

CS375 Small Assignment 3 Solutions

Due Sep 26th 10PM ET

Consider the following pseudo code for selection sort:

```
SelectionSort(A[0, ..., n-1]):  
  for i in range(n):  
    mIndex = i  
    for j in range(i + 1, n):  
      if A[j] < A[mIndex]:  
        mIndex = j  
    swap A[i] and A[mIndex]
```

1. What is the worst case runtime of the above function? (I am looking for a Θ bound here, so this is really two questions in one)
2. What is the best case runtime of the above function? (I am looking for a Θ bound here, so this is really two questions in one)
3. Using the loop invariant method we've discussed in class, show that the above algorithm correctly sorts A.

Solution: This question ended up being a bit silly, because both the best and worst case runtime were the same. In fact, the runtime of this function is entirely independent of the input: it has exactly the same runtime on any list of length n . The total number of iterations due to the form loops is

$$\begin{aligned} \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} c &= c \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = c \sum_{i=0}^{n-1} (n-1-(i+1)+1) \\ &= c \sum_{i=0}^{n-1} (n-i-1) = c \frac{(n-1)n}{2} = \Theta(n^2) \end{aligned}$$

As a result, both the worst and best case runtimes are $\Theta(n^2)$. However, had I chosen a better question, in principle you would need:

1. An upperbound on the worst case runtime by looking at the code (which in this case would give you $O(n^2)$) and a lowerbound on the worst case runtime by providing any input on which the runtime would match the upperbound you gave (in this case, literally any input will work)

2. An upperbound on the best case runtime by analyzing any reasonable input (in this case, any input will work because the runtime is independent of the input) and a matching lowerbound as observed from the pseudocode.

Lastly, regarding the correctness: let's use the technique of loop invariants to show correctness. Specifically, let's show that the above code maintains the invariant **prior to the start of iteration i** :

1. $A[0], \dots, A[i - 1]$ contains exactly the i smallest items in all of A
2. The array is sorted from $A[0]$ to $A[i - 1]$.

On the first iteration, this is vacuously true (note that if $i = 0$ then $i - 1 = -1$, so this statement basically says nothing). Now, we'll show that if the invariant is held at the start of a particular iteration i , it will still be true prior to the start of the next iteration ($i + 1$). Since it's true at the start of iteration i , we know that $A[0], \dots, A[i - 1]$ are the i smallest items in A in sorted order. The algorithm then finds the next smallest item from $A[i], \dots, A[n - 1]$ and swaps it with whatever is at index i , guaranteeing us the first invariant is held for $i + 1$. Note that since it's the next smallest item, it must be the case that now $A[i - 1] \leq A[i]$, which shows that the second invariant is held. As such, at the start of the next invariant for some iteration i , it is held for the next iteration at $i + 1$. Hence is true for all iterations, in particular at iteration n , at which point by the second invariant we will have the array is sorted from $A[0]$ to $A[n - 1]$ as desired.