1. Write down the three traversal orders for the BST in question 2.
   Preorder:
   Postorder:
   Inorder:

2. Insert the following values into this tree, in this order: 18, 2, 32, 41, 55, 43, 53.

3. What did you notice about where the new nodes go?

Next, we will develop method `insert`. Its API is below:

```python
class BinarySearchTree:
    # === Private Attributes ===
    # The item stored at the root of the tree, or None if the tree is empty.
    _root: Optional[object]
    # The left subtree, or None if the tree is empty
    _left: Optional['BinarySearchTree']
    # The right subtree, or None if the tree is empty
    _right: Optional['BinarySearchTree']

    # === Representation Invariants ===
    # - If _root is None, then so are _left and _right. This represents an empty BST.
    # - If _root is not None, then _left and _right are BinarySearchTrees.
    # - (BST Property) All items in _left are <= _root, and all items in _right are >= _root.

    def __init__(self, root: Optional[object]) -> None:
        """Initialize a new BST with the given root value and no children.

        If <root> is None, make an empty tree, with subtrees that are None.
        If <root> is not None, make a tree with subtrees are empty trees.
        ""
    
    def is_empty(self) -> bool:
        """Return True if this BST is empty."
    
    def insert(self, item: object) -> None:
        """Insert <item> into this BST, maintaining the BST property.

        Do not change positions of any other nodes.
        """
```
1. Suppose we are to insert 13 into a BST. In the space below, identify each case that may need to be handled separately. For each, describe the case and draw a tree that is an instance of it, then draw or describe what the tree should look like after deletion.

<table>
<thead>
<tr>
<th>Case (description and tree)</th>
<th>Outcome (tree)</th>
</tr>
</thead>
</table>

2. Are there any cases that can be collapsed and handled in the same way?

3. What helper method(s) would be useful? On a separate piece of paper, write appropriate API(s).

4. Now you are ready to write method `insert`. Assume that your helper(s) are implemented correctly.

5. Finally, write your helper method(s).