SortOfSort method Analysis

|  |  |
| --- | --- |
| **ONE** | **REPEATING** |
| int counter = 1; | pointer1 < last |
| int maxIndex; | maxIndex = maxIn(pointer1, pointer2, givenArray); |
| int pointer1= 0; | counter>=1 &&counter<=2 |
| int pointer2 = givenArray.length-1; | temp = givenArray[pointer2]; |
| int temp ; | givenArray[pointer2] = givenArray[maxIndex]; |
|  | givenArray[maxIndex] = temp; |
|  | pointer2--; |
|  | counter++; |
|  | counter>=3 &&counter<=4 |
|  | temp = givenArray[pointer1]; |
|  | givenArray[pointer1] = givenArray[maxIndex]; |
|  | givenArray[maxIndex] = temp; |
|  | pointer1++-; |
|  | counter++; |
|  | counter == 5 |
|  | counter = 1 |

maxIn method Analysis(helper)

|  |  |
| --- | --- |
| **ONE** | **REPEATING** |
| int max = 0; | i <= pointer2; |
| int maxIndex = 0; | i++; |
| int i = pointer1; | nums[i] > max |
| return maxIndex; | max = nums[i]; |
|  | maxIndex = i; |

**Time Complexity Analysis:**  We break down both methods into instructions that happen once and some that occur for an n amount of times. We conclude that only 9 instructions occur once within both methods. The while loop will run 16 instructions for n times(16n). The for loop in the maxIn method runs 5 instructions for n times(5n). Therefore, the time complexity for this algorithm becomes this equation, T(n) = 21n + 9 for both methods combined. The run time will be dependent on how large the array is.