

CS 535 Homework 3

Due: February 16 (Th), in class.

Undergraduate students, answer problems 1, 2, 3. Problem 4 is a bonus problem. Graduate students, answer all problems.

1. Let L_1 and L_2 be two LIST ADT's whose elements are distinct and sorted in increasing order. For example, each L_i can contain a list of students, and the students are sorted according to their student ID number. Assume that a doubly linked list was used to implement a LIST ADT.
 - a. Suppose that L_1 and L_2 contain n_1 and n_2 elements respectively. Describe an $O(n_1 + n_2)$ -time algorithm that combines the elements in L_1 and L_2 into one LIST ADT *where the elements are still in sorted order*.
 - b. Let us call the procedure described in part (a) $\text{UNION}(L_1, L_2)$. Suppose we now have k LIST ADT's L_1, L_2, \dots, L_k whose elements are pairwise distinct and sorted in increasing order. We again want to combine all of them into a single LIST ADT so that the elements are still in sorted order. Here's a simple procedure for doing this: First, take the union of L_1 and L_2 . Then take the union of L_3 and the list of $L_1 \cup L_2$. Then take the union of L_4 and the list of $L_1 \cup L_2 \cup L_3$, etc.
 - b1. Write the above procedure in pseudocode.
 - b2. Assuming that L_1, \dots, L_k all have n elements, what is the running time of this procedure? Your answer should be in terms of n and k .
2. Suppose T is a BinaryTree ADT. Assume, as the book does, that T is a *proper* binary tree. That is, if v is an internal node, it has both a left and a right child. Design an algorithm for the following operations:

inorderBefore(v): returns the node visited before v in an inorder traversal of T .

inorderNext(v): returns the node visited after v in an inorder traversal of T .

What is the running time of your algorithm?

Note that calling an inorder traversal on T to implement the above operations is *not* a legitimate answer. The goal here is for you to think about the location of the nodes that are visited before and after v in relation to v .
3. C-2.15.
4. This problem is a continuation of problem 1b. Here's a "tournament" way of creating the list that combines L_1, L_2, \dots, L_k . Think of it in terms of rounds. At the beginning of round i , suppose there are k_i LIST ADT's whose elements are pairwise distinct and sorted in increasing order: J_1, J_2, \dots, J_{k_i} . For $i = 1, \dots, \lfloor k_i/2 \rfloor$, take the union of J_{2i-1} and J_{2i} . The resulting list moves to the next round. If k_i is odd, J_{k_i} simply moves to the next round. Stop when there is only one list left.

a: Write the above procedure in pseudocode.

b: Assume that L_1, \dots, L_k all have n elements. What is the running time of this procedure. Again, your answer should be in terms of n and k .

(Hint: To analyze the running time of this procedure, consider how much time is spent at each round and then add them up. Keep in mind that while the lists have size n at the beginning of round 1, the lists get longer as the tournament progresses.)

c. Of the two procedures we have described, which one is more efficient? Can you provide an intuitive reason as to why this is the case?