Week 1 Assignment

Name – Sumit Phalke

Class – TY CSE

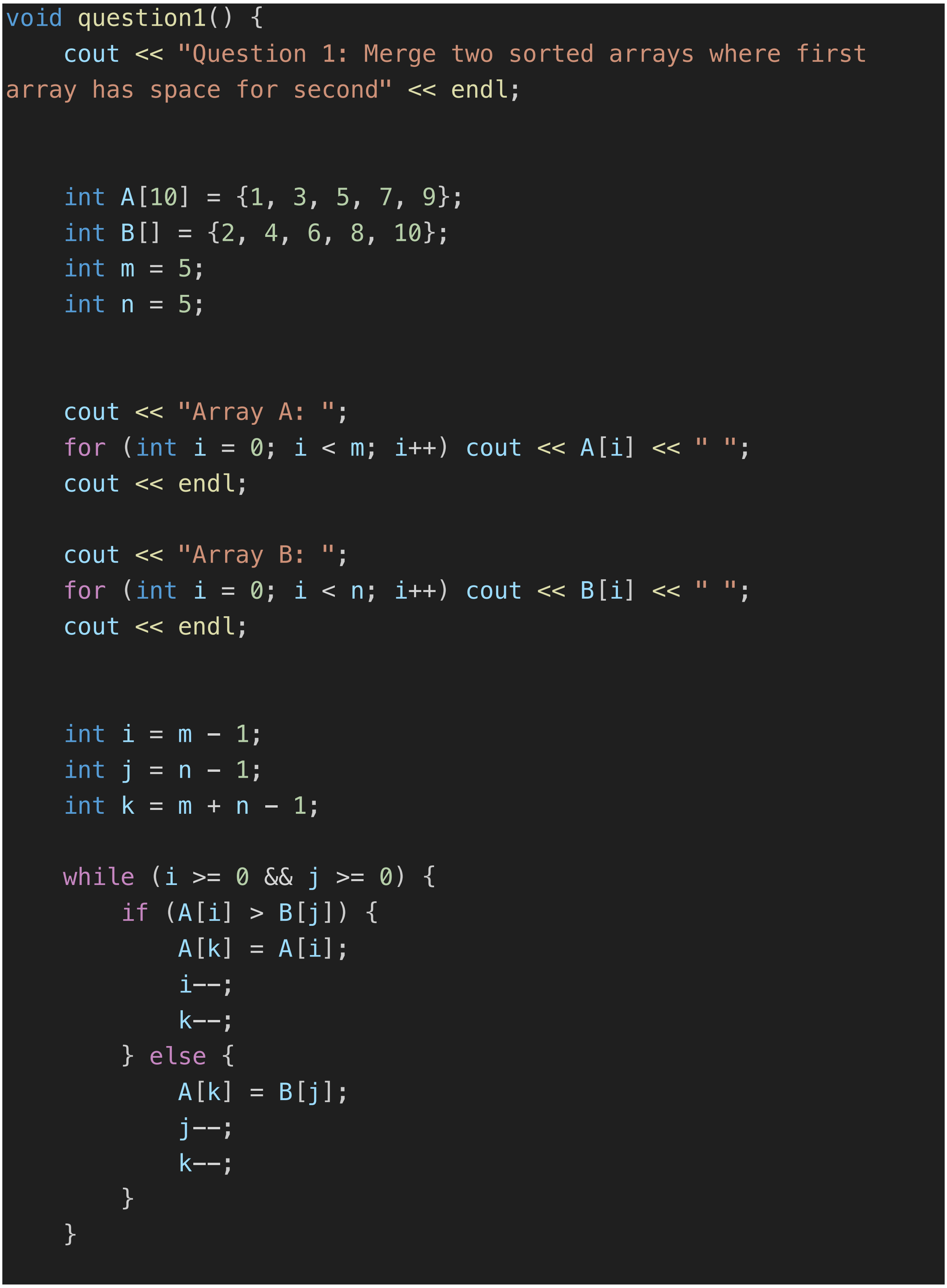
PRN – 23510023

Question1:
You are given two sorted arrays, A and B, where A has extra buffer space at the end to hold B. Merge B into A in sorted order.

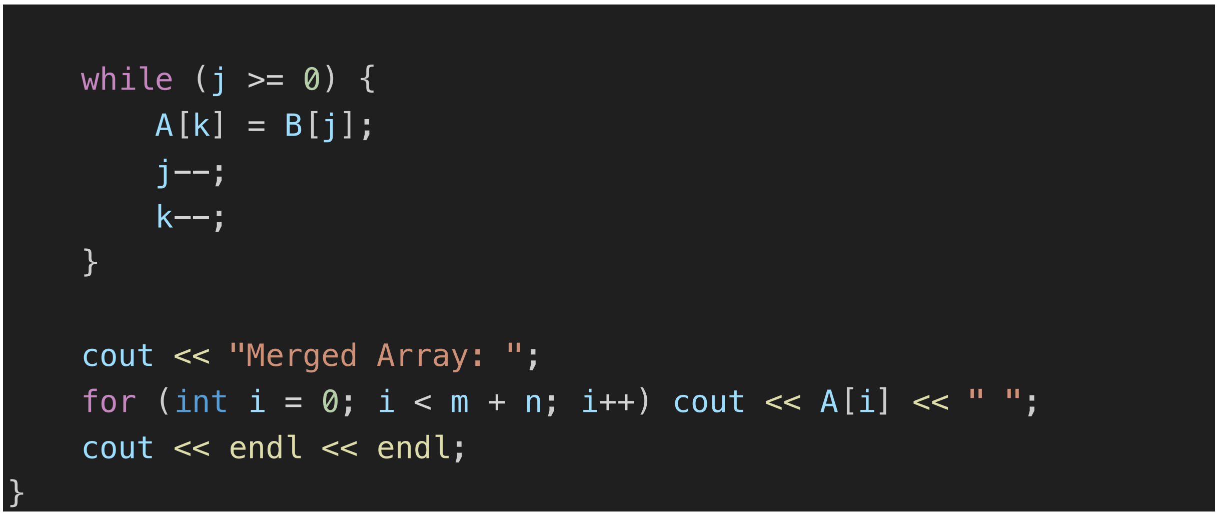
Algorithm:

1. Start with two sorted arrays `A` and `B`.
2. Place three pointers:
    - `i` at the end of valid elements in A.
    - `j` at the end of B.
    - `k` at the very end of the buffer in A.
3. Compare elements of A and B from the back. Place the larger element at position `k`.
4. Move the respective pointer (`i` or `j`) backward and also move `k` backward.
5. Continue until all elements of B are merged into A.
6. The result is a single sorted array inside A.

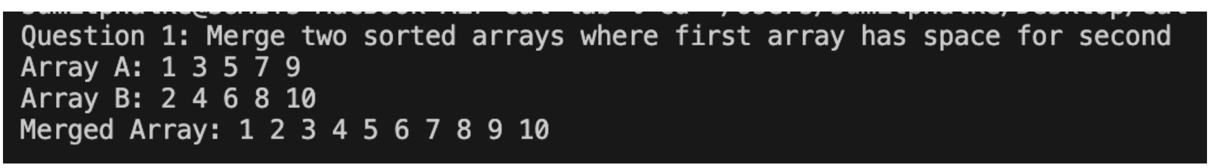
Input Screenshot & Output Screenshots;



```
void question1() {
    cout << "Question 1: Merge two sorted arrays where first
array has space for second" << endl;


    int A[10] = {1, 3, 5, 7, 9};
    int B[] = {2, 4, 6, 8, 10};
    int m = 5;
    int n = 5;


    cout << "Array A: ";
    for (int i = 0; i < m; i++) cout << A[i] << " ";
    cout << endl;

    cout << "Array B: ";
    for (int i = 0; i < n; i++) cout << B[i] << " ";
    cout << endl;


    int i = m - 1;
    int j = n - 1;
    int k = m + n - 1;

    while (i >= 0 && j >= 0) {
        if (A[i] > B[j]) {
            A[k] = A[i];
            i--;
            k--;
        } else {
            A[k] = B[j];
            j--;
            k--;
        }
    }
```



```
    while (j >= 0) {
        A[k] = B[j];
        j--;
        k--;
    }

    cout << "Merged Array: ";
    for (int i = 0; i < m + n; i++) cout << A[i] << " ";
    cout << endl << endl;
}
```



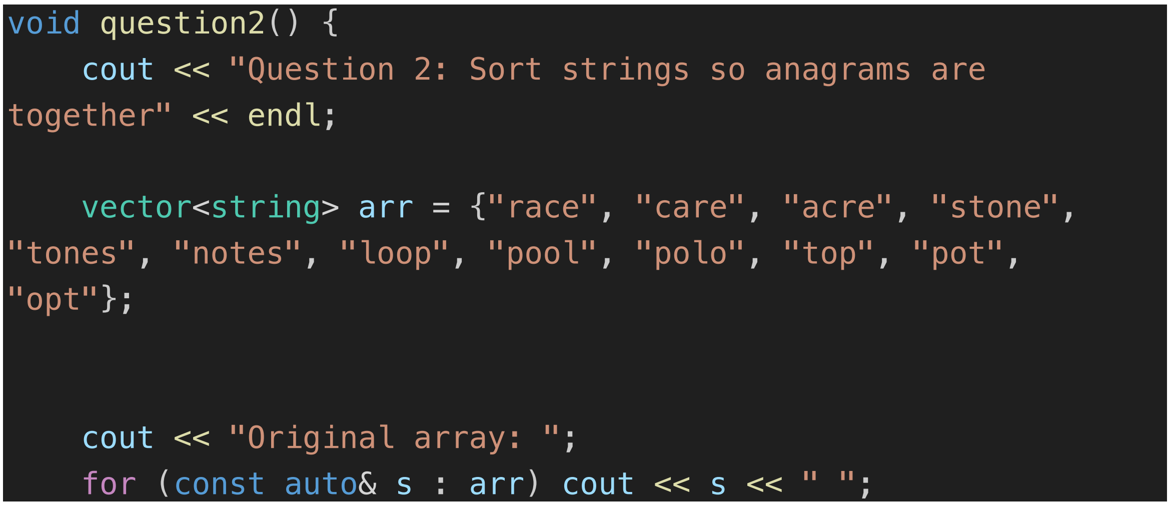
```
Question 1: Merge two sorted arrays where first array has space for second
Array A: 1 3 5 7 9
Array B: 2 4 6 8 10
Merged Array: 1 2 3 4 5 6 7 8 9 10
```

Question2:
Write a method to sort an array of strings so that all anagrams appear next to each other.

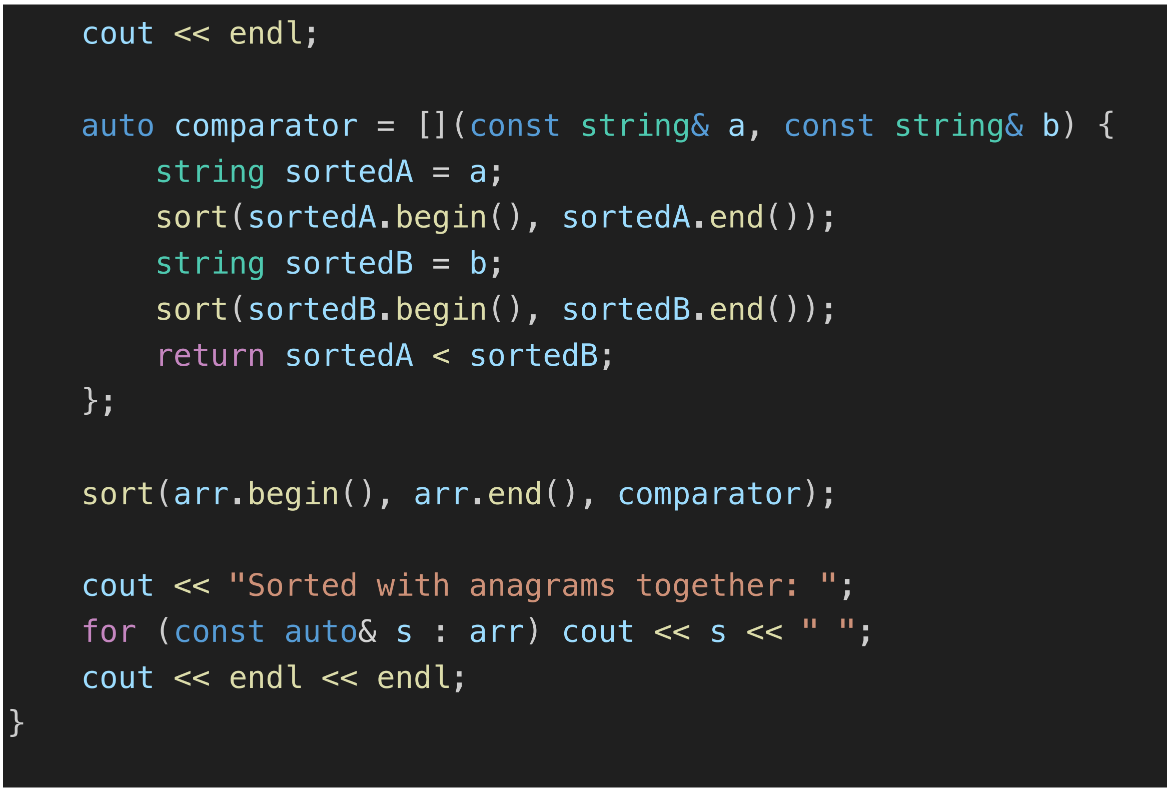
Algorithm:

1. Take a list of strings.
2. For each string, sort its characters alphabetically.
3. Use the sorted version as a "key" to group anagrams.
4. Sort the entire list based on these keys.
5. This ensures that words with the same character composition (anagrams) appear together.

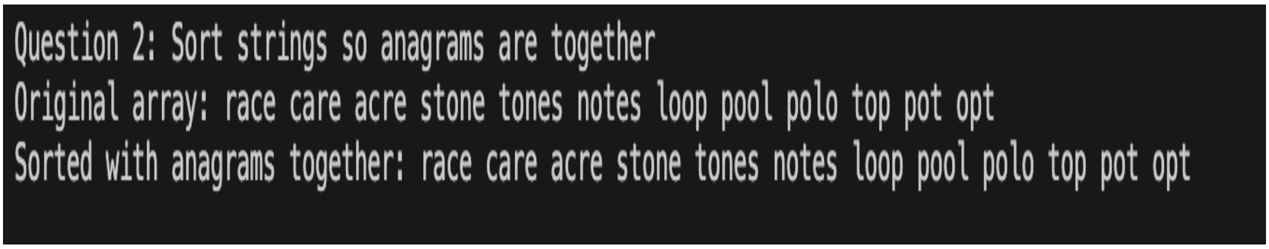
Input Screenshot & Output Screenshots;



```
void question2() {
    cout << "Question 2: Sort strings so anagrams are
together" << endl;

    vector<string> arr = {"race", "care", "acre", "stone",
"tones", "notes", "loop", "pool", "polo", "top", "pot",
"opt"};


    cout << "Original array: ";
    for (const auto& s : arr) cout << s << " ";
```



```
    cout << endl;

    auto comparator = [](const string& a, const string& b) {
        string sortedA = a;
        sort(sortedA.begin(), sortedA.end());
        string sortedB = b;
        sort(sortedB.begin(), sortedB.end());
        return sortedA < sortedB;
    };

    sort(arr.begin(), arr.end(), comparator);

    cout << "Sorted with anagrams together: ";
    for (const auto& s : arr) cout << s << " ";
    cout << endl << endl;
}
```



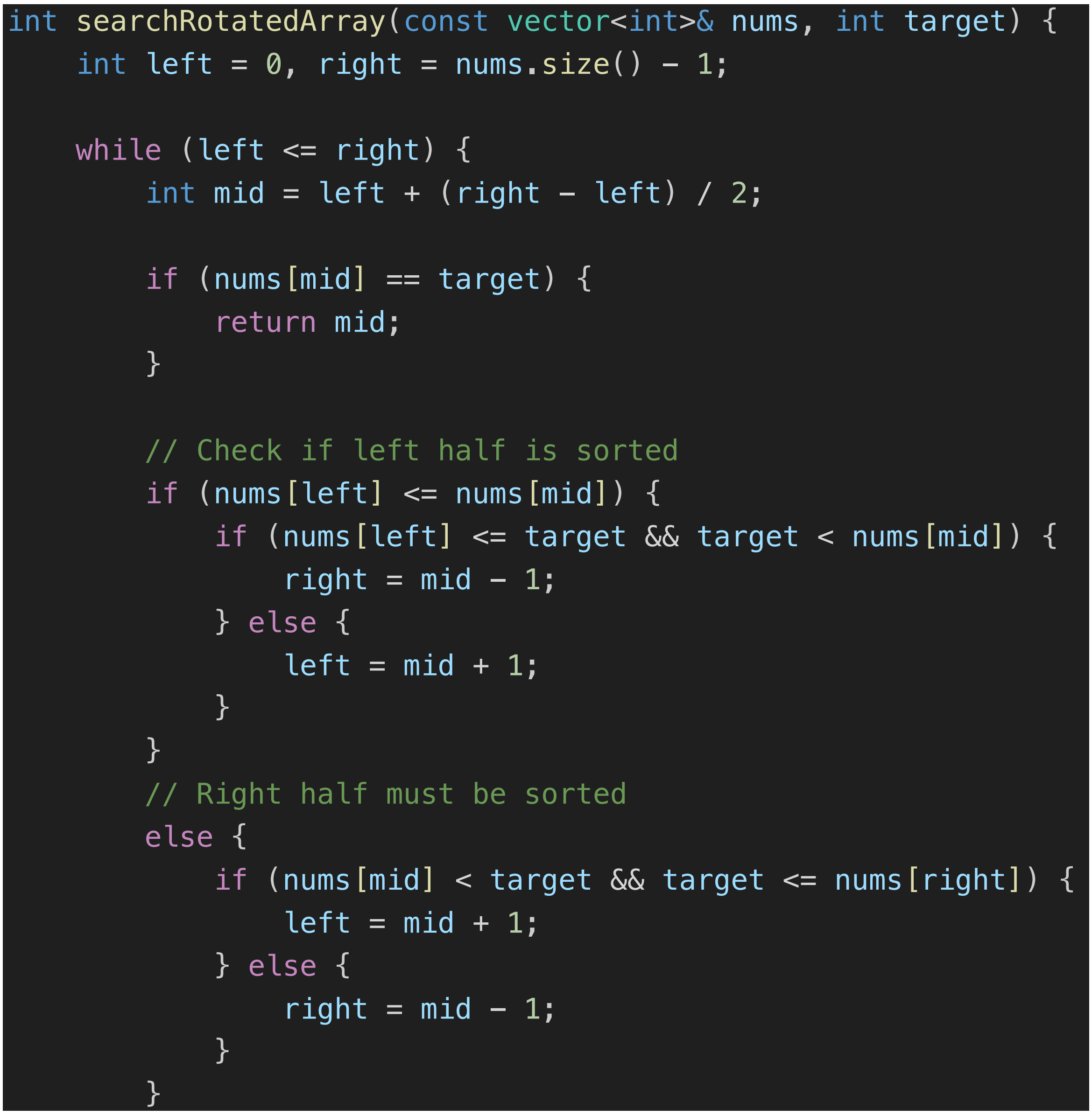
```
Question 2: Sort strings so anagrams are together
Original array: race care acre stone tones notes loop pool polo top pot opt
Sorted with anagrams together: race care acre stone tones notes loop pool polo top pot opt
```

Question3:
Given a sorted array that has been rotated an unknown number of times, find the index of a target element.

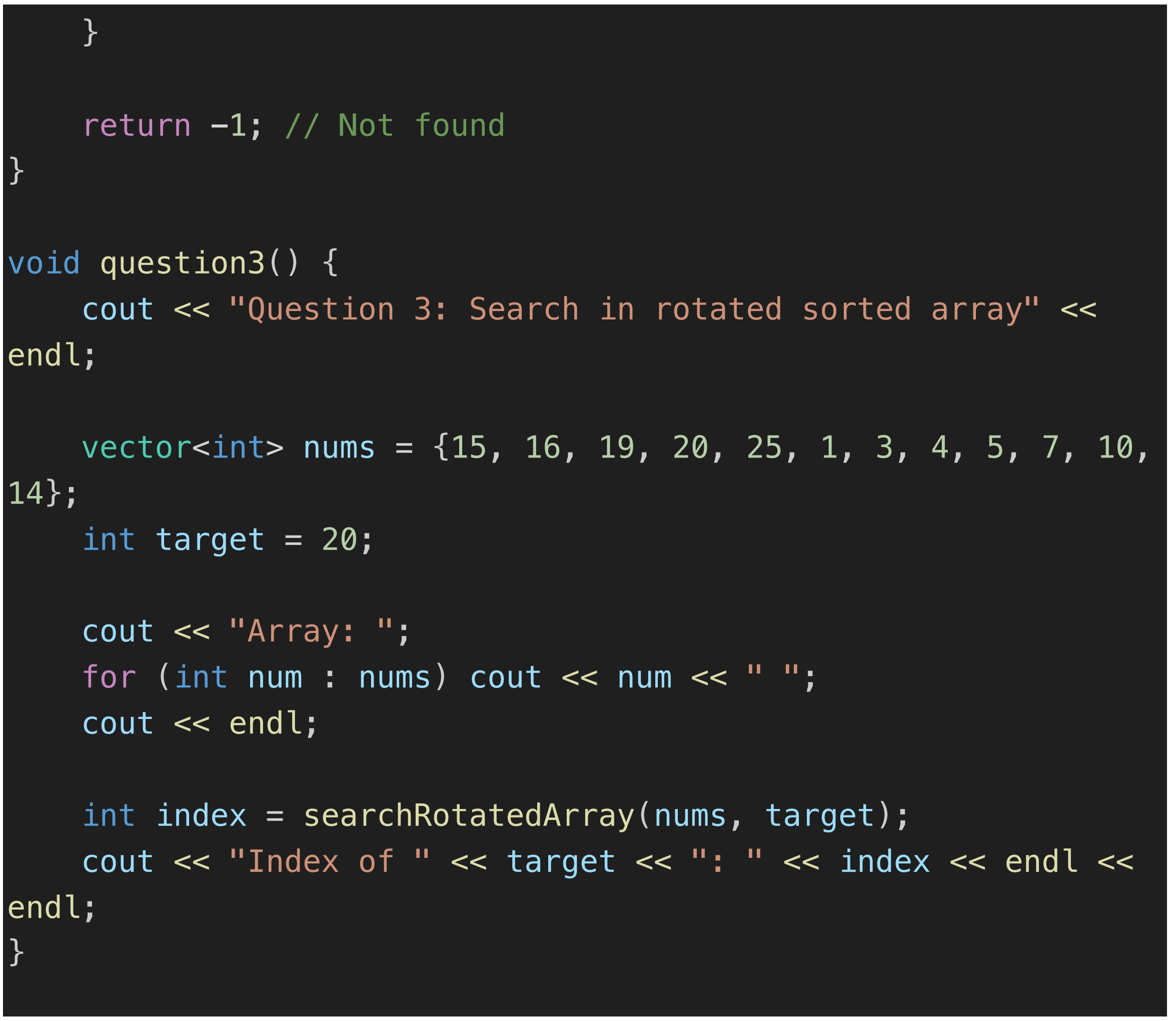
Algorithm:

1. Use a modified **binary search**.
2. At each step, check if the middle element is the target.
3. Determine whether the left half or the right half of the array is sorted.
4. If the target lies within the sorted half, move the search there. Otherwise, search in the other half.
5. Continue until the element is found or the search ends.

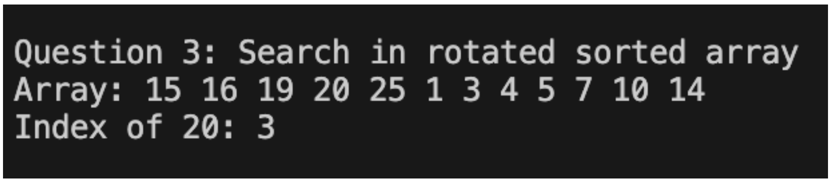
Input Screenshot & Output Screenshots;



```
int searchRotatedArray(const vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;

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

        if (nums[mid] == target) {
            return mid;
        }

        // Check if left half is sorted
        if (nums[left] <= nums[mid]) {
            if (nums[left] <= target && target < nums[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        // Right half must be sorted
        else {
            if (nums[mid] < target && target <= nums[right]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
```



```
    }

    return -1; // Not found
}

void question3() {
    cout << "Question 3: Search in rotated sorted array" <<
endl;

    vector<int> nums = {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10,
14};
    int target = 20;

    cout << "Array: ";
    for (int num : nums) cout << num << " ";
    cout << endl;

    int index = searchRotatedArray(nums, target);
    cout << "Index of " << target << ": " << index << endl <<
endl;
}
```



```
Question 3: Search in rotated sorted array
Array: 15 16 19 20 25 1 3 4 5 7 10 14
Index of 20: 3
```

Question4:
Imagine you have a 20GB file with one string per line. Explain how you would sort the file.

Algorithm (Explanation):

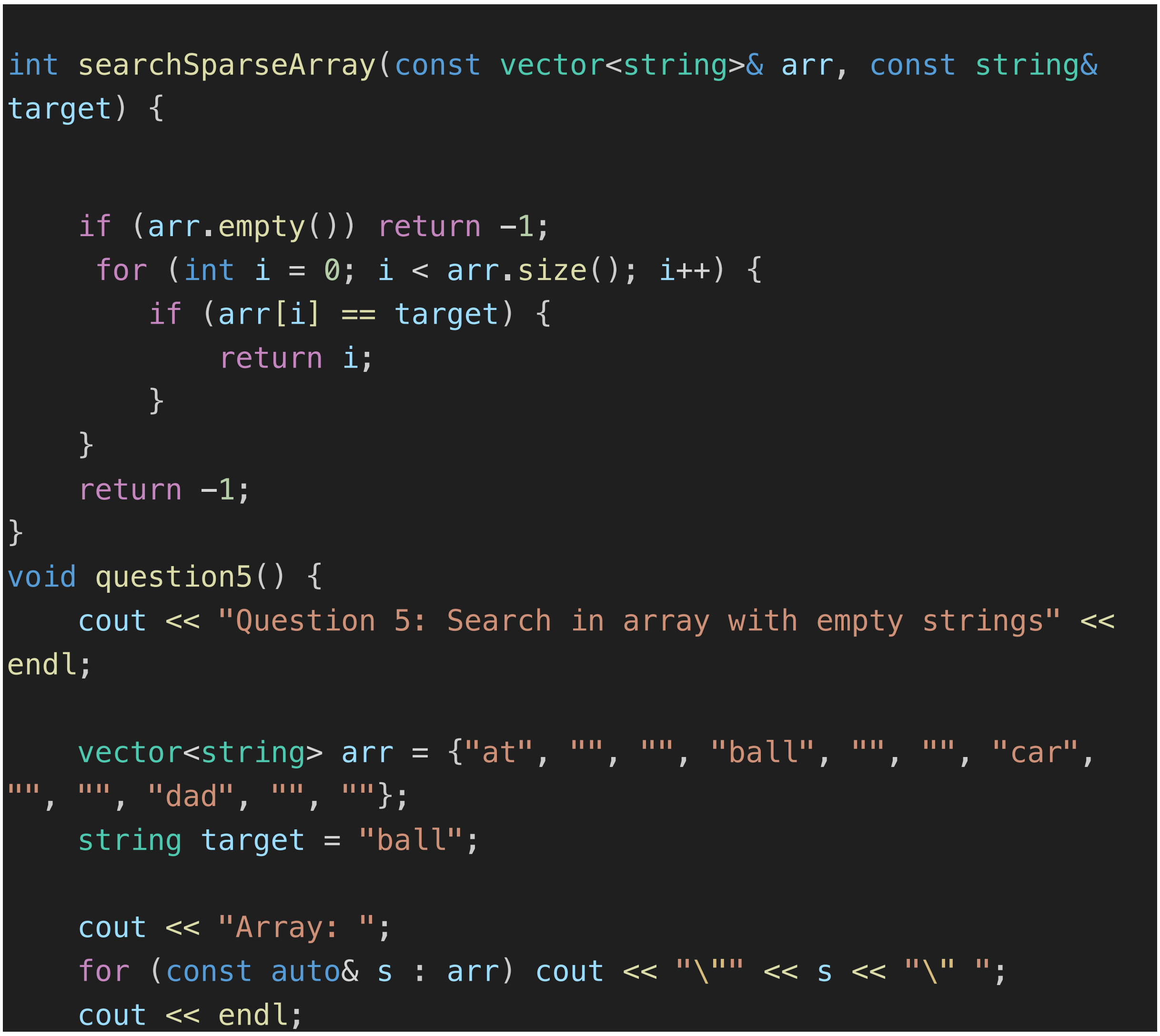
1. Since the file is too large to fit in memory, use **external sorting**.
2. Break the file into smaller chunks that can fit into memory (e.g., 100MB each).
3. Sort each chunk separately using an in-memory sorting algorithm.
4. Write the sorted chunks back to disk.
5. Use a **k-way merge algorithm** to merge all sorted chunks into a single sorted file

Question5:
Given a sorted array of strings interspersed with empty strings, find the index of a target string.

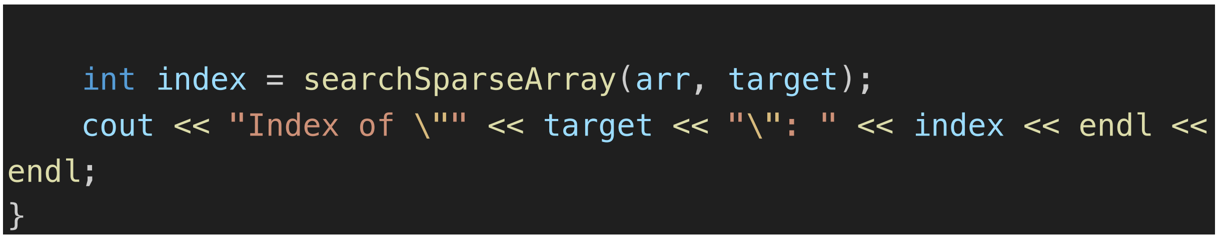
Algorithm:

1. Traverse the array from start to end.
2. Skip empty strings while searching.
3. Compare only non-empty strings with the target.
4. Return the index when the target is found, otherwise return -1.

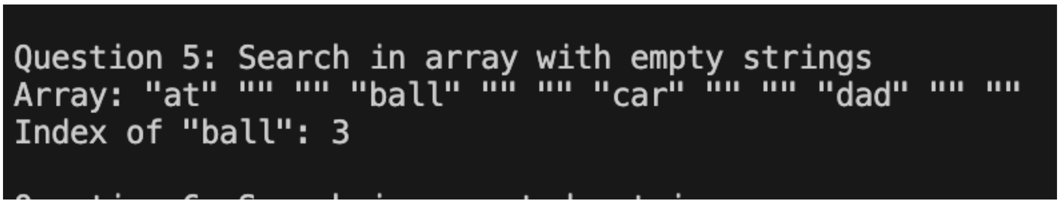
Input Screenshot & Output Screenshots;



```
int searchSparseArray(const vector<string>& arr, const string&
target) {


    if (arr.empty()) return -1;
     for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
void question5() {
    cout << "Question 5: Search in array with empty strings" <<
endl;

    vector<string> arr = {"at", "", "", "ball", "", "", "car",
"", "", "dad", "", ""};
    string target = "ball";

    cout << "Array: ";
    for (const auto& s : arr) cout << "\"" << s << "\" ";
    cout << endl;
```



```cpp
    int index = searchSparseArray(arr, target);
    cout << "Index of \"" << target << "\": " << index << endl <<
endl;
}
```



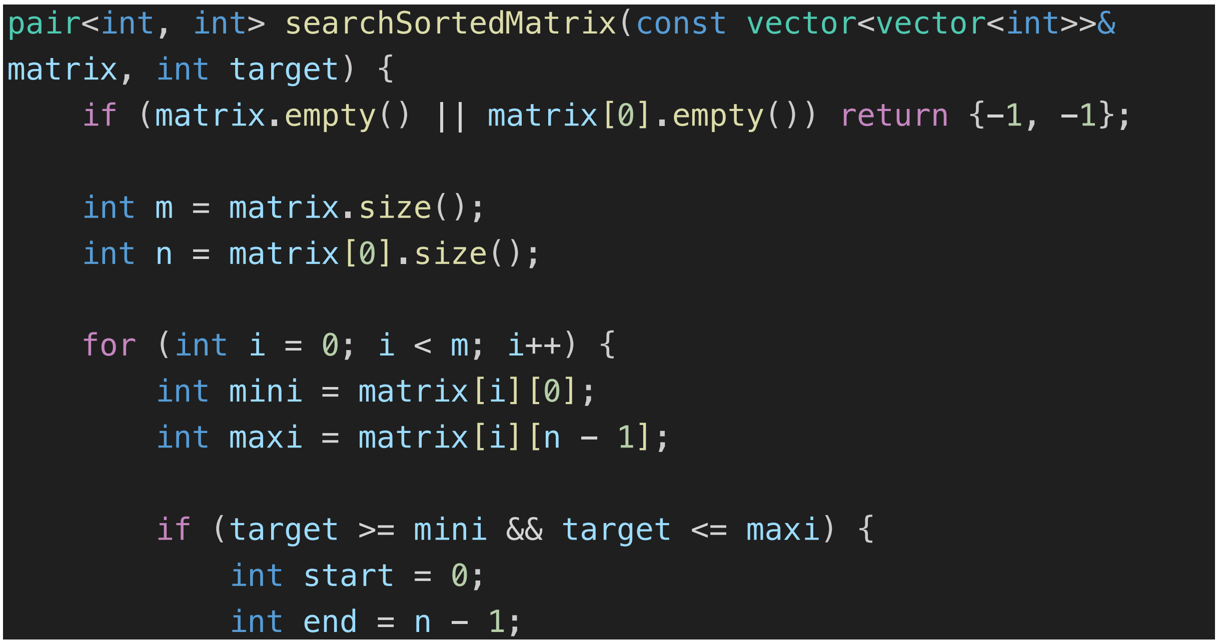
```
Question 5: Search in array with empty strings
Array: "at" "" "" "ball" "" "" "car" "" "" "dad" "" ""
Index of "ball": 3
```

Question6:
Given an MxN matrix where each row and column is sorted in ascending order, find the location of a target element.

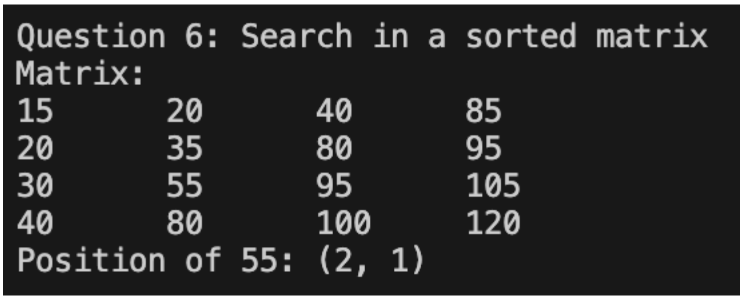
Algorithm:

1. Each row is sorted, so check if the target lies within the minimum and maximum values of the row.
2. If yes, apply binary search on that row.
3. If not, move to the next row.
4. Continue until the target is found or all rows are searched.

Input Screenshot & Output Screenshots;



```cpp
pair<int, int> searchSortedMatrix(const vector<vector<int>>&
matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) return {-1, -1};

    int m = matrix.size();
    int n = matrix[0].size();

    for (int i = 0; i < m; i++) {
        int mini = matrix[i][0];
        int maxi = matrix[i][n - 1];

        if (target >= mini && target <= maxi) {
            int start = 0;
            int end = n - 1;
            while (start <= end) {
                int mid = start + (end - start) / 2;

                if (matrix[i][mid] == target) {
                    return {i, mid};
                }
                else if (matrix[i][mid] < target) {
                    start = mid + 1;
                }
                else {
                    end = mid - 1;
                }
            }
        }
    }
    return {-1, -1};
}
void question6() {
    cout << "Question 6: Search in a sorted matrix" << endl;

    vector<vector<int>> matrix = {
        {15, 20, 40, 85},
        {20, 35, 80, 95},
        {30, 55, 95, 105},
        {40, 80, 100, 120}
    };
    int target = 55;

    cout << "Matrix:" << endl;
    for (const auto& row : matrix) {
        for (int num : row) cout << num << "\t";
        cout << endl;
    }

    auto result = searchSortedMatrix(matrix, target);
    cout << "Position of " << target << ": (" << result.first <<
", " << result.second << ")" << endl << endl;
}
```



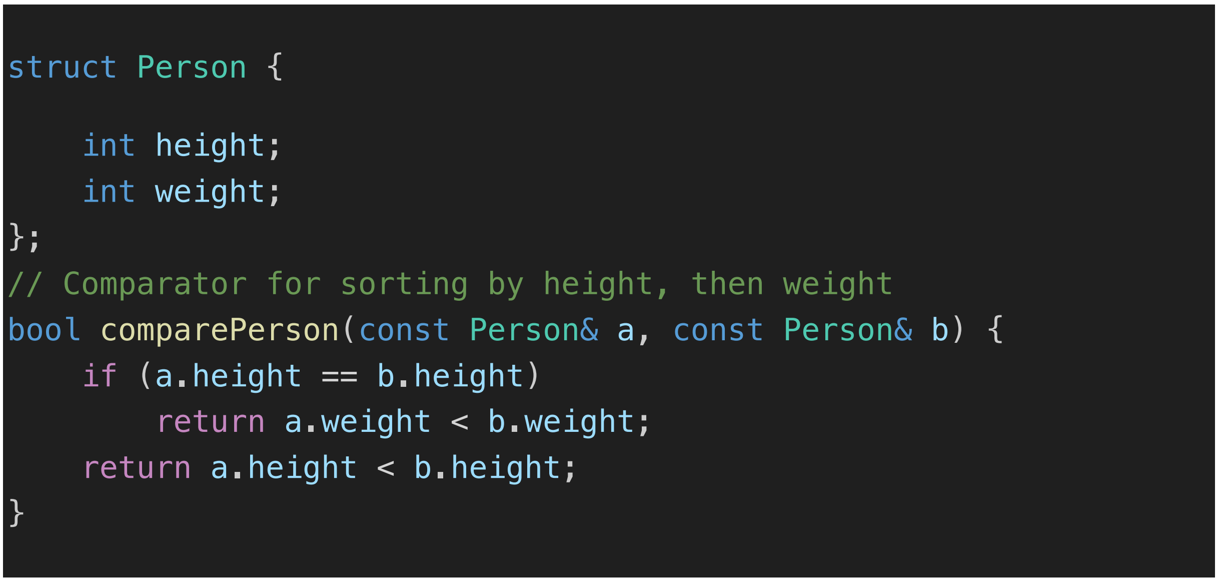
```
Question 6: Search in a sorted matrix
Matrix:
15      20      40      85
20      35      80      95
30      55      95      105
40      80      100     120
Position of 55: (2, 1)
```

Question7:
You are given the height and weight of people. Find the largest possible tower such that each person above is both shorter and lighter than the one below.

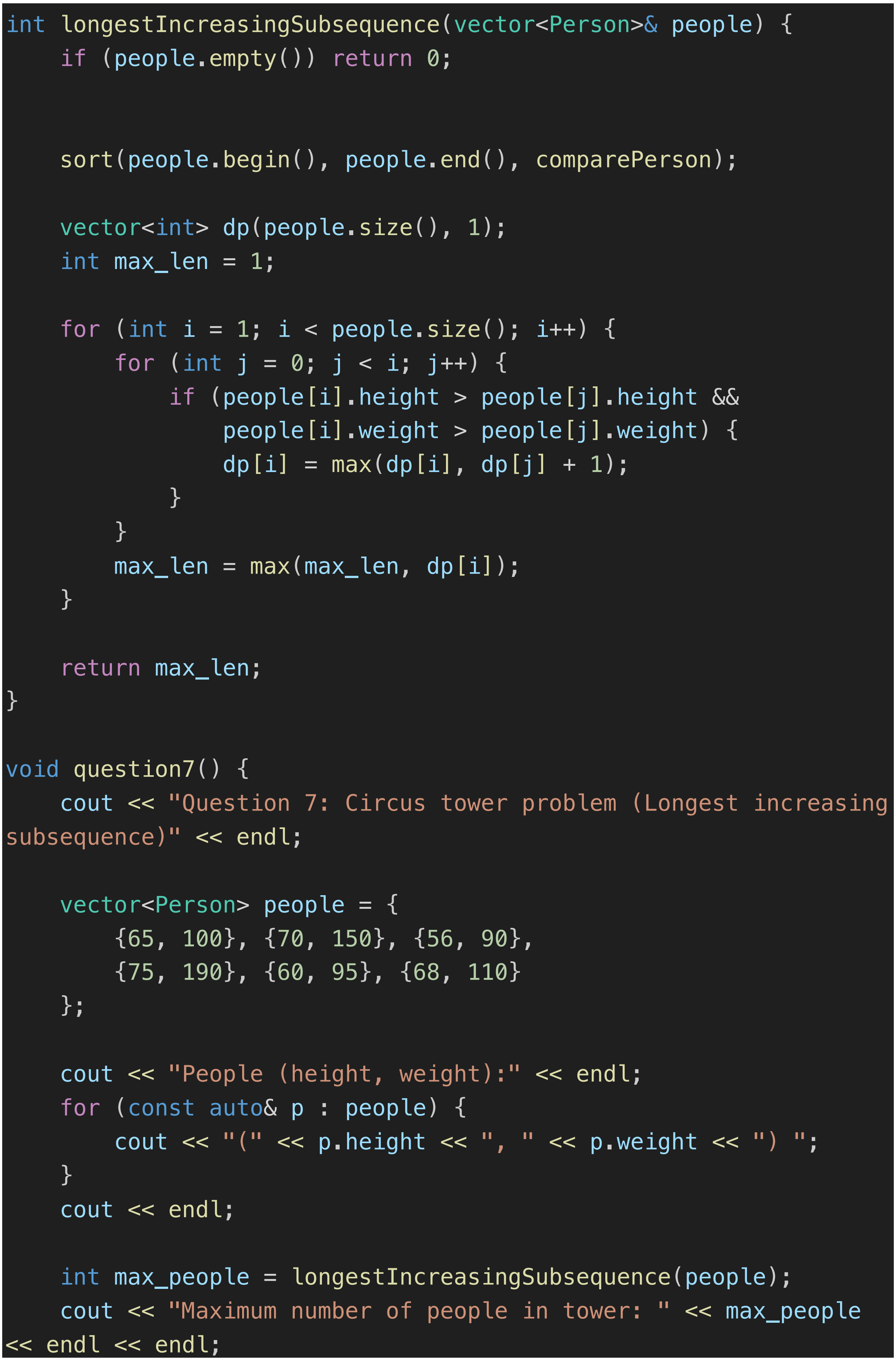
Algorithm:

1. Represent each person with (height, weight).
2. Sort people first by height, then by weight.
3. Apply **Longest Increasing Subsequence (LIS)** on weights.
4. The LIS gives the maximum number of people that can be stacked.

Input Screenshot & Output Screenshots;

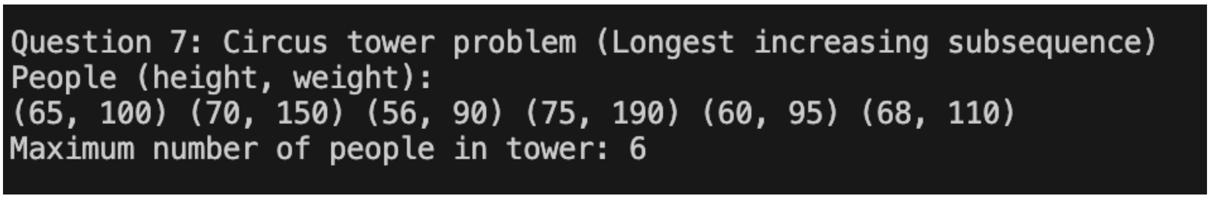


```
struct Person {

    int height;
    int weight;
};
// Comparator for sorting by height, then weight
bool comparePerson(const Person& a, const Person& b) {
    if (a.height == b.height)
        return a.weight < b.weight;
    return a.height < b.height;
}
```



```cpp
int longestIncreasingSubsequence(vector<Person>& people) {
    if (people.empty()) return 0;


    sort(people.begin(), people.end(), comparePerson);

    vector<int> dp(people.size(), 1);
    int max_len = 1;

    for (int i = 1; i < people.size(); i++) {
        for (int j = 0; j < i; j++) {
            if (people[i].height > people[j].height &&
                people[i].weight > people[j].weight) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        max_len = max(max_len, dp[i]);
    }

    return max_len;
}

void question7() {
    cout << "Question 7: Circus tower problem (Longest increasing subsequence)" << endl;

    vector<Person> people = {
        {65, 100}, {70, 150}, {56, 90},
        {75, 190}, {60, 95}, {68, 110}
    };

    cout << "People (height, weight):" << endl;
    for (const auto& p : people) {
        cout << "(" << p.height << ", " << p.weight << ") ";
    }
    cout << endl;

    int max_people = longestIncreasingSubsequence(people);
    cout << "Maximum number of people in tower: " << max_people << endl << endl;
```

```
}
```



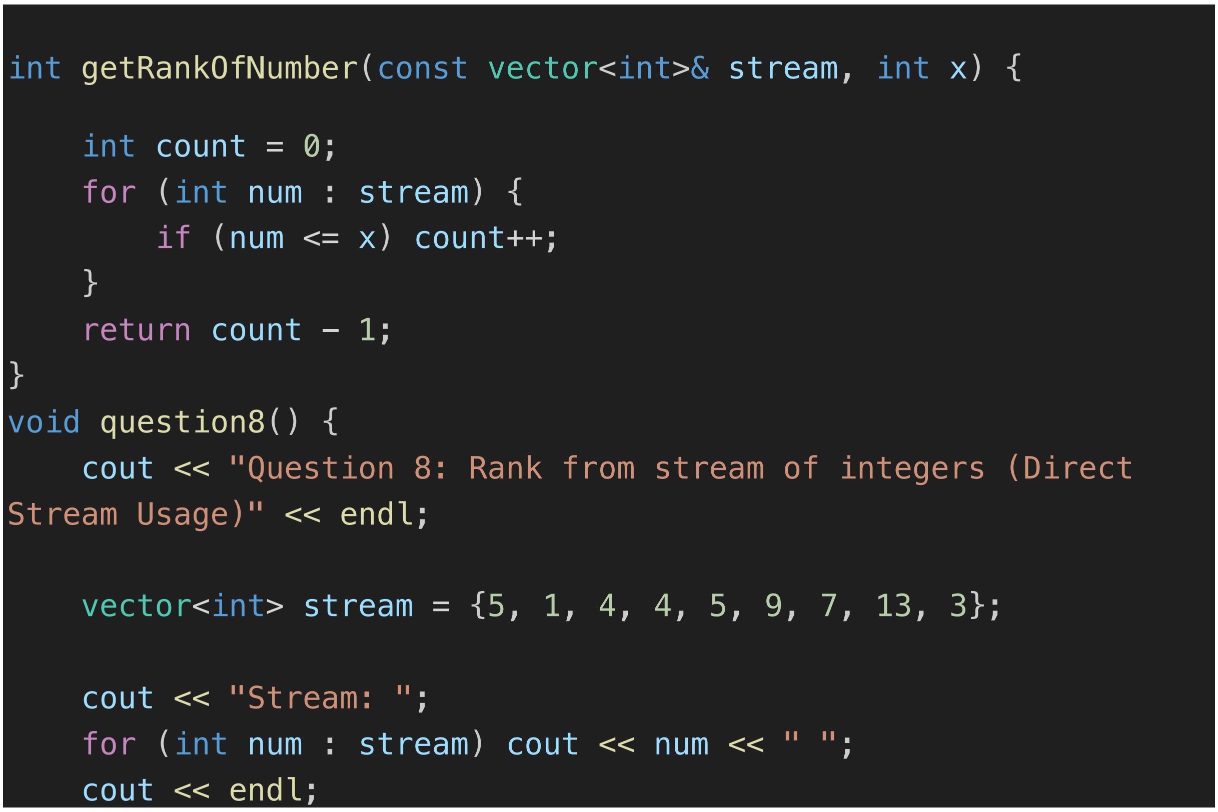
```
Question 7: Circus tower problem (Longest increasing subsequence)
People (height, weight):
(65, 100) (70, 150) (56, 90) (75, 190) (60, 95) (68, 110)
Maximum number of people in tower: 6
```

Question8:
Given a stream of integers, implement a method to return the rank of a number (number of values less than or equal to it).

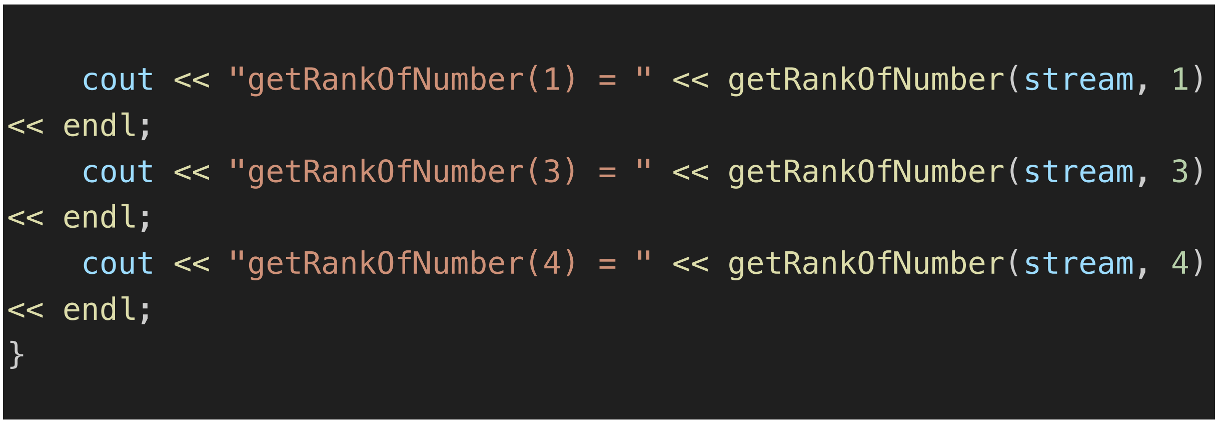
Algorithm:

1. Maintain a stream of integers as they arrive.
2. For a query `getRankOfNumber(x)`:
   - Traverse the stream.
   - Count how many numbers are less than or equal to `x`.
   - Subtract one (as per problem statement requirement).
3. Return the count as the rank.

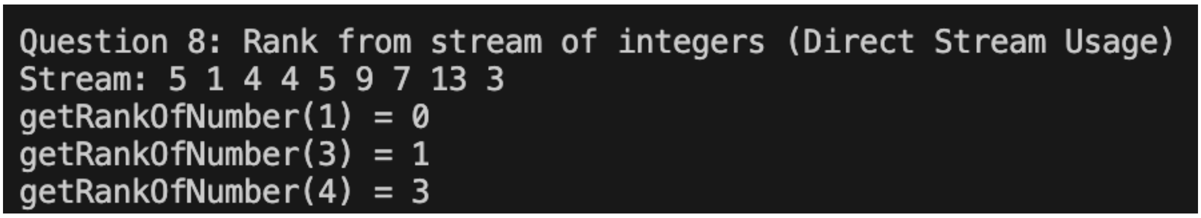
Input Screenshot & Output Screenshots;



```
int getRankOfNumber(const vector<int>& stream, int x) {

    int count = 0;
    for (int num : stream) {
        if (num <= x) count++;
    }
    return count - 1;
}
void question8() {
    cout << "Question 8: Rank from stream of integers (Direct
Stream Usage)" << endl;

    vector<int> stream = {5, 1, 4, 4, 5, 9, 7, 13, 3};

    cout << "Stream: ";
    for (int num : stream) cout << num << " ";
    cout << endl;
```



```
    cout << "getRankOfNumber(1) = " << getRankOfNumber(stream, 1)
<< endl;
    cout << "getRankOfNumber(3) = " << getRankOfNumber(stream, 3)
<< endl;
    cout << "getRankOfNumber(4) = " << getRankOfNumber(stream, 4)
<< endl;
}
```



```
Question 8: Rank from stream of integers (Direct Stream Usage)
Stream: 5 1 4 4 5 9 7 13 3
getRankOfNumber(1) = 0
getRankOfNumber(3) = 1
getRankOfNumber(4) = 3
```