Quiz 06: Binary Search Trees

You will still be able to edit your answers up until the due date: **Jun. 30 12:59 AM**

Submit

QUESTION 1

ANSWERED

Binary Search Tree construction

0 points possible

9, 8, 6, 5, 3, 2, 1

5, 3, 2, 1, 9, 8, 6

3, 5, 2, 9, 1, 8, 6

5, 2, 1, 3, 9, 6, 8

Go To Question

QUESTION 2

ANSWERED

Binary Search Tree construction

0 points possible

2, 3, 6, 1, 4, 9, 7

2, 1, 3, 6, 4, 7, 9

1, 3, 2, 4, 6, 9, 7

4, 2, 3, 6, 1, 7, 9

Go To Question

QUESTION 3

ANSWERED

AVL Tree Construction

0 points possible

0

1

2

3

4

Go To Question

QUESTION 4

ANSWERED

AVL Tree Construction

0 points possible

3, 2, 1, 5, 6, 8, 9

5, 1, 2, 3, 6, 8, 9

1, 2, 3, 5, 6, 8, 9

5, 2, 1, 3, 8, 6, 9

Go To Question

QUESTION 5

ANSWERED

AVL Tree Construction

0 points possible

0

1

2

3

4

Go To Question

QUESTION 6

ANSWERED

AVL Tree Construction

0 points possible

3, 2, 1, 6, 4, 7, 9

2, 1, 3, 4, 7, 6, 9

2, 3, 1, 4, 6, 7, 9

3, 4, 1, 2, 7, 6, 9

Go To Question

QUESTION 7 ANSWERED BSTree Implementation

code.py

```
class BSTree:
    class Node:
        def __init__(self, val, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right

    def __init__(self):
        self.root = None

    def __contains__(self, val):
        def contains_rec(node):
            if not node:
                return False
            elif val < node.val:
                return contains_rec(node.left)
            elif val > node.val:
                return contains_rec(node.right)
            else:
                return True
        return contains_rec(self.root)

    def add(self, val):
        assert(val not in self)
        def add_rec(node):
            if not node:
                return BSTree.Node(val)
            elif val < node.val:
                return BSTree.Node(node.val, left=add_rec(node.left), right=node.right)
            else:
                return BSTree.Node(node.val, left=node.left, right=add_rec(node.right))
        self.root = add_rec(self.root)

    def __iter__(self):
        def iter_rec(node):
            if node:
                yield from iter_rec(node.left)
                yield node.val
                yield from iter_rec(node.right)
        return iter_rec(self.root)

    def count_internal(self):
        def rec_count(node):
            if node:
                c=0
                if node.left or node.right:
                    c=1
                return c+rec_count(node.left)+rec_count(node.right)
            else:
                return 0
        return rec_count(self.root)
```

QUESTION 8 ANSWERED **BSTree Implementation**

code.py

```
class BSTree:
    class Node:
        def __init__(self, val, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right

    def __init__(self):
        self.root = None

    def __contains__(self, val):
        def contains_rec(node):
            if not node:
                return False
            elif val < node.val:
                return contains_rec(node.left)
            elif val > node.val:
                return contains_rec(node.right)
            else:
                return True
        return contains_rec(self.root)

    def add(self, val):
        assert(val not in self)
        def add_rec(node):
            if not node:
                return BSTree.Node(val)
            elif val < node.val:
                return BSTree.Node(node.val, left=add_rec(node.left), right=node.right)
            else:
                return BSTree.Node(node.val, left=node.left, right=add_rec(node.right))
        self.root = add_rec(self.root)

    def __iter__(self):
        def iter_rec(node):
            if node:
                yield from iter_rec(node.left)
                yield node.val
                yield from iter_rec(node.right)
        return iter_rec(self.root)

    def is_full(self):
      def rec_full(node):
            if node is None:
                return True
            if node.left is None and node.right is None:
                return True
            if node.left is not None and node.right is not None:
                return(rec_full(node.left) and rec_full(node.right))
            return False
      return rec_full(self.root)
```