Soru 1(FizzBuzz): This is slightly more difficult version of the famous FizzBuzz problem which is sometimes given as a first problem for job interviews. Consider the series of numbers beginning at **start** and running up to but not including **end**, so for example start=1 and end=5 gives the series 1, 2, 3, 4. Return a new String[] array containing the string form of these numbers, except for multiples of 3, use "Fizz" instead of the number, for multiples of 5 use "Buzz", and for multiples of both 3 and 5 use "FizzBuzz". In Java, String.valueOf(xxx) will make the String form of an int or other type. This version is a little more complicated than the usual version since you have to allocate and index into an array instead of just printing, and we vary the start/end instead of just always doing 1..100.

fizzBuzz(1, 6) → ["1", "2", "Fizz", "4", "Buzz"]  
fizzBuzz(1, 8) → ["1", "2", "Fizz", "4", "Buzz", "Fizz", "7"]  
fizzBuzz(1, 11) → ["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz"]

Soru 2(makeBricks): We want to make a row of bricks that is **goal** inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return true if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops.

makeBricks(3, 1, 8) → true  
makeBricks(3, 1, 9) → false  
makeBricks(3, 2, 10) → true

Soru 3(groupSum):Given an array of ints, is it possible to choose a group of some of the ints, such that the group sums to the given target? This is a classic backtracking recursion problem. Once you understand the recursive backtracking strategy in this problem, you can use the same pattern for many problems to search a space of choices. Rather than looking at the whole array, our convention is to consider the part of the array starting at index **start** and continuing to the end of the array. The caller can specify the whole array simply by passing start as 0. No loops are needed -- the recursive calls progress down the array.

groupSum(0, [2, 4, 8], 10) → true  
groupSum(0, [2, 4, 8], 14) → true  
groupSum(0, [2, 4, 8], 9) → false

Soru 4(countClumps):Say that a "clump" in an array is a series of 2 or more adjacent elements of the same value. Return the number of clumps in the given array.

countClumps([1, 2, 2, 3, 4, 4]) → 2  
countClumps([1, 1, 2, 1, 1]) → 2  
countClumps([1, 1, 1, 1, 1]) → 1

Soru 5(seriesUp):Given n>=0, create an array with the pattern {1,    1, 2,    1, 2, 3,   ... 1, 2, 3 .. n} (spaces added to show the grouping). Note that the length of the array will be 1 + 2 + 3 ... + n, which is known to sum to exactly n\*(n + 1)/2.

seriesUp(3) → [1, 1, 2, 1, 2, 3]  
seriesUp(4) → [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]  
seriesUp(2) → [1, 1, 2]

Soru 6(fix34): Return an array that contains exactly the same numbers as the given array, but rearranged so that every 3 is immediately followed by a 4. Do not move the 3's, but every other number may move. The array contains the same number of 3's and 4's, every 3 has a number after it that is not a 3, and a 3 appears in the array before any 4.

fix34([1, 3, 1, 4]) → [1, 3, 4, 1]  
fix34([1, 3, 1, 4, 4, 3, 1]) → [1, 3, 4, 1, 1, 3, 4]  
fix34([3, 2, 2, 4]) → [3, 4, 2, 2]

Soru 7(stringYak):Suppose the string "yak" is unlucky. Given a string, return a version where all the "yak" are removed, but the "a" can be any char. The "yak" strings will not overlap.

stringYak("yakpak") → "pak"  
stringYak("pakyak") → "pak"  
stringYak("yak123ya") → "123ya"

Soru 8(wordCount): The classic word-count algorithm: given an array of strings, return a Map<String, Integer> with a key for each different string, with the value the number of times that string appears in the array.

wordCount(["a", "b", "a", "c", "b"]) → {"a": 2, "b": 2, "c": 1}  
wordCount(["c", "b", "a"]) → {"a": 1, "b": 1, "c": 1}  
wordCount(["c", "c", "c", "c"]) → {"c": 4}

Soru 9(splitOdd10): Given an array of ints, is it possible to divide the ints into two groups, so that the sum of one group is a multiple of 10, and the sum of the other group is odd. Every int must be in one group or the other. Write a recursive helper method that takes whatever arguments you like, and make the initial call to your recursive helper from splitOdd10(). (No loops needed.)

splitOdd10([5, 5, 5]) → true  
splitOdd10([5, 5, 6]) → false  
splitOdd10([5, 5, 6, 1]) → true

Soru 10(equlisNot):Given a string, return true if the number of appearances of "is" anywhere in the string is equal to the number of appearances of "not" anywhere in the string (case sensitive).

equalIsNot("This is not") → false  
equalIsNot("This is notnot") → true  
equalIsNot("noisxxnotyynotxisi") → true