

INTERVAL

greedy strategy: sort by the end time

Because ending earlier gives **more room** for future intervals. It's a classic greedy trick: choose the interval that **frees up time** as quickly as possible.

Problem – 57. Insert Interval

Medium



LeetCode

leetcode.com/problems/insert-interval

Problem Statement

- You are given an array of **intervals**, where **intervals[i] = [start_i, end_i]** and **newInterval = [start, end]**
- **newInterval** must be inserted into **intervals**
- Overlapping intervals must be merged
- Example

intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]] **newInterval** = [4,8]

Output: [[1,2],[3,10],[12,16]]

Solution – 57. Insert Interval

Medium



LeetCode

leetcode.com/problems/insert-interval

Solution

- Sort intervals by the first element (start)
- Initialize **result**
- Solve in three loops:
 1. While there is no overlap with **newInterval**, add to **intervals[i]** to **result**
 2. While it overlaps, merge **newInterval**
 3. While until the end intervals and add the remaining **intervals[i]**

Code – 57. Insert Interval

Medium



LeetCode

leetcode.com/problems/insert-interval

Code Time: $O(n)$ Space: $O(n)$ where n is the size of intervals

```
vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterval) {
    vector<vector<int>> result;
    int tupleIndex = 0;
    int totalTuples = intervals.size();
    // 1. check if it overlaps
    // 1 ----- 2
    //           4 ----- 8
    while (tupleIndex < totalTuples && intervals[tupleIndex][1] < newInterval[0]) {
        result.push_back(intervals[tupleIndex]);
        ++tupleIndex;
    }

    // 2. merge overlap. We already know there is an overlap here,
    // otherwise it should be sorted out in the previous step
    // 3 ---- 5
    //      4 ----- 8
    while (tupleIndex < totalTuples && intervals[tupleIndex][0] <= newInterval[1]) {
        newInterval[0] = min(newInterval[0], intervals[tupleIndex][0]);
        newInterval[1] = max(newInterval[1], intervals[tupleIndex][1]);
        ++tupleIndex;
    }
    result.push_back(newInterval);

    // 3. add remaining parts
    while (tupleIndex < totalTuples) {
        result.push_back(intervals[tupleIndex]);
        ++tupleIndex;
    }
    return result;
}
```

Problem – 56. Merge Intervals

Medium



LeetCode

leetcode.com/problems/merge-intervals

Problem Statement

- You are given an array of intervals, example:

```
intervals = [[1,3],[2,6],[8,10],[15,18]]
```

- Merge all overlapping intervals. So the output should be:

```
[[1,6],[8,10],[15,18]]
```

- Interval [1,3] was merged with [2,6]

Solution – 56. Merge Intervals

Medium



LeetCode

leetcode.com/problems/merge-intervals

Solution

- Sort the array based on the beginning of the interval
- In C++, when applying `sort(intervals.begin(), intervals.end())` the default comparator compares `vector<vector<int>>` lexicographically:
 - it first compares the first element [0] of each sub-vector
 - if those are equal, it compares the second element [1] and so on
- Go over each interval and compare
- `interval[i][begin] <= interval[i - 1][end]` ? then merge
- To merge, set the current `interval[i][begin]` to `interval[i - 1][begin]` and set the `interval[i][end]` to the maximum value between `interval[i][end]` and `interval[i - 1][end]`
- If no merge is necessary, push the previous interval to the result array
- Once the loop finishes, add the last element and return the result

Code – 56. Merge Intervals

Medium



LeetCode

leetcode.com/problems/merge-intervals

Code Time: $O(n \log n)$ Space: $O(n)$

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    if (intervals.empty()) return {};
    sort(intervals.begin(), intervals.end());
    vector<vector<int>> result;

    result.push_back(intervals[0]);

    for (int i = 1; i < intervals.size(); ++i) {
        vector<int>& current = intervals[i];
        vector<int>& previous = result.back();

        // check if they overlap, if so merge...
        // they're sorted, we know that:
        // previous[0] >= current[0]
        // 1 --- 3 (previous)
        // 2 ----- 6 (current)
        if (current[0] <= previous[1]) {
            // merge
            previous[1] = max(previous[1], current[1]);
        } else {
            result.push_back(current);
        }
    }
    return result;
}
```


Problem – 435. Non-overlapping Intervals

Medium



LeetCode

leetcode.com/problems/non-overlapping-intervals

Problem Statement

- ...

Solution – 435. Non-overlapping Intervals

Medium



LeetCode

leetcode.com/problems/non-overlapping-intervals

Solution

- ...

Code – 435. Non-overlapping Intervals

Medium



LeetCode

leetcode.com/problems/non-overlapping-intervals

Code Time: $O(n)$ Space: $O(n)$

■ ...