COMP5408: Winter 2014 — Assignment 2

This assignment contains a theory part and an implementation part. You should do either the theory part or the implementation part, but not both.

1 Theory Part

If you choose to do this part of the assignment then you should write up your solutions on paper (word processed in LaTeX would be nice) and give them to me in class.

1. Suppose we have a probability distribution $p_1, \ldots, p_n$ where each p_i is of the form 2^{-x_i} and x_i is an integer. Describe a data structure that allows us to generate random numbers according to this distribution using only coin flips. The expected number of coin flips required to generate a number should be $H(p_1, \ldots, p_n)$.

2. **Warning: Not easy!** Suppose you have access to a random number generator that allows you to generate integers uniformly at random in $\{0, \ldots, W-1\}$ in constant time. You are given a probability distribution $p_1, \ldots, p_n$, where each p_i is of the form k_i/W, for some integer k_i. Describe a data structure that allows you to generate random numbers according to this distribution. Your data structure should be as fast as possible and use as little space as possible. Ideally, you could generate each sample in constant time using a data structure that uses $O(n)$ space.

3. Tarjan has shown that, given any splay tree containing the keys $1, \ldots, n$, if we search for the keys $1, \ldots, n$ in order then the total cost of all searches is $O(n)$. Show how this implies that, given any two binary trees T_1 and T_2 each on n nodes, the tree T_1 can be converted into the tree T_2 using $O(n)$ rotations.

4. Skiplists can do finger searches in $O(\log f)$ expected time. Using this fact, show how to merge two skiplists of size m and n ($m \leq n$) into one skiplist in $O(m \log \frac{n}{m})$ expected time. (Hint: You'll have to make use of Jensen's Inequality, from the chapter on entropy.)

5. Describe a priority queue that has the *working set* property. That is, insertions should take $O(1)$ amortized time and, if a call to DeleteMin() deletes the element x, then this should take $O(\log t(x))$ time, where $t(x)$ is the number of elements in the priority queue that were inserted after x was inserted. Thus, if we use this priority queue like a stack, the insertions and deletions (pushes and pops) will take constant amortized time.

2 Implementation Part

You are expected to thoroughly test your code to make sure that it implements these operations correctly, that it doesnt crash, and that it performs well. Your submission (a zip file) should include your testing code and should include a README file that describes the tests you have performed as well as instructions on how to run your test code.

Your testing should test both the correctness and performance of your implementations. All the operations you are asked to implement are fast, so your performance tests should include tests in which

(at least) hundreds of thousands of operations are performed on trees containing hundreds of thousands of elements.

1. Complete an implementation of fractional cascading for iterated search as discussed in class. In particular, you need to the implement the constructor and the `find(x)` method in the `FractionalIteratedSearcher` class. For comparison purposes the `TrivialIteratedSearcher` class is provided.

2. Complete the Implementation of Sleator and Tarjan's *splay trees* discussed in class. Do this by completing the implementation found in the file `SplayTree.java`. In particular, you need implement the `splay(u)` operation and the `remove(x)` operation.