CS 261L	**Latex Homework5**
Fall 2022	**Due:** Week 5

Roll Number: 2021-CS-33

1-1 Comparison of running times

For each function f(n)and time t in the following table, determine the largest size n of a problem that can be solved in time t , assuming that the algorithm to solve the problem takes f(n) microseconds.

4-6 Monge arrays

An $m \times$n array A of real numbers is a **Monge** array if for all i , j , k, and l such that
$1 \leq$i $<$ $k \leq$m and $1 \leq$j $<$ $l \leq$n, we have
A[i,j]+E[k,l] $\leq A[i, l] + E[k, j]$

In other words, whenever we pick two rows and two columns of aMonge array and consider the four elements at the intersections of the rows and the columns, the sum of the upper-left and lower-right elements is less than or equal to the sum of the lower-left and upper-right elements. For example, the following array is Monge:

10	17	13	28	23
17	22	16	29	23
24	28	22	34	24
11	13	6	17	7
45	44	32	37	23
36	33	19	21	6
75	66	51	53	24

a. Prove that an array is Monge if and only if for all i = 1,2,... ,m - 1 and j = 1,2,... , n - 1, we have

A[i,j]+E[k,l] $\leq A[i, l] + E[k, j]$

(Hint: For the "if" part, use induction separately on rows and columns.)

b. The following array is not Monge. Change one element in order to make it Monge. (Hint: Use part (a).)

37	23	22	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

c. Let f(i) be the index of the column containing the leftmost minimum element of row i . Prove that $f(1) \leq$f (2) $\leq ... \leq$f(m) forany $m\times$n $Monge$array.

d. Here is a description of a divide-and-conquer algorithm that computes the leftmost minimum element in each row of an $m \leq$n Monge array A: Construct a submatrix A of A consisting of the even-numbered rows of $A\prime$ Recursively determine the leftmost minimum for each row of A0. Then compute the leftmost minimum in the odd-numbered rows of $A\prime$. Explain how to compute the leftmost minimum in the odd-numbered rows of A (given that the leftmost minimum of the even-numbered rows is known) in O(m+ n) time.

e. Write the recurrence describing the running time of the algorithm described in part (d). Show that its solution is O(m + n) log m.

13-1 Persistent dynamic sets

During the course of an algorithm, we sometimes find that we need to maintain past versions of a dynamic set as it is updated. We call such a set persistent. Oneway to implement a persistent set is to copy the entire set whenever it is modified, but this approach can slow down a program and also consume much space. Sometimes, we can do much better.

Consider a persistent set S with the operations INSERT, DELETE, and SEARCH, which we implement using binary search trees as shown in Figure 13.8(a). We maintain a separate root for every version of the set. In order to insert the key 5 into the set, we create a new node with key 5. This node becomes the left child of a new node with key 7, since we cannot modify the existing node with key 7. Similarly, the new node with key 7 becomes the left child of a new node with key 8 whose right child is the existing node with key 10. The new node with key 8 becomes, in turn, the right child of a new root r0 with key 4 whose left child is the existing node with key 3. We thus copy only part of the tree and share some of the nodes with the original tree, as shown in Figure 13.8(b).

Assume that each tree node has the attributes key, left, and right but no parent. (See also Exercise 13.3-6.)

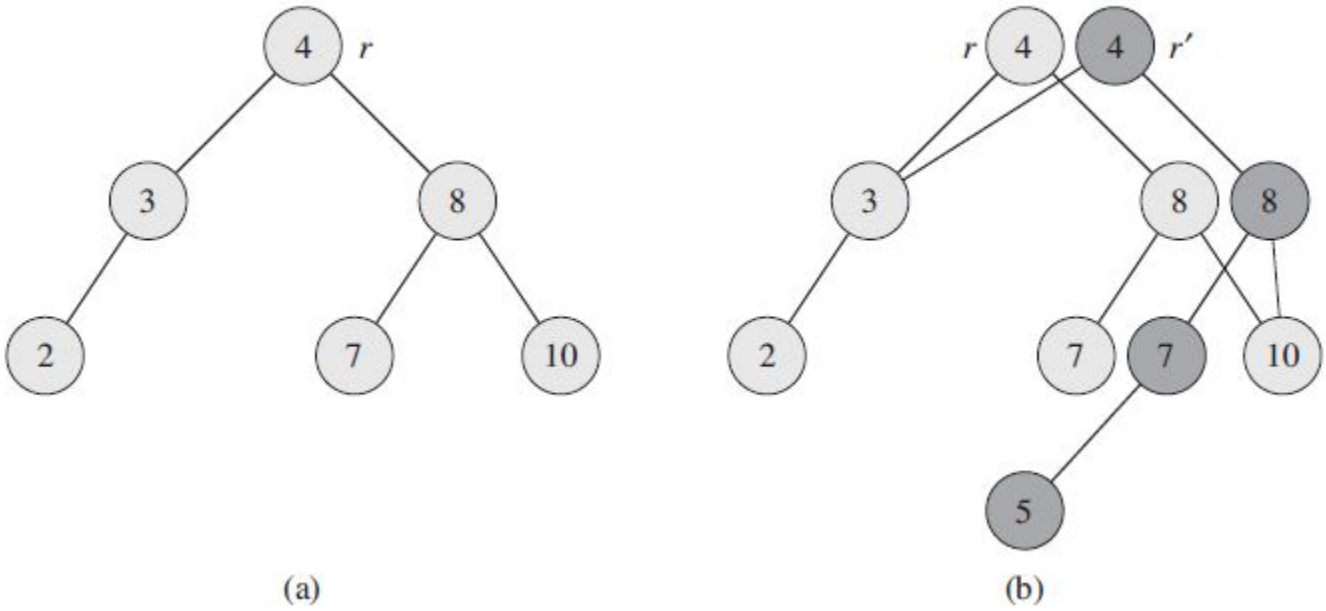

Figure 13.8 (a) A binary search tree with keys 2; 3; 4; 7; 8; 10. **b**. The persistent binary search tree that results from the insertion of key 5. The most recent version of the set consists of the nodes reachable from the root r0, and the previous version consists of the nodes reachable from r. Heavily shaded nodes are added when key 5 is inserted.

a. For a general persistent binary search tree, identify the nodes that we need to change to insert a key k or delete a node y.

b. Write a procedure PERSISTENT-TREE-INSERT that, given a persistent tree T and a key k to insert, returns a new persistent tree T 0 that is the result of inserting k into T .

c. If the height of the persistent binary search tree T is h, what are the time and space requirements of your implementation of PERSISTENT-TREE-INSERT? (The space requirement is proportional to the number of new nodes allocated.)

d. Suppose that we had included the parent attribute in each node. In this case, PERSISTENT-TREE-INSERT would need to perform additional copying. Prove that PERSISTENT-TREE-INSERT would then require .n/ time and space, where n is the number of nodes in the tree.

e. Show how to use red-black trees to guarantee that the worst-case running time and space are O.lg n/ per insertion or deletion.