# Analyze sortOfSort – What it does explained to show understanding.

The class initializes a default array, and then it initializes a sorting array that assorts the array in an ascending sort.

Next, the sorting method that pretty much tells the program what to look for and then what to do. First, it initializes the array’s length, the index location of the array’s max when found, the starting position of where to look in the array, also for the for loop, and then the count to keep track.

The sorting array method is first called telling it to call the sorting array. Next, there is a while loop that tells the program that as long as the starting position “count” is less than the array’s length, keep going. First, we initialize the max position to start so that it can be compared to the rest of the array. Next, a for loop is started that says at index position 0, check if it’s less than the array’s length -1 and if it is, enter the if statement and then increment. The if-statements then tells the program that if the current index is greater than the current max index, make that number the new max, then make note that at this index this is where the max value current resides.

Once the max value is found, it then goes to the if else statement and checks if the incCount is less than 2. If it is, then continue to inside the if statement. In the if statement, this is where we initialize a temporary value and in this case it is the value at the end of the array. Then, we take that value and replace it with the max value, and take the temporary value and place it at the index that once held the maximum value. Lastly, once that is complete you decrement the arrays length by 1 and increase the count for each swap implement.

This goes for the else statement as well but placing it at the beginning of the array and not the end and you increase the start amount because it is now the new starting place for you to look for the new max in the now shortened array, but this statement is only implemented if the count reaches anywhere above 2.

Lastly once the count reaches 4 you restart the count so that it can repeat the process at the if statement. Once completed you print the array, being sure to add the “arrays.toString” method else it would only print out the location and not the content of the array.

# Time Complexity – Sort of Sort

Next, I am going to break this down in parts because each block of code does something different.

**sortArray Method** - This method as you see, is for sorting the array from least to greatest. In this method the Arrays.sort method is used instead of an iterative process. Why? When using Arrays.sort it uses a quick sort algorithm and what we know of the quick sorting algorithms is that it has a best and average case of O(nLog(n)). According to Baeldung, that is a very good sorting time for many collections. The best case only occurs when the there is a middle element picked as the pivot. The worst case is O(n^2) and for this, it’s when the smallest or largest element is used as the pivot, or when the array is already sorted.

**sortOfSort Method** – This method calls the sort array method and then after that it takes the sorted array and commences to sort them in the way they are supposed to be as per the assignment.

First, we find the max of the array. The time complexity of this is 0(n) as the average case, because it goes down each variable in the array to see if the current number is greater than the current max in the array. If yes, then that variable becomes then new max and it continues until it reaches the end of the array. The worse case is, also, 0(n), because when the array is sorted the algorithm requires that you search through every element in the array. The best case using what I just said, would be 0(1), but only when the array is sorted descending or ascending, because then you can either look at the first or last element depending on how it’s assorted.

Next, we tell the code what to do with the max value and it’s index. I would think that the if-else statement is a form of quick sorting because we are in fact switching indexes, I believe the average case time complexity on the if- else statements would be 0(n), because unlike the quick sorting method these if-else statements isn’t sorting them in constant ascending or descending assorted order, it’s just swapping the max an n number of times it’s iterated. In this case, its as long as the incCount variable is less than 2. The best case would be O(1), but that would occur if the value is already sorted and/or if the value is already where it’s supposed to be. Else, it’d be 0(n) as well. As far as this code goes, the maxes at the end of the arrays area already in their correct place. I wouldn’t think it would have a worse case because the index and the max value is already known. It’s just switching the values to the end or beginning of the array.

A lot the time complexities worse cases seem to be heavily dependent on the length the array it’s self because it requires at of time having go back and find the array’s max with the new length after decrementing it in the for loop. Even when tracing and using the visualizations website, the longer the array, the more steps it takes to sort them.

Websites used

<https://www.baeldung.com/arrays-sortobject-vs-sortint>

<https://www.geeksforgeeks.org/quick-sort/>