Objective(s):

a. To be able to implement binary-search-tree insert(int d) method
b. To be able to implement binary tree traversal method
c. To be able to implement binary tree search method

Task 1: (MyBST_Basic_XXYYYY.java and BNode.java are implemented in pack,)

Implement the constructor `MyBST_Basic_XXYYYY()` and complete `void insert(int d)`

```
package Lab10a.pack;

public class MyBST_Basic_XXYYYY {
    private BNode root;
    public MyBST_Basic_XXYYYY() {  root = null; }
    public MyBST_Basic_XXYYYY(Integer[] input) {
        if (input.length == 0)
            return;
        root = null;
        for (int i = 0; i < input.length; i++) {
            /* your code 3 */
                insert(input[i]);
        }
    }
    public void insert(int d) {
        if (root == null) {
            root = new BNode(d);
        } else {
            BNode cur = root;
            while (cur != null) {
                if (d < cur.data) {
                    if (cur.left != null) {
                        cur = cur.left;
                    }
                    else {
                        /* your code 1 */
                        return;
                    }
                } else {  //(d >= p.data)
                    if (cur.right != null) {
                        /* your code 2 */
                    }
                    else {
                        cur.right = new BNode(d);
                        cur.right.parent = cur;
                        return;
                    }
                }
            }
        }
    }
```

```
1  package Lab10a.pack;
2
3  public class BNode {
4      public int data;
5      public BNode left, right, parent;
6
7      public BNode(int d) {
8          data = d;
9      }
10
11     @Override
12     public String toString() {
13         if (left == null && right == null) return "null <- " + data + " -> null";
14         if (left != null && right == null) return left.data + " <- " + data + " -> null";
15         if (left == null && right != null) return "null <- " + data + " -> " + right.data;
16         return left.data + " <- " + data + " -> " + right.data;
17     }
18 }
```

Note that, implement getRoot() if required.

```
    public String toString() {
        if (root.left == null && root.right == null)
        return "null<-" + root.data + "->null";
        // no child
        StringBuilder sb = new StringBuilder();
        stringInOrder(root, sb);
        return sb.toString();
    }
    private void stringInOrder(BNode node, StringBuilder sb) {
        if (node == null) return;
        stringInOrder(node.left, sb);
        sb.append(node.data + " ");
        stringInOrder(node.right, sb);
    }
```

```
public static void demo1() {
    System.out.println("----insert----");
    MyBST_Basic_XXYYYY bst = new MyBST_Basic_XXYYYY();
    bst.insert(2);
    bst.insert(5);
    bst.insert(1);
    System.out.println(bst); // 1 2 5
}
```

Task 2: Complete `MyBST_Basic_XXYYYY(Integer[] input)`

```
public static void demo2() {
  System.out.println("----construct from array of int----");
  // 4 2 6 5 7 1 3 9 8
  Integer[] tree =
    {4,2,6,1,3,5,7,null,null,null,null,null,null,null,9,8,null};
  MyBST_Basic_XXYYYY bst = new MyBST_Basic_XXYYYY(tree);
  System.out.println(bst); // 1 2 3 4 5 6 7 8 9
}
```

Task 3: Complete `void printPreOrder()` and `void printPostOrder()`

```
public static void demo3() {
  Integer[] tree = {4,2,7,1,3,5,8,null,null,null,null,null,6};
  MyBST_Basic_XXYYYY bst = new MyBST_Basic_XXYYYY(tree);
  System.out.println("----traversal methods----");

  System.out.println("\t Pre order:");
  bst.printPreOrder();    // 4 2 1 3 7 5 6 8
  System.out.println("\t Post order:");
  bst.printPostOrder();   // 1 3 2 6 5 8 7 4
}
```

Task 4: Complete `BNode search(int d)`

```
public static void demo4() {
  System.out.println("----search----");
  Integer[] tree = {4,2,7,1,3,5,8,null,null,null,null,null,6};
  MyBST_Basic_XXYYYY bst = new MyBST_Basic_XXYYYY(tree);
  System.out.println("search 4");
  System.out.println(bst.search(4)); // 2 <- 4 -> 7
  System.out.println("search 5");
  System.out.println(bst.search(5)); // null <- 5 -> 6
  System.out.println("search 8");
  System.out.println(bst.search(8)); // null <- 8 -> null
  System.out.println("search 9");
  System.out.println(bst.search(9)); // null
}
```

Task 5: Complete `int size() and int height()`

```
public static void demo5() {
  System.out.println("----size and height----");
  Integer[] tree1 = {4,2,7,1,3,5,8,null,null,null,null,null,6};
  MyBST_Basic_XXYYYY bst1 = new MyBST_Basic_XXYYYY(tree1);
  System.out.println("tree 1 size: " + bst1.size());      // 8
  System.out.println("tree 1 height: " + bst1.height());  // 4

  Integer[] tree2 = {1,null,2,null,3,null,4,null,5,null,6,null,7,null,8};
  MyBST_Basic_XXYYYY bst2 = new MyBST_Basic_XXYYYY(tree2);
  System.out.println("tree 2 size: " + bst2.size());      // 8
  System.out.println("tree 2 height: " + bst2.height());  // 8
}
```

Task 6: Complete `void printLevelOrder()`

```
public static void demo6() {
  System.out.println("print Level Order");
  Integer[] tree = {4,2,7,1,3,5,8,null,null,null,null,null,6};
  MyBST_Basic_XXYYYY bst = new MyBST_Basic_XXYYYY(tree);
  bst.printLevelOrder();
}
```

```
print Level Order
Level 0: 4
Level 1: 2 7
Level 2: 1 3 5 8
Level 3: 6
```

Hint: the number of nodes in each level is the size of the queue (bfs).

Submission: Your **MyBST_Basic_XXYYYY.java** where. XX are the first two digit of your student id and YYYY are the last four.

Due date: TBA