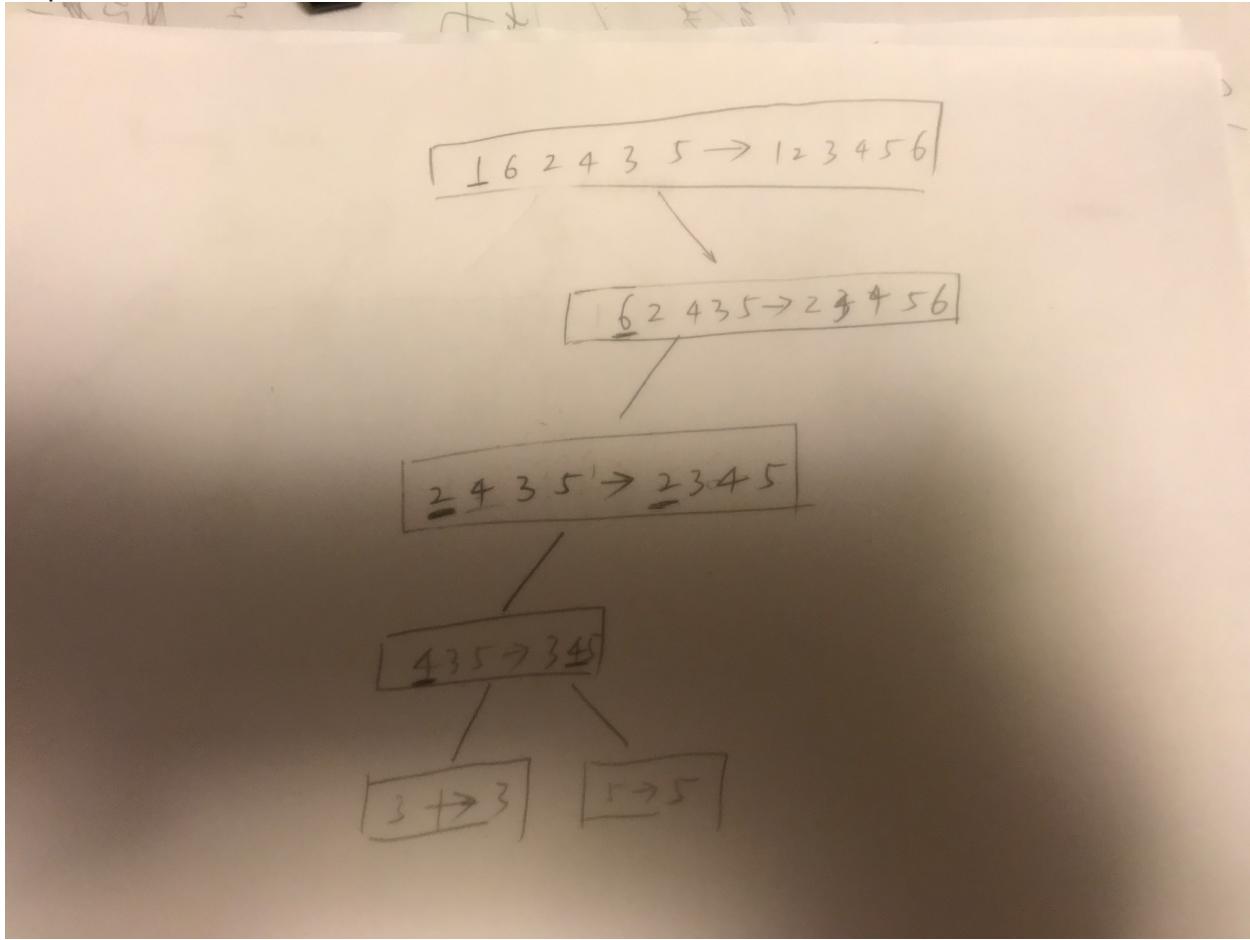


Yuliang Jin 986381

Problem 1

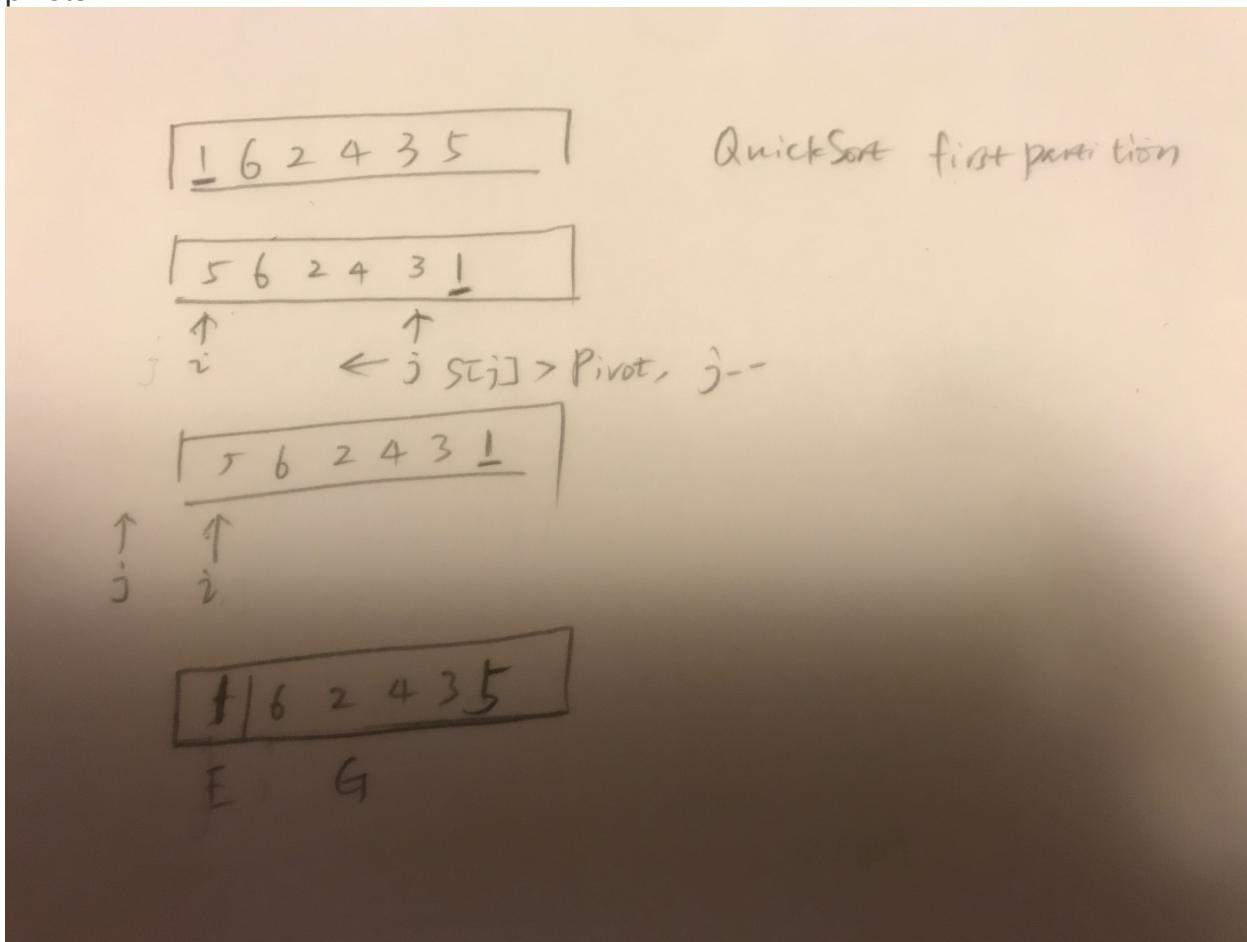
Show all steps of QuickSort in sorting the array [1, 6, 2, 4, 3, 5]. Use leftmost values as pivots at each step.



Problem 2

Show all steps of In-Place QuickSort in sorting the array [1, 6, 2, 4, 3, 5] when doing first partition. Use leftmost values as

pivots.



Problem 3

In our average case analysis of Quicksort, we defined a good self-call to be one in which the pivot x is chosen so that number of elements $< x$ is less than $3n/4$, and also the number of elements $> x$ is less than $3n/4$. We call an x with these properties a good pivot. When n is a power of 2, it is not hard to see that at least half of the elements in an n -element array could be used as a good pivot (exactly half if there are no duplicates). For this exercise, you will verify this property for the array $A = [5, 1, 4, 3, 6, 2, 7, 1, 3]$ (here, $n = 9$). Note: For this analysis, use the version of Quicksort in which partitioning produces 3 subsequences L, E, R of the input sequence S .

Which x in A are good pivots?

2, 3, 4, 5 in A are good pivots.

Is it true that at least half the elements of A are good pivots?

It's true at least half the elements of A are good pivots.

Problem 4

Interview Question. Give an $o(n)$ ("little-oh") algorithm for determining whether a sorted array A of distinct integers contains an element m for which $A[m] = m$. You must also provide a proof that your algorithm runs in $o(n)$ time. Because it's already sorted, so in one for loop each element could be checked the time complexity of which is $O(n)$. For $o(n)$ algorithm, binary search could be used to check the elements.

Problem 5

Devise a pivot-selection strategy for QuickSort that will guarantee that your new QuickSort has a worst-case running time of $O(n \log n)$. Explain why your new QuickSort has this running time. (Hint: Use QuickSelect to pick the pivot.) In the QuickSort if the pivot was not selected properly then the worst case will happen. So in order to avoid the worst case, the pivot value selected should guaranteed to be not the unique max or minimum one. So in this algorithm, we choose the pivot as the $(n/2)$ th smallest value. So L and E could always tends to be equal. This is even better than average performance of the quick sort.