CS 221 Analysis of Algorithms Homework

[your name]

*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.*

# Algorithm: find()

## Constant Factor

### Initial Analysis

#### What statements are executed in a call to find() before reaching a return statement when the array size is zero (n == 0)? (Exclude initialization of method parameters or return statements.)

#### The number of statements when n == 0, the minimum possible number of statements, is the constant factor for the find() growth function. What is t(0) for find()?

Predicted t(0) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual constant factor: t(0) = 2 How does your prediction compare to the actual constant factor?

#### 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 actual constant factor?

## Best Case Scenario

### Initial Analysis

#### The best case scenario for any algorithm describes the contents or organization of an input that results in the fewest possible number of statements being executed for a large input size n. For find(), the best case scenario is when the target element is located at index 0. How many loop iterations will occur if the target element is found at index 0?

#### What statements are executed before the index is returned? (Exclude initialization of method arguments or return statements.) Are there any statements in find() that will never be reached?

#### The best case growth function reflects the sum of all statements executed for any large input size n under the best case conditions. What is your predicted best case growth function t(n)?

Predicted tbest(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual best case growth function: tbest(n) = 3 How does your prediction compare to the actual best case growth function?

#### 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 actual best case growth function?

## Worst Case Scenario

### Initial Analysis

#### The worst case scenario for any algorithm describes the contents or organization of an input that results in the maximum possible number of statements being executed for a large input size n. For find(), the worst case scenario is an input where the method returns -1. Assuming a large array size n, what would be necessary such that the method returns -1?

#### How many times does the loop iterate in the worst case scenario for large array size n?

#### What statements are executed in each loop iteration?

#### The worst case growth function reflects all statements executed for any large input size n under the worst case conditions. Multiply the number of loop iterations by the number of statements per iteration and add the constant factor to get the complete growth function. What is your predicted worst case growth function t(n)?

Predicted tworst(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual worst case growth function: tworst(n) = 3n + 2 How does your prediction compare to the actual worst case growth function?

#### 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 actual worst case growth function?

## Expected Average Case Scenario

### Initial Analysis

#### 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 under these conditions?

#### What statements are executed in each complete loop iteration? Are there any loop statements that will **not** be executed when the target is found?

#### Multiply statements executed in each loop iteration by the expected number of loop iterations. Add statements that occur before the loop and account for any statements that aren’t executed when the method exits. What is the expected average growth function t(n) under these conditions?

Predicted tavg(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual average case growth function: tavg(n) = floor(3n/2 + 3/2) That is probably not exactly what you predicted, but how does your prediction compare to the actual function?

#### Some of the weirdness, here, has to do with code not conforming to continuous functions. You can’t have half a statement. A statement either gets executed or it doesn’t. We won’t worry about non-continuous functions going forward. If your prediction’s constant factor was only off by ½, up or down, you really got close enough. If not, what (if anything) do you need to modify about your analysis to better align with the empirical results?

## Order

#### Identify the largest factor from your growth functions and eliminate the coefficient, if any. What is the runtime order (big-O) of find()?

O( ? )

# Algorithm: replaceAll()

## Constant Factor

### Initial Analysis

#### What statements are executed in a call to replaceAll() when the array size is zero (n == 0)? Do not overlook statements executed in calls to find().

#### So what is t(0), the constant factor, for replaceAll()?

Predicted t(0) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual constant factor for replaceAll(): t(0) = 4 How does your prediction compare with the actual constant factor?

#### 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 actual constant factor?

## Best Case Scenario

### Initial Analysis

#### The best case scenario for replaceAll() avoids ever entering the while loop in replaceAll(). 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 find() case does this represent?

#### What other statements are executed in replaceAll(), itself, under these conditions?

#### Add the cost of all statements under best case conditions and combine like terms. What is the total simplified best case growth function t(n) for replaceAll() under these conditions?

Predicted tbest(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual best case growth function for replaceAll(): tbest(n) = 3n + 4 How does your prediction compare with the actual best case growth function?

#### 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 actual growth function?

## Worst Case Scenario

### Initial Analysis

#### The worst case scenario for replaceAll() would maximize the number of while loop iterations and calls to find(). 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 has to happen before the while loop can exit?

#### What is the cost of the first call to find() before reaching the while loop under these conditions?

#### What is the cost of the first call to find() within the while loop?

#### What is the cost of the last call to find() within the while loop?

#### 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?

#### Multiply the average cost of all statements within the while loop by the number of while loop iterations. Add this to all statements prior to the while loop and simplify the function, combining like terms. What is the total worst case growth function t(n) under these conditions?

Predicted tworst(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual worst case growth function: tworst(n) = (3/2)n^2 + (15/2)n + 4 How does your prediction compare to the actual worst case growth function for replaceAll()? Did you have the same factors, but coefficients were off? If coefficients were off, how close were they?

#### 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 actual worst case growth function?

## Expected Case Scenario

### Initial Analysis

#### 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(), before the while loop?

#### What is the cost of the last call to find(), within the while loop?

#### What is the expected growth function t(n) for replaceAll() under these conditions?

Predicted tavg(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual expected average case growth function: tavg(n) = (9/2)n + 7 How does your prediction compare to the actual expected average case growth function?

#### 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 actual growth function?

## Order

#### Identify the largest factor from your growth functions and eliminate the coefficient, if any. What is the runtime order (big-O) of replaceAll()?

O( ? )

# Algorithm: sortIt()

## Minimum Statements, Constant Factor

### Initial Analysis

#### What statements are executed in a call to sortIt() when the array size is zero (n == 0) or one (n == 1)?

#### What is the t(0 or 1), the constant factor for sortIt()?

Predicted t(0 or 1) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual constant factor for sortIt(): t(0 or 1) = 2 How does your prediction compare with the actual constant factor?

#### 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 actual constant factor?

## Best Case Scenario

### Initial Analysis

#### 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 under best case scenario conditions?

#### What statements are executed in every iteration of the outer loop? *(Note that the compound inner loop condition could be legitimately counted as 1, 2, 3, or even 4 statements. I am compromising and counting the inner loop condition as 2 statements.)*

#### Working inside-out, multiply the number of statements within each loop by the number of loop iterations under best case conditions. Add this to statements prior to the loop. Simplify the final function by combining like terms. What is the best case growth function for sortIt()?

Predicted tbest(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the best case growth function for sortIt(): tbest(n) = 7n - 5 How does your prediction compare to the actual best case growth function?

#### 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?

## Worst Case Scenario

### Initial Analysis

#### 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 (the first time through the inner loop)?

#### How many inner loop iterations would there be when next == array.length - 1 (the last time through the inner loop)?

#### 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? *(Note that the compound inner loop condition could be legitimately counted as 1, 2, 3, or even 4 statements. I am compromising and counting the inner loop condition as 2 statements.)*

#### Working from inside to outside, multiply the statements in each loop by the number of loop iterations and add to statements outside the loop. Simplify and combine like terms. What is the total worst case t(n) for sortIt() under these conditions?

Predicted tworst(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual worst case growth function for sortIt(): tworst(n) = 2n^2 + 5n - 5 How does your prediction compare with the actual worst case growth function?

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 actual worst case growth function?

## Expected Average Case Scenario

### Initial Analysis

#### 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?

#### Again, working from the inside out, multiply statements in each loop iteration by the expected average number of loop iterations and add to statements outside the loop. Simplify and combine like terms to get the total expected t(n) growth function for sortIt() under the expected average conditions. *(Note that the compound inner loop condition could be legitimately counted as 1, 2, 3, or even 4 statements. I am compromising and counting the inner loop condition as 2 statements.)*

Predicted tavg(n) =

### Compare and Resolve

#### Change the font of the next line to black to reveal the actual average growth function for sortIt(): tavg(n) = n^2 + 6n - 5 How does your prediction compare to the actual average growth function for sortIt()?

#### 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 actual expected average case growth function for sortIt()?

## Order

#### What is the runtime order (big-O) of sortIt() based on the above growth functions?

O( ? )