* **Analyse der rekursiven Funktionen:**

1.Insert: Einfügen eines Knotens in den Binärbaum.

1.1 Abbruchbedingung: Wenn der aktuelle Knoten null ist, wird ein neuer Knoten erstellt und zurückgegeben.

1.2 Parameter: Aktueller Knoten und Schlüsselwert zum Einfügen.

1.3 Rückgabewert: Der aktualisierte Knoten.

2. Height: Berechnung der Höhe eines Knotens im Baum.

2.1 Abbruchbedingung: Wenn der aktuelle Knoten null ist, wird die Höhe als zurückgegeben.

2.2 Parameter: Aktueller Knoten.

2.3 Rückgabewert: Die Höhe des Knotens.

3 TraverseAndCheckAvl: Durchlaufen des Baums in postorder und überprüfen, ob es sich um einen AVL-Baum handelt.

3.1 Abbruchbedingung: Wenn der aktuelle Knoten null ist, wird die Funktion beendet.

3.2 Parameter: Aktueller Knoten und boolscher Wert, der anzeigt, ob der Baum AVL ist.

3.3 Rückgabewert: Keiner.

4.TraverseAndCollectKeys: Durchlaufen des Baums in inorder und Sammeln der Schlüsselwerte.

4.1 Abbruchbedingung: Wenn der aktuelle Knoten null ist, wird eine leere Liste zurückgegeben.

4.2 Parameter: Aktueller Knoten.

4.3 Rückgabewert: Eine Liste von Schlüsselwerten.

5.PrintPathToKey: Druckt den Pfad zu einem bestimmten Schlüssel im Baum.

5.1 Abbruchbedingung: Wenn der aktuelle Knoten null ist, wird 'false' zurückgegeben.

5.2 Parameter: Aktueller Knoten, Schlüsselwert zum Suchen und optionale Liste von Pfadwerten.

5.3 Rückgabewert: Ein boolscher Wert, der angibt, ob der Schlüssel gefunden wurde oder nicht.

6. CheckSubtreeStructure: Überprüft, ob eine gegebene Struktur als Teilbaum im Hauptbaum vorhanden ist.

6.1 Abbruchbedingung: Wenn die Struktur leer ist, wird 'true' zurückgegeben. Wenn der aktuelle Knoten null ist, wird 'false' zurückgegeben.

6.2 Parameter: Aktueller Knoten und Liste von Schlüsselwerten, die die Struktur des Teilbaums darstellen.

6.3 Rückgabewert: Ein boolscher Wert, der angibt, ob die Struktur als Teilbaum vorhanden ist oder nicht.

* **Aufwandsabschätzung:**

1. Insert: O(h), wobei h die Höhe des Baums ist. Im schlimmsten Fall kann dies O(N) sein, wenn N die Anzahl der Integer-Werte in der Eingabedatei ist.
2. Height: O(h), wobei h die Höhe des Baums ist.
3. TraverseAndCheckAvl: O(N), da jeder Knoten im Baum einmal besucht wird, wobei N die Anzahl der Knoten im Baum ist.
4. TraverseAndCollectKeys: O(N), da jeder Knoten im Baum einmal besucht wird, wobei N die Anzahl der Knoten im Baum ist.
5. PrintPathToKey: O(h), wobei h die Höhe des Baums ist. Im schlimmsten Fall kann dies O(N) sein, wenn N die Anzahl der Integer-Werte in der Eingabedatei ist.
6. CheckSubtreeStructure: O(N \* M), wobei N die Anzahl der Knoten im Hauptbaum und M die Anzahl der Knoten im Teilbaum ist. Dies liegt daran, dass im schlimmsten Fall jeder Knoten im Hauptbaum besucht wird und die Struktur des Teilbaums an jedem Knoten überprüft wird.

* **Fazit zur Aufwandabschätzung**

Zusammenfassend sind die meisten Funktionen in der BinaryTree-Klasse in Bezug auf die Anzahl der Knoten im Baum, N, in der O-Notation effizient. Die Insert- und PrintPathToKey-Funktionen haben jedoch eine Komplexität von O(h), die im schlimmsten Fall O(N) sein kann. Die CheckSubtreeStructure-Funktion hat die höchste Komplexität von O(N \* M), wobei M die Anzahl der Knoten im Teilbaum ist.