Student Name: Monika Szucs

The goals of this lab are to:

Give you the background skills that are required for you to apply the *General Plan for Analyzing Time Efficiency of Non-recursive Algorithms* (page 62 of your textbook). Ultimately this is what you need to be able to do.

Today's background skills include:

* ability to express an algorithm in pseudo-code
* identification of the basic operation in an algorithm
* ability to set up summations to represent the number of times a basic operation is executed
* manipulation of summations (transform into a closed-form formula)

Note: this is an individual lab assignment in that you have to learn the material. Please feel free to discuss answers or to work through some questions with a partner if this helps you learn the material – but you do need to have your own work on your own lab at the end of the class.

Grading:

* This is a graded lab out of 15.
* Please type your answers with Blue color or take a photo of your handwritten answers.

Consider the following algorithm from your textbook (page 23):

1. Algorithm CCS (A[0..n−1])

2. for i ← 0 to n−1 do

3. Count[i] ← 0

4. for i ← 0 to n-2 do

5. for j ← i+1 to n−1 do

6. if A[i] < A[j]

7. Count[j] ← Count[j]+1

8. else

9. Count[i] ← Count[i]+1

10. for i ← 0 to n−1 do

11. S[Count[i]] ← A[i]

12. return S

1. Consider only lines 1 to 3 of this algorithm. Call these 3 lines "Part A".

1. Algorithm CCS (A[0..n−1])

2. for i ← 0 to n−1 do

3. Count[i] ← 0

1. [1 mark] what does Part A of the algorithm do?

Line 1 indicates that the algorithm is called CSS and it will take an array A with the elements index that starts at 0 and ends at n – 1 where n is the length of the array.

Line 2 has a fore loop that will iterate over the indices of the array A that start at 0 and end at n – 1. This means the loop will run n times.

Line 3will initialize the new array called Count to be set to 0 for all indices i that starts at 0 and ends at n – 1.

1. [1 mark] Assume that Part A is all there is to the algorithm. What is the basic operation in Part A, and on what line does it occur?

The basic operation will be on line 3 and it is Count[i] <- 0

1. [1 mark] set up a summation that counts the number of times the basic operation is executed in Part A for an input array of size *n* and solve it. *Note: Appendix A (pg 476) contains some very useful formulas to help you solve summations to closed form.*

1. Algorithm CCS (A[0..n−1])

2. for i ← 0 to n−1 do

3. Count[i] ← 0

2. Consider only lines 4 to 9 of the algorithm from question 1. Call these 6 lines "Part B".

4. for i ← 0 to n-2 do

5. for j ← i+1 to n−1 do

6. if A[i] < A[j]

7. Count[j] ← Count[j]+1

8. else

9. Count[i] ← Count[i]+1

1. [1 mark] what does Part B of the algorithm do? Use the following array as test data, showing the contents of Count [] after each assignment execution (line 7 or 9) of the inner for loop.   
     
   Given: A= [42, 17, 18, 23, 37, 9]

Before starting Part B: Count= [ 0, 0, 0, 0, 0, 0]

Fill in:

Count = [\_5\_, \_0\_, \_0\_, \_0\_,\_0\_, \_0\_], i = 0

Count = [\_5\_, \_1\_, \_0\_, \_0\_,\_0\_, \_0\_], i = 1

Count = [\_5\_, \_1\_, \_2\_, \_0\_,\_0\_, \_0\_], i = 2

Count = [\_5\_, \_1\_, \_2\_, \_3\_,\_0\_, \_0\_], i = 3

Count = [\_5\_, \_1\_, \_2\_, \_3\_,\_4\_, \_0\_], i = 4

Count = [\_5\_, \_1\_, \_2\_, \_3\_,\_4\_, \_0\_], i = 5

1. [1 mark] what is the basic operation in Part B, and on what line does it occur?

The basic operation for part B is line 7 or 9:

Line 7 Count[j] <- Count[j]+1 when A[i] < A[j]

Line 9 Count[i] <- Count[i]+1 when A[i] >= A[j]

1. [1 mark] set up a summation that counts the number of times the basic operation is executed in Part B for an input array of size *n* and solve your summation.

???

3. Consider only lines 10 to 11 of the algorithm from question 1. Call these 2 lines "Part C".

10. for i ← 0 to n−1 do

11. S[Count[i]] ← A[i]

1. [1 mark] what does Part C of the algorithm do? Use your final value of Count[] from question 2a as input to Part C. Show the contents of S [] after each assignment on line 11.

Fill in:

S= [\_\_, \_\_, \_\_, \_\_,\_\_, \_\_], i = 0

S = [\_\_, \_\_, \_\_, \_\_,\_\_, \_\_], i = 1

S = [\_\_, \_\_, \_\_, \_\_,\_\_, \_\_], i = 2

S = [\_\_, \_\_, \_\_, \_\_,\_\_, \_\_], i = 3

S = [\_\_, \_\_, \_\_, \_\_,\_\_, \_\_], i = 4

S = [\_\_, \_\_, \_\_, \_\_,\_\_, \_\_], i = 5

???

1. [1 mark] what is the basic operation in Part C, and on what line does it occur?

???

1. [1 mark] set up a summation that counts the number of times the basic operation is executed in Part C for an input array of size *n*, and solve your summation.

???

4. [1 mark] Consider the entire CCS algorithm (including all lines in Parts A, B, and C). What is the basic operation for the entire algorithm? How many times is this basic operations executed for an input array of size *n*?

The basic operation for the entire algorithm is incrementing operation of the Count or S arrays on the following lines:

Line 3: Count[i] <- 0

Line 7: Count[j] <- Count[j] + 1

Line 9: Count[i] <- Count[i] + 1

Line 11: S[Count[i]] <- A[i]

The number of times the basic operations is executed ????

5. Consider an algorithm to insert an integer*K*into a *sorted* array of integers. We will make these assumptions about the algorithm:

- We are working with primitive array types – not automatically resizable classes like ArrayList or Vector

- The array has space allocated for *max* items, where *max* >> n

A prototype for the algorithm might be: Algo: Insert (A [0..n-1], *K* ) returns S[0..n]

1. [3 mark] Write the pseudocode for this algorithm, using the same style of pseudocode shown in your textbook. Do not use any unstructured programming constructs in your solution (ie: no goto, break, or continue statements).

??? is this ok?

Algo: Insert (A [0..n-1], K)

// Step 1: Start from the last element and move left until the correct position for K is found

i ← n - 1 // Start from the last element in the array

while i ≥ 0 and A[i] > K // Find the correct position by comparing elements

A[i+1] ← A[i] // Shift the element to the right

i ← i - 1 // Move to the previous element

// Step 2: Insert K in the correct position

A[i+1] ← K // Place K at the correct position

return A[0..n] // Return the updated array

b. [1 mark] what is the basic operation in your algorithm?

A[i+1] ← A[i]

c. [1 mark] Set up a summation that counts the number of times the basic operation is executed for an array containing n items, and solve it.