CS 221 Analysis of Algorithms Homework

Kyle Truschel

*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?

The for loop can be visualized as a while loop, meaning the variable int declaration of “i” is one statement (total: 1).

The while loop conditional check of “i < array.length” is one statement (total: 2).

However, since the array size is known and is zero, the while loop conditional check never passes (it is not less than or equal to “i”) it would never enter the loop. The next statement is the return statement, so the number of statements up to this point is 2.

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 returned: “find() minimum statements: 2”, so my prediction 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?

Similar to the previous question, if we visual the loop as a while loop, it first declares an int variable “i” (total: 1) which is one statement, and then a condition check while “i” is less than the array length which another statement (total: 2). Since the array length is large but still unknown, it will start out as a linear growth factor n.

Within the loop, the growth factor n will always have the if block, a conditional check, as one statement (total: 2 + n(1) ), and then increment one time, with i++, as a statement (total: 2 + n(2) ). Finally, the while loop’s conditional statement must be checked again (I < array.length), which counts as another statement outside of the growth factor, n (total: 3 + n (2) ).

However, to consider the best case scenario, this assumes the first element value at index 0 is the same value, meaning while loop’s conditional statement does not need to be checked again, and the return statement is reached. This subtracts 1 from: 3 + n(2) ).

This results in my prediction of: 2 + n(2) or 4 statements.

Predicted tbest(n) = 4

### 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: 4

AoATester find best 100 Statements: 3

I can see now for the best-case scenario, that int “i” gets declared as a statement (total: 1), the while loop is created (total: 1 + n(1) ), since I forgot that the while loop creation for an unknown value is explained as a dominant factor, n(1). However, this time the n value is known and finds the first element value on the first go around of the loop.

The conditional if block is checked as one statement, inside the while loop, and passes, since the element value is found on the first go around (total: 1 + n (2) ).

This means there was no incrementation of i++ within the loop and re-check of the while loop condition, the method reaches the return statement, stopping at a total of 3 statements.

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?

To return -1 would mean the inner loop conditional statement never passes. Given my correction on the previous best case scenario and the while loop visualization, we start with one statement which declares an int variable, “i” (total: 1) and then the first while loop conditional statement of ( i < array.length), but since the length is NOT known its statement is 1 + n(1), where the inside “n(1)” represents the re-checking of the while statement each go around (total: 1 + n(1) ).

Within the while loop, the if conditional statement is checked, and it is represented now as the second statement in the while loop/parentheses (total: 1 + n(2) ).

Each time this conditional is checked and “i” increments as i++ (total: ( 1 + n(3) ).

However, this conditional statement never passes under the worst case scenario, so the while loop finishes with 1 + n(3). Since this scenario involves the method returning -1, this leads to the last statement, which is outside the loop (total: 2 + n(3) ).

Therefore, I believe the worst case scenario would be 2 + n(3) or 5 statements.

Predicted tworst(n) = 5

### 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?

Plugging in n == 100 into: 2 + 100(3) results in 302 statements.

Predicted tworst(100) Statements: 302

AoATester find worst 100 Statements: 302

The AoATester returned: “find() worst case statements for n==100: 302”, so my prediction was correct.

Final tworst(n) = 5

## 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 of array[i] can be found according to our class lecture by taking the best-case scenario and worst-case scenario dominant factors and dividing by two.

The best case scenario function is: 3

The worst case scenario function is: 2 + n(3)

Combining is: (5 + n(3)) / 2

Predicted texp(n) = (5 + n(3)) / 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?

Since the n value is known, 100, it can be plugged into the two equations for best case, but recall the value is found on the first occurrence, so n is 1:

1 + 1(2) = 3 statements.

And the worst-case scenario, where all 100 occurrences are not found:

2 + 100(3) = 302 statements

Taking the average of the two combined: 305/2 = 152.5

Predicted texp(100) Statements: 152.5

AoATester find expected 100 Statements: 151.5

I was incorrect in my initial assessment because I found the combination tricky because in fact you add the worst case scenario which is 3 to the best case which is 2 + n(3).

However, the math works by taking the constant out of the worst case scenario since it was already accounted for. In reality it should be: (3 + n(3)) / 2.

This also works when applying the n value of 100.

Final texp(n) = (3 + n(3))/2 = 151.5

## Order

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

O(n), since the linear order is the dominant factor.

# 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 declaration of an int variable called index is one statement (total: 1), but since a method is being called it must be accounted for in addition to the declaration statement.

The find method using a zero length array still declares the int value of zero (total: 2), and the while conditional statement is created but never passes, since the value inside must be less than, not less than or equal (total: 3).

The return value -1, is included in the find() method.

Now looking at the actual replaceAll() method, the while statement is created but never entered since index is not greater than -1 (the find statement returned -1 but the conditional statement is greater, not greater than or equal) (total: 4).

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

The AoATester returned: “replaceAll() minimum statements: 4”, so my prediction was 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 is where find() returns -1, so the while loop in replaceAll() is never entered. The worse case scenario for replaceAll() would involve entering the while loop and entering the find() method again.

Taking the worst case from the find() method is: 2 + n(3). Also accounting for the int index declaration in replaceAll() becomes its own statement (total: 3 + n(3) ). The while loop conditional check is created, so that is one statement. (total: 4 + n(3) ). However, it is not a linear dominant factor since it just compares a known value of index.

To explain why using the worst case scenario of find() is actually the best case scenario for replaceAll() is because if we have an array that attempts to replace an oldValue of 6 with a newValue of 5, if 6 never existed and loops through the entire array once, despite still checking every element, it will return -1 and never have to continuing checking to replace values again; 6 was never found.

On the contrary, if an array existed such as [5, 6, 6, 7, 7, 8, 6], and 6 was the oldValue to be replaced with newValue of 5, six would need to be replaced three times.

Therefore, the total is: 4 + n(3) or 7 statements

Predicted tbest(n) = 7

### 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?

Plugging in n==100 into: 4 + 100(3) = 304 statements.

Predicted tbest(100) Statements: 304

AoATester replaceAll best 100 Statements: 304

The AoATester returned: “replaceAll() best case statements for n==100: 304”, so my prediction was correct.

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?

If we take the same logic applied for the best case scenario, the best case scenario from find() when applied to replaceAll() becomes the worst case scenario. The absolute worst case scenario would be if the find() method returns the same value, oldValue, in the value.

So in the previous example if oldValue was 6 to be replaced by newValue of 5, but the array was [6, 6, 6, 6, 6].

So taking the best case scenario from find() which is 3, then applying the declaration of the index is another statement: 4.

The while loop conditional statement is declared as another statement n(1), but recall this will always pass since -1 was not returned (every value in the array was found): 4 + n(1).

The array[index] = newValue is one statement: 5 + n(1)

The index = find() is declared as one statement (1), and then the find method utilizes another loop inside it to become: 1 + n(1)

Combining the values: (4 + n(1 + n(1))

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()