TECHNICAL TRAINING DSA- CODING PRACTICE PROBLEMS

Name: Sachin A

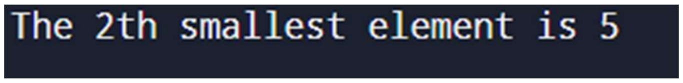
Dept: CSBS

Date: 11-11-2024

Kth smallest element:

```
#include <iostream>
using namespace std;

int kthSmallest(int arr[], int n, int k) {
   int max_element = arr[0];
   for (int i = 1; i < n; i++) {
      if (arr[i] > max_element) {
         max_element = arr[i];
      }
   }

   int freq[max_element + 1] = {0};
   for (int i = 0; i < n; i++) {
      freq[arr[i]]++;
   }

   int count = 0;
   for (int i = 0; i <= max_element; i++) {
      if (freq[i] != 0) {
         count += freq[i];
         if (count >= k) {
                return i;
            }
        }
    }
    return -1;
}

int main() {
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    cout << "The " << k << "th smallest element is " << kthSmallest(arr, n, k) << endl;

    return 0;
}
```

OUTPUT:

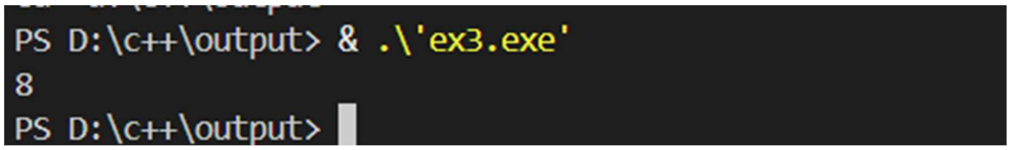


```
The 2th smallest element is 5
```

minimize the heights ii

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

int minimizeHeightDifference(vector<int> &heights, int offset) {
    int n = heights.size();
    sort(heights.begin(), heights.end());
    int minDiff = heights[n - 1] - heights[0];

    for (int i = 1; i < heights.size(); i++) {
        if (heights[i] - offset < 0)
            continue;

        int minHeight = min(heights[0] + offset, heights[i] - offset);
        int maxHeight = max(heights[i - 1] + offset, heights[n - 1] - offset);
        minDiff = min(minDiff, maxHeight - minHeight);
    }
    return minDiff;
}

int main() {
    int offset = 6;
    vector<int> heights = {12, 6, 4, 15, 17, 10};

    int result = minimizeHeightDifference(heights, offset);
    cout << result;

    return 0;
}
```

OUTPUT:



```
PS D:\c++\output> & .\'ex3.exe'
8
PS D:\c++\output>
```

parenthesis checker:

```
class Solution {
  public:
    bool isParenthesisBalanced(string& s) {
        stack<char>st;
        for(int i=0;i<s.length();i++){
            if(s[i]=='{' || s[i]=='[' || s[i]=='(') st.push(s[i]);
            else{
                if(!st.empty()&&
                   ((st.top()=='(' && s[i]==')') ||
                   (st.top()=='{' && s[i]=='}') || (st.top()=='[' && s[i]==']'))) st.pop();
                else return false;
            }
        }
        return st.empty();
    }
};
```

Equilibrium Point

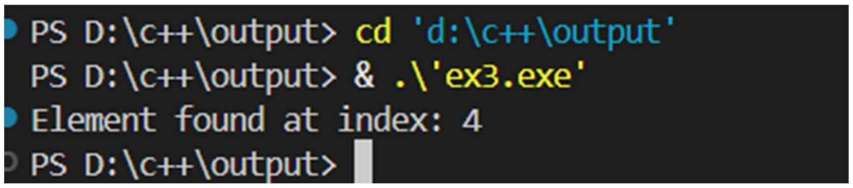
```
class Solution {
  public:
    // Function to find equilibrium point in the array.
    int equilibriumPoint(vector<int> &arr) {
        int n = arr.size();
        if (n == 1)
            return 1;
        int prefix[n] = { 0 };
        int suffix[n] = { 0 };

        prefix[0] = arr[0];
        suffix[n - 1] = arr[n - 1];

        for (int i = 1; i < n; i++) {
            prefix[i] = prefix[i - 1] + arr[i];
        }

        for (int i = n - 2; i >= 0; i--) {
            suffix[i] = suffix[i + 1] + arr[i];
        }

        for (int i = 0; i < n; i++) {
            if (prefix[i] == suffix[i]) {
                return i + 1;
            }
        }
        return -1;
    }
};
```

Binary search:

```
#include <iostream>
using namespace std;

int search(int arr[], int target, int low, int high) {
    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == target)
            return mid;

        if (arr[mid] < target)
```

```
          low = mid + 1;
        else
          high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14};
    int target = 10;
    int size = sizeof(arr) / sizeof(arr[0]);
    int result = search(arr, target, 0, size - 1);

    if (result != -1)
        cout << "Element found at index: " << result << endl;
    else
        cout << "Element not found." << endl;

    return 0;
}
```

OUTPUT:

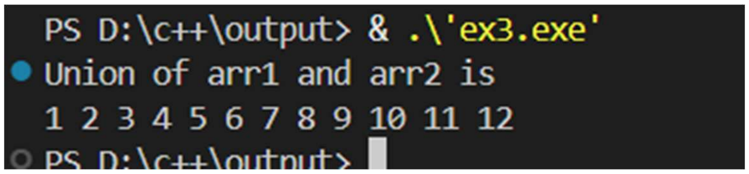


```
PS D:\c++\output> cd 'd:\c++\output'
PS D:\c++\output> & .\'ex3.exe'
Element found at index: 4
PS D:\c++\output>
```

union and intersection of two sorted arrays

```
#include <bits/stdc++.h>

using namespace std;

vector < int > FindUnion(int arr1[], int arr2[], int n, int m) {

  set < int > s;

  vector < int > Union;

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

    s.insert(arr1[i]);

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

    s.insert(arr2[i]);

  for (auto & it: s)

    Union.push_back(it);

  return Union;

}

int main(){

  int n = 10, m = 7;

  int arr1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  int arr2[] = {2, 3, 4, 4, 5, 11, 12};

  vector < int > Union = FindUnion(arr1, arr2, n, m);

  cout << "Union of arr1 and arr2 is  " << endl;

  for (auto & val: Union)

    cout << val << " ";

  return 0;

}
```

OUTPUT:



```
PS D:\c++\output> & .\'ex3.exe'
Union of arr1 and arr2 is
1 2 3 4 5 6 7 8 9 10 11 12
PS D:\c++\output>
```