CS 373: Combinatorial Algorithms, Spring 2001

Homework 2 (due Thu. Feb. 15, 2001 at 11:59 PM)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates. For full credit, your algorithm should run in $\Theta(\log n)$ time. *[Hint: First try to solve the special case $k = n$.]*

2. Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

   (a) Show that a rotation in an augmented binary tree can be performed in constant time.

   (b) Describe an algorithm ScapegoatSelect(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time.

   (c) Describe an algorithm SplaySelect(k) that selects the kth smallest item in an augmented splay tree in $O(\log n)$ *amortized* time.

(d) Describe an algorithm TreapSelect(k) that selects the kth smallest item in an augmented treap in $O(\log n)$ *expected* time.

3. (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.

   (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree. (Recall that $I(v) = \max\{0, |T| - |s| - 1\}$, where T is the child of greater height and s the child of lesser height, and $|v|$ is the number of nodes in subtree v. A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)

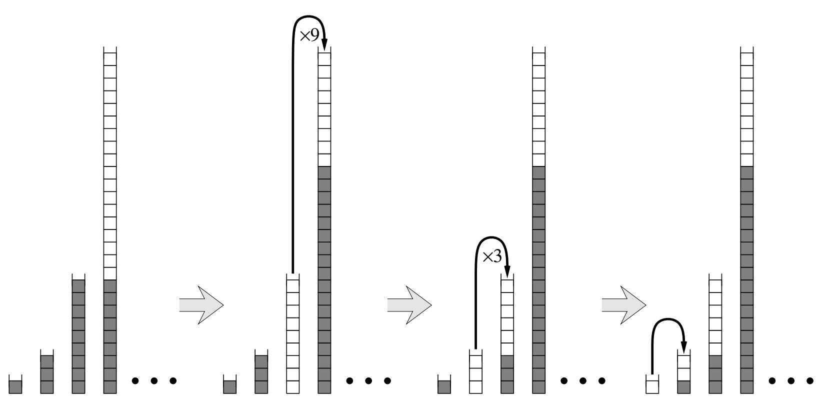
   *(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in $O(n)$ time using only $O(\log n)$ additional memory.

4. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

   - After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
   - After a deletion, if the table is less than 1/4 full, we we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

   Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (it makes it much more difficult).

5. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first move all the elements in S_i to stack S_{i+1} to make room. But if S_{i+1} is already full, we first move all its members to S_{i+2}, and so on. Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

   (a) *[1 point]* In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) *[9 points]* Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

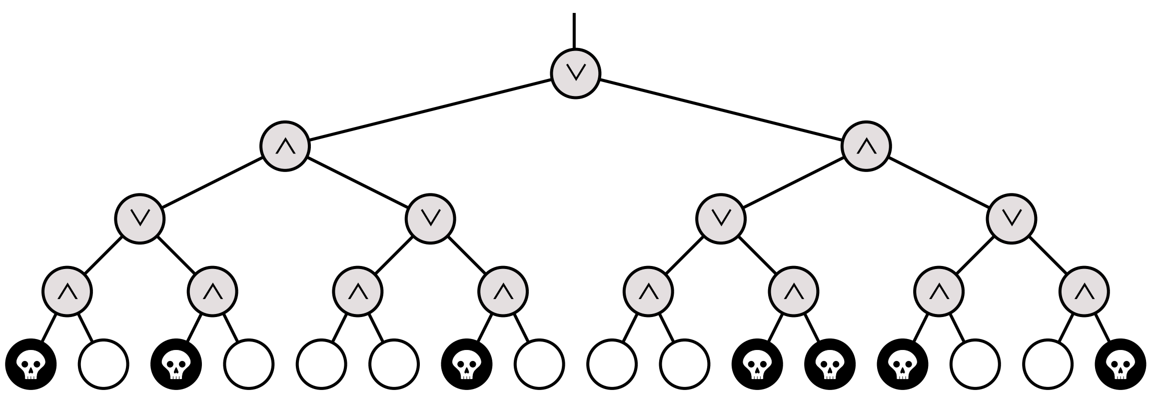
6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

   You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are *and* gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) (2 pts) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

   (b) (8 pts) Unfortunately, Death won't let you even look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $\Theta(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*



Practice Problems

1. (a) Show that it is possible to transform any n-node binary search tree into any other n-node binary search tree using at most $2n-2$ rotations.

   *(b) Use fewer than $2n-2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n-6$ rotations, and there are pairs of trees that are $2n-10$ rotations apart. These are the best bounds known.

2. Faster Longest Increasing Subsequence(LIS)
   Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. [Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an $O(n^2)$ algorithm for this. If you are having difficulty, look at the solution provided in the HW 1 solutions.]

3. Amortization

   (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

   [Hint: Suppose p is the number of significant bits in P, and n is the number of significant bits in N. For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p-n$ always has the same sign as $P-N$. Assume you can update p and n in $O(1)$ time.]

   *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*4. Amortization
   Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of 'fits', where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

5. Detecting overlap

   (a) You are given a list of ranges represented by min and max (*e.g.*, [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$-time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

   (b) You are given a list of rectangles represented by min and max x- and y-coordinates. Give an $O(n \log n)$-time algorithm that decides whet her or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

6. Comparison of Amortized Analysis Methods
   A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

   Determine the amortized cost per operation using the following methods of analysis:

   (a) Aggregate method

   (b) Accounting method

   *(c) Potential method