

1. Implement non-recursive version of quick sort algorithm using stack.

SOURCE CODE

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

class QuickSort {
public:
    void iterativeQuickSort(vector<int>& arr) {
        stack<int> bounds;
        int left = 0;
        int right = arr.size() - 1;

        bounds.push(left);
        bounds.push(right);

        while (!bounds.empty()) {
            right = bounds.top(); bounds.pop();
            left = bounds.top(); bounds.pop();

            if (left < right) {
                int pivot = arr[right];
                int partitionIndex = left - 1;

                for (int i = left; i < right; ++i) {
                    if (arr[i] <= pivot) {
                        partitionIndex++;
                        swap(arr[partitionIndex], arr[i]);
                    }
                }

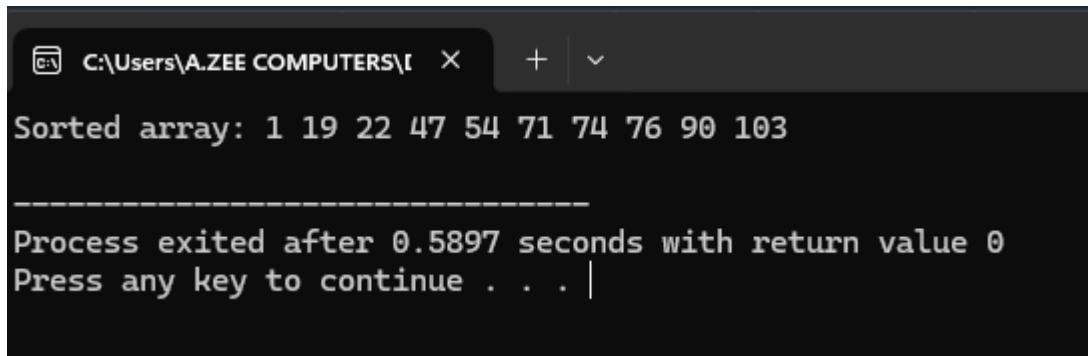
                swap(arr[partitionIndex + 1], arr[right]);
                int pivotPosition = partitionIndex + 1;

                bounds.push(left);
                bounds.push(pivotPosition - 1);

                bounds.push(pivotPosition + 1);
                bounds.push(right);
            }
        }
    }
}
```

```
    }  
};  
  
int main() {  
    vector<int> numbers = {76,90,1,47,103,71,19,22,74,54};  
    QuickSort sorter;  
    sorter.iterativeQuickSort(numbers);  
  
    cout << "Sorted array: ";  
    for (int num : numbers) {  
        cout << num << " ";  
    }  
    cout << endl;  
  
    return 0;  
}
```

OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I  X  +  v  
Sorted array: 1 19 22 47 54 71 74 76 90 103  
-----  
Process exited after 0.5897 seconds with return value 0  
Press any key to continue . . . |
```

2. Implement non-recursive version of the merge sort algorithm

SOURCE CODE

```
#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
class MergeSort {  
public:  
    void mergeHalves(vector<int>& arr, int low, int mid, int high) {  
        vector<int> left(arr.begin() + low, arr.begin() + mid + 1);  
        vector<int> right(arr.begin() + mid + 1, arr.begin() + high + 1);
```

```
int i = 0, j = 0, k = low;
while (i < left.size() && j < right.size()) {
    if (left[i] <= right[j]) {
        arr[k++] = left[i++];
    } else {
        arr[k++] = right[j++];
    }
}

while (i < left.size()) {
    arr[k++] = left[i++];
}

while (j < right.size()) {
    arr[k++] = right[j++];
}
}

void iterativeMergeSort(vector<int>& arr) {
    int n = arr.size();
    for (int currentSize = 1; currentSize < n; currentSize *= 2) {
        for (int start = 0; start < n; start += 2 * currentSize) {
            int mid = min(start + currentSize - 1, n - 1);
            int end = min(start + 2 * currentSize - 1, n - 1);
            if (mid < end) {
                mergeHalves(arr, start, mid, end);
            }
        }
    }
}

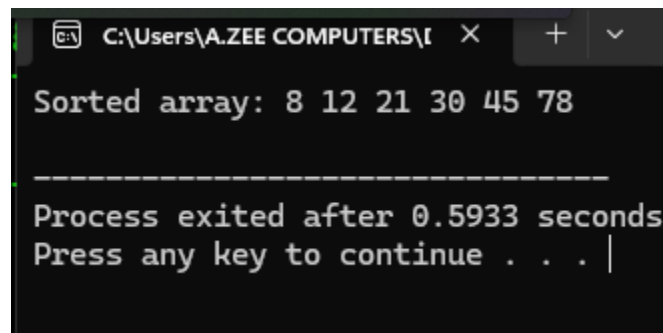
};

int main() {
    vector<int> numbers = {45, 12, 30, 21, 8, 78};
    MergeSort sorter;
    sorter.iterativeMergeSort(numbers);

    cout << "Sorted array: ";
    for (int num : numbers) {
        cout << num << " ";
    }
    cout << endl;
```

```
    return 0;  
}
```

OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I X + v  
Sorted array: 8 12 21 30 45 78  
-----  
Process exited after 0.5933 seconds  
Press any key to continue . . . |
```

3. Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.
Example 1: Input: intervals = [[1,3],[2,6],[8,10],[15,18]], Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
Example 2: Input: intervals = [[1,4],[4,5]], Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.

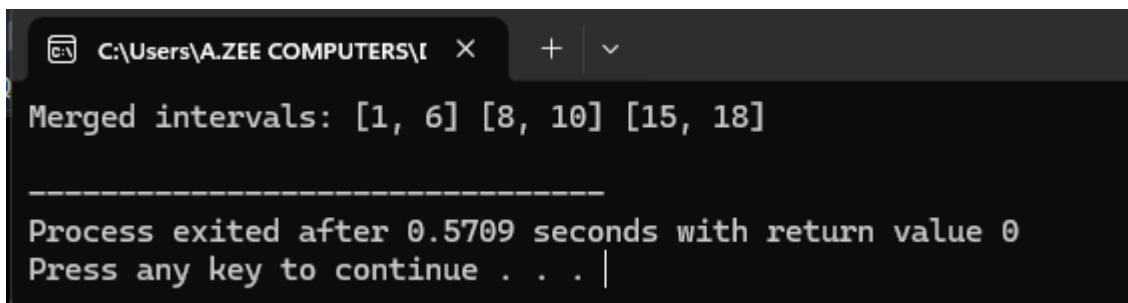
SOURCE CODE

```
#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
class IntervalMerger {  
public:  
    vector<vector<int>> mergeIntervals(vector<vector<int>>& intervals) {  
        sort(intervals.begin(), intervals.end());  
        vector<vector<int>> merged;  
  
        for (const auto& interval : intervals) {  
            if (merged.empty() || merged.back()[1] < interval[0]) {  
                merged.push_back(interval);  
            } else {  
                merged.back()[1] = max(merged.back()[1], interval[1]);  
            }  
        }  
  
        return merged;  
    }  
};  
  
int main() {
```

```
IntervalMerger merger;
vector<vector<int>> intervals1 = {{1, 3}, {2, 6}, {8, 10}, {15, 18}};
vector<vector<int>> result1 = merger.mergeIntervals(intervals1);
cout << "Merged intervals: ";
for (const auto& interval : result1) {
    cout << "[" << interval[0] << ", " << interval[1] << "] ";
}
cout << endl;

return 0;
}
```

OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I  X  +  v
Merged intervals: [1, 6] [8, 10] [15, 18]
-----
Process exited after 0.5709 seconds with return value 0
Press any key to continue . . . |
```

4. We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly 1. Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences. A subsequence of array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.

Example 1: Input: nums = [1,3,2,2,5,2,3,7], Output: 5 Explanation: The longest harmonious subsequence is [3,2,2,2,3].

Example 2: Input: nums = [1,2,3,4], Output: 2

Example 3: Input: nums = [1,1,1,1], Output: 0

SOURCE CODE

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

class HarmoniousSubsequence {
public:
    int findLHS(vector<int>& nums) {
        unordered_map<int, int> frequency;
        for (int num : nums) {
            frequency[num]++;
        }
    }
};
```

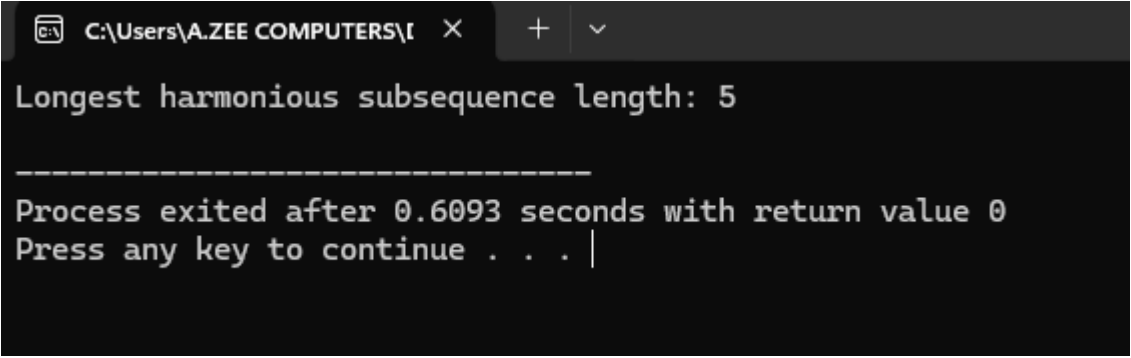
```
        int maxLength = 0;
        for (const auto& entry : frequency) {
            if (frequency.find(entry.first + 1) != frequency.end()) {
                maxLength = max(maxLength, entry.second + frequency[entry.first + 1]);
            }
        }

        return maxLength;
    }
};

int main() {
    HarmoniousSubsequence hs;
    vector<int> nums = {1, 3, 2, 2, 5, 2, 3, 7};
    cout << "Longest harmonious subsequence length: " << hs.findLHS(nums) << endl;

    return 0;
}
```

OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I  X  +  v

Longest harmonious subsequence length: 5

-----

Process exited after 0.6093 seconds with return value 0
Press any key to continue . . . |
```

5. There is a car with capacity empty seats. The vehicle only drives east (i.e., it cannot turn around and drive west). You are given the integer capacity and an array trips where $\text{trips}[i] = [\text{numPassengers}_i, \text{from}_i, \text{to}_i]$ indicates that the i th trip has numPassengers_i passengers and the locations to pick them up and drop them off are from_i and to_i respectively. The locations are given as the number of kilometers due east from the car's initial location. Return true if it is possible to pick up and drop off all passengers for all the given trips, or false otherwise.
- Example 1: Input: $\text{trips} = [[2,1,5],[3,3,7]]$, capacity = 4, Output: false
- Example 2: Input: $\text{trips} = [[2,1,5],[3,3,7]]$, capacity = 5, Output: true

SOURCE CODE

```
#include <iostream>
```

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

class CarPooling {
public:
    bool canCompleteTrips(vector<vector<int>>& trips, int capacity) {
        vector<int> passengers(1001, 0);

        for (const auto& trip : trips) {
            passengers[trip[1]] += trip[0];
            passengers[trip[2]] -= trip[0];
        }

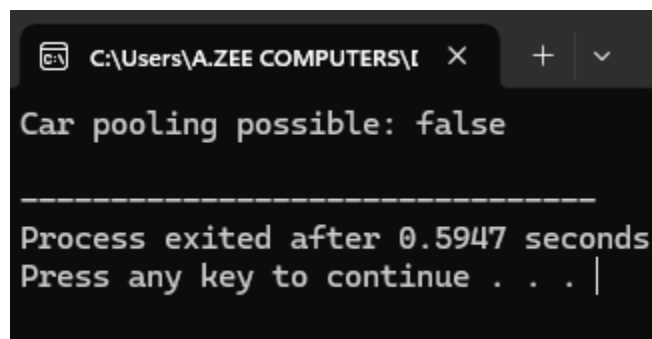
        int currentLoad = 0;
        for (int p : passengers) {
            currentLoad += p;
            if (currentLoad > capacity) return false;
        }

        return true;
    }
};

int main() {
    CarPooling carpool;
    vector<vector<int>> trips = {{2, 1, 5}, {3, 3, 7}};
    int capacity = 4;
    cout << "Car pooling possible: " << (carpool.canCompleteTrips(trips, capacity) ?
    "true" : "false") << endl;

    return 0;
}
```

OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I  X  +  v
Car pooling possible: false
-----
Process exited after 0.5947 seconds
Press any key to continue . . . |
```