Romerico David, Haley Elliott, Julian Halsey, Nkechiyem Molokwu
COSC 336
04/16/2024

Assignment 6

Exercise 1:

The elements that could have been the pivot element are 4, 5, and 9.

Exercise 2:

The probability that the Partition function produces a split in which the size of both the resulting sub problems is at least size $\alpha * n$ is $1 - 2\alpha$.

Let $\alpha = 0.3$ and $n = 10$

$n*\alpha = 3$ and $n-\alpha*n = 10-3 = 7$ $\therefore$ The pivot must be located between the 3rd element (exclusive) and 7th element (inclusive). This means there is 4 possible pivots and the probability of choosing one of these possibles pivot is $4/10 = 2/5 = 0.4$.

Plugging into the answer choices:

$\alpha = 0.3$

$1 - \alpha = 0.7$

$1 - 2\alpha = 1 - 2(0.3) = 0.4$

$2 - 2\alpha = 2 - 2(0.3) = 1.4$

$\therefore$ The probability is $1 - 2\alpha$.

Programming Task:

Description of Algorithm:

For our insert function, we modified the if statement so it inserts a duplicate into the left subtree. Then, we incremented the size of the node. The leftRotate function takes a Node t as a parameter and creates a new Node x as the right subtree of t. The right subtree of t is then replaced by the left subtree of x, which itself is replaced with t. Finally, the function returns Node x. Similarly, the function rightRotate also takes a Node t as a parameter and creates a new Node x as the left subtree of t. The left subtree of t is then replaced by the right subtree of x, which itself is replaced with t. Again, Node x is returned. Additionally, these functions adjust the size of

the Node t tree and Node x tree. Lastly, we created a helper method to print the preorder traversal of the BST, that is Node, Left, Right.

Code:

```
class Node {
        int key;
        // New field for the size of the subtree rooted at this node
        int size;
        Node left, right;

        public Node(int item) {
                key = item;
                size = 1; // Initialize size to 1 for a single node
                left = right = null;
        }
}

class BinarySearchTree {
        Node root;

        BinarySearchTree() {
                root = null;
        }

        Node insert(Node node, int key) {
                if (node == null) {
                        node = new Node(key);
                        return node;
                }

            // Check if key is less than or EQUAL to node's key
                if (key <= node.key)
                        node.left = insert(node.left, key);
                else
                        node.right = insert(node.right, key);

                node.size++;
```

```
        return node;
}

Node search(Node root, int key) {
        if (root == null || root.key == key)
                return root;

        if (root.key < key)
                return search(root.right, key);

        return search(root.left, key);
}

Node leftRotate(Node t) {
        if (t.right == null)
                return t;
        Node x = t.right;
        t.right = x.left;
        x.left = t;

        // Update sizes of the rotated nodes
        t.size = 1 + (t.left != null ? t.left.size : 0)
    + (t.right != null ? t.right.size : 0);
        x.size = 1 + (x.left != null ? x.left.size : 0)
    + (x.right != null ? x.right.size : 0);

        return x;
}

Node rightRotate(Node t) {
        if (t.left == null)
                return t;
        Node x = t.left;
        t.left = x.right;
        x.right = t;

        // Update sizes of the rotated nodes
        t.size = 1 + (t.left != null ? t.left.size : 0)
```

```
            + (t.right != null ? t.right.size : 0);
                x.size = 1 + (x.left != null ? x.left.size : 0)
            + (x.right != null ? x.right.size : 0);

                return x;
        }

        void printPreorder(Node node) {
                if (node == null)
                        return;
                //Node, Left, Right
                System.out.print("("+node.key+","+node.size+"),");
                printPreorder(node.left);
                printPreorder(node.right);
        }
}
```

Results Table: Programming Task 1

Data Set #	Before Left Rotate	After Left Rotate
6.1	(448, 1000), (184, 447), (43, 187), (10, 43), (4, 8), (0, 4), (3, 3), (1, 1), (4, 1), (9, 3), (5, 2), (8, 1), (32, 34), (23, 23), (11, 12), (13, 11), (12, 2), (12, 1), (16, 8), (14, 2), (15, 1), (23, 5), (21, 4), (18, 2), (20, 1)	(964, 1000), (448, 973), (184, 447), (43, 187), (10, 43), (4, 8), (0, 4), (3, 3), (1, 1), (4, 1), (9, 3), (5, 2), (8, 1), (32, 34), (23, 23), (11, 12), (13, 11), (12, 2), (12, 1), (16, 8), (14, 2), (15, 1), (23, 5), (21, 4), (18, 2)
6.2	(745, 10000), (151, 767), (8, 141), (3, 6), (2, 2), (3, 1), (6, 3), (4, 2), (4, 1), (105, 134), (63, 86), (63, 48), (9, 47), (54, 46), (21, 38), (21, 10), (20, 9), (18, 8), (18, 7), (16, 6), (14, 4), (11, 2), (12, 1), (16, 1), (18, 1)	(5102, 10000), (745, 5096), (151, 767), (8, 141), (3, 6), (2, 2), (3, 1), (6, 3), (4, 2), (4, 1), (105, 134), (63, 86), (63, 48), (9, 47), (54, 46), (21, 38), (21, 10), (20, 9), (18, 8), (18, 7), (16, 6), (14, 4), (11, 2), (12, 1), (16, 1)