CS 2420 – Spring 2018
Program 5 –30 points Priority Queues Dynamic Median

Part 1: Implement a min heap as skew heap or a leftist heap, whichever you prefer. The input file, Prog5In.txt, consists of (word,frequency) pairs which indicate the frequency of word use in some set of documents. We will insert this information into the priority queue. The priority is the word frequency. Note that keys with the same priority value may be deleted in different orders, depending on implementation differences.

Test your implementation to make sure the min priority queue is working.

Part 2: We have seen a min priority queue in the past, but you can imagine a max priority queue. A max priority queue is a data structure that allows at least two operations: insert which does the obvious thing and deleteMax, that finds, returns, and removes the maximum element in the priority queue.

For practice, we are going to implement the max priority queue using a different type of priority queue. The max priority queue will be implemented as a d-heap (a table which logically represents an almost complete d-ary tree) Code for a binary heap is provided in PQHeap.cpp. You just need to convert it to a d-ary heap. Use a value of d=4.

The starter code: TestPQ.cpp will allow you to read in the file and test the original binary heap code. As always, adapt the starter code as you see fit. You want to KNOW the operations are working properly (not just think it seems to be working). [You are the car mechanic. You don't want to guess things are working okay. You want to KNOW.]

Part 3: After every 100 word/frequency pairs you read in, print the word with the median frequency. Recall, a median is the value for which half of the values in the collection are bigger and half are smaller.

Dynamic-median finding. Implement a data type that supports *insert* in logarithmic time, *find the median* in constant time, and *remove the median* in logarithmic time.

Solution. Keep the median key in a variable. We'll call it currMedian. Use a max heap for values less than currMedian. Use a min heap for keys greater than currMedian. To insert a new value (call it newValue), add newValue into the appropriate heap. If the sizes of the two heaps are not within one of each other. Shift the values by deleting from the larger sized heap, making that value the median, and inserting the median into the smaller sized heap.

For simplicity, I'm only printing the frequency values in the following example:

Operation	currMedian	Values < currMedian	Values >currMedian
Add 5	5		
Add 7	5		7
Add 10	5		7 10
(shift)	7	5	10
Add 15	7	5	10 15
Add 12	7	5	10 12 15

(shift)	10	5 7	12 15
Add 8	10	5 7 8	12 15
Add 3	10	3 5 7 8	12 15
(shift)	8	3 5 7	10 12 15
Add 1	8	1 3 5 7	10 12 15
Add 2	8	1 2 3 5 7	10 12 15
(shift)	7	1 2 3 5	8 10 12 15
Add 42	7	1 2 3 5	8 10 12 15 42
Add 20	7	1 2 3 5	8 10 12 15 20 42
(shift)	8	1 2 3 5 7	10 12 15 20 42