Implementation of data structures and algorithms
Fall 2018
Short Project 3: Priority queues
Tue, Sep 11, 2018

Version 1.0: Initial description (Tue, Sep 11).

Due: 11:59 PM, Sun, Sep 23.

Submission procedure:

* Create a folder whose name is your netid (NId).
* Place all files you are submitting in that folder.
* Use "package NId;" in all your java files.
* Include the class files also in your zip file.
* Include a text file named "readme.txt", that explains how to compile and run the code.
* Zip the contents into a single zip or rar file.
* If the zip file is bigger than 1 MB, you have included unnecessary files.
* Delete them and create the zip file again.
* Upload the zip or rar file on elearning.
* Submission can be revised before the deadline.
* The final submission before the deadline will be graded.
* Only one member of each team needs to submit project.
* Include the names of all team members in ALL files.

Team task:

1. Implement binary heap. Starter code is provided.

Optional tasks (for individual submission):

2. Implement heap sort and its related methods in the binary heap class.
   Include in your submission, a driver that uses heap sort to sort an array
   in ascending order, and then in descending order.

3. Implement the problem of finding the kth largest element of a stream (or k largest elements)
   using the following algorithm: maintain the k largest elements seen so far in
   a priority queue (min heap for finding k largest elements). Implement replace()
   method in binary heap class. Compare its performance with code that uses
   Java's priority queue (use small values for k (100, say) and large values of n).

Use Java's PriorityQueue for the following questions.

4. Implement Huffman Coding algorithm. Create a class for representing coding
   trees. Use a priority queue to hold the trees. In each step, the algorithm
   removes two trees with the smallest frequencies, merges them, and inserts it
   back into the priority queue. At the end, there is a single coding tree.
   Traverse the tree and output the binary codes for each symbol.

5. Perfect powers: Write an algorithm to output exact powers (numbers of the
   form a^b, b>1), in the range 2 to n.

6. Given an array of prime numbers, output numbers in [2,n] all of whose
   prime factors are only from the given set of prime numbers.
   For example, given {3,7}, the program outputs {3,7,9,21,27,49,63,...}.
   Make sure that your program outputs each number only once, in sorted order.