Step 1: Please use Loop Analysis method to analyze the function

void bubbleSort(int arr[])

Please explain your answer.

```
class BubbleSort
{
    void bubbleSort(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1])
                {
                    // swap arr[j+1] and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
    }

    /* Prints the array */
    void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BubbleSort ob = new BubbleSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        ob.bubbleSort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
    }
}
```

Because the loops are nested, for each value i of the outer loop, we have a different value range of j, and thus we have a different time complexity for each run.

Outer Loop (value of i)	Inner Loop (value range of j)	Outer Loop Time Complexity	Inner Loop Time Complexity	Time Complexity
0	[0, n-2]	O(1)	O(n-1)	O(1) * O(n-1) = O(n-1)
1	[0, n-3]	O(1)	O(n-2)	O(1) * O(n-2) = O(n-2)
2	[0, n-4]	O(1)	O(n-3)	O(1) * O(n-3) = O(n-3)
…	…	…	…	…
n-3	[0, 1]	O(1)	O(2)	O(1) * O(2) = O(2)
n-2	[0]	O(1)	O(1)	O(1) * O(1) = O(1)

Now that we have the time complexity for each i, we must sum up to get the total time complexity:

$O((n-1) + (n-2) + (n-3) + \ldots + 2 + 1) = O((1 + (n-1))*(n-1)/2) = O(n(n-1)/2) = O(n^2)$

Step 2: Optional homework: Subject: Sorting and Searching

75. Sort Colors.

```
class Solution:

    def sortColors(self, nums: List[int]) -> None:

        """

        Do not return anything, modify nums in-place instead.

        """

        n = len(nums)

        for i in range(n-1):

            for j in range(0, n-i-1):

                if nums[j] > nums[j+1]:

                    nums[j], nums[j+1] = nums[j+1], nums[j]
```

Similarly, for each value i of the outer loop, we have a different value range of j, and thus we have a different time complexity for each run.

Outer Loop (value of i)	Inner Loop (value range of j)	Outer Loop Time Complexity	Inner Loop Time Complexity	Time Complexity
0	[0, n-2]	O(1)	O(n-1)	O(1) * O(n-1) = O(n-1)
1	[0, n-3]	O(1)	O(n-2)	O(1) * O(n-2) = O(n-2)
2	[0, n-4]	O(1)	O(n-3)	O(1) * O(n-3) = O(n-3)
…	…	…	…	…
n-3	[0, 1]	O(1)	O(2)	O(1) * O(2) = O(2)
n-2	[0]	O(1)	O(1)	O(1) * O(1) = O(1)

Now that we have the time complexity for each i, we must sum up to get the total time complexity:

$O((n-1) + (n-2) + (n-3) + \ldots + 2 + 1) = O((1 + (n-1))*(n-1)/2) = O(n(n-1)/2) = O(n^2)$