CSC265 Fall 2008 Homework Assignment 1

due Thursday, October 2, 2008, 10:10am

1. (a) Give a recurrence that exactly describes the expected number of element comparisons performed by QUICKSORT on an input array of size $n \geq 1$ when every element of the input array is uniformly and independently chosen from $\{1, \ldots, m\}$ where m is a positive integer. Justify why your recurrence is correct.

   (b) Compute a numerical solution to your recurrence in (a) for input array size $n = 4$, where every element is uniformly and independently chosen from $\{1, \ldots, 4\}$. Compare your answer to the expected number of element comparisons when the sample space consists of all permutations of $\{1, \ldots, 4\}$ and each is equally likely (i.e. the probability distribution considered in class for $n = 4$).

2. Consider the following algorithm to merge two linked lists that are in nondecreasing order into one list containing all of the elements arranged in nondecreasing order.
   Each list element consists of a value and a pointer to the next element in the list.
   The variables a and b point to the beginning of the input lists, c is a pointer to the beginning of the output list and d is a pointer to the end of the output list.

```
if a = nil then c ← b
else if b = nil then c ← a
else %Create a list with one dummy element.
    c ← (0, nil)
    d ← c
    loop
        if a.value < b.value
        then %Move the first element of a's list to the end of the output list.
                d.next ← a
                d ← a
                a ← a.next
                d.next ← nil
                if a = nil
                then %Concatenate the rest of b's list to the end of the output list.
                        d.next ← b
                        exit loop
        else %Move the first element of b's list to the end of the output list.
                d.next ← b
                d ← b
                b ← b.next
                d.next ← nil
                if b = nil
                then %Concatenate the rest of a's list to the end of the output list.
                        d.next ← a
                        exit loop
    end loop
%Remove the dummy element from the beginning of the list.
c ← c.next
```

   Suppose that the lists a and b are disjoint and together they contain the values $1, 2, \ldots, n$. Neither list contains a duplicate element. Suppose that each element is equally likely to be in either list.

   (a) Assume that which list each element is in is independent of which lists the other elements are in. In other words, 1 is equally likely to be in either a or b, 2 is equally likely to be in a or b

independent of which list 1 is in, 3 is equally likely to be in a or b independent of which list 1 and 2 are in, etc. What is the expected number of operations (comparisons and assignments) performed by this algorithm? Justify your answer.

(b) Suppose that list a has $\lceil n/2 \rceil$ elements, which are equally likely to be any of the elements of $\{1, 2, \ldots, n\}$. What is the expected number of operations (comparisons and assignments) performed by this algorithm? Justify your answer.

Compute your answer exactly. It is not sufficient to get answers within constant factors.
Be sure to formally define your sample space, probability distribution, and any necessary random variables for each part.

3. A *half-height-balanced tree* is a leaf-oriented binary search tree (i.e. a strict binary tree with all dictionary elements stored at leaves) with the following properties:

   - Each internal node v stores
     - a pointer $p(v)$ to its parent or nil (if it is the root),
     - a pointer $r(v)$ to the rightmost leaf in its left subtree,
     - $l(v)$ = the length of the longest path from v down to a leaf, and
     - $s(v)$ = the length of the shortest path from v down to a leaf.
   - Each leaf node stores one dictionary element and a pointer to its parent or nil (if it is the root).
   - $l(v) \leq 2s(v)$ for all internal nodes v.

   (a) Prove that a half-height-balanced tree with n leaves has $O(\log n)$ height.

   (b) Give an algorithm to INSERT a new element into a half-height-balanced tree, given a pointer to the root of the tree. Explain why your algorithm is correct. How many rotations does your algorithm perform, in the worst case?