CSE 5311 Lab Assignment 2

Due July 9, 2003

Goals:

1. Review of binary heaps.

2. Understanding of d-heaps.

Requirements:

1. Write (and test) a program that performs the following computation for various branching (d) values:

   1. Generate two million d-heap elements with ID numbers from 0 . . . 1,999,999 and random priorities from 0 . . . 20,000.
   2. Build a d-heap (min-heap ordering).
   3. Insert two million additional elements into your d-heap (ID numbers from 2,000,000 . . . 3,999,999).
   4. Randomly change the priority of each of the four million d-heap elements.
   5. Extract each d-heap element (ascending priority order).

   Your program must compile and execute on OMEGA. There should be a comment near the beginning of your code that indicates how to compile on OMEGA. Your debugging trace should be disabled in the version you submit.

2. Prepare a brief report summarizing the performance of your code for various d values. Your report may be a text, html, PostScript, PDF, or MS Word file.

3. Email your code and report (as attachments) to `yxb4544@omega.uta.edu` before 3:00 pm on July 9. The subject should include your name as recorded by the University.

Getting Started:

1. Either array element 0 or array element 1 may be used as the root of your tree. Regardless of your choice, you should first work out the details of the mapping.

2. Besides keeping the priorities in your min-heap. it is important to simulate the maintenance of the data that accompanies each priority. Each heap item will have a ID number from 0 . . . 3,999,999 that will move within the heap tree along with its prioriy. There will also be a separate table that will allow finding the heap item for a particular ID number. This separate table is useful when the priority of an item changes.

3. Priorities should be values from 0 . . . 20,000. Priorities may either increase or decrease.

4. You should initially run your program with a variety of values for d. After observing the values that give good results, the version you submit should use five of these values.

5. Using a compiler code optimization option (-O2 for C) will be worthwhile.

6. `getrusage()` may be used to capture CPU times for the various phases of your code:

```
#include <sys/time.h>
#include <sys/resource.h>

float CPUtime()
{
  struct rusage rusage;
  getrusage(RUSAGE_SELF,&rusage);
  return rusage.ru_utime.tv_sec+rusage.ru_utime.tv_usec/1000000.0
    + rusage.ru_stime.tv_sec+rusage.ru_stime.tv_usec/1000000.0;
}
```