CS 221 Analysis of Algorithms Homework

Gabriel Tinsley

*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?

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

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

AoATester returned 2 as the statement count as predicted, so I think my analysis for minimum statements 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?

The for loop is going to iterate once for every index in the array. So, whatever happens in the loop is going to be multiplied by the size of the array, which is n for this algorithm.

There are 2 statements that happen if the array length is zero as seen above. If there is an array length, n, then there is an if statement that checks if the integer in the array index is equal to the integer value entered. That will return true if the target element is at index 0.

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

### 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: 402

AoATester find best 100 Statements: 3

The problem with my prediction is that I found the worst-case scenario, when reading the first sentence of best-case scenario I found “Assuming a large array of size n and the target element is located at index 0”. That would mean the return statement “return i” would come before the for loop goes to the next line with i++. So, the three statements that happen are initializing i = 0, then checking if i is less than array length, which would return true, then checking if the integer in the index of the array is equal to the integer value entered which would return true. So, in total there are three statements done.

Final tbest(n) = n + 2

## 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 would return -1 when the array does not contain the integer value the user is looking for. For a large array of size n, that would be the int i would be initialized to zero, then checks if that is less than the array length which would return true. That is two statements that happen no matter what.

Then first value in the array at index 0 is checked if it is equal to the value being looked for. If statement is false, then the loop goes to the next value at index 1 with i++. After incrementing one value that i is then checked with the array length, if less than array length returns true until the i is greater than the array length at that point that would return false. That is three statements that happen for every loop iteration.

Predicted tworst(n) = 2 + n(3) = 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 am glad I fixed my error for the growth function from my original thought of worst case scenario when I thought I had the right answer to best case scenario part. I realized that the initialization of i only happens once and is part of what happens no matter what. So, there is no n\*4 only n\*3 for this growth function.

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?

The best case scenario and worst case scenario have added together and divided by 2 to take the average is 2n + 2. I believe this is correct because there is most likely a chance that the number you are looking for is at one of the index at the n place. That would account for half of the statements to be executed before the return statement.

Predicted texp(n) = 2n + 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: 202

AoATester find expected 100 Statements: 151.5

This actually makes sense, taking the worst case scenario and if the condition was true half the time the equation would be (3/2)n + 2, which is 152

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

The index is initialized then jumps into the find for loop, since the array length is zero, n == 0, then the for loop for find initializes i and checks if that is less than array length which would be false and return -1. So, the while loop is checked in replaceAll and is false. So, 4 statements happen no matter what.

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

AoATester returned 4 as the statement count as predicted, so I think my analysis for minimum statements is correct.

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 best case scenario where the replaceAll() while loop never iterates is if the oldValue is never found in the array. That would mean the first find() call would happen 3n + 2 times before returning -1 because that is the worst case scenario for find(). Since 4 statements happen no matter what, that would mean the equation is 3n+4 for replaceAll() best.

Predicted tbest(n) = 3n + 4

### 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 am glad to get the growth function correct, the fact there is 304 statements makes a lot of since because this would happen when find() has the worst case scenario.

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?

We know no matter what happens the replaceAll() has 4 statements. Then when the index value is checked through the find method there is 3 statements that happen multiplied by n. Also, every index in the array that is the oldValue is changed to the newValue, so 3 + 1 is multiplied by n. Lastly, index is then in another find() algorithm, that means another 4 statements happen which is multiplied by n then multiplied by n.

Predicted tworst(n) = 4 + n(3 + 1 + n(4)) = 4n^2 + 4n + 4

### 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: 40404

AoATester replaceAll worst 100 Statements: 15754

Wow, I am way off with my prediction. The while (index > -1) makes more sense now, this condition is checked every time through the loop, only returning false when -1 if returned from find() algorithm. Assuming there is the average for find() algorithm then there is n(3/2) plus 3 statements that happen no matter what multiplied by n. Lastly, the find statement in the while loop is average as well, so 3 statements happen no matter what multiplied by 3n/2. (3/2)(100)^2 + (15/2)(100) + 4 = 15754

Final tworst(n) = (3/2)n^2 + (15/2)n + 4

## 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?

The expected cost of the first call to find() is n(3/2). Assuming this is the average number of times for replaceAll() then the worst case scenario would be divided by 2. This would result in the equation (3/4)n^2 + (15/4)n + 4. The 4 stays the same because 4 statements happen no matter what.

Predicted texp(n) = (3/4)n^2 + (15/4)n + 4

### 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: 7879

AoATester replaceAll expected 100 Statements: 458.5

Wow, I expected way more again. This could be due to the fact I assumed in my prediction the average would be squared again. In this case, replaceAll() uses the first find() which is (3/2)n and the send find is part of three statements that always happen. So, 3(3/2)n + 4 is the equation. (9/2)n + 4 = 454 which is close to 458.5.

Final texp(n) = (9/2)n + 4

## Order

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

O(n^2)

The worst case has a big-O of n^2 while the average case has a big-O of n, so the big-O in this situation is 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 int next is initialized to 1, then next is checked if less than array length. If n == 0 or n == 1 than the second statement would return false. That means there are two statements that happen no matter what.

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

AoATester for sortIt minimum statements returned 2 as predicted, so I believe my analysis is correct for minimum statements.

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 is checking if next < array.length and array.length could be n long. Value is than initialized and set to the second element in the array at index 1. Index is initialized and set to value of next which begins at 1. The inner loop depends on index > 0 and value being less than the array index behind it. Because the values are in ascending order the while loop would return false and the array value at index 1 would stay the same. Next++ then happens. The given equation would look like,

Predicted tbest(n) = n(5) + 2 = 5n + 2

### 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 close to the right answer, I did not realize the inner while loop counts as 2 statements. That would change my equation to being within 100 of the answer. I still like my analysis a lot for my prediction.

Final tbest(n) = 6n + 2

## 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?

When next == 1 there would be 2 inner loop iterations because the array index 0 gets set to whatever index 1 was, and then index goes to 0, then the while loop is checked and returns false. When next == array.length – 1 there would be n loop iterations because we are going from the back of the array to the very front checking every number with the number next to it. The average number of inner loop iterations under these conditions would be n/2. The statements executed in each iteration of the inner loop are setting the array index equal to the index 1 less than it that is larger and going down one index so two statements in each iteration.

Predicted tworst(n) = 5n (n/2 + 2) + 2 = (5/2)n^2 + 10n + 2

### 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: 26002

AoATester sortIt worst 100 Statements: 20495

Wow, I am proud of my analysis even though I am wrong. One thing I would say is that my (5/2)n^2 is close but not quite right since 25000 is 5000 more than the empirical result. There might be a discrepancy because the average number of inner loop iterations I have is not correct. There would be 2n since the inner loop would have to change every value to ascending order. Adding 2n^2 straight into the best case we get 2n^2 + 5n + 2, that would give 20502 statements which is a lot closer.

Final tworst(n) = 2n^2 + 5n + 2

## 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?

The expected average of the inner loop iteration compares to the worst case because the worst case was 2n now the expected average is n/2. Some numbers would be in correct order while others would not. n/2 is the perfect middle ground for this. Since there are two n’s there has to be an n^2 in the equation. My expected equation would be,

Predicted texp(n) = n(n/2) + 5n + 2 = (1/2)n^2 + 5n + 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.)*

Predicted texp(100) Statements: 3002

AoATester sortIt expected 100 Statements: 10595

Wow, I am way off with my prediction. This has to do with my expected average for the inner loop. Perhaps the inner loop goes through n times on average since 2n was the worst case and 1 is the best case. The middle between 2n and 1 is n. So, when writing the equation I would get n^2 + 5n + 2 and that is 10502 which is a lot close to the empirical result.

Final texp(n) = n^2 + 5n + 2

## Order

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

O(n^2)