Name: Raisa Nkweteyim

CS2 – SortOfSort Time Complexity Analysis

|  |  |
| --- | --- |
| Once | Repeating |
| int i |  |
| int index = 0 |  |
| int count = 0 |  |
| int begin = 0 |  |
| int end = 0 |  |
| int end = array.length – 1 |  |
| LOOP | |
| begin = end | begin != end |
| i = begin | int max = array[begin] |
|  | **LOOP** |
|  | i <= end |
|  | i++ |
|  | array[i] >= max |
|  | index = i |
|  | **END OF LOOP 2** |
|  | count < 2 |
|  | int temp = array[index] |
|  | array[index] = array[end] |
|  | array[end] = temp |
|  | count ++ |
|  | end = end – 1 |
|  | int temp = array[index] |
|  | array[index] = array[begin] |
|  | array[begin] = temp |
|  | begin = begin + 1 |
|  | count++ |
|  | count == 4 |
|  | count = 0 |
| SUM | |
| 8 | (Loop2 = 4 x n)( Loop = 19 x n) |
| Total Sum | |
| 8 + 76n^2 |  |

T(n) = 8 + 76N^2

O(n) = N^2

Where, N is the size of the array.

The best case, worst case and average case will always be the same because the sequence of steps will always pass through 2 loops:

1. To go through every element in the array until it is sort of sorted.
2. To find the biggest element in the array

This leads to a big O notation of N squared.