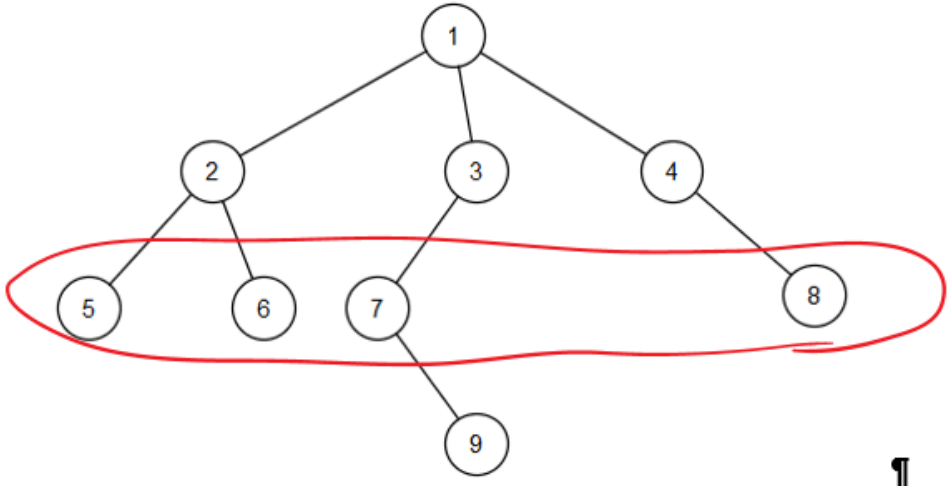
Aufgabe 1a – Baum Theorie



M. 3

¶

1.) 5, 6, 2, 9, 7, 3, 8, 4, 1

2.) siehe Bild

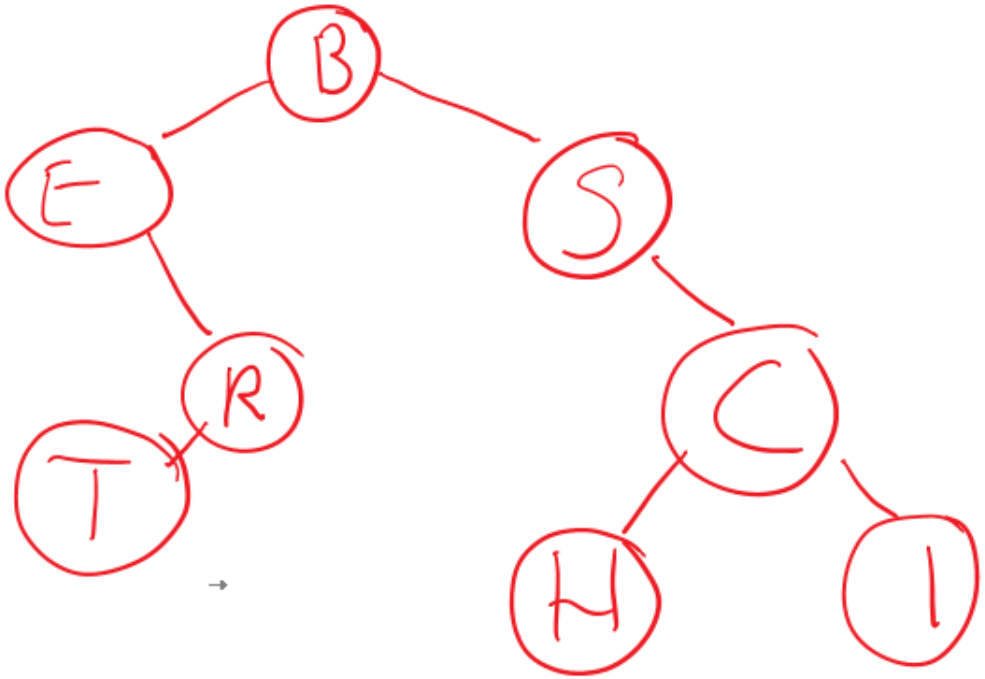
3.) mindestens 3, da die Wurzel 3 Kinder hat

4.) 1, 2, 3, 4, 7

5.) Nein, nicht jeder innere Knoten hat 3 Kinder

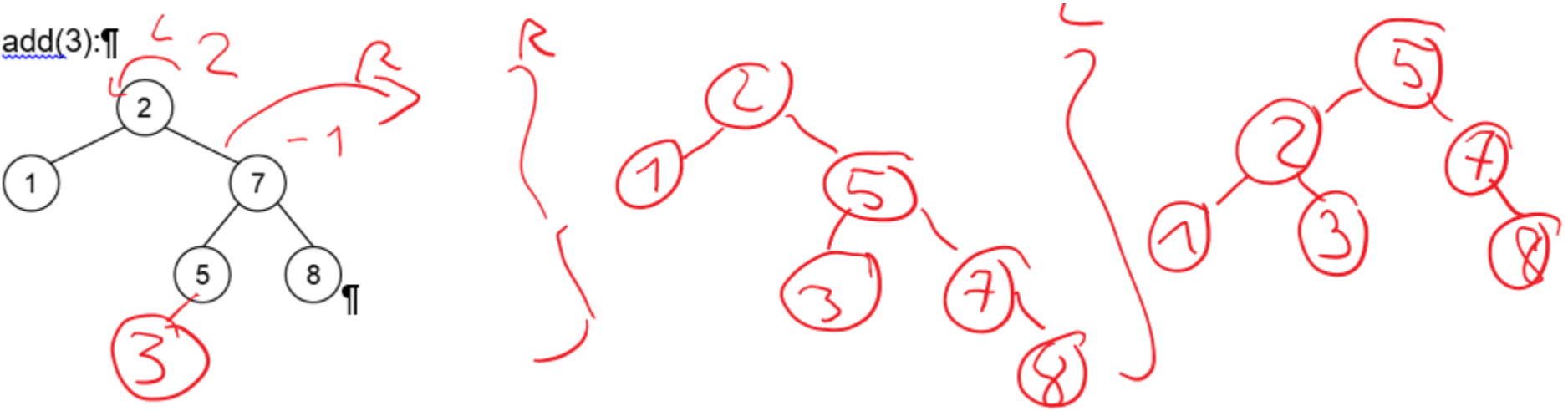
Aufgabe 1b – Baum Theorie

- Jeder Knoten speichert einen Buchstaben.
- Die Preorder-Reihenfolge der Knoten des Baumes ist: BERTSCHI
- Die Inorder-Reihenfolge lautet: ETRBSHCI

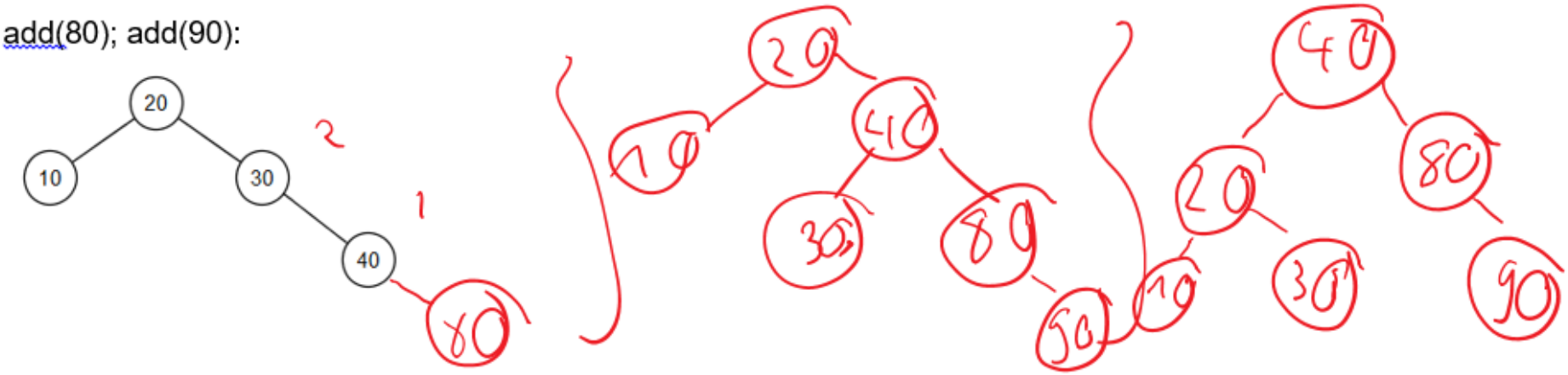


Aufgabe 2 – AVL Bäume

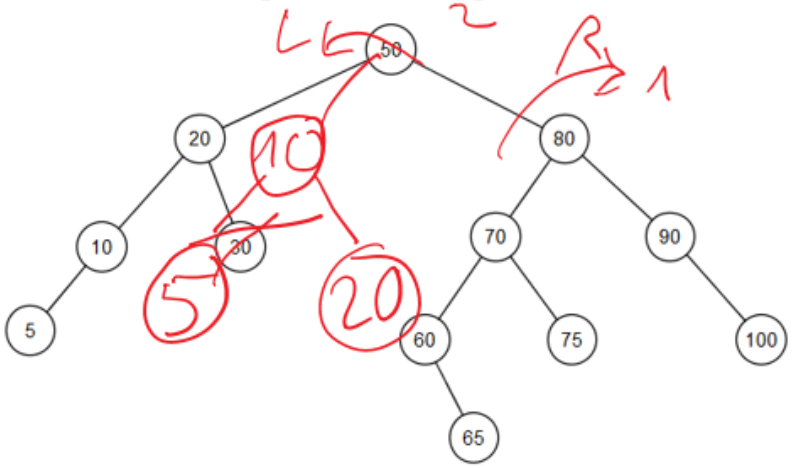
add(3):¶

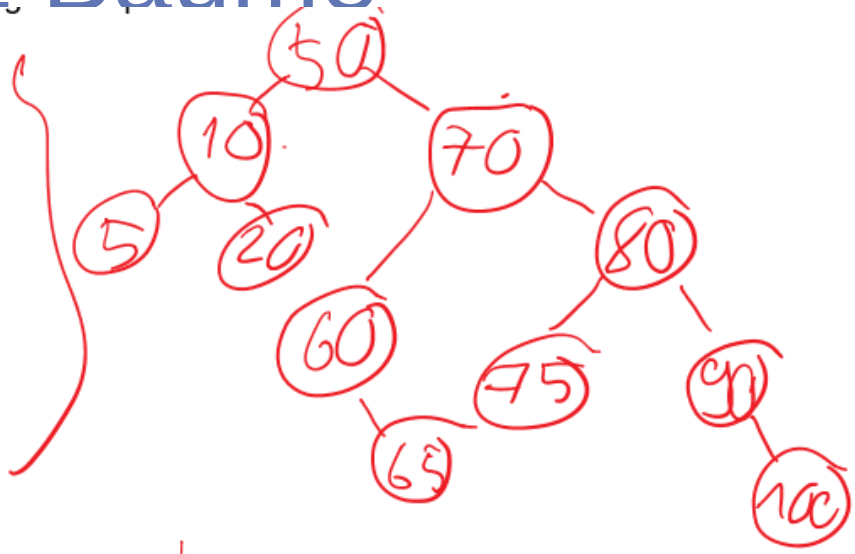


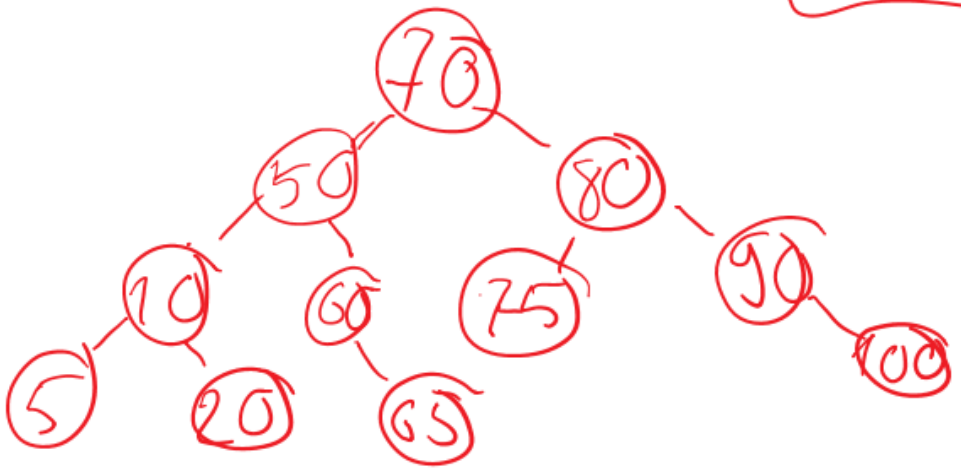
add(80); add(90):



Aufgabe 2 – AVL Bäume

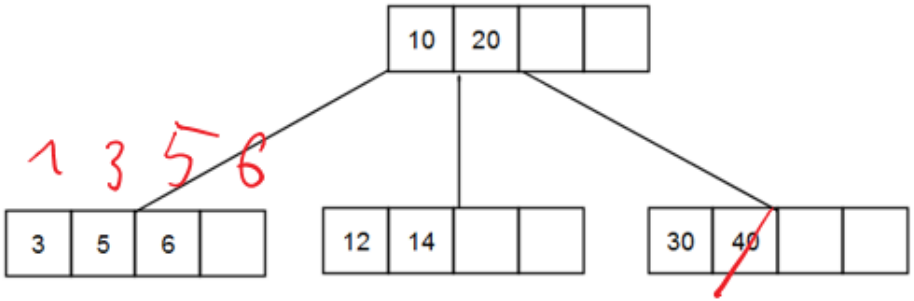


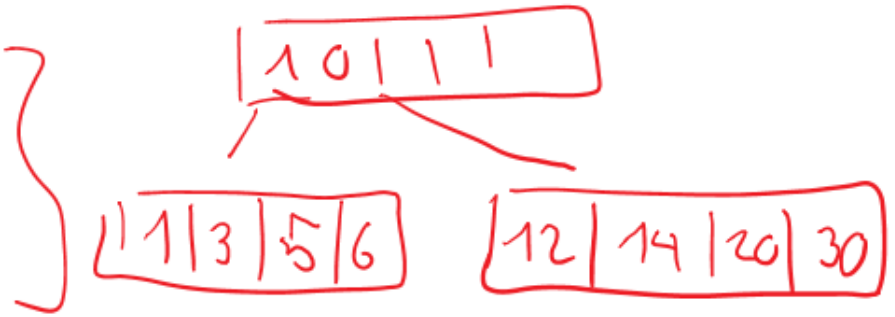




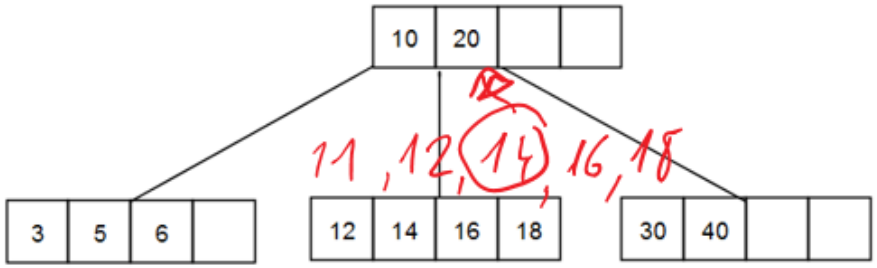
Aufgabe 3 – B-Bäume

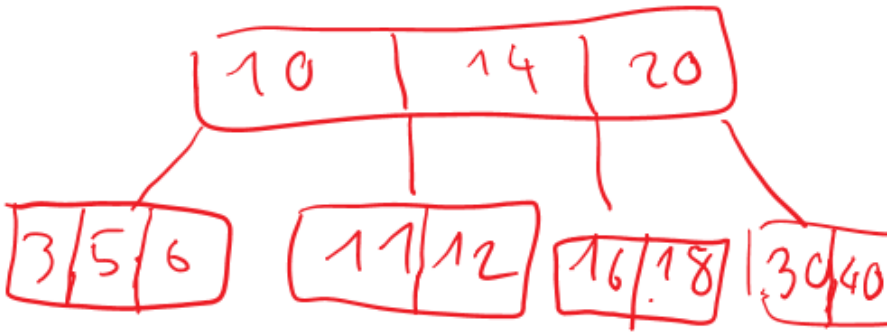
a) B-Baum der **Ordnung 2**: Add(1); Delete(40)





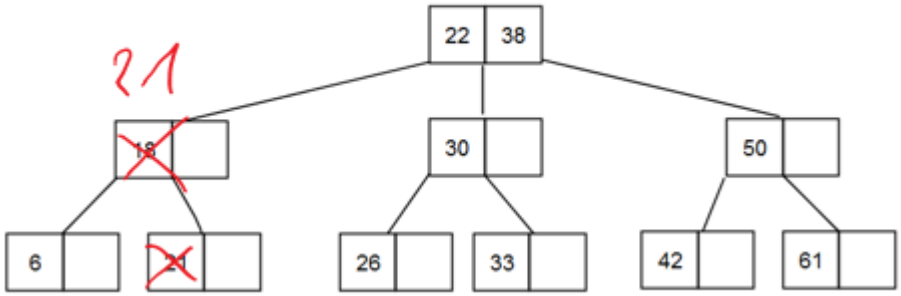
b) B-Baum der **Ordnung 2**: Add (11)

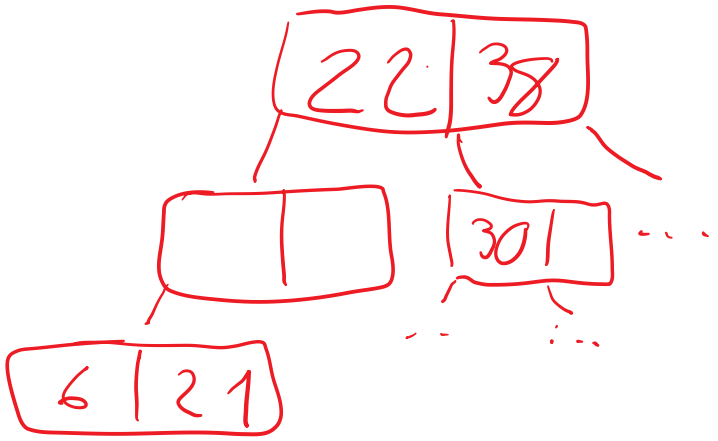


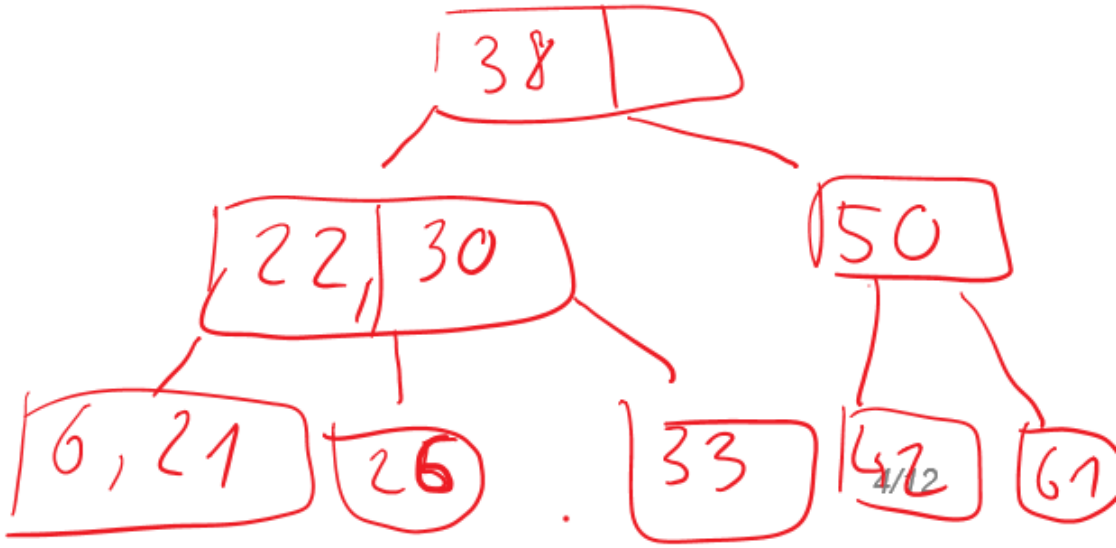


Aufgabe 3 – B-Bäume

c) B-Baum der **Ordnung 1**: Delete(18)

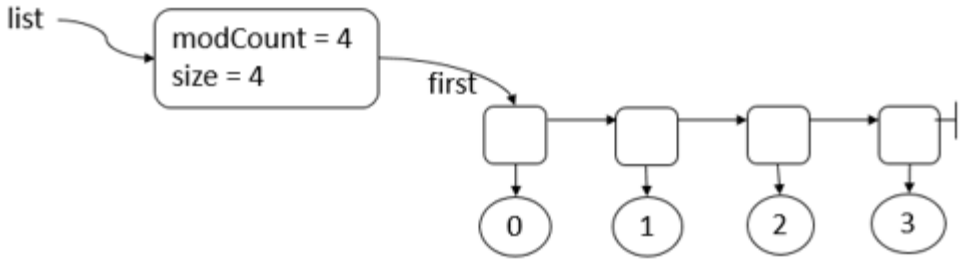






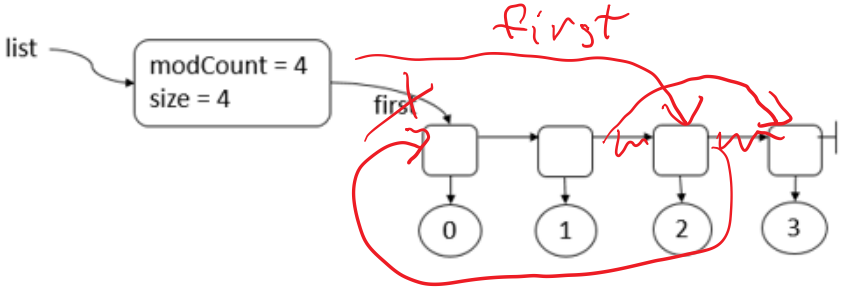
Aufgabe 4 – Special Method auf Liste

```
public boolean specialMethod(Object elem) {
    Node<E> n = first, p = null;
    while (n != null && !n.elem.equals(elem)) {
        p = n;
        n = n.next;
    }
    if (n != null) {
        if (p != null) {
            p.next = n.next;
            n.next = first;
            first = n;
        }
        return true;
    } else {
        return false;
    }
}
```



a.) Gibt der Aufruf specialMethod(0) true oder false zurück?

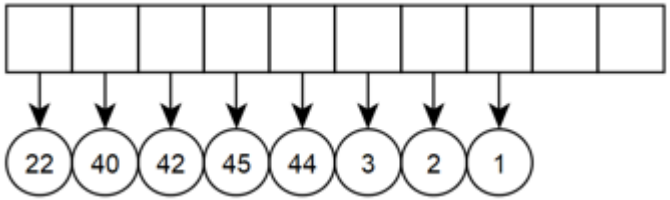
b.) Liste, nachdem specialMethod(2) aufgerufen wurde:



c.) Nein, die Methode verändert die Listen-Struktur

Aufgabe 5: Pyramid-Array

Beispiel:

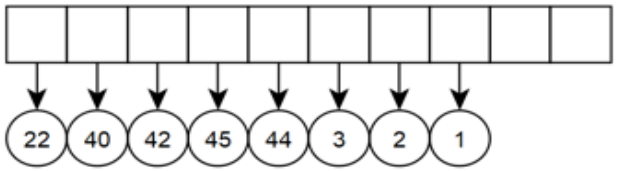



```
public class PyramidCollection_Stud<E extends Comparable<? super E>>
    extends AbstractCollection<E> implements Collection<E> {

    private static final int DEFAULT_CAPACITY = 10;

    private E[] data;
    private int size = 0;
    private int modCount = 0;
```

Aufgabe 5: Pyramid-Array

```
public class PyramidCollection_Stud<E extends Comparable<? super E>>
    extends AbstractCollection<E> implements Collection<E> {

    private static final int DEFAULT_CAPACITY = 10;

    private E[] data;
    private int size = 0;
    private int modCount = 0;
```

Beispiel:



Funktionierende Strategie alle Elemente zu durchlaufen

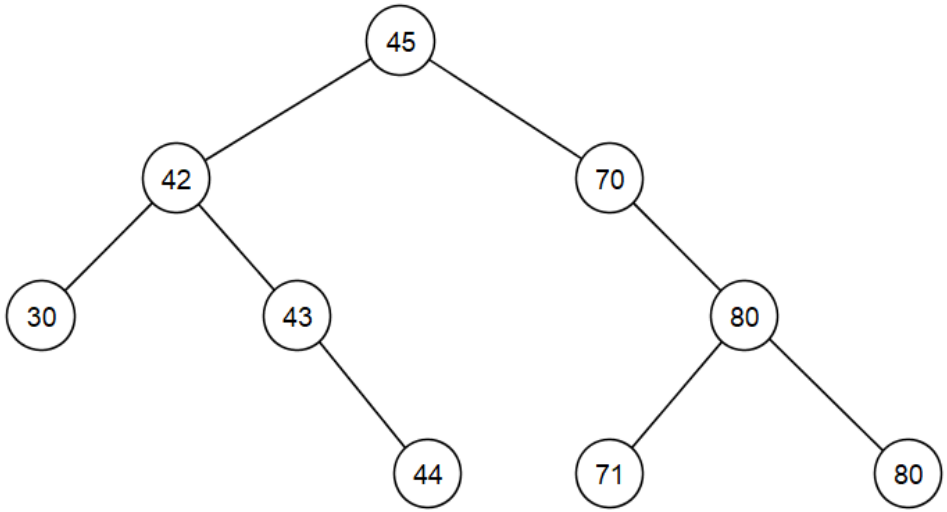
Alle Elemente werden zurückgegeben

Strategie ersichtlich, die alle Elemente sortiert zurück gibt

```
private class PyramidIterator implements Iterator<E> {
    int left = 0;
    int right = size - 1;
    int itModCount = modCount;

    @Override
    public boolean hasNext() {
        return left <= right;
    }

    @Override
    public E next() {
        if (itModCount != modCount) {
            // ConcurrentModifictionException auch ok
            throw new IllegalStateException();
        }

        if (!hasNext()) {
            throw new NoSuchElementException();
        }

        if (data[left].compareTo(data[right]) <= 0) {
            return data[left++];
        } else {
            return data[right--];
        }
    }
}
```

Aufgabe 6 – isAusgefuellt auf BST

```java
public boolean isAusgefuellt() {
    return isAusgefuellt(root);
}

private boolean isAusgefuellt(Node node) {
    if (node == null) {
        return true;
    }
    // return false if only one child
    if ((node.left == null && node.right != null)
            || (node.left != null && node.right == null)) {
        return false;
    }

    return isAusgefuellt(node.left) && isAusgefuellt(node.right);
}
```

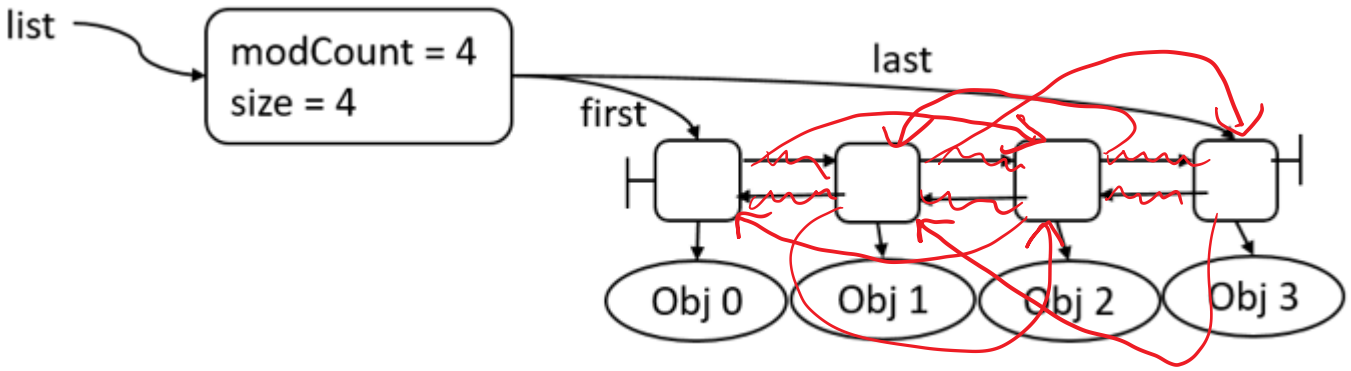


Aufgabe 7 – swapWithPrevious

```
public class DoublyLinkedList_Solution<E>
    extends AbstractList<E> implements List<E> {

    private int size = 0;
    private int modCount = 0;

    private Node<E> first;
    private Node<E> last;
```

```
public class Node<E> {
    public final E elem;
    public Node<E> next;
    public Node<E> prev;
```

- a.) Korrigieren Sie die Node-Referenzen in der unteren Abbildung, nachdem swapWithPrevious(2) aufgerufen wurde: Abzug pro fehlender / fehlerhafter Referenz



Aufgabe 7 – swapWithPrevious

```
public void swapWithPrevious(int index) {
    if (index < 1 || index >= size) {
        throw new IndexOutOfBoundsException();
    }

    Node<E> toggle = first;
    for (int i = 0; i < index; i++) {
        toggle = toggle.next;
    }

    Node<E> before = toggle.prev;

    toggle.prev = before.prev;
    if (before.prev != null) {
        before.prev.next = toggle;
    }

    before.next = toggle.next;
    if (toggle.next != null) {
        toggle.next.prev = before;
    }

    toggle.next = before;
    before.prev = toggle;

    // update first and last
    if (index == 1) {
        first = toggle;
    }

    if (index == size - 1) {
        last = before;
    }

    // Update mod count
    ++modCount;
}
```