Lösningsförslag

Träd

U 1. Båda är korrekta. Den första har dock onödiga basfall.

U 2. a)
```
private void printInorder(Node<E> n) {
    if (n != null) {
        printInorder(n.left);
        System.out.println(n.element);
        printInorder(n.right);
    }
}
```

b) Utskriftssatsen placeras före respektive efter de två rekursiva anropen.

U 3. a) Följande metoder läggs till i trädklassen:

```
public int nbrLeaves() {
    return nbrLeaves(root);
}

private int nbrLeaves(Node<E> n) {
    if (n == null) {
        return 0;
    } else if (n.left == null && n.right == null) {
        return 1;
    } else {
        return nbrLeaves(n.left) + nbrLeaves(n.right);
    }
}
```

b) Den publika metoden läggs till i trädklassen och den rekursiva hjälpmetoden i nodklassen:

```
public int nbrLeaves() {
    if (root == null) {
        return 0;
    } else {
        return root.nbrLeaves();
    }
}

private int nbrLeaves() {
    if (left == null && right == null) {
        return 1;
    } else {
        int nbr = 0;
        if (left != null) {
            nbr += left.nbrLeaves();
        }
        if (right != null) {
            nbr += right.nbrLeaves();
        }
        return nbr;
    }
}
```

U 4.
```
/**
 * Returnerar en teckensträng som representerar uttrycket. Teckensträngen
 * innehåller parenteser runt alla deluttryck, utom runt talen.
 */
public String fullParen() {
    if (root == null) {
        return "";
    } else {
        StringBuilder sb = new StringBuilder();
        recFullParen(root, sb);
        return sb.toString();
    }
}

/*
 * Lägger i sb till de tecken som representerar uttrycket i det träd där n är
 * rot. Tecknen innehåller parenteser runt alla deluttryck, utom runt talen.
 */
private void recFullParen(ExprNode n, StringBuilder sb) {
    if (n.left == null && n.right == null) {
        sb.append(n.element);
    } else {
        sb.append('(');  // Trädet är strikt binärt, bägge subträden icke-tomma
        recFullParen(n.left, sb);
        sb.append(n.element);
        recFullParen(n.right, sb);
        sb.append(')');
    }
}
```

U 5.
```
/** Returnerar en lista med alla vanliga filer som är större än size bytes. */
public static List<File> biggerThan(File file, int size) {
        List<File> list = new ArrayList<File>();
        recBiggerThan(file, size, list);
        return list;
}

private static void recBiggerThan(File file, int size, List<File> list) {
    if (file.isFile()) {
        if (file.length() > size) {
            list.add(file);
        }
    } else if (file.isDirectory() {
        File[] files = file.listFiles();
        for (File f: files) {
            recBiggerThan(f, size, list);
        }
    }
}
```

U 6. Man börjar med att sätta in roten i kön. Därefter gör man så här:

```
så länge inte kön är tom
    tag ut den första noden i kön (och behandla den)
    om noden har vänster barn
        sätt in vänster barn i kön
    om noden har höger barn
        sätt in höger barn i kön
```