Questions

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

```
          j
       /  |  \
     f    c   e
    /       / | \
   d       a  h  i
 /              /
g              b
```

2. Consider the polynomial

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

   (a) 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.

   (b) Write the polynomial as *postfix* expression.

3. 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) `a*b/(c-d)`

   (b) `a/b+(c-d)`

   (c) `a/b+c-d`

4. 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-`

5. 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.

```
         5                               5
       /    \                          /     \
      8      4                        4       8
    /  \                                     / \
   2    6                                   2   6
       / \                                     / \
      3   2                                   2   3
```

Write a recursive algorithm `isIsomorphic( n1, n2)` for checking if two binary 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.

6. Prove by induction that an in-order traversal of a binary search tree visits the nodes in order defined by the keys.

Hint: What is the base case? what is the variable used for the proposition ? The number of nodes of the tree? Something else? What is the induction step?

7. Give a non-recursive version of the binary search tree operations `find`, `findMin`, `findMax`

8. Consider the binary tree (with null child references not shown):

```
        "easy"
      /         \
   "in"        "november"
      \          /    \
     "can"    "be"    "quizzes"
             /
         "fun"
```

Transform this binary tree into a *binary search tree* of height 2, defined by the natural ordering on strings.

9. (a) Draw a binary tree of height 2 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.

10. How many binary search trees can you make from **a,b,c,d** assuming their natural ordering?

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

```
             +
          /      \
         -        2
      /     \
   *         *
 /    \     /  \
5     ^     3   y
    /   \
   y     2
```

   (b) The postfix representation of this tree is:

```
5  y  2 ^ *  3  y * - 2 +
```

3. 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-`

4. (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. Consider left to right is
bottom to top in stack
```



```
6,3,4,2  Then apply ^ to the top 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
```

5. 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;
}
```

6. 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 left 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.

7.
```
find(root,key){
  cur = root
  while ((cur != null) and (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
   }
```

8.

```
          fun
        /     \
     can      november
    / \      /       \
  be  easy  in     quizzes
```

9. (a) Some detective work is required for this one. From the pre-order traversal, we see that A is the root. The inorder traversal then tells us that DBE are in the left subtree and FC are in the right subtree. But how are these nodes arranged in the two subtrees. For the left subtree of A, the pre-order traveral tells us that root of that subtree is B. The inorder traversal tells us that D and E must be left and right children of B. Simiarly right subtree of A contains nodes F and C, the right child of A must be C because the preorder traversal says C gets visited before F. But F follows C in the inorder traversal and so F must be the right child of C. So the answer is:

```
      A
    /   \
   B     C
  / \   /
 D   E F
```

(b)

```
DEBFCA
```

(c)

```
      D                 D                 C                C
    /   \             /   \             /   \            /   \
   B     F           B     E           B     E          A     E
  / \   /           / \     \         /     / \          \   / \
 A   C E           A   C     F       A     D   F          B D   F
```

10. 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 give ways to have `d` in the root. (not shown)

```
a              a            a            a            a
 \              \            \            \            \
  b              b            c            d            d
   \              \          / \          /            /
    c              d        b   d        c            b
     \            /                     /              \
      d          c                     b                c
```

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

```
   b           b
  / \         / \
 a   c       a   d
      \         /
       d       c
```

11. (a)

```
       Form
     /      \
    a         search
     \        /     \
    binary   of      tree
                     /
                 strings
```

(b)

```
        of
     /      \
    a         search
     \             \
    binary          tree
                     /
                 strings
```

or

```
       binary
     /      \
    a         search
              /     \
            of      tree
                     /
                 strings
```

Note: In the first solution, we are moving the smallest element from the right subtree to the root position as per the lecture slides whereas in the second solution, we are moving the largest element in the left subtree to the root position and both the solutions are valid.