CS 221 Analysis of Algorithms Homework

Brayden Bowen

*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?

Only two statements will be called. The first of which will be initializing i to 0 and then checking if i < array.length. Since the array size is 0 the loop ends there.

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

My prediction for t(0) was 2, which is also what the test reported.

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?

The first statement will be setting i to 0 and then checking if it’s less than the array size. After that it will run through the if statement and the code will end there because it reaches the index first try.

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

My prediction was right this time as well, which I’m pretty happy about. We will see if that keeps up though

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?

The index won’t exist in the array if the method returns -1. The loop will iterate 100 times, from 0-99. The variable i will be initialized to 0 and then after that each loop will check if i is less than the array’s size, run through the if statement, and then increment i. After it runs through the full for loop it will then return -1.

Predicted tworst(n) = 3n + 2

### 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: 302

AoATester find worst 100 Statements: 302

I managed to nail my prediction on this one too, which definitely was a little more complicated, but I think I’m doing well so far and I’m quite happy with it.

Final tworst(n) = 3n + 2

## 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?

I would imagine the average case would just be the average between the best and worst case scenarios. If that’s the case, then I would just take worst case and divide it by 2

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

It looks like I was somewhat right again. I wasn’t sure if you could have half of a statement so I rounded up, but it looks like you can

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

## 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?

If the array size is zero, then you will combine the 2 minimum statements from find with the statement to assign -1 to the index variable and then run through the condition of the while loop once

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 was also right for this one, which was a pleasant surprise because I wasn’t too confident

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?

In order for the replaceAll() while loop to never iterate, index would have to be -1. This would mean that oldValue doesn’t exist in the array. The cost of the first find call would be 2 statements. The statements executed in replaceAll() would be assigning index and then checking the parameter in the while loop.

Predicted tbest(n) = (3n + 2) + 2

### 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

I definitely don’t think my formula is the correct one, but I just knew what the correct number of statements was by walking through it in my head.

Final tbest(n) = ?

## 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?

This scenario will contain significantly more statements than the best case scenario. The first call to find won’t cost too much because the value we’re looking for is add index 0, which means the first call to find() would be 3 statements. The last call, however, will cost 302 statements because it’s the worst case scenario

Predicted tworst(n) = n2 + (n(n+2) / 2)

### 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?

Predicted tworst(100) Statements: 15100

AoATester replaceAll worst 100 Statements: 15754

It looks like I was off by a decent amount on this one, which I’m honestly ok with. This stuff is still very new to me and honestly doesn’t make the most sense yet.

Final tworst(n) = ?

## 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?

Predicted texp(n) = n2 / ?

### 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?

Predicted texp(100) Statements:

AoATester replaceAll expected 100 Statements: 458.5

I’m starting to get especially lost when trying to come up with formulas and equations for these growth functions and It’s quite obvious I need some more practice.

Final texp(n) = ?

## Order

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

O(n2)

# 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 first statement executed is setting next to 1. It then compares next to array.length, and since the array size is 0 or 1, the statement next < array.length is false and the code stops there.

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?

Predicted t(0 or 1) Statements: 2

AoATester sortIt min Statements: 2

I was actually able to get this one right because it was pretty simple and straightforward

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 iterate through n-1 times and the inner loop will loop through 0 times because the array is already sorted and doesn’t need sorting. The outer loop will cycle through each value in the array and update the index with this new value

Predicted tbest(n) = 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.)*

Predicted tbest(100) Statements:

AoATester sortIt best 100 Statements: 695

For some reason I really struggle with trying to figure these growth functions out, but I’m sure I will get better with practice. I also tried to use vscode’s debugger for this project, but for some reason it just wasn’t working and I think that definitely makes this a lot harder.

Final tbest(n) = ?

## 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?

…

Predicted tworst(n) =

### 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.)*

Predicted tworst(100) Statements:

AoATester sortIt worst 100 Statements: 20495

…

Final tworst(n) = ?

## 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?

…

Predicted texp(n) =

### 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.)*

Predicted texp(100) Statements:

AoATester sortIt expected 100 Statements: 10595

…

Final texp(n) = ?

## Order

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

O(n2 ?)

I honestly don’t know why, but none of this seemed to click at all with me and I felt as if the first find() function was easy enough for me to get through, but then it just went all downhill from there, so I apologize for the cruddy assignment, but this is about the best I can do at the moment.