SortOfSort Performance

Theoretical Analysis:

Using an array of length n, the algorithm runs the array through in the following time complexity:

|  |  |
| --- | --- |
| Once | Repeating |
| Int numPass = 0; | While loop (10 + 4n) |
| Int front = 0; |  |
| Int end = array.length - 1 |  |

3 + 10n + 4n^2

While loop

|  |  |
| --- | --- |
| Once | Repeating |
|  | Front != end |
|  | int index = findMaxIndex(); |
|  | If(numPass % 4 < 2) |
|  | swapValue() |
|  | placeOfChange ++ / -- ;  \**place of change may be end or front depending on the position that is to be swapped\** |
|  | numPass ++; |

1. + 4n + 3n + (4n)n + 3n

findMaxIndex()

|  |  |
| --- | --- |
| Once | Repeating |
| Int maxIndex = front; | i <= front; |
| Int i = front; | If(Nums[i] > nums[maxIndex]) |
| Return maxIndex; | maxIndex = i; |
|  | i++; |
|  |  |

3 + 4n

swapValues()

|  |  |
| --- | --- |
| Once | Repeating |
| Int temp = array[placeOfChange]; |  |
| Array[placeOfChange] = array[index]; |  |
| Array[index] = temp; |  |
|  |  |
|  |  |

3

T(n) = 3 + 10n + 4n^2

O(n) = O(n^2)

Best case:

In this scenario, the best case would be when the array is of an empty length or if it is that of one value. Since the algorithm would only go through the code once, the time complexity would remain O(1), considering using only one value would keep the output the same, and having no value would result in no output after being ran once.

Worst / Average cases:

With this code, the average and worst cases of time complexity would be the same since the code still runs at O(n^2) for all lengths of the array. There is really no value of length that will change this outcome since the algorithm is really straight forward and simply traverses through the while loop and for loop of the findMaxIndex to give you a result.