CSE 332 Data Abstractions, Spring 2013
Homework 6

Due: **Friday, May 24, 2013** at **2pm via catalyst dropbox**. You will want to download the bundle of code skeletons posted on the homework web page that goes with this homework.

Problem 1: Forkjoin Parallelism: IsOver

Write a parallel fork-join algorithm in Java using the ForkJoin Framework to determine if any integer in a given array is over the provided value. Your algorithm should have work O(n) and span O(log n) where n is the length of the array. For example, if `arr` is `{21,17,35,8,17,1}` then `isOver(21,arr)` is true and `isOver(35,arr)` is false. You can assume that: `arr` will always contain at least one element. You may include a sequential cut-off or not – your choice.

```
public static boolean isOver(int val, int[] arr)
```

We have provided a test harness `TestProblem1.java` that your code should work with (note: it only contains a couple of tests). Please write any additional code (including additional classes) that you need to solve the problem. **You may find it useful to look at the code we provided for Problem 2, examples from section, and [Introduction to the ForkJoin Framework (JSR 166)](#) to help you get started**.

Problem 2: Forkjoin Parallelism: Longest Sequence

Consider the problem of finding the longest sequence of some number in an array of numbers: `getLongestSequence(i,arr)` returns the longest number of consecutive i in arr. For example, if arr is `{2,17,17,8,17,17,17,0,17,1}` then `getLongestSequence(17,arr)` is 3 and `getLongestSequence(9,arr)` is 0. You can assume that: `arr` will always contain at least one element.

a) Write a parallel fork-join algorithm in Java using the ForkJoin Framework for `getLongestSequence`. Your algorithm should have work O(n) and span O(log n) where n is the length of the array. Do *not* employ a sequential cut-off: your base case should process an array range containing one element. We have provided 3 files to use for this: `TestProblem2.java`, `LongestSequence.java`, and `Results.java`. If you would prefer to solve the problem in a different manner, say using a `RecursiveAction` or by modifying or not using the `Result` class, that will be fine. Just be sure that your method has this signature and works as specified above:

```
public static int getLongestSequence(int i, int[] arr)
```

b) Modify your solution to part (a) to use a sequential cut-off. Thus your code will need to show what you would do below this cut-off. Name this new method as follows and write a new class called `LongestSequenceCutOff`:

```
public static int getLongestSequenceCutOff(int i, int[] arr)
```

c) ANSWER THIS QUESTION AS A COMMENT INSIDE OF `LongestSequenceCutOff.java` file: In general, how does having using a sequential cut-off make code more efficient?

TURN PAGE OVER FOR THE LAST PROBLEM

Problem 3: Forkjoin Parallelism: Leftmost Occurrence of Substring

Consider the problem of finding the leftmost occurrence of a sequence of characters in an array of characters, returning the index of the leftmost occurrence or -1 if there is none. For example, if the characters stored in the array `arr1` are: `cse332`, and the characters stored in the array `arr2` are: `Dudecse4ocse332momcse332Rox,` then `getLeftMostIndex(arr1, arr2)` is 9.

Write a parallel fork-join algorithm in Java using the ForkJoin Framework for `getLeftMostIndex`. Your code should be callable via the method signature listed below. You may assume that `arr1` (the sequence you are searching for) will be at most 6 characters in length (you do not have to check for this). You should include a sequential cut-off (pick one that is large-ish, say 100). You can assume that: `arr1` and `arr2` will always contain at least one element

```
public static int getLeftMostIndex(char[] arr1, char[] arr2)
```

Hint: Your solution will be _much_ simplified if you solve slightly overlapping subproblems.

Tip: You may find it useful to convert Strings to arrays of characters using toCharArray():

```
char[] arr = "Dudecse4ocse332momcse332Rox".toCharArray();
```

**Really, did you remember to answer Problem 2, Part c??