Department of Computer Science and Engineering

Name:- Harleen Kaur UID:- 22BCS12125 Section:- 22KPIT-901-B

DAY-5

QUES 1: Given an integer k and array arr. Your task is to return the position of the first occurrence of k in the given array and if element k is not present in the array then return -1.

Example1:

Input: k = 16 , arr = [9, 7, 16, 16, 4]

Output: 3

Explanation: The value 16 is found in the given array at positions 3 and 4, with position 3 being the first occurrence.

Example2:

Input: k=98 , arr = [1, 22, 57, 47, 34, 18, 66]

Output: -1

Example2:

Input: k=9 , arr = [1, 22, 57, 47, 34, 9, 66]

Output: 6

Explanation: k = 98 isn't found in the given array.

Expected Time Complexity: O(n)

Expected Auxiliary Space: O(1)

Constraints:

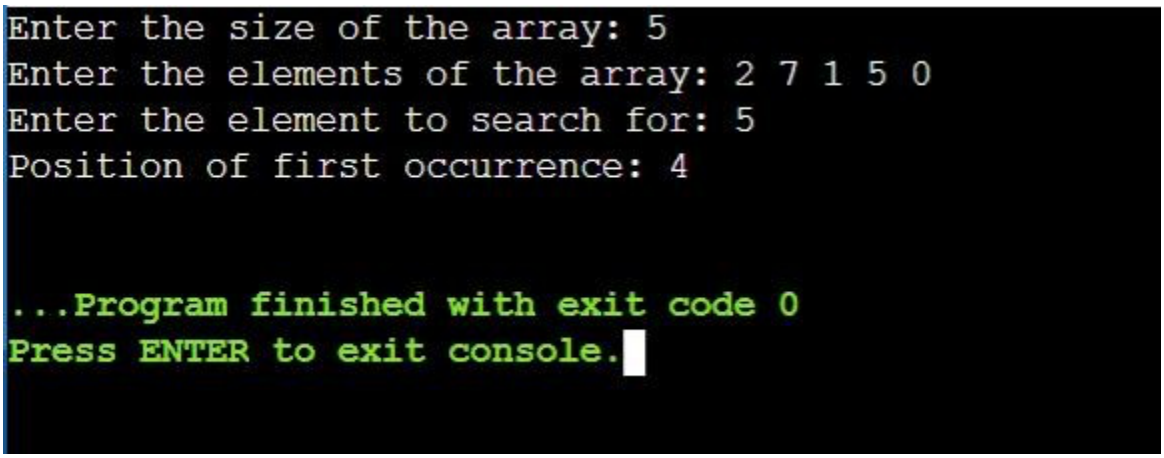
- $1 <= arr.size <= 10^6$
- $1 <= arr[i] <= 10^9$
- $1 <= k <= 10^6$

Program Code:-

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

int findFirstOccurrence(int k, const vector<int>& arr) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == k) {
            return i + 1;
        }
    }
    return -1;
}

int main() {
    int n, k;

    cout << "Enter the size of the array: ";
    cin >> n;

    vector<int> arr(n);

    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << "Enter the element to search for: ";
    cin >> k;

    int result = findFirstOccurrence(k, arr);
    cout << "Position of first occurrence: " << result << endl;
```

```
    return 0;
}
```

Output:-



QUES2: There are n available seats and n students standing in a room. You are given an array seats of length n, where seats[i] is the position of the ith seat. You are also given the array students of length n, where students[j] is the position of the jth student.
You may perform the following move any number of times: Increase or decrease the position of the ith student by 1 (i.e., moving the ith student from position x to x + 1 or x - 1)
Return the minimum number of moves required to move each student to a seat such that no two students are in the same seat.

Example 1:

Input: seats = [3,1,5], students = [2,7,4]
Output: 4
Explanation: The students are moved as follows:
- The first student is moved from position 2 to position 1 using 1 move.
- The second student is moved from position 7 to position 5 using 2 moves.
- The third student is moved from position 4 to position 3 using 1 move.
In total, 1 + 2 + 1 = 4 moves were used.

Example 2:

Input: seats = [4,1,5,9], students = [1,3,2,6]
Output: 7
Explanation: The students are moved as follows:
- The first student is not moved.
- The second student is moved from position 3 to position 4 using 1 move.
- The third student is moved from position 2 to position 5 using 3 moves.

- The fourth student is moved from position 6 to position 9 using 3 moves.
In total, 0 + 1 + 3 + 3 = 7 moves were used.

Program Code:-

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

int minMovesToSeat(vector<int>& seats, vector<int>& students) {
    sort(seats.begin(), seats.end());
    sort(students.begin(), students.end());

    int moves = 0;

    for (int i = 0; i < seats.size(); i++) {
        moves += abs(seats[i] - students[i]);
    }

    return moves;
}

int main() {
    int n;

    cout << "Enter the number of seats/students: ";
    cin >> n;

    vector<int> seats(n), students(n);

    cout << "Enter the positions of the seats: ";
    for (int i = 0; i < n; i++) {
        cin >> seats[i];
    }

    cout << "Enter the positions of the students: ";
    for (int i = 0; i < n; i++) {
        cin >> students[i];
    }

    int result = minMovesToSeat(seats, students);
    cout << "Minimum number of moves required: " << result << endl;

    return 0;
}
```

Output:-

```
Enter the number of seats/students: 3
Enter the positions of the seats: 4 1 5
Enter the positions of the students: 2 7 3
Minimum number of moves required: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

QUES 3: You are given an m x n integer matrix matrix with the following two properties:

Each row is sorted in non-decreasing order.

The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m * n)) time complexity.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

Output: true

Example2:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13

Output: false

Constraints:

m == matrix.length

n == matrix[i].length

$1 <= m, n <= 10^0$

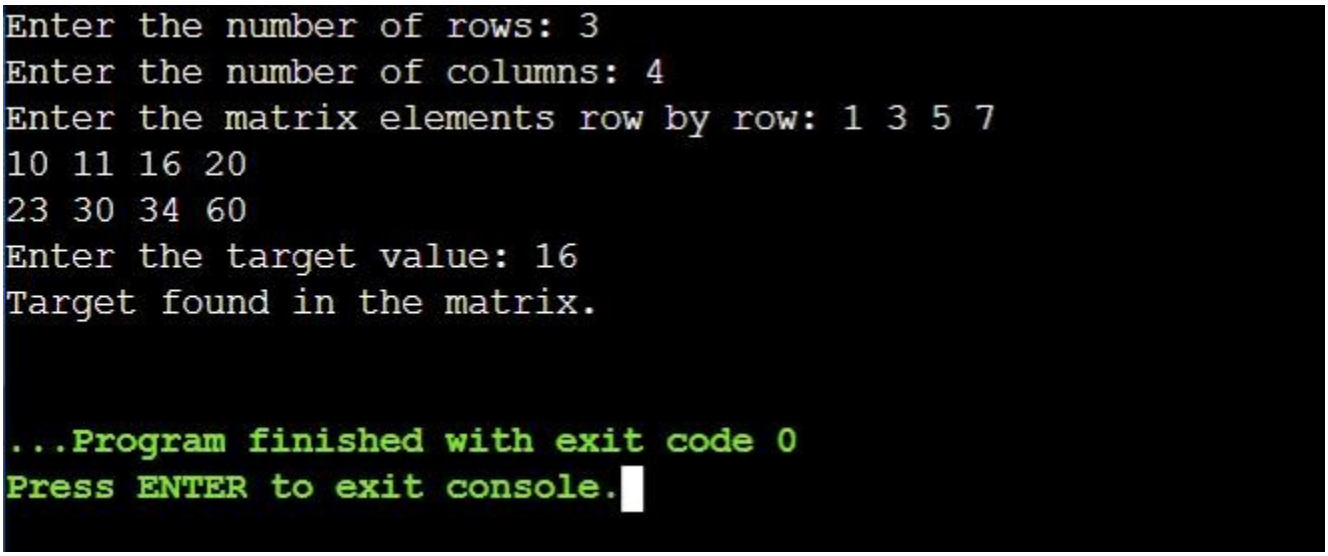
$-10^4 <= matrix[i][j], target <= 10^4$

Program Code:-

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

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size();
    int n = matrix[0].size();
    int left = 0, right = m * n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        int midValue = matrix[mid / n][mid % n]; // Map 1D index to 2D matrix

        if (midValue == target) {
            return true;
        } else if (midValue < target) {
```

```
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return false;
}

int main() {
    int m, n, target;

    cout << "Enter the number of rows: ";
    cin >> m;
    cout << "Enter the number of columns: ";
    cin >> n;

    vector<vector<int>> matrix(m, vector<int>(n));

    cout << "Enter the matrix elements row by row: ";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> matrix[i][j];
        }
    }

    cout << "Enter the target value: ";
    cin >> target;

    if (searchMatrix(matrix, target)) {
        cout << "Target found in the matrix." << endl;
    } else {
        cout << "Target not found in the matrix." << endl;
    }

    return 0;
}
```

Output:-



QUES 4: You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.

You are allowed to choose exactly one element from each row to form an array.

Return the kth smallest array sum among all possible arrays.

Example 1:

Input: mat = [[1,3,11],[2,4,6]], k = 5
Output: 7
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.

Example 2:

Input: mat = [[1,3,11],[2,4,6]], k = 9
Output: 17

Example 3:

Input: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7
Output: 9
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.

Constraints:

- m == mat.length
- n == mat.length[i]
- 1 <= m, n <= 40
- 1 <= mat[i][j] <= 5000
- 1 <= k <= min(200, n^m)
- mat[i] is a non-decreasing array.

Program Code:-

```
#include <iostream>
#include <vector>
#include <queue>
#include <functional>

using namespace std;

int kthSmallestSum(vector<vector<int>>& mat, int k) {
    int m = mat.size(), n = mat[0].size();
    auto cmp = [](const pair<int, vector<int>>& a, const pair<int, vector<int>>& b) {
        return a.first > b.first; // Min-heap based on sum
    };

    priority_queue<pair<int, vector<int>>, vector<pair<int, vector<int>>>, decltype(cmp)>
minHeap(cmp);

    vector<int> indices(m, 0);
    int initialSum = 0;
    for (int i = 0; i < m; ++i) {
        initialSum += mat[i][0];
    }
    minHeap.push({initialSum, indices});

    int count = 0;

    while (!minHeap.empty()) {
        pair<int, vector<int>> topElement = minHeap.top();
        minHeap.pop();

        int currentSum = topElement.first;
        vector<int> currentIndices = topElement.second;

        count++;
```

```
        if (count == k) {
            return currentSum;
        }

        for (int i = 0; i < m; ++i) {
            if (currentIndices[i] + 1 < n) {
                vector<int> newIndices = currentIndices;
                newIndices[i]++;
                int newSum = currentSum - mat[i][currentIndices[i]] + mat[i][newIndices[i]];
                minHeap.push({newSum, newIndices});
            }
        }
    }

    return -1;
}

int main() {
    int m, n, k;
    cout << "Enter the number of rows and columns in the matrix: ";
    cin >> m >> n;

    vector<vector<int>> mat(m, vector<int>(n));
    cout << "Enter the elements of the matrix: ";
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> mat[i][j];
        }
    }

    cout << "Enter the value of k: ";
    cin >> k;

    int result = kthSmallestSum(mat, k);
    cout << "The " << k << "th smallest array sum is: " << result << endl;

    return 0;
}
```

Output:-

```
Enter the number of rows and columns in the matrix: 2 3
Enter the elements of the matrix: 1 3 11
2 4 6
Enter the value of k: 5
The 5th smallest array sum is: 7


...Program finished with exit code 0
Press ENTER to exit console.
```

QUES 5: Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Example 1:

Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.

Example 2:

Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

- nums1.length == m
- nums2.length == n
- 0 <= m <= 1000
- 0 <= n <= 1000
- 1 <= m + n <= 2000
- -10^6 <= nums1[i], nums2[i] <= 10^6

Program Code:-

```
#include <iostream>
#include <vector>
#include <algorithm>
#include<climits>
using namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if (nums1.size() > nums2.size()) {
        return findMedianSortedArrays(nums2, nums1); // Ensure nums1 is the smaller array
    }

    int m = nums1.size();
    int n = nums2.size();
    int left = 0, right = m;
    int totalLeft = (m + n + 1) / 2;

    while (left <= right) {
        int partition1 = left + (right - left) / 2;
        int partition2 = totalLeft - partition1;

        int maxLeft1 = (partition1 == 0) ? INT_MIN : nums1[partition1 - 1];
        int minRight1 = (partition1 == m) ? INT_MAX : nums1[partition1];

        int maxLeft2 = (partition2 == 0) ? INT_MIN : nums2[partition2 - 1];
        int minRight2 = (partition2 == n) ? INT_MAX : nums2[partition2];

        if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {
            if ((m + n) % 2 == 0) {
                return (max(maxLeft1, maxLeft2) + min(minRight1, minRight2)) / 2.0;
            } else {
                return max(maxLeft1, maxLeft2);
            }
        } else if (maxLeft1 > minRight2) {
            right = partition1 - 1;
        } else {
            left = partition1 + 1;
        }
    }

    throw invalid_argument("Input arrays are not sorted.");
}
```

```
int main() {
    int m, n;

    cout << "Enter the size of the first array: ";
    cin >> m;
    vector<int> nums1(m);
    cout << "Enter the elements of the first array: ";
    for (int i = 0; i < m; i++) {
        cin >> nums1[i];
    }

    cout << "Enter the size of the second array: ";
    cin >> n;
    vector<int> nums2(n);
    cout << "Enter the elements of the second array: ";
    for (int i = 0; i < n; i++) {
        cin >> nums2[i];
    }

    double median = findMedianSortedArrays(nums1, nums2);
    cout << "The median of the two sorted arrays is: " << median << endl;

  return 0;
}
```

Output:-