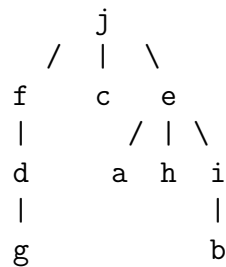
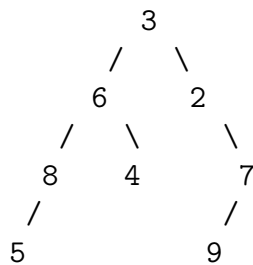


Questions

- What is the sequence of nodes visited for the following tree for a preorder, postorder, and breadth first traversal?



- Give the order of nodes visited in a pre-, in-, post-, and level-order traversal.



- Consider the polynomial

$$5y^2 - 3y + 2.$$

- Write the polynomial as an *expression tree* that obeys the usual ordering of operations.
Hint: to clarify the ordering for yourself, first write the expression with brackets indicating the order of operations.
 - Write the polynomial as *postfix* expression.
- Convert the following infix expressions to postfix expressions.
Assume the usual ordering of operations: multiple $+$, $-$ (or $*$, $/$) are evaluated left to right, and $*$, $/$ has precedence over $+$, $-$.
 - $a*b/(c-d)$
 - $a/b+(c-d)$
 - $a/b+c-d$
 - Evaluate the following postfix expressions made out of *single* digit numbers and the usual integer operators.

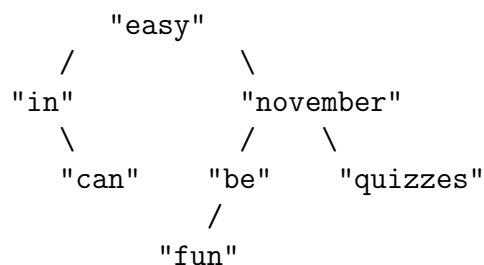
- (a) 26+35-/
 - (b) 234*5*-
 - (c) 234-/5*
 - (d) 6342^*+5-
6. Two binary trees A and B are said to be *isomorphic* if, by swapping the left and right subtrees of some nodes of A, one can obtain a tree identical to B. For example, the following two binary trees are isomorphic.



Write a recursive algorithm `isIsomorphic(n1, n2)` for checking if two trees are isomorphic. The arguments should be references to the root nodes of the two trees.

To test if two non-null nodes are equal, assume each node has a key and check if the keys of the two nodes are equal. That way, our definition of “equal” does not involve other fields defined at each node, namely references to children.

- 7. Prove that an in-order traversal of a binary search tree visits the nodes in order defined by the keys. Hint: what is the variable used for the proposition ? The number of nodes of the tree? Something else?
- 8. Give a non-recursive version of the binary search tree operations `find`, `findMin`, `findMax`
- 9. Consider the binary tree (with null child references not shown):



- (a) What is the order of nodes visited in a *pre-order* traversal?
- (b) What is the order of nodes visited in a *post-order* traversal?

- (c) Transform this binary tree into a *binary search tree* of height 2, defined by the natural ordering on strings.
 - (d) Represent this binary tree using an array (of references to strings), such that the parent/child indices in the array are the same as that used in a *heap*.
10. (a) Draw the binary tree whose in-order traversal is DBEAFC and whose pre-order traversal is ABDECF.
- (b) What is the *post-order* traversal of this tree?
- (c) Draw all *binary search trees* of height 2 that can be made from *all* the letters ABCDEF, assuming the natural ordering.
11. How many binary search trees can you make from **a,b,c,d** assuming their natural ordering?
12. (a) Form a binary search tree of strings from the sentence:

Form a binary search tree of strings

In particular, the tree must be constructed by adding the words in the order they are found in the sentence, namely first add the word “Form” to an empty tree, then add the word “a”, etc.

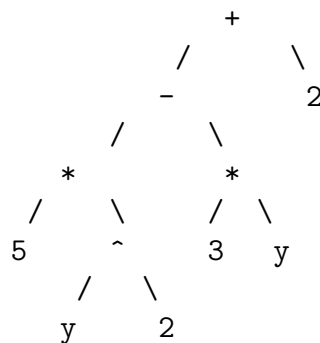
- (b) What is the tree that results, if you run `remove(‘Form’)` on the tree in (a) ?

Answers

1. preorder: jfdgceahib
 post: gdfcahbiej
 breadth: jfcedahigb

2. pre-order: 3 6 8 5 4 2 7 9
 in-order: 5 8 6 4 3 2 9 7
 post-order: 5 8 4 6 9 7 2 3
 level order: 3 6 2 8 4 7 5 9

3. (a) Writing with brackets, we get $((5 * (y^2)) - (3 * y)) + 2$, and the corresponding expression tree is:



(b) The postfix representation of this tree is:

5 y 2 ^ * 3 y * - 2 +

4. The solution is the right column. I have written intermediate solutions which is what you should go through (in your head, or explicitly as I have done).

	IN-FIX	BRACKETED IN-FIX	BRACKETED POST-FIX	POSTFIX
(a)	$a*b/(c-d)$	$((a*b)/(c-d))$	$((ab*)(cd-)/)$	$ab*cd-/$
(b)	$a/b+(c-d)$	$((a/b)+(c-d))$	$((ab/)(cd-)+)$	$ab/cd-+$
(c)	$a/b+c-d$	$((a/b)+c)-d$	$((ab/)c+)d-$	$ab/c+d-$

5. (a) -4 i.e. $8 / (-2)$
 (b) -58 i.e. $2 - 60$
 (c) -10 i.e. $2 * -5$
 (d) 49 i.e. $6 + (3*(4^2)) - 5$

To solve these problems, you use a stack. (You might solve the problem in your head, but be aware that your solution is using a stack.) For example, the last problem is done like this:

```

-      First we push the first four numbers on the stack..
6,3,4,2 Then apply ^ to the two two elements (4^2)...
6,3,16  Then apply * to the top two elements  (3*16)...
6,48    Then apply + to the top two elements  (6+48)...
54       Then push 5 onto the stack...
54,5    Then apply '-' to the top two elements (54 - 5)
49

```

6. The idea of the algorithm is to check the various possible situations. First check that the roots match, namely both are not null and the keys are identical.

Once the two trees pass that test, you compare the left and right children of the two trees. For the two trees to be isomorphic, either (1.) the left subtree of first tree is isomorphic to the left subtree of the second tree AND the right subtree of first tree is isomorphic to the right subtree of the second tree, or (2) the left subtree of first tree is isomorphic to the right subtree of the second tree AND the right subtree of first tree is isomorphic to the left subtree of the second tree.

Here is Java code to solve this problem:

```

isIsomorphic( n1, n2){

    if(n1 == null && n2==null)
        return true;
    else if(n1 == null || n2==null)
        return false;

    if( n1.key != n2.key )
        return false;

    if( isIsomorphic(n1.getRightChild(), n2.getRightChild()) &&
        isIsomorphic(n1.getLeftChild(), n2.getLeftChild())){
        return true}
    if( isIsomorphic(n1.getRightChild(), n2.getLeftChild()) &&
        isIsomorphic(n1.getLeftChild(), n2.getRightChild())){
        return true}

    return false;
}

```

7. The induction variable is the height of the tree. The base case is that the height is 0, so there is just the root node. Obviously the claim is true for the base case.

Suppose that for any binary search tree of height k , an in-order traversal visits the nodes in their correct order. Consider now a binary search tree of height $k + 1$. The inorder traversal first visits all the nodes in the left subtree and, by the induction hypothesis, these nodes are visited in order (since the left subtree is of height no more than k). It then visits the root node. The root element is greater than all nodes visited in the subtree, by definition. Then it visits all the nodes in the right subtree. These nodes are visited in order (by induction hypothesis) and the right subtree nodes have keys that are all greater than the root.

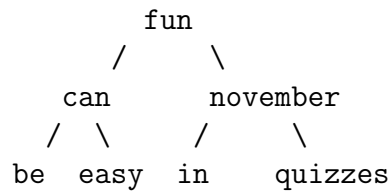
```
8.  find(root,key){
    cur = root
    while ((cur != null) or (cur.key != key))
        if (key < cur.key)
            cur = cur.left
        else
            cur = cur.right
    }
    return cur
}
```

```
findMin(root){
    if (root == null)
        return null
    else{
        cur = root
        while (cur.left != null){
            cur = cur.left}
        return cur
    }
}
```

```
findMax(root){
    if (root == null)
        return null
    else{
        cur = root
        while (cur.right != null)
            cur = cur.right
        return cur
    }
}
```

9. (a) easy, in, can, november, be, fun, quizzes
 (b) can, in, fun, be, quizzes, november, easy

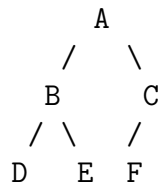
(c)



(d) 1 2 3 4 5 6 7 8 9 10 11 12
 easy in november * can be quizzes * * * * fun

where * is a null reference. Note that the strings are not stored in the array, but rather the array stores references to the strings.

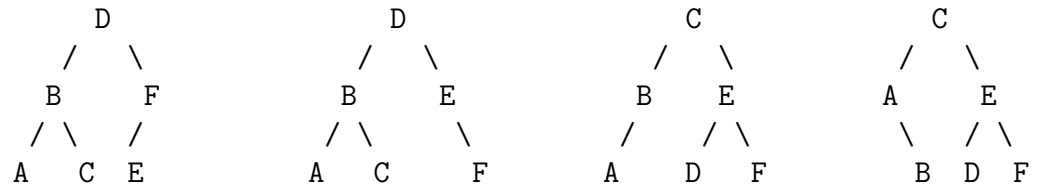
10. (a)



(b)

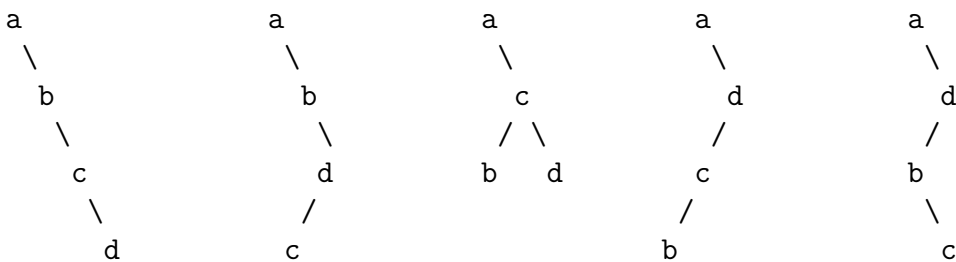
DEBFCA

(c)

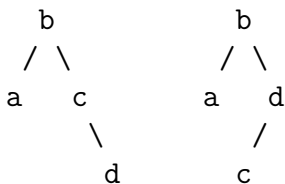


11. The systematic way to do this would be to write down all the BST's with **a** at the root, **b** at the root, **c** at the root, **d** at the root. It turns out there are $5+5+2+2 = 14$ possibilities.

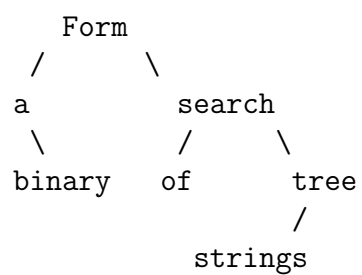
For the BSTs with **a** at the root, the keys **b**, **c**, **d** would all be in the right subtree. There are five ways – see below. Similarly, there are five ways to have **d** in the root. (not shown)



What about **b** in the root? There are 2 ways. (Similarly there are 2 ways to have **c** in the root – not shown.)



12. (a)



(b)

