CS 221 Analysis of Algorithms Homework

Nathan Marquis

*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?

In the condition that the array has 0 elements there would be 2 statements run(not method/return). The first statement initializes int i=0, the second statement does a check to see if 0 < 0(array.length). This would amount to false and therefore not run the internal if statement. There is not worse or better case, this number of statements would be consistent whenever the array.length is 0.

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 AoATester gave 2 as the number of statements, which matches my prediction. My analysis seemingly holds true because there were no loop iterations.

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?

There are always the first 2 statements executed, so this stays consistent. Each loop iteration has the potential for 3 extra lines of code corresponding to 1) if-statement check, 2) iterating I, 3) checking the for loop conditional for another loop. This would theoretically be a 3n+2 except for the fact that when the if-statement is true the method returns, thereby ending it. The best case scenario would be the 2 initial statements and one TRUE if-statement. This would effectively make a best case number of statements 3, 2 initial + 1 if-statement check.

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 appears to be accurate as compared to the Tester. It seems that the if statement ran for a single time as expected, ending the method early using the included return statement.

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?

In this case we would again have the inescapable first 2 statements start the statement tally, following by what would happened when the if-statement is NEVER true. Every loop iteration in this scenario would have 3 statements, performing as many times as there are elements which we would correspond to the integer ‘n’. Therefore the equation for worst case would be 3n + 2.

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: 3(100)+2 → 302

AoATester find worst 100 Statements: 302

My analysis is consistent with the total number of statements AoATester output. Plugging in 100 for n into the equation 3n+2 → 3(100)+2 → 300+2 → 302.

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?

Still assuming that there are 2 statements that will happen in any case, we can observe a pattern. If the target element is guaranteed to be found within a non-empty array then when n=1 there is one additional step (if-statement).

When n=2 we have 2 scenarios, equal to n=1 OR 4 statements(if-statement and 1 loop of [i++, loop-check, if-statement]). The mean of these 2 scenarios leaves us with (1+4)/2 → 5/2 additional statements on average.

When n=3 we have 3 scenarios, the same as n=1 OR n=2 OR 7 statements(if-statement and 2 loops of [i++, loop-check, if-statement]). The mean of these 3 scenarios leaves us with (1+4+7)/3 → 12/3 → 4 → 8/2 additional statements on average.

This pattern continues with each additional statement averaging to 11/2 additional steps when n=4, and 14/2 additional steps when n=5. The numerator always equals 3n-1, with the denominator remaining 2.

Predicted texp(n) = when n>0

### 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:

AoATester find expected 100 Statements: 151.5

My analysis was consistent with the Tester. My average number of statements estimated equally to the test version from the program.

Final texp(n) =

## 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 first statement would be inside the find() method, where there would be 1 for-loop check. This would be false because it is an empty array, moving on to then return (not counting towards the statement total). +1

Next, int index would be assigned ‘ -1 ‘ as a statement. +1

Finally, the while-loop check would be the final statement executed. +1

Predicted t(0) = 3

### 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: 3

AoATester replaceAll min Statements: 4

I had mistakenly forgotten that the for-loop had a minimum of 2 statements! The first statement is assigning int I = 0 and the second one is the for-loop check statement. I would need to modify my analysis to account for this discrepancy. The find() function ALWAYS has at least 2 statements involved. I had even made an equation for it but did not cross reference, next time I need to pay more attention to detail.

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 (one in which the while-loop never iterates) would be one where there does not exist in the array the element being sought. This would cause the statement for the find() method, 3n+2 to be executed first. As many loops as there are n would need to execute with 3 statements per, just the same as above in the find() method. + 3n+2

Then, the int index would be assigned as ‘ -1 ‘. +1

Finally, the while-loop condition would be checked and return false thereby ending the program. +1

Predicted tbest(n) = 3n+2 + 1 + 1 = 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

My assessment here was accurate. My expectation is consistent with the Tester output. It seems like the best course of action is one where there is no use of the while-loop.

Final tbest(n) = 3n + 4

## 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) =

### 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:

AoATester replaceAll worst 100 Statements:

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) =

### 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:

Final texp(n) =

## Order

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

O()

# 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()?

Predicted t(0 or 1) =

### 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:

AoATester sortIt min Statements:

Final t(0 or 1) =

## 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?

Predicted tbest(n) =

### 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:

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:

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:

Final texp(n) =

## Order

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

O()