In order to analyze the performance of the method sortOfSort, I have compiled the number of times that each statement in the program is executed. This compilation was made with the notion that an array of length n was passed to the method. The leftmost column assigns a number to each statement in order to facilitate discussion. The middle column contains the statement itself and the rightmost column contains the number of times that specific statement was executed. Further discussion of this table will be found below.

|  |  |  |
| --- | --- | --- |
| # | Statement | Number of Times Executed |
|  | sortOfSort | |
| 1 | int maxIndex; | 1 |
| 2 | int endIndex = integerArray.length – 1; | 1 |
| 3 | int beginningIndex = 0; | 1 |
| 4 | int counter = 1; | 1 |
| 5 | int arrayEndLimit = 0; | 1 |
| 6 | for(int i = 0; i < integerArray; i++) | n |
| 7 | maxIndex = beginningIndex; | n - 1 |
| 8 | for(int i = beginningIndex; i < integerArray.length – arrayEndLimit; i++) | Σ(n(n+1))/2 |
| 9 | if(integerArray[i] > integerArray[maxIndex]) | Σ(n(n+1))/2 |
| 10 | maxIndex = I; | 0 to Σ(n(n+1))/2 |
| 11 | if(counter == 1) | n - 1 |
| 12 | swap(integerArray, maxIndex, endIndex); | 3(n – 1) |
| 13 | endIndex--; | n - 1 |
| 14 | counter++; | n - 1 |
| 15 | arrayEndLimit++; | n - 1 |
| 16 | else if(counter == 2) | n - 1 |
| 17 | swap(integerArray, maxIndex, endIndex); | 3(n – 1) |
| 18 | endIndex--; | n - 1 |
| 19 | counter = -1; | n - 1 |
| 20 | arrayEndLimit++; | n - 1 |
| 21 | else if(counter == -1) | n - 1 |
|  |  |  |
| 22 | swap(integerArray, maxIndex, beginningIndex); | 3(n – 1) |
|  |  |  |
| 23 | else | n - 1 |
| 24 | swap(integerArray, maxIndex, beginningIndex); | 3(n – 1) |
| 25 | beginningIndex++; | n - 1 |
| 26 | counter = 1; | n - 1 |
|  | swap | |
| 27 | int temp = integerArray[largest]; | 1 |
| 28 | integerArray[largest] = integerArray[toSwap]; | 1 |
| 29 | integerArray[toSwap] = temp; | 1 |

For lines 1-5, each statement is only executed once, which explains the value of 1 in the number of times executed. For line 6, this for-loop will execute n times since that is the length of the array. For line 7, this value is within the previous for-loop, so it will execute n – 1 times. For line 8, the number of times executed is a summation because in the first loop, it will iterate n times, and in the next loop it will iterate n – 1 times, and so on. Therefor, the total steps for this statement is the summation of n(n+1)/2. Line 9 will also iterate this same amount of times because it is within the for-loop and it only executes once each iteration, so 1 times the summation is equal to the same summation. Line 10 can either execute between 0 times to the previous summation amount of times. It will execute 0 times if the max index was at the beginning index, so the if-statement is always false. On the other hand, it could find the max each time it loops, so it will iterate Σ(n(n+1))/2 times. The lines 11, 16, 21, and 23 contain conditionals, which means that either they will all be executed, or just one will be executed. If just one is executed, then the number of times executed will be (n – 1) because it is within a for-loop. If they are all executed, then (n – 1) will be added four times. Lines 12, 13, 14 and 17, 18, 19, 20 are identical and each line will execute (n – 1) times because it is within the for-loop.

Analyze the performance of your method sortOfSort. You are expected to clearly explain / justify your answer to this question. Your answer should address the best / worst / and average cases for the performance of your method. You will describe in a word document called **YourLastName-YourFirstName-Week7.docx** the performance of your sortOfSort method in terms of n, the length of the input array.