Quiz #9: Binary Search Trees

Read over the declaration of class **BinarySearchTree**, and the header and docstring for method **num_in_range**:

```python
class BinarySearchTree:
    # Binary Search Tree class.
    
    This class represents a binary tree satisfying the Binary Search Tree property: for every node, its value is >= all items stored in its left subtree, and <= all items stored in its right subtree.
    
    # === Private Attributes ===
    _root: Optional[object]
    # The item stored at the root of the tree, or None if the tree is empty.
    _left: Optional['BinarySearchTree']
    # The left subtree, or None if the tree is empty
    _right: Optional['BinarySearchTree']
    # The right subtree, or None if the tree is empty

    # === 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 num_in_range(self, start: int, end: int) -> int:
        # Return the number of items between <start> and <end>, inclusive.
        # Use the BST property to ensure you don’t make unnecessary recursive calls.
```

Below is a picture of a **binary search tree**, with several levels. We’ll refer to this tree as the variable bst.
1. What is the output of `bst.num_in_range(5, 50)`?

2. What are the outputs of `bst._left.num_in_range(5, 50)` and `bst._right.num_in_range(5, 50)`?

3. Give an example of numbers x and y such that the call `bst.num_in_range(x, y)` would only require one subtree to be checked.

4. Implement `num_in_range` using only the attributes `_root`, `_left`, `_right`, and the method `is_empty`.

   ```python
def num_in_range(self, start, end):
```