CSCI 4452.2 2015 Assignment #4

Show all your work.

(1) (a) Give an analysis to determine the average cost of sequential search assuming that there is a 25% chance that the target will not be found in the list, and thus that there is a 75% chance that it will be in the list. In the latter case, assume that any of the list locations are equally likely to be the target.
(b) Repeat part (a) except this time assume that when the target is in the list, there is a 75% chance that it will be in the first half of the list.

(2) (a) Give an iterative (i.e., loop-based) pseudo-code algorithm for Selection Sort. On the ith pass this algorithm should search for the ith largest element from the unsorted part of the list and, at the end of this search, it should swap this element with whatever element is currently occupying the ith location. The swap function should have the form $swap(A, i, j)$, where A is the list and the ith and jth elements are to be swapped.
(b) Provide an analysis of this algorithm to determine the total number of comparisons for a list of size N.
(c) Provide an analysis to determine the total number of swaps performed by this algorithm.
(d) A typical swap operation to swap list elements i and j would copy element i to a temporary location, copy element j to the ith location, and then copy element i from the temporary location to the jth location. A "smart" swap would first check to see if $i = j$ and in such a case it would simply return with out doing the work of copying the elements. Assume a smart swap and analyze the Selection Sort algorithm to determine how many swaps are performed, assuming that we will not count swaps for which $i = j$.

(3) Write down a **recursive** pseudo-code algorithm for the Selection Sort. Explain how to obtain the corresponding cost function to count the number of comparisons for a list of size N, and then perform an analysis of this cost function *using the characteristic polynomial approach*.

(4) When you look closely at the Insertion Sort you will notice that the insertion of the next element into the already sorted list basically does a sequential search to determine where the element should be inserted. Consider a variation of the standard Insertion Sort algorithm that does a binary search on the already sorted part of the list to determine the location where the element should be inserted. Assuming that the keys are unique, note that a standard binary search would always return a failure. Therefore assume that you are using a slightly modified version of binary search that returns the location where the key to be inserted belongs. What is the average case cost for the number of comparisons for this new version of Insertion Sort?