### **Week 3 - EXAM**

### Guessing the Return Value

0.0/2.0 points (graded)

Consider the following method:

**int foo(int[] array, int num){**

**int i = array.length - 1;**

**while (i >= 0){**

**if (array[i] > num){**

**return i;**

**}**

**i--;**

**}**

**return i;**

**}**

Assuming that we are calling the method with a non-empty array.

What can we say about a call to this method that returns a -1?

1. The method does not work properly.
2. All values in the array are different from num.
3. All values in the array are greater than num
4. All values in the array are greater than or equal to num
5. **All values in the array are less than or equal to num (Answer)**
6. All values in the array are less than num

What can we say about a call to this method that returns a 12?

1. The method does not work properly.
2. All values in the array with indices from 13 to array.length-1 are different from num.
3. All values in the array with indices from 13 to array.length-1 are less than num
4. **All values in the array with indices from 13 to array.length-1 are less than or equal to num (Answer)**
5. All values in the array with indices from 13 to array.length-1 are greater than num
6. All values in the array with indices from 13 to array.length-1 are greater than or equal to num

**Explanation:**

The method explores the array from its end to its beginning searching for the first value that is greater than **num**.

For example, if we declare **int array[] = {12, 13, 14, 1, 2, 3, 5};** and the call to the method is **position = foo(array, 7);**. Within the method, **int i = array.length -1 = 7 - 1= 6**. The loop will check if **6=i >= 0**. If so, it will check if the position **array[i] > num**. As **array[i] = array[6] = 5** is lower than **7**, the method will decrement **i**.

This behaviour will continue until **array[i] > num** or when we finish checking the array, that is, when **i == -1**. As turns out, **array[2] = 14** is greater than **7**. So, the method returns **2**, the position of the last element greater than **num**. If our array is **int array[] = {2, 3, 4, 1, 2, 3, 5};**, and the call to the method is **position = foo(array, 7);** the method will return a **-1**, telling us that there is no element greater than **7**.

### 2. Finding the length of the longest block of sorted values

0.0/2.0 points (graded)

The method **countSorted** is intended to find the length of the longest consecutive block of sorted values in an array, where *sorted* means that the next element in the array is higher than or equal to the previous element in the array.

For example, if the array **arr** contains the values [25, 7, 7, 14, 14, 14, 21, 3, 3, 3, 5, 12, 12, 13, 13], the call **countSorted(arr)** should return 8, the length of the longest consecutive block of sorted numbers: 3, 3, 3, 5, 12, 12, 13, 13.

However, the method **countSorted** that you can see below **does not work as intended**.

**Line 1: int countSorted(int[] array){**

**Line 2: int count = 1;**

**Line 3: int max = 1;**

**Line 4: for (int k = 1; k < array.length; k++) {**

**Line 5: if (array[k-1] <= array[k]) {**

**Line 6: count++;**

**Line 7: } else {**

**Line 8: if (count > max) {**

**Line 9: max = count;**

**Line 10: }**

**Line 11: count = 1;**

**Line 12: }**

**Line 13: }**

**Line 14: return max;**

**Line 15: }**

**Line 16: int [] arr = {25, 7, 7, 14, 14, 14, 21, 3, 3, 3, 5, 12, 12, 13, 13};**

**Line 17: System.out.println(countSorted(arr));**

Enter the value printed on screen by this code segment (Line 17): **6 (Answer)**

Which of the following changes should be made in the code so that the method works as intended? **Be sure that you mark all the correct answers (*Wrong answers subtract points*):**

1. It should return **count** instead of **max**
2. Before line 14, it should check if **count** is greater than **max** and, in that case, do **max = count+1;**
3. It should return **max +1**
4. **Before line 14, it should check if count is greater than max and, in that case, do max = count; (Answer)**

**Explanation:**

If the last block of ordered numbers is at the end of the array, it is counted but not taken into account to update max. So, it is needed to check before leaving if **count** is greater than **max** and, in this case, update **max**.

### 3. Fill the gaps

0.0/7.0 points (graded)

Consider the following methods:

**int foo(String x, int number, int[] array){**

**number = array.length;**

**x = x + "xAtfoo";**

**for (int i=0; i < number; i++){**

**array[i] = array[i] + x.length();**

**}**

**x = x + "Success";**

**return x.length();**

**}**

**void bar(){**

**String s = new String("Hello");**

**int num = 6;**

**int [] array = {1, 2, 3, 4};**

**int result = 0;**

**result = foo(s, num, array);**

**}**

Assuming that we call the method **bar()**, fill the gaps with the values of the variables before and after the execution of **result = foo(s, num, array);**

|  |  |  |
| --- | --- | --- |
|  | **Before** | **After** |
| **s** | **Hello** | **Hello** |
| **num** | **6** | **6** |
| **array[0]** | **1** | **12** |
| **array[1]** | **2** | **13** |
| **array[2]** | **3** | **14** |
| **array[3]** | **4** | **15** |
| **result** | **0** | **18** |

**Explanation:**

This an exercise about the scope of variables and return values.

As **num** is an int, and the value modified only inside the method foo (where **number** is another variable), its value in **bar()** did not change. We have studied in other exercises that you cannot change a String outside a method. So, the string **s** is not changed.

However, remember that you can change the content of and **array**, without having to return it as return value (and this is what happens in this exercise). **result** stores the return value of **foo()**. So, its value after calling **foo()** will be the length of the string **"HelloxAtfooSuccess"**, that is, **18**.

### 4. What does foo return?

0.0/6.0 points (graded)

Consider the following method:

**int foo(int n){**

**if (n < 1){**

**return 0;**

**} else {**

**return 1 + foo(n / 10);**

**}**

**}**

What values are returned as a result of the following different calls to **foo**?

|  |  |
| --- | --- |
| **call** | **return value** |
| **foo(0)** | **0** |
| **foo(1)** | **1** |
| **foo(10)** | **2** |
| **foo(234)** | **3** |
| **foo(1234)** | **4** |

Assuming that **k** is a non-negative integer and **m = 10(to the power of k)**, what value is returned as result of the call **foo(m)?**

1. m/10
2. **K+1 (Answer)**
3. 0
4. k
5. m/10 + 1
6. k/10
7. k/10 + 1

### 5. Number of times that sentences are executed

**0.0/3.0 points (graded)**

Consider the following method:

**/\*\* Precondition: array contains only positive values.**

**\* @param num a positive int**

**\*/**

**void bar(int[] array, int num, int lim){**

**int i = 0;**

**int v = 0;**

**while ((i < array.length) && (array[i] + num < lim)){**

**if (array[i] <= array[v]){**

**v = i; /\* Statement A \*/**

**}**

**i++; /\* Statement B \*/**

**}**

**System.out.println(v);**

**}**

Assume that **bar** is called and executes without error. Which of the following are possible combinations for the value of **lim**, the number of times Statement A is executed, and the number of times Statement B is executed?

**Note: Be aware that 0 is not a positive value.**



|  |  |  |  |
| --- | --- | --- | --- |
|  | Value of lim | Executions of Statement A | Executions of Statement B |
| I. | 5 | 0 | 2 |
| II. | 2 | 0 | 0 |
| III. | 8 | 3 | 10 |

1. All the combinations are possible
2. I only
3. III only
4. **II and III only (Answer)**
5. II only
6. I and II only
7. I and III only

Explanation:

If the number of executions of B is greater than 0, the number of executions of A should be at least 1, as the inner condition array[v] <= array[i] is true at least in one occasion (the first iteration, when **v == i == 0**). So, I is not possible.

As num and all the values of array should be positive (non zero), if the value of **lim** is equal or smaller than 2, the statements A and B are not executed. So, II is possible (whatever the values of array and num).

If the number of items of the array is 10 or more, the number of executions of B could be 10. In that case, we would have at least 10 cases in which **array[i] + num < lim** some of which could make the execution of statement A possible. So, III is also possible.