Partition array into left and right sub-arrays such that elements in left sub-array < elements in right sub-array. Recursively sort left and right sub-arraysConcatenate left and right sub-arrays with pivot in middle

How to Partition the Array:

Choose an element from the array as the pivot

Move all elements < pivot into left sub-array and all elements

pivot into right sub-array

Sort the array containing:

9(pivot) 16 4 15 2 5 17 1

Partition 4 2 5 1< 9(pivot) <16 15 17

Partition 2 14 5 15 16 17

1 2 5 15 17

Concatenate 1 2 4 5 15 16 17

Concatenate 1 2 4 5 9 15 16 17

Best Case Performance: Algorithm always chooses best

pivot and keeps splitting sub-arrays in half at each recursion

$T(0) = T(1) = O(1)$ (constant time if 0 or 1 element)

For $N > 1$, 2 recursive calls plus linear time for partitioning

$T(N) = 2T(N/2) + O(N)$ (Same recurrence relation as Mergesort)

$T(N) = O(N \log N)$

Worst Case Performance: Algorithm keeps picking the worst pivot – one sub-array empty at each recursion

$T(0) = T(1) = O(1)$

$T(N) = T(N-1) + O(N)$

$= T(N-2) + O(N-1) + O(N) = \ldots = T(0) + O(1) + \ldots + O(N)$

$T(N) = O(N2)$

Great answers here. I'm adding few more points for justifying why QuickSort is better than other sorting algorithms with same asymptotic complexity O(nlogn) (merge sort, heap sort). Even though quicksort has O(n^2) in worst case, it can be easily avoided with high probability by choosing the right pivot.