Name: Sai Gulve

PRN No: 202201040145

Batch: A4

Roll No: 63

Problem Statement:

In a city, there are **N persons**, and each person needs exactly **one cab** for their journey. Every person has a **start time** and an **end time**, which indicate when they need a cab. The goal is to determine the **minimum number of cabs required** at any point in time to accommodate all travelers.

Brief About the Problem:

This problem is an example of **interval scheduling optimization**, which is commonly solved using **Greedy Algorithms**. The challenge is to efficiently assign cabs to travelers in such a way that the number of cabs is minimized while ensuring that no person is left without a ride.

The **key observation** is that whenever a new person starts their journey, they require a cab. However, when another person's journey ends, their cab becomes available. Using this information, we can track the number of cabs needed at any given time.

Approach:

1. **Sorting the Start and End Times:**
    - To process events in chronological order, we sort both **start times** and **end times** separately.
    - The sorting is implemented manually using **Bubble Sort** to avoid using the built-in sort() function.
2. **Using Two Pointers to Count Overlapping Rides:**
    - We traverse both **start** and **end** times using two pointers (i for start, j for end).
    - If a new person's ride starts before or when another ride ends, a new cab is required.
    - If a ride ends before the next ride starts, a cab is freed and reused.
    - The maximum number of cabs used at any moment gives the required answer.

Algorithm:

1. **Read the number of persons (N).**
2. **Input the start and end times** for each person.
3. **Sort the start and end times** manually using **Bubble Sort**.
4. **Initialize variables:**
    - cabs = 0 (current number of active cabs).
    - maxCabs = 0 (tracks the peak number of cabs used).
    - Two indices: i = 0 for start times, j = 0 for end times.

5. **Process events in order:**
    - If a person starts their journey (start[i] <= end[j]), increase the cab count.
    - If a ride ends (start[i] > end[j]), decrease the cab count.
    - Track the **maximum** cab count at any point.
6. **Output the minimum number of cabs required.**

Complexity Analysis:

Operation	**Complexity**
Bubble Sort	$O(N^2)$
Two-pointer traversal	O(N)
Overall Complexity	**$O(N^2)$**

Since Bubble Sort is used for sorting, the worst-case complexity remains **$O(N^2)$**, which is not the most optimal but works fine for small to moderate values of **N**.

Code :-

```
#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

// Custom function to sort an array using Bubble Sort
void bubbleSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            // Swap elements
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
```

```
    }
}

// Function to find the minimum number of cabs required
int minCabsRequired(int n, int start[], int end[]) {
    bubbleSort(start, n);
    bubbleSort(end, n);

    int cabs = 0, maxCabs = 0;
    int i = 0, j = 0;

    while (i < n) {
        if (start[i] <= end[j]) {
            cabs++;
            if (cabs > maxCabs)
                maxCabs = cabs;
            i++;
        } else {
            cabs--;
            j++;
        }
    }
    return maxCabs;
}

int main() {
    int n;
    cout << "Enter number of persons: ";
    cin >> n;

    int start[n], end[n];
```

```
    cout << "Enter start times: ";
    for (int i = 0; i < n; i++)
        cin >> start[i];

    cout << "Enter end times: ";
    for (int i = 0; i < n; i++)
        cin >> end[i];

    auto start_time = high_resolution_clock::now();

    int result = minCabsRequired(n, start, end);

    auto end_time = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end_time - start_time);

    cout << "Minimum number of cabs required: " << result << endl;
    cout << "Execution time: " << duration.count() << " microseconds" << endl;

    return 0;
}
```

Output:-

```
Enter number of persons: 3
Enter start times: 1
3
5
Enter end times: 6
7
9
Minimum number of cabs required: 3
Execution time: 280 mnanoseconds


=== Code Execution Successful ===
```