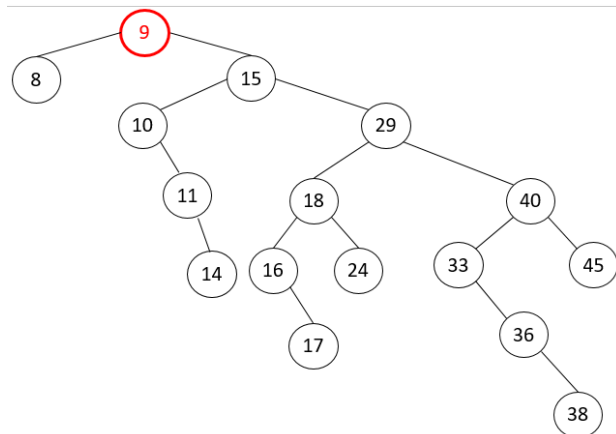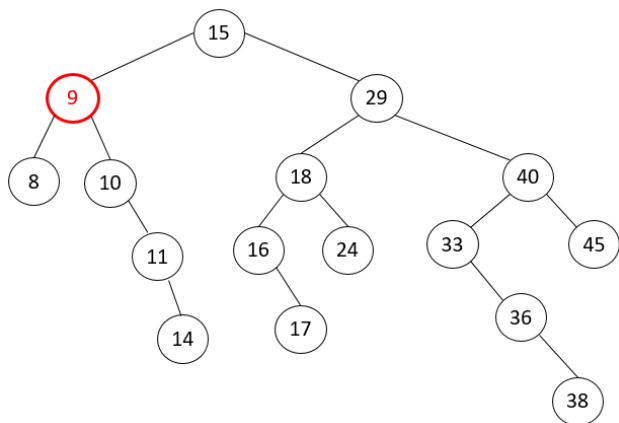1a) 4
 b) 0
 c) 3
 d) 40
 e) 24
 f) 8, 14, 11, 10, 17, 16, 24, 18, 15, 38, 36, 33, 45, 40, 29
 g) BST insert:
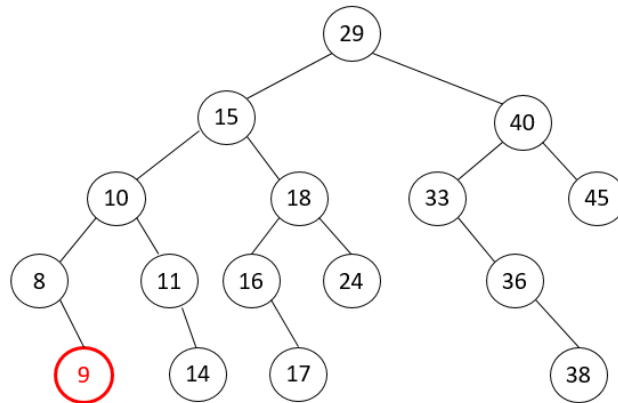
Zig-zag:
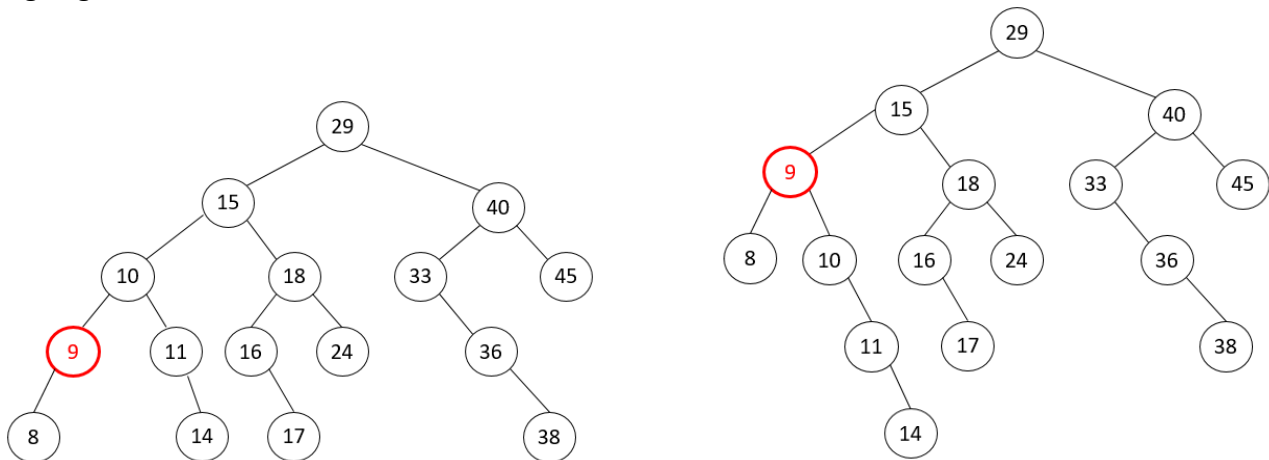
Zig-zig:

h) Remove 18:
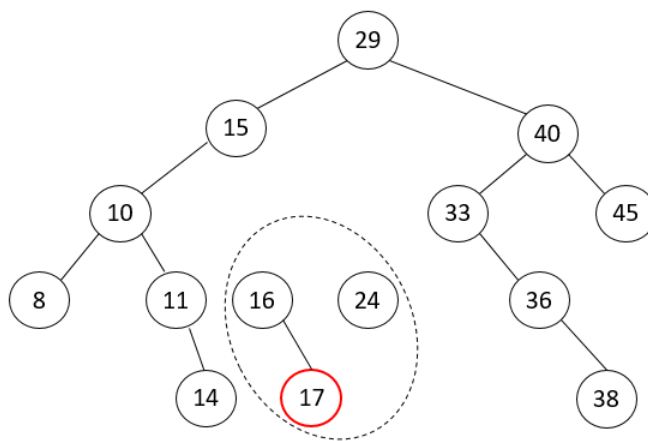
Join the subtrees of 18:
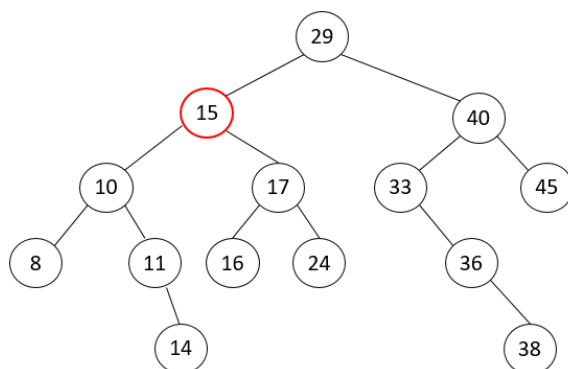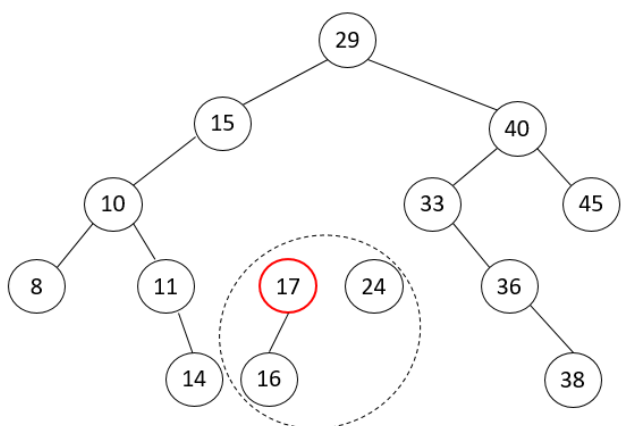
Splay at the (former) parent 15 (zig):
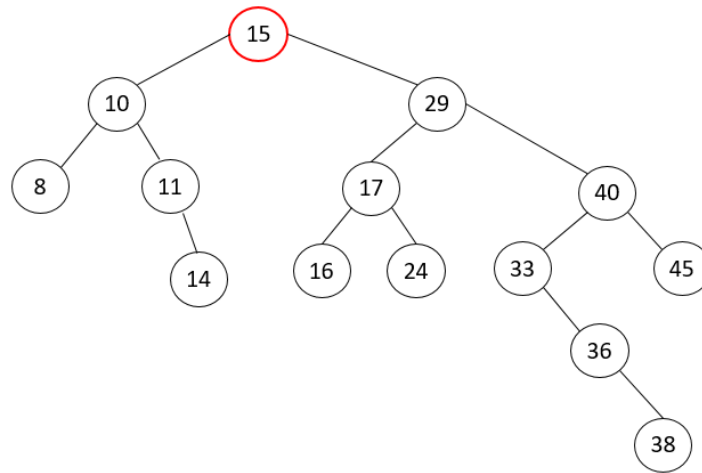
2a) 5
b) 2
c) Yes
d) DFS forest
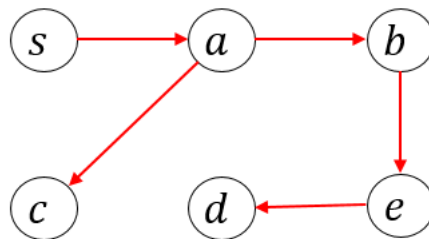
e) Back edges

f) Forward edges

g) Cross edges

h) Seven simple cycles:
a, d
f, h
s, a, c
a, c, d
a, b, e, d
b, e, d
f, g, h

3. There exists a unique sorting result.

c	f	d	g	h	e	a	b

4.
a) $O(V + E)$

b) $O(V + E)$
c) $O(V^2)$
d) $O(V + E)$
e) $O(V + E)$

5.

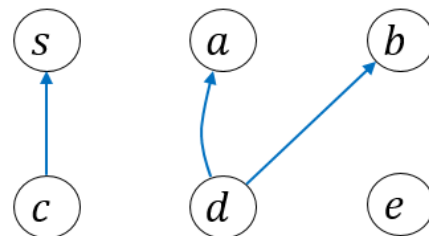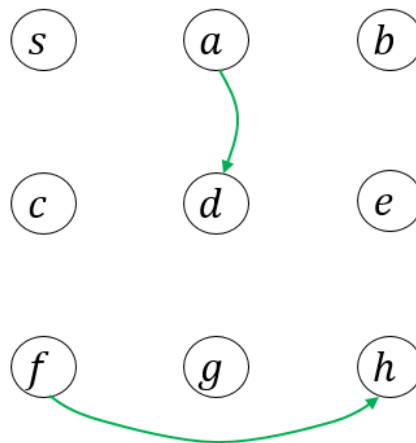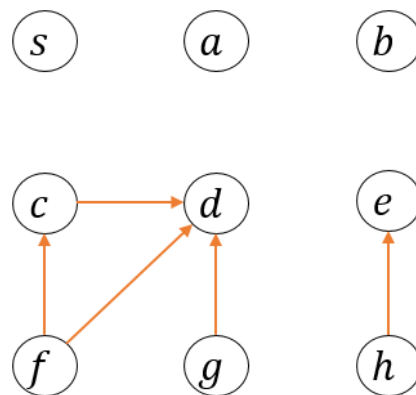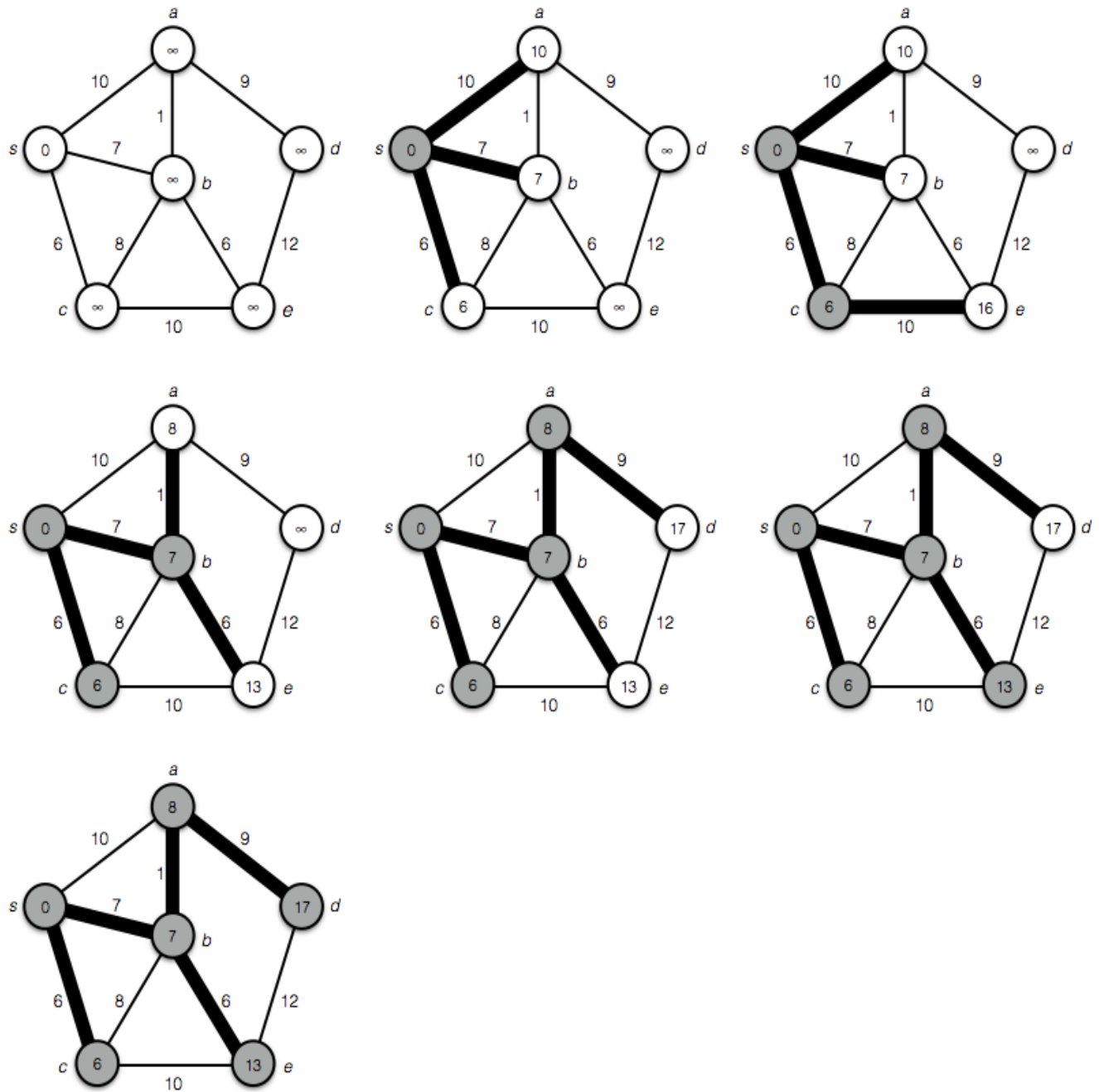| Row				Array			
0	3	4	6	5	2	0	1
1	3	5	6	4	2	0	2
2	**6**	**5**	**3**	**4**	**2**	**0**	**1**
3	1	5	3	4	2	0	6
4	5	1	3	4	2	0	6
5	5	4	3	1	2	0	6
6	0	4	3	1	2	5	6
7	4	0	3	1	2	5	6
8	4	2	3	1	0	5	6
9	0	2	3	1	4	5	6
10	3	2	0	1	4	5	6
11	1	2	0	3	4	5	6
12	2	1	0	3	4	5	6
13	0	1	2	3	4	5	6
14	1	0	2	3	4	5	6
15	0	1	2	3	4	5	6
16	0	1	2	3	4	5	6

7a)

```
/**
 * Perform a left rotation on the edge between the parent node p
 * and its right child node r.
 *
 * @param p        parent node
 * @param r        child node
 * @throws NullPointerException  if either p or r is null
 * @throws IllegalArgumentException  if neither p nor r is null
 *                                 but r is not a right child of p
 */
```

```java
public void leftRotate(Node<E> p, Node<E> r)
        throws NullPointerException, IllegalArgumentException
{
    // handle exceptions.
    //
    // insert code below (4 pts)
    if (p == null || r == null)
        throw new NullPointerException();
    if (p.right != r)
        throw new IllegalArgumentException();

    // make the left subtree of r the new right subtree of p.
    //
    // insert code below (3 pts)
    p.right = r.left;
    if (r.left != null)
        r.left.parent = p;

    // establish the relationship between r and the parent of p
    // (if p has one).
    //
    // insert code below (6 pts)
    Node<E> g = p.parent;    // grandparent of r
    r.parent = g;
    if (g == null)           // p is the root.
        root = r;
    else
    {
        if (g.left == p)
            g.left = r;
        else
            g.right = r;
    }

    // reverse the parent-child relationship between p and r.
    //
    // insert code below (2 pts)
    r.left = p;
    p.parent = r;
}
```

b)

```java
/**
 * Change the binary tree into its mirror image.
 */
public void mirrorImage()
```

```
{
    // insert code below (1 pt)

    mirrorImageRec(root);
}

/**
 * Replace the subtree rooted at n with its mirror image. Link
 * updates only. No creation of a new node.
 *
 * @param n   root of the subtree to be replaced with its mirror
 *            image.
 */
private void mirrorImageRec(Node<E> n)
{
    // handle the case n == null
    //
    // insert code below (1 pt)
    if (n == null) return;

    // swap the left and right subtrees and then generate their
    // mirror images.
    //
    // insert code below (5 pts)
    Node<E> tmp = n.left;
    n.left = n.right;
    n.right = tmp;

    mirrorImageRec(n.left);
    mirrorImageRec(n.right);
}
```