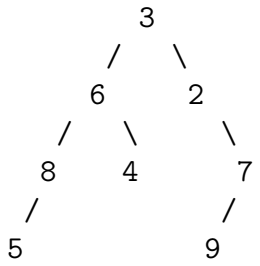
Questions

1. 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.

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



3. Convert the following infix expressions to postfix expressions. You must assume that expressions with multiple `+, -` (or `*, /`) are evaluated left to right, and that `*, /` has precedence over `*, -` i.e. the usual ordering of operations.

   [The wording in the question was modified March 15.]

   (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. Suppose you have a complete binary tree with $n = 2^{h+1} - 1$ nodes, i.e. all levels of the tree up to h are filled. What is the sum of the pathlengths from the root to all nodes ?

Didn't get

6. Consider the following binary numbers, which are each between 0 and 1:

$$p_1 = .0011, \quad p_2 = .0101, \quad p_3 = .0001, \quad p_4 = .0101 \quad , p_5 = .001$$

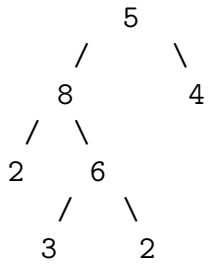
Define

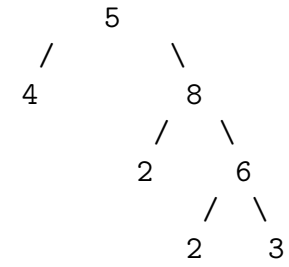
$$F_i = \sum_{k=1}^{i} p_k \ , \quad i = 1, 2, 3, 4, 5.$$

(a) Write each F_i as a binary number.

(b) Draw a complete binary tree of height $h = 4$ and *label the leaves* with labels from $\{1, \ldots, 5\}$ increasing left to right, such that p_i is the fraction of leaves with label i.

7. Two binary trees A and B are said to be *isomorphic* if, by swapping the left and right subtrees of certain (not necessarily all) 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.)

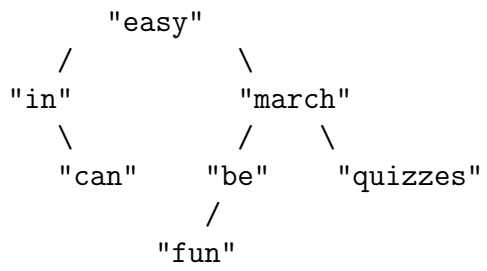
8. Give a non-recursive version of the binary search tree operations `find`, `findMinimum`, `findMaximum`

9. Suppose you have an array with the following characters in it.

```
f  b  u  e  l  a  k  d  w
```

Show how the array evolves after each of the $n = 9$ loops of the slow `buildHeap` method.

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



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

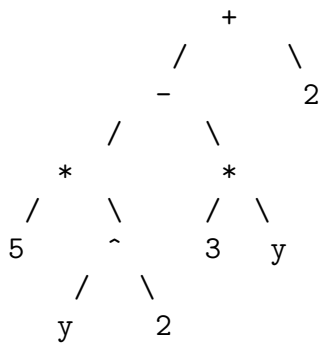
(b) What is the order of nodes visited in a *post-order* traversal?

(c) 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*.

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

Answers

1. (a)



   (b) `5 y 2 ^ * 3 y * - 2 +`

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. 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`

   You should be solving this problem using a stack. For the last example, you could do it 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
```

5. Counting the root as node 1, we saw from our analysis of heaps that the pathlength from root to node i is $\lfloor \log i \rfloor$. So the sum of these is

$$\sum_{i=1}^{n} 2^i \lfloor \log i \rfloor = \sum_{l=0}^{h} l2^l$$

where the expression on the right is a sum of two terms, namely the pathlength sum up to but not including level h, and the pathlength sum for level h only.

Following a similar argument as lecture 24, we consider

$$\sum_{l=0}^{h} lx^{l-1} = \sum_{l=0}^{h} \frac{dx^l}{dx} = \frac{d}{dx} \sum_{l=0}^{h} x^l = \frac{d}{dx}(\frac{1-x^{h+1}}{1-x})$$

which you can now solve using basic calculus. Then substitute in $x = 2$, and plug into the above formula to get:

$$\sum_{l=0}^{h} l2^{l-1} = (h+1)2^h + 1 - 2^{h+1}$$

and so

$$\sum_{l=0}^{h} l2^l = (h+1)2^{h+1} + 2 - 2^{h+2} = (h-1)2^{h+1} + 2$$

But $n = 2^h - 1$, and so $h = \log(n+1)$, and so the sum of pathlengths from root to all nodes is $(\log(n+1) - 1)2(n+1)$ i.e. roughly $n \log n$.

6. (a)

$$F_1 = .0011, F_2 = .1000, F_3 = .1001, F_4 = .1110, F_5 = 1.0000$$

(b) Note

did not get!

$$p_1 = \frac{3}{16}, p_2 = \frac{5}{16}, p_3 = \frac{1}{16}, p_4 = \frac{5}{16}, p_5 = \frac{2}{16}$$

The answer is a binary tree of height 4 and the 16 nodes at level 4 are labelled 1112222234444455.

7. 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;
}
```

8.

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

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

```
findMaximum(root){
   cur = null
   while ((cur != null) or (cur.right != null))
      cur = cur.right
   return cur
}
```

9. The boundary between the heap and non-heap elements is marked with |.

```
  1   2   3   4   5   6   7   8   9
-------------------------------------
| f   b   u   e   l   a   k   d   w
  f | b   u   e   l   a   k   d   w    (added f)
  b   f | u   e   l   a   k   d   w    (added b)
  b   f   u | e   l   a   k   d   w    (added u)
  b   e   u   f | l   a   k   d   w    (added e)
  b   e   u   f   l | a   k   d   w    (added l)
  a   e   b   f   l   u | k   d   w    (added a)
  a   e   b   f   l   u   k | d   w    (added k)
  a   d   b   e   l   u   k   f | w    (added d)
  a   d   b   e   l   u   k   f   w |  (added w)
```

Note: A similar example was given at the end of lecture 24 for the build heap fast algorithm.

10. (a) `easy, in, can, march, be, fun, quizzes`

(b) `can, in, fun, be, quizzes, march, easy`

(c)

```
  1    2    3     4    5    6      7      8   9   10   11   12
 easy  in  march   *   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.

(d)