CSCI 430: Homework 3

John E. Buckley III

September 10, 2017

1 2.2-1

As the problem is worded, $\Theta(n^3/1000 - 100n^2 - 100n + 3)$ would be a suitable answer. However, lets assume that the problem meant to ask to find all values of x for which $n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^x)$. For this we will take the limit of $(n^3/1000 - 100n^2 - 100n + 3)/(n^x)$ as n approaches infinity. Since this is a rational function we can compare the degrees of the numerator and the denominator following different rules for when the degree of the numerator is less than, greater than, or equal to the degree in the denominator. So lets substitute x for 3 which will mean the degrees of the numerator and denominator are the same, in which case we divide the coefficients of the terms with the largest exponent, which are both 1. $1/1 = 1$ which means that this function approaches 1 when x=3. Knowing this, it means that when x is less than 3 it will approach infinity and if x is greater than 3 we approach 0. $n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^x)$ if and only if x=3 thus $\Theta(n^3)$.

2 2.2-2

Pseudocode:

```
SelectionSort(A):
1.)     for $i = 1$ to A.length-1
2.)         smallest=i
3.)         for $j = i + 1$ to A.length
4.)             if $A[j] < A[smallest]$
5.)                 smallest=j
6.)         $temp = A[i]$
7.)         $A[i] = A[smallest]$
8.)         $A[smallest] = temp$
```

Loop Invariant:

At the start of each iteration of the outer for loop, (line 1), the subarray $A[1...i-1]$ contains the smallest i-1 elements of the array sorted in non-decreasing order. At the start of each iteration of the inner for loop, (line 3), $A[smallest]$ is the smallest number in the subarray $A[i...j-1]$.

Why n-1:

If we were to do it n times instead of n-1 times, we would end up with a redundant step that sorts a single-element array, and since any single element array is already sorted we use n-1 which will leave the algorithm with two elements to compare.

Running Times:

In the best case, where the array is already sorted, then the if statement on line 4 is never invoked. Using the same system as the running time example in class for insertion sort, which is also explained on pages 26-27 in the book, we find that the running time for the best case scenario is $\Theta(n^2)$.
In the worst case, where the array is in reversed sorted order, then the if statement on line 4 is invoked on every occasion which also gives us $\Theta(n^2)$.

3 2.2-3

On the average, for example, if A.length== 10 and we call search 10 times with the same inputs, then it will check element #1 10 times, and element #2 9 times, and so on. So on average it will check $A.length + A.length - 1 + A.length - 2 + ...)/A.length$ which is about half of all the elements. Thus, on the average case, assuming the element being searched for is equally likely to be any element in the array, it will check roughly half of the elements.
On the worst case, v will match with the last element of A, which means it will have to check all of the elements in A.
In both cases the run time is $\Theta(n)$. Which could have been assumed since this is a linear search.