# CSE332 Data Abstractions, Spring 2010
# Homework 6

Due: **Friday, May 21, 2010** at the beginning of class. Your work should be readable as well as correct.

This assignment has **three** problems.

### Problem 1. Fork-Join Parallelism: Longest Series

Consider the problem of finding the longest sequence of some number in an array of numbers: `longest_sequence(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 `longest_sequence(17,arr)` is `3` and `longest_sequence(9,arr)` is `0`.

(a) In pseudocode, give a parallel fork-join algorithm for implementing `longest_sequence`. 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. Hint: Use this definition:

```
class Result {
  int numLeftEdge;
  int numRightEdge;
  int numLongest;
  boolean entireRange;
  Result(int l, int r, int m, boolean a) {
    numLeftEdge=l; numRightEdge=r; numLongest=m; entireRange=a;
  }
}
```

For example, `numLeftEdge` should represent the length of the sequence at the *beginning* of the range processed by a subproblem. Think carefully about how to combine results.

(b) In English, describe how you would make your answer to part (a) more efficient by using a sequential cut-off. In pseudocode, show the code you would use below this cut-off.

### Problem 2. Fork-Join Parallelism: Leftmost Occurrence of Substring

Consider the problem of finding the leftmost occurrence of the sequence of characters `cseRox` in an array of characters, returning the index of the leftmost occurrence or `-1` if there is none. For example, the answer for the sequence `cseRhellocseRoxmomcseRox` is `9`.

(a) In English (though some high-level pseudocode will probably help), describe a fork-join algorithm similar in design to your solution in problem 1. Use a sequential cut-off of at least 6 (the length of `cseRox`) and explain why this significantly simplifies your solution. Notice you still must deal with the leftmost occurrence being "split" across two recursive subproblems.

(b) Give a much simpler fork-join solution to the problem that avoids the possibility of a "split" by using slightly overlapping subproblems. Assume a larger sequential cut-off, for example 100. Give your solution precisely in pseudocode. Avoid off-by-one errors.

**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. Turn in the graphs and tables with the data.

(a) Consider the speed-up ($T_1/T_P$) 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 .01 (1% sequential) to .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 three curves, one each for $S = .01$, $S = .1$, and $S = .25$.