CS61B Week 3: Arrays

1. What does the following code snippet print?

```
int[] x = {1, 2, 10, 4, 5};
int[] y = x;
x[2] = 3;
System.out.println(y);
System.out.println(Arrays.toString(y));
```

```
[I@2e6e1408
[1, 2, 3, 4, 5]
```

Note: the exact output of the first statement isn't important, just know that it's gibberish since you're printing a reference to an object.

2.
```
/** Return a new array that is the reverse of the given array L.
 *  Don't modify the original array. */
public static int[] reverseList(int[] L) {
    int res = new int[L.length];
    for (int i = 0; i < L.length; i++) {
        res[L.length - 1 - i] = L[i];
    }
    return res;
}
```

3. Fill in the blanks with either the word 'static' or 'instance' (non-static).

   (a) Static methods should be invoked with the class name, without the need for creating an instance of the class.

   (b) You must instantiate an object to call a(n) instance method of the class.

   (c) Static methods cannot access instance variables or instance methods directly; they must use an object reference.

   (d) Static methods cannot use the 'this' keyword.

4.
```
/** The given array LISTOFNAMES is a list of names (all lowercase)
 *  of more than 9000 important people. Every letter of the alphabet
 *  except one starts at least one of the names. Return the character
 *  that is not the first letter of any of the names in the given list. */
public static char missingFirstLetter(String[] listOfNames) {
    boolean[] characterFound = new boolean[26];
    for (String name : listOfNames) {
        char firstLetter = name.charAt(0);
        int index = firstLetter - 'a';
        characterFound[index] = true;
    }
    for (int i = 0; i < 26; i++) {
        if (!characterFound[i]) {
            return 'a' + i;
        }
    }
}
```

5.
```
/** Return an array that is the reverse of the given array.
 *  Don't use 'new'. You may modify the original array. **/
public static int[] destructiveReverseList(int[] L) {
    for (int i = 0; i <= L.length / 2; i++) {
        int temp = L[i];
        L[i] = L[L.length - 1 - i];
        L[L.length - 1 - i] = temp;
    }
    return L;
}
```

Sample Interview Question of the Week:
You are given an array of doubles of length n (could be very large). You want to return a new array of doubles of length n such that: for all i where $0 \leq i < n$, the number at index i of this new array is the product of every number except the one at index i in the original array. For example, if you are given the array {2, 5, 3}, you should return {15, 6, 10}. The catch is, you can only multiply numbers together—you cannot use divide! The most efficient solution should be able to do this in less than $3n$ multiplications.
Algorithm: Assume our input array is called a and has length n. We will make two new arrays, one (call it b) to store products starting from the beginning of a, and one (call it c) to store products starting from the end of a. So $b[0] = 1, b[1] = a[0], b[2] = a[0] * a[1], b[3] = a[0] * a[1] * a[2]$, etc. Note that we can create each element of b by multiplying the previous element of b with the proper element from a, so that creating our entire b array takes only about n multiplications. Our c array can be formed similarly except from the end of a, so we have $c[n-1] = 1, c[n-2] = a[n-1], c[n-3] = a[n-2]*a[n-1], c[n-4] = a[n-3]*a[n-2]*a[n-1]$, etc. This also only takes about n multiplications. At then end, we can multiply corresponding elements from b and c together to get our answer. This takes another n multiplications, for $3n$ multiplications total.

```
static double[] divideWithoutDivide(double[] a) {
    int len = a.length;
    double[] b = new double[len];
    double[] c = new double[len];

    b[0] = 1;
    for (int i = 1; i < len; i++) {
        b[i] = b[i-1] * a[i-1];
    }

    c[len-1] = 1;
    for (int i = len - 2; i >= 0; i--) {
        c[i] = c[i+1] * a[i+1];
    }

    double[] res = new double[len];
    for (int i = 0; i < len; i++) {
        res[i] = b[i] * c[i];
    }
    return res;
}
```