Aufgabe 2

a)

Wir definieren k als den Abstand zum root und nennen die Funktion zum berechnen der Tiefe des Baumes als maxDepth. Wenn also root das Einzige Element im Baum ist dann gibt maxDepth die 0 zurück.
Wenn root Kinder hat dann rufen wir maxDepth für das jeweilige Kind mit dem Kind als Argument auf, also maximal noch zwei weitere Instanzen von maxDepth. Dann schauen wir welches der zwei Kinder das größere maxDepth haben und geben dieses als k+1 zurück.
Wir haben hier also eine Wost case Laufzeit von $\mathcal{O}(2^k)$, weil wir im schlimmsten Fall bei jedem Aufruf von maxDepth zwei weitere maxDepth instanzen erzeugen, und somit eine Exponentialfunktion haben.

b)

Wir können hier mit zum Beispiel dem Depth first search alle values von dem Baum in ein Array speichern. Dann sortieren wir mit einem stabilem binärem Algorithm das Array, also zum Beispiel mit Mergesort und geben dann die ersten k Elemente zurück. Die Worst case Laufzeit ist also $\mathcal{O}(n+n\log n) = \mathcal{O}(n\log n)$, falls die keys Integer sind kann man sogar mit zum Beispiel Radixsort eine Worst case Laufzeit von $\mathcal{O}(n+n) = \mathcal{O}(n)$ erreichen.

c)

Hier gehen wir alle Knoten einmal durch. Wir benutzen 2 Variablen: kleiner, größer. Wir schauen bei jedem Knoten ob der key dessen gleich dem key unserem gesuchtem Element ist, wenn ja dann geben wir das Element zurück, wenn nicht schauen wir ob dieser kleiner als der key von unserem gesuchtem Element ist und größer als die kleiner Variable, wenn ja dann speichern wir diesen neuen key wert in unsere kleiner Variable. Für die größer machen wir analog das gleiche nur mit den umgekehrten vergleichen. Falls wir keine Knoten mehr haben geben wir die beiden Variablen kleiner und größer zurück. Die worst case Laufzeit ist also $\mathcal{O}(n)$, weil wir alle Knoten maximal einmal besuchen und sonst auch keine weitere Funktion erzeugen.