**CS 221 Analysis of Algorithms Homework**

Diego Dominguez

*All growth functions must be in simplified t(n) = \_\_\_\_ format with only* ***one*** *constant factor,* ***one*** *n factor, etc. Runtime order must be presented in proper big-O notation. All writing is required to be proofread for professional-quality grammar, spelling, capitalization, punctuation, complete sentences, etc.*

*Empirical results to compare with your predicted results come from the pre-compiled AoATester class given with the assignment. Run AoATester directly from the command line. AoATester configures an array of integers appropriate for the specified method and use case and reports the actual number of executed statements. The first command line argument specifies the method to test. The second argument specifies the use case. The optional third argument specifies the length of the array, which must be a positive integer. For the minimum statements use case, the third argument is ignored, even if a value is given. For other use cases, the length defaults to 100 unless specified otherwise.  
AoATester usage:*

$ java AoATester <find|replaceAll|sortIt> <min|best|worst|expected> [array length]

**Algorithm: find()**

**Minimum Statements, Constant Factor**

What statements are executed in a call to find() before reaching a return statement when the array size is zero (n == 0)? (Do not count the initialization of method arguments or return statements.) What is t(0) for find(), the minimum cost and the constant factor?

When the array size is 0 the number of statements that still need to get executed is 2, this is because int I is initialized then it checks if I is less than the array size, therefore the loop would not start.

Predicted t(0) = 2

**Run: AoATester find min**

What is your prediction for t(0)? How many statements does the test report? How do the results compare to your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results?

Predicted t(0) Statements: 2

AoATester find min Statements: 2

The test reported back 2, which is what I expected given the variable I needing to be initialized then the conditional check based off the array size. Overall I believe the t(0)=2 is correct.

Final t(0) = 2

**Best Case Scenario**

Assuming a large array size n and the target element is located at index 0, what statements are executed before the index is returned? What is the best case growth function t(n) under these conditions?

Predicted tbest(n) = 3

**Run: AoATester find best 100**

What is your predicted number of statements when n == 100? How does the number of reported statements align with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What do you need to modify about your analysis to better align with the empirical results?

Predicted tbest(100) Statements: 3

AoATester find best 100 Statements: 3

The number reported by the test is the same as my prediction, this is because int I = 0 is 1 statement, I < array.length is the second statements and lastly the if conditional is the third statement, then the index is returned since the value we are looking for is at index 0.

Final tbest(n) = 3

**Worst Case Scenario**

Assuming a large array size n, what would be necessary such that the method returns -1? How many times does the loop iterate? What statements are executed in each loop iteration? What is the worst case growth function t(n) under these conditions?

Predicted tworst(n) = 1 + 3(n)

**Run: AoATester find worst 100**

What is your predicted number of statements when n == 100? How does the number of reported statements for the actual worst case compare to your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results?

Predicted tworst(100) Statements: 301

AoATester find worst 100 Statements: 302

I am off by 1 number but I can’t seem to figure this out, since my constant is 3 and I thought we aren’t suppose to count a return as a statement, unless we are for this scenario. If that’s the case the growth function would be 2 + 3(n).

Final tworst(n) = 302

**Expected Average Case Scenario**

Assuming a randomly ordered array of unique elements and the target element is in the array, where would a target element be located **on average**? What is the expected average number of loop iterations if this is the case? What statements are executed in each complete loop iteration? Are there any loop statements that will **not** be executed when the target is found? What is the expected average growth function t(n) under these conditions?

Predicted texp(n) = 2 + 3(n/2)

**Run: AoATester find expected 100**

What is your predicted number of statements when n == 100? How does the average number of statements to find all elements align with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results?

Predicted texp(100) Statements: 152

AoATester find expected 100 Statements: 151.5

The average number given by the test does not exactly match my prediction, I am not sure why, my thought process was to take the number of statements that will run regardless of the array size plus the constant times n/2. The two comes from assuming the the value will match the value in the array or not, making it a fifty fifty chance.

Final texp(n) = 151.5

**Order**

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

O(n)

**Algorithm: replaceAll()**

**Minimum Statements, Constant Factor**

What statements are executed in a call to replaceAll() when the array size is zero (n == 0)? Do not overlook statements executed in find() or the assignment of its return value. So what is t(0) for replaceAll(), the minimum cost and constant factor?

Predicted t(0) = 4

**Run: AoATester replaceAll min**

What is your predicted number of statements when n == 0? How do the test results compare to your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results?

Predicted t(0) Statements: 4

AoATester replaceAll min Statements: 4

My prediction is the same as the result given by the test, this is because the number of statements ran in find is 2 if the array size is 0, then the other 2 in replaceAll are the initialization of the variable index, then the conditional in the while loop to check if index is greater than -1.

Final t(0) = 4

**Best Case Scenario**

Assuming a large array size n, what would cause the replaceAll() while loop to never iterate? What would be the cost of the first find() call? What statements are executed in replaceAll(), itself? What is the total best case growth function t(n) under these conditions?

The while loop would never iterate if the oldValue sent to find() is never found in the array. The cost of the first find call would be 2 + 2(n), the number of statements executed varies based off the variable index, the constant is 2 but if the while loop runs it is 2 + 2(n)

Predicted tbest(n) = (2+ 3(n / 2)) + (2 + 3(n / 2)) OR 4 + 3(n)

**Run: AoATester replaceAll best 100**

What is your predicted number of statements when n == 100? How does the number of reported statements compare with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What do you need to modify about your analysis to better align with the empirical results?

Predicted tbest(100) Statements: 304

AoATester replaceAll best 100 Statements: 304

My result was the same as the test, this is because based off my growth function I was able to figure out we have to take into account the growth function of find() and add it to the growth function of replaceAll, because this is the best case scenario I divided n by 2 in both growth function because its a fifty fifty that the index value will be the same as the value provided and the fifty fifty that the index variable in replaceAll will be greater than -1, and the constant value of replaceAll is 2 due to the initialization of index and the conditional of the while loop, then 3(n) is the 3 statements ran inside the while loop (setting a new value to array[index], calling find() again, and lastly checking the conditional in the while loop again.

Final tbest(n) = 304

**Worst Case Scenario**

Assuming n is large, all values in the array equal oldValue, and newValue does not equal oldValue, how many times will the while loop iterate? What is the cost of the first call to find()? What is the cost of the last call to find()? What is the average cost of a find() call within the while loop? What other statements are executed in every iteration of the while loop? What is the total worst case growth function t(n) under these conditions?

Predicted tworst(n) = 2 + 3(n) + 3(n \* (2 + 3(n)))

**Run: AoATester replaceAll worst 100**

What is your predicted number of statements when n == 100? How does the number of reported statements for the actual worst case align with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results?

My prediction is no where close to the output of the test, my thought process was I had to multiply the growth function of the method find times n within the growth function of replaceAll, I did this because I thought the function would run n times because the while condition would never be true. Then I added the growth function of find one more time just for the instance of it getting called once outside the loop regardless of n. I am very lost on this scenario and don’t know what I would have to change if I am going to be honest.

Predicted tworst(100) Statements: 90904

AoATester replaceAll worst 100 Statements: 15754

Final tworst(n) = 15754

**Expected Case Scenario**

Assuming a large, randomly ordered array of ***unique*** elements and oldValue is a value in the array, how many replaceAll() while loop iterations will occur? What is the expected cost of the first call to find()? What is the expected cost of the second call to find()? What is the expected growth function t(n) for replaceAll() under these conditions?

Assuming its randomly ordered, the while loop should iterate about 50 times, the expected cost of the first call of find will be 2 + 3(n), the expected cost for the second call to find will be 3 + 3(n). The expected growth function of replaceAll would be.

Predicted texp(n) = (2+3(n/2)) + (3 + 3(100/2)) + (3+3(100/2))

**Run: AoATester replaceAll expected 100**

What is your predicted number of statements when n == 100? How does the number of reported statements compare with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results?

My prediction is almost correct compared to the test, I used the growth function of the first iteration of find based on a fifty fifty chance of the value being the same plus the second iteration of find with the same prediction plus the growth function of replaceAll.

Predicted texp(100) Statements: 458

AoATester replaceAll expected 100 Statements: 458.5

Final texp(n) = 458.5

**Order**

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

O(n^2)

**Algorithm: sortIt()**

**Minimum Statements, Constant Factor**

What statements are executed in a call to sortIt() when the array size is zero (n == 0) or one (n == 1)? So what is t(0) and t(1), the minimum cost and constant factor for sortIt()?

The statements executed regardless of n, is 2 and since next is never less than the array size the loop will never iterate.

Predicted t(0 or 1) = 2

**Run: AoATester sortIt min**

How does the number of reported statements compare with your expectations? If there is a discrepancy, go back to the code to figure out why that might be. What do you need to modify about your analysis to better align with the empirical results?

My prediction is correct, like I stated before the only two statements ran are the initialization of the next variable and the condition to run the loop, and since next is never less than the array size the loop won’t run.

Predicted t(0 or 1) Statements: 2

AoATester sortIt min Statements: 2

Final t(0 or 1) = 2

**Best Case Scenario**

Assume a large array size n and elements in the array are already in ascending sorted order. The sortIt() outer loop depends only on n, but the inner loop is sensitive to the ordering of elements in the array and the current index of the outer loop. How many times will the outer loop iterate? How many times will the inner loop iterate? What statements are executed in every iteration of the outer loop? What is the growth function under these conditions?

The outer loop will run based off of n, and the inner while loop will run based off of the current array[next] value being greater than the previous value in the array, ex: array[next -1].The inner loop will iterate n -1 times since it starts at index one and its ordered in ascending order. The initialization of variable value and index, I++, conditional of the inner loop, and array[index] = value, will be executed with every iteration, in total being 5.

5(n) + 3(n - 1)

Predicted tbest(n) = 5(n) + 3(n - 1)

**Run: AoATester sortIt best 100**

What is your predicted number of statements when n == 100? How does the number of reported statements compare with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results? *(Note that the inner loop condition could be legitimately counted as 1, 2, 3, or even 4 statements. AoATester compromises and counts the inner loop condition as 2 statements.)*

*Based off the compromise I changed my growth function to* 5(n) + 2(n - 1), my prediction came out to be 698, which is not exactly what the test gave. I am not sure if it has to do with the constant being different but I am not sure what I could change to make my results the same.

Predicted tbest(100) Statements: 698

AoATester sortIt best 100 Statements: 695

Final tbest(n) = 695

**Worst Case Scenario**

Assume a large array size n and elements in the array are arranged in descending order. The sortIt() outer loop depends only on n, but the inner loop is sensitive to the ordering of elements in the array and the current index of the outer loop. How many inner loop iterations would there be when next == 1? How many inner loop iterations would there be when next == array.length - 1? What is the average number of inner loop iterations per outer loop iteration under these conditions? What statements are executed for each iteration of the inner loop? What is the total worst case t(n) for sortIt() under these conditions?

The loop iteration would be (n -1) since it would end the iteration 1 less than the length of the array since it started at 1 and not 0. The average would be (n - 1) \* 3(n-1).

Predicted tworst(n) = 5(n) + ((n-1)\*(2(n-1)))

**Run: AoATester sortIt worst 100**

What is your predicted number of statements when n == 100? How does the number of reported statements compare with your expectation? If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results? *(Note that the inner loop condition could be legitimately counted as 1, 2, 3, or even 4 statements. AoATester compromises and counts the inner loop condition as 2 statements.)*

*My prediction didn’t exactly match the outcome of the test, I think I may be a little confused on how the while loop statement counts work, I can’t really visualized how a while statement loop works, but I do know it still needs to check the conditional, keep a counter, and initialize a counter for it behind the scenes which is why I am using 3 as a constant for the inner loop.*

Predicted tworst(100) Statements: 20102

AoATester sortIt worst 100 Statements: 20495

Final tworst(n) = 20495

**Expected Average Case Scenario**

Assume a large array size n and the array contains unique elements in random order. How does the expected average number of inner loop iterations per outer loop iteration compare to the worst case? Why? How many inner loop iterations are expected on average? What is the total expected t(n) growth function for sortIt() under these conditions?

It is cut in half because we have to assume that the current index value will either be less than or not, so its a fifty fifty chance. Inner loop iterations would be ((n-1)\*(2(n-1))/2), divided by 2 to account for the random order.

Predicted texp(n) =5(n) + ((n-1)\*(2(n-1))/2)

**Run: AoATester sortIt expected 100**

What is your predicted number of statements when n == 100? How does the number of reported statements for a random case align with your expectation? (You may want to run the test several times.) If there is a discrepancy, go back to the code to figure out why that might be. What (if anything) do you need to modify about your analysis to better align with the empirical results? *(Note that the inner loop condition could be legitimately counted as 1, 2, 3, or even 4 statements. AoATester compromises and counts the inner loop condition as 2 statements.)*

*I am surprised to even get sort of close to the test, I think I am just overall confused on how a inner loop can affect the growth function if that inner loop is based on a certain condition of an outer loop.*

Predicted texp(100) Statements: 10301

AoATester sortIt expected 100 Statements: 10595

Final texp(n) = 10595

**Order**

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

O(n^2)