Opgaver tirsdag den 14. december

Opgave 1

1. Hvordan vil det binære søgetræ se ud, såfremt nedenstående tal indsættes i den angivne rækkefølge: 11, 21, 6, 13, 9, 22, 15, 10, 5, 17

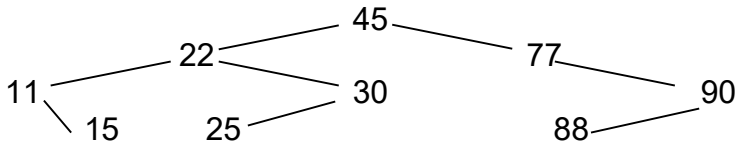
2. I hvilken rækkefølge skal nedenstående tal indsættes i et binært søgetræ, såfremt man skal opnå det mest balancerede søgetræ: 10, 5, 6, 13, 15, 8, 14, 7, 12, 4

3. I hvilken rækkefølge skal nedenstående tal indsættes i et binært søgetræ, såfremt man skal opnå det træ med den maksimale højde: 10, 5, 6, 13, 15, 8, 14, 7, 12, 4

Opgave 2

I filen BStree.jar kan der findes en implementation af et binært søgetræ.

Lav en `App` klasse med en `main()` metode der anvender
Implementationen af det binæresøgetræ til at opbygge nedenstående træ.



Dvs kald metoden add med værdierne i træet i en passende rækkefølge.

Tilføj nedenstående metoder til klassen `BinarySearchTree`.

- `findMax` – metoden skal returnere det største element i søgetræet
- `findMin` – metoden skal returnere det mindste element i søgetræet
- `removeMin` – metoden skal fjerne og returnere det mindste element i søgetræet. Metoden må ikke gå igennem træet flere gange, og må derfor ikke kalde den eksisterende `remove` metode.
- `removeMax` – som `removeMin`, men her skal fjernes det største element

Afprøv metoderne på ovenstående træ.

Opgave 3

I denne opgave skal der laves en ny implemetation af et Dictionary. Implementationen skal basere sig på anvendelse af et binært søgetræ, idet det kan antages at K er Comparable.

I jar filen BSTree kan du finde klassen `DictionaryBST`, I denne er der startet på implementationen. Se også koden nedenfor.
Læs og forstå den udleverede kode. Hjælpe metoden `find`, kan være til stor hjælp i de manglende metoder, så hver sikker på, at du forstår hvad den gør.

Programmer de manglende metoder: //TODO's. I `Node` klassen er der lagt op til at anvende en hjælpe metode `addNode`, der kan kaldes fra `put` metoden. Hent inspiration i den udleverede implementation af `BinarySearchTree`.

```
public class DictionaryBST<K extends Comparable<K>, V> implements
Dictionary<K, V> {

   private Node root;

   public DictionaryBST() {
      root = null;
   }

   @Override
   public V get(K key) {
      // TODO
      return null;
   }

   private Node find(K key) {
      Node current = root;
      boolean found = false;
      while (!found && current != null) {
         int d = current.key.compareTo(key);
         if (d == 0) {
            found = true;
         } else if (d > 0) {
            current = current.left;
         } else {
            current = current.right;
         }
      }
      if (found) {
         return current;
      } else {
         return null;
      }
   }

   @Override
   public boolean isEmpty() {
      // TODO
      return false;
   }

   @Override
   public V put(K key, V value) {
      // TODO
      return null;
   }
   @Override
```

```java
    public int size() {
        // TODO
        return -1;
    }

    private class Node {
        private K key;
        private V value;
        private Node left;
        private Node right;

        public Node(K key, V value) {
            this.key = key;
            this.value = value;
            this.left = null;
            this.right = null;
        }

        /**
         * Inserts a new node as a descendant of this node.
         *
         * @param newNode
         *            the node to insert
         */
         private void addNode(Node newNode) {
            // TODO
         }
    }
}
```