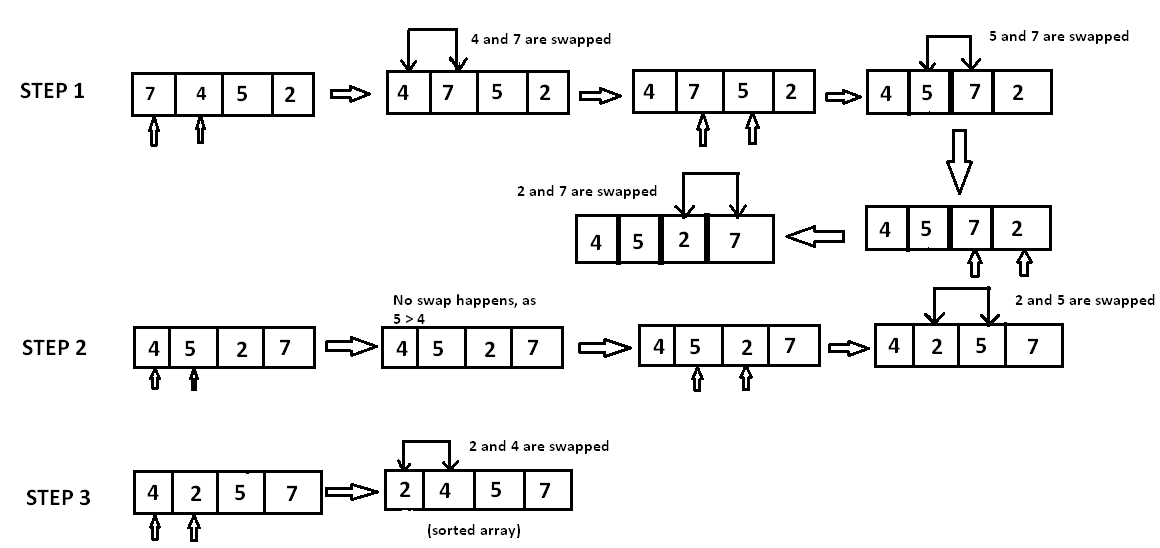
**Bubble Sort:**

The **bubble sort** makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place.

Let’s try to understand the pseudo code with an example: A [] = {7, 4, 5, 2}



**Time Complexity:**The complexity of bubble sort is O(n2) in both worst and average cases, because the entire array needs to be iterated for every element.

**Selection Sort:**

The Selection sort algorithm is based on the idea of finding the minimum or maximum element in an unsorted array and then putting it in its correct position in a sorted array.



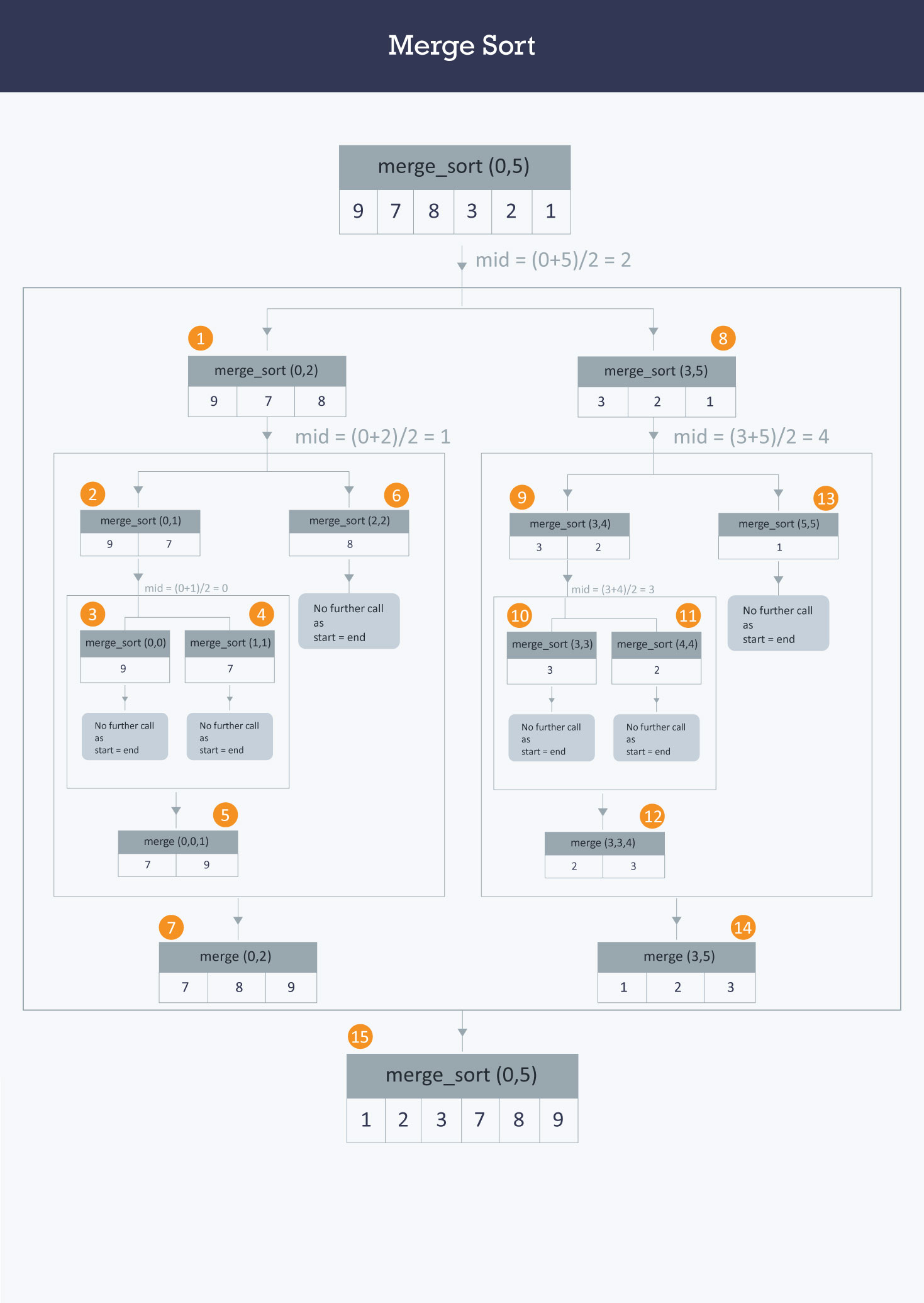
**Time Complexity:**

To find the minimum element from the array of N elements, N−1 comparisons are required. After putting the minimum element in its proper position, the size of an unsorted array reduces to N−1 and then N−2comparisons are required to find the minimum in the unsorted array.

Therefore (N−1) + (N−2) + ....... + 1 = (N⋅(N−1))/2 comparisons and N swaps result in the overall complexity of O(N2).

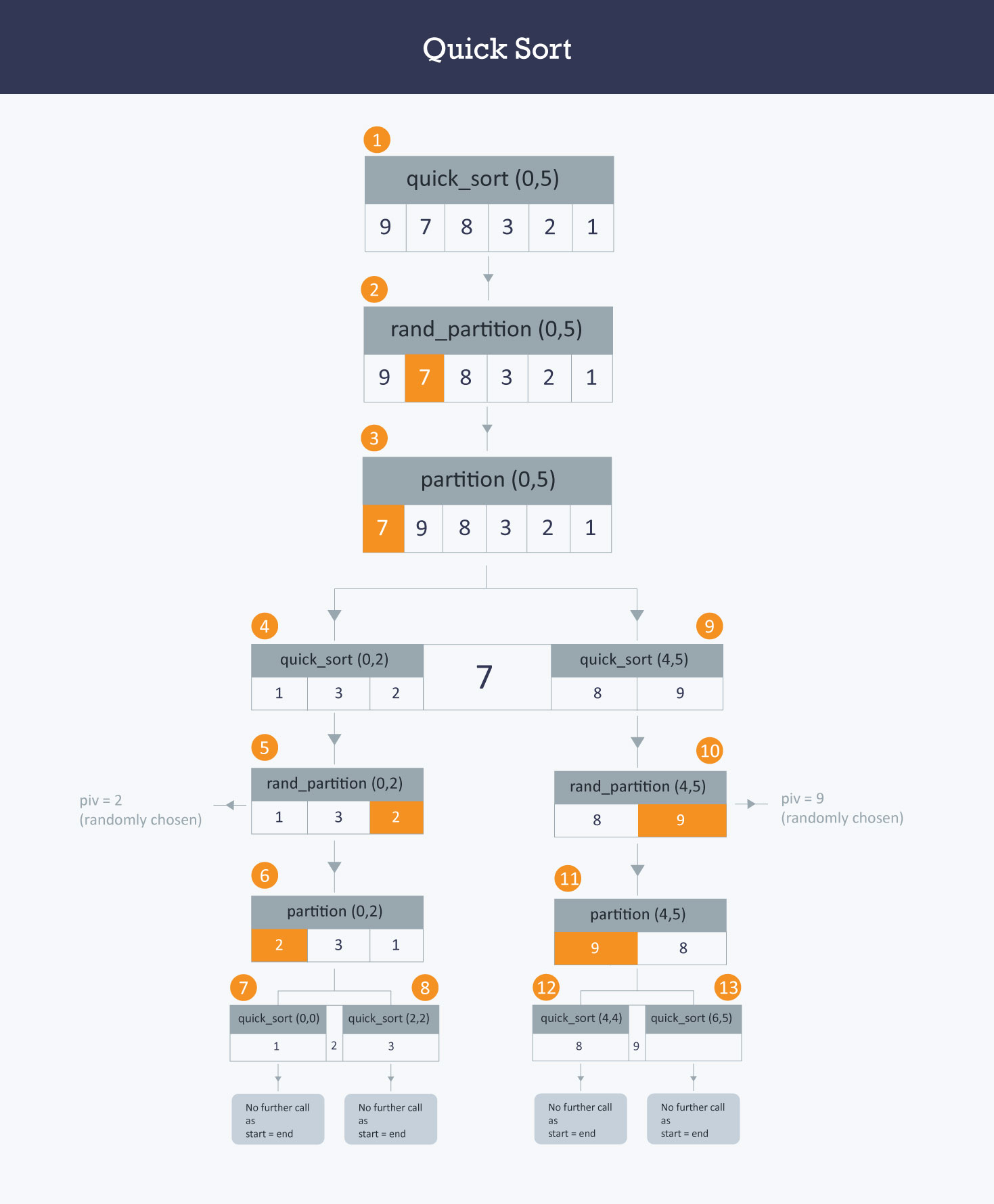
**Merge Sort:**

Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sub-list consists of a single element and merging those sub-lists in a manner that results into a sorted list.



**Time Complexity:**  
The list of size N is divided into a max of logN parts, and the merging of all sublists into a single list takes O(N)time, the worst case run time of this algorithm is O(NLogN)

**Quick Sort:** Quick sort is based on the divide-and-conquer approach based on the idea of choosing one element as a pivot element and partitioning the array around it such that: Left side of pivot contains all the elements that are less than the pivot element Right side contains all elements greater than the pivot. It reduces the space complexity and removes the use of the auxiliary array that is used in merge sort. Selecting a random pivot in an array results in an improved time complexity in most of the cases.



**Time Complexity:** Use randpartition() instead of partition() function in quicksort() function to reduce the time complexity of this algorithm.

Complexity The worst case time complexity of this algorithm is O(N2) , but as this is randomized algorithm, its time complexity fluctuates between O(N2) and O(NlogN) and mostly it comes out to be O(NlogN)