Name: Georges Hatem

CS 590 Homework 7
Reinforcement Exercises

Due Date: March 13, 2022

Problem 8.5.4:

Quick sort:

Best = O(n * logn)

Average = O(n * logn)

Worst = O(n ^ 2), when the elements are already sorted in ascending or descending order.

Since you are taking middle element as pivot which means we have elements in sorted manner.
In worst case if this middle element is placed at starting or ending of list after partition procedure. then recursive equation will be like T(n)=T(n-1)+o(n) which result in time complexity as $O(n^2)$.

The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

1. **Choose a pivot value. We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.**
2. **Partition. Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.**
3. **Sort both parts. Apply quicksort algorithm recursively to the left and the right parts.**

Problem 9.5.1:

As defined in Section 9.2, stability has been defined as follows:

We say that a sorting algorithm is stable if, for any two items (K_i, e_i) and (K_j, e_j) of S such that $K_i = K_j$ and (K_i, e_i) precedes (K_j, e_j) in S before sorting, we have that item (K_i, e_i) also precedes item (K_j, e_j) after sorting.

Merge-Sort:

Considering the statement above, Merge-sort is a stable algorithm. The reason behind that lies in the merge function. As long as the merge function moves equal left elements before right elements, merge-sort will be stable.

Quick-Sort:

Considering the definition about stability above, quick-sort is not a stable algorithm. The reason behind that is because the swapping of factors is done in step with pivot's function (without thinking about their unique positions).

Bubble-Sort:

Considering the definition about stability above, bubble-sort is a stable algorithm. For example, imagine an array with 2 instances of the number 3 (A = [7, 2, 3, 3]). When comparing the 2 instances of 3, the algorithm will not swap them if the one on the left is not larger than the one on the right. Thus, their relative order would remain the same.

Heap-Sort:

Based on the above definition of stability, heap-sort is not a stable algorithm. The reason behind that is that the final sequence of the results from heapsort comes from removing items from the created heap in purely size order (based on the key field). Any information about the ordering of the items in the original sequence was lost during the heap creation stage, which came first.