Array Problems in Java

In these problems, imagine that we are building a Java class called `ArrayTools` that will collect useful functions for arrays. The problems will suggest several such functions.

Problem 1: Duplicate

Write a static function

```
public static int[] duplicate(int[] data)
```

that behaves as follows:

1. If `data` is `null` then return `null`.
2. Otherwise, create a new array `copy` whose size is the same as that of `data`, then copy the information from `data` into `copy`, and then return `copy`.

Follow-Up Questions:

1. Suppose we want a `duplicate` function for arrays of the form `double[] data`. Is is OK to define a second function

```
public static double[] duplicate(double[] data)
```

with the same name as the first `duplicate` function? Explain. How would the code of this new `duplicate` function have to be modified from the original code?

2. Suppose we want a `duplicate` function for arrays of the form `Object[] data`. What would the new function header be? How would the code of this new `duplicate` function have to be modified from the original code? Explain why duplicating `Object[]` does not duplicate the contents of the individual array cells. Can this be a problem in some situations?

Note: Think about similar questions for each of the remaining problems.

Problem 2: Linear Sequence

For testing purposes, it is often convenient to fill an array `data` with a linear sequence of values of the form:

```
b, a + b, 2*a + b, 3*a + b, ...
```

Write a static function

```
public static void linearSequence(int[] data, int a, int b)
```

that will do this task. This function must execute a loop that fills each array cell in turn. Of course, if `data` is `null` then do nothing.

Follow-Up Questions:

1. What function call would you make to fill `data` with the constant 13?
2. What function call would you make to fill `data` with 0, 1, 2, 3, ...?
3. What function call would you make to fill `data` with 1, 2, 3, 4, ...?
4. What function call would you make to fill data with 1, 3, 5, 7, ...?
5. What function call would you make to fill `data` with N, N-1, N-2, ..., 3, 2, 1 where N is the length of the array `data`?

Problem 3: Simple Statistics

Write the following four static functions to gather simple *statistical information* about an array `data`:

```
public static int minimum(int[] data)

public static int maximum(int[] data)

public static int sum(int[] data)

public static int average(int[] data)
```

These functions are not well-defined if `data` is `null` or has 0 length so make the convention that these function return 0 in these cases.

Follow-Up Questions:

1. Should the `average` function return a `double` value for more accuracy even though the array is of type `int[]`?
2. If `average` returns `double` should `sum` also return `double` or is this a bad idea?

Problem 4: Simple Data Movement

Using the ideas of the *Simple Algorithms Lab,* write the following static data movement functions:

```
public static void rotateLeft(int[] data)

public static void rotateRight(int[] data)

public static void reverse(int[] data)
```

The `rotateLeft` function will rotate the array data left and put the value of cell 0 into the last cell. The `rotateRight` function will rotate the array data right and put the value of the last cell into cell 0. The `reverse` function will swap corresponding values from opposite ends of the data array.

Problem 5: Sorting

Using the ideas of the *Simple Algorithms Lab* and the *Faster Algorithms Lab*, write the following static functions:

```
public static void insertionSort(int[] data)

public static void selectionSort(int[] data)

public static void quickSort(int[] data)

public static void mergeSort(int[] data)
```

You should define any `protected` helper functions you need to implement these functions. In particular, for the recursive sorts, `quickSort` and `mergeSort`, it is absolutely essential that you define helper functions.

Problem 6: Linear Search

It is often important to search for a value in an array. *If the array is not sorted, then the only way to search is to examine each array cell one-by-one.* This technique is called *linear search*. Define a function:

```
public static int linearSearch(int[] data, int key)
```

The precise specification of this function is as follows. Loop through the array in increasing order. If you find an index `i` such that `data[i] == key` then return this index `i`. Thus, the return value is the *first index at which* `data[i]` *is equal to the* `key`. If the `key` cannot be found or in the case when `data` is `null` or has length 0 then return `-1` to signal that the search has failed.

Follow-Up Questions:

How would you write this algorithm to return the *last index at which* `data[i]` *is equal to the* `key`? What would you name this alternate algorithm?

Problem 7: Binary Search

The technique of *binary search* is used to find a `key` in an array `data` that is *sorted*. The idea is to eliminate half of the possible array cells at each stage. Suppose that we are examining the range of cells `data[i]` where `min <= i <= max`. Compute the `midpoint` of `min` and `max`. Examine the `data` value at `midpoint`, that is, `data[midpoint]`. If this value is equal to `key`, then return the index `midpoint` since the search is done. If `key < data[midpoint]` then we can eliminate all cells to the right since the array is sorted, that is, we need only examine the sub-range from `min` to `(midpoint - 1)`. Similar arguments apply if `key > data[midpoint]`. If we fail to find `key`, then we return `-1` as in `linearSearch`. Notice that `binarySearch` has a recursive flavor. We will sketch the code below.

```
    public static int binarySearch(int[] data, int key) {
        if (data == null)
            return -1;

        return binarySearch(data, key, 0, data.length - 1);
    }

    protected static int binarySearch(int[] data, int key, int min, int max) {
        // if there is no interval to search
        // return -1 to signal failure
        if (max < min)
            return -1;

        // find the midpoint of the sub-range: min, max
        // if data[midpoint] equals key return midpoint
        // otherwise return what is returned by an appropriate recursive call
    }
```

Follow-Up Questions:

Program `binarySearch` without using recursion and compare the two approaches.

Testing:

You may test all of these functions by placing the definitions in a class named `ArrayTools` in a file `ArrayTools.java`. You may then add this file to a copy of the *Problem Set Framework* and create appropriate test functions within the framework. Notice that the framework already has some helper function to read arrays and create random arrays. This should speed up your testing process.

Notes:

It is sufficient for you to write functions for arrays of type `int[]`. If we were being really fancy, we would define subclasses of `ArrayTools` for each type of array

```
    public class ArrayTools {

        public static class Int {

        }

        public static class Double {

        }

        public static class Object {

        }

    }
```

and then put the array tools in the proper subclasses according to the type of the array `data`. For the purpose of this problem set, we will not expect you to do this.