Kevin Jamanar

"QuickSort"

Following the Divide & Conquer Paradigm, 2 steps are used as opposed to 3.

1) Divide (Split array into 2 parts via a randomly chosen pivot)

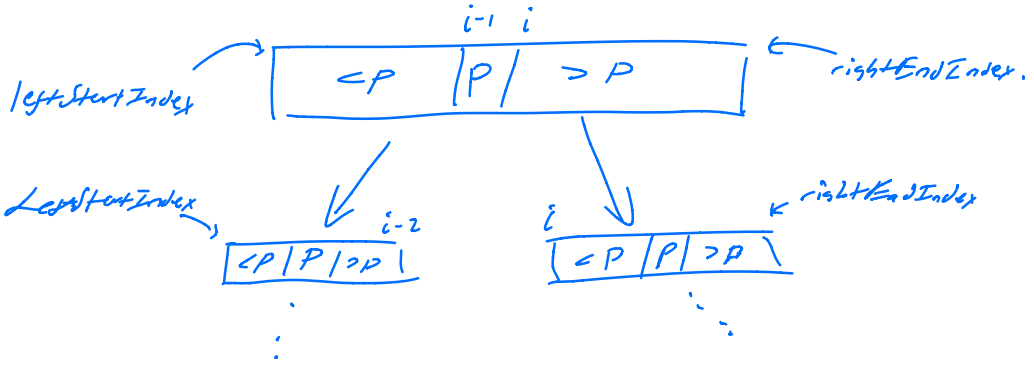
2) Conquer (recursively approach each subproblem)

There is no "merge" or "combine" step therefore requiring no extra space be used, except for the recursive calls.

Partitioning of the array:

The array is sorted in such a way that elements less than the pivot go the left & elements greater than the pivot go to the right.

This leaves the pivot sorted.



Similar to mergeSort (although simpler as there is no combine step for this Divide & Conquer algorithm.)

Picturing the above diagram in mind is enough to code this algorithm.

⇓

1) Partition the array as desired above (There are multiple partition subroutines)

This should take linear time & use no extra memory. If we ended up using extra memory we may as well implement MergeSort.

2) Call QuickSort sub-routine on the left sub-array & right sub-array respectively.

Worst-Case = $O(n^2)$ {
- all element sorted
- pivot is always the leftmost / rightmost index

Leading to a $n + n-1 + n-2 + \dots + 1 \approx n^2$ # of comparisons.

Avg-Case - $\Theta(n \log n)$ (with randomly chosen pivots)

↳ this can be proven via Linearity of Expection where the random variable is the # of comparisons in the partitioning function as it requires the most work.