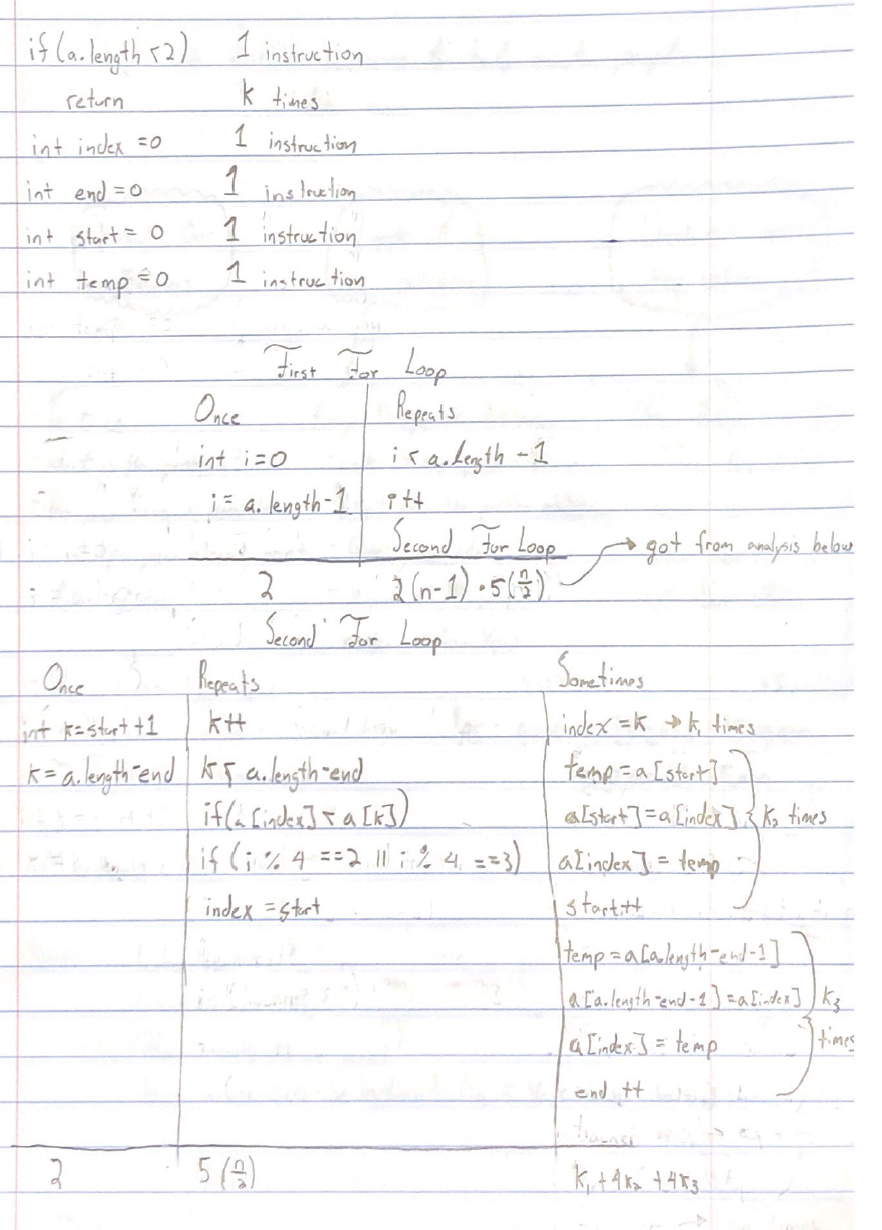
**Week 7 Lab Report**

T(n) and O(n):

The total time complexity equation would be:

Which consist of 5 instructions in the beginning and 4 statements that always occur for both for loops which is where the 9 comes from. The first k occurs for the return statement, k1 is for the statement that assigns the largest index, 4k2 is the three swap statements plus the increase in start, and the 4k3 is the three swap statements plus the increase in end. The reason these values are constants is because those statements are not guaranteed to execute and are depended on whether the if statements are true. The 5 times n divided by 2 comes from the second for loop which has five constant statements which occur on average of n divided by 2 times. Lastly, 2 times n-1 comes from the first for loop which has two constant statements and occurs the length of the array minus one times. Please note the analysis on the image below.



**Image 1:** Showing instruction analysis line by line

The big O notation is depended on the highest factor of the T(n) equation. The highest factor in the T(n) equation is the term which simplifies to . Having n2 as the largest factor, the big O notation for this algorithm is O(n2).

Best, worst, and average case:

In terms of the total time complexity the best case is when the length of the array is less than two. In this case the method only runs two instructions and then returns. However, the likelihood of the method getting an array of less than two is very low so there is a more practical best case. Besides the n2 that always happens, the algorithm’s number of instructions is driven by the constants k2 and k3 because when their if statement passes they each perform four instructions. Therefore, the higher the frequency of k2 and k3 the more instructions the algorithm goes through. Please note that although k1 might have a higher frequency of being called, k1 only results in one instruction. Understanding that minimizing k2 or k3 will result in fewer instructions, the practical best case is when the remainder of the length divided by four is equal to zero. The reason that is, is because k2 and k3 are both executed a normal number of times and then k3 is ran only one additional time meaning only four additional statements would be executed. With that said, the worst case would be when the remainder of the length divided by four is equal to three. This is because k2 and k3 are both executed twice for a normal number of elements. That means that there would be an additional sixteen instructions that would be executed. The average case would then be when the remainder of the length divided by four is equal to one or two. This means there would be between eight and twelve additional statements being executed which is the most probable.



In terms of big O notation, the best, worst, and average case are all the same. No matter what, the algorithm goes through the array n2 times. Even if the array is already sort of sorted, both for loops will still result in the algorithm performing n2 executions.

In Summary

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Case | Worst Case | Average Case |
| T(n) | Array.length < 2  or  Array.length % 4 == 0 | Array.length % 4 == 3 | Array.length % 4 ==3 or == 2 |
| O(n) | n2 | n2 | n2 |