CS 143 Assignment 3
Introduction to Recursion
See Canvas for due date!

For this assignment, you will implement a few recursive methods in a Java class called `RecursionIntro`. For this assignment, there are some rules spelled out below:

You may not make use of any Java classes other than incidental use of Strings, calls to System.out.println, and accessing an existing array passed to your method. The tester program will verify that your program contains no imports, no use of "new", and the only uses of . (dot) are within calls to System.out.println, accessing array .length, or followed by a digit (double literals).

You may not use `for` or `while`. Any looping must be accomplished using recursion. The tester program will check for no "for"s or "while"s and may be triggered by the word in a comment.

You may not declare static or non-static member fields – only local or parameter variables allowed.

You may implement additional helper methods to use the "recursive driver" technique as we did with the find method in class, or for any other subtasks you may need. (You may reuse code from class!)

Part 1. Even digits up, odd digits down (30 pts)
Implement the method `public static long eduodd(long n)`. eduodd(n) returns a value which increases each of the even decimal digits of n by one and decreases each of the odd digits of n by one. Leading one digits will disappear. The sign of eduodd(n) should be the same as n, unless n is negative and eduodd(n) is zero as seen in the last example below.

Please review the written practice recursion problems as seen in class. Table of examples:

n	0	27	987654321	-8443	11121113	-11
eduodd(n)	1	36	896745230	-9552	30002	0

Part 2a. Recursive definition (20 pts)
Implement the method `public static int fibby(int n)`. fibby is mathematically defined for nonnegative values in the following way:

fibby(0) = 1
fibby(n) = fibby($\lfloor n/3 \rfloor$) + fibby($\lfloor 2n/3 \rfloor$) where $n > 0$ and $\lfloor x \rfloor$ means the floor of x (round down).
HINT: recall Java's integer division behavior. Table of examples:

n	1	2	3	4	5	6	7	8	9	10	20	100
fibby(n)	2	3	5	5	7	8	8	10	13	13	23	119

Part 2b. Sparse table generation (20 pts)
Notice that for many values i, fibby(i) = fibby(i+1). Implement the method `public static void printSparseTable(int start, int end)`. Output using System.out.println all consecutive values of n and fibby(n) (just the two numeric values separated by a space) where $n \geq start$ and $n \leq end$. However, skip a line of output if the previous row printed has the same fibby value.

For example, if `printSparseTable(4, 10);` is called, you would print:

```
4 5
5 7
6 8
8 10
9 13
```

Note that even though fibby(3) == fibby(4), since we didn't print fibby(3), we still print fibby(4). But we skip fibby(7) because it equals the previously printed fibby value. We also leave out fibby(10) because it equals the last printed fibby(9). A helper method will probably help with this. (As an aside, "sparse" refers to the existence of gaps in the table and is used for matrices, etc.)

Part 3a. Largest power of two less than (10 pts)

Implement the method `public static int lp2lt(int n)` which calculates and returns the largest integer power of 2 less than n. You may assume n is greater than 1. For example, if n is 10, the largest power of 2 less than n is 8 (8 is 2 cubed). If n is 8, the answer is 4. If n is 2, $2^0 = 1$.

Part 3b. There can be only one (20 pts)

You are holding a contest to determine the ultimate penny champion. Each participant in the contest has decided which side of the penny they like, heads or tails, and keeps that decision to themselves. We represent heads using the `boolean` value `true` and tails using `false`. The contest participants all get in a long line (array). They start by pairing up (the first two, second two, and so forth). After the two members reveal their pennies to each other (battle), if they are different boolean values, the first (left) person wins, otherwise, the second (right) person wins. There are no draws/ties. The battles are single elimination: once a participant loses, they stop playing in the contest. The remaining members then battle it out the same way in the next round (first winner of the first round battles the second winner of the first round, third winner of the first round battles the fourth winner, etc.). Battle rounds continue to occur in this way until only one winner emerges. If there are an odd number of people in any round, the last person gets a "bye" and automatically survives to the following round.

Write the following recursive method: `public static int champion(boolean[] a)`

It should return the **index** corresponding to the winner of the contest. You do not need to allocate any arrays for this problem and **you should not modify the input array as it contains the decisions that the participants have made, which never change**. Instead, adapt what we learned from binary search. **You will probably need a helper method. You may assume that the array is at least length 1.**

Note: It can be proven that the above problem is equivalent to dividing up the line of participants into two parts (where each part is a contiguous "piece" of the line), performing a totally separate champion contest for each part, and having the winners of the two separate contests pairing up for a final match. The length of the first part is the largest power of two less than the number of participants, and the second part is the remaining participants. (Part 3a will help!)

For example, if there are 11 participants, you would divide into a contest of the first 8 participants followed by a contest of the other 3. The 3 participants would have a pair battling with the other in a bye. The winner of the faceoff between the pair winner and the bye would survive until the final battle! See if you understand how this version of the problem results in the same sequence of battles that would occur in the earlier description.