6.5 Binary Search Tree Implementation and Search

TREE:

- Empty tree:
  - _root = None
  - List = [] for the subtrees

BINARY SEARCH TREE:

- **Empty tree:**
  - _root = None
  - _left = None
  - _right = None
  - ONLY CASE WHEN ANY OF THE VALUES CAN BE NONE.
- **Non-empty tree:**
  - _root has a value != None
  - _left and _right might refer to empty BUT AREN'T NONE
- _left and _right aren't set as parameters, since there is more restriction on where values can be located In the tree

- FOR EVERY NODE:
  - **The right children must be greater and the left children must be less**
- MAX: the rightmost node
- MIN: the leftmost node

SEARCHING A BINARY SEARCH TREE:

- Binary search: comparing the target item with the middle of the list
- **Search algorithm:**
  - compare the item against the root
  - then search in each of the subtrees until item is found/ all subtrees have been searched

IMPLEMENTATION:

```
from __future__ import annotations
from typing import Optional, Any

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: The item stored at the root of the tree, or None if the tree is empty.
    _left: The left subtree, or NOne if the tree is empty.
    _right: the right subtree, or NOne if the tree is empty.
    """
    _root: Optional[Any]
    _left: Optional[BinarySearchTree]
    _right: Optional[BinarySearchTree]

    # === Representation Invariants ===
    # If _root is None , so are _left and _right -> REPRESENTS AN EMPTY BST
    # If _root is not None, then _left and _right are BinarySearchTrees.
    # (BST Property): All items in _left <= _root,  All items in_right >= _root.

    def __init__(self, root: Optional[Any]) -> None:
        """Initialize a new BST containing only the given root value.
        If <root> is None, initialize an empty BST.
        """
        if root is None:
            self._root = None
            self._left = None
            self._right = None

        else:
            self._root = root
            self._left = BinarySearchTree(None)  # Empty BST
            self._right = BinarySearchTree(None)  # Empty BST

    def is_empty(self) -> bool:
        """Return whether this binary search tree is empty or not"""
        return self._root is None

    # VERSION 1
    def __contains2__(self, item: Any) -> bool:
        """Return whether <item> is in this BST."""
        if self.is_empty():
            return False  # THE ITEM DOESN'T EXIST IN AN EMPTY
        else:
            if item == self._root:
                return True
            elif item < self._root:
                return self._left.__contains2__(item)  # to show the recursiveness but,
                # you can also write item in self._left
            else:
                return self._right.__contains2__(item) # to show the recursiveness but,
                # you can also write item in self._right

    # VERSION 2
    def __contains__(self, item: Any) -> bool:
        """Return whether <item> in this BST"""
        if self.is_empty():
            return False
        elif item == self._root:
            return True
        elif item < self._root:
            return item in self._left
        else:
            return item in self._right
```