Tom Fenyak

CPSC 3273 Homework Assignment 05

BubbleSort works by repeatedly swapping adjacent elements that are out of order. Below is its pseudocode.

**BubbleSort**(A)

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]

Part I (45 points) Correctness

a. Let A’ denote the output of **BubbleSort** (A). To prove that **BubbleSort** is correct, we need to prove that it terminates and that

A’[1] ≤ A’[2] … A’[n] , **(2.3)**

where n = A.length. In order to show that **BubbleSort** actually sorts, what else do we need to prove?

* We need to prove that the final array A consists of the same elements as the original array A in a sorted manner.

Answer the next two questions b. and c. to prove inequality (2.3).

b. (20 points) State precisely a **loop invariant** for the for loop in **lines 2–4**, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter:

(5 points) **Loop invariant** is:

* The loop invariant for the loop on lines 2 through 4 is that the smallest element of A[i…n] is at most ‘j’. We could also express this as A[j] = min {A[k] : j <= k <= n} and that the subarray A[j…n] are the same values in array A when we started the loop.

(3 points) **Initialization**:

* When we start j = n and the subarray will only hold one element and this obviously holds the loop invariant.

(8 points) **Maintenance**:

* We need to check the iteration for some value of ‘j’. A[j] has to be the smallest value in the subarray. The 3rd and 4th lines will swap A[j] and A[j – 1] if A[j] is the lesser value. This makes A[j – 1] at the lowest element in A[j -1…n]. This is the only change that is possible with this loop. This checks out the fact that the elements in the subarray are the same. We then decrement j to start the next iteration.

(4 points) **Termination**:

* The loop will terminate when j reaches i. The loop invariant holds true because the values that are in A[i…n] are from the original array A.

c. Using the termination condition of the loop invariant proved in part (b), state a **loop invariant** for the for loop in **lines 1–4** that will allow you to prove inequality (2.3). Your proof should use the structure of the loop invariant proof presented in this chapter:

(6 points) **Loop invariant** is:

* The loop invariant for lines 1 through 4 is that at the start of each iteration the subarray A[1…i – 1] has i – 1 smallest values that were in the original array A, but are now in a sorted manner.

(4 points) **Initialization**:

* Before we run the first iteration of this loop i = 1 and the subarray of A[1…i – 1] contains no elements. So the loop invariant will hold true.

(10 points) **Maintenance**:

* For the maintenance portion we need to consider the loop when i has a certain value. If we go off of our loop invariant where A[1…i – 1] has the i smallest values in the array in a sorted manner. We also have to look at what we proved for loops 2 – 4. When that loop is complete we know that A[i] is the smallest value in A[i…n] and A[1…i] is now the i smallest values from A[i…n] but sorted. Also the subarray A[i + 1…n] has the n – i values that are from the original array.

(5 points) **Termination**:

* The for loop ends when i = n which also makes the case i – 1 = n – 1 which means that the subarray A[1…i – 1] has the n-1 smallest elements from the original array but sorted. the largest value will be at the end of the array, which means that the entire array has been sorted.

**Part II (45 points) Running Time of BubbleSort**

1. (2 points) What is the input size?

The input size of BubbleSort is n or the size of the array.

1. (3 points) What is the operation that you will count?

For BubbleSort we should be counting the number of comparisons that are executed in the for loop on lines 2-4.

1. (40 points) Let T(n) be the running time of BubbleSort. Derive the asymptotic bound for T(n).

Since we are looking at the number of comparisons that are performed in lines 2-4. The loop will execute n – i times.

The total number of iterations is the sum of n – i from 1 to n – 1.

T(n) = n(n – 1) – n(n-1)/2

= n(n-1)/2

= (n^2/2) – (n/2)

Therefore we can say that BubbleSort belongs to (theta) (n^2) for best and average cases. I would say that BubbleSort has a T(n) of n for best case scenario when the array is already sorted.

Part III (10 points) Space Complexity

1. (2 points) Is BubbleSort an in place algorithm?

Yes BubbleSort is an in-place algorithm because it just goes through and swaps adjacent elements when they are not in the correct order. No other array needs to be created for BubbleSort.

(8 points) What is the space complexity of BubbleSort

BubbleSort operates in place and only needs one temporary variable for the swap so the space complexity will be O(1).