CSE 542 – Advanced Data Structures and Algorithms *Jon Turner*

Lab 4

Due 4/2/2013

General notes for labs. Review and follow the general notes from lab 1.

The round robin algorithm uses leftist heaps because they support an efficient melding operation. In this lab, you will re-implement the round-robin algorithm using Fibonacci heaps, which also support efficient melding. In order to use Fibonacci heaps with the round-robin algorithm, we need to extend them so that they also support lazy deletion. (We won't need to implement lazy melding, since the meld operation for Fibonacci heaps is already a constant time operation.) To implement lazy deletion, we need to add support for a *deleted* function. See the provided implementation of leftist heaps (called *LlheapSet*) to see how this is done. In addition, we need to implement a *purge()* method to eliminate deleted nodes from the top of the heap, and return a list of non-deleted tree roots that can be combined to form a single heap. *Findmin()* and *deletemin()* should both use *purge()* followed by the existing *mergeRoots()* method which restructures the heap to reduce the number of separate trees. Your repository includes a class called *LfheapSet*, which is a sub-class of *FheapSet*. You will need to complete the missing elements of *LfheapSet*. More details can be found in the source code files.

The *unit* sub-directory of the *heaps* directory contains a program *testLfheapSet* that you can use to test your implementation of *LfheapSet.* Note that the test program expects a certain specific form for the heaps after each test is run. There are many equivalent ways to represent a given heap, so don't be concerned if your program produces heaps that do not exactly match those expected by the unit test. You should verify that they are equivalent, but they need not be exactly the same. Once you have *LfheapSet* working correctly, you will need to implement a new version of the round-robin algorithm, called *rrobinF* that uses your *LfheapSet*.

You will also evaluate the performance of *rrobinF*, comparing it to the original *rrobin* and to *prim.* For each algorithm you will measure the time spent on initialization and the total time. In addition, for each algorithm you will count two additional components of the running time that will vary among the three algorithms.

- For *prim*, you will measure *siftupCount* and *siftdownCount* in the underlying heap data structure. These counts keep track of the number of elementary steps used by *siftup* and *siftdown*.
- For *rrobin*, you will measure the total number of calls to the recursive *meld()* method in the leftist heap data structure (this is called *meldCount*) and the total number of find steps in the partition data structure (this is called *findCount*).
- For *rrobinF*, you will measure the total number of steps performed by the recursive *mergeRoots()* method in the Fibonacci heap data structure (this is called *mrCount*) and the total number of find steps in the partition data structure.

You'll find more details in the lab report template.