1. Consider the leftmost and rightmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority).

```java
public static int maxSpan(int[] nums) {
    // Implementation...
}
```

2. 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 or 4, and a 3 appears in the array before any 4.

```java
public static int[] fix34(int[] nums) {
    // Implementation...
}
```
3. Return an array that contains exactly the same numbers as the given array, but rearranged so that every 4 is immediately followed by a 5. Do not move the 4’s, but every other number may move. The array contains the same number of 4's and 5's, and every 4 has a number after it that is not a 4. Note that 5’s may appear anywhere in the original array.

```java
fix45({5, 4, 9, 4, 9, 5}) → {9, 4, 5, 4, 5, 9}
fix45({1, 4, 1, 5}) → {1, 4, 5, 1}
fix45({1, 4, 1, 5, 5, 4, 1}) → {1, 4, 5, 1, 1, 4, 5}
```

```java
public static int[] fix45(int[] nums) {
    return null;  // Implement the function
}
```

4. 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.

**Hint**: 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.

```java
groupSum(0, {2, 4, 8}, 10) → true
groupSum(0, {2, 4, 8}, 14) → true
groupSum(0, {2, 4, 8}, 9) → false
```

```java
public static boolean groupSum(int start, int[] nums, int target) {
    return false;  // Implement the function
}
```

5. 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).

```java
equalIsNot("This is not") → false
equalIsNot("This is notnot") → true
equalIsNot("noisxxnotynnottxisi") → true
```

```java
public static boolean equalIsNot(String str) {
    return false;  // Implement the function
}
```
6. Given a string, return a string where every appearance of the lowercase word "is" has been replaced with "is not". The word "is" should not be immediately preceded or followed by a letter - so for example the "is" in "this" does not count. (Note: Character.isLetter(char) tests if a char is a letter.)

notReplace("is test") → "is not test"
notReplace("is-is") → "is not-is not"
notReplace("This is right") → "This is not right"

```java
public static String notReplace(String str) {
}
```

7. Start with two arrays of strings, a and b, each in alphabetical order, possibly with duplicates. Return the count of the number of strings which appear in both arrays. Provide a "linear" solution that makes a single pass over both arrays, taking advantage of the fact that they are in alphabetical order.

```java
public static int commonTwo(String[] a, String[] b) {
}
```


**הוראות המגשת:**

1. קראו聿ן את קובצי הדוגמה הרכיינוים אשיר נמצאים בölüח הקורס.
3. להתחכש ממעטא את קובץ קובצי של השומרון ותffd 언제 המוראה המנהלית הקורס.