CIS 313 Intermediate Data Structures
Winter 2010

Extra Credit

The following are some slightly more challenging problems you might enjoy trying. We will assign some extra credit for these. This extra credit will be added to your score after the grade curve is determined - in this way there is no penalty for not doing any.

You may turn in a problem whenever you choose (note the first Note, however). You do not need to turn them in together. They will be graded strictly. Do not guess at a solution - you will receive no partial credit unless you are very close to being correct. It is possible that it will be returned to you ungraded for further elaboration, which you can then give for reconsideration (at no penalty).

These will be worth a varying number of points, depending on how hard the instructor feels they are and how much work you do. For example, the second one would be usually worth two points but if you generalize the problem you could get even more. In general, an extra credit point is worth more than a point on a homework.

Note 1: Only two extra credit problems per person will be accepted during dead week. None will be accepted during finals week. However, resubmissions of previously submitted problems are not subject to these constraints.

Note 2: This page may be updated throughout the term, so check occasionally for further problems.

- On the first homework, you were asked to find the min and the max of n elements using $\frac{3}{2}n + O(1)$ element comparisons. Show that this is the best possible - no algorithm which successfully finds the min and max of n elements can run in time $cn + o(n)$ for any $c < \frac{3}{2}$. (This might be kind of difficult compared to the others.)
- Here are two strange looking recurrence relations which can be massaged into a form that you will have seen before. Your goal is to solve them.
  - Solve $T(n) = 8T(\sqrt{n}) + \lg n$. Hint: Let $S(n) = T(2^n)$, derive a recurrence relation for S, solve it, and back-substitute to get a solution for T.
  - Solve $T(n) = 3T(n-1) + n2^n$. Hint: let $S(n) = T(\lg n)$.
- On one of the homeworks you were asked to simulate a stack with two queues. More interesting is to simulate a queue with two stacks. This can be done more efficiently - $\overline{\text{any}}$ sequence of n *enqueue* and *dequeue* operations can be performed in a total of $O(n)$ time. Explain how. **NOTE:** This problem was covered in CIS 323, so it is no longer available for extra credit. However, it's still a really nice problem!
- Explain how to find the *second* largest of a list of n elements using at most $n+\lg n$ comparisons. (*Hint:* run a tournament as a complete binary tree.)

- Let T be a binary tree. It is not necessarily full - there may be nodes with 0, 1, and 2 children - and in this context we consider a *leaf* to be a node with no children. Let nodes $v_1, v_2, \ldots, v_k$ be all the leaves, and for each i let $l_i = \text{depth}(v_i)$. Prove that

  $$\sum_{i=1}^{k} 2^{-l_i} \leq 1.$$

  This is known as *Kraft's Inequality.*

- Actually, Kraft's Inequality is more general and refers to codes. The previous question is an application to binary trees, and is only one direction (of an "if an only if"). For this question, show the following converse to the previous problem: If we have integers $l_1, l_2, \ldots, l_k$ such that

  $$\sum_{i=1}^{k} 2^{-l_i} \leq 1,$$

  then there is a binary tree T with leaves $v_1, v_2, \ldots, v_k$ such that

  $$\forall i,\ l_i = \text{depth}(v_i).$$

- Recall that the *internal path length* I of a binary (search) tree T is defined as the sum of the depth of all nodes of T. If T has n nodes, the average depth of a node in T is I/n. Describe a family of trees of n nodes of depth $\omega(\lg n)$ with $I/n = \Theta(\lg n)$.

- You saw on a homework that there is a tree that can be colored as a red-black tree which is not an AVL tree. Now prove that *any* AVL tree can be colored as a red-black tree.

- **(2-3-4 heaps)** exercise 19-4, pp 529-530

- Suppose we have a list of n items in an array, but these items have no order. For example, the array may be pointers to a jpeg file. However, we can test whether two items are equal - that is, for any i and j, we can use a routine `equal(A[i],A[j])`, which returns true or false. The problem is to determine whether some item appears in the list in a majority of the locations ($> \frac{n}{2}$). Show how to do this with $O(n)$ calls to the "equal" routine.

- Let A be an $n \times n$ array of integers. We say that A has a *local minimum* at (i, j) if $A[i, j]$ is less than each of its neighbors (above, below, left, and right). A local minimum can appear on the boundary of the array, it just needs to be less than all the neighbors that exist in the array. Describe an $O(n)$ time algorithm to test whether A has a local minimum, and if so, return one.

  Note: This sounds deceptively easy. It's not. The only solution I have seen is very very weird.