What is Merge Sort?

Merge Sort is a popular divide-and-conquer sorting algorithm that divides the input array into two halves, recursively sorts them, and then merges the sorted halves into one sorted result.

It is an example of a **stable sort** that guarantees **O(n log n)** performance in all cases — best, worst, and average.

Approach: Divide & Conquer

Divide: Split the array into two halves.
Conquer: Recursively sort each half using merge sort.
Combine: Merge the two sorted halves into one sorted array.

Key Concept: Merge Step

The key step is merging two sorted arrays efficiently into one sorted array. This is done using a two-pointer approach, comparing elements from both arrays and adding the smaller one into a new result array.

Time & Space Complexity

Time Complexity: O(n log n) — Divide takes log n steps and merging takes linear time.
Space Complexity: O(n) — Additional space is needed to store the merged arrays.

Dry Run Example: Merge Sort

Input: [5, 2, 4, 1]

Step 1: Divide

[5, 2, 4, 1] →
[5, 2] and [4, 1] →
[5] and [2] | [4] and [1]

Step 2: Merge

Merge [5] and [2] → [2, 5]
Merge [4] and [1] → [1, 4]

Step 3: Final Merge

Merge [2, 5] and [1, 4]:
Compare 2 and 1 → [1]
Compare 2 and 4 → [1, 2]
Compare 5 and 4 → [1, 2, 4]
Remaining elements → [1, 2, 4, 5]

Output: [1, 2, 4, 5]

JavaScript | C | C++ | Java | Python | C#

```javascript
/**
 * @param {number[]} nums
 * @return {number[]}
 */
var sortArray = function(nums) {
    if (nums.length <= 1) return nums;
    let mid = Math.floor(nums.length / 2);
    let left = sortArray(nums.slice(0, mid));
    let right = sortArray(nums.slice(mid));
    return merge(left, right);
};

function merge(left, right) {
    let res = [], i = 0, j = 0;
    while (i < left.length && j < right.length) {
        if (left[i] < right[j]) {
            res.push(left[i++]);
        } else {
            res.push(right[j++]);
        }
    }
    return [...res, ...left.slice(i), ...right.slice(j)];
}
```