CS 221 Analysis of Algorithms Homework

Rylee Hilde

*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]

*GO THROUGH EACH WRITTEN PORTION AND MAKE SURE YOU ARE GOING THROUGH YOUR THOUGHT PROCESS*

# 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?

If the array size is zero, the for loop initializes the int variable i to 0 and checks if the variable i is less than the array length. Based on this, the method will at least execute 2 statements.

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 tester reported 2 minimum statements, which is what I expected from my prediction. From that, I believe that my analysis for minimum statements was 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?

The for loop iterates through each value once in the array and it uses the index to compare the array value at that index to the value parameter given in the method. Since the for loop iteration amount is dependent on the array size, it will represent n for the algorithm.

Then it will execute an if statement that compares the array value at the index to value parameter provided in the method. Since the target element is located at index 0, the condition is true and it will return the index 0. That is 1 statement that happen when target index is 0 before the index is returned.

Predicted tbest(n) = 2 + n

### 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: 202

AoATester find best 100 Statements: 3

The number of reported statements did not align with my expectations. I actually had realized when analyzing the find() method, I included n because I saw a loop. Which I realize now is not necessarily true in a best case scenario. Knowing this now, I found that only 3 statements that get executed before index i is returned: the for loop initialization of index i, the condition to check if i is less than array length, and the if statement that compares the array value at the index to value parameter provided in the method.

Final tbest(n) = 2 + 1 = 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 for loop will iterate n times based on the array size n. The loop will check the if statement, loop condition, and will increment the index n times. That will be three statements executed with each loop iteration. Then the loop will end once the condition is not met, therefore returning -1.

Predicted tworst(n) = 2 + 3n

### 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 was pretty happy to see the number of reported statements match my number of predicted statements. From that, I believe that my analysis for worst case scenario was correct.

Final tworst(n) = 2 + 3n

## 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?

The target element would be located, on average, near the middle of the array. In this scenario, I would expect the number of loop iterations to be half of the array size. My initial thought is to multiply n by 1/2 to predict half of the loop iterations. That is what leads me to my prediction below.

Predicted texp(n) = 2 + 1/2(3n) = 2 + 3/2n = 2 + 1.5n

### 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

I am also happy to see that the number of reported statements are pretty close to my number of predicted statement. I am curious as to why it is not exact, but I do not think it is necessary to modify the growth statements as the results are so closely aligned.

Final texp(n) = 2 + 1.5n

## 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?

When the array size is zero, the index is initialized with the find() method call. Where the for loop goes through initializing the i variable and checking that i is less than the array length, which then evaluates to false and returns -1 for no value found. Once it returns this value it moves down to a while loop statement back in the replaceAll() method, which has the condition if the index is greater than -1. From this, I was able to gather that there is five statements that happen no matter what.

Predicted t(0) = 5

### 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: 5

AoATester replaceAll min Statements: 4

I was kind of shocked to see that there were four statements that were executed instead of five. I believe I made the mistake of counting a return statement as an executed statement.

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?

When I assume a large array size n, the value that would cause the while loop to never iterate is if the oldValue is never found. This means that it would have to iterate over the whole array with the find() call. The cost of the first find() call would be the worst scenario situation of the find call, which is 2 + 3n. Based on the total cost of the find() call combined with the rest of the statements in the replaceAll() method would result in the best case scenario below.

Predicted tbest(n) = 4 + (2 + 3n) = 6 + 3n

### 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: 306

AoATester replaceAll best 100 Statements: 304

I am happy to see that the number of reported statements aligns with my number of predicted statements. Although, I think that the number could be improved as I believe I counted the same statements twice when porting over my worst growth function from the find() method analysis. Which leaves me with the exact value returned from the tester.

Final tbest(n) = 4 + (3n) = 4 + 3n

## 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?

If all values in the array equal old value then the while loop will iterate n times. In this loop it will set the index to the new value, call the find() method, and then check that the index is still greater than one. In summary, the while loop iterating n times will execute 3 statements per iteration with the 3 statement additionally from the find() call.

Predicted tworst(n) = 4 + 3n(3n) = 4 + 9n2

### 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: 90,004

AoATester replaceAll worst 100 Statements: 15,754

I am shocked to see that my predicted number of statements is horribly off from the reported number of statements. I realized that the find() call each time in the while loop is only going the n sized array from the first value detected and then incrementing through each value is detected until the whole array is changed to the new value. I am going to guess that the find() call only gets fully executed about half the time. Which gives me a predicted number of statements of about 15,000. I am much happier with this number as it is much closer to the reported number of statements.

Final tworst(n) = 4 + 3n(1/2n) = 4 + 3/2n2 = 4 + 1.5n2

## 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?

If I was presented with a random unique array, I would expect the while loop to iterate a little less than half the time as it is so unique to find the oldValue in a random scenario. So I expect the find() call to also loop half the time.

Predicted texp(n) = 4 + 3/2n(1/2n) = 4 + 3/4n2 = 4 + .75n2

### 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: 7,504

AoATester replaceAll expected 100 Statements: 458.5

I am shocked to see that my number of predicted statements are so much higher than the number of reported statements. I believe I found out what was wrong as I was halving the amount of loop iterations for the while loop, but it should only iterate once because the array is a unique set of elements. Based on that I changed my growth function so that it counts the minimum statements as 4 and counts the find() call as 3/2n because it is expected that it will be found halfway. As well as adding the worst case scenario growth function for the find call to represent it not finding the value for the rest of the array in the last loop iteration. This growth function gets me way closer to the reported number of statements with a predicted number of 454.

Final texp(n) = 4 + (3/2n) + (3n) = 4 + 4.5n

## Order

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

O(n)

# 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 minimum cost and constant factor of sortIt() is 2. I say this because if array size is 0 or 1, then the only statements executed would be the initialization of the next variable and the conditional check if next is less than the array length, which is false. With this, it will not execute any statements past that.

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’m happy to see that the number of reported statements aligns with my predicted number of statement. With that, it must mean that that my minimum statements are correct.

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?

Assuming that my array size is n, then my outer loop will iterate n times. It will execute 3 statements before going into the inner loop. For the inner loop condition, since it is dependent on the the current value being less than the value before it, in a best case scenario would not execute at all. The best case scenario will loop over all of the elements in the array, but only executes 5 statements per iteration.

Predicted tbest(n) = 2 + 5n

### 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: 502

AoATester sortIt best 100 Statements: 695

I am surprised to see that the number of reported statements does not align with my predicted number of reported statements. One thing that I thought was a discrepancy in my analysis was that I was counting the compound condition as one statement rather than 2. Since the compound condition has an AND operator, it means that both conditions need to be checked in order to both be true. I changed my growth function to be 2 + 6n instead of 2+5n. Which gets me 602, and is way closer to the reported statements, than the previous growth function.

Final tbest(n) = 2 + 6n

## 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?

I assume that the array is going to be descending order, so the inner loop is going to trigger 1/2 the amount as it triggers an increasing amount of times for each outer loop iteration. Which means in addition to 2 + 6n in the best case scenario growth function I will multiply 2n to 6n to represent 4/2n where the 4 statements execute on average half the time.

Predicted tworst(n) = 2 + 12n2

### 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: 120,002

AoATester sortIt worst 100 Statements: 20,495

I am shocked to see that the reported number of statements do not align with my predicted number of statements. I had come to realize though that I should have been counting the outer loop as n and then multiplying it by the growth function of the inner loop. What my mistake is that I was counting the number of statements twice, which caused my number to be way greater than it should have been. I also made the mistake of when I’m taking the average inner loop iterations I was dividing the amount of statements in the inner loop by a half rather than n. This gives me the result of my new growth function, which is 2 + n(2 + (n/2)4) = 2 + 2n + 4n2. This gives a predicted number of statements of 20,202, which is way closer than what was previously predicted. I assume that since it is close, my growth function is correct.

Final tworst(n) = 2 + 2n + 4n2

## 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?

Assuming that the array is in a random order and is sized n, I think that the amount of inner loop iterations would differ from worst case as I would believe it to only happen half the time, so I halved the amount of statements executed in the inner loop.

Predicted texp(n) = 2 + n(2 + (n/2) 4/2) = 2 + 2n + 2n2 /2 = 2 + 2n + n2

### 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: 10,202

AoATester sortIt expected 100 Statements: 10,595

I am really happy with how close the reported number of statements is aligning with my predicted number of statements. Even though they are a couple hundred apart, I will assume that my algorithm is correct.

Final texp(n) = 2 + 2n + n2

## Order

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

O(n2)