Comp 3761 Lab Assignment 1 Counting Basic Operations

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
```

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

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

Line 2 has a for 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 3 will initialize the new array called Count to be set to 0 for all indices i that starts at 0 and ends at n – 1.

b. [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 which is an assignment

c. [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
```

$$f(x) = \sum_{i=0}^{n-1} 1 = (\text{n-1}) - 0 + 1 = \text{n}$$

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
```

a. [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.

Line 4 of the algorithm starts a for loop where it starts at 0 and ends at n-2.
Line 5 is the inner loop which starts at i+1 and ends at n-1 which iterates through the rest of the elements after i.
Line 6 compares the values A[i] and A[j] if A[j] is bigger than A[i] then it will move to line 7 to increment Count[j] but if it is false then it will go to line 8 containing the else and then it will move to line 9 to increment Count[i].

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_, _1_, _1_,_1_, _0_], i = 1
Count = [_5_, _1_, _2_, _2_,_2_, _0_], i = 2
Count = [_5_, _1_, _2_, _3_,_3_, _0_], i = 3
Count = [_5_, _1_, _2_, _3_,_4_, _0_], i = 4
Count = [_5_, _1_, _2_, _3_,_4_, _0_], i = 5

b. [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 6 if A[i] < A[j] which is a comparison

c. [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.

```
for i □ 0 to n-2 do
   for j ← i+1 to n−1 do
```

$$f(x) = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2}((n-1)-(i+1)+1) = \sum_{i=0}^{n-2}(n-1-i) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$\sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i = ((n-2)-0+1)n - ((n-2)-0+1) - (n-2)((n-2)+1)/2$$

$(n^2)/2 - n/2$

n^2 is bigger so the answer is n^2 which is BigO

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]
```

a. [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.

Line 10 of the algorithm sets up a for loop that starts at 0 and ends at n-1.
Line 11 then starts sorting the numbers in the array called S from the smallest to largest number.

Fill in:

S= [__, __, __, __,__, _42_], i = 0
S = [__, _17_, __, __,__, _42_], i = 1
S = [__, _17_, _18_, __,__, _42_], i = 2
S = [__, _17_, _18_, _23_,__, _42_], i = 3
S = [__, _17_, _18_, _23_,_37_, _42_], i = 4
S = [_9_, _17_, _18_, _23_,_37_, _42_], i = 5

** From previous
A= [42, 17, 18, 23, 37, 9]
Count = [_5_, _1_, _2_, _3_,_4_, _0_], i = 5

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

The basic operation for part C is:
Line 11 S[Count[i]] ← A[i] which is an assignment

c. [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.

$$f(x) = \sum_{i=0}^{n-1} 1 = (n\text{-}1) - 0 + 1 = n$$

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 line 6 if A[i] < A[j] which is a comparison

The basic operation is executed n^2 because it is the biggest. Considering all three parts of the algorithm it appears that the part B of the algorithm takes maximum operations / longest (n^2), therefore the runtime on n in part a and c can be ignored and the efficiency ends up being n^2.

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]

   a. [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).

   A = [_9_, _17_, _18_, _23_,_37_, _42_]

```
1. Insert (A [0..n-1], K )
2.     insertIndex ← 0
3.         for i ← 0 to n−1 do
4.             S[i] ← A[i]
5.             if A[i] > K AND A[i-1] <= K
6.                 insertIndex ← i
7.     insertionVal ← K
8.     for i ← insertIndex to n do
9.         temp ← S[insertIndex]
10.        S[insertIndex] ← insertionVal
11.        insertionVal ← temp
12.    return S
```

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

   The basic operation is
   Line 4 S[i] ← A[i] which is an assignment

   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.

$$f(x) = \sum_{i=0}^{n-1} 1 + \sum_{i=insertIndex}^{n} 1 = (n-1-0+1) + (n - insertIndex + 1) = n + n - insertIndex + 1$$

$$= n$$

InsertIndex is always going to be the maximum of n therefore it can be safely ignored. The efficiency ends up being bigO of n.