Manuel Melendez – 80542914

Week 7 Lab

Performance of sortOfSort method

The method sortOfSort receives an array as input and sorts it as specified in the lab instructions. The following is the number of instructions performed then the method is called (to make things simple, I did not copy the whole instruction, but I will refer the line number and the instruction performed):

line 5: if to check length of array = 1

line 6 return nothing = 1

lines 9,12,13,16, and19: initialize and declare counter,

startIndex, endIndex, max, and lastIndex = 5

line 21: for loop = n

initialize and declare i (only once) = 1

line 24 initialize and declare max with a call to findMax = (n(n+1))/2

line 26 if lastIndex = 1

line 28 call to swapIndex = 3

line 30 endIndex -- = 1

line 34 else

line 35 call to swapIndex = 3

line 36 startIndex++ = 1

line 40 counter++ = 1

line 44 if counter > 1 = 1

line 45 counter = 0 = 1

line 46 lastIndex = !lastIndex = 1

Helper methods:

findMax

line 54: if checking startIndex == endIndex = 1

line 55: return startIndex = 1

line 56: initialize and declare max = 1

line 58: for loop = (endIndex-startIndex times)

\*Inside for loop\*

line 58 if checking arr[i] > max[0] = 1

lines 59 and 60 assigning values to max[0] and max[1] = 2

line 63 return max[1] = 1

swapIndex

line 68 initialize and declare temp = 1

lines 69 and 70 assign valies to arr[a] and arr[b] = 2

The best cases for this method are when it receives at empty array or an array of length 1 as there is nothing to sort or what is in the array is technically already sorted, respectively. In both of these cases, the only instructions executed are the if and return from lines 5 and 6, meaning that there are only two instructions executed. Therefore, the time complexity in these two cases is O(1) as it is constant and will always perform only these two instructions in these two cases.

In all of the cases where the length is larger than two, the variables from the start should be declared and initialized (5 instructions), followed by the for loop. Inside this for loop, it will go through the whole array (n loops) finding the maximum value by calling the method findMax, this method has its own for loop, but this one can vary in terms of its iterations as it will be decreasing every time is called ((n\*(n+1))/2 loops). After this, there is the if and else, with very similar instructions that are performed, either one will perform a call to swap index, which has 3 instructions and then increasing or decreasing the respective variable. There is the increase in the counter, and the if statement to check for the counter, which has 2 instructions inside it.

To break it down from smaller pieces into the big picture, I will count the number of instructions of everything that is inside a loop first.

There is are the if and else from inside the loop, which both will perform the same number of instructions, which is 5 as swapIndex has 3 instructions plus the if and the increase or decrease at the end.

findMax has an if at the beginning, and max is declared and initialized (2 instructions so far), in the for loop, i is declared and initialized only once. Then it will check the if statement every time, as well as checking i <= endIndex and i++, with a total of 3 instructions (minimum, in case it never goes into the if statement) or 5 in the worst case (where it goes into the if every time) performed (n\*(n+1))/2 times = (3n(3n+1))/2 in case it never goes inside the if, or (5n(5n+1))/2 if it always goes inside the if. This is taking into consideration that it will be performed n times because of the loop in the main method.

The if(lastIndex) and its respective else will always be executed (either or), which each one has the same amount of instructions, being 5, as mentioned before.

The counter will always increase with every iteration in the loop from the main method and will always check the if statement. The amount of times it goes inside the if statement will vary by the length of the array but it will be (n/2)-1 times for odd length and n/2 for even length, with two instructions inside it.

The main for loop will be executed n times, with everything it has on the inside, being the declaration and initialization of max (n(3n+1))/2, the if/else (5), the increase in the counter, and checking the if statement(2) done n times. For a total of (n(3n+1))/2 + 7 instructions executed, where it never goes inside of any of the if statements. In the absolute worst-case scenario, the number of instructions performed is (n(5n+1))/2 + (n/2) + 7.

By simplifying our equations, we obtain: (3n^2 + n)/2 + 7 and (5n^2 + 2n)/2 + 7

In conclusion, in terms of Big O notation, the very best cases (when the length of the input array is less than 2) for this method have a complexity of O(1), and any other cases will always have a complexity of O(n^2) as there are no worst case scenarios, even considering if all the instructions inside every if are executed (this is pointed out as it is the only aspect that could cause any differences in the number of instructions performed.

In simple terms, the main reason why the overall performance of the method other than the best cases, is because of the nested for loops from the main method and the findMax method, yielding O(n^2) performance. And the reason for the best cases to have a complexity of O(1) is because in these two cases, only the same two instructions will always be executed.