CS 473U: Undergraduate Algorithms, Fall 2006
Homework 3

Due Wednesday, October 4, 2006 in 3229 Siebel Center

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Consider a perfect tree of height h, where every non-leaf node has 3 children. (Therefore, each of the 3^h leaves is at distance h from the root.) Every leaf has a boolean value associated with it - either 0 or 1. Every internal node gets the boolean value assigned to the majority of its children. Given the values assigned to the leaves, we want to find an algorithm that computes the value (0 or 1) of the root.

   It is not hard to find a (deterministic) algorithm that looks at every leaf and correctly determines the value of the root, but this takes $O(3^h)$ time. Describe and analyze a *randomized* algorithm that, on average, looks at asymptotically fewer leaves. That is, the expected number of leaves your algorithm examines should be $o(3^h)$.

2. We define a *meldable heap* to be a binary tree of elements, each of which has a priority, such that the priority of any node is less than the priority of its parent. (Note that the heap does **not** have to be balanced, and that the element with greatest priority is the root.) We also define the priority of a heap to be the priority of its root.

   The *meld* operation takes as input two (meldable) heaps and returns a single meldable heap H that contains all the elements of both input heaps. We define *meld* as follows:

   - Let H_1 be the input heap with greater priority, and H_2 the input heap with lower priority. (That is, the priority of $root(H_1)$ is greater than the priority of $root(H_2)$.) Let H_L be the left subtree of $root(H_1)$ and H_R be the right subtree of $root(H_1)$.
   - We set $root(H) = root(H_1)$.
   - We now flip a coin that comes up either "Left" or "Right" with equal probability.
     - If it comes up "Left", we set the left subtree of $root(H)$ to be H_L, and the right subtree of $root(H)$ to be $meld(H_R, H_2)$ (defined recursively).
     - If the coin comes up "Right", we set the right subtree of $root(H)$ to be H_R, and the left subtree of $root(H)$ to be $meld(H_L, H_2)$.
   - As a base case, melding any heap H_1 with an empty heap gives H_1.

   (a) Analyze the expected running time of $meld(H_a, H_b)$ if H_a is a (meldable) heap with n elements, and H_b is a (meldable) heap with m elements.

   (b) Describe how to perform each of the following operations using only melds, and give the running time of each.

   - $DeleteMax(H)$, which deletes the element with greatest priority.
   - $Insert(H, x)$, which inserts the element x into the heap H.
   - $Delete(H, x)$, which - given a pointer to element x in heap H - returns the heap with x deleted.

3. Randomized Selection. Given an (unsorted) array of n distinct elements and an integer k, SELECTION is the problem of finding the kth smallest element in the array. One easy solution is to sort the array in increasing order, and then look up the kth entry, but this takes $\Theta(n \log n)$ time. The randomized algorithm below attempts to do better, at least on average.

```
QuickSelect(Array A, n, k)
pivot ← Random(1, n)
S ← {x | x ∈ A, x < A[pivot]}
s ← |S|
L ← {x | x ∈ A, x > A[pivot]}
if (k = s + 1)
   return A[pivot]
else if (k ≤ s)
   return QuickSelect(S, s, k)
else
   return QuickSelect(L, n − (s + 1), k − (s + 1))
```

Here we assume that Random(a, b) returns an integer chosen uniformly at random from a to b (inclusive of a and b). The pivot position is randomly chosen; S is the set of elements smaller than the pivot element, and L the set of elements larger than the pivot. The sets S and L are found by comparing every other element of A to the pivot. We partition the elements into these two 'halves', and recurse on the appropriate half.

(a) Write a recurrence relation for the expected running time of QuickSelect.

(b) Given any two elements $x, y \in A$, what is the probability that x and y will be compared?

(c) Either from part (a) or part (b), find the expected running time of QuickSelect.

4. **[Extra Credit]**: In the previous problem, we found a $\Theta(n)$ algorithm for selecting the kth smallest element, but the constant hidden in the $\Theta(\cdot)$ notation is somewhat large. It is easy to find the *smallest* element using at most n comparisons; we would like to be able to extend this to larger k. Can you find a randomized algorithm that uses $n + \Theta(k \log k \log n)$[1] expected comparisons? (Note that there is no constant multiplying the n.)

Hint: While scanning through a random permutation of n elements, how many times does the smallest element seen so far change? (See HBS 0.) How many times does the kth smallest element so far change?

[1] There is an algorithm that uses $n + \Theta(k \log(n/k)$ comparisons, but this is even harder.