Fork-Join Parallelism

Start at
https://homes.cs.washington.edu/~djg/teachingMaterials/spac/grossmanSPAC_forkJoinFramework.html

If you do not already have one, create an account at replit.com (or, use your own local installation of Java). These will be a current version of Java that includes the fork-join framework, so you don't need to install anything else.

Jump to "Getting Started" on that page.

1) Useful Example

Get the "Useful Example" on that page running. In replit you'll have to create two files for the two classes.

2) Reverse an Array

Complete the fork-join code below to solve the following problem:
• Input: An int[] (though the element type happens to be irrelevant)
• Output: A new int[] where the elements are in the reverse order. For example, the element in the last array index of the input will be at index 0 in the output.
Your solution should have O(n) work and O(log n) span where n is the array length (in other words, it should use efficient fork-join parallelism). Don't bother with a sequential cut-off: the base case should process one array element.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class Main {
  static final ForkJoinPool fjPool = new ForkJoinPool();

  static int[] reverse(int[] array) {
    // ADD A LINE HERE
    fjPool.invoke(new Reverse(answer,array,0,array.length)); // DO NOT CHANGE
    // ADD A LINE HERE
    }
  }
}

// DEFINE A SECOND CLASS TO CONTAIN YOUR DEFINITION OF REVERSE AND COMPUTE
```

3) Analyzing the Previous Problem

Suppose a program uses your solution to the previous problem to reverse an array containing 2^27 elements.
(a) In English, about how big is 2^27? For example, "one thousand" is an answer in the right form, but

is the wrong answer.
(b) When the program executes, how many fork-join threads will your solution to the previous problem create? Give both an exact answer and an approximate English answer.
(c) Suppose you modify your solution to use a sequential cut-off of 1000. When this modified program executes, how many fork-join threads will it create? Give both an exact answer and an approximate English answer.

4) Parallel Merge Sort

Write a parallel fork-join merge sort algorithm. You do NOT have to write a parallel merge phase — instead, you can use a normal sequential merge. Once you have the parallel algorithm working, add a sequential cutoff, below which you just call Arrays.sort(). Write a test harness that generates random numbers for the algorithm to sort. Experiment to find the optimal cutoff point (i.e., when is it preferable to just sort sequentially). If you're using replit.com, you may never see gains from parallelism, but on a local Java installation you should be able to get around 2x speedup over sequential code using 2-4 cores for millions of elements.