

Homework 5, CS420, Fall 2011

Due Friday 11/4; the problem on minimum spanning trees has been postponed to the next homework

1. For the $O(n)$ worst-case selection algorithm, we divided the elements into groups of five, found the medians of the groups, found the medians of these medians with a recursive call, and then used this element as the pivot. We found the time bound by getting a recurrence with unequal-sized subproblems, and showed that this expanded to a geometric series whose largest element was n .

What time bound would we get if we divided the elements into groups of 3 instead of groups of 5? How about groups of 7?

2. Here is a variant on the binomial heap.
 - (a) We can replace the binomial tree as follows. Our tree of rank 0 is again a single node. The one for rank k is obtained from three of rank $k - 1$ by making two of them be the leftmost children of the third. Draw the trees of ranks 1, 2, and 3.
 - (b) Why is it that for any n , we can have a collection of these trees that has a total of n nodes, and no more than two trees of any rank?
 - (c) Let our heap therefore consist of a collection of these trees, with at most two of any rank, each node containing one key of the priority queue, and the keys in each tree observing the heap property. Describe how to perform a meld operation by relating it to addition base-3. Justify that it still takes $O(\log n)$ time.
 - (d) Prove by induction that if you remove the root of a tree of rank k , you have exactly two trees of each rank from 0 through $k - 1$. Use the book's proof of the analogous property for binomial trees as a guide.
 - (e) In English, describe how to perform the extract-min operation, and argue that it takes $O(\log n)$ time.

3. **Postponed to next assignment** Do problem 19-2 from the binomial-heap handout. Here is what they are hinting at: you need a way to figure out quickly which sets u and v belong to. Here's a nice trick: have each set V_i be labeled the number of vertices in it, and have each vertex in it labeled with i . When you want to merge V_i and V_j , if $|V_i| < |V_j|$ swap their roles.

That way, the vertices in the smaller of the two sets get relabeled. Each time a vertex is relabeled, it finds itself in a set that's at least twice as large as the one it was in before the merge that caused it to get relabeled. It never finds itself in a set larger than n , so the total number of times a vertex can be relabeled is $O(\log n)$. The total time spent relabeling the n vertices is therefore $O(n \log n)$.

Make sure you understand this trick. Then derive the time bound for the algorithm. You don't need to repeat the explanation of the trick; you can just reference the result.

4. Leftist heaps do not have $O(\log n)$ height. Therefore, if we want to obtain an $O(\log n)$ bound for **Reduce-Key**, we can't afford to reduce the key and bubble it up, since the path we bubble it up on might be too long.

Here is an alternative. Remove the subtree rooted at the node v containing the key. The removed tree is a leftist heap. Reduce the key. We don't need to bubble it up since it is already at the root v of this tree. Then meld this heap back into the original, which takes $O(\log n)$ time.

There is a step that is missing from this description. What is it? Think about the properties of the tree that you have to maintain as invariants. See if you can figure this out before looking at the answer on the next page.

You have to fix up the leftist property in the original tree when you remove the subtree. The leftist property can only be messed up at ancestors of the removed root. Luckily, it can only be messed up at those ancestors where the shortest path to a null pointer is the path to the null pointer left by the removal of v . Explain why only the lowest $O(\log n)$ ancestors can be messed up. (If you can show this, then it follows that if you include parent pointers in your leftist heap, you can fix up the leftist property at the messed-up ancestors in $O(\log n)$ time.)

5. Show how to build a binomial heap or a leftist heap in $O(n)$ time. *Hint: start with each element in its own heap. The master theorem can be useful.*