# Lab 5 – Proofs and Problems

## Problem 1: Pseudocode analysis (3 points)

Function1 (A)

1 **for** i = 1 **to** *A.length*−1

2 minIndex = i

3 for j = i+1 to *A.length*

4 if A[j] < A[minIndex] and j ≠ minIndex

5 minIndex = j

6 swap A[i] with A[minIndex] ​

1. Count the number of operations in this algorithm using CLRS Chapter 2.2’s analysis of insertion sort **worst case** as a model. State the number of operations (use Sigma notation as appropriate; do not simplify). Ignore constants (c1, c2, etc.)

Line 1 is n times

Line 2 is n-1 times

3: sigma(t\_j), j from 2 to n

4: sigma(t\_j-1), j from 2 to n

5: sigma(t\_j-1), j from 2 to n

6: sigma(t\_j-1), j from 2 to n

1. Simplify your statement in (1) to a single algebraic expression in terms of n by summing all the operations. Ignore constants (c1, c2, etc.)

T(n) = n + (n-1) + sigma(t\_j) + sigma(t\_j-1) + sigma(t\_j-1) + sigma(t\_j-1) = n + (n-1) + (n) + (n-1) \*3

1. State which big O category (2) belongs in , and prove the expression in (2) belongs to that big O category (use the method from the video Asymptotic Notation Part 1). (From now on whenever we ask you to “prove the big O” of an expression, this is the technique we’re referring to.)

O(n\*\*2)

## Problem 2: Proofs of correctness (2 points)

Describe all of the steps required to prove the correctness of a sorting algorithm using a loop invariant, and explain why each step is necessary. Use the CLRS proof of insertion sort as a model, but describe each of the steps generally for proving the correctness of any sorting algorithm, and explain why each step is necessary. Remember that you’re proving correctness by proving that the algorithm solving the sorting problem, so you will need to mention what’s necessary to solve the sorting problem.

-Intialization: before the first loop iteration, subarray A[1..j-1] consisnt only single element A[1], it is sorted before first iteration of loop. It is true prior to first iteration of the loop.

-Maintenance: showing that each iteration maintain invariant. Next, we tackle the second property: showing that each iteration maintains the loop invariant. Informally, the body of the for loop works by moving AŒj 1, AŒj 2, AŒj 3, and so on by one position to the right until it ﬁnds the proper position for AŒj  (lines 4–7), at which point it inserts the value of AŒj  (line 8). The subarray AŒ1 : : j  then consists of the elements originally in AŒ1 : : j , but in sorted order. Incrementing j for the next iteration of the for loop then preserves the loop invariant.

-termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.Finally, we examine what happens when the loop terminates. The condition causing the for loop to terminate is that j > A:length D n. Because each loop iteration increases j by 1, we must have j D n C 1 at that time. Substituting n C 1 for j in the wording of loop invariant, we have that the subarray AŒ1 : : n consists of the elements originally in AŒ1 : : n, but in sorted order. Observing that the subarray AŒ1 : : n is the entire array, we conclude that the entire array is sorted. Hence, the algorithm is correct.

## Problem 3: Bubblesort (2 points)

1. Suppose that Bubblesort on an array A uses the following pseudocode:

1 for i = 1 to A.length – 1

2 for j = A.length downto i + 1

3 if A[j] < A[j-1]

4 exchange A[j] with A[j-1]

1. A loop invariant is a complete sentence that describes a condition that is true for every iteration of a loop, and that describes a property that helps solve or make progress towards creating the solution for the algorithm’s problem. An example of a loop invariant can be found on page 18 of CLRS in their example proof of correctness. The loop invariant is the complete sentence inside the red square.

For 3.1, state precisely the loop invariant for the *for* loop in lines 2-4 in the pseudocode above, and prove that this loop invariant is maintained for the entire algorithm above. Use the CLRS loop invariant proof in Chapter 2 as a model for this proof.

Loop invriant: At the start of each iteration of the for loop of lines 1–8, the subarray A[i+1, A.length] consists of the elements originally in A[i+1, A.length], but in sorted order.

1. Using the termination condition of the loop invariant that you proved in part (b), state a loop invariant for the *for* loop in lines 1-4 that will allow you to prove the output is correct in the ways stated in (a). Use the CLRS loop invariant proof in Chapter 2 as a model for this proof.

## Problem 4: Best Case Performance (1 point)

What modifications could you make to any sorting algorithm to have a good best case running time, returning a definitely sorted array in O(n)? Remember that, in the best case, you can assume the data is in the most beneficial format that you need for your solution to work. Generally this will only add 1-2 lines of code to your implementation.

## If a subarray is already sorted, we can skip checking it.