Homework-1: Quick Sort

Your Name Student ID: 19100000

With the following set of problems we will develop a divide and conquer based algorithm for sorting an array of n numbers. This algorithm is called QUICK-SORT and is one of the most widely used algorithms for sorting in large applications.

We discussed the notion of and an algorithm for $rank_A(z)$ (the number of elements in A that are smaller than z). We can similarly find the number of elements in A that are larger than z, this is equal to $n - rank_A(z)$ if $z \notin A$ and $n - rank_A(z) - 1$, if $z \in A$ and A contain distinct elements. More generally, the number of elements in A that are larger than z is given by $n - rank_A(z) - freq(z)$ if z appears $freq(z)$ number of times in A.

Problem 1. We want to extend the algorithms for $rank_A(z)$ to actually find the elements that are smaller than z (and larger than z).

Suppose $z \notin A$, make two arrays X and Y (of appropriate length) such that X contains all and only those elements that are smaller than z and Y contains all the larger elements. Your algorithms should not make more than n comparisons.

Problem 2. Suppose $z \in A$, modify the same array A so as z is at its sorted location (the location of z when we sort A). Other elements in A can be in arbitrary locations (of course all elements of A must be somewhere). You should not use more than n comparisons and a few extra variables.

Problem 3. Suppose $z \in A$, modify A so as it has the properties that z is at its sorted location and all elements to the left of A are smaller than z and all elements to the right of z are larger than z. This is the combination of the both parts above, except that you are not allowed to use extra arrays but only a few variables. Again your algorithm should not make than n comparisons.

Problem 4. Extend the above idea into an algorithm to sort A. To sort an array use $A[1]$ to play the role of z, partition around this z. Recursively sort the left part (elements that are smaller than and to the left of z) and the right part. There is not much needed to combine the outputs of the two recursive calls.

The running time of QUICKSORT clearly depends on the sizes of the subproblems, which depends on what the rank of the first element is.

Problem 5. Determine what is the worst case for the QUICKSORT algorithm and derive a recurrence relation for the runtime in the worst case case and its closed form.

Problem 6. Describe an example input for which the worst case behavior of the QUICK-SORT algorithm occurs.

Problem 7. Determine what is the best for the QUICKSORT algorithm and derive a recurrence relation for the runtime in the best case and its closed form.