**Beschreibung der Rekursiven Funktionen:**

findMin(node):

* Startet bei dem gegeben Knoten und durchläuft den linken Teilbaum, um den kleinsten Schlüssel zu finden.

findMax(node):

* Startet bei dem gegeben Knoten und durchläuft den rechten Teilbaum, um den größten Schlüssel zu finden.

checkAVL(node):

* Überprüft, ob der Unterbaum des gegebenen Knoten die Eigenschaften eines AVL-Baums erfüllt 🡪 Balancefaktor <= 1

getHeight (node):

* Berechnet die Höhe des Unterbaums von dem gegebenen Knoten aus.

getBalance(node):

* Berechnet den Balancefaktor eines Knotens, indem die Höhe des rechten Teilbaums minus der Höhe des linken Teilbaums gerechnet wird

startSearch(node, key, searchedValues = []):

* Der Key wird dem Wert des aktuellen Nodes verglichen. Falls es gleich ist, wird die Suche direkt beendet und ausgegeben, dass der Wert enthalten wird.
* Wenn der Key kleiner als der aktuelle Node ist, wird, wenn ein Wert im linken Knoten gespeichert ist, der linke Node zum neuen aktuellen Node und der alte Node wird in die Liste gespeichert.
* Dieselbe Logik, wenn der Key größer als der aktuelle Node ist, nur dieses Mal, wird der rechte Knoten betrachtet und nicht der linke.
* Falls bei den oben genannten Fällen herauskommt, dass im nächsten Knoten kein Wert gespeichert ist, wird die Funktion beendet und in der Konsole wird ausgegeben, dass der gesuchte Wert nicht in der Liste vorhanden ist.
* Wenn der Key gleich dem aktuellen Knoten ist, wird der gesuchte Key sowie das Array mit allen durchsuchten Knoten ausgegeben.

**Aufwandsabschätzung:**

* findMin, findMax: Bei einem balancierten Baum, laufen beide Funktionen ungefähr die Hälfte des Baums durch und brauchen daher durchschnittlich O(log n). Die Laufzeit ist abhängig von der Höhe und ob der Baum balanciert ist, wenn der Baum nicht balanciert ist, kann es im schlimmsten Fall zu einer Laufzeit von O(n) kommen.
* checkAVL: Da die Funktion alle Knoten durchgehen muss, hat sie eine Laufzeit von O(n)
* insert: Hat eine durchschnittliche Laufzeit von O(log n), da nur ein Teil des Baums durchsucht werden, um den Knoten zu finden wo der Neue eingefügt werden soll (siehe Analyse startSearch). Das Einfügen selbst ist konstant. Das Aktualisieren der Höhen und gegebenenfalls rotieren, benötigt beides auch durchschnittlich O(log n).
* startSearch: Hat wie findMax und findMin eine Laufzeit von O(log n), da nur ein Teil des Baums durchlaufen werden, vorausgesetzt der Baum ist balanciert.