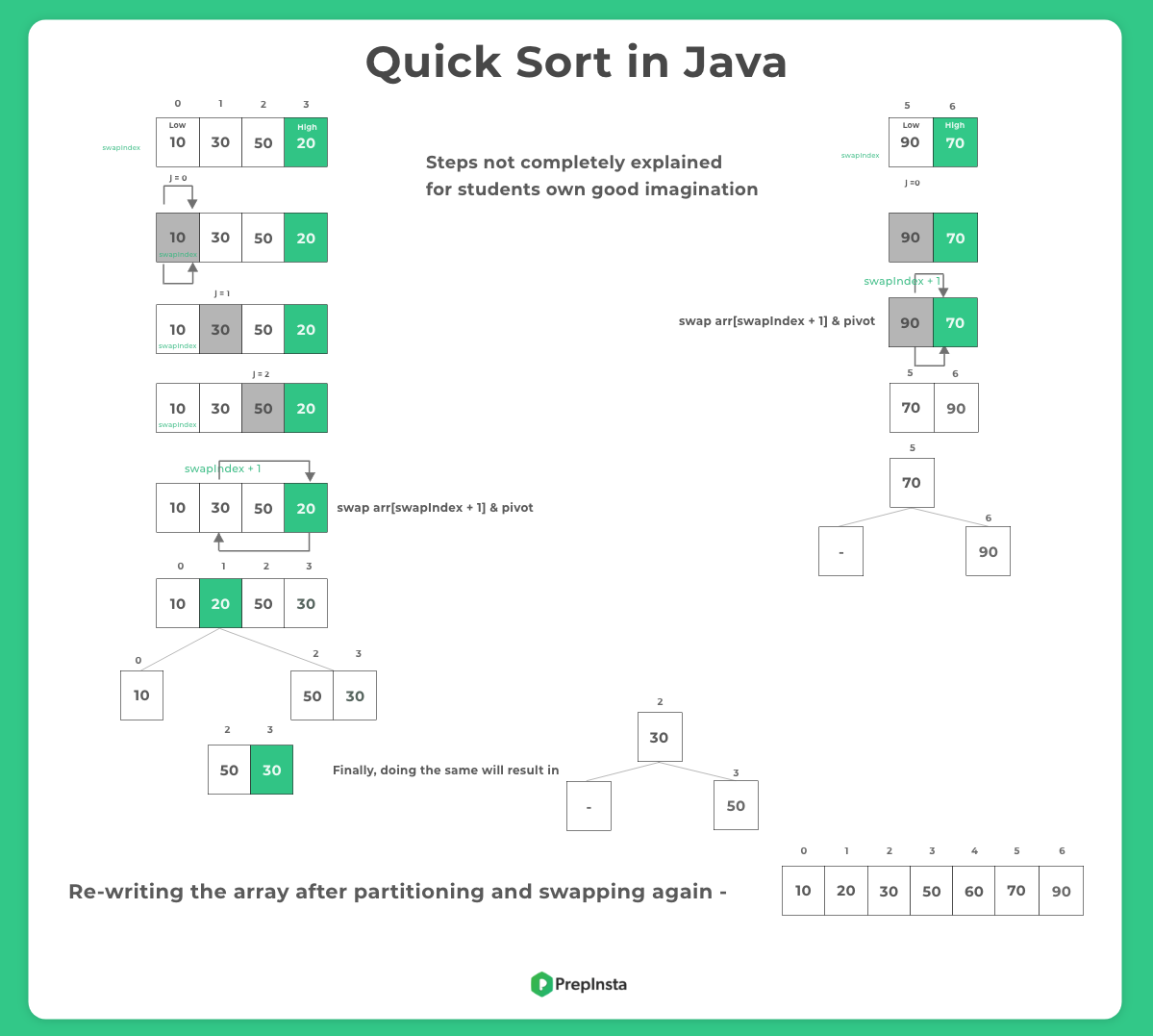
­­

**Quick Sort**

Similar to merge sort, quick sort also uses divide and conquer strategy. It selects an element as a pivot and partitions the array around it. It moves all the elements less than pivot to its left and all the elements greater than it to the right, then recursively sorts the subarrays.

**Time Complexity**

Time complexity of quick sort depends upon the pivot element.

**Worse Case**

When the array is already sorted, and we select the leftmost elements as a pivot, the algorithm recursively creates N subarrays of size N, N - 1, N - 2,.......,1. As the array is sorted, each subarray requires linear time for partitioning, **resulting in quadratic time complexity O(n^2)**.

**Best Case**

If we pick the median element as a pivot every time, then the algorithm creates logN subarrays (similar to merge sort). So for such a case, the time complexity is O(NlogN), which is the best case time complexity.

**Average Case**

There are many ways to avoid the worst case of quicksort, like choosing the element from the middle of the array as pivot, randomly generating a pivot for each subarray etc. By using these methods, we can ensure equal partitioning, on average. Thus, quick sort's average case time complexity is O(NlogN).

**Space Complexity**

Unlike merge sort quicksort doesn't require any extra space to store arrays but additional memory is required for creating stack frames in recursive calls. So, the space complexity again depends on the pivot.

**Worse Case (unbalanced partitioning)**

If we select the largest or smallest element as a pivot, there are in total N recursive calls, so the size of the recursion tree will be N. Thus, the space complexity for such a case is O(N) which is the worst case.

**Best Case (balanced partitioning)**

If we manage to partition the array in equal halves each time, then the size of the recursion tree is logN. So, in this case, the space complexity is O(N log N).

