CSE 332 Data Abstractions, Spring 2016
Homework 6

Due: Wednesday, May 17, 2016 at the BEGINNING of lecture. Your work should be readable as well as correct. The first two problems should be submitted through GitLab, and the second two on paper.

Problem 1: `getLeftMostIndex`
Submit your solution to this problem using Gitlab.
Use the ForkJoin framework to write the following method in Java:

```java
public static int getLeftMostIndex(char[] needle, char[] haystack, int seqCutoff)
```

Returns the index of the left-most occurrence of needle in haystack (think of needle and haystack as strings) or -1 if there is no such occurrence.

For example, `getLeftMostIndex("cse332","Dudecse4ocse332momcse332Rox")` == 9
and `getLeftMostIndex("sucks","Dudecse4ocse332momcse332Rox")` == -1.

Your code must actually use the `seqCutoff` argument. You may assume that `needle.length` is much smaller than `haystack.length`. A solution that solves overlapping subproblems will be significantly cleaner and simpler than one that does not.

Problem 2: `hasOver`
Submit the solution to this problem using Gitlab.
Use the ForkJoin framework to write the following method in Java:

```java
public static int[] filterEmpty(String[] arr)
```

Returns an array with the lengths of the non-empty strings from arr (in order).

For example, if `arr` is `"", "", "cse", "332", ",", "hw", ",", "7", "rox"],` then
`filterEmpty(arr) == [3, 3, 2, 1, 3].`

A parallel algorithm to solve this problem in $O(\lg n)$ span and $O(n)$ work is the following:
1) Do a parallel map to produce a bit set
2) Do a parallel prefix over the bit set
3) Do a parallel map to produce the output.
Problem 3: Amdahl’s Law: Graphing the Pain

Use a graphing program such as a spreadsheet to plot the following implications of Amdahl’s Law. For both part a and part b, turn in 1) the graphs and 2) tables with the data. (You may take the definition of Amdahl’s law from the course notes, section 4.2, page 27-28).

(a) Consider the speed-up \( \frac{T_1}{T_P} \) where \( P = 256 \) of a program with sequential portion \( S \) where the portion \( 1 - S \) enjoys perfect linear speed-up. Plot the speed-up as \( S \) ranges from 0.01 (1% sequential) to 0.25 (25% sequential).

(b) Consider again the speed-up of a program with sequential portion \( S \) where the portion \( 1 - S \) enjoys perfect linear speed-up. This time, hold \( S \) constant and vary the number of processors \( P \) from 2 to 32. On the same graph, show four curves, one each for \( S = 0.01, S = 0.1, S = 0.2, \) and \( S = 0.4 \).

Problem 3: Parallel Quicksort

Lecture presented a parallel version of quicksort with best-case \( O(\log^2 n) \) span and \( O(n \log n) \) work. This algorithm used parallelism for the two recursive sorting calls and the partition.

(a) For the algorithm from lecture, what is the asymptotic worst-case span and work. For both, state a recurrence and solve it – show your work solving the recurrence.

(b) Suppose we use the parallel partition part of the algorithm, but perform the two recursive calls in sequence rather than parallel. What is the asymptotic worst-case span and work? For both, state a recurrence and solve it – show your work solving the recurrence.