DAY-5 ASSIGNMENT

Student Name: Harshit Bansal   **UID: 22BCS15323**
Branch: BE-CSE   **Section/Group: 620 - A**
Date of Performance:26/12/24

Winter Winning Camp

Problem 1

Aim:

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.

Code:

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

int findFirstOccurrence(const vector<int>& arr, int k) {
    for (int i = 0; i < arr.size(); ++i) {
        if (arr[i] == k) {
            return i; // Return the position of the first occurrence
        }
    }
    return -1; // Return -1 if k is not found
}

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 value of k: ";
        cin >> k;

        int result = findFirstOccurrence(arr, k);
        if (result != -1) {
            cout << "The first occurrence of " << k << " is at position: " << result
    << endl;
        } else {
            cout << k << " is not present in the array." << endl;
        }

        return 0;
    }
```

Output:

```
Enter the size of the array: 5
Enter the elements of the array: 2
3
6
5
4
Enter the value of k: 6
The first occurrence of 6 is at position: 2
```

Problem 2

Aim:

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.

Code:

```cpp
#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 and 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 << "The minimum number of moves required is: " << result << endl;

    return 0;
```

Output:

```
Enter the number of seats and students: 4
Enter the positions of the seats: 1
3
5
2
Enter the positions of the students: 7
4
7
8
The minimum number of moves required is: 15
```

Problem 3

Aim:
Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

Code:

```cpp
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> sortedSquares(vector<int>& nums) {
```

```cpp
    int n = nums.size();

    vector<int> result(n);

    int left = 0, right = n - 1;

    int 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() {

    int n;

    cout << "Enter the size of the array: ";

    cin >> n;

    vector<int> nums(n);

    cout << "Enter the elements of the array (sorted in non-decreasing order): ";
```

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

        cin >> nums[i];

    }

    vector<int> result = sortedSquares(nums);

    cout << "The sorted squares of the array are: ";

    for (int num : result) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the array (sorted in non-decreasing order): -3
-1
2
4
5
The sorted squares of the array are: 1 4 9 16 25
```

Problem 4

Aim: You are given three arrays sorted in increasing order. Find the elements that are common in all three arrays.

Code:

```cpp
#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; // Pointers for arr1, arr2, arr3

    vector<int> common;

    while (i < arr1.size() && j < arr2.size() && k < arr3.size()) {

        if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {

            common.push_back(arr1[i]);

            i++; j++; k++;

        }

        else if (arr1[i] < arr2[j]) {

            i++;

        } else if (arr2[j] < arr3[k]) {

            j++;

        } else {

            k++;
```

```cpp
        }

    }

    return common;

}

int main() {

    int n1, n2, n3;

    cout << "Enter the size of the first array: ";

    cin >> n1;

    cout << "Enter the size of the second array: ";

    cin >> n2;

    cout << "Enter the size of the third array: ";

    cin >> n3;

    vector<int> arr1(n1), arr2(n2), arr3(n3);

    cout << "Enter elements of the first array: ";

    for (int i = 0; i < n1; ++i) cin >> arr1[i];

    cout << "Enter elements of the second array: ";

    for (int i = 0; i < n2; ++i) cin >> arr2[i];

    cout << "Enter elements of the third array: ";

    for (int i = 0; i < n3; ++i) cin >> arr3[i];

    vector<int> result = findCommonElements(arr1, arr2, arr3);

    if (result.empty()) {

        cout << "No common elements found." << endl;

    } else {

        cout << "Common elements are: ";
```

```
        for (int num : result) {

            cout << num << " ";

        }

        cout << endl;

    }

    return 0;

}
```

Output:

```
Enter the size of the first array: 4
Enter the size of the second array: 4
Enter the size of the third array: 4
Enter elements of the first array: 3
5
6
7
Enter elements of the second array: 2
4
2
7
Enter elements of the third array: 8
4
6
9
No common elements found.
```

Problem 5

Aim:

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.

Code:

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

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

   int m = matrix.size();    // Number of rows
   int n = matrix[0].size(); // Number of columns
   int left = 0, right = m * n - 1;

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

      if (midValue == target) {
         row = mid / n;  // Calculate row index
         col = mid % n;  // Calculate column index
         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 and columns: ";
   cin >> m >> n;

   vector<vector<int>> matrix(m, vector<int>(n));
   cout << "Enter the elements of the matrix row by row:" << endl;
```

```
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;

int row = -1, col = -1;
if (searchMatrix(matrix, target, row, col)) {
    cout << "Target found at position: (" << row << ", " << col << ")" << endl;
} else {
    cout << "Target not found in the matrix." << endl;
}

return 0;
```

Output:

```
Enter the number of rows and columns: 3
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 value: 16
Target found at position: (1, 2)
```

Problem 6

Aim: 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].

Code:

```cpp
#include <vector>

using namespace std;

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

    auto findBound = [&](bool isFirst) -> int {

        int left = 0, right = nums.size() - 1;

        while (left <= right) {

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

            if (nums[mid] == target) {

                if (isFirst) {

                    if (mid == 0 || nums[mid - 1] < target) {

                        return mid;

                    }

                    right = mid - 1;

                } else {

                    if (mid == nums.size() - 1 || nums[mid + 1] > target) {

                        return mid;
```

```cpp
            }

                left = mid + 1;

            }

        } else if (nums[mid] < target) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }

    return -1;

};

int start = findBound(true);

if (start == -1) {

    return {-1, -1};

}

int end = findBound(false);

return {start, end};

}

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;
```

```
    return 0;

}
```

Output:


```
[3, 4]
```

Problem 7

Aim: You are given an integer array arr[]. Your task is to find the smallest positive number missing from the array.

Code:

```cpp
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int fiIstMissingPositive(vector<int>& nums) {

    int n = nums.size();

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

        if (nums[i] <= 0 || nums[i] > n) {

            nums[i] = n + 1;

        }

    }

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

        int num = abs(nums[i]);
```
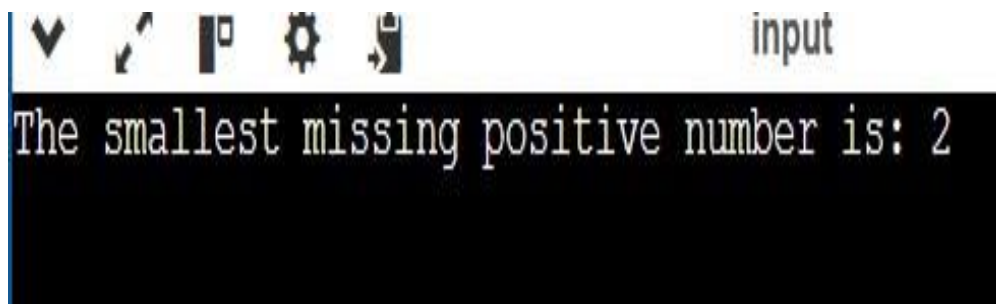
```cpp
            if (num <= n && nums[num - 1] > 0) {

                nums[num - 1] = -nums[num - 1]; // Mark it as found by making it
negative

            }

        }

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

        if (nums[i] > 0) {

            return i + 1; // i+1 is the smallest missing positive

        }

    }    return n + 1;

}

int main() {

    vector<int> arr = {3, 4, -1, 1};

    cout << "The smallest missing positive number is: " <<
firstMissingPositive(arr) << endl;

    return 0;
}
```

Output:

Problem 8

Aim:

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:

```cpp
#include <iostream>

#include <vector>

#include <queue>

#include <set>

using namespace std;

struct Node {

    int sum;

    vector<int> indices;
```

```cpp
        Node(int sum, vector<int> indices) : sum(sum), indices(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();

        vector<int> indices(m, 0);

        int initialSum = 0;

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

            initialSum += mat[i][0];

        }

        priority_queue<Node, vector<Node>, greater<Node>> minHeap;

        minHeap.push(Node(initialSum, indices));

        set<vector<int>> visited;

        visited.insert(indices);

        int count = 0;

        while (!minHeap.empty()) {

            Node current = minHeap.top();

            minHeap.pop();
```

```cpp
            count++;

        if (count == k) {

            return current.sum;

        }

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

            if (current.indices[i] + 1 < n) {

                vector<int> newIndices = current.indices;

                newIndices[i]++;

                if (visited.find(newIndices) == visited.end()) {

                    visited.insert(newIndices);

                    int newSum = current.sum - mat[i][current.indices[i]] +
    mat[i][newIndices[i]];

                    minHeap.push(Node(newSum, newIndices));

                }

            }

        }

    }

    return -1;

}

int main() {

    vector<vector<int>> mat = {
```
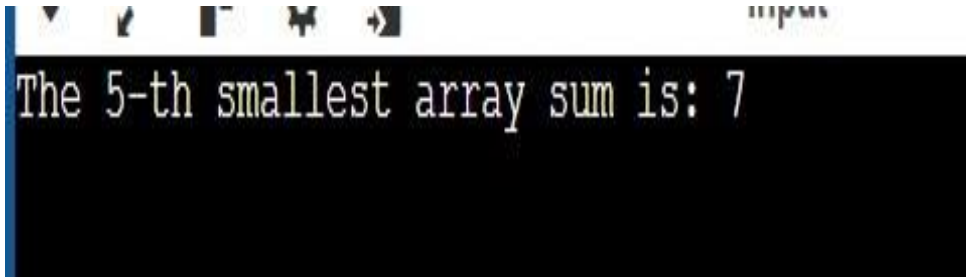
```
        {1, 3, 11},

        {2, 4, 6}

    };

    int k = 5;

    cout << "The " << k << "-th smallest array sum is: " <<
    kthSmallest(mat, k) << endl;



    return 0;

}
```

Output:

The 5-th smallest array sum is: 7

Problem 9

Aim: 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.

Code:

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
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) {}
};

struct compare {
    bool operator()(ListNode* l1, ListNode* l2) {
        return l1->val > l2->val;  // Min-heap: pop the smaller value
    }
};

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 build the result
    ListNode* current = dummy;
```

```cpp
    while (!minHeap.empty()) {
        // Get the smallest node from the heap
        ListNode* node = minHeap.top();
        minHeap.pop();

        current->next = node;
        current = current->next;
        if (node->next != nullptr) {
            minHeap.push(node->next);
        }
    }
    return dummy->next;  // The next node of dummy is the head of the
merged list
}
void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    ListNode* l1 = new ListNode(1, new ListNode(4, new ListNode(5)));
    ListNode* l2 = new ListNode(1, new ListNode(3, new ListNode(4)));
    ListNode* l3 = new ListNode(2, new ListNode(6));
    vector<ListNode*> lists = {l1, l2, l3};
    ListNode* mergedList = mergeKLists(lists);
    cout << "Merged sorted list: ";
    printList(mergedList);

    return 0;
}
```

Output:

Merged sorted list: 1 1 2 3 4 4 5 6

Problem 10

Aim: You are given an integer array arr.We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.Return the largest number of chunks we can make to sort the array.

Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int maxChunksToSorted(vector<int>& arr) {

    int n = arr.size();

    vector<int> left_max(n);

    left_max[0] = arr[0];

    for (int i = 1; i < n; ++i) {

        left_max[i] = max(left_max[i - 1], arr[i]);

    }

    vector<int> right_min(n);

    right_min[n - 1] = arr[n - 1];

    for (int i = n - 2; i >= 0; --i) {

        right_min[i] = min(right_min[i + 1], arr[i]);

    }
```

```cpp
    int chunks = 0;

    for (int i = 0; i < n - 1; ++i) {

        if (left_max[i] <= right_min[i + 1]) {

            chunks++;

        }

    }

    return chunks + 1;

}
int main() {

    vector<int> arr = {5, 4, 3, 2, 1};

    cout << "Maximum number of chunks: " << maxChunksToSorted(arr) << endl;  // Output: 1

    return 0;

}
```
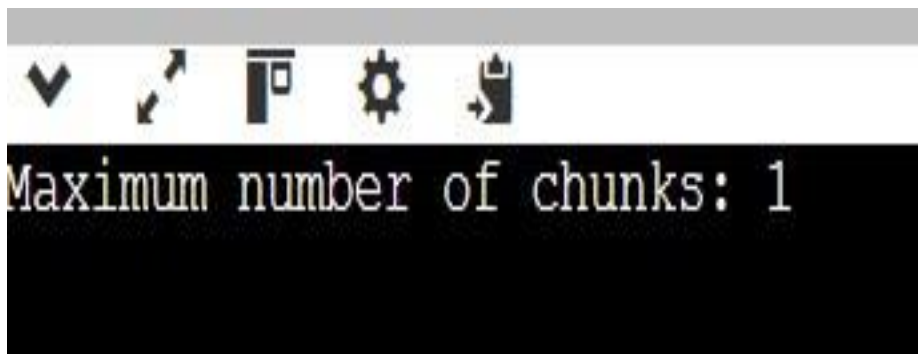
Output: