CS29003 ALGORITHMS LABORATORY
Assignment No: 3
Last Date of Submission: 12–August–2015

Imagine a priority queue storing jobs/processes, each possessing a priority value. The heap ordering is with respect to these priority values. The queue is assumed to be dynamic, that is, jobs may join the queue at any time. A job with the highest priority is dequeued, and scheduled for execution. If the system is overloaded, there is a necessity to get rid of some job(s) without scheduling them at all. Quite naturally, we throw out those jobs which have the lowest priority. In a max-heap, it is as such difficult (linear-time) to locate a minimum (or run a general search). The basic max-heap data structure can be modified in many ways so that the minimum can be located and deleted efficiently (logarithmic-time). This assignment deals with one such idea.

Part 1: Each node in the heap now consists of two values: the priority of the node (job), and a min value which stores the minimum of all priority values stored in the subtree rooted at that node (including the priority of that node itself). Define a contiguous representation of a heap of this type. Use a static array with a maximum supported size.

Part 2: Write a function *initHeap*(Q, n) that converts an array Q of n elements to a max-heap with respect to the priority values. This function does nothing with the min values stored in the nodes. The running time should be O(n).

Part 3: Write a function *initMin*(Q, n) that, given a heap initialized by the function of Part 2, correctly populates the min values at all the nodes. The running time of this function should be O(n).

Part 4: In Parts 5–7, you write the functions for insertion and deletion in Q. After each of these operations, Q must retain its heap structure and heap ordering with respect to priority values. Moreover, the min values must be correctly restored at all the nodes. We finally demand these operations to finish in O(log n) time. If you call the naïve *initMin*() function of Part 3, you end up with O(n)-time algorithms. Fortunately, each of the insert and delete functions can be so designed that the min values at only O(log n) number of nodes are (potentially) affected. It suffices to recalculate the min values only at these nodes. To that effect, write a function *adjustMin*(Q, n, i) that takes a queue Q of size n along with an index i as input. The function recalculates the min values at all the nodes lying on the (simple) path connecting the node at index i with the root. Since any (simple) path between the root and a node is of length O(log n), this function would run in O(log n) time only. You need to figure out in the following three parts the calls of *adjustMin*() (the indices i to be precise) that you should make in order to restore the min values at (only) the affected nodes.

Part 5: Write a function *insert*(Q, n, p) that inserts a priority value p to a queue Q of (pre-insertion) size n. The function should also recalculate the min values at the nodes that are potentially affected by the insertion. The running time of this function should be O(log n).

Part 6: Write a function *deleteMax*(Q, n) that deletes the maximum from the queue Q of (pre-deletion) size n, and recalculates the min values at the affected nodes. This function should run in O(log n) time.

Part 7: Write a function *deleteMin*(Q, n) that deletes a node storing the minimum value from the queue Q of (pre-deletion) size n, and recalculates the min values at the affected nodes. This function should run in O(log n) time.

Part 8: Write a *main*() function to do the following:

- Read *ninit* from the user. Store *ninit* priority values supplied by the user in your queue Q. Call the function of Part 2 to convert this array to a max-heap with respect to the priority values. Then, call the function of Part 3 to initialize the min values at all the nodes. Print Q.
- Read *nins* from the user. The user then supplies *nins* priority values to be inserted in Q. Call the function of Part 5 for each priority value supplied by the user. Print Q after all of the *nins* insertions are made.
- Read *ndelmax* from the user. Invoke the function of Part 6 in order to make *ndelmax* maximum deletions from Q. Print Q after all these deletions are made.
- Read *ndelmin* from the user. Invoke the function of Part 7 in order to make *ndelmin* minimum deletions from Q. Print Q after all these deletions are made.

Submit a single C/C++ file solving all the parts.

Sample Output

In the sample run below, the user supplies *ninit*, *nins*, *ndelmax*, *ndelmin*, *ninit* initializing values, and *nins* priority values for insertion. The maximum and minimum values to be deleted are not supplied by the user. They are only printed in order to show which values are deleted. Each line prints the priority and min values stored in a node along with these values (inside parentheses) in its two child nodes. Underscore stands for the non-existence of the child node. Of course, making suitable jumps in the listing will allow you to retrieve the child information (Line *i* has its children in Lines 2*i* and 2*i*+1 under one-based array indexing). But having the child information adjacent to a node's information is helpful for visualizing and debugging.

```
ninit = 10
+++ Initializing queue: 40 69 50 85 84 26 74 8 3 4
+++ Queue initialized
    85,3   (84,3  : 74,26)
    84,3   (69,3  : 40,4 )
    74,26  (26,26 : 50,50)
    69,3   ( 8,8  :  3,3 )
    40,4   ( 4,4  :   _  )
    26,26  (  _   :   _  )
    50,50  (  _   :   _  )
     8,8   (  _   :   _  )
     3,3   (  _   :   _  )
     4,4   (  _   :   _  )

nins = 5
+++ Inserting elements: 18 2 47 79 26
+++ Insertions done
    85,2   (84,3  : 79,2 )
    84,3   (69,3  : 40,4 )
    79,2   (47,2  : 74,26)
    69,3   ( 8,8  :  3,3 )
    40,4   ( 4,4  : 18,18)
    47,2   ( 2,2  : 26,26)
    74,26  (50,50 : 26,26)
     8,8   (  _   :   _  )
     3,3   (  _   :   _  )
     4,4   (  _   :   _  )
    18,18  (  _   :   _  )
     2,2   (  _   :   _  )
    26,26  (  _   :   _  )
    50,50  (  _   :   _  )
    26,26  (  _   :   _  )

ndelmax = 5
+++ Deleting maximum: 85 84 79 74 69
+++ Deletions done
    50,2   (40,3  : 47,2 )
    40,3   (26,3  : 18,4 )
    47,2   ( 2,2  : 26,26)
    26,3   ( 8,8  :  3,3 )
    18,4   ( 4,4  :   _  )
     2,2   (  _   :   _  )
    26,26  (  _   :   _  )
     8,8   (  _   :   _  )
     3,3   (  _   :   _  )
     4,4   (  _   :   _  )

ndelmin = 5
+++ Deleting minimum: 2 3 4 8 18
+++ Deletions done
    50,26  (40,26 : 47,47)
    40,26  (26,26 : 26,26)
    47,47  (  _   :   _  )
    26,26  (  _   :   _  )
    26,26  (  _   :   _  )
```