The exam will be in class (one hour) and closed book. You may use one page of notes (one side of an 8.5×11" piece of paper, hard copy required), which will be handed in with the exam. The sums and recurrence relations tables and logarithm/exponent rules from class will be provided if needed.

It is strongly recommended that you treat this as a practice exam and work on it in a similar environment.

---

1. You need to implement an ADT characterized by the following operations:

   - `insert(x)` — insert element x
   - `replace(x,y)` — replace element x with y
   - `rank(x)` — return the number of elements larger than x
   - `successor(x)` — return the smallest element larger than x, if any

   Explain how to implement this ADT using various data structures — briefly describe how the specific operations indicated are carried out. In most cases, a sentence or so should be sufficient for each answer. You can refer to standard building blocks like "traverse", "sequential search", or "bubble up" without further explanation, but do need to explain "insert", "remove", and "find" instead of just citing them as standard operations.

   Also give the running time for each operation. It should be clear from your description of how the operation is carried out where the running time comes from so you don't need additional explanation.

   For full credit, explain how to *efficiently* implement this ADT using the specified data structure along with any additions or variations on the basic data structure (such as storing additional information) to improve the efficiency of the operations. Keep in mind the goal of an efficient implementation of the ADT (supporting all of the operations) even though you only need to describe how the operations specifically asked about.

   (a) unsorted array

      (i) `insert(x)`
      (ii) `rank(x)`

   (b) sorted array

      (i) `replace(x,y)`
      (ii) `rank(x)`
      (iii) `successor(x)`

(c) unsorted linked list

(i) `insert(x)`

(ii) `successor(x)`

(d) sorted linked list

(i) `replace(x,y)`

(ii) `successor(x)`

(e) balanced search tree (AVL or 2-4)

(i) `replace(x,y)`

(ii) `rank(x)`

(iii) `successor(x)`

(f) heap

(i) `replace(x,y)`

(ii) `rank(x)`

(iii) `successor(x)`

(g) hashtable (separate chaining or open addressing)

(i) `replace(x,y)`

(ii) `rank(x)`

(iii) `successor(x)`

2. Design a data structure which holds a collection of elements, each of which has a key and a numeric value. It should efficiently support the following operations:

   - `insert(k,v)` — insert the value v associated with the key k
   - `replace(k,v)` — replace the value associated with key k with v
   - `getValue(k)` — return the value associated with key k
   - `getMinValue()` — return the smallest value in the collection
   - `add(k,a)` — add a to the value associated with key k
   - `addAll(a)` — add a to all of the values

   Give an efficient implementation for this ADT. No operation should have a running time worse than $O(\log n)$ in the worst case, though for full credit you should obtain $O(1)$ time where possible.

   Describe how the elements are stored in your data structure and how each of the operations are carried out. You can refer to standard operations like "insert into a balanced binary search tree" without further explanation, but in such a case you would still need to explain how the elements are arranged in the BST and any additions or variations to the standard insert operation.

   Also give the running time for each operation. It should be clear from your description of how the operation is carried out where the running time comes from so you don't need additional explanation.