Opgave 5: Trees

Opdracht 1

```
public static void createTree()
{
    BSTNode<Integer,Integer> root = new BSTNode( k: 40, val: 40);

    BSTNode node1 = new BSTNode( k: 25, val: 25);
    root.setLeft(node1);

    BSTNode node2 = new BSTNode( k: 78, val: 78);
    root.setRight(node2);

    BSTNode node3 = new BSTNode( k: 10, val: 10);
    node1.setLeft(node3);

    BSTNode node4 = new BSTNode( k: 32, val: 32);
    node1.setRight(node4);

    System.out.print("\nPre-order: ");
    root.preorder(root);

    System.out.print("\nIn-order: ");
    root.inorder(root);

    System.out.print("\nPost-order: ");
    root.postorder(root);
}
```

Voor het creëren van de boom begin ik met BSTNode met daarin het type van de key en value. Hier zijn deze allebei Integers. Deze geven we de naam root, omdat dit het begin is van de boom en geven de twee waarden mee. Daarna maak ik een nieuwe node aan genaamd node1. Deze zet ik daarna links t.o.v. van root. Dit doen we daarna ook een keer rechts van de root. Daaronder zien we ook dat we andere nodes aan node 1 kunnen vasthangen.

Daarna printen we de tree af op verschillende manieren.

```java
public void preorder(BinNode trav) {
  if(trav == null) return;
  System.out.print(trav.element()+" ");
  preorder(trav.left());
  preorder(trav.right());

}

public void inorder(BinNode trav) {
  if(trav == null) return;
  inorder(trav.left());
  System.out.print(trav.element()+" ");
  inorder(trav.right());

}

public void postorder(BinNode trav) {
  if(trav == null) return;
  postorder(trav.left());
  postorder(trav.right());
  System.out.print(trav.element()+" ");

}
```

Het resultaat dat we hiervan krijgen is

```
Pre-order: 40 25 10 32 78
In-order: 10 25 32 40 78
Post-order: 10 32 25 78 40
```

Opgave 2

```
public void insert(Key key, E value)
{
  if(key instanceof Integer)
  {
    if((int)key < (int)this.key)
    {
      if(this.left == null)
        this.setLeft(new BSTNode<>(key,value));
      else
      {
        this.left.insert(key,value);
      }
    }
    else
    {
      if(this.right == null)
        this.setRight(new BSTNode<>(key,value));
      else
        this.right.insert(key,value);
    }
  }
}
```

Eerst kijken we of de key van het type Integer is, wanneer dit het geval is gaan we kijken of de waarde van dat getal kleiner is dat de key van de node. Wanneer we niet op het einde zijn doen we nog eens een insert in de node daar onder. Als er zich geen node meer onder ons bevind (this.left == null) dan plaatsen we daar een nieuwe met onze meegegeven key.

Wanneer het getal groter is doen we hetzelfde maar langst de rechterkant.

```
public int hoogte()
{
  int  hoogteLinks= 0;
  int  hoogteRechts = 0;

  if(this.left !=null)
    hoogteLinks =this.left.hoogte();

  if(this.right != null)
    hoogteRechts = this.right.hoogte();

  if(hoogteRechts > hoogteLinks)
    return hoogteRechts + 1;
  else
    return hoogteLinks + 1;
}
```

Bij de method hoogte zetten we twee integers op nul. We gaan eerst kijken of we naar links kunnen, we blijven dezelfde method oproepen totdat we onderaan zitten. Dit doen we hetzelfde voor de vertakkingen naar de rechterkant. We testen dan welke teller het grootste is en geven deze mee met return.

```java
public static void testTree1()
{
    System.out.println("");
    BSTNode<Integer,Integer> root = new BSTNode( k: 1, val: 1);
    for (int i = 2; i < 9; i++) {
        root.insert(i,i);
    }
    System.out.println("");
    root.inorder(root);
    System.out.println("\n"+root.hoogte()+"\n");
}
public static void testTree2()
{
    Random random = new Random();
    int a = random.nextInt( bound: 20);
    BSTNode<Integer,Integer> root = new BSTNode(a,a);

    for (int i = 1; i < 10; i++) {
        int b= random.nextInt( bound: 20);
        root.insert(b,b);
    }

    root.inorder(root);
    System.out.println("");
    System.out.println(root.hoogte());

}
```

Voor het testen heb ik dit deeltje code geschreven. In het eerste deel gaat er een gesorteerde rij in en in de tweede willekeurige getallen. Het resultaat geeft:

```
1 2 3 4 5 6 7 8
8


2 5 13 13 17 18 19 19 19 19
6
```

Conclusie:

Wanneer we inorder gebruiken bij een BSTNode staan de waarden gesorteerd in optellende volgorde.