RecursionFun

This project asks you to solve 8 problems using recursion. In each, the method must call other invocations of the same method in order to find the solution. Each method outlined below should be placed in one Java class called RecursionFun in a file named RecursionFun.java. There should be a corresponding set of JUnit test methods in the unit test class RecursionFunTest in a file named RecursionFunTest.java. Be sure to test your code thoroughly.

To avoid declaring a new RecursionFun object in every test method, consider adding one instance variable to RecursionFunTest as shown below (referenced there by rf). Use this one instance variable in all test methods. Here is a start to RecursionFunTest and RecursionFun:

```java
// This class has test methods for eight recursive methods in RecursionFun.
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class RecursionFunTest {
    RecursionFun rf = new RecursionFun(); // Use rf in all test methods
    @Test
    public void testFibonacci() {
        assertEquals(1, rf.fibonacci(1));
        //...
    }
}

// This class contains ten stand alone recursive methods
public class RecursionFun {
    public int fibonacci(int n) {
        //...
    }
}
```

1) fibonacci
These are the first 9 Fibonacci numbers:

```
1 1 2 3 5 8 13 21 34
```

Each Fibonacci number is the sum of the preceding two (except for the first two, which are both 1). Implement a public static recursive method named fibonacci that returns the nth Fibonacci number.

```java
fibonacci(1) → 1
fibonacci(2) → 1
fibonacci(3) → 2
fibonacci(4) → 3
fibonacci(5) → 5
fibonacci(6) → 8
```

Use this method heading:

```java
// Precondition: n > 0 && n <= 40 (takes a long time when n > 40)
public int fibonacci(int n)
```
2) combinations (from n choose k)

Write a public recursive method `combinations` that implements this recursive definition:

\[
\text{combinations}(n, k) = \begin{cases} 
1 & \text{if } k = n \\
0 & \text{if } k > n \\
\text{combinations}(n-1, k-1) + \text{combinations}(n-1, k) & \text{if } 0 < k < n
\end{cases}
\]

The method must return the number of possible combinations when selecting k elements from n choices. For example, there are six possible combinations when n = 4 and k = 2. If there are 4 letters to choose from such as A, B, C, and D, then there are these six possible sets of two elements: AB, AC, AD, BC, BD, and CD.

- `combinations(3, 2) → 3`
- `combinations(6, 6) → 1`
- `combinations(9, 1) → 9`
- `combinations(100, 1) → 100`
- `combinations(52, 5) → 2598960 // possible poker hands`

Use this method heading:

```java
public int combinations(int n, int k)
```

3) GCD

Implement the Greatest Common Divisor algorithm as a public recursive method `GCD`.

\[
\text{GCD}(m, n) = \begin{cases} 
m & \text{if } n == 0 \\
\text{GCD}(n, m \mod n) & \text{if } n > 0
\end{cases}
\]

Use recursion. Do not use a loop. If only one argument is 0, GCD should return the other argument. GCD is undefined when both arguments are 0. In this case return 0 (Rick has no tests for this undefined case). Hint: Convert the arguments to their absolute value with Math.abs(int). These assertions should pass:

- `assertEquals(3, rf.GCD(24, 9));`
- `assertEquals(3, rf.GCD(-24, 9));`
- `assertEquals(3, rf.GCD(24, -9));`
- `assertEquals(3, rf.GCD(-24, -9));`
- `assertEquals(99, rf.GCD(99, 0));`
- `assertEquals(99, rf.GCD(0, 99));`

Use this method heading:

```java
public int GCD(int m, int n)
```

4) intWithCommas

Write recursive method `intWithCommas` that returns the argument as a String with commas in the correct places.

- `intWithCommas(999) → "999"
- `intWithCommas(1234) → "1,234"
- `intWithCommas(1007) → "1,007"
- `intWithCommas(1023004567) → "1,023,004,567"

Use this method heading:

```java
public String intWithCommas(int n)
```
5) sumArray
Write recursive method **sumArray** that returns the sum of all the int elements in the given range of indexes. Do not use a loop. You must have a recursive call in your answer. The following assertions must pass where the 2nd and 3rd arguments represent the index range:

```java
// Add a test method to RecursionTest.java with assertions like these:
int[] x = {1, 5, 7, 2, 3, 4};
assertEquals(0, rf.sumArray(x, 5, 0)); // No elements in range of 5 through 0
assertEquals(22, rf.sumArray(x, 0, 5)); // Indexes 0..5 represents all elements
assertEquals(13, rf.sumArray(x, 0, 2)); // First 3 elements
assertEquals(9, rf.sumArray(x, 2, 3));
```

Use this method heading:

```java
// Precondition: x.length >= 1, beginIndex < x.length,
// endIndex < x.length, and beginIndex <= endIndex.
public int sumArray(int[] x, int beginIndex, int endIndex)
```

6) reverseArray
Write recursive method **reverseArray** that reverses the array elements in a filled array of ints. Use recursion; do not use a loop. The following assertions must pass:

```java
// Add a test method to RecursionTest.java with assertions like these:
int[] a = {2, 4, 6};
rf.reverseArray(a);
assertEquals(6, a[0]);
assertEquals(4, a[1]);
assertEquals(2, a[2]);
```

Use this method heading:

```java
// Precondition: beginIndex == 0 and rightIndex == x.length-1,
public void reverseArray(int[] x)
```

7) arrayRange
Write recursive method **arrayRange** that returns the maximum integer minus the minimum integer in the filled array of ints. Use recursion; do not use a loop. The following assertions must pass (note the shortcut way to pass a reference to a new array--it saves your writing a bit of code):

```java
assertEquals(2, rf.arrayRange(new int[] {1, 2, 3}));
assertEquals(2, rf.arrayRange(new int[] {3, 2, 1}));
assertEquals(0, rf.arrayRange(new int[] {3}));
assertEquals(3, rf.arrayRange(new int[] {-3, -2, -5, -4}));
```

Use this method heading:

```java
// Precondition: a.length > 0
public int arrayRange(int[] a)
```

8) binarySearch
Implement recursive method **binarySearch** to return the index of the first String that equals searchValue. Use recursion; do not use a loop. Use the binary search algorithm so searches run O(log n). Use a sorted array of unique strings to test binarySearch. The following assertions must pass:

```java
// Add a test method to RecursionTest.java with assertions like these:
String[] strs = {"Aaa", "Ccc", "Ddd", "Fff", "Hhh", "Ttt"};
assertEquals(0, rf.binarySearch("Aaa", strs));
assertEquals(4, rf.binarySearch("Hhh", strs));
```

Use this method heading:

```java
// Precondition: strings.length > 0 (no empty arrays)
public int binarySearch(String searchValue, String[] strings)
```
Grading Criteria

+/100 Web-Cat correctness and code coverage: To get 100 points, you will need 100% code coverage and 100% problem coverage (Rick's tests pass and you exercised all statements in your code via your unit tests). These 100 points are derived from Web-Cat. You may submit as many times as you wish until you have 100% on both. Notice that a multiplication feature is employed that means 90% code coverage and 90% problem coverage results in $0.9 \times 0.9 \times 80$ for 64/80 points. This is only 81% rather than 90% for this portion of the grade.

Please note these other possibilities for loosing points (0 is the minimum score)

+/ -12 For any method that fails to use recursion in either the public method or a private helper method

+/ -100 If you any one of YOUR assertions fails on WebCat, or if you have any one compiletime error —and all it takes is one wrong letter in a class or method name or the wrong type or number of parameters for one compile time error. Please read all WebCat feedback.