CSCI 3104-Fall 2016: Paper and Pencil Assignment #3

Assigned date: Saturday 9/10/2016,
Due date: Friday, 9/16/2016, in class.
Maximum Points: 22 points + 3 for legibility

Note: This assignment *must be turned in on paper before or after class.* Please do not email: it is very hard for us to keep track of email submissions.

P1 (10 points) Acme company buys and sells oil futures over the Internet. Its trading system receives buy orders from customers that seek to buy from the company. Each order has a bidding price. Orders may be deleted by the customer at any point in time and new orders may also come in at any time.

The trading desk would like you to implement a system that keeps track of the top 100 highest buy orders at any point in time.

You decide to build a data structure that has two parts: Minheap M of the top 100 orders, and a maxheap H that stores the remaining orders. The contents of the minheap are then displayed to the trading desk, whenever demanded.

You are allowed heap operations: (a) `min`(M): returns the minimum element of a minheap M, (b) `max`(H): returns the maximum element of maxheap H, (c) `insert`(G,x): insert the element x in heap G and (d) `delete`(G,i): delete the i^{th} element $G[i]$ from heap G.

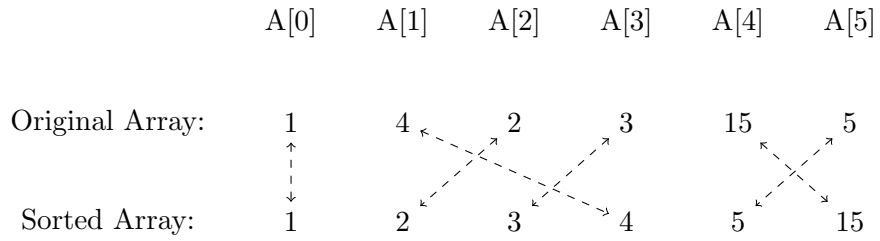
Write down pseudocode for how you would update the M and H, under the following situations. Also, write down the worst-case running time in terms of the number of orders n that are currently active. You can assume for simplicity that $n \gg 100$.

(A) A new order with price $\$B$ arrives. (**Hint:** You will first have to decide if the price is a top 100 price or not. Based on that you have to perform the appropriate heap operations).

(B) The top 100 order $M[i]$ is withdrawn by the customer.

P2 (12 points) An array is almost k sorted if every element is no more than k positions away from where it would be if the array were actually sorted in an ascending order.

As an example, here is an almost 2-sorted array.



(A) Write down pseudo code for an algorithm that sorts the original array in place in time $nk\log(k)$. Your algorithm can use a function $sort(A, l, r)$ that sorts the subarray $A[l], \ldots, A[r]$.

(B) Suppose you are allowed extra array of size $k+1$ on the side. Modify heap sort using a min heap of size $k+1$, to sort the given array in time $n\log(k)$. (**Hint:** Take the first $k+1$ elements and make a min heap. Keep extracting the minimum element from this min heap and adding a new element from the original array.).