespace std;


void solve(){
    int n ; cin >> n ;
    int ans ; cin >> ans ;
    for(int i = 1 ; i < n ; i++){
    int a ; cin >> a ;
    ans = int(ans^a);
    }
    cout << ans ;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t ; cin >> t ;
    while(t--){
    solve();
    cout << endl ;
    }
}
```

Q3.

Code :

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

int main(){
     //declaring variables for storing name and number :
     string name, number , ans;
     bool found = false ;

     //taking inputs :
     while(cin >> name){
     cin >> number ;
     if(number == "103101"){
          ans = name ;
          found = true ;
     }
     }

     //condition to print
     (found)?cout << "Student register number 103101 exists" +
ans: cout << "Student register number 103101 does not exist";
     return 0 ;
}
```

Output :



Explanation :

Time complexity :

Worst Case : O(n) , All the names and roll numbers have to be iterated once. The worst case corresponds when the roll number does not match with the key.

Best Case : O(1) , If the reuired roll number is found on the first index , the number of operations req will be 1

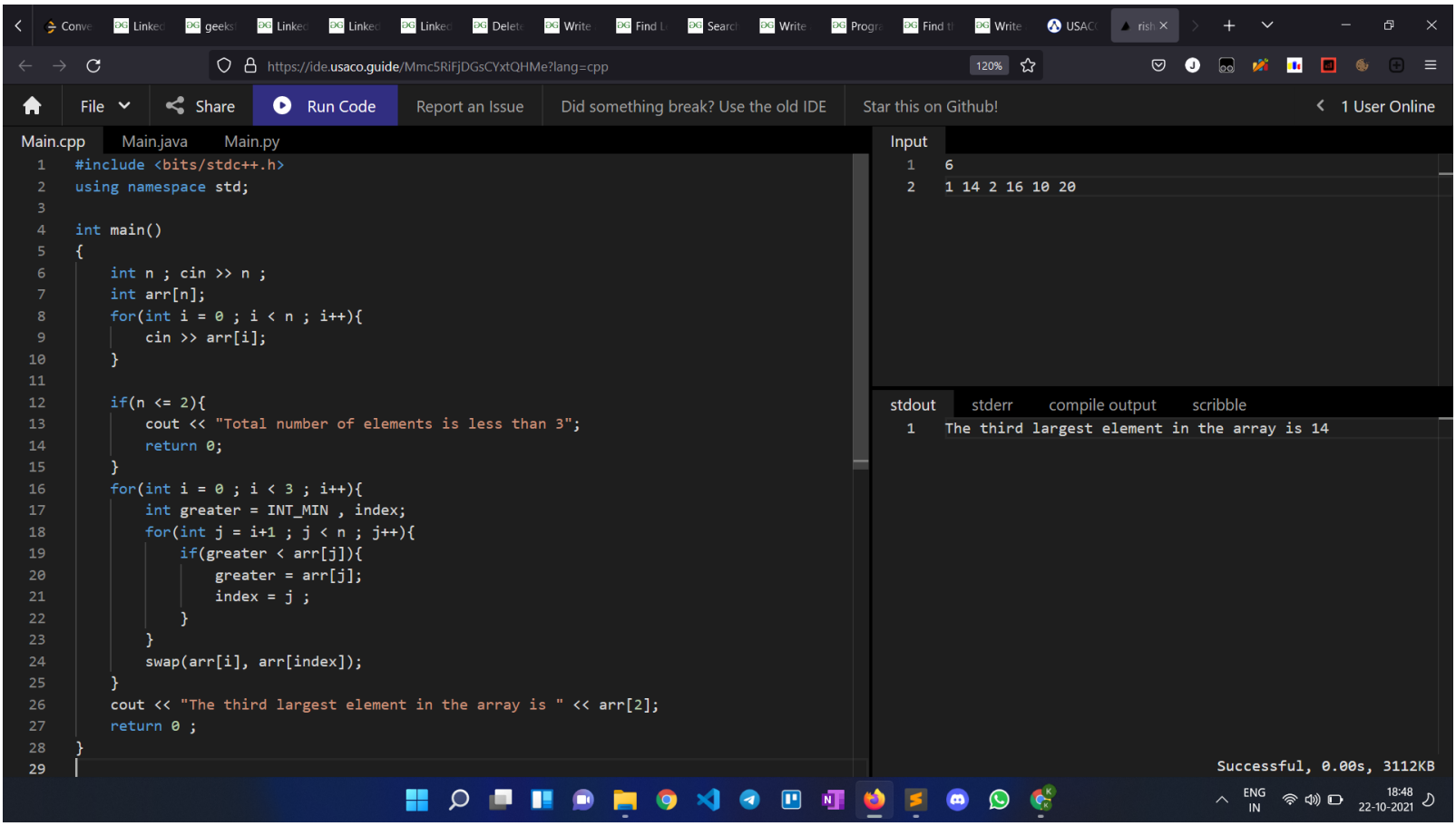
Q4.

Code :

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

void solve(){
     //taking inputs :
     int n ; cin >> n ;
     int arr[n];
     for(int i = 0 ; i < n ; i++){
      cin >> arr[i];
     }

     //handling corner case :
     if(n <= 2){
      cout << "Total number of elements is less than 3";
      return ;
     }

     //applying the 3 steps of inserting sort and sorting in
decreasing order :
     for(int i = 0 ; i < 3 ; i++){
      int greater = INT_MIN , index;
      for(int j = i+1 ; j < n ; j++){
           if(greater < arr[j]){
                greater = arr[j];
                index = j ;
           }
      }
      swap(arr[i], arr[index]);
     }
     cout << "The third largest element in the array is " <<
arr[2];
```

```
}

int main()
{
     ios_base::sync_with_stdio(false);
     cin.tie(NULL);

     int t; cin >> t ;
     while(t--){
     solve();
     cout << endl ;
     }

}
```

Output :



Explanation :

//Time complexity :

// # Worst Case : O(n) , The array is iterated atmost 3 times to find the 3rd largest number.

// # Best Case : O(n) , The array needs to be iterated atleast 3 times to find out the largest elements and insert it in the right order.

Q5.

Code :

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

void solve(){
    //taking inputs and storing it in 1d array of size m*n :
    int n , m ; cin >> n >> m ;
    int matrix[n*m];
    for(int i = 0 ; i < n*m ; i++){
    cin >> matrix[i];
    }
    int x ; cin >> x ;
    int position = -1 ;

    //searching key in the array
    for(int i = 0 ; i < m*n ; i++){
    if(matrix[i] == x){
        position = i+1;
        break;
    }
```

```
    }
    //returning the rows and columns if element is found
    (position != -1)?cout << "1 , " <<
ceil(position/(float)n) << "*" <<  (position-1)%n + 1 : cout
<< 0 ;

}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t = 1;
    // cin >> t ;
    while(t--){
    solve();
    cout << endl ;
    }

}
```

Explanation :

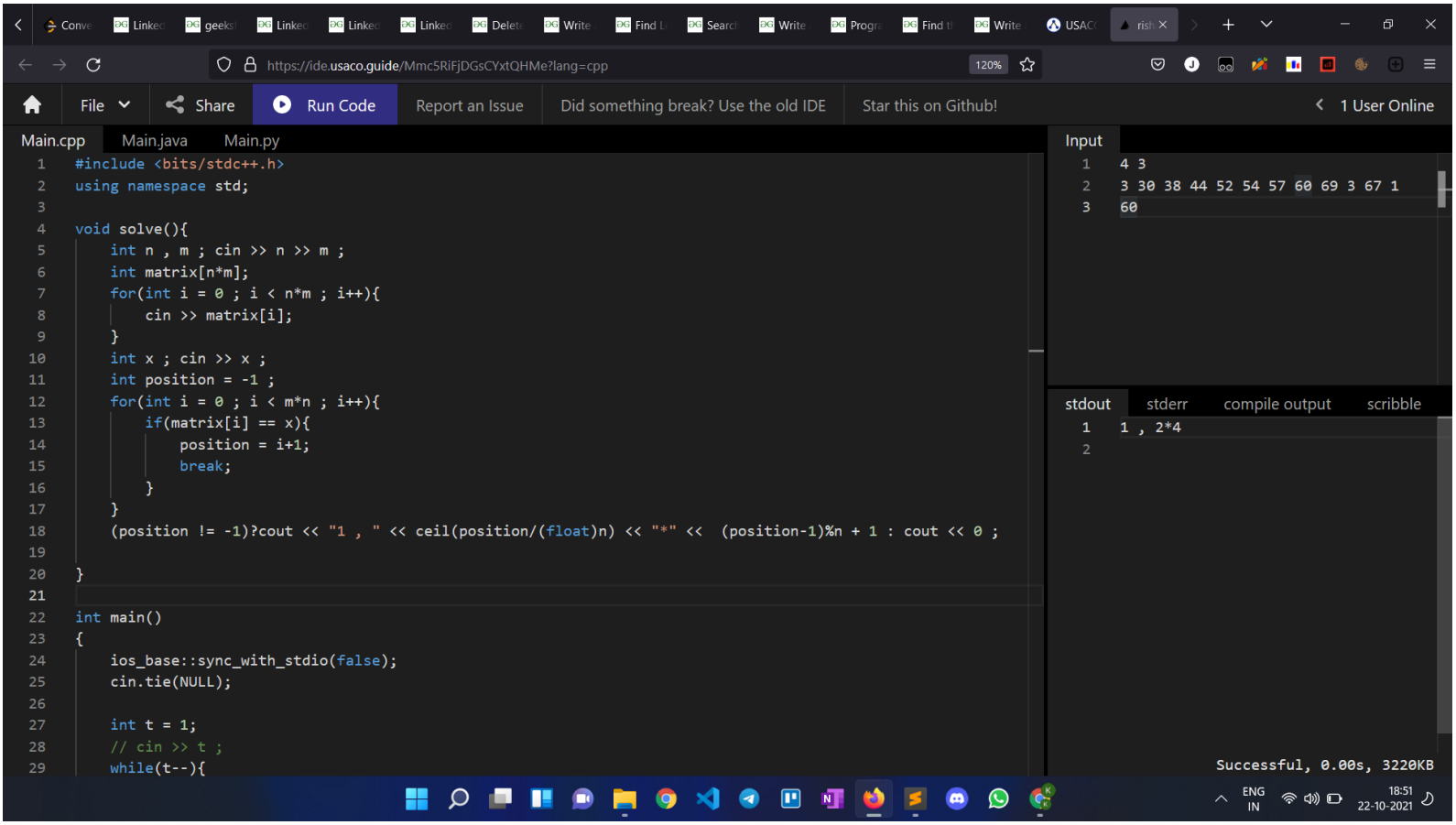
Algorithm : As the array given is always sorted , binary search can be applied to find if the key exists or not in O(log(n))

Time complexity :

Worst Case - O(log(n)) , the array input takes O(n) time and the same with linear search but for finding the element using binary search , time complexity is reduced to O(logn)

Best Case - O(1) , if the key is found at the starting index in linear search or in the mid in binary search, we can find it in constant time.

Output :



Optimised Code

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

//binary searching the key in the sorted array :
int binary_search(int arr[], int n, int k){
     int low = 0 , high = n-1 ;
     while(low < high){
     int mid = (low + high)/2 ;
     if(arr[mid] == k){
          return mid ;
     }else if(arr[mid] < k){
          low = mid + 1 ;
     }else{
          high = mid - 1 ;
     }
     }

     return -1 ;
}

void solve(){
     //taking inputs and storing it in 1d array of size m*n :
     int n , m ; cin >> n >> m ;
     int matrix[n*m];
     for(int i = 0 ; i < n*m ; i++){
     cin >> matrix[i];
     }
     int x ; cin >> x ;
     int position = -1 ;

     //searching key in the array
     position = binary_search(matrix, n*m, x) + 1;
```

```
    //returning the rows and columns if element is found :
    (position != 0)?cout << "1 , " << ceil(position/(float)n)
<< "*" <<  (position-1)%n + 1 : cout << 0 ;

}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t = 1;
    // cin >> t ;
    while(t--){
    solve();
    cout << endl ;
    }

}
```

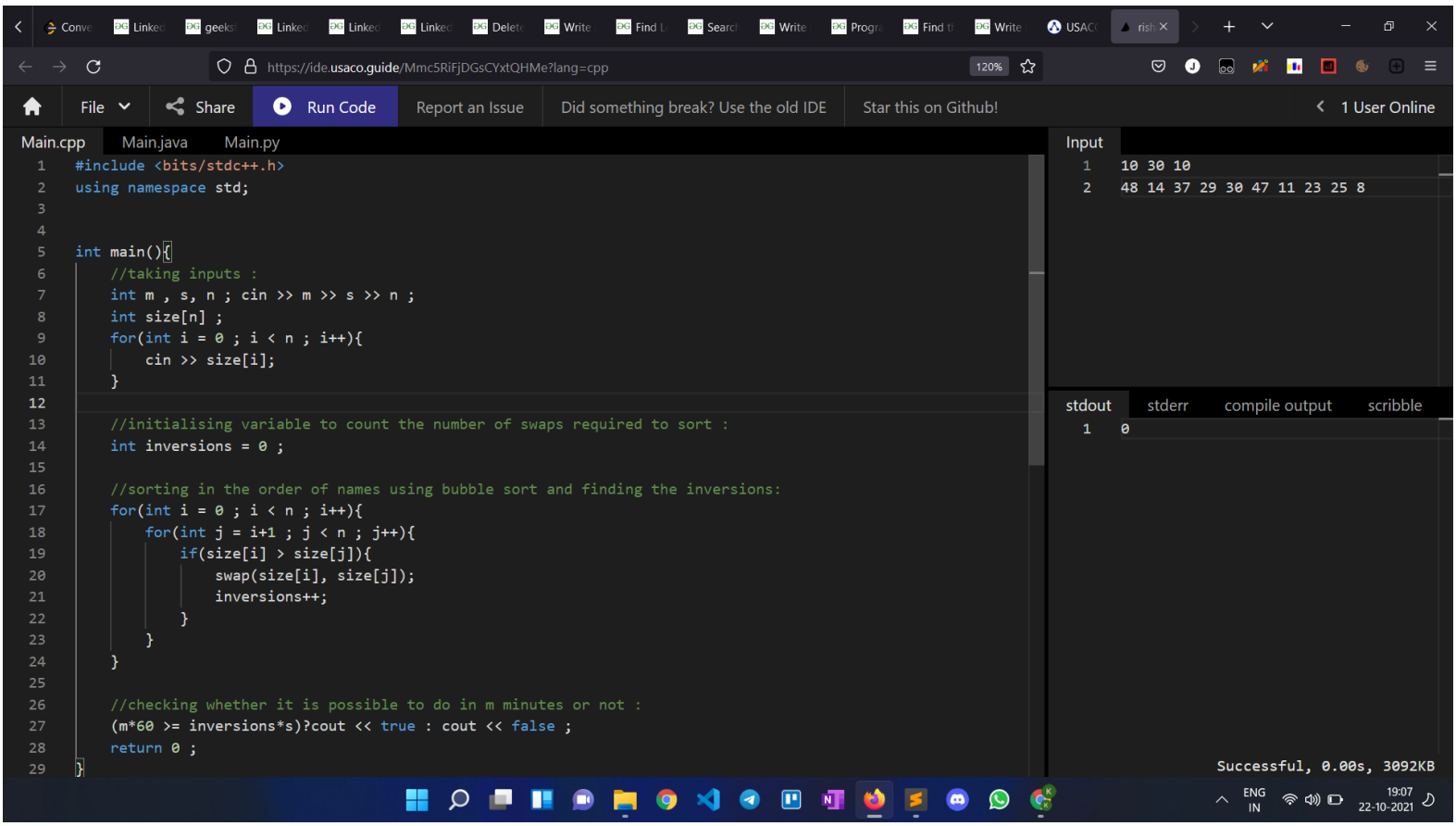
Q6.

Code :

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


int main(){
    //taking inputs :
    int m , s, n ; cin >> m >> s >> n ;
    int size[n] ;
    for(int i = 0 ; i < n ; i++){
```

```
    cin >> size[i];
    }

    //initialising variable to count the number of swaps
required to sort :
    int inversions = 0 ;

    //sorting in the order of names using bubble sort and
finding the inversions:
    for(int i = 0 ; i < n ; i++){
    for(int j = i+1 ; j < n ; j++){
        if(size[i] > size[j]){
            swap(size[i], size[j]);
            inversions++;
        }
    }
    }

    //checking whether it is possible to do in m minutes or
not :
    (m*60 >= inversions*s)?cout << true : cout << false ;
    return 0 ;
}
```

Output :



Explanation :

Improvement : The time complexity can be improved using sorting and then counting the number of inversions using two pointers to O(nlogn)

Time complexity :

Worst Case : O(n^2) , Bubble sort is used to count the number of inversions which results in n^2 iterations in worst case.

Best Case : O(n) , if the array is already sorted , the best case complexity turns out to be O(n)

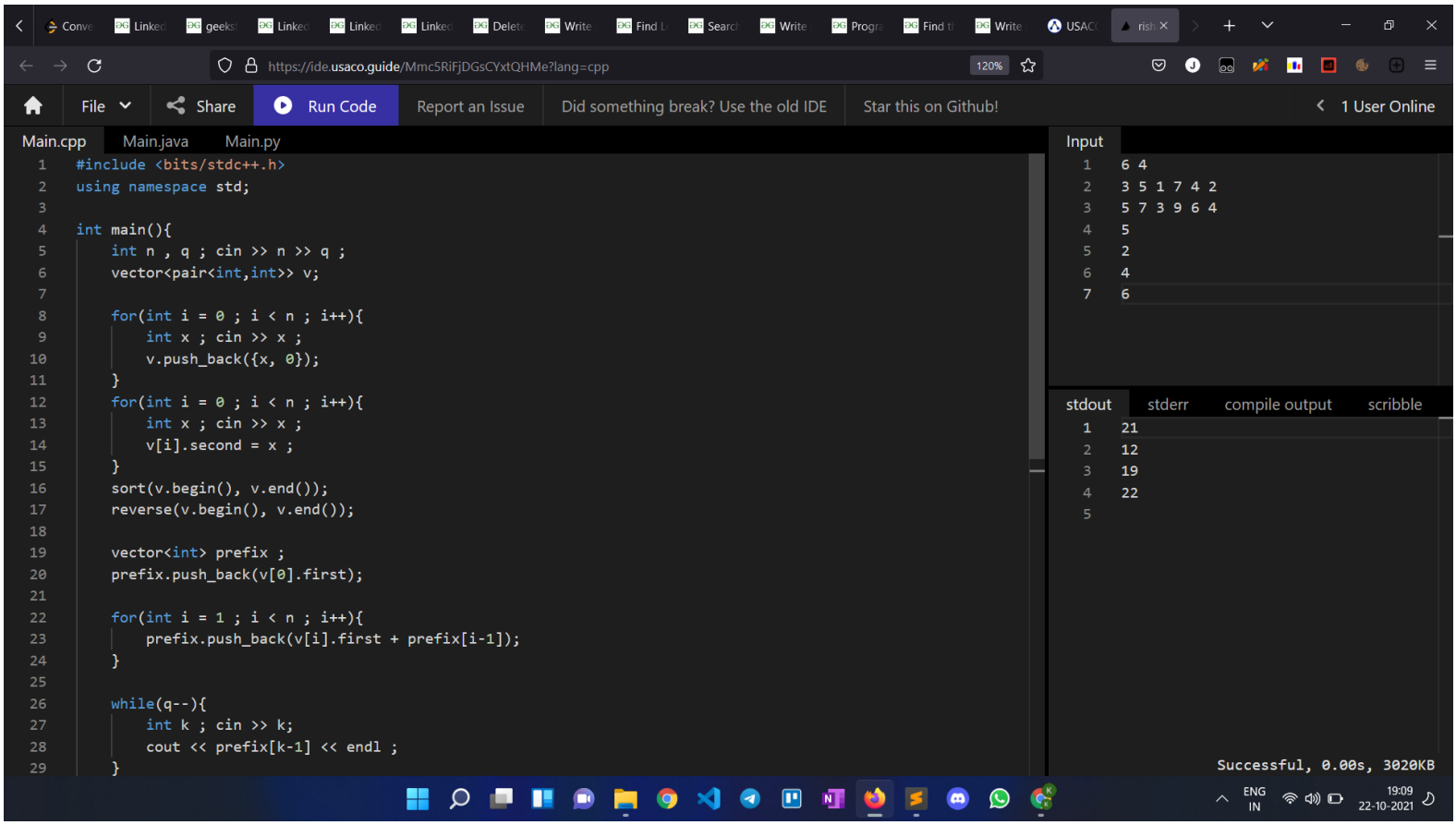
Q7.

Code :

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

int main(){
    //taking inputs :
    int n , q ; cin >> n >> q ;
    vector<pair<int,int>> v;

    for(int i = 0 ; i < n ; i++){
    int x ; cin >> x ;
    v.push_back({x, 0});
    }
    for(int i = 0 ; i < n ; i++){
    int x ; cin >> x ;
    v[i].second = x ;
    }

    //sorting the vector pair in descending order :
    sort(v.begin(), v.end());
    reverse(v.begin(), v.end());

    //prefix array to store running sum of all indices :
    vector<int> prefix ;
    prefix.push_back(v[0].first);

    for(int i = 1 ; i < n ; i++){
    prefix.push_back(v[i].first + prefix[i-1]);
    }

    //printing output for q queries :
    while(q--){
```

```
    int k ; cin >> k;
    cout << prefix[k-1] << endl ;
    }
}
```

Output :



Explanation :

Time complexity :

Worst Case : The best case time complexity is O(n + nlogn + q) = O(nlogn). The sorting of vector takes O(nlogn) time using inbuilt usage of sort which is made up of 3 sorting algos and called introsort.

Best Case : The worst case complexity is also O(nlog(n)) as sorting is always required.

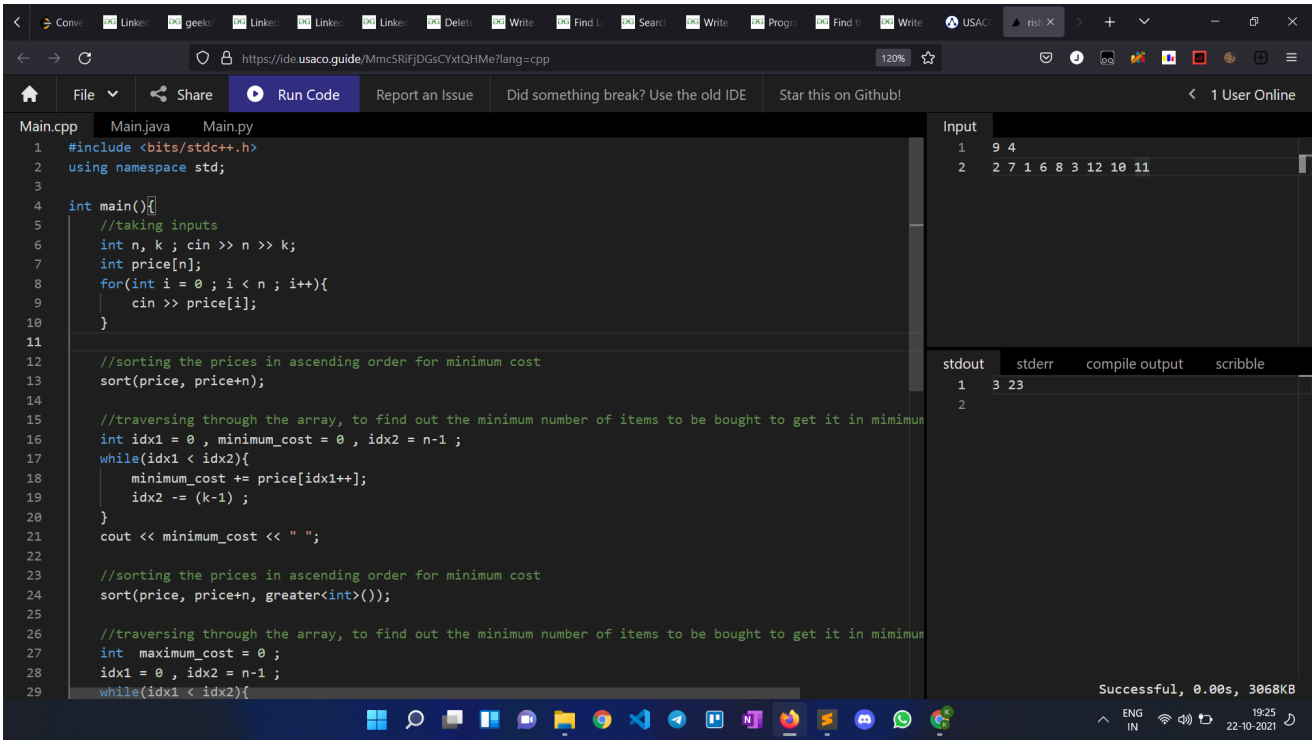
Q8.

Code :

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

int main(){
    //taking inputs
    int n, k ; cin >> n >> k;
    int price[n];
    for(int i = 0 ; i < n ; i++){
    cin >> price[i];
    }

    //sorting the prices in ascending order for minimum cost
    sort(price, price+n);

    //traversing through the array, to find out the minimum
number of items to be bought to get it in minimum cost
    int idx1 = 0 , minimum_cost = 0 , idx2 = n-1 ;
    while(idx1 < idx2){
    minimum_cost += price[idx1++];
    idx2 -= (k-1) ;
    }
    cout << minimum_cost << " ";

    //sorting the prices in ascending order for minimum cost
```

```
    sort(price, price+n, greater<int>());

    //traversing through the array, to find out the minimum
number of items to be bought to get it in minimum cost
    int  maximum_cost = 0 ;
    idx1 = 0 , idx2 = n-1 ;
    while(idx1 < idx2){
    maximum_cost += price[idx1++];
    idx2 -= (k-1) ;
    }
    cout << maximum_cost << "\n";

}
```

Output :



Explanation :

Time complexity :

Worst Case : The best case time complexity is O(n + nlogn) = O(nlogn). The sorting of vector takes O(nlogn) time using inbuilt usage of sort which is made up of 3 sorting algos and called introsort.

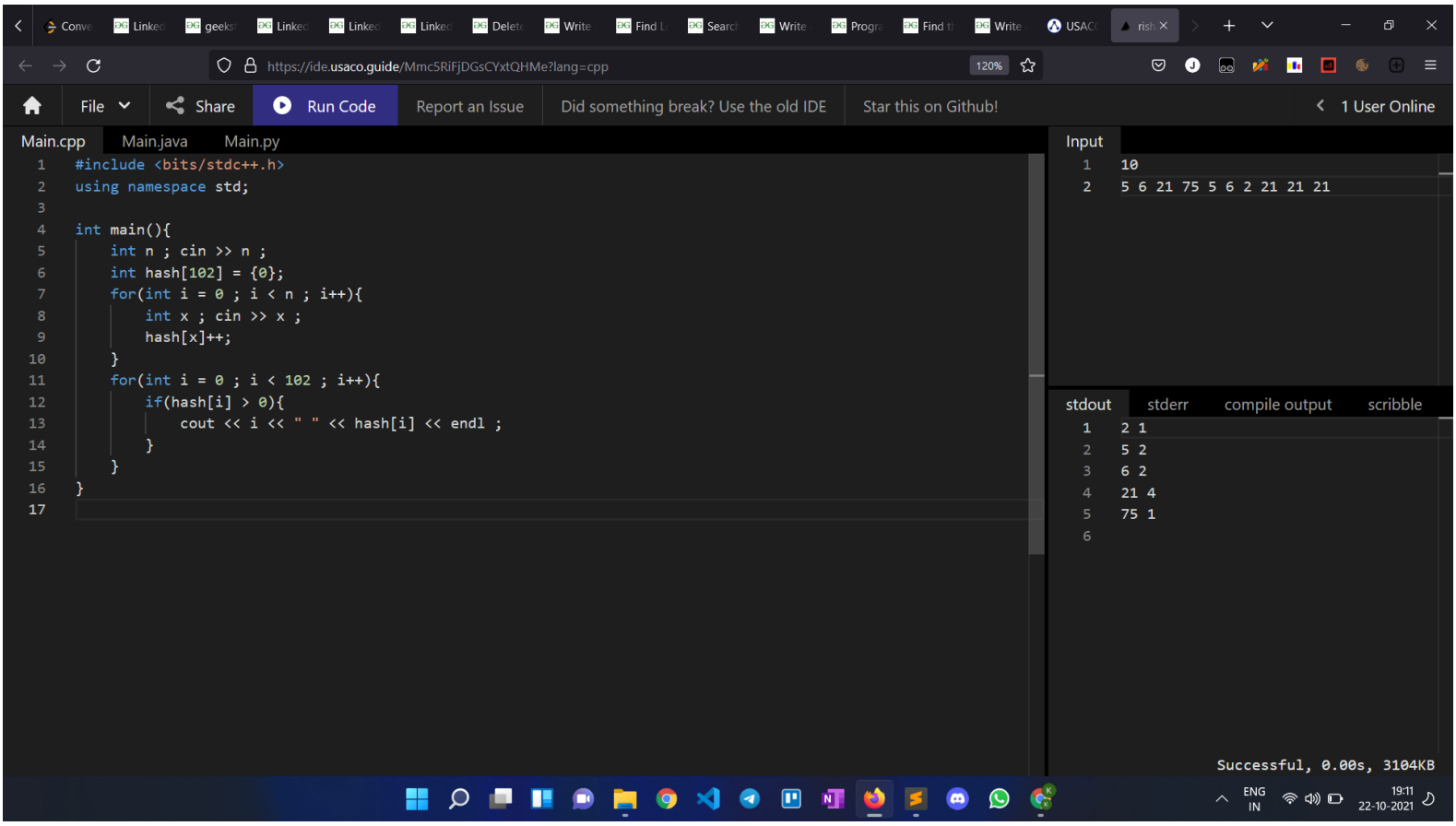
Best Case : The worst case complexity is also O(nlog(n)) as sorting is always required.

Q9.

Code :

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

int main(){
     int n ; cin >> n ;
     int hash[102] = {0};
     for(int i = 0 ; i < n ; i++){
     int x ; cin >> x ;
     hash[x]++;
     }
     for(int i = 0 ; i < 102 ; i++){
     if(hash[i] > 0){
          cout << i << " " << hash[i] << endl ;
     }
     }
}
```

Output :



Explanation :

Time complexity :

Worst Case : O(n) , The array is iterated once to make the hash array, resulting in linear complexity.

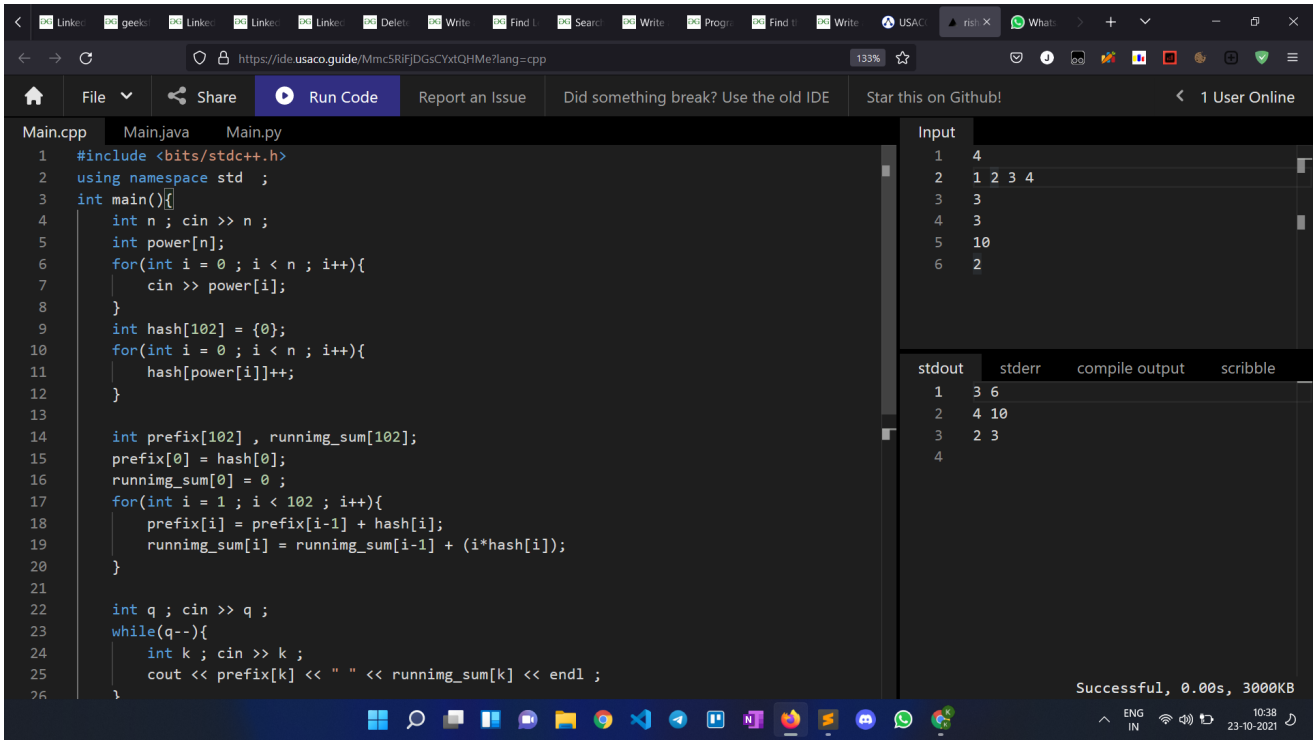
Best Case : O(n) , The array needs to be traversed at least once, so in the best case also , time complexity is O(n).

Q10.

Code :

```
#include <bits/stdc++.h>
using namespace std  ;
int main(){
     int n ; cin >> n ;
     int power[n];
     for(int i = 0 ; i < n ; i++){
     cin >> power[i];
     }
     int hash[102];
     for(int i = 0 ; i < n ; i++){
     hash[power[i]]++;
     }

     int prefix[102];
     prefix[0] = hash[0];
     for(int i = 1 ; i < 102 ; i++){
     prefix[i] = prefix[i-1] + hash[i];
     }

     int q ; cin >> q ;
     while(q--){
     int k ; cin >> k ;
     cout << prefix[k] << endl ;
     }
     return 0 ;
}
```

Output :



Explanation :

Algorithm : The frequency of all the elements of the array is stored in hash array and the running sum is calculated to find the number of elements smaller than any other number in the array, so the query can be returned by the precomputed result in O(1) time.

Time complexity :

Worst Case : O(n) , The hash array creation takes O(n) time and the query answering takes O(q) time , as in worst case q == n , worst case complexity is linear.

Best Case : O(n) , best case complexity turns out to be O(n) because the iteration of the array takes O(n) time.