CS 373: Combinatorial Algorithms, Fall 2000

Homework 2 (due September 28, 2000 at midnight)

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. Faster Longest Increasing Subsequence (15 pts)
   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 HW 1 solutions.]*

2. SELECT(A, k) (10 pts)
   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.

[Hint: The answers for (b), (c), and (d) are almost exactly the same!]

3. Scapegoat trees (15 pts)

   (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. For 5 extra credit points, use only $O(1)$ additional memory.

4. Memory Management (10 pts)
   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 the problem much too hard!

5. Fibonacci Heaps: SECONDMIN (10 pts)

   (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.

   *(b) Modify the Fibonacci Heap data structure to implement the SECONDMIN operation in constant time, without degrading the performance of any other Fibonacci heap operation.

Practice Problems

1. 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.

*2. 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!]*

3. Rotations

   (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.

4. Give an efficient implementation of the operation CHANGEKEY(x, k), which changes the key of a node x in a Fibonacci heap to the value k. The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $key[x]$.

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