Practice Problem Set 8

.

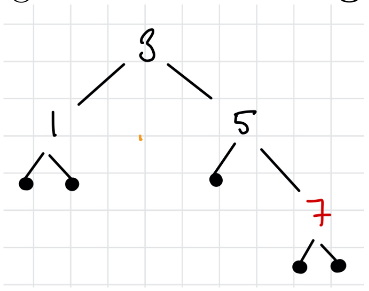
Problem 1

Construct an interval tree by inserting the following intervals in the order given below. Assume that the tree structure is a red-black tree. Show the steps of your insertions

$$[7,10]\,,[12,15]\,,[13,14]\,,[5,12]\,,[2,10]\,,[4,20]$$

Problem 2

Below is an example of a red-black tree. Determine the black height of the tree. What is the maximum number of inserts you can carry out on the this tree such that the black height **does not change**? Demonstrate an example illustrating the inserts.



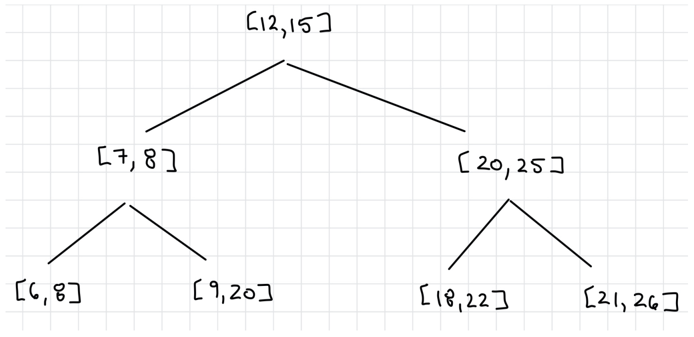
Problem 3.

Suppose A is an array consisting of n distinct numbers, sorted in increasing order. Write the pseudo-code for the algorithm Build-BST(A,s,f) which takes as input array A and start and finish indices, s and f. The algorithm must return a pointer/reference the resulting BST which has height $\Theta(\log n)$. You may assume you can construct a new node using simple line of pseudocode like $x = NewTreeNode$ and then define x's attributes. Your implementation must include parent pointers. Justify why your algorithm runs in time $O(n)$.

Problem 4

Below is an example of a binary search tree that is built using $x.int.left$ as the search key. Currently, the values of $x.max$ are not set for any of the nodes. Write the pseudo-code for an algorithm called AssignMax(T) which takes as input a reference to a node of this tree, and correctly assigns the maximum attribute for each node in the tree. Justify the runtime of $O(n)$.

Next, given an example of an interval i for which the interval search algorithm searches in the left subtree, *even though* there is an interval that overlaps with i in the right subtree.



Problem 5

How would you update the interval-search algorithm so that it returns the interval with the minimum lower endpoint? Justify the runtime of your new algorithm.

Problem 6

The algorithm in class for *BST-Select(k,x)* was written iteratively. Re-write the algorithm recursively. It should have the same runtime. Repeat for *Rank(t,x)*.

Problem 7

Given a binary search tree T and a key a that is known to be in the tree, describe an algorithm that outputs **all keys** in T that are **larger** than a. Do *not assume* the tree is augmented with subtree sizes! Write the pseudo-code for your algorithm and justify its runtime.

Problem 8:

Suppose a binary search tree contains n nodes (the n nodes are the actual nodes, excluding the NIL nodes). The tree is currently not augmented with any additional information. We now wish update this tree so that it is augmented with subtree sizes. Write the pseudo-code for an $O(n)$ algorithm that adds the subtree size to each node of the tree. Justify the run in time $O(n)$.

Problem 9:

Suppose tree T is a binary search tree that is augmented with subtree sizes. The n keys in T are distinct numbers. Describe an algorithm that will find the pair of nodes x, y in T for which $|x.key - y.key|$ is minimized. This value is called the *gap* between they keys of x and y. The algorithm must run in time $O(n)$.

Problem 10:

A project manager would like to store a set of n project intervals. Each interval consists of a start and end time (over a year-long period). The manager would like a data structure that organizes the project intervals in such a way that she can carry out the following operations:

1. Given a new project interval, i, determine if project interval i overlaps with any of the current project intervals. If not, insert the project i.

2. Given a project interval x that is currently in the list of projects, determine how many other projects will start before project x

3. Given a new project interval i, return the earliest project from the current list that overlaps with i. For example, if the project list contains intervals $(2, 6)$ and $(4, 10)$ and $i = (3, 5)$, then i overlaps with both projects in the system, but the earliest of the two is interval $(2, 6)$.

4. Given a project interval x that is currently in the list of projects, determine the next project to start immediately after project x starts.

You must describe what data structure you use to maintain this information, and justify the construction time of $O(nlogn)$. You must describe how to carry out each of the above operations in $O(logn)$ time.

Problem 11:

- Write an algorithm called FindHeight(T) which returns the height of the BST tree T.

- Use the above algorithm to solve the following problem:

  Given a *complete* binary search tree, how could you color the nodes of such a tree so that the result is a valid red-black tree? Take this coloring idea and convert it into an algorithm called ColorTree(T,h) which takes as input a reference to a binary search tree T of height h and properly assigns the $x.color$ attributes so that the result is a red-black tree. Your algorithm must also add NIL nodes where necessary.

Problem 12:

The red-black tree implementation from class augments each node x of the tree with an attribute $x.color$ which was either red or black. No attribute for the black height of each node is actually used! Your job is to develop an algorithm that augments a red-black tree with the attribute x.bh, which is set to the black height of node x. Write the pseudo-code for a recursive algorithm called AssignBH(T) which takes as input a reference T to the root of a red-black tree, and correctly sets the value of $x.bh$ for each node in the tree. Your algorithm must also return the black height of the tree rooted as T. You must justify the $O(n)$ runtime of your algorithm. You may assume that NIL nodes can be identified as x.isNIL = true and that $x.bh = 0$ for NIL nodes.

Problem 13:

Update the pseudo-code for TREE-INSERT(T,z) (from the lecture notes on BST) assuming the reference T is now a reference to a BST that is *augmented with subtree sizes*. As new node z is inserted into the tree, the attribute for *size* must be properly updated through the tree.

Problem 14

Show how up update the Rank algorithm from class so that it outputs the number of keys that are *smaller than or equal* than k. Call your new algorithm KeyRank(T, k). Note that in this update, k is *not* a tree node, but it instead a value that is comparable to the keys of the BST.