Arrays

Q14 Merge overlapping sub-intervals

i/p → [[1,3], [2,6], [8,10], [15,18]]
o/p → [[1,6], [8,10], [15,18]]

The first step would be to sort the intervals on the basis of starting point. Now simply take a pair & linearly traverse the input & if we found they are merging, then simply merge them if not merging then simply push that to the ans vector of vector.

Merging step

① [1,3]
[2,6] ⌐ max. of 3 and 6
3 >= 2 & hence merge ⇒ [1,6]

② [1,4]
[2,3] ⌐ max. of 4 & 3
4 >= 2 and hence merge ⇒ [1,4]

Dry run

{1,3}, {2,6}, {8,10}, {15,18}

1) Simply sort it on the basis of starting number.

{1,3}, {2,6}, {8,10}, {15,18}

2) Considering the pair {1,3}

(i) {1,3} is merging with {1,3} and hence we get {1,3}

(ii) {1,3} is merging with {2,6} & hence we merged to get {1,6}

(iii) Now {1,6} is not merging with {8,10} & hence push {1,6} in the ans vector.

3) Considering the pair {8,10}

(i) {8,10} is merging with {8,10} and hence we get {8,10}

(ii) {8,10} is not merging with {15,18} & hence push {8,10} in the answer vector.

4) Considering the pair {15,18}

(i) {15,18} is merging with {15,18} & hence we get {15,18}. Push this in ans vector. {left} Now the i/p vector/array has finished & we got ans vector as

{{1,6}, {8,10}, {15,18}}

Why the above algorithm work?
This is because of sorting & if overlapping intervals exist they would be consecutive only.

Code

```
vector <vector <int>> merge Intervals (
vector <vector <int>> & intervals) {
// Creation of ans
 vector <vector <int>> ans;
// Edge case → no interval
 if (intervals. size() == 0) {
        return ans;
 }
// Temp vector to be created
 vector <int> temp = intervals [0];
// Linear traversal
 for (auto i : intervals) {
        //Merging condition
        if (i [0] <= temp [1]) {
starting ↵                    ↳ ending
   temp[1]= max (temp [1], i [1]);
 }
        //Not merging case
        else {
            //Push the pair
            ans. push_back (temp);
            // Update or Consider new pair
            temp = i;
        }
 }
 // Push left interval
 ans. push_back (temp);
 // return
 return ans;                    ↗ Sorting
} Time complexity = O(n log n)   Space = O(n)
```