Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Spring 2004
Homework 3b
Due on 2/10/04 (beginning of class)

1. **[20] Prioritizing your Heap!** In addition to serving as an efficient means for comparison-based sorting, heaps are also a great means for implementing a priority queue. This problem deals with a *maximum* priority queue.

   (a) Give pseudo-code for `Increase_Key(i,key)` where `i` is the index in the heap of the element who's key is being changed, and `key` is the new value. You can assume that the function is only called when `key` $\geq$ `A[i]`. Your pseudo-code should use a `loop` (as opposed to recursion).

   (b) What loop invariant is maintained throughout?

   (c) Use this invariant to argue that your algorithm is correct.

2. [**35 Points**] **Extendible Arrays Revisited.** In class we examined extendible arrays. We showed that by doubling the length of the array each time we need to perform an extension, a sequence of n insertions takes time $O(n)$. Now imagine that we allow deletions in addition to insertions. Just as an insertion always inserted an element at the end of the array, a deletion always deletes the last element in the array. You can imagine that an array may become quite large but later, due to deletions, most of the array becomes empty. In this kind of scenario, it would be nice to contract the size of the array to give the memory back to the operating system.

   Professor I. Lai of the Pasadena Institute of Technology has suggested the following rule: Use the regular doubling rule to extend arrays. However, when an array becomes less than half full (because of deletions, presumably), allocate a new array of half the length of the current array and copy the elements into the new array. (At that point the old array is released and its memory is recovered by the operating system.)

   In these analyses you should assume you start with an initially empty array.

   (a) Play the role of the "malicious adversary" and describe a sequence of n insert and delete operations for which Professor Lai's rule would incur a cost of $\Theta(n^2)$. Explain your analysis.

   (b) Professor Lai was not granted tenure and he was replaced by Professor Anna Litik. Professor Litik had a much better idea: Use the regular doubling rule to extend arrays. However, only contract an array when it becomes less than or equal to 1/4 full. At that point, allocate a new array of half the length of the current array and copy the elements into the new array. Use an amortization argument to explain why under this scheme any sequence of n inserts and deletes costs a total of $O(n)$ time. Still use 3 rubles for the insertions but now explain how much the deletions should pay and why this all works! (Recall that allocating a new block of memory can be done in constant time. However, copying one element from one array to another takes constant time. Thus, copying k elements takes k time.)

(c) Professor Litik's colleague, Professor Polly Nomial, has proposed another variant of extendible arrays: Still just double the array when it becomes full. However, we contract the array (due to deletions, presumably) only when it becomes less than or equal to 1/3 full. At that point, allocate a new array of 2/3 the length of the current array and copy the elements into that new array. Does a sequence of n inserts and deletes still cost a total of $O(n)$ time in this scheme? If so, give a clear amortization argument to explain why. If not, describe a sequence of n inserts and deletes which would cause this scheme to use more than $O(n)$ time.

3. **[10 Points] Multipop Stacks with Automatic Backup!**

   CLRS Excercise 17.2-1 (page 412).

4. **[10 Points] Sorting with Trees.** (This is a short review problem.)

   (a) Imagine that we have a binary search tree with n elements in it. Recall that an in-order traversal of the tree is a recursive traversal which takes a node v as input and does the following: Recursively traverse the left subtree of v, print out the value of v, recursively traverse the right subtree of v.

   Note that in a binary search tree, an in-order traversal beginning at the root prints out the contents of the tree in increasing order. Explain briefly why an in-order traversal of a tree with n elements takes $O(n)$ time regardless of how imbalanced the tree may be.

   (b) Professor Lai, who now teaches at the Massachusetts Institute of Typography, wants to use in-order traversals to get a fast sorting algorithm. Here is his idea: Given an unsorted array of n elements, first insert them one-at-a-time into an initially empty binary search tree. Then do an in-order traversal of the binary search tree. Each time the in-order traversal prints out a number, put that in the next available position in a new array. The new array will be the sorted version of the original array! What is the worst-case running time of Professor Lai's algorithm? Poor Professor Lai. No tenure again!

   (c) What would happen if the Professor had used a binary tree that was guaranteed in advance to always remain balanced, for example, by using one of your favorites from CS70—a Red-Black Tree, an AVL Tree, etc. (Maybe you should be getting tenure instead!)