Fun with 1-d Arrays 2016

A `String` has the **Vasco** property if the `String` has the following properties (Yes all of them):

1. The `String` has a length of 5 or greater.
2. At least one letter in the `String` is repeated at least once (somewhere – the letters do not need to be adjacent).
    a. You may assume all letters will be lower case

```
str.length() != null
```

The following code shows the results of the `hasVasco` method.

The following code	Returns
`FunWith1DArrays.hasVasco("bookkeeper"));`	`true`
`FunWith1DArrays.hasVasco("zuzim"));`	`true`
`FunWith1DArrays.hasVasco("java"));`	`false`
`FunWith1DArrays.hasVasco("computer"));`	`false`

An array of `String`s is called a **Morath** array if:

1. The array has 5 elements or more;
2. No `String` appears more than once in the array.
3. 50% or more of Strings in the array have the **Vasco** property

You may assume the array is not `null`.

The following code shows the results of the `isMorath` method.

The following code	Returns

`String[] str1 = {"bookkeeper", "zuzim", "java",`
`"computer", "program", "orange"};`
 	
`FunWith1DArrays.isMorath(str1);`
	`True`

`String[] str2 = {"bookkeeper", "zuzim", "java",`
`"computer", "abba"};`
 	
`FunWith1DArrays.isMorath(str2);`
	`False`

Fun with 1-d Arrays 2016

```
/*
 *    has737   -  an int array has the 737 property if,
 *                every number which contains a 3 is adjacent to
 *                  (previous AND following) a number that contains 7
 *                That is, if num[ind] contains a 3, and BOTH
 *                    num[ind-1] and num[ind+1] contain a 7,
 *                    then num has the 737 property
 *    The following ints contain a 3:  13, 10003, 59834, -783, 3333, -30
 *    and the following ints contains a 7:  70, -948765, 378, 28974, -7
 *
 *    note     -  if num[ind] contains a 3, and
 *                    if either ind - 1 < 0 or ind + 1 >= num.length,
 *                    then the array does not contains the 737 property
 *    note     -  if no value in num contains a 3,
 *                    then the array num has the 737 property
 *
 *    precondition:  num.length >= 0 (num != null)
 *
 *    note -      if num.length == 0, return true
 */
   public static boolean has737(int[] num)
```

The following code shows the results of the `has737` method.

The following code	Returns
`int[] num1 = {74, 23, 17, 80};` `FunWith1DArrays.has737(num1));`	`true`
`int[] num2 = {74, 23, 17, 30};` `FunWith1DArrays.has737(num2));`	`false`

```
/*
 *    remove all occurrences of the digit d, 0 <= d < 10 from the int num.
 *    for example:
 *       186 == removeD(158556, 5)
 *       2168 == removeD(201680, 0)
 *     & -123 == removeD(-123, 5) */
   public static int removeD(int num, int d)
```

The following code shows the results of the `removedD` method.

The following code	Returns
`FunWith1DArrays.removeD(158556, 5);`	`186`
`FunWith1DArrays.removeD(201680, 0);`	`2168`
`FunWith1DArrays.removeD(100057, 1);`	`57`
`FunWith1DArrays.removeD(-123, 7);`	`-123`

Fun with 1-d Arrays 2016

```
/*
 *    returns a List of all the int(s) in the array num which
 *       are the largest after removing the digit d, 0 <= d < 10
 *
 *     remember you are returning the original number
 *
 *    The order of the numbers in the List is not important
 *    You may not alter the array num
 *    You may use your implementation of the method removedD
 *    You may assume num.length() > 0
 */
  public static List<Integer> largestWithOutDigitD(int[] num, int d)
```

The following code shows the results of the `largestWithOutDigitD` method.

The following code	Returns
`int[] num = {1195, 941, 100057, 3186};` `List<Integer> ans1 = largestWithOutDigitD(num, 1)`	
`ans1.get(0).intValue();`	3186
`ans1.size();`	1
`int[] nums = {30936, 9334, 30137, 393336};` `List<Integer> ans2 = largestWithOutDigitD(nums, 3);`	
`ans2.contains(new Integer(30936));`	true
`ans2.contains(new Integer(393336));`	true
`ans2.size();`	2
`int[] nNums = {-387, -9834, -80187};` `List<Integer> nAns = FunWith1DArrays.largestWithOutDigitD(nNums, 8);`	
`nAns.size()`	1
`nAns.contains(new Integer(-80187))`	true

Special note: Both `removedD` and `largestWithOutDigitD` has one test each with only positive numbers. An additional test test both `removedD` and `largestWithOutDigitD` with non positive numbers.

Fun with 1-d Arrays 2016

Degree of Inversion

Compute the extent to which an array of four `int`s is out of sorted order with sorted order meaning that the smallest index has the smallest value, the second index has the second smallest value, and so on until the largest index has the largest value. The metric for computing the degree of inversion is defined as follows. For each array index i count how many elements at larger indexes have values strictly smaller than the value of the element at index i. The degree of inversion is the sum of this count over all indexes. Notice that if the array is sorted, then the degree of inversion is 0.

Borrowed from: 16th Annual Computer Science Programming Contest Department of Mathematics and Computer Science Western Carolina University April 5, 2005

The following code shows the results of the `degreeOfInversion` method.

The following code	Returns
`int[] n1 = {4, 3, 2, 1};` `degreeOfInversion(n1);`	6
`int[] n2 = {1, 2, 3, 4};` `degreeOfInversion(n2);`	0

Array Rank

The rank of an element in an array of **`Comparable`** elements is the number of smaller elements in the array plus the number of equal elements that appear to its left. For example, for the array [“d”, “c”, “i”, “c”, “g”]. The respective ranks of these elements are 2, 0, 4, 1, and 3; thus the rank array is the array [2, 0, 4, 1, 3].

Borrowed from: 15th Annual Computer Science Programming Contest - Western Carolina University March 30, 2004

See next page for a sample input/output.

Fun with 1-d Arrays 2016

The following code shows the results of the `arrayRank` method.

The following code	Returns

`String[] st1 = {"d", "c", "i", "c", "g"};`
`int[] ans = FunWith1DArrays.arrayRank(st1);`
`ans.length;`
	5

`ans[0];`	2
`ans[1];`	0
`ans[2];`	4
`ans[3];`	1
`ans[4];`	3

`Comparable[] com2 = {new  Double(3.5), new  Double(39.5),`
`new  Double(15.7), new  Double(3.7),`
`new  Double(16.9), new  Double(5.9),`
`new  Double(5.8) };`
`ans = FunWith1DArrays.arrayRank(com2);`	

`ans.length;`	7
`ans[0];`	0
`ans[1];`	6
`ans[2];`	4
`ans[3];`	1
`ans[4];`	5
`ans[5];`	3
`ans[6];`	2