DOMAIN WINTER CAMP

(Department of Computer Science and Engineering)

Name: Ankit Vashisth UID: 22BCS13378 Section: KPIT 901-B

DAY-5

(Easy)

Q1 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.

Note: 1-based indexing is followed here.

Example1:

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

Output: 3

Program code:

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

int main(){


    int n,k;
    cin>>k;
    cin>>n;
    int arr[n];

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

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

    int count=0;

    for(int j=0;j<n;j++){
        if(arr[j]==k){
            count++;
        }

    }
    cout<<count;
    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting.cpp -o
sorting && ./sorting
2
4
1 2 2 3
2
```

Q 2 (Medium) 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.

Program code:

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

```
bool searchMatrix(const vector<vector<int>>& matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) return false;

    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];

        if (midValue == target) {
            return true;
        } else if (midValue < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return false;
}

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

    // Input dimensions of the matrix
    cout << "Enter the number of rows: ";
    cin >> m;
    cout << "Enter the number of columns: ";
    cin >> n;

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

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

```
    // Input the target element
    cout << "Enter the target element: ";
    cin >> target;

    // Call the function and display the result
    bool result = searchMatrix(matrix, target);
    cout << (result ? "true" : "false") << endl;

    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting2.cpp -o
 sorting2 && ./sorting2
Enter the number of rows: 3
Enter the number of columns: 4
Enter the elements of the matrix row by row:
1 3 5 7
10 11 16 20
23 30 34 60
Enter the target element: 3
true
```

Ques 3. Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

Program Code:

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

// Function to perform binary search for the first or last occurrence
int binarySearch(const vector<int>& nums, int target, bool findFirst) {
    int left = 0, right = nums.size() - 1, result = -1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
```

```
        if (nums[mid] == target) {
            result = mid; // Record the index
            if (findFirst) {
                right = mid - 1; // Search in the left half for the first occurrence
            } else {
                left = mid + 1; // Search in the right half for the last occurrence
            }
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return result;
}

// Main function to find the starting and ending positions
vector<int> searchRange(const vector<int>& nums, int target) {
    int start = binarySearch(nums, target, true);  // Find the starting position
    if (start == -1) {
        return {-1, -1};  // If target is not found, return [-1, -1]
    }
    int end = binarySearch(nums, target, false); // Find the ending position
    return {start, end};
}

int main() {
    int n, target;

    // Input size of the array
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    // Input the elements of the array
    vector<int> nums(n);
    cout << "Enter the elements of the array in sorted order: ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }

    // Input the target element
```

```
    cout << "Enter the target element: ";
    cin >> target;

    // Call the function and display the result
    vector<int> result = searchRange(nums, target);
    cout << "[" << result[0] << ", " << result[1] << "]" << endl;

    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting3.cpp -o
 sorting3 && ./sorting3
Enter the number of elements in the array: 6
Enter the elements of the array in sorted order: 5 7 7 8
 8 10
Enter the target element: 8
[3, 4]
```

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.

Program Code:

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

struct Node {
    int sum;
    vector<int> indices;

    // Constructor
    Node(int sum, vector<int>& indices) : sum(sum), indices(indices) {}
```

```
};

// Comparator to compare Node by their sum
struct Compare {
    bool operator()(Node& n1, Node& n2) {
        return n1.sum > n2.sum;
    }
};

int kthSmallestSum(vector<vector<int>>& mat, int k) {
    int m = mat.size();
    int n = mat[0].size();

    // Min-Heap to store the current smallest sums
    priority_queue<Node, vector<Node>, Compare> pq;

    // Initialize the heap with the smallest sum of each row (first element of each row)
    vector<int> initialIndices(m, 0);
    int initialSum = 0;
    for (int i = 0; i < m; i++) {
        initialSum += mat[i][0];
    }

    pq.push(Node(initialSum, initialIndices));

    // Extract k smallest sums
    int count = 0;
    while (!pq.empty()) {
        Node currentNode = pq.top();
        pq.pop();

        count++;
        if (count == k) {
            return currentNode.sum;
        }

        // Try to move to the next element in each row, if possible
        for (int i = 0; i < m; i++) {
            if (currentNode.indices[i] + 1 < n) {
                vector<int> newIndices = currentNode.indices;
                newIndices[i]++;
```

```
            int newSum = currentNode.sum - mat[i][currentNode.indices[i]] +
mat[i][newIndices[i]];
            pq.push(Node(newSum, newIndices));
        }
    }
}

return -1; // If k is out of bounds
}

int main() {
    int m, n, k;
    cout << "Enter number of rows (m): ";
    cin >> m;
    cout << "Enter number of columns (n): ";
    cin >> n;
    vector<vector<int>> mat(m, vector<int>(n));
    cout << "Enter the matrix values:\n";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> mat[i][j];
        }
    }
    cout << "Enter k: ";
    cin >> k;

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

    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting4.cpp -o
 sorting4 && ./sorting4
Enter number of rows (m): 2
Enter number of columns (n): 3
Enter the matrix values:
1 3 11
2 4 6
Enter k: 5
The 5th smallest sum is: 7
```

Ques 5 You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Program Code:

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

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        // Min-Heap to store the nodes
        auto cmp = [](ListNode* a, ListNode* b) {
```

```
            return a->val > b->val; // Min-heap: smaller value has higher
priority
        };

        priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)>
pq(cmp);

        // Push the head of each list into the priority queue
        for (auto list : lists) {
            if (list) {
                pq.push(list);
            }
        }

        ListNode* dummy = new ListNode(0);
        ListNode* current = dummy;

        // While the priority queue is not empty, extract the smallest element
        while (!pq.empty()) {
            ListNode* node = pq.top();
            pq.pop();

            current->next = node;
            current = current->next;

            // If the extracted node has a next node, push it into the queue
            if (node->next) {
                pq.push(node->next);
            }
        }

        return dummy->next;
    }
};

// Helper function to print the linked list
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " -> ";
        head = head->next;
    }
    cout << "NULL" << endl;
```

```
}

// Helper function to create a linked list from a vector of integers
ListNode* createList(const vector<int>& vals) {
    if (vals.empty()) return nullptr;
    ListNode* head = new ListNode(vals[0]);
    ListNode* current = head;
    for (int i = 1; i < vals.size(); i++) {
        current->next = new ListNode(vals[i]);
        current = current->next;
    }
    return head;
}

int main() {
    vector<ListNode*> lists;

    // Create multiple linked lists
    lists.push_back(createList({1, 4, 5}));
    lists.push_back(createList({1, 3, 4}));
    lists.push_back(createList({2, 6}));

    Solution sol;
    ListNode* mergedList = sol.mergeKLists(lists);

    // Print the merged linked list
    printList(mergedList);

    return 0;
}
```

Output:

```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ sorting5.cpp -o
 sorting5 && ./sorting5
1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6 -> NULL
```