# Heads up!

We do not drop terms from different data sets when we are not aware of the data they contain, because we are unable to ascertain if they will be dominant.

In this example, we iterate through arrA for every item in arrA. The runtimes are multiplied together making the Big O Notation O(arrA * arrA) or more simply O($arrA^2$) (abstracted as O($N^2$), *then* we add the runtime of arrB making the final Big O notation of this function O($arrA^2$ + arrB) which may be abstracted as O ($N^2$ + M)

```
3   function logTwoArrays(arrA, arrB){
4     for(let i=0; i<arrA.length; i++){
5       console.log(arrA[i])
6       for(let j=0; j<arrA.length; j++){
7         console.log(arrA[j])
8       }
9     }
10    for(let j=0; j<arrB.length; j++){
11      console.log(arrB[j])
12    }
13  }
```

# Heads up!

This runs counter to how we calculate Big O when engaging with a single data input, because we always know that the single for loop will be a non-dominate term compared to the nested for loop.

In this example, the Big O notation is $O(N^2)$ - we have dropped the non-dominant loop through arrA.

```
3   function logTwoArrays(arrA, arrB){
4     for(let i=0; i<arrA.length; i++){
5       console.log(arrA[i])
6       for(let j=0; j<arrA.length; j++){
7         console.log(arrA[j])
8       }
9     }
10    for(let j=0; j<arrA.length; j++){
11      console.log(arrA[j])
12    }
13  }
14
```