CS 315 Week 10 (April 8 and 10) summary and review questions

Topics covered

- Binary trees – expression trees, converting expression to expression tree
- Tree traversals – inorder, preorder and postorder traversals
- Binary search trees – motivation, review of dictionary operations, algorithm for binary search in an array
- Algorithm for search and insert

Review questions:

1) Exhibit the binary search tree that results by inserting the following sequence of keys: 8, 11, 3, 12, 35, 8, 14, 23, 1.

2) What is the height of the above tree? What is the depth of the node containing the key 14?

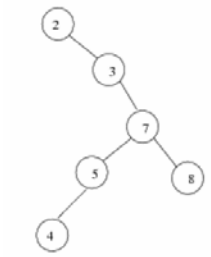
3) There are 24 possible ways in which we could insert the keys 1, 2, 3 and 4 into a (initially empty) binary search tree. Of these, how many would result in a tree of height = 3? (For example, the sequence 1, 4, 3, 2 will produce such a tree.) Answer the same question for the collection {1, 2, 3} and {1, 2}. Can you guess a general formula for this number?

4) Exhibit the expression tree corresponding to the expression: ((a + b * c))* d + e * (f + g)

5) Preorder traversal of a binary search tree produces the sequence 5 3 18 13 9 14 21 24. Construct the tree.

6) * Given a pointer to a node N in a binary search tree and a key x, you are to write a procedure that finds outputs the node in the subtree rooted at N that contains the smallest key larger than x. If such a node does not exist, it should a NULL pointer.

Example: Consider the binary search tree shown below. If n is the node containing 7, and x is 4 (or 4.7), the output should be the node containing 5. If x = 8, NULL pointer should be returned.



7) The following recursive procedure takes as input a binary tree node T, and answers true (false) if T represents (does not represent) a binary search tree. Determine if the procedure is correct. If it is correct, prove that it is correct. If not, construct an input for which it fails.

```
boolean checkTree ( TreePtr T)

{

  if ((T == null) || (T -> left == null && T->right == null))

     return true;

  else if (checkTree (T->left) && checkTree(T->right)

           && (T->key > T->left->key) &&  (T->key < T->right->key))

  return true;

  else return false;

}
```