Final Assignment: Control Structures

A three to five minute presentation to the class on your approach.
Time allocation (max): thinking, 8 hours; designing, 8 hours; implementing 20 hours
HAND IN YOUR WORK (notes, code, comments, results)

Select from and complete as many of the exercises as you can within the time allocation.

I. Sorting

Comprehension exercise: Implement several *sorting algorithms* (possibly by copying algorithms from the book). Design and implement a *record generator* which generates random collections of record indices (ie numbers or key words). Design and implement a simple *test statistics package* which counts the number of sorting steps for each sorting algorithm. Answer these questions:

1. Make a graph of sorting steps (record swaps or relocations) vs size of input (number of records) for each sorting algorithm. Do your algorithms perform as the book predicts?

2. Characterize the essential difference between each algorithm. Why do some algorithms perform better or worse than others?

3. Design several different input orderings to sharpen the difference in performance between your algorithms. That is, build input sets which are almost completely ordered, almost completely out-of-order, in some random ordering, each value duplicated once, all the same value, etc. Vary the size the input sets and their ordering characteristics, and test the efficiency of your algorithms.

4. For all experiments, abstract the performance in terms of asymptotic notation.

5. Characterize the stability of each algorithm. That is, differentiate between sorting which may move elements with the same key, and those that will not move them.

6. ***Hybrid algorithm* (challenge):** mix what appear to you to be the best parts of your algorithms, to build a hybrid sorting algorithm with better characteristics than the ones you started with.

II. Algorithm Variety

1. Factorial: Implement many substantively different versions of the factorial algorithm (factorial computes the product of 1..n integers). Compare each for efficiency and for ease of writing and maintenance.

If you look on the web, there are collections of dozens of ways to implement factorial (that is, this problem can be addressed with research as well as with creativity). Some methods require an appropriate engine, which you may or may not have. For example, an object-oriented implementation requires an object-oriented language. Don't implement engines, do use different languages when appropriate.

Can you find the fastest possible algorithm?

2. Fibonacci: Implement many substantively different versions of the Fibonacci algorithm Fibonacci computes the sum of the previous two Fibonacci numbers:

```
        Fib[0] = 0
        Fib[1] = 1
        Fib[n] =  Fib[n-1] + Fib[n-2]
```

Compare for efficiency and ease of use. Identify three classes of algorithm (highly inefficient, efficient, and highly efficient) with an implementation of each.

3. Generalization (challenge): Design and implement an abstract procedure which computes any simple recursive function, given the base and the recursion relation.

III. Tree Searching

Assume that you have a tree data structure and you have to search it for a leaf with a particular value.

1. Searching: Design and implement several search algorithms for visiting the leaves of the tree. Standard techniques include depth-first, breadth-first, best-first, hill-climbing, iterative deepening, and iterative broadening. Which are most efficient and why?

2. Structured search: In what ways can specialized tree data structures (such as balanced search trees) help to improve the efficiency and ease of programming search algorithms? Analyze the tradeoff between complex data structures for search and complex search algorithms.

3. Generic search: Design and implement a single algorithm which takes the type of search as a parameter. Other generalization parameters which may be of interest include

- the goal predicate which tests when the leaf being sought after is found
- a function which returns the children of an interior node in the tree. (This is used for determining the cost of various search strategies.)
- a priority function which determines which node to search next.

4. Smart search (challenge): Design and implement an algorithm which dynamically determines which kind of search is best for the particular context, given the depth, branchiness, and other characteristics of the tree at the node currently being explored.