Name : Tamanna **UID : 22BCS13649**

Section : 22BCS_IOT_615-B **Assignment : 5**

Searching and Sorting QUESTIONS :-

Very Easy (Questions 1–5)

1.Searching a Number

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.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;


int searchNumber(int k, vector<int>& arr) {

    for (int i = 0; i < arr.size(); ++i) {

        if (arr[i] == k) {

            return i + 1; // 1-based indexing

        }

    }

    return -1; // Element not found

}

int main() {

    int k = 16;

    vector<int> arr = {9, 7, 16, 16, 4};
```

```
    int position = searchNumber(k, arr);
    cout << "Position: " << position << endl;
    return 0;
}
```

OUTPUT:

```
Position: 3
```

2.Sorted array Search.

Given an array, arr[] sorted in ascending order and an integer k. Return true if k is present in the array, otherwise, false.

Example 1:

Input: arr[] = [1,2,3,4,6], k=6

Output: true

Explanation: Since, 6 is present in the array at index4 (0-based indexing), Output is true.

CODE:

```
#include <iostream>
#include <vector>
using namespace std;
bool searchElement(const vector<int>& arr, int k) {
    int left = 0;
    int right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == k) {
```

```
            return true;
        }

        if (arr[mid] < k) {
            left = mid + 1;
        }
                else {
            right = mid - 1;
        }
    }
    return false;
}
int main() {
    vector<int> arr1 = {1, 2, 3, 4, 6};
    int k1 = 6;
    cout << "Output: " << (searchElement(arr1, k1) ? "true" : "false") << endl;
    vector<int> arr2 = {1, 2, 4, 5, 6};
    int k2 = 3;
    cout << "Output: " << (searchElement(arr2, k2) ? "true" : "false") << endl;
    return 0;
}
```

OUTPUT:

3. Find Target Indices After Sorting Array.

You are given a 0-indexed integer array nums and a target element target.

A target index is an index i such that nums[i] == target.

Return a list of the target indices of nums after sorting nums in non-decreasing order. If there are no target indices, return an empty list. The returned list must be sorted in increasing order.

CODE:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;



vector<int> targetIndicesAfterSorting(vector<int>& nums, int target) {

    // Sort the array

    sort(nums.begin(), nums.end());



    vector<int> result;



    // Iterate through the sorted array and find the target

    for (int i = 0; i < nums.size(); ++i) {

        if (nums[i] == target) {

            result.push_back(i);

        }

    }



    return result;
```

```
}

int main() {
    vector<int> nums1 = {1, 2, 5, 2, 3};
    int target1 = 2;
    vector<int> result1 = targetIndicesAfterSorting(nums1, target1);
    cout << "Target Indices: ";
    for (int index : result1) {
        cout << index << " ";
    }
    cout << endl;

    vector<int> nums2 = {4, 5, 6, 7};
    int target2 = 3;
    vector<int> result2 = targetIndicesAfterSorting(nums2, target2);
    cout << "Target Indices: ";
    for (int index : result2) {
        cout << index << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Target Indices: 1 2
```

Easy

1. Minimum Number of Moves to Seat Everyone

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 totalMoves = 0;
        for (int i = 0; i < seats.size(); ++i) {
        totalMoves += abs(seats[i] - students[i]);
    }
        return totalMoves;
}
int main() {
    vector<int> seats = {3, 1, 5};
    vector<int> students = {2, 7, 4};
        int result = minMovesToSeat(seats, students);
        cout << "Minimum moves required: " << result << endl;
        return 0;
```

}

OUTPUT:

```
Minimum moves required: 4
```

2.Squares of a Sorted Array

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> sortedSquares(vector<int>& nums) {
    int n = nums.size();
    vector<int> result(n);
        int left = 0, right = n - 1, index = n - 1;
        while (left <= right) {
        int leftSquare = nums[left] * nums[left];
        int rightSquare = nums[right] * nums[right];
                if (leftSquare > rightSquare) {
            result[index--] = leftSquare;
            left++;
        } else {
            result[index--] = rightSquare;
            right--;
        }
    }
        return result;
```

```
}


int main() {
    vector<int> nums = {-4, -1, 0, 3, 10};
    vector<int> result = sortedSquares(nums);
        cout << "Sorted squares: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;
        return 0;
}
```

OUTPUT:

```
Sorted squares: 0 1 9 16 100
```

3. Common in 3 Sorted Arrays.

CODE:

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


vector<int> findCommonElements(vector<int>& arr1, vector<int>& arr2, vector<int>&
arr3) {
    int i = 0, j = 0, k = 0;
    vector<int> result;
```

```
    while (i < arr1.size() && j < arr2.size() && k < arr3.size()) {
        if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {
            if (result.empty() || result.back() != arr1[i]) {
                result.push_back(arr1[i]);
            }
            i++;
            j++;
            k++;
        }
                else if (arr1[i] < arr2[j]) {
            i++;
        }
        else if (arr2[j] < arr3[k]) {
            j++;
        }
        else {
            k++;
        }
    }

    if (result.empty()) {
        result.push_back(-1);
    }
        return result;
}
```

```
int main() {
    vector<int> arr1 = {1, 5, 10, 20, 40, 80};
    vector<int> arr2 = {6, 7, 20, 80, 100};
    vector<int> arr3 = {3, 4, 15, 20, 30, 70, 80, 120};
        vector<int> commonElements = findCommonElements(arr1, arr2, arr3);
       for (int num : commonElements) {
        cout << num << " ";
    }
    cout << endl;
     return 0;
}
```

OUTPUT:

```
20 80
```

Medium

1. Search in 2D Matrix.

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 low = 0, high = m * n - 1;
        while (low <= high) {
        int mid = low + (high - low) / 2;
        int row = mid / n;
        int col = mid % n;
                if (matrix[row][col] == target) {
            return true;
        } else if (matrix[row][col] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
        return false;
}
int main() {
    vector<vector<int>> matrix = {{1, 3, 5, 7}, {10, 11, 16, 20}, {23, 30, 34, 60}};
    int target = 3;
        bool result = searchMatrix(matrix, target);
    cout << (result ? "true" : "false") << endl;
        return 0;
}
```

OUTPUT:

```
true
```

2.Find First and Last Position of Element in Sorted Array.

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


vector<int> searchRange(vector<int>& nums, int target) {
    vector<int> result = {-1, -1};

    // Lambda function to find the left or right bound
    auto findBound = [&](bool findFirst) {
        int left = 0, right = nums.size() - 1;
        int bound = -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                bound = mid;
                if (findFirst) {
                    right = mid - 1;  // Look on the left half
                } else {
                    left = mid + 1;   // Look on the right half
                }
            } else if (nums[mid] < target) {
                left = mid + 1;  // Move right
            } else {
                right = mid - 1; // Move left
```

```
            }
        }
        return bound;
    };

    result[0] = findBound(true);  // Find the first occurrence
    if (result[0] != -1) {
        result[1] = findBound(false); // Find the last occurrence
    }
    return result;
}

int main() {
    vector<int> nums = {5, 7, 7, 8, 8, 10};
    int target = 8;
    vector<int> result = searchRange(nums, target);
    cout << "[" << result[0] << ", " << result[1] << "]" << endl; // Output the range
    return 0;
}
```

OUTPUT:

```
[3, 4]
```

13. Find Minimum in Rotated Sorted Array.

CODE:

```
#include <iostream>
```

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

int findMin(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;

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

        if (nums[mid] > nums[right]) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }
    return nums[left];
}

int main() {
    vector<int> nums = {3, 4, 5, 1, 2};
    cout << findMin(nums) << endl;
    return 0;
}
```

Hard (Questions 16–20)

1. Sort Items by Groups Respecting Dependencies

There are n items each belonging to zero or one of m groups where group[i] is the group that the i-th item belongs to and it's equal to -1 if the i-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it. Return a sorted list of the items such that: The items that belong to the same group are next to each other in the sorted list. There are some relations between these items where beforeItems[i] is a list containing all the items that should come before the i-th item in the sorted array (to the left of the i-th item). Return any solution if there is more than one solution and return an empty list if there is no solution.

CODE:

```
#include <iostream>

#include <vector>

#include <unordered_map>

#include <queue>

using namespace std;



vector<int> topologicalSort(int n, unordered_map<int, vector<int>>& graph, vector<int>& indegree) {

    queue<int> q;

    vector<int> order;



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

        if (indegree[i] == 0) {

            q.push(i);

        }

    }



    while (!q.empty()) {

        int node = q.front();

        q.pop();
```

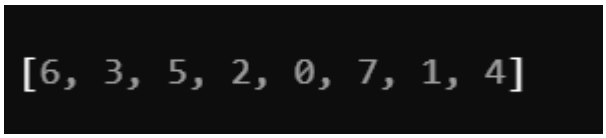
```
            order.push_back(node);

            for (int neighbor : graph[node]) {
                --indegree[neighbor];
                if (indegree[neighbor] == 0) {
                    q.push(neighbor);
                }
            }
        }

        if (order.size() == n) {
            return order;
        }

        return {}; // return empty if there's a cycle or invalid
    }

    vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>& beforeItems) {
        for (int i = 0; i < n; ++i) {
            if (group[i] == -1) {
                group[i] = m++; // Assign new groups to ungrouped items
            }
        }

        unordered_map<int, vector<int>> itemGraph;
```

```
unordered_map<int, vector<int>> groupGraph;
vector<int> itemIndegree(n, 0);
vector<int> groupIndegree(m, 0);

for (int i = 0; i < n; ++i) {
    for (int before : beforeItems[i]) {
        itemGraph[before].push_back(i);
        ++itemIndegree[i];

        if (group[i] != group[before]) {
            groupGraph[group[before]].push_back(group[i]);
            ++groupIndegree[group[i]];
        }
    }
}

vector<int> itemOrder = topologicalSort(n, itemGraph, itemIndegree);
vector<int> groupOrder = topologicalSort(m, groupGraph, groupIndegree);

if (itemOrder.empty() || groupOrder.empty()) {
    return {};
}

unordered_map<int, vector<int>> groupedItems;
for (int item : itemOrder) {
```

```
            groupedItems[group[item]].push_back(item);
        }

        vector<int> result;
        for (int grp : groupOrder) {
            for (int item : groupedItems[grp]) {
                result.push_back(item);
            }
        }

        return result;
    }

int main() {
    int n = 8, m = 2;
    vector<int> group = {-1, -1, 1, 0, 0, 1, 0, -1};
    vector<vector<int>> beforeItems = {{}, {6}, {5}, {6}, {3, 6}, {}, {}, {}};

    vector<int> result = sortItems(n, m, group, beforeItems);
    if (result.empty()) {
        cout << "[]" << endl;
    } else {
        cout << "[";
        for (size_t i = 0; i < result.size(); ++i) {
            cout << result[i];
```

```
            if (i < result.size() - 1) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
    }
    return 0;
}
```

OUTPUT:



2. Find the Kth Smallest Sum of a Matrix With Sorted Rows.

CODE:

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

struct Node {
    int sum;
    vector<int> indices;
    bool operator>(const Node &other) const {
        return sum > other.sum;
```

```
    }
};

int kthSmallest(vector<vector<int>> &mat, int k) {
    int m = mat.size(), n = mat[0].size();
    priority_queue<Node, vector<Node>, greater<Node>> minHeap;
    vector<int> initialIndices(m, 0);
    int initialSum = 0;

    for (int i = 0; i < m; ++i) {
        initialSum += mat[i][0];
    }

    minHeap.push(Node{initialSum, initialIndices});
    set<vector<int>> visited;
    visited.insert(initialIndices);

    for (int count = 0; count < k - 1; ++count) {
        Node currentNode = minHeap.top();
        minHeap.pop();

        for (int i = 0; i < m; ++i) {
            if (currentNode.indices[i] + 1 < n) {
                vector<int> newIndices = currentNode.indices;
                newIndices[i]++;
```

```
            if (visited.find(newIndices) == visited.end()) {
                visited.insert(newIndices);
                int newSum = currentNode.sum - mat[i][currentNode.indices[i]] + mat[i][newIndices[i]];
                minHeap.push(Node{newSum, newIndices});
            }
        }
    }
}

return minHeap.top().sum;
}

int main() {
    vector<vector<int>> mat = {{1, 3, 11}, {2, 4, 6}};
    int k = 5;
    int result = kthSmallest(mat, k);
    cout << "The " << k << "th smallest sum is: " << result << endl;
    return 0;
}
```

OUTPUT:

```
The 5th smallest sum is: 17
```

3. Merge k Sorted Lists.

CODE:

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

struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

struct compare {
    bool operator()(ListNode *a, ListNode *b) {
        return a->val > b->val;  // Min-heap
    }
};

ListNode *mergeKLists(vector<ListNode *> &lists) {
    priority_queue<ListNode *, vector<ListNode *>, compare> minHeap;
    for (ListNode *list : lists) {
        if (list != nullptr) {
            minHeap.push(list);
        }
    }
```

```
    ListNode *dummy = new ListNode(0);  // Dummy node to start the merged list
    ListNode *current = dummy;

    while (!minHeap.empty()) {
        ListNode *node = minHeap.top();
        minHeap.pop();
        current->next = node;
        current = current->next;

        if (node->next != nullptr) {
            minHeap.push(node->next);  // Push the next node from the same list
        }
    }

    return dummy->next;  // Return the merged list starting from the first node
}

void printList(ListNode *head) {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}
```

```
int main() {
    ListNode *l1 = new ListNode(1);
    l1->next = new ListNode(4);
    l1->next->next = new ListNode(5);

    ListNode *l2 = new ListNode(1);
    l2->next = new ListNode(3);
    l2->next->next = new ListNode(4);

    ListNode *l3 = new ListNode(2);
    l3->next = new ListNode(6);

    vector<ListNode *> lists = {l1, l2, l3};

    ListNode *mergedList = mergeKLists(lists);
    printList(mergedList);  // Output: 1 1 2 3 4 4 5 6

    return 0;
}
```

OUTPUT:

Very Hard

1.Find Minimum in Rotated Sorted Array II.

CODE:

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

int findMin(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] > nums[right]) {
            left = mid + 1;
        } else if (nums[mid] < nums[right]) {
            right = mid;
        } else {
            right--;  // Handle the case where nums[mid] == nums[right]
        }
    }
    return nums[left];
}

int main() {
    vector<int> nums = {1, 3, 5};
    cout << "The minimum element is: " << findMin(nums) << endl;
    return 0;
}
```

OUTPUT:

```
The minimum element is: 1
```

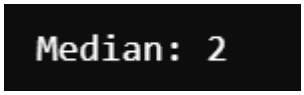
2.Median of Two Sorted Arrays.

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()) {

        swap(nums1, nums2);

    }

    int m = nums1.size();

    int n = nums2.size();

    int left = 0, right = m;


    while (left <= right) {

        int partition1 = left + (right - left) / 2;

        int partition2 = (m + n + 1) / 2 - 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 == 1) {
                return max(maxLeft1, maxLeft2);
            } else {
                return (max(maxLeft1, maxLeft2) + min(minRight1, minRight2)) / 2.0;
            }
        } else if (maxLeft1 > minRight2) {
            right = partition1 - 1;
        } else {
            left = partition1 + 1;
        }
    }

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

int main() {
    vector<int> nums1 = {1, 3};
    vector<int> nums2 = {2};
    cout << "Median: " << findMedianSortedArrays(nums1, nums2) << endl;
    return 0;
}
```

OUTPUT:



3.Create Sorted Array through Instructions.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;


const int MOD = 1e9 + 7;


class FenwickTree {

public:

    FenwickTree(int size) : bit(size + 1, 0) {}


    void update(int index, int value) {

        for (; index < bit.size(); index += index & -index) {

            bit[index] += value;

        }

    }


    int query(int index) {

        int sum = 0;

        for (; index > 0; index -= index & -index) {

            sum += bit[index];
```

```
        }
        return sum;
    }

private:
    vector<int> bit;
};

int createSortedArray(vector<int>& instructions) {
    int max_val = 100000;
    FenwickTree fenwick(max_val);
    long long total_cost = 0;

    for (int i = 0; i < instructions.size(); ++i) {
        int current = instructions[i];
        int less_than_current = fenwick.query(current - 1);
        int greater_than_current = i - less_than_current;
        total_cost += min(less_than_current, greater_than_current);
        total_cost %= MOD;
        fenwick.update(current, 1);
    }

    return total_cost;
}
```

```
int main() {
    vector<int> instructions = {1, 5, 6, 2};
    cout << "Total Cost: " << createSortedArray(instructions) << endl; // Output: 1
    return 0;
}
```

OUTPUT: