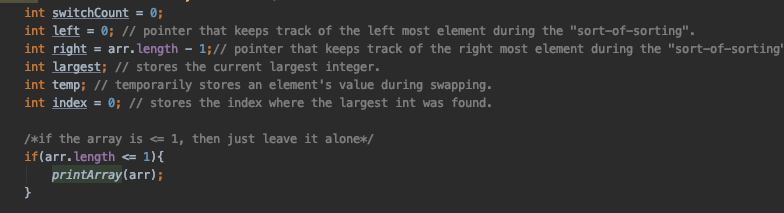
Analyze the performance of your method sortOfSort. You are expected to clearly explain / justify your answer to this question.

* I decided to analyze my code by first determining the constant instructions, and then calculating the repeating instructions to find out what T(N) is.
* Here, is a screencap of the constant code, which are initializers for values:

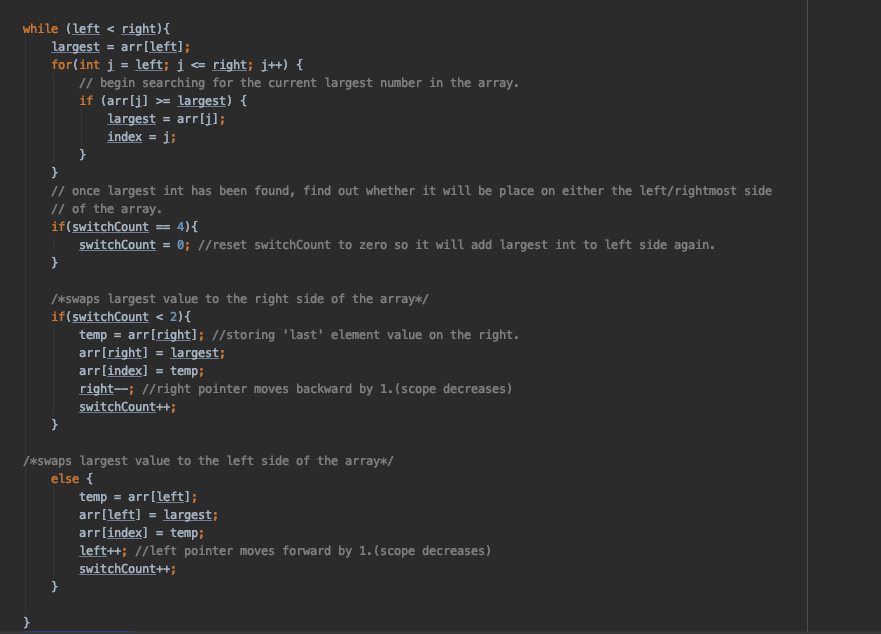


Putting these values into a table, I calculated how my code was performing so far:

|  |  |
| --- | --- |
| ONCE | VALUE |
| int switchCount = 0; | 1 |
| int left = 0; | 1 |
| int right = arr.length - 1; | 1 |
| int largest: | 1 |
| int temp; | 1 |
| int index = 0; | 1 |
| arr.length <= 1 | 1 |
| printArray(arr); | 1 + 1(at the end of the code) |
| Total steps: | 9 |

So at this point in my method, it is taking 9 steps so far which is a constant value.

* The next step was to analyze my while loop and the contents inside of it shown here:



I created a table meant for the repeating instructions

|  |  |  |  |
| --- | --- | --- | --- |
| CODE | INSTRUCTIONS | STEPS | STEPS |
| While loop | left < right  largest = arr[left]; | -Repeats n+1 times for both even and odd size arrays.  Unless if the size of the array  is <=1, then this only happens once.  -Repeats n/2 times. | N +1 or 1(constant)  +  N /2 |
| For loop | int j = left;  j <= right;  J++;  arr[j] >= largest;  largest = arr[j];  index = j; | -Repeats n/2 times if length > 1.  -repeats n times if length > 1.  -repeats depending on n(right) after decreasing.  -repeats n times, if array length is greater than 1.  -repeats n times because its iterating through the available scope  -only repeats based on the  amount of times arr[j] was  satisfied. | N/2  +  N  +  N  +  N  +  N  +  ?(constant) |
| If statement | switchCount = 4;  switchCount = 0; | -repeats n times  - repeats for every 4th switchCount increment, meaning that an array must be a minimum size of 4 to reach this point. | N  +  4 |
| If statement | temp = arr[right];  arr[right] = largest;  arr[index] = temp;  right++;  switchCount++; | -repeats a minimum of 1 for an array of size >=2. Repeats for 2 counts  -repeats for 2 counts  -repeats for 2 counts  -Repeats for n-2 times  -repeats twice in total when it accesses this if statement. | 1 or 2(constant)  +  2  +  2  +  n-2  +  2 |
| Else statement | temp = arr[left];  arr[left] = largest;  arr[index] = temp;  Left++;  switchCount++; | -repeats after every 2nd increment of switchCount, meaning that for for an array of size n, it will reach this point n/2 times  - this will reach n/2 times  - this will reach n/2 times  - this will reach n/2 times  - increases twice while in this else statement | N/2  +  N/2  +  N/2  +  N/2  +  2 |
| Total | 20 \* | 10N + 13 (drop constants) | = 20\*N |

After finding both the constance and repeating final values combining them gave me the final value for T(N):

|  |
| --- |
| T(N) = 9 + 20N |

For the Big O, I believe my code runs in O(N^2) due to the while loop at its average case of n+1 times, and the for loop repeating n times as it is searching for the largest number. Dropping the constants and multiplying N \* N we get N^2.

BestCase: O(1) for a size of 1 array.

The Worst Case is the same as the average case for O(N^2).