CSCI 2312: Discrete Structures II: Discussion: Timing Analysis

We will attempt to understand how to compare the amount of time taken by one algorithm with that taken by another. First, we note that we cannot measure the exact time, because that would depend on the hardware used. What we can do is compare the number of operations required. We often focus on the operations that take longer. For example, comparison and assignment are assumed to take less time than addition and subtraction, which, in turn, would take less time than multiplication.

1. Let's consider an algorithm for computing the maximum of an array and rearranging the array so that a maximum value is the first one. Pseudocode below (sketches the algorithm and will not compile).

```
Max(A, n) /*rearrange array A, of size n, so that a max value is the first one*/
/*compare first element with every other value and swap if any value is larger*/
for i=0:n-1 i++
   if A[0] ≤ A[i]
      swap(A[i], A[0])
return A
```

How many comparisons were performed? How many assignments? How many times does the loop execute? Can you make this more efficient? How significant is this change in efficiency?

2. Now let's consider Selection Sort to sort an array. We can use MaxBetter(A):

```
MaxBetter(A, n) /*rearrange array A of size n so that a max value is the first one*/
/*compare first element with every other value and swap if any value is larger*/
for i=1:n-1 i++
   if A[0] < A[i]
      swap(A[i], A[0])
return A

SelectionSort(A) /*returns array A, sorted in descending order*/
/*Beginning with the ith element, repeatedly place max element at the top of the rest
of the array*/
for i=0:n-2 i++
   MaxBetter(A[i:n-1], n-i)
return A
```

How many worst case swaps in SelectionSort?

3. Consider pseudocode for `BubbleSort`

```
BubbleSort(A,n) /*returns array A, of size n, sorted in descending order*/
/*find A[i] of sorted array by swapping neighboring values*/
for i=n-1:1 i-
  for j=0:1-i
     if A[j] > A[j+1]
        swap(A[j], A[j+1])
return A
```

How many worst case swaps?

4. Consider `InsertionSort.` Each element of a given unsorted array `A` is placed in its correct position in sorted linked list `B.` How many comparisons are performed in the worst case?