Merge Overlapping Intervals - Two Solutions

Solution 1: Brute-force style merging 1. Sort the intervals based on start time. 2. For each interval, expand the end as long as there is overlap with the next intervals. 3. Append the merged interval once no more overlap is found. 4. Skip intervals that are already covered. **Time Complexity:** O(n log n) for sorting + O(2n) for scanning and merging. **Space Complexity:** O(n), storing merged intervals.

```
class Solution:
    def mergeOverlap(self, arr):
        arr.sort()
        n = len(arr)
        ans = []

        for i in range(n):
            start, end = arr[i]
            if ans and end <= ans[-1][1]:
                continue

            for j in range(i + 1, n):
                if arr[j][0] <= end:
                    end = max(arr[j][1], end)
                else:
                    break

            ans.append([start, end])

        return ans
```

Solution 2: Optimized merging 1. Sort intervals based on start time. 2. Traverse intervals once: - If the current interval does not overlap with the last one in the result, add it as a new interval. - If it overlaps, extend the last interval's end to the max end. 3. Return merged intervals. **Time Complexity:** O(n log n) for sorting + O(n) for single traversal. **Space Complexity:** O(n), storing merged intervals.

```
class Solution:
    def mergeOverlap(self, arr):
        arr.sort()
        n = len(arr)
        ans = []

        for i in range(n):
            if not ans or arr[i][0] > ans[-1][1]:
                ans.append(arr[i])
            else:
                ans[-1][1] = max(ans[-1][1], arr[i][1])

        return ans
```