CPE217 – Homework 7

Homework: AVLTree Data Structure and Splay Tree Data Structures

Homework Due Date: 3 November 2019

Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

- คำชี้แจงการส่งงาน
- แต่ละกลุ่ม ควรให้ core person เป็นคนส่งงาน และในช่องข้อความต้องระบุรหัสประจำตัวนักศึกษาของทุกคนที่เป็นสมาชิกในกลุ่ม หาก core person ไม่สามารถส่งงานได้ ให้สมาชิกคนใดก็ได้ส่งงานแทน แต่ต้องบอกว่าส่งแทน core person ซึ่งก็คือใคร มีรหัสอะไร
- โค้ดของคุณต้องมีคอมเม้นต์ (comment) เพื่ออธิบายว่าโค้ดดังที่เห็นอยู่นี้ทำงานอะไร หรือ if นี้ทำตรวจหาอะไร หากกลุ่มไหนไม่มีคอมเม้นต์ในโค้ดจะไม่ได้รับการตรวจ การเขียนคอมเม้นต์ไม่ต้องเขียนแบบละเอียดยิบก็ได้ เท่าที่คุณต้องการให้ผู้ตรวจทราบก็พอ
- งานที่ส่งต้องประกอบด้วย ZIP file ของ src folder ที่สามารถกด F6 รันได้เลย
- สามารถส่งการบ้านช้าได้ แต่หักคะแนนวันละ 10%
- การลอกงานเพื่อนมาส่ง เป็นการทุจริตและมีความผิดทางวินัย หากตรวจพบอาจารย์อาจพิจารณาให้คะแนนการบ้านนั้นหรือคะแนนการบ้านทั้งหมดของคุณ และ/หรือ คะแนนจิตพิสัย ทั้งหมดได้ศูนย์คะแนน ซึ่งนั่นอาจเป็นปัจจัยของการตัดสินใจถอนกระบวนวิชานี้ของคุณและลงทะเบียนใหม่ในปีการศึกษาหน้า
- การลอกงานมาส่งต้องรับผิดชอบพร้อมกันทั้งกลุ่ม จะให้คนทำผิด รับผิดชอบแต่เพียงคนเดียวไม่ได้
- เพื่อนในกลุ่มที่เหลือมีหน้าที่ต้องตรวจสอบความถูกต้องและรับประกันผลงานว่าไม่ได้นำผลงานของกลุ่มอื่นมาส่ง

การบ้านนี้ให้นักศึกษา implement AVL Tree และ Splay Tree ตามที่เรียนในห้อง โดยใช้ Java โดยให้มีคลาสดังต่อไปนี้

1. ให้สร้าง class Node โดย class Node นี้มีคุณสมบัติดังต่อไปนี้
    a. ให้ Node แต่ละ Node สามารถบรรจุ data (key) ได้ค่า ๆ หนึ่ง โดยให้เป็นตัวแปรชนิด integer
    b. ให้ Node แต่ละ Node สามารถต่อกันเพื่อเป็นโครงสร้างข้อมูล Binary Search Tree ตามที่เรียนในห้องได้
    c. สมาชิกของ class Node ควรที่จะมี reference ชี้ไปยัง ลูกคนซ้าย (left child), ลูกคนขวา (right child) และ parent node
2. ให้ class Node มี 1 Constructor คือ Node(int data) ซึ่งทำหน้าที่ กำหนดค่าเริ่มต้นของ key จาก data
3. ให้ class Node มี operation ดังต่อไปนี้
    a. public static int height(Node node) ทำหน้าที่หาว่า Node node นี้อยู่ที่ความสูงที่เท่าไหร่เมื่อเทียบกับลูกที่อยู่ลึกที่สุด [อาจารย์ implement ให้เรียบร้อยแล้ว ใช้ได้เลย]
    b. public boolean isImbalance() ทำหน้าที่คืนค่าเพื่อที่จะบอกว่า node ปัจจุบันนี้ สมดุลด้วยความสูง (height balance) หรือไม่ นิยามความสมดุลด้วยความสูงของ Node ให้ใช้ตามที่เรียนในห้อง
4. ให้ศึกษา class BSTree ซึ่งทำหน้าที่บรรจุ Node ตามคุณสมบัติของ Binary Search Tree โดยอาจารย์จะ implement ฟังก์ชันส่วนใหญ่ให้หมดแล้ว ให้คุณเติมบางฟังก์ชันที่ไม่ครบถ้วน ดังต่อไปนี้
    a. ฟังก์ชันที่อาจารย์ implement ให้เรียบร้อยแล้ว ใช้ได้เลย
        - 2 constructor คือ public BSTree() และ public BSTree(Node root) โดย constructor ตัวที่สองทำหน้าที่กำหนด root node จาก calling function
        - ฟังก์ชัน printTree() ที่จะเรียกฟังก์ชันชื่อเดียวกันของ super class BTreePrinter เพื่อทำหน้าที่แสดงแผนภาพต้นไม้ออกมาอย่างคร่าว ๆ ได้
        - ฟังก์ชัน public Node find(int search_key) และ public static Node find(Node node, int search_key) เพื่อทำหน้าที่หา Node ที่บรรจุใน BSTree ที่มี key ดังที่ระบุ
        - ฟังก์ชัน public Node findMin() และ public static Node findMin(Node node) ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key น้อยที่สุด
        - ฟังก์ชัน public Node findMax() และ public static Node findMax(Node node) ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key มากที่สุด
        - ฟังก์ชัน public void insert(int key) และ public static void insert(Node node, int key) ทำหน้าที่สร้าง Node ใหม่ที่บรรจุค่า key แล้วนำไปต่อใน BST ตามที่เรียนในห้อง
        - ฟังก์ชัน public void delete(int key) และ public static void delete(Node node, int key) ทำหน้าที่ค้นหา Node ที่บรรจุอยู่ใน BST แล้วทำการแทนที่ด้วย Node ที่มีค่าน้อยที่สุดจาก Right subtree ตามที่เรียนในห้อง

b. ในการบ้านนี้คุณต้องเพิ่มฟังก์ชันดังต่อไปนี้ลงใน BSTree

- public void singleRotateFromLeft(Node y) ทำหน้าที่ หมุนลูกทางซ้ายของ Node y ขึ้นมาแทน Node y ตามที่เรียนในห้อง
- public void singleRotateFromRight(Node y) ทำหน้าที่ หมุนลูกทางขวาของ Node y ขึ้นมาแทน Node y ตามที่เรียนในห้อง
- public void doubleRotateFromLeft(Node y) ทำหน้าที่ หมุนหลานของ Node y จากทางด้านซ้ายใน ขึ้นมาแทน Node y ตามที่เรียนในห้อง
- public void doubleRotateFromRight(Node y) ทำหน้าที่ หมุนหลานของ Node y จากทางด้านขวาใน ขึ้นมาแทน Node y ตามที่เรียนในห้อง
- public static boolean isMergeable(Node r1, Node r2) ทำหน้าที่ ตรวจสอบว่า ต้นไม้ที่กำหนดให้อยู่ด้านซ้ายซึ่งมี root เป็น r1 จะสามารถรวมกับต้นไม้ที่มี root เป็น r2 ซึ่งกำหนดให้อยู่ด้านขวาได้หรือไม่ เงื่อนไขการรวมกันได้หรือไม่ คือ ข้อมูลทุกตัวของต้นไม้ด้านซ้ายจะต้องน้อยกว่าข้อมูลทุกตัวของต้นไม้ด้านขวา
- public void merge(BSTree tree2) ทำหน้าที่ รวมต้นไม้ปัจจุบัน (this) ซึ่งอยู่ด้านซ้าย กับต้นไม้ที่อยู่ด้านขวาที่ชื่อว่า tree2 เข้าด้วยกัน โดยฟังก์ชันนี้จะสร้าง root ใหม่ขึ้นมาได้เอง ตามที่เรียนในห้อง กำหนดให้ root ใหม่คือ Node ที่มี key เป็นค่าสูงที่สุด ของต้นไม้ปัจจุบัน (this) ถ้าต้นไม้ทั้งสองรวมกันไม่ได้ดังเงื่อนไขข้างบน ให้แจ้งให้ user ทราบว่า “All nodes in T1 must be smaller than all nodes from T2”
- public static Node mergeWithRoot(Node r1, Node r2, Node t) ทำหน้าที่ รวมต้นไม้ที่อยู่ด้านซ้ายซึ่งมี root node ชื่อว่า r1 และ ต้นไม้ที่อยู่ด้านขวาที่ชื่อว่า r2 เข้าด้วยกัน โดยกำหนดให้ใช้ Node t เป็น root ที่ใช้ในการเชื่อมต้นไม้ทั้งสองเข้าด้วยกัน ตามที่เรียนในห้อง ถ้าต้นไม้ทั้งสองรวมกันไม่ได้ดังเงื่อนไขข้างบน ให้แจ้งให้ user ทราบว่า “All nodes in T1 must be smaller than all nodes from T2”
- public NodeList split(int key) และ public static NodeList split(Node r, int key) ทำหน้าที่ แบ่งต้นไม้ปัจจุบัน (this) ออกเป็นสองต้น โดยใช้ key ที่กำหนดขึ้นมาเป็นตัวแบ่ง ต้นแรกให้มีสมาชิกทุกตัวมีค่าน้อยกว่าหรือเท่ากับ key ต้นที่สองมีสมาชิกมากกว่า key ผลลัพธ์ คือ root node ของต้นไม้ต้นแรกกับต้นที่สอง โดยให้ class NodeList ซึ่งมีสมาชิกคือ r1 และ r2 ชี้ไปที่ root ของต้นไม้ทั้งสองต้น ตามลำดับ อาจารย์จะ implement class NodeList และ public NodeList split(int key) ให้คุณทำส่วนเป็นที่ recursion ให้เสร็จ

5. เมื่อคุณสร้าง class BSTree ตามที่กำหนดเสร็จเรียบร้อยแล้ว ให้ทำการสร้าง class AVLTree ให้มีคุณสมบัติของ Height Balancing ตามที่เรียนในห้อง
    a. โค้ดดังต่อไปนี้ใน AVLTree จะต้องมีหน้าตาเหมือนกันกับ BSTree ให้คุณ copy โค้ดของฟังก์ชันดังต่อไปนี้มาเติมลงให้ AVLTree ให้สมบูรณ์
        - public void singleRotateFromLeft(Node y)
        - public void singleRotateFromRight(Node y)
        - public void doubleRotateFromLeft(Node y)
        - public void doubleRotateFromRight(Node y)
        - public static boolean isMergeable(Node r1, Node r2)
    b. ให้คุณ implement function เพิ่ม ดังต่อไปนี้
        - public static void rebalance(AVLTree tree, Node node) ทำหน้าที่ rebalance node ซึ่งเป็นส่วนหนึ่งใน tree กฎการ rebalance ให้เป็นไปตามที่เรียนในห้อง ฟังก์ชัน rebalance ควรที่จะทำการเรียกฟังก์ชัน singleRotateFromLeft(Node y), singleRotateFromRight(Node y), doubleRotateFromLeft(Node y), doubleRotateFromRight(Node y) ให้ถูกต้องตามสมควรแก่สถานการณ์
        - นอกจากนี้ หากฟังก์ชันทั้งสี่ดังกล่าวถูกเรียกโดย rebalance ให้คุณทำการ print ออกทาง Console อีกด้วย ด้วย pattern ดังตัวอย่างต่อไปนี้
            i. ถ้า Node 5 ถูกหมุนจากลูกทางด้านซ้าย ให้คุณ print ออกไปว่า “Perform SingleRotationFromLeft(Node 5)”
            ii. ถ้า Node 5 ถูกหมุนจากหลานทางด้านขวาใน ให้คุณ print ออกไปว่า “Perform DoubleRotationFromRight(Node 5)”
            iii. การเขียนโค้ดการ print นี้ให้ implement ในฟังก์ชัน rebalance
        - ผมได้แยกกรณีเอาไว้ให้คุณแล้วเพียงคุณเติมโค้ดที่ขาดหายไปเท่านั้นเอง ถ้าหากคุณไม่เข้าใจโค้ดอาจารย์ ผมอนุญาตให้คุณเขียน function นี้ขึ้นมาใหม่ได้เลย โดยไม่ต้องใช้ template function นี้ของอาจารย์
    c. เมื่อทำการ implement rebalance แล้ว ให้คุณปรับปรุงโค้ดดังต่อไปนี้ เพื่อให้ operation ดังกล่าวมีคุณสมบัติของ Self-balancing tree (AVL Tree) โดยปกติแล้วคุณแค่เติมให้มีการเรียกฟังก์ชัน rebalance ในไม่กี่บรรทัด แต่บาง operation คุณอาจจะต้องแก้โค้ดใหม่ และมีบาง operation คุณอาจจะไม่ต้องแก้ไขอะไรเลย ก็เป็นได้
        - public static void insert(AVLTree tree, Node node, int key)
        - public static void delete(AVLTree tree, Node node, int key)
        - public static Node mergeWithRoot(Node r1, Node r2, Node t)
        - public static NodeList split(Node r, int key)

- public void merge(AVLTree tree2)

6. เมื่อคุณสร้าง class SplayTree เพื่อให้มีคุณสมบัติของ Self-Adjusting ตามที่เรียนในห้อง
   a. ฟังก์ชันดังต่อไปนี้อาจารย์ได้ implement สำเร็จรูปไว้แล้ว สามารถใช้ได้เลย ไม่ต้องแก้ไขเพิ่มเติมอีก รวมทั้งไม่ต้องทำการ splay ด้วย ฟังก์ชันดังกล่าวได้แก่
      - Constructor สองตัว public SplayTree() และ public SplayTree(Node root)
      - public void printTree()
      - private Node findMin()
      - private static Node findMin(Node node)
      - public static Node findWithoutSplaying(Node node, int search_key)
      - private static Node find(Node node, int search_key)
   b. ให้คุณทำการ implement operation ดังต่อไปนี้
      - public void zig(Node node) ทำหน้าที่นำ node ขึ้นไปอีกหนึ่งระดับ โดยทำการสลับที่กับ parent ด้านบน หาก node นั้นเป็น root node ให้แจ้งแก่ user ว่า Cannot perform Zig operation on the root node คุณควรแยกกรณีออกเป็น สี่กรณี ได้แก่ (1) กรณีที่ node นั้นเป็นลูกคนซ้ายของ root node (2) กรณีที่ node นั้นเป็นลูกคนขวาของ root node (3) กรณีที่ node นั้นเป็นลูกคนซ้ายของ Node ใด ๆ ที่ไม่ใช่ root node (4) กรณีที่ node นั้นเป็นลูกคนขวาของ Node ใด ๆ ที่ไม่ใช่ root node
      - public void zigzig(Node node) ทำหน้าที่นำ node ขึ้นไปอีกสองระดับ ตามแนววงนอก ให้สลับที่ parent กับ grandparent ก่อน แล้วจึงสลับตัว node หลังสุด หากไม่เข้าใจ ให้กลับไปดูในสไลด์อาจารย์ ฟังก์ชันนี้ต้องเรียกฟังก์ชัน zig ในการนำ node ขึ้นไปหนึ่งระดับ
      - public void zigzag(Node node) ทำหน้าที่นำ node ขึ้นไปอีกสองระดับ ตามแนววงใน ให้สลับที่ตัวมันเอง กับ parent ก่อน แล้วจึงสลับที่กับ grandparent ทีหลัง หากไม่เข้าใจ ให้กลับไปดูในสไลด์อาจารย์ ฟังก์ชันนี้ต้องเรียกฟังก์ชัน zig ในการนำ node ขึ้นไปหนึ่งระดับ
      - public void splay(Node node) ทำหน้าที่นำ node ขึ้นไปไว้ตรง root แบบ recursive โดยฟังก์ชันนี้สามารถประเมินได้ว่าควรจะนำ node ขึ้นไปข้างบนได้อย่างไร หากนำขึ้นไปในสองระดับได้ ให้เรียกใช้ฟังก์ชัน zigzig หรือ zigzag ตามสมควร หากนำขึ้นไปได้แค่หนึ่งระดับก็ให้เรียกใช้ฟังก์ชัน zig
   c. เมื่อคุณสร้างฟังก์ชัน splay สำเร็จแล้ว ให้คุณทำการปรับปรุงฟังก์ชันดังต่อไปนี้
      - public Node find(int search_key) โดยทุก ๆ ครั้งที่ Node ถูกหาเจอ Node นั้นจะถูก splay ขึ้นไปอยู่ที่ root
      - private static void insert(SplayTree tree, Node node, int key) หลังจาก insert ตามปกติแล้ว ให้ Node ที่เพิ่งถูกบรรจุเข้ามาใหม่ ถูก splay ขึ้นไปอยู่บน root

- public void delete(int key) ให้ใช้อัลกอริทึมดังต่อไปนี้ เพื่อลบ Node ใด ๆ สำหรับ SplayTree (วิธีใหม่ ซึ่งแตกต่างจากการลบ Node ใน BST)
    i. หา Node ที่บรรจุ key ให้เจอ (ถ้าไม่เจอ ให้แจ้ง Error) แล้ว splay Node นั้นขึ้นมาที่ root
    ii. สร้างต้นไม้ต้นใหม่ โดยแยกลูกด้านขวาของ root (ของต้นไม้ต้นเก่า) ออกมาเป็น root Node ของต้นไม้ต้นใหม่
    iii. หา Node ที่มีค่าน้อยที่สุดของ ต้นไม้ต้นใหม่ แล้ว splay มันขึ้นมาที่ root
    iv. นำลูกด้านซ้ายของต้นไม้ต้นเก่ามาต่อเป็นลูกด้านซ้ายของ root ของต้นไม้ต้นใหม่
    v. แทนที่ root ของต้นไม้ต้นเก่า (this.root) ด้วย root ของต้นไม้ต้นใหม่

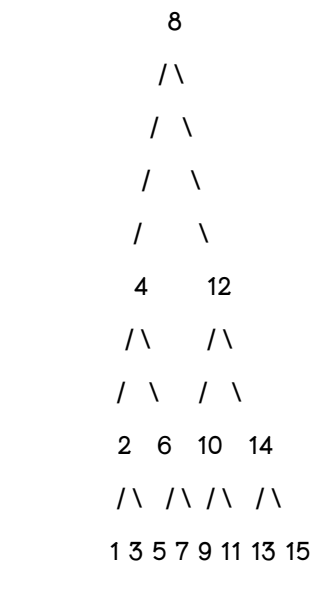
7. ตัวอย่างการทำงาน

Java code

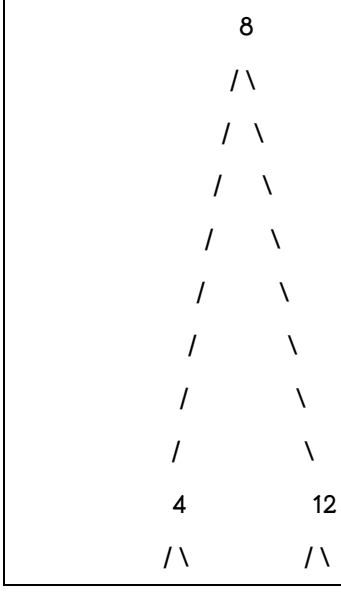
```
public static void main(String[] args) {
        BSTree tree;
        tree = generateTree1();
        tree.printTree();
        System.out.println("---- Test1 singleRotateFromLeft at Lv 3 ----");
        tree.singleRotateFromLeft(tree.find(6));
        tree.singleRotateFromLeft(tree.find(2));
        tree.printTree();
        System.out.println("---- Test2 singleRotateFromRight at Lv 3 ----");
        tree = generateTree1();
        tree.singleRotateFromRight(tree.find(10));
        tree.singleRotateFromRight(tree.find(14));
        tree.printTree();
        System.out.println("---- Test3 singleRotateFromLeft at Lv 2 ----");
        tree = generateTree1();
        tree.singleRotateFromLeft(tree.find(4));
        tree.printTree();
        System.out.println("---- Test4 singleRotateFromRight at Lv 2 ----");
        tree = generateTree1();
        tree.singleRotateFromRight(tree.find(12));
        tree.printTree();
```

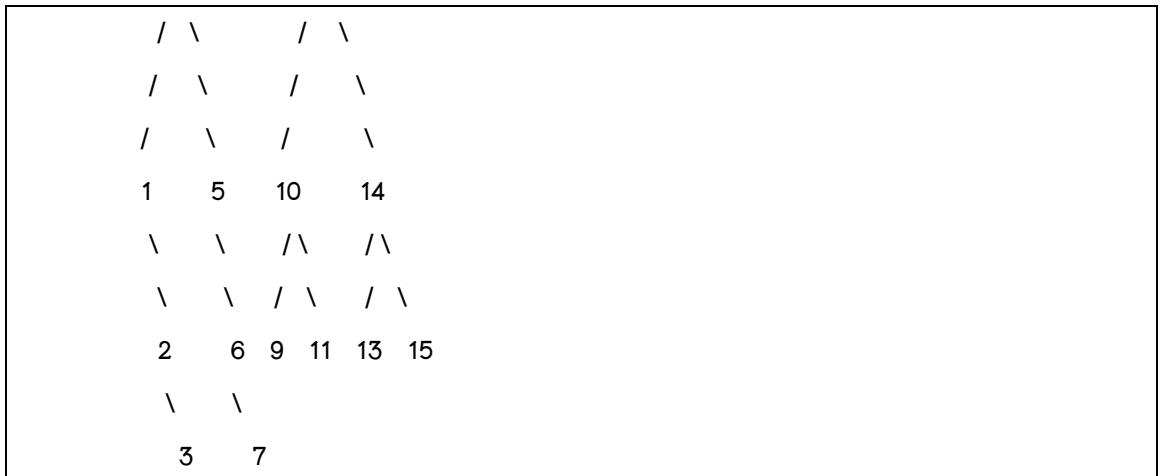
```
        System.out.println("---- Test5 singleRotateFromLeft at Lv 1 ----");
        tree.singleRotateFromLeft(tree.find(8));
        tree.printTree();
        System.out.println("---- Test6 singleRotateFromRight at Lv 1 ----");
        tree.singleRotateFromRight(tree.find(8));
        tree.printTree();
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

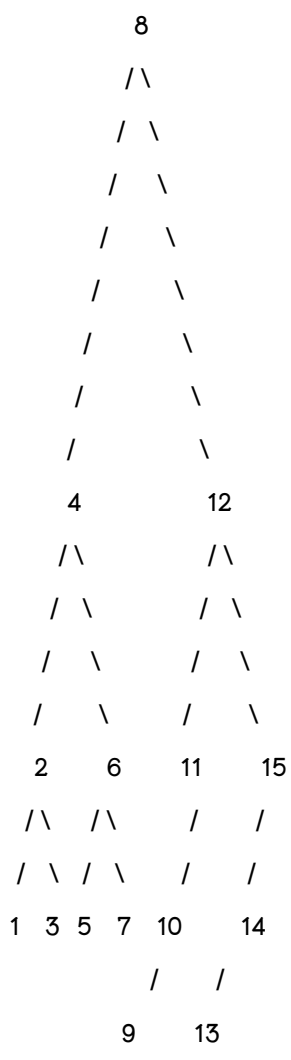



```
---- Test1 singleRotateFromLeft at Lv 3 ----
```

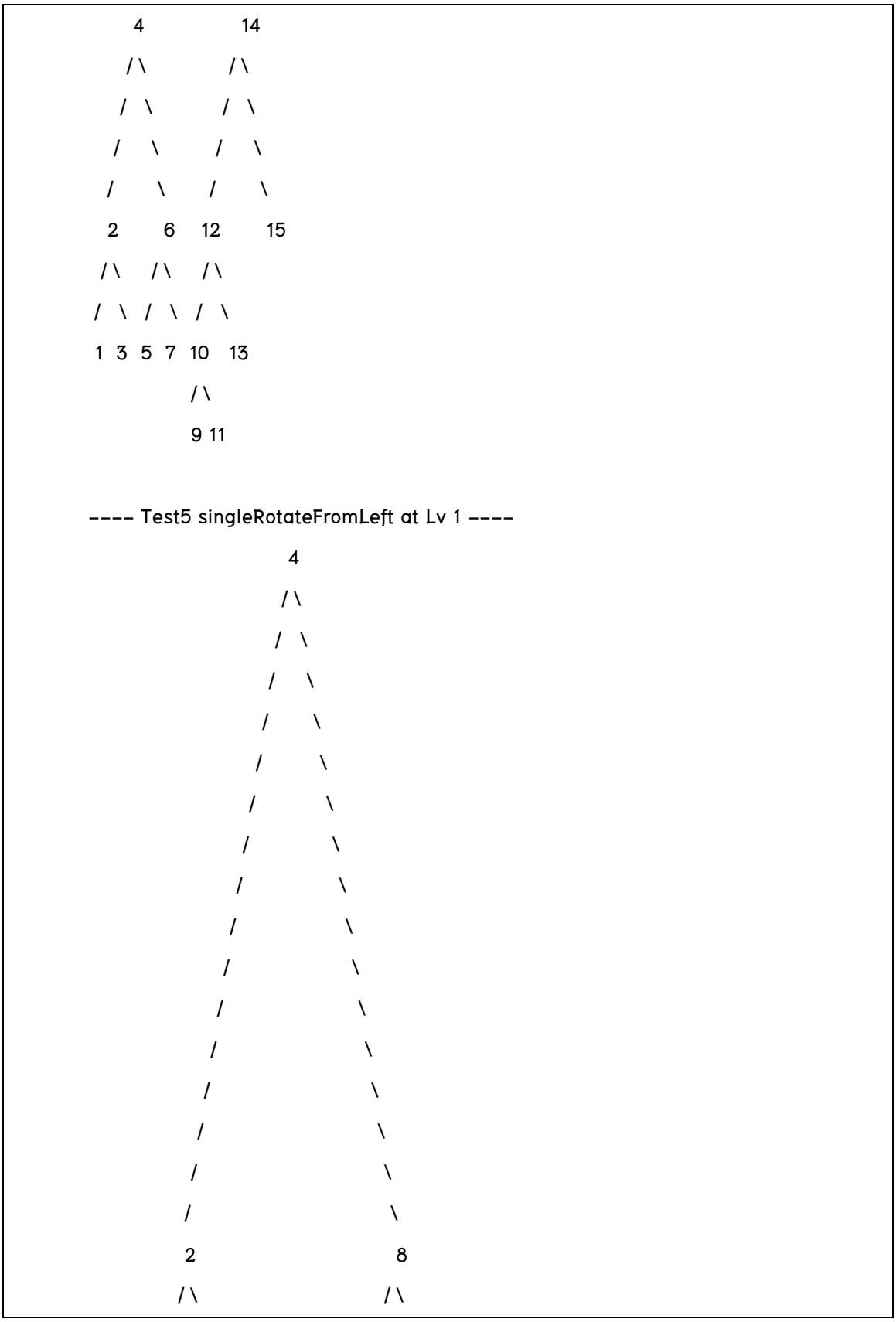




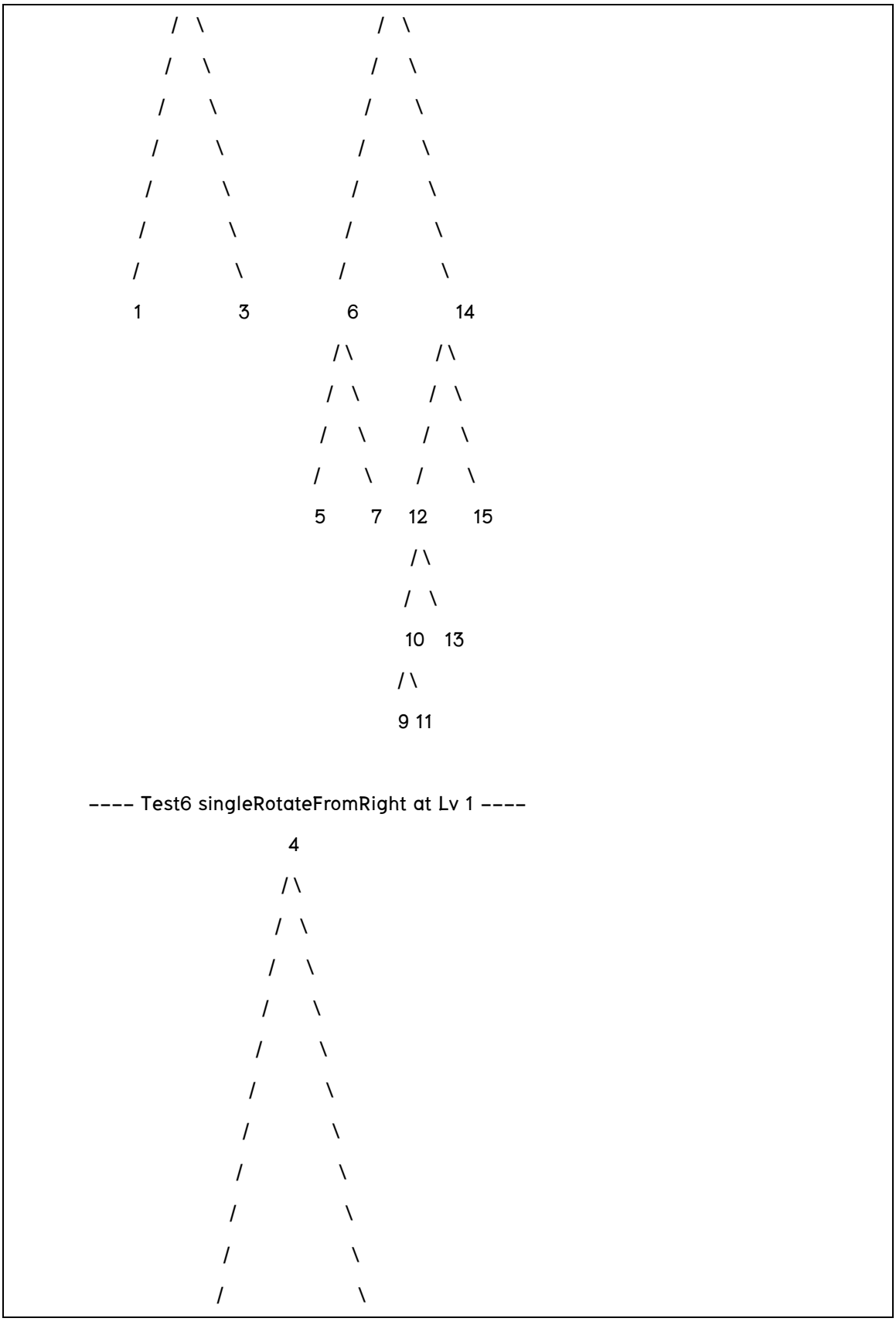
---- Test2 singleRotateFromRight at Lv 3 ----



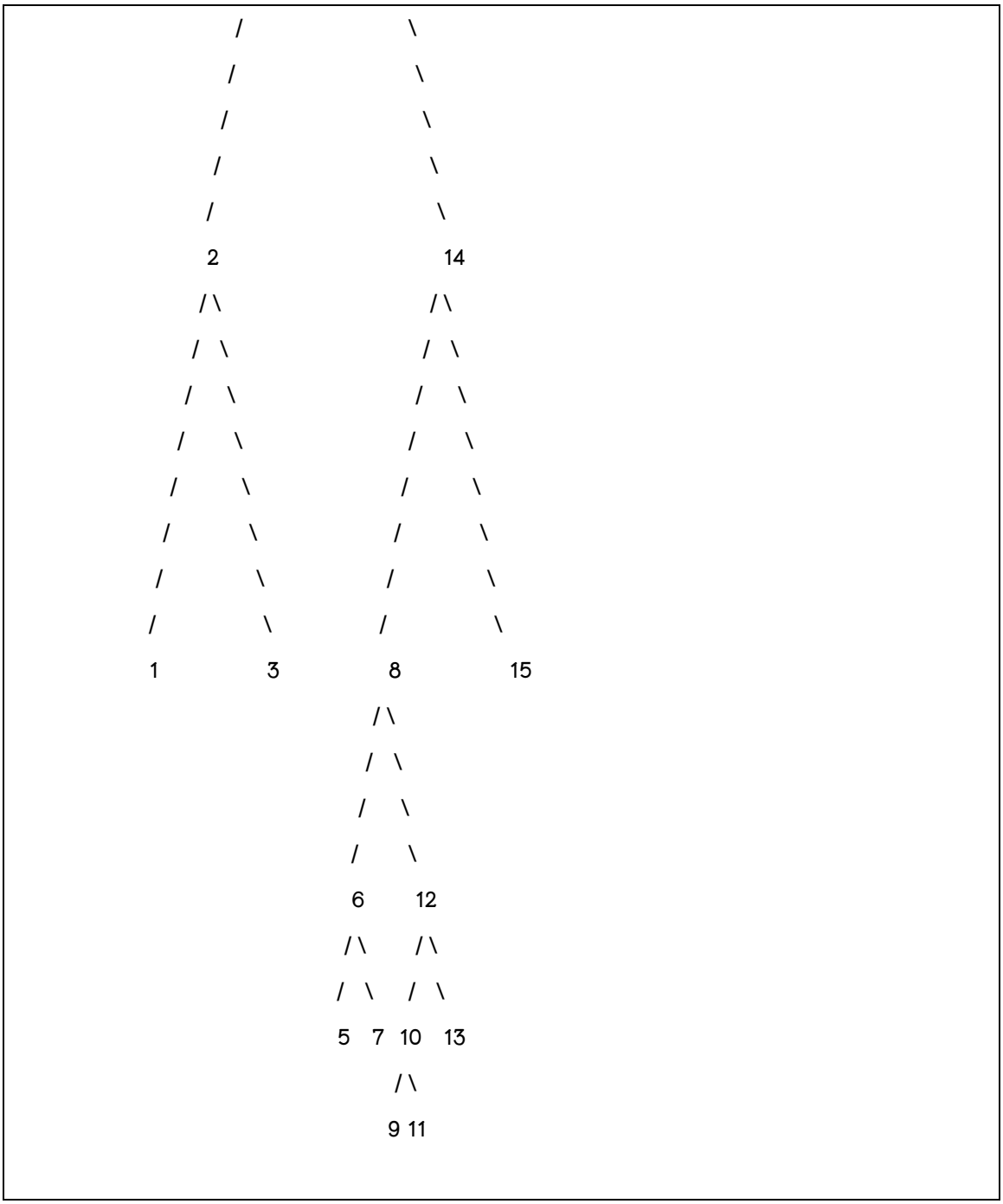
```
---- Test3 singleRotateFromLeft at Lv 2 ----
                8
               / \
              /   \
             /     \
            /       \
           /         \
          /           \
         /             \
        /               \
       2                 12
      / \               / \
     /   \             /   \
    /     \           /     \
   /       \         /       \
  1         4       10       14
           / \     / \       / \
          /   \   /   \     /   \
         3     6 9     11  13    15
              / \
             5   7


---- Test4 singleRotateFromRight at Lv 2 ----
                8
               / \
              /   \
             /     \
            /       \
           /         \
          /           \
         /             \
        /               \
```



---- Test5 singleRotateFromLeft at Lv 1 ----



---- Test6 singleRotateFromRight at Lv 1 ----



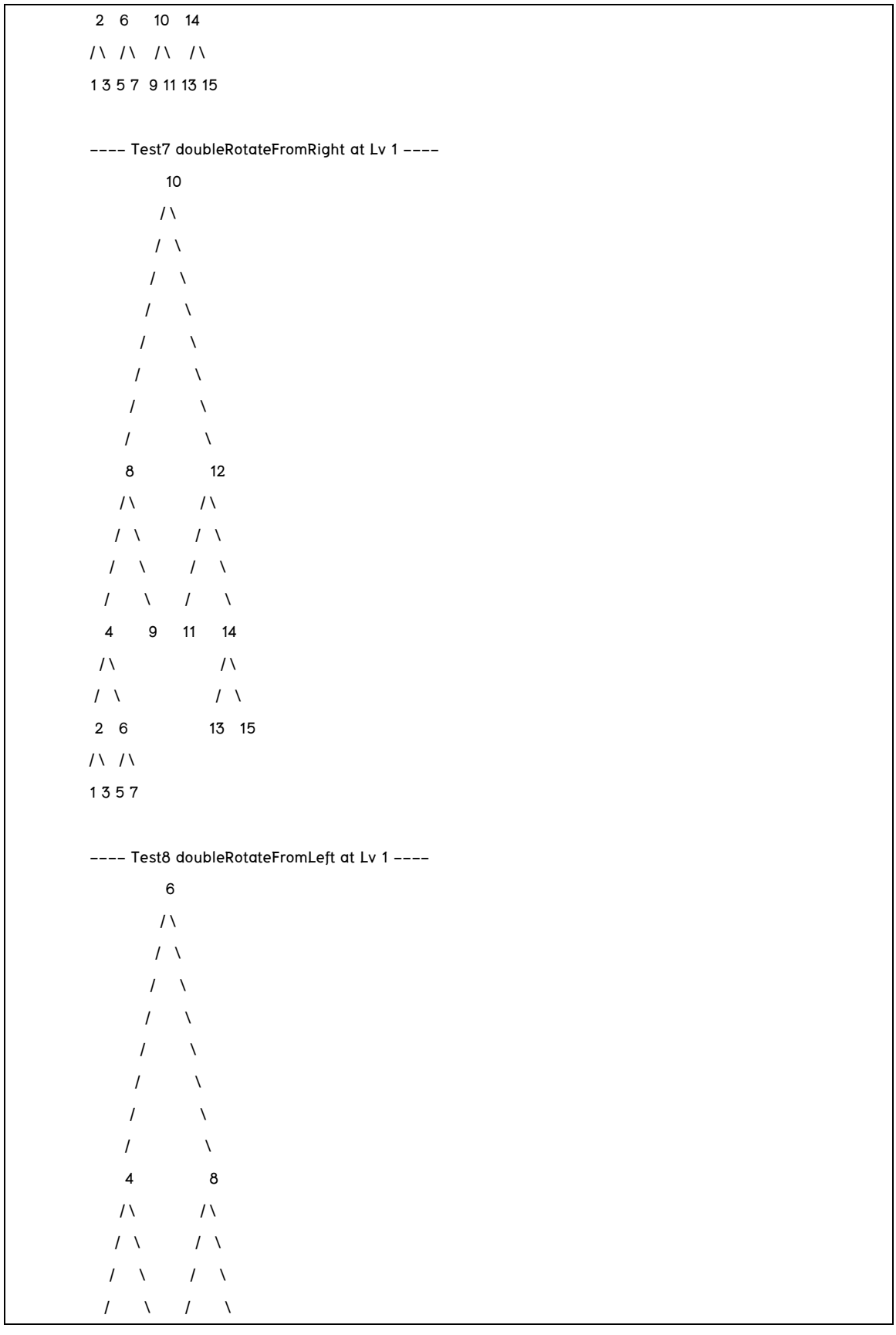
Java code

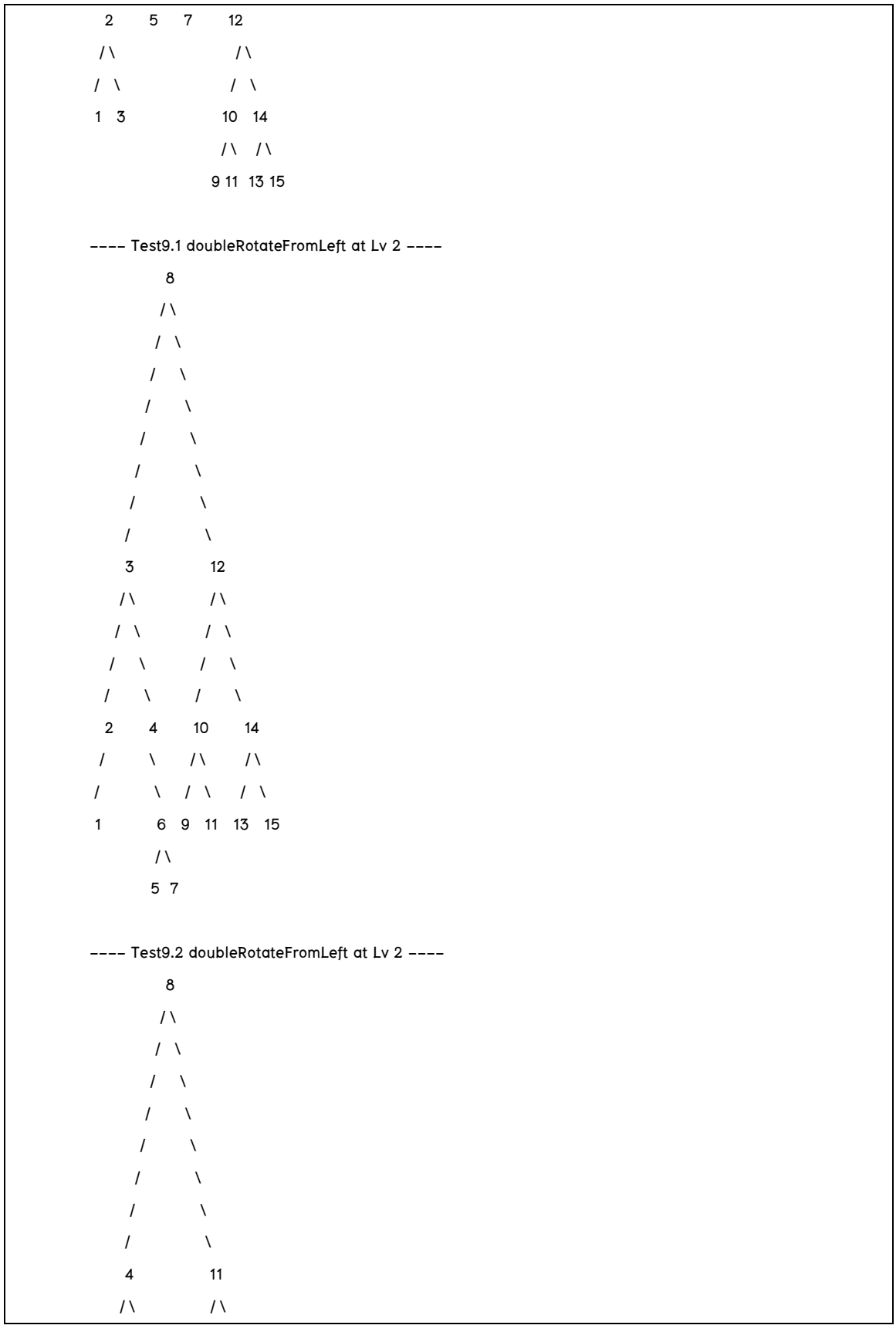
```
public static void main(String[] args) {
        BSTree tree;
        tree = generateTree1();
```

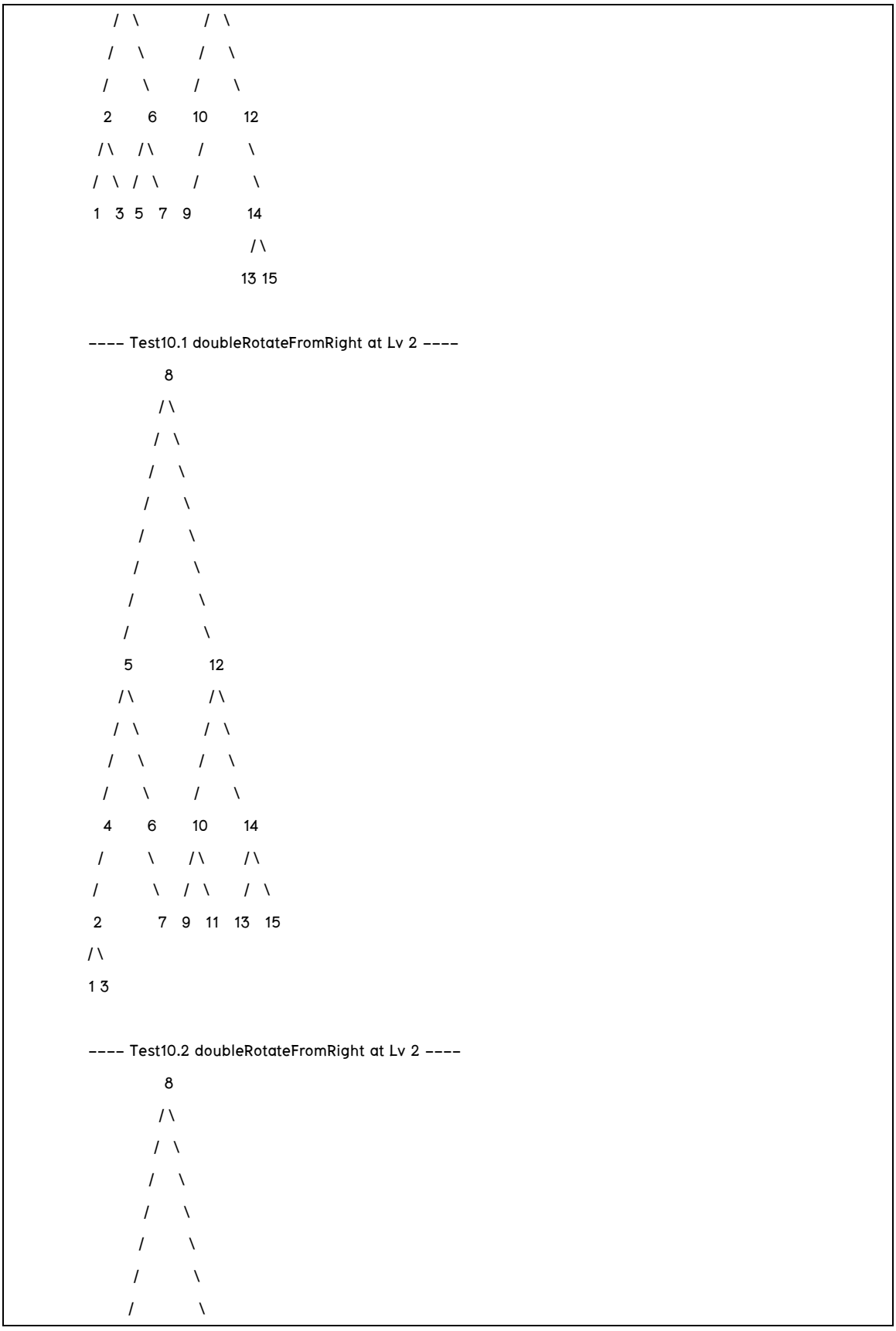
```
        tree.printTree();
        System.out.println("---- Test7 doubleRotateFromRight at Lv 1 ----");
        tree.doubleRotateFromRight(tree.find(8));
        tree.printTree();
        System.out.println("---- Test8 doubleRotateFromLeft at Lv 1 ----");
        tree=generateTree1();
        tree.doubleRotateFromLeft(tree.find(8));
        tree.printTree();
        System.out.println("---- Test9.1 doubleRotateFromLeft at Lv 2 ----");
        tree=generateTree1();
        tree.doubleRotateFromLeft(tree.find(4));
        tree.printTree();
        System.out.println("---- Test9.2 doubleRotateFromLeft at Lv 2 ----");
        tree=generateTree1();
        tree.doubleRotateFromLeft(tree.find(12));
        tree.printTree();
        System.out.println("---- Test10.1 doubleRotateFromRight at Lv 2 ----");
        tree=generateTree1();
        tree.doubleRotateFromRight(tree.find(4));
        tree.printTree();
        System.out.println("---- Test10.2 doubleRotateFromRight at Lv 2 ----");
        tree=generateTree1();
        tree.doubleRotateFromRight(tree.find(12));
        tree.printTree();
}
```

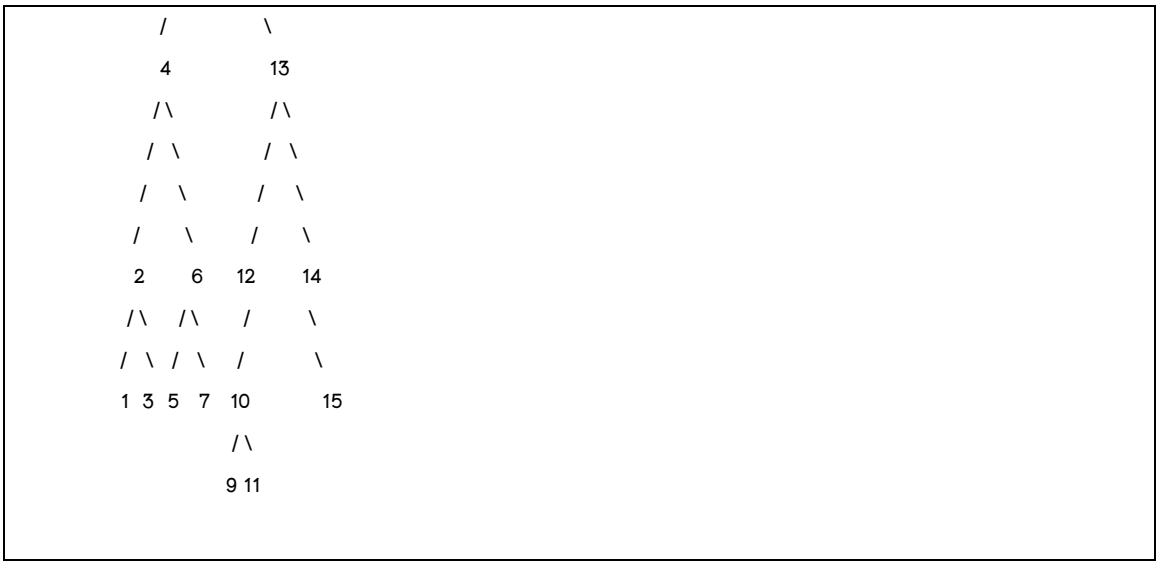
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```
              8
             / \
            /   \
           /     \
          /       \
         4         12
        / \       / \
       /   \     /   \
```


2 6 10 14
/ \ / \ / \ / \
1 3 5 7 9 11 13 15
---- Test7 doubleRotateFromRight at Lv 1 ----
10
8 12
4 9 11 14
2 6 13 15
1 3 5 7
---- Test8 doubleRotateFromLeft at Lv 1 ----
6
4 8


2 5 7 12
1 3 10 14
9 11 13 15
---- Test9.1 doubleRotateFromLeft at Lv 2 ----
8
3 12
2 4 10 14
1 6 9 11 13 15
5 7
---- Test9.2 doubleRotateFromLeft at Lv 2 ----
8
4 11


/ \ / \
/ \ / \
/ \ / \
2 6 10 12
/ \ / \ / \
/ \ / \ / \
1 3 5 7 9 14
/ \
13 15
---- Test10.1 doubleRotateFromRight at Lv 2 ----
8
/ \
/ \
/ \
/ \
/ \
/ \
/ \
5 12
/ \ / \
/ \ / \
/ \ / \
/ \ / \
4 6 10 14
/ \ / \ / \
/ \ / \ / \
2 7 9 11 13 15
/ \
1 3
---- Test10.2 doubleRotateFromRight at Lv 2 ----
8
/ \
/ \
/ \
/ \
/ \
/ \
/ \



Java code

```
public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        int[] keyList = {5, 2, 6, 1, 3, 4};
        for (int i=0; i<keyList.length; i++)
            tree.insert(keyList[i]);
        tree.printTree();

        keyList = new int[]{4, 3, 7, 6, 8, 5};
        tree = new AVLTree();
        for (int i=0; i<keyList.length; i++)
            tree.insert(keyList[i]);
        tree.printTree();

        tree.insert(9);
        tree.printTree();
        tree.insert(2);
        tree.insert(1);
        tree.printTree();
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```
Perform DoubleRotationFromLeft(Node 5)
   3
  / \
 /   \
 2   5
/   / \
1   4 6


Perform DoubleRotationFromRight(Node 4)
   6
  / \
 /   \
 4   7
/ \   \
3 5   8


Perform SingleRotationFromRight(Node 7)
   6
  / \
 /   \
 4   8
/ \  / \
3 5 7 9


Perform SingleRotationFromLeft(Node 3)
       6
      / \
     /   \
    /     \
   /       \
   4       8
  / \     / \
 /   \   /   \
```

```
 2   5 7   9
/ \
1 3
```

Java code [อ้างอิงตัวอย่างที่เรียนในห้อง ถ้าคุณอยากรู้ว่า Tree นี้หน้าตาเป็นอย่างไรให้ไปดูใน Slide]

```
public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        int[] keyList = {51, 30, 69, 18, 42, 63, 87, 12, 24, 36, 45, 57, 66, 81,
            93, 15, 21, 27, 33, 39, 48, 54, 60, 75, 84, 90, 96, 72, 78};
        for (int i=0; i<keyList.length; i++)
            tree.insert(keyList[i]);
        tree.insert(73); // must perform SingleRotationFromLeft(Node 81)
        tree.insert(77); // must perform DoubleRotationFromLeft(Node 87)
        tree.insert(76); // must perform SingleRotationFromLeft(Node 78)
        tree.insert(80); // must perform DoubleRotationFromRight(Node 69)
        tree.insert(74); // must perform SingleRotationFromRight(Node 72)
        tree.insert(64); // do nothing
        tree.insert(55); // must perform SingleRotationFromLeft(Node 69)
        tree.insert(70); // must perform DoubleRotationFromRight(Node 51)
}
```

Output (ต้นไม้หน้าตาเป็นอย่างไรให้ไปดูใน Slide ที่เรียนในห้อง)

```
Perform SingleRotationFromLeft(Node 81)
Perform DoubleRotationFromLeft(Node 87)
Perform SingleRotationFromLeft(Node 78)
Perform DoubleRotationFromRight(Node 69)
Perform SingleRotationFromRight(Node 72)
Perform SingleRotationFromLeft(Node 69)
Perform DoubleRotationFromRight(Node 51)
```

Java code [อ้างอิงตัวอย่างที่เรียนในห้อง ถ้าคุณอยากรู้ว่า Tree นี้หน้าตาเป็นอย่างไรให้ไปดูใน Slide]

```
public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        int[] keyList = {21, 8, 34, 3, 16, 26, 42, 2, 5, 11, 18, 23, 31, 37, 47,
            1, 4, 6, 9, 13, 17, 19, 22, 24, 28, 33, 35, 40, 45, 52, 7, 10, 12,
            14, 20, 25, 27, 30, 32, 36, 38, 41, 43, 46, 49, 53, 15, 29, 39, 44,
            48, 51, 54, 50};
        for (int i=0; i<keyList.length; i++)
            tree.insert(keyList[i]);
        tree.delete(1);
}
```

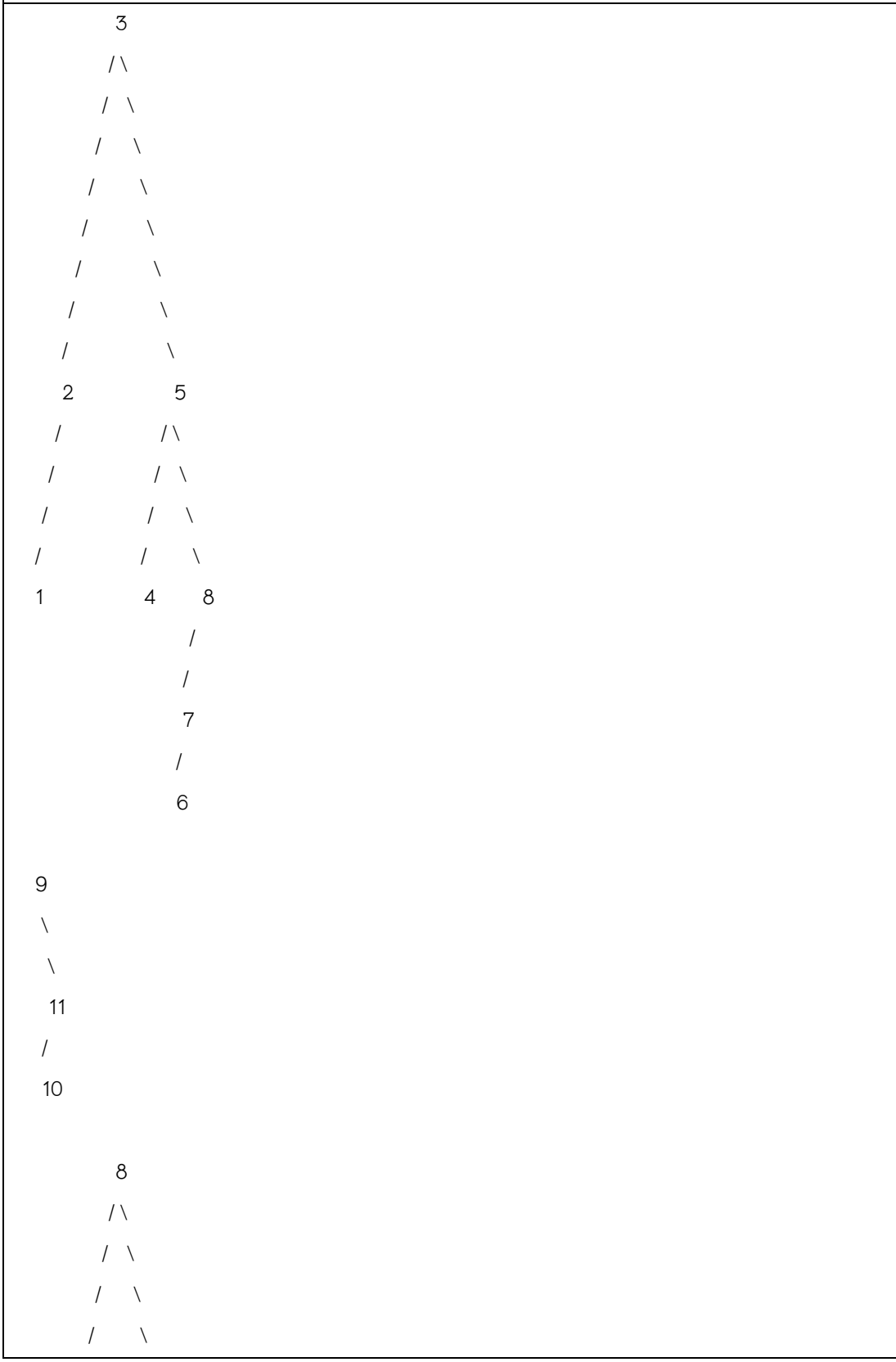
Output (ต้นไม้หน้าตาเป็นอย่างไรให้ไปดูใน Slide ที่เรียนในห้อง)

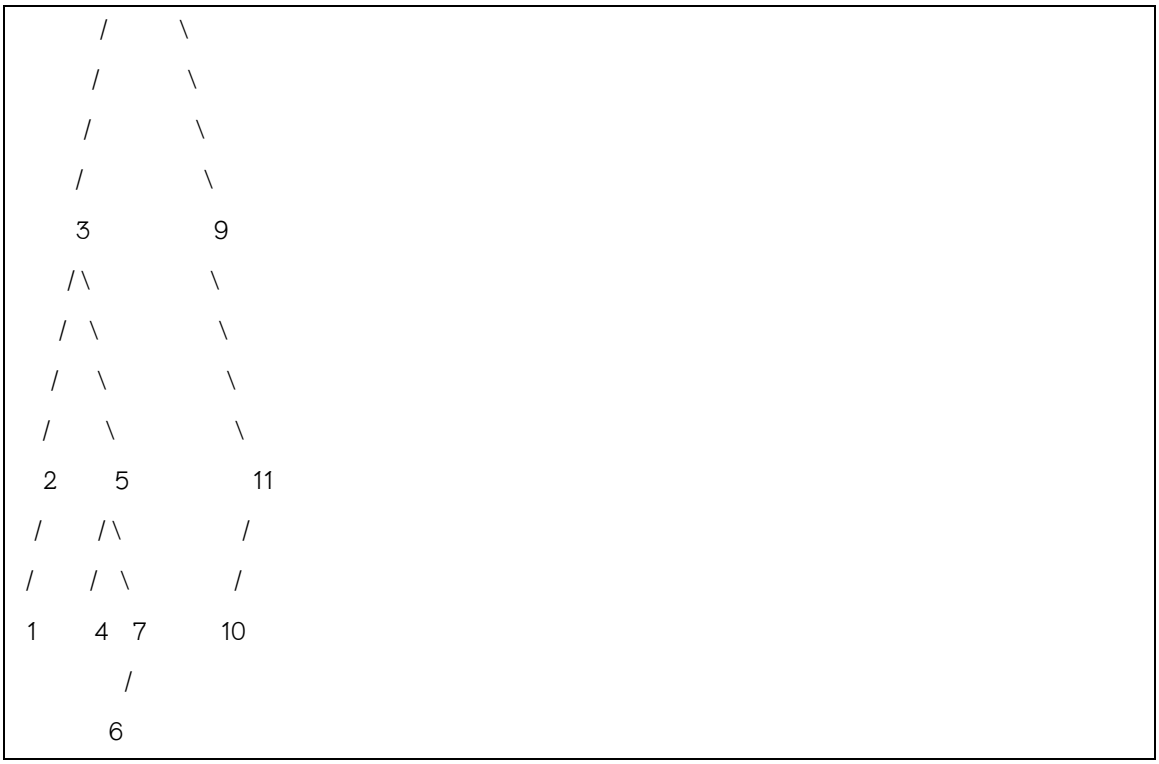
```
Perform SingleRotationFromRight(Node 3)
Perform DoubleRotationFromRight(Node 8)
Perform SingleRotationFromRight(Node 21)
```

Java code

```
public static void main(String[] args) {
        BSTree tree1 = new BSTree();
        int[] keyList = {3, 2, 5, 1, 4, 8, 7, 6};
        for (int i=0; i<keyList.length; i++)
            tree1.insert(keyList[i]);
        BSTree tree2 = new BSTree();
        keyList = new int[]{9, 11, 10};
        for (int i=0; i<keyList.length; i++)
            tree2.insert(keyList[i]);
        tree1.printTree();
        tree2.printTree();

        tree1.merge(tree2);
        tree1.printTree();
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)





Java code

```
public static void main(String[] args) {
        AVLTree tree1 = new AVLTree();
        int[] keyList = {4, 2, 6, 1, 3, 5, 8, 7, 9};
        for (int i=0; i<keyList.length; i++)
            tree1.insert(keyList[i]);
        System.out.println("Tree 1");
        tree1.printTree();
        AVLTree tree2 = new AVLTree();
        keyList = new int[]{12, 11};
        for (int i=0; i<keyList.length; i++)
            tree2.insert(keyList[i]);
        System.out.println("Tree 2");
        tree2.printTree();
        System.out.println("Try to merge tree1 to (the right of) tree2...");
        tree2.merge(tree1);
        tree2.printTree();
```

```
        System.out.println("Try to merge tree2 to (the right of) tree1...");
        tree1.merge(tree2);
        tree1.printTree();
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```
 Tree 1
       4
      / \
     /   \
    /     \
   /       \
   2       6
  / \     / \
 /   \   /   \
 1   3   5   8
              / \
             7 9


Tree 2
 12
/
11


Try to merge tree1 to (the right of) tree2...
All nodes in T1 must be smaller than all nodes from T2
 12
/
11


Try to merge tree2 to (the right of) tree1...
Perform DoubleRotationFromRight(Node 4)
       6
      / \
```

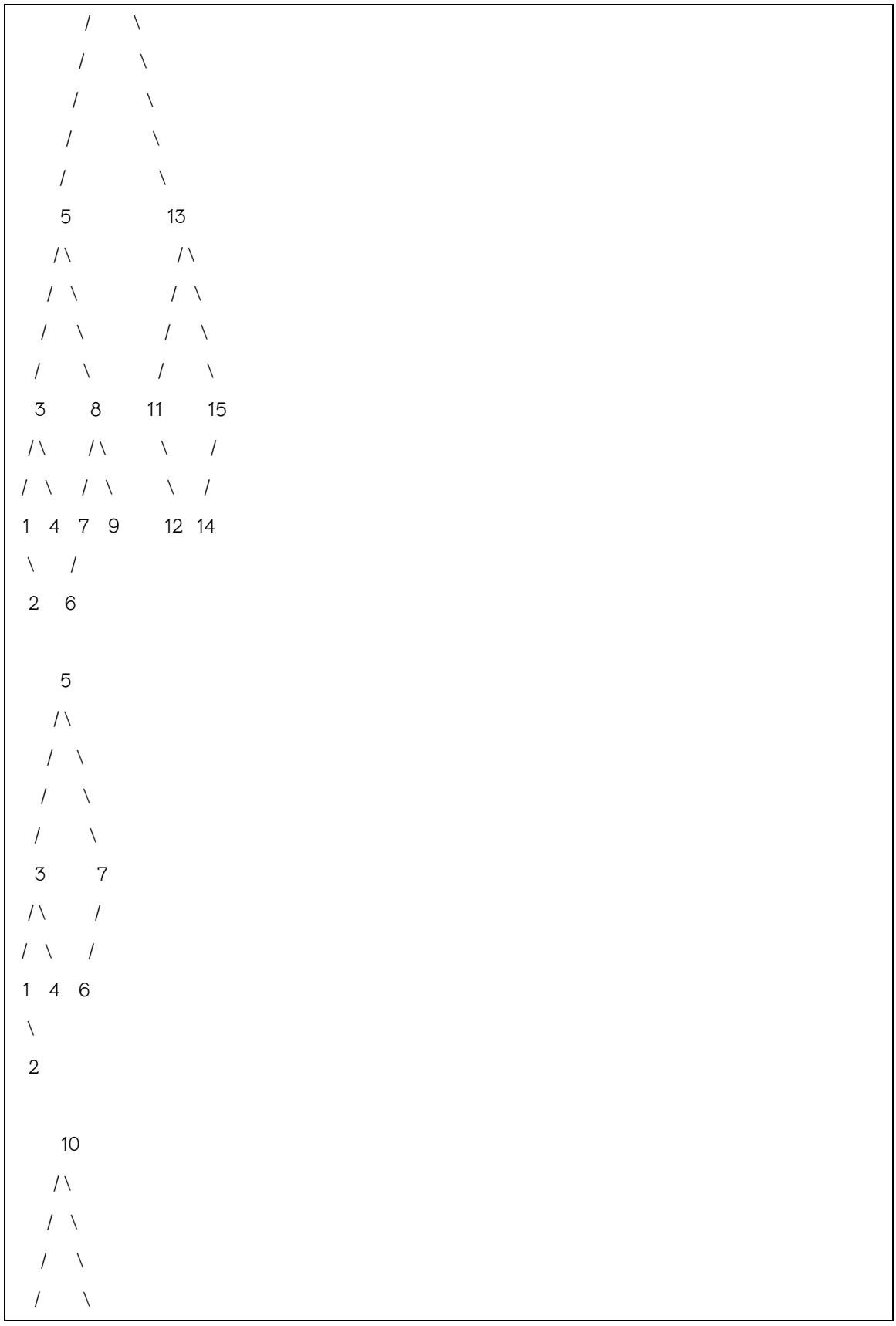
```
     /   \
    /     \
   /       \
   4       9
  / \     / \
 /   \   /   \
 2   5   8   12
/ \     /   /
1 3     7   11
```

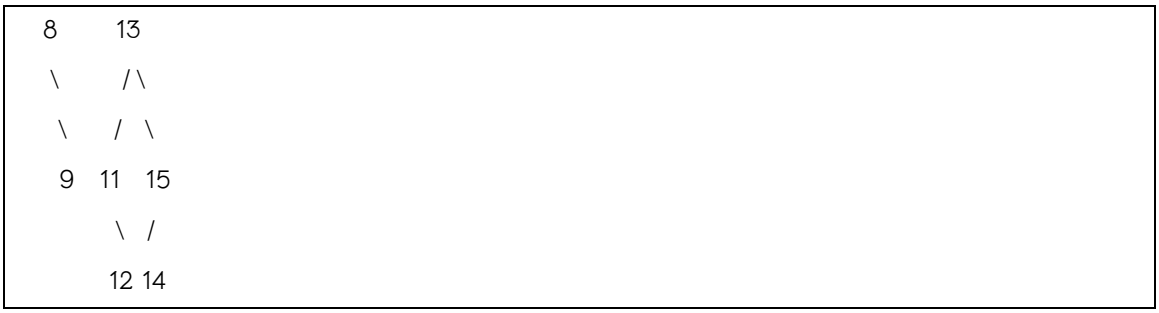
Java code

```
public static void main(String[] args) {
        AVLTree tree1 = new AVLTree();
        int[] keyList = {1, 15, 3, 13, 5, 11, 9, 10, 8, 4, 12, 7, 2, 6, 14};
        for (int i=0; i<keyList.length; i++)
            tree1.insert(keyList[i]);
        tree1.printTree();
        NodeList list = tree1.split(7);
        (new AVLTree(list.r1)).printTree();
        (new AVLTree(list.r2)).printTree();
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```
 Perform DoubleRotationFromRight(Node 1)
Perform SingleRotationFromLeft(Node 15)
Perform DoubleRotationFromRight(Node 3)
Perform DoubleRotationFromLeft(Node 11)
Perform SingleRotationFromLeft(Node 13)
Perform SingleRotationFromRight(Node 5)
Perform SingleRotationFromLeft(Node 9)
              10
             / \
            /   \
           /     \
```

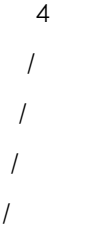

5 13
3 8 11 15
1 4 7 9 12 14
2 6
5
3 7
1 4 6
2
10

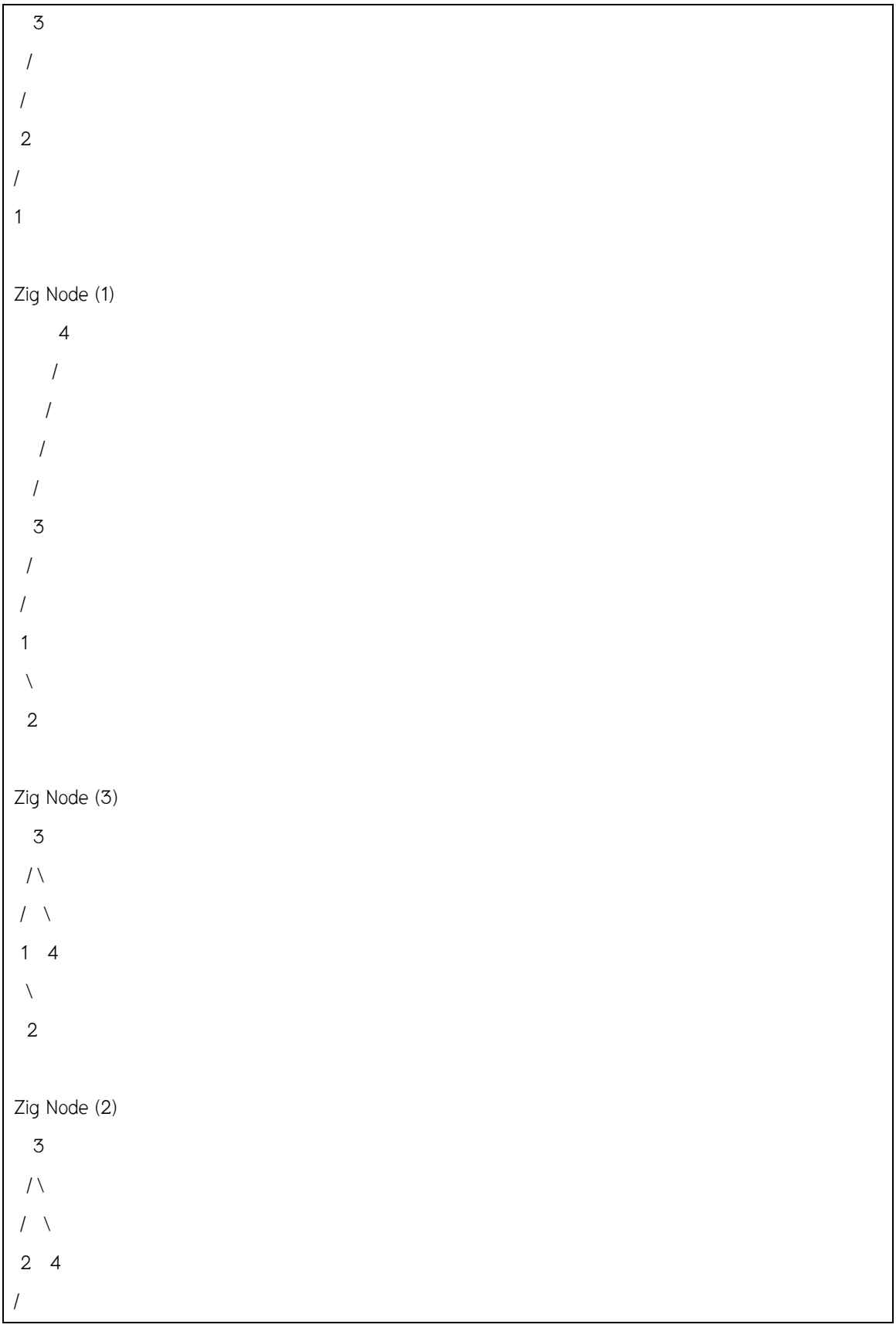


Java code

```
public static void main(String[] args) {
        SplayTree tree1 = new SplayTree();
        int[] keyList = {1, 2, 3, 4};
        for (int i = 0; i < keyList.length; i++) {
            tree1.insert(keyList[i]);
        }
        tree1.printTree();
        System.out.println("Zig Node (1)");
        tree1.zig(tree1.findWithoutSplaying(1));
        tree1.printTree();
        System.out.println("Zig Node (3)");
        tree1.zig(tree1.findWithoutSplaying(3));
        tree1.printTree();
        System.out.println("Zig Node (2)");
        tree1.zig(tree1.findWithoutSplaying(2));
        tree1.printTree();
        System.out.println("Zig Node (4)");
        tree1.zig(tree1.findWithoutSplaying(4));
        tree1.printTree();
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)




3
2
1
Zig Node (1)
4
3
1
2
Zig Node (3)
3
1
4
2
Zig Node (2)
3
2
4

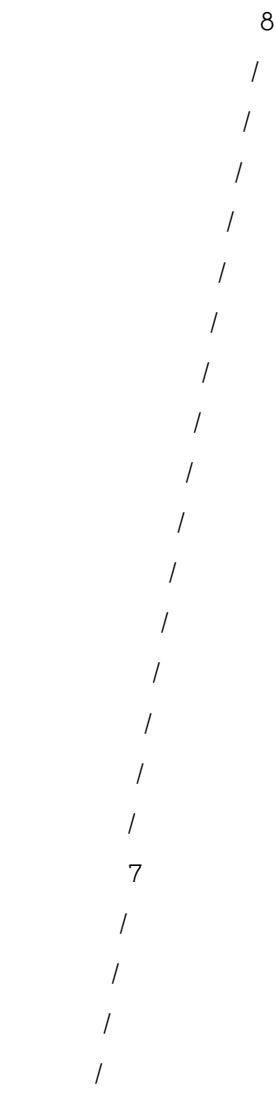
```
1


Zig Node (4)
        4
       /
      /
     /
    /
   /
   3
  /
 /
 2
/
1
```

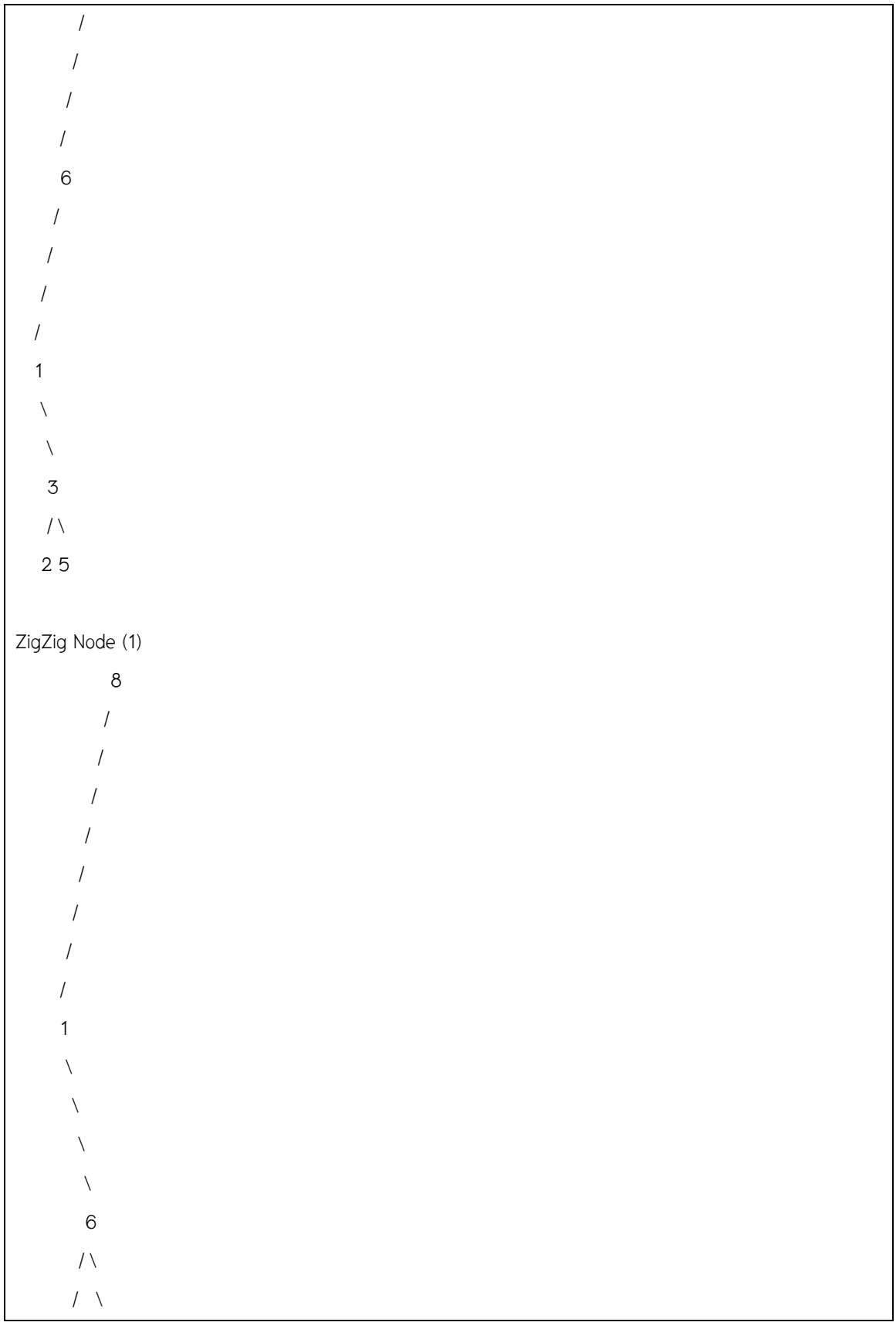
Java code

```
public static void main(String[] args) {
        SplayTree tree1 = new SplayTree();
        int[] keyList = {5, 7, 2, 3, 1, 6, 8};
        for (int i = 0; i < keyList.length; i++) {
            tree1.insert(keyList[i]);
        }
        tree1.printTree();
        System.out.println("ZigZig Node (1)");
        tree1.zigzig(tree1.findWithoutSplaying(1));
        tree1.printTree();
        System.out.println("ZigZag Node (5)");
        tree1.zigzag(tree1.findWithoutSplaying(5));
        tree1.printTree();
        System.out.println("ZigZag Node (5)");
        tree1.zigzag(tree1.findWithoutSplaying(5));
        tree1.printTree();
        System.out.println("ZigZag Node (7)");
        tree1.zigzag(tree1.findWithoutSplaying(7));
```

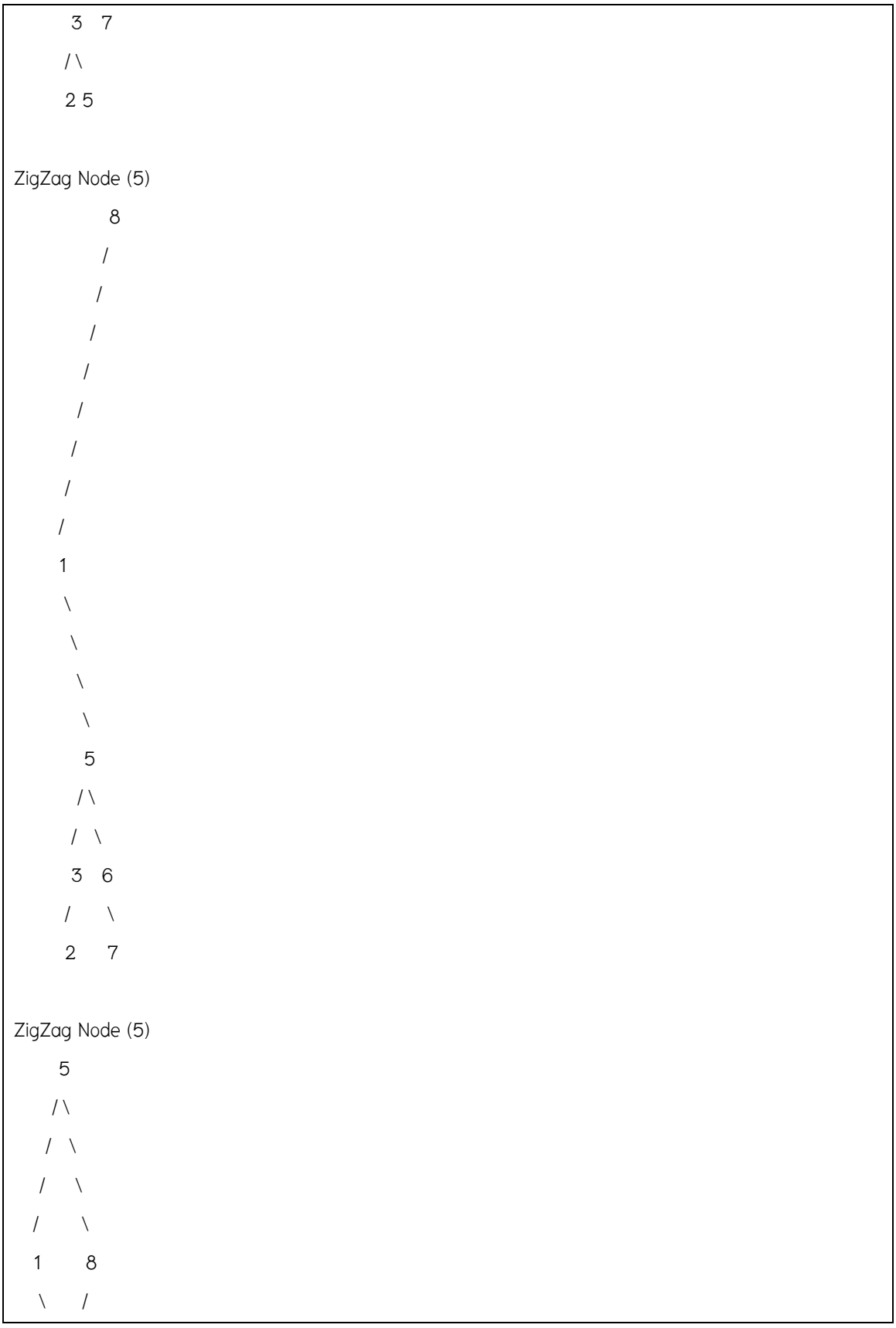
```
        tree1.printTree();
        System.out.println("ZigZag Node (2)");
        tree1.zigzag(tree1.findWithoutSplaying(2));
        tree1.printTree();
        System.out.println("ZigZag Node (3)");
        tree1.zigzag(tree1.findWithoutSplaying(3));
        tree1.printTree();
        System.out.println("ZigZig Node (7)");
        tree1.zigzig(tree1.findWithoutSplaying(7));
        tree1.printTree();
}
```

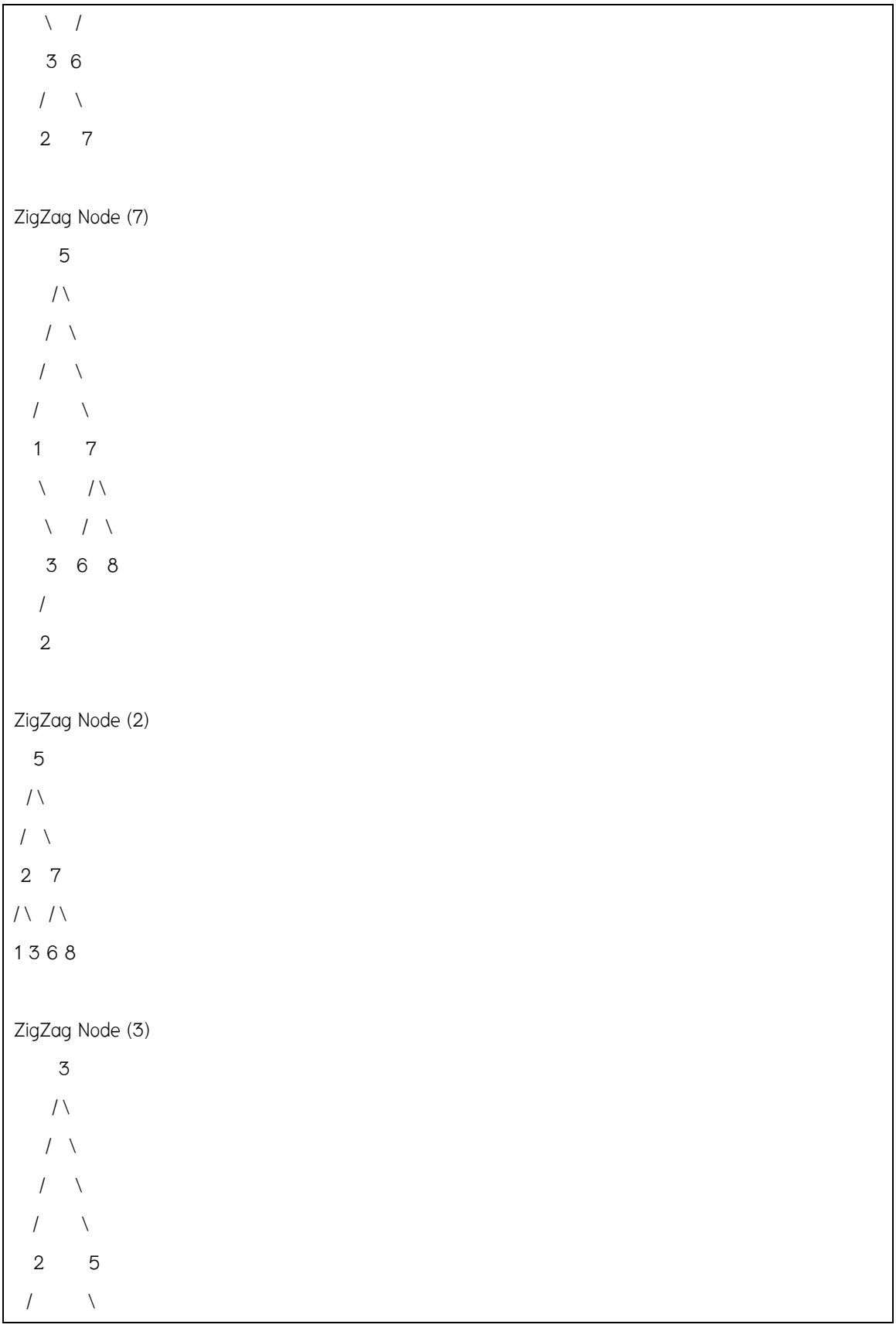
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

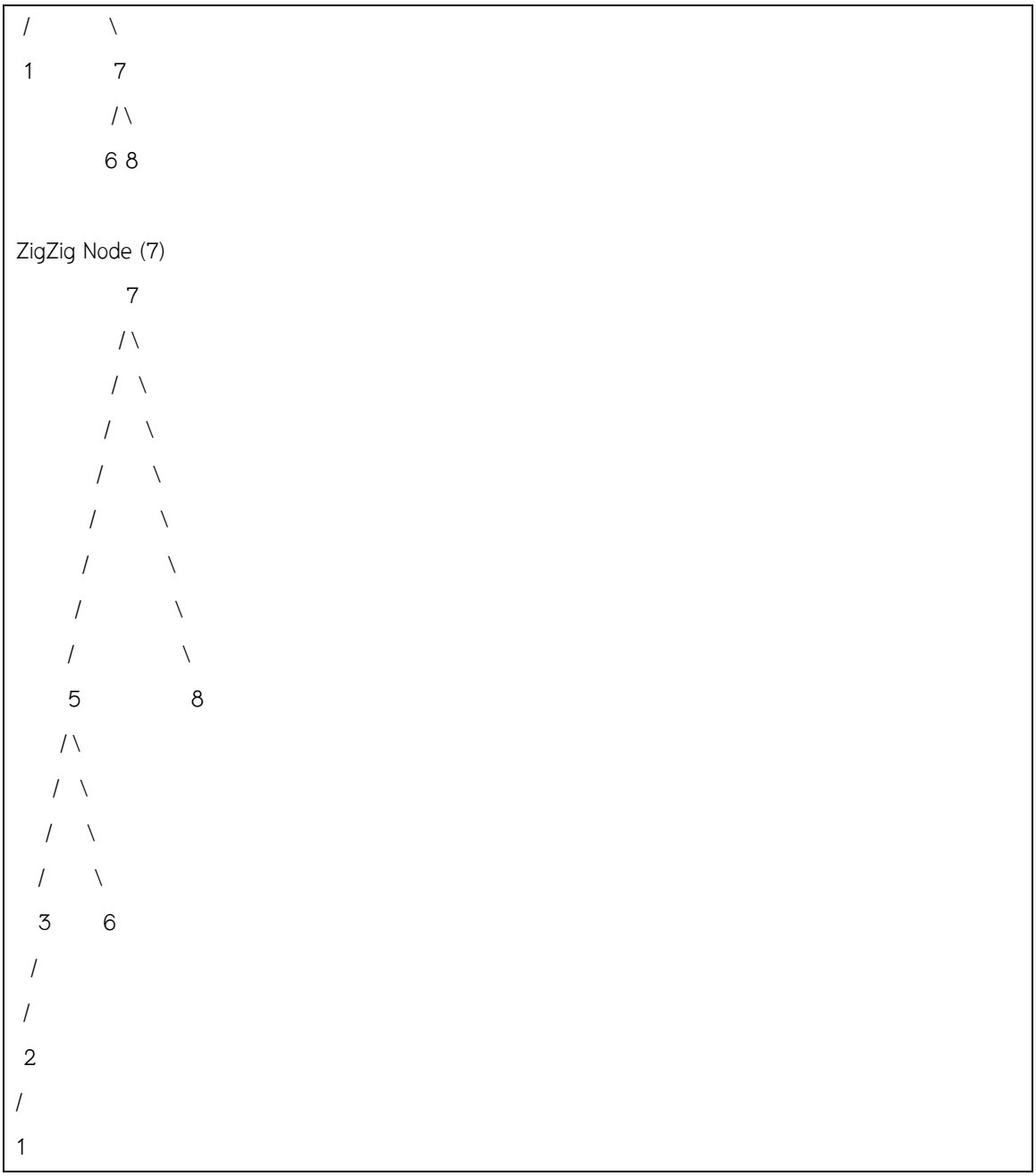




ZigZig Node (1)


3 7
/ \
2 5
ZigZag Node (5)
8
/
/
/
/
/
/
/
/
1
\
\
\
\
5
/ \
/ \
3 6
/ \
2 7
ZigZag Node (5)
5
/ \
/ \
/ \
/ \
1 8
\ /


\ /
3 6
/ \
2 7
ZigZag Node (7)
5
/ \
/ \
/ \
/ \
1 7
\ / \
\ / \
3 6 8
/
2
ZigZag Node (2)
5
/ \
/ \
2 7
/ \ / \
1 3 6 8
ZigZag Node (3)
3
/ \
/ \
/ \
/ \
2 5
/ \



Java code

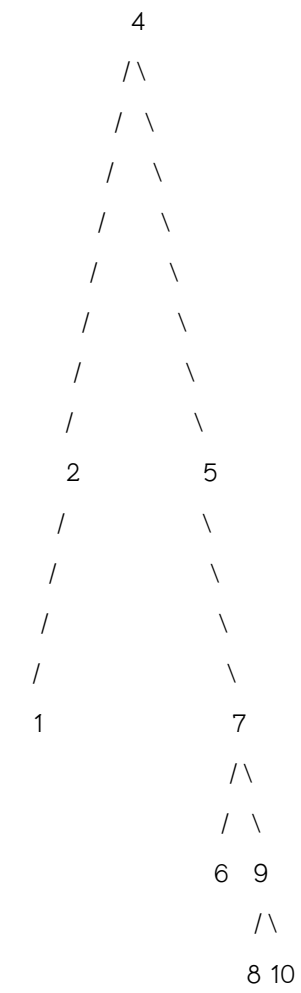
```
public static void main(String[] args) {
        SplayTree tree1 = new SplayTree();
        for (int i=10; i>=1; i--)
            tree1.insert(i);
        System.out.println("Initial tree height = " + Node.height(tree1.root));
```

```
        tree1.find(1);
        tree1.find(3);
        tree1.find(9);
        tree1.find(5);
        tree1.find(7);
        tree1.find(2);
        tree1.printTree();
        System.out.println("Tree height after multiple accesses = " + Node.height(tree1.root));
        tree1.delete(3);
        tree1.printTree();
        System.out.println("Tree height after one deletion = " + Node.height(tree1.root));
}
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```
Initial tree height = 9
              2
             / \
            /   \
           /     \
          /       \
         /         \
        /           \
       /             \
      /               \
     /                 \
     1                  5
                       / \
                      /   \
                     /     \
                    /       \
                   3         7
                    \       / \
                     \     /   \
                      4   6    9
                              / \
                             8  10
```

Tree height after multiple accesses = 4



Tree height after one deletion = 4

Java code

```
public static void main(String[] args) {
        BSTree tree1 = new BSTree();
        long start = System.currentTimeMillis();
        int N = 13000;
        for (int i = 0; i < N; i++) {
            tree1.insert(i);
        }
        System.out.println("Time for sequentially inserting " + N
                + " objects into BST = " + (System.currentTimeMillis() – start) + " msec");
        start = System.currentTimeMillis();
        for (int i = 0; i < N; i++) {
            tree1.find((int) (Math.random() * (N / 10)));
        }
        System.out.println("Time for finding " + (N/10)
                + " different objects in BST= " + (System.currentTimeMillis() – start) + " msec");

        SplayTree tree2 = new SplayTree();
        start = System.currentTimeMillis();
        for (int i = 0; i < N; i++) {
            tree2.insert(i);
        }
        System.out.println("Time for sequentially inserting " + N
                + " objects into SplayTree = " + (System.currentTimeMillis() – start) + " msec");
        start = System.currentTimeMillis();
        for (int i = 0; i < N; i++) {
            tree2.find((int) (Math.random() * (N / 10)));
        }
        System.out.println("Time for finding " + (N/10)
                + " different objects in SplayTree = " + (System.currentTimeMillis() – start) + " msec");
}
```

Output
Time for sequentially inserting 13000 objects into BST = 311 msec Time for finding 1300 different objects in BST= 28 msec Time for sequentially inserting 13000 objects into SplayTree = 3 msec Time for finding 1300 different objects in SplayTree = 9 msec
หมายเหตุ **เนื่องจาก มีการสุ่มตัวเลขขึ้น เพราะฉะนั้นผลลัพธ์ของ นศ จะแตกต่างจากผลลัพธ์ของอาจารย์ข้างบน แน่นอน อีกทั้ง Spec เครื่อง นศ กับเครื่องอาจารย์ต่างกัน การประมวลผลย่อมต่างกัน** **แต่หลักสำคัญคือ Insert และ Find ใน BST จะช้ากว่าของ SplayTree เสมอ** **ถ้า หาก Netbeans ของ นศ ฟ้อง StackOverflow ให้ นศ ลดค่า N ลง จาก 13000 เป็นค่าที่น้อยลง**

8. โปรดใช้ Starter code ที่อาจารย์แนบให้