

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
1  #include <bits/stdc++.h>
2  using namespace std;
3
4
5  void solve(){
6      //taking inputs :
7      int n ; cin >> n ;
8      string name[n], number[n];
9      for(int i = 0 ; i < n ; i++){
10         cin >> name[i] >> number[i];
11     }
12
13     //sorting in the order of names using bubble sort :
14     for(int i = 0 ; i < n ; i++){
15         for(int j = i+1 ; j < n ; j++){
16             if(name[i] > name[j]){
17                 swap(name[i], name[j]);
18                 swap(number[i], number[j]);
19             }
20         }
21     }
22
23     cout << "Ordered List \n";
24     for(int i = 0 ; i < n ; i++){
25         cout << name[i] << " " << number[i] << endl ;
26     }cout << endl ;
27
28     //for finding the name of the person with the given phone number :
29     string phone_number; cin >> phone_number ;
```

```
Input
1  10
2  Rahul 9598454222
3  Ashwin 7501202255
4  saleem 8545222522
5  Rithwik 7853266523
6  Anu 8832266636
7  Jancy 7852366336
8  Atul 7515555655
9  Jibin 9852453662
10 Jithin 7855656332
11 Stebin 8762556625
12 9598454222

stdout  stderr  compile output  scribble
1  Ordered List
2  Anu 8832266636
3  Ashwin 7501202255
4  Atul 7515555655
5  Jancy 7852366336
6  Jibin 9852453662
7  Jithin 7855656332
8  Rahul 9598454222
9  Rithwik 7853266523
10 Stebin 8762556625
11 saleem 8545222522
12
13 Name Telephone Number
14 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 $O(n \log n)$

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 :

Home

File

Share

Run Code

Report an Issue

Did something break? Use the old IDE

Star this on Github!

1 User Online

Main.cppMain.javaMain.py

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4
5 void solve(){
6     int n ; cin >> n ;
7     int hash[102] = {0};
8     for(int i = 0 ; i < n ; i++){
9         int x ; cin >> x ;
10        hash[x]++;
11    }
12    for(int i = 0 ; i < 102 ; i++){
13        if(hash[i] == 1){
14            cout << i ;
15            return ;
16        }
17    }
18 }
19
20 int main()
21 {
22     ios_base::sync_with_stdio(false);
23     cin.tie(NULL);
24
25     int t ; cin >> t ;
26     while(t--){
27         solve();
28         cout << endl ;
29     }
30 }
```

Input

1 2
2 5
3 1 1 2 5 5
4 7
5 2 2 5 5 20 30 30

stdoutstderrcompile outputscribble

1 2
2 20
3

Successful, 0.00s, 3108KB

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. $a \oplus a = 0$, 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 :

```
#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:

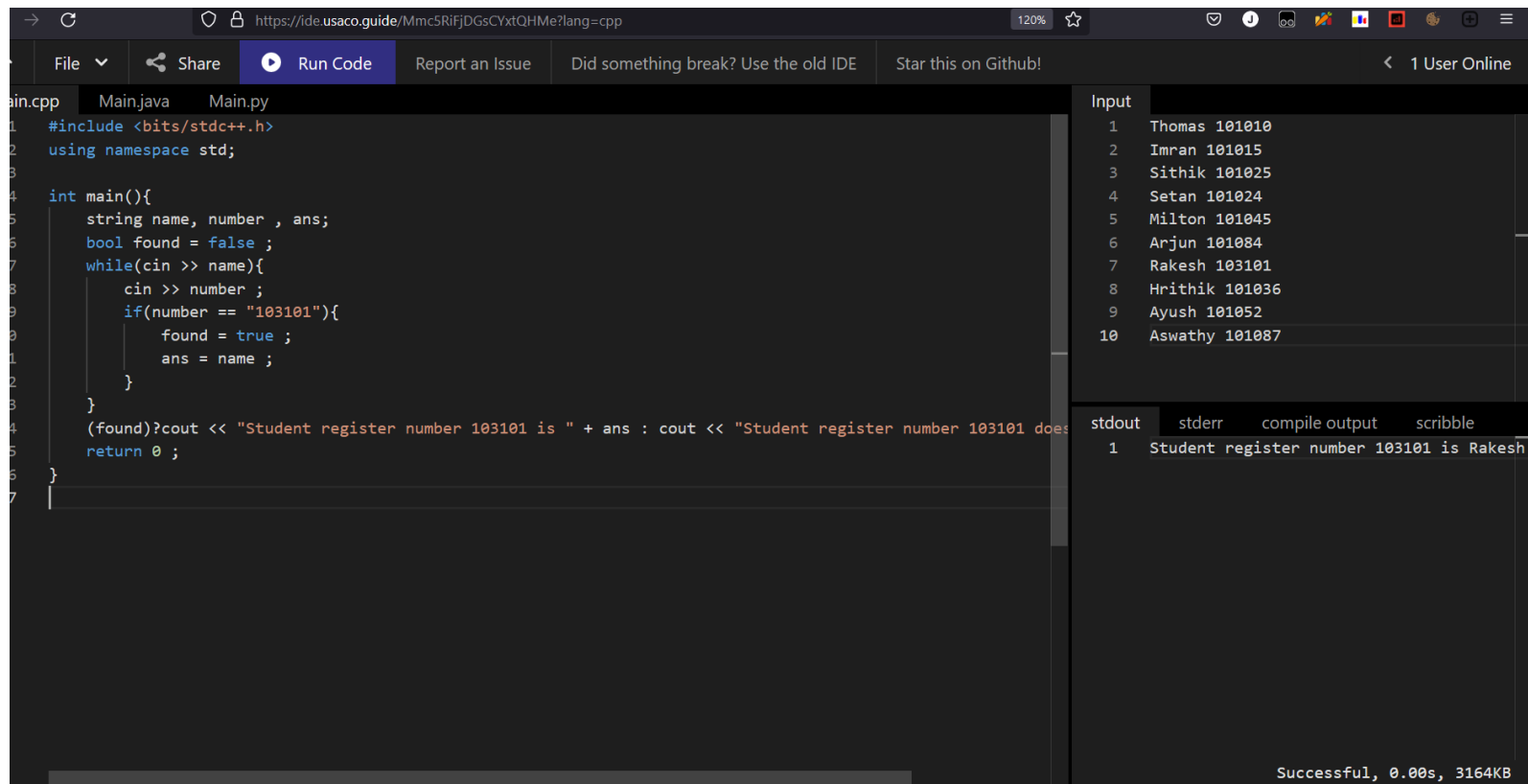
```
#include <bits/stdc++.h>
using namespace std;

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 :



The screenshot shows an online C++ IDE with the following components:

- File Explorer:** Main.cpp, Main.java, Main.py
- Code Editor:** Contains C++ code for searching a student by roll number.
- Input:** A list of 10 students with their names and roll numbers.
- Output:** The result of the search operation.
- Status Bar:** Shows 'Successful, 0.00s, 3164KB'.

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

int main(){
    string name, number , ans;
    bool found = false ;
    while(cin >> name){
        cin >> number ;
        if(number == "103101"){
            found = true ;
            ans = name ;
        }
    }
    (found)?cout << "Student register number 103101 is " + ans : cout << "Student register number 103101 does not exist" ;
    return 0 ;
}
```

Input:

Index	Name	Roll Number
1	Thomas	101010
2	Imran	101015
3	Sithik	101025
4	Setan	101024
5	Milton	101045
6	Arjun	101084
7	Rakesh	103101
8	Hrithik	101036
9	Ayush	101052
10	Aswathy	101087

Output:

```
1 Student register number 103101 is Rakesh
```

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 required 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 :

The screenshot shows an online IDE interface. The top bar includes navigation icons, a search bar, and a URL: <https://ide.usaco.guide/MmcSRiFJDGsCYxtQHMe?lang=cpp>. Below the bar are tabs for 'Main.cpp', 'Main.java', and 'Main.py'. The 'Main.cpp' tab is active, displaying the following code:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     int n ; cin >> n ;
7     int arr[n];
8     for(int i = 0 ; i < n ; i++){
9         cin >> arr[i];
10    }
11
12    if(n <= 2){
13        cout << "Total number of elements is less than 3";
14        return 0;
15    }
16    for(int i = 0 ; i < 3 ; i++){
17        int greater = INT_MIN , index;
18        for(int j = i+1 ; j < n ; j++){
19            if(greater < arr[j]){
20                greater = arr[j];
21                index = j ;
22            }
23        }
24        swap(arr[i], arr[index]);
25    }
26    cout << "The third largest element in the array is " << arr[2];
27    return 0 ;
28 }
29

```

To the right of the code editor is the 'Input' section, which contains the following input:

```

1 6
2 1 14 2 16 10 20

```

Below the input section is the 'stdout' section, which shows the output of the program:

```

1 The third largest element in the array is 14

```

At the bottom right of the IDE, a status bar indicates: 'Successful, 0.00s, 3112KB'. The Windows taskbar is visible at the very bottom of the screen.

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 :

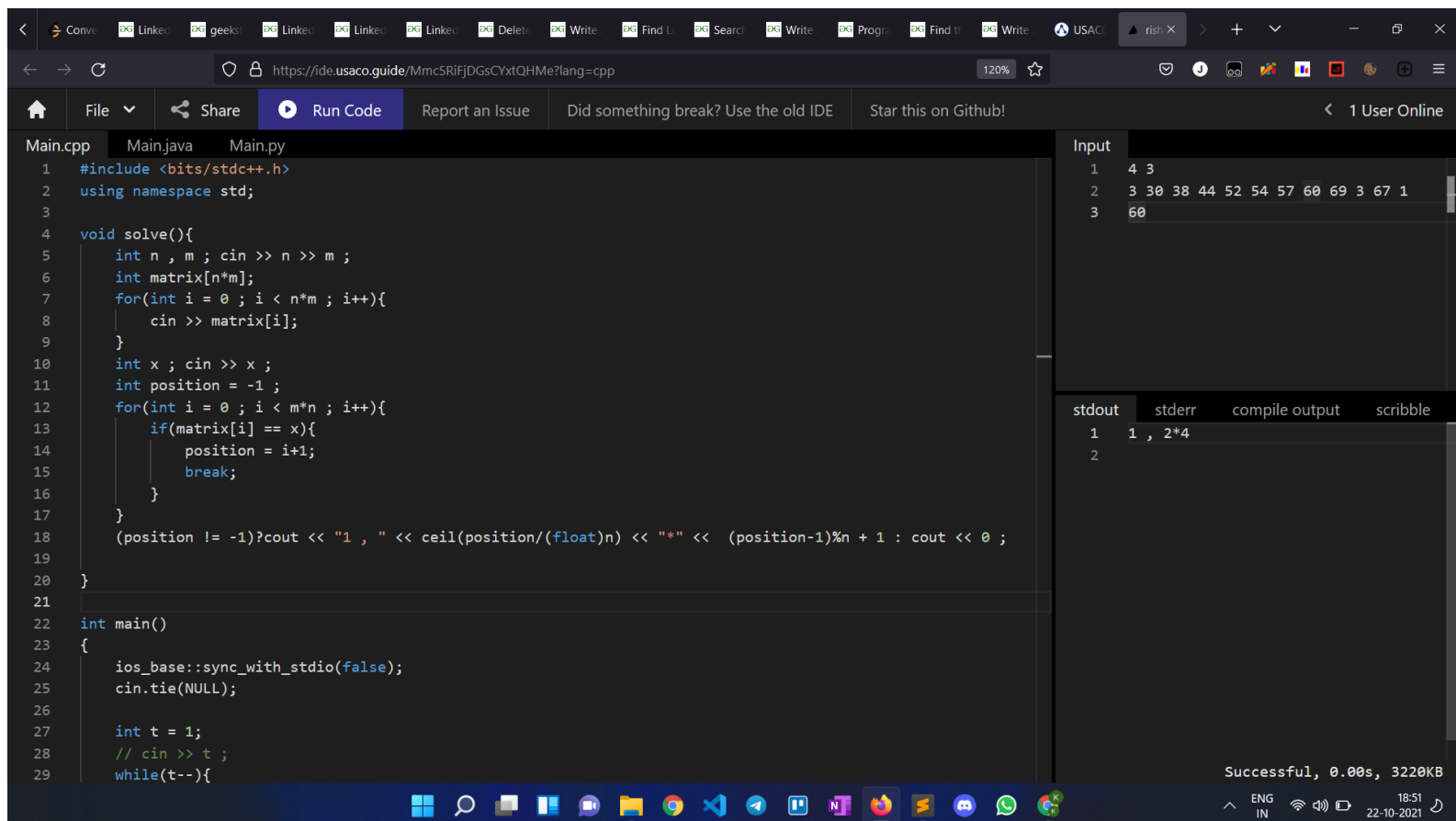
Algorithm : As the array given is always sorted , binary search can be applied to find if the key exists or not in $O(\log(n))$

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 $O(\log n)$

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 :



The screenshot shows an online C++ IDE interface. The main editor displays a C++ program for linear search. The code reads the dimensions of a matrix (n rows and m columns), reads the matrix elements, reads a target value x, and searches for x in the matrix. If found, it prints the row and column indices; otherwise, it prints 0. The program is executed, and the output is shown in the right-hand pane.

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void solve(){
5     int n , m ; cin >> n >> m ;
6     int matrix[n*m];
7     for(int i = 0 ; i < n*m ; i++){
8         cin >> matrix[i];
9     }
10    int x ; cin >> x ;
11    int position = -1 ;
12    for(int i = 0 ; i < m*n ; i++){
13        if(matrix[i] == x){
14            position = i+1;
15            break;
16        }
17    }
18    (position != -1)?cout << "1 , " << ceil(position/(float)n) << " " << (position-1)%n + 1 : cout << 0 ;
19
20 }
21
22 int main()
23 {
24     ios_base::sync_with_stdio(false);
25     cin.tie(NULL);
26
27     int t = 1;
28     // cin >> t ;
29     while(t--){
```

Input

1	4 3
2	3 30 38 44 52 54 57 60 69 3 67 1
3	60

stdout

1	1 , 2*4
2	

stderr

compile output

scribble

Successful, 0.00s, 3220KB

18:51
22-10-2021

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 :

The screenshot shows an online C++ IDE with the following components:

- Editor:** Contains a C++ program (Main.cpp) that reads an array of integers, sorts it using bubble sort, and counts the number of inversions. The code is as follows:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4
5 int main(){
6     //taking inputs :
7     int m, s, n ; cin >> m >> s >> n ;
8     int size[n] ;
9     for(int i = 0 ; i < n ; i++){
10         cin >> size[i];
11     }
12
13     //initialising variable to count the number of swaps required to sort :
14     int inversions = 0 ;
15
16     //sorting in the order of names using bubble sort and finding the inversions:
17     for(int i = 0 ; i < n ; i++){
18         for(int j = i+1 ; j < n ; j++){
19             if(size[i] > size[j]){
20                 swap(size[i], size[j]);
21                 inversions++;
22             }
23         }
24     }
25
26     //checking whether it is possible to do in m minutes or not :
27     (m*60 >= inversions*s)?cout << true : cout << false ;
28     return 0 ;
29 }
```
- Input:** A text area on the right showing two lines of input:

```
1 10 30 10
2 48 14 37 29 30 47 11 23 25 8
```
- Output:** A text area on the right showing the result:

```
1 0
```
- Status Bar:** At the bottom right, it says "Successful, 0.00s, 3092KB".

Explanation :

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

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:

```
#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 :

The screenshot shows an online IDE with the following components:

- Code Editor:** Contains C++ code for finding the k-th largest element in an array. It uses a vector to store elements, sorts them, and then iterates to find the k-th largest element.
- Input:** A text area with the following input:


```

      1 6 4
      2 3 5 1 7 4 2
      3 5 7 3 9 6 4
      4 5
      5 2
      6 4
      7 6
      
```
- Output:** A text area showing the output of the program:


```

      1 21
      2 12
      3 19
      4 22
      5
      
```
- Status Bar:** Indicates "Successful, 0.00s, 3020KB".

Explanation :

Time complexity :

Worst Case : The best case time complexity is $O(n + n \log n + q)$ = $O(n \log n)$. The sorting of vector takes $O(n \log n)$ 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 $O(n\log(n))$ 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
```

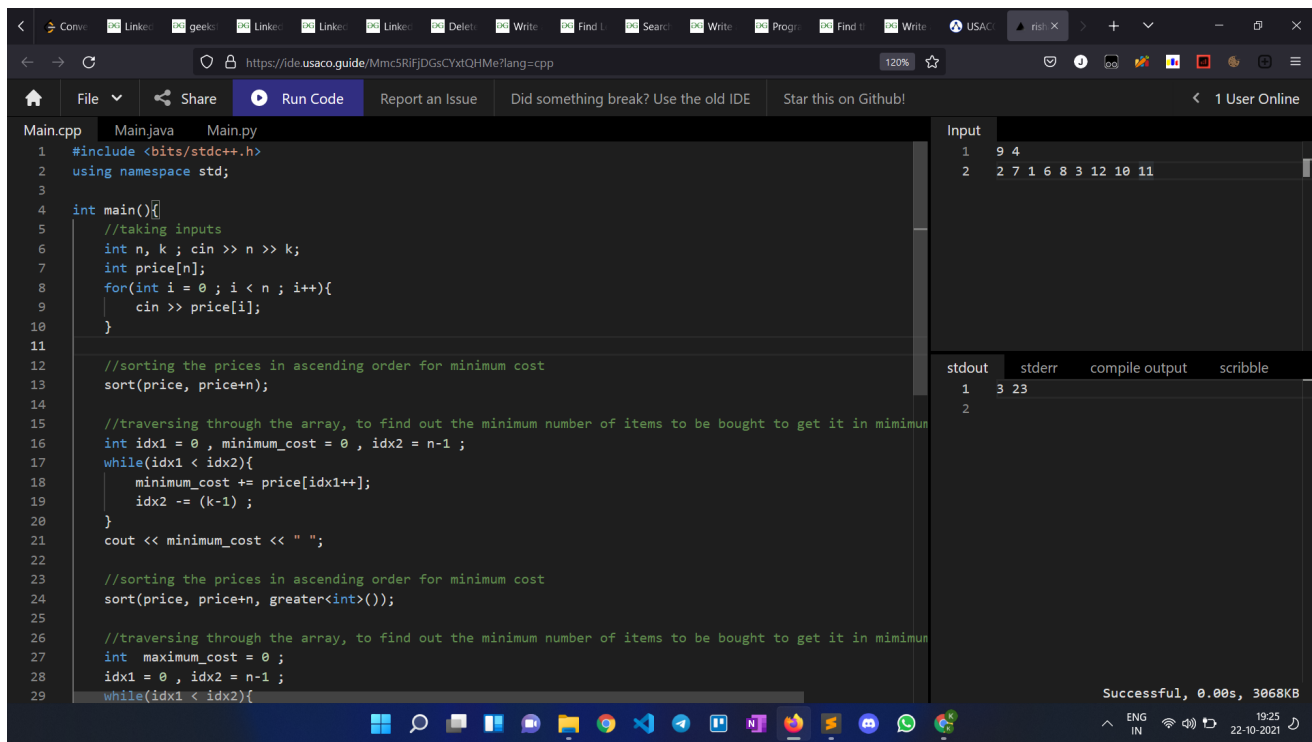
```
sort(price, price+n, greater<int>());
```

```
//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 :



The screenshot shows an online C++ IDE interface. The main editor displays a C++ program that reads an array of prices and a value k, then sorts the array and calculates the minimum cost by traversing the sorted array. The code is as follows:

```
1 #include <bits/stdc++.h>  
2 using namespace std;  
3  
4 int main(){  
5     //taking inputs  
6     int n, k ; cin >> n >> k;  
7     int price[n];  
8     for(int i = 0 ; i < n ; i++){  
9         cin >> price[i];  
10    }  
11  
12    //sorting the prices in ascending order for minimum cost  
13    sort(price, price+n);  
14  
15    //traversing through the array, to find out the minimum number of items to be bought to get it in minimum  
16    int idx1 = 0 , minimum_cost = 0 , idx2 = n-1 ;  
17    while(idx1 < idx2){  
18        minimum_cost += price[idx1++];  
19        idx2 -= (k-1) ;  
20    }  
21    cout << minimum_cost << " ";  
22  
23    //sorting the prices in ascending order for minimum cost  
24    sort(price, price+n, greater<int>());  
25  
26    //traversing through the array, to find out the minimum number of items to be bought to get it in minimum  
27    int maximum_cost = 0 ;  
28    idx1 = 0 , idx2 = n-1 ;  
29    while(idx1 < idx2){
```

The right sidebar shows the input and output. The input is:

```
1 9 4  
2 2 7 1 6 8 3 12 10 11
```

The output is:

```
1 3 23  
2
```

The status bar at the bottom indicates "Successful, 0.00s, 3068KB".

Explanation :

Time complexity :

Worst Case : The best case time complexity is $O(n + n \log n) = O(n \log n)$. The sorting of vector takes $O(n \log n)$ 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 $O(n \log(n))$ 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 :

The screenshot shows an online C++ IDE with the following components:

- Code Editor:** Contains a C++ program that reads an array of size n and counts the frequency of each element using a hash array of size 102. The code is as follows:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int n ; cin >> n ;
6     int hash[102] = {0};
7     for(int i = 0 ; i < n ; i++){
8         int x ; cin >> x ;
9         hash[x]++;
10    }
11    for(int i = 0 ; i < 102 ; i++){
12        if(hash[i] > 0){
13            cout << i << " " << hash[i] << endl ;
14        }
15    }
16 }
17
```
- Input:** The input consists of two lines:

```
1 10
2 5 6 21 75 5 6 2 21 21 21
```
- Output:** The output shows the frequency of each element in the array:

```
1 2 1
2 5 2
3 6 2
4 21 4
5 75 1
6
```
- Status:** The execution was successful, taking 0.00s and using 3104KB of memory.

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.

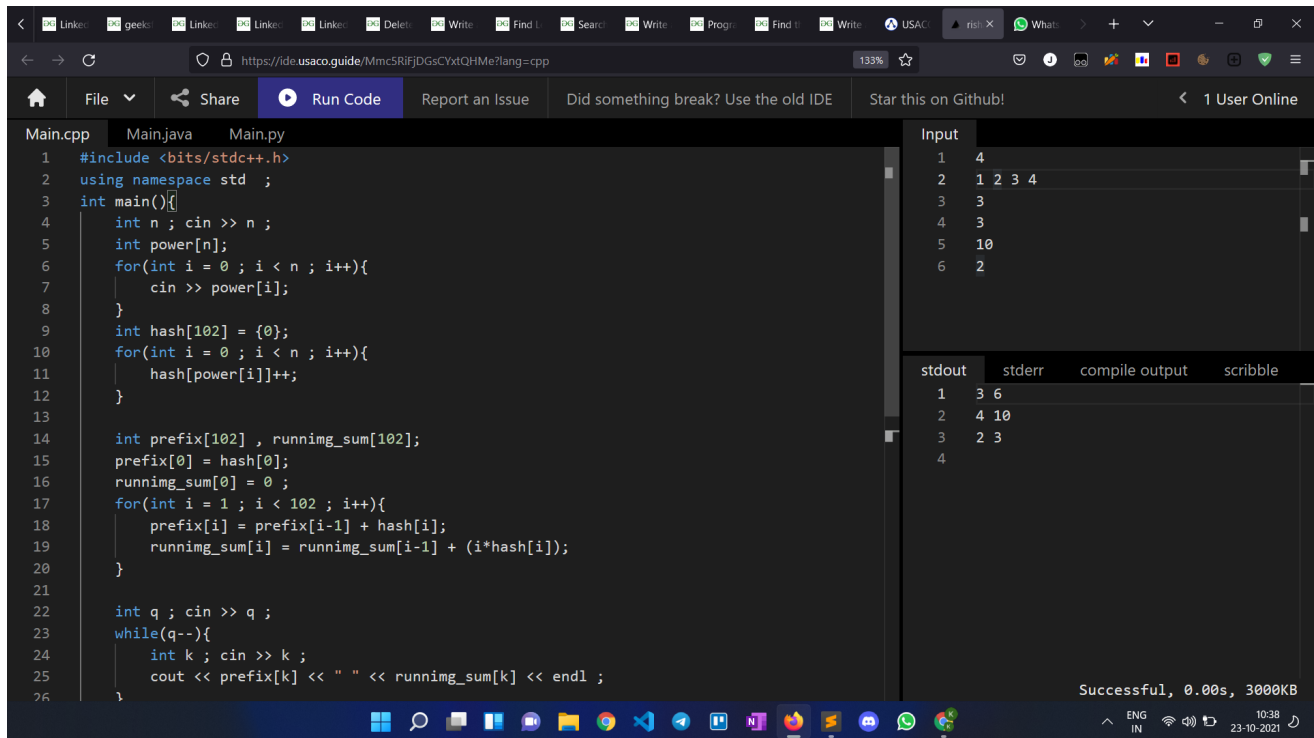
Code:

```
#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 :



The screenshot shows an online C++ IDE with the following components:

- Code Editor:** Contains C++ code for calculating prefix sums and frequency counts.
- Input:** A text area with the following input:

```
1 4
2 1 2 3 4
3 3
4 3
5 10
6 2
```
- Output:** A text area showing the program's output:

```
1 3 6
2 4 10
3 2 3
4
```
- Status Bar:** Indicates "Successful, 0.00s, 3000KB".

```
1 #include <bits/stdc++.h>
2 using namespace std ;
3 int main(){
4     int n ; cin >> n ;
5     int power[n];
6     for(int i = 0 ; i < n ; i++){
7         cin >> power[i];
8     }
9     int hash[102] = {0};
10    for(int i = 0 ; i < n ; i++){
11        hash[power[i]]++;
12    }
13
14    int prefix[102] , running_sum[102];
15    prefix[0] = hash[0];
16    running_sum[0] = 0 ;
17    for(int i = 1 ; i < 102 ; i++){
18        prefix[i] = prefix[i-1] + hash[i];
19        running_sum[i] = running_sum[i-1] + (i*hash[i]);
20    }
21
22    int q ; cin >> q ;
23    while(q--){
24        int k ; cin >> k ;
25        cout << prefix[k] << " " << running_sum[k] << endl ;
26    }
```

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.