Problem 1

Suppose that you start with an empty BST (binary search tree), and then insert the following values (in this order): 7, -2, 10, 0, 13, 14, 1, 3. Draw the tree that results (to save time, you can draw just the tree, not all of the steps to get there).

Then, give the in-order, pre-order, and post-order traversals of this tree. Compare these to the original values you inserted - is there any similarity?

Finally, give rearrange the numbers in such a way that, if you started over and inserted them again, you would end up with exactly the same tree (even though you inserted in another order).
Problem 2

Suppose that you have a simple BSTNode class, which has fields left, right, and val. (For simplicity, assume that they are public fields. In a real program, they would probably be private.)

Write a function, print_pre_order(node), which will print out a pre-order traversal of all of the values in the tree (one value per line). Use recursion!

```
def print_pre_order(root):
```
Problem 3

Repeat Problem 2, but this time, **DO NOT** use recursion. Instead, use a list to keep a stack of “TODO” items - that is, trees that you need to print out.

It's easy to get this wrong. **After** you've written it on paper, open up your computer and try out your code. Does it work? Debug it if not.

(My hope is that, after trying this, you'll decide that recursion was the easier way to get this done!)

```python
def print_pre_order(root):
```
Problem 4

In this problem, you will recreate a BST, using only a pre-order traversal. The input to this function will be a list (or tuple) of values, representing the pre-order traversal of some old tree. Recreate the tree, and return it. (You may use the nested-tuples format, or you can actually create BSTNodes. Your choice.)

**Use recursion.** The key to this is to realize that, in a pre-order traversal, the first node in the list is the root of the subtree - and what follows is the left subtree, followed by the right subtree. And you can tell the place where the left subtree ends, and the right begins, because you know that this is a BST. (What do you know about the values in each subtree?)

Your function must handle the empty list (return an empty tree).

**Bonus.** Rewrite your function (if you didn’t do it this way the first time) to use list comprehensions.

```python
def build_bst_from_pre_order(elems):
```