Binary Search Tree Inputs:

```
BSTree<int> bsTree;
bsTree.insert(1);
bsTree.display();
bsTree.insert(1);
bsTree.display();
bsTree.remove(1);
bsTree.display();
bsTree.remove(1);
bsTree.display();
bsTree.insert(1);
bsTree.insert(2);
bsTree.insert(3);
bsTree.insert(0);
bsTree.display();
cout << "Max is: " << bsTree.findMax() << endl;
cout << "Min is: " << bsTree.findMin() << endl;
cout << "Size is: " << bsTree.size() << endl;
cout << "Contains 2?: " << bsTree.contains(2) << endl;
cout << "Contains 4?: " << bsTree.contains(4) << endl;
```

Binary Search Tree Outputs:

```
      1

      1

                    3

            2

      1

            0
Max is: 3
Min is: 0
Size is: 4
Contains 2?: 1
Contains 4?: 0
```

AVL Tree Inputs:

```
AVLTree<int> avlTree;
avlTree.insert(1);
avlTree.display();
avlTree.insert(1);
avlTree.display();
avlTree.remove(1);
avlTree.display();
avlTree.remove(1);
avlTree.display();
avlTree.insert(1);
avlTree.insert(2);
avlTree.insert(3);
avlTree.insert(0);
avlTree.display();
cout << "Max is: " << avlTree.findMax() << endl;
cout << "Min is: " << avlTree.findMin() << endl;
cout << "Size is: " << avlTree.size() << endl;
cout << "Contains 2?: " << avlTree.contains(2) << endl;
cout << "Contains 4?: " << avlTree.contains(4) << endl;
```

AVL Tree Outputs:

```
    1

    1

        3

    2

        1

            0
Max is: 3
Min is: 0
Size is: 4
Contains 2?: 1
Contains 4?: 0
```

Red-Black Tree Inputs:

```
RBTree<int> rbTree;
rbTree.insert(1);
rbTree.display();
rbTree.insert(1);
rbTree.display();

rbTree.insert(1);
rbTree.insert(2);
rbTree.insert(3);
rbTree.insert(0);
rbTree.display();
cout << "Max is: " << rbTree.findMax() << endl;
cout << "Min is: " << rbTree.findMin() << endl;
cout << "Size is: " << rbTree.size() << endl;
cout << "Contains 2?: " << rbTree.contains(2) << endl;
cout << "Contains 4?: " << rbTree.contains(4) << endl;
```

Red-Black Tree Outputs:

```
    1 (BLK)

    1 (BLK)

            3 (RED)

        2 (BLK)

    1 (BLK)

        0 (RED)
Max is: 3
Min is: 0
Size is: 4
Contains 2?: 1
Contains 4?: 0
```

I chose the above inputs for my code to check edge cases such as inserting nodes with values that are already present in the tree as well as removing from an empty tree. The rest of the inputs were mostly insertions so that I could display the tree and prove that everything was filling in correctly, which proved robustness of the insert and display function. I also called the findMin(), findMax(), size(), and contains(x) functions in scenarios in which they could give different outputs, proving their robustness as well. The outputs of the test cases I created matched perfectly between the three different tree types, showing that they were all implemented correctly. The only difference is found in the red-black tree’s display, which tracks color as well as value of each node.