**Level 1. Basic**

**#1.**

[Write a Java program to find the index of an array element. Print "notfound" if the element does not exist in the array.](https://www.w3resource.com/java-exercises/array/java-array-exercise-6.php)

**#2.**

[Write a Java program to copy an array by iterating the array. Use a for or while loop to do this.](https://www.w3resource.com/java-exercises/array/java-array-exercise-8.php)

**#3.**

[Write a Java program to find the maximum and minimum value of an array.](https://www.w3resource.com/java-exercises/array/java-array-exercise-10.php)

**Leve 2. Advance**

**#1.**

Ratiorg got statues of *different* sizes as a present from CodeMaster for his birthday, each statue having an non-negative integer size. Since he likes to make things perfect, he wants to arrange them from smallest to largest so that each statue will be bigger than the previous one exactly by 1. He may need some additional statues to be able to accomplish that. Help him figure out the minimum number of additional statues needed.

Example

For statues = [6, 2, 3, 8], the output should be  
makeArrayConsecutive2(statues) = 3.

Ratiorg needs statues of sizes 4, 5 and 7.

Input/Output

* **[execution time limit] 3 seconds (java)**
* **[input] array.integer statues**

An array of *distinct* non-negative integers.

*Guaranteed constraints:*  
1 ≤ statues.length ≤ 10,  
0 ≤ statues[i] ≤ 20.

* **[output] integer**
  + The minimal number of statues that need to be added to existing statues such that it contains every integer size from an interval [L, R] (for some L, R) and no other sizes.

Test1: **statues:** [6, 2, 3, 8]

Test2. **statues:** [0, 3]

Test3. **statues:** [5, 4, 6]

Test4.**statues:** [6, 3]

Test5. **statues:** [1]

**#2**.

Given an array of integers, find the pair of adjacent elements that has the largest product and return that product.

Example

For inputArray = [3, 6, -2, -5, 7, 3], the output should be  
adjacentElementsProduct(inputArray) = 21.

7 and 3 produce the largest product.

Input/Output

* **[execution time limit] 3 seconds (java)**
* **[input] array.integerinputArray**

An array of integers containing at least two elements.

*Guaranteed constraints:*  
2 ≤ inputArray.length ≤ 10,  
-1000 ≤ inputArray[i] ≤ 1000.

* **[output] integer**
  + The largest product of adjacent elements.

**#3**

Given an array a that contains only numbers in the range from 1 to a.length, find the first duplicate **number** for which the second occurrence has the minimal index. In other words, if there are more than 1 duplicated numbers, return the **number** for which the second occurrence has a smaller index than the second occurrence of the other number does. If there are no such elements, return -1.

Example

* For a = [2, 1, 3, 5, 3, 2], the output should be  
  firstDuplicate(a) = 3.

There are 2 duplicates: numbers 2 and 3. The second occurrence of 3 has a smaller index than the second occurrence of 2 does, so the answer is 3.

* For a = [2, 4, 3, 5, 1], the output should be  
  firstDuplicate(a) = -1.

Input/Output

* **[execution time limit] 3 seconds (java)**
* **[input] array.integer a**

*Guaranteed constraints:*  
1 ≤ a.length ≤ 105,  
1 ≤ a[i] ≤ a.length.

* **[output] integer**
  + The element in a that occurs in the array more than once and has the minimal index for its second occurrence. If there are no such elements, return -1.