

Lösung 1

Zunächst könnte man meinen, dass `equal(Num n)` die Methode `equal(X x)` aus der Oberklasse ersetzt. Das ist aber nicht der Fall, weil eine andere Signatur vorliegt. Der Name `equal` ist also für `Num`-Objekte überladen und kann sich sowohl auf `equal(Num n)` als auch auf `equal(Integer x)` beziehen. Der Aufruf `equal(0)` passt – wegen des Autoboxing – zu `equal(Integer x)` und führt damit zu einer *nichtabbrechenden Rekursion* (daraus folgt: *stack overflow*).

Lösung 2

```
class TreeBag<X extends Comparable<X>> {
    private class Node { X value; Node left, right;
        void check() { // inorder traversal
            if(left!=null) left.check();
            if(!result) return;
            if(value.equals(last)){
                result = false; return; }
            else last = value;
            if(right!=null)right.check();
        }
    }
    private Node root;
    private X last = null;
    private boolean result;
    .....
    public boolean isSet() {
        result = true;
        if(root!=null) root.check();
        return result;
    }
}
```

Lösung 3

```
data ZDBaum = Empty | T1 ZDBaum Int ZDBaum
              | T2 ZDBaum Int ZDBaum Int ZDBaum

abst :: ZDBaum -> {Int}
abst (T2 lt a mt b rt) = (abst lt)++{a}++(abst mt)++{b}++(abst rt)
abst (T1 lt a rt)      = (abst lt)++{a}++(abst rt)
abst Empty = {}
```

```

inv :: ZDBaum -> Bool
inv Empty = True
inv (T1 lt a rt) = all (<a) (abst lt) && -- Suchbaumeingenschaft
                  all (>a) (abst rt) &&
                  h lt == h rt &&      -- B-Baum (Wege zu Bl.)
                  inv lt && inv rt     -- Rekursion

inv (T2 lt a mt b rt) = a < b &&      -- Suchbaumeingenschaft
                  all (<a) (abst lt) &&
                  all (\x->x>a&&x<b) (abst mt) &&
                  all (>b) (abst rt) &&
                  h lt==h mt && h mt==h rt && -- B-Baum
                  inv lt && inv mt && inv rt -- Rekursion

h (T2 lt _ mt _ rt) = max (h lt) (max (h mt) (h rt)) + 1
h (T1 lt _ rt) = max (h lt) (h rt) + 1
h Empty = 0

```

Lösung 4

```

static boolean existsGap(SortedSet<Pair<Float, Float>> inter) {
    if (inter.isEmpty()) return false;
    Float last = inter.first().second;
    for (Pair<Float, Float> i : inter)
        if (last.compareTo(i.first) < 0) return true;
        else last = Math.max(last, i.second);
    return false;
}

```

Lösung 5

```

class SuffixTree {
    static class Node {
        char c;
        List<Node> subtrees = new Vector<Node>();
        Node(char c){ this.c = c; }
    }
    private Node root = new Node('.'); // ignored in search
    .....
    public boolean contains(String s) {
        Node current = root;
        loop: for(int i=0; i<s.length(); i++) {
            char c = s.charAt(i);
            for(Node n: current.subtrees)
                if(n.c==c){current = n; continue loop;}
            return false; }
        return true;
    }
}

```