Analysis of Algorithms

Sam Stoodent

CPSC 221

*Defend all answers based on specific references to the code. You are also encouraged to add statement-counting code to given methods to collect data to test and defend your answers.*

# Algorithm: doSomething()

# Minimum Statements. How many statements would be executed in a call to doSomething() when the array size is zero (n == 0) or one (n == 1)?

If the array size was 0, the for loop would initialize int i and check if it is less than the array length, which would be false and that would be the end of the method. So, no matter what, at least 2 statements get executed.

T(0) = 2

If there is 1 element in the array, the for loop condition would be true one time. Two variables, left and right, would be initialized and compared to see if left is less than right, which would be false. After than, the for loop’s i would get incremented and the loop condition checked, again.

T(1) = 2 + 5 = 7

I wrote a driver class and added some statement counting code to doSomething() and got the expected number of statements with an empty array and a single-element array.

# Best Case Scenario. Under what conditions would the minimum number of statements be executed for an array where n is large? What is the growth function under these conditions?

The for loop is going to iterate once for every index in the array, although it doesn’t look like that index is ever used. So whatever happens in the loop is going to be multiplied by the size of the array, which is n for this algorithm.

Variables left and right always get initialized and compared, so that’s 3 statements that always happen in the loop. The while loop condition and the left++ and right-- statements inside make it look like the loop is always walking from the ends toward the middle of the array. I think that means the inner loop is going to happen n/2 times, every time through the outer loop.

Inside the while loop, there is a condition statement that will always get checked, left and right both get updated, and the while loop’s condition is going to get checked, so that’s 4 statements that happen for every while loop iteration. The part that changes is whether or not the value at array[left] and array[right] are out of order. If so, three statements get executed to swap those values. If everything in the array is already in order, those three statements would never get executed, so that seems like the best case scenario for this algorithm: everything is already in order.

Even so:

minimum T(n) = 2 + n(5 + (n/2)4) = 2n^2 + 5n + 2

I tried this with my test class and it seems to be accurate.

# Worst Case Scenario. Under what conditions would the maximum number of statements be executed for an array where n is large? What is the growth function under these conditions?

If the best case scenario was everything already being in order so (array[left] > array[right]) is always false, the worst case is going to make it always true. If the array is in reverse order, I expect that would be the case.

maximum T(n) = 2 + n(5 + (n/2)\*(4+3)) = (7/2)n^2 + 5n + 2

Wow. I tried this with my test class and didn’t get these results at all. The actual numbers were much lower than I predicted. In fact, they weren’t much worse than the best case numbers. It looks like the if (array[left] > array[right]) condition for each right and left index can only ever be true once. So the first time through the for loop, the condition is true n/2 times, but then every subsequent time through the for loop is totally wasted effort! Nothing every changes after the first time!

So, reverse order is still the worst case, but the growth function should really be:

maximum T(n) = 2 + n(5 + (n/2)4) + **(n/2)3** = 2n^2 + (13/2)n + 2

This function matches much better with my test results.

# Expected Average Case Scenario. Assuming a random array of unique elements, what is the expected average number of statements (the expected growth function) for a call to doSomething()?

The only difference between the worst case scenario above is that the if condition would be true only about half the time on that first time through the outer loop.

average T(n) = 2 + n(5 + (n/2)4) + (n/2)\*3/2 = 2n^2 + (23/4)n + 2

Testing with random arrays doesn’t perfectly match because of randomness, but it is very close and I am confident in this analysis.

# What is the runtime order (big-O) of doSomething()?

The largest factor in the growth function is 2n^2. The order, then, is O(n^2).