Analysis of Algorithms

*Defend all answers based on specific references to the code. Do not count return statements or initialization of method arguments. You are* ***strongly*** *encouraged to walk through algorithms in the debugger and to add statement-counting code to given methods to test and refine your analysis.*

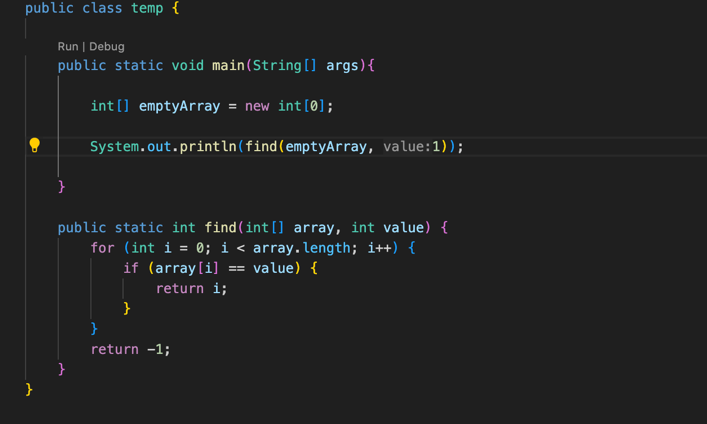
# Algorithm: find()

A screenshot of a computer

Description automatically generated

# Minimum Statements:

# How many statements would be executed in a call to find() when the array size is zero (n == 0)?

A black background with white text

Description automatically generated

If the array size was 0 then only 2 statements would be executed. First the for loop which will determine the size, and second the return statement returning -1.

# Best Case Scenario:

# Under what conditions would the minimum number of statements be executed for an array where n is large?

The minimum number of statements executed would be 3 statements. One, for the for loop, another for the if statements, and the last one for the return statement.

# Where would the target element be located in the array?

In the best case scenario the target element would be located in the first index of the array (array[0]).

# What is the growth function under these conditions?

The growth function for the find(); method in the best case scenario would be o(n) or logarithmic.

# Worst Case Scenario:

# Under what conditions would the maximum number of statements be executed for an array where n is large?

The maximum number of statements that would be executed for an array that is n large is: 2n + 1. Which means the loop will run n times(+n), and also the if statement will run n times (+n) and then finally when it cycles through the whole array it will run the return statement once (+1).

* Where would the target element be located?

In the worst case scenario the target element would be located in the very last index of the array, Or not in the array at all.

# What is the growth function under these conditions?

The growth function for the find(); method in the worst case scenario would still be o(n) or logarithmic.

# Expected Average Case Scenario:

# Assuming a random array of unique elements and the target element is in the array, where would a target element be located on average?

In an average case scenario the target element would be located in the middle of the array.

# What is the expected average number of statements (the expected growth function) for a call to find()?

The average expected number of statements for a call to find would be (2n + 1) / 2. I wrote this because instead of running through all the statements in the loop it would only run through half, which would be half the number of statements for the worst case scenario.

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

The big-O notation for find base on the above growth functions is O(n) or logarithmic.

# Algorithm: replaceAll()

A paper with writing on it

Description automatically generated

# Minimum Statements:

# How many statements would be executed in a call to replaceAll() when the array size is zero (n == 0)?

The amount of statements that would run if the size of the array was zero would be 3 statements. The first statement would be declaring the integer variable (.int index = find(array, oldValue);). The second statement would be the for loop inside the find(); function, and the last statement would be the return -1 from the find(); function.

# Best Case Scenario:

# Under what conditions would the minimum number of statements be executed for an array where n is large?

# How many occurrences of the oldValue element would be in the array?

# Where would it/they be located in the array?

# What is the growth function under these conditions?

# Worst Case Scenario:

# Assuming newValue and oldValue are not equal, under what conditions would the maximum number of statements be executed for an array where n is large?

# How many occurrences of oldValue are in the array?

# Where would it/they be located?

It would be located at either the end of the array or not in the array at all.

# What is the growth function under these conditions?

The growth function under the worst-case scenario would be 4n^3 +14n^2+14n+4.

# Expected Average Case Scenario:

# Assuming a random array of unique elements and oldValue is a value in the array, what is the average number of statements (the expected growth function) for a call to replaceAll()?

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

# Algorithm: sortIt()

# Minimum Statements:

# How many statements would be executed in a call to sortIt() when the array size is zero (n == 0) or one (n == 1)?

# Best Case Scenario:

# Under what conditions would the minimum number of statements be executed for an array where n is large?

# Would the algorithm execute a different number of statements if the elements in the array were already in sorted order? Reverse order? Random order? All the same value?

# What is the growth function under the best case conditions?

# Worst Case Scenario:

# Under what conditions would the maximum number of statements be executed for an array where n is large? (Already in some kind of sorted order? Duplicates?)

# What is the growth function under the worst case conditions?

# Expected Average Case Scenario:

# Assuming a random array of unique elements, what is the expected average number of statements (the expected growth function) for a call to sortIt()?

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