11. Aufgabenblatt zur Vorlesung

# Informatik A

WiSe 2015/16

Wolfgang Mulzer

**Abgabe** am 19. Januar 2016 bis 12 Uhr in die Tutorenfächer

**Aufgabe 1** Traversieren von binären Suchbäumen *10 Punkte*

Eine *Traversierung* eines binären Suchbaums $T$ ist eine Funktion, welche alle Knoten von $T$ systematisch besucht und die gesehenen Einträge in einer Liste zurückgibt. Es gibt vier Arten der Traversierung:

- **Preorder**: Besuche erst die Wurzel, dann rekursiv das linke Kind, gefolgt von dem rechten Kind.
- **Inorder**: Besuche erst rekursiv das linke Kind, danach die Wurzel, dann das rechte Kind.
- **Postorder**: Besuche erst rekursiv das linke Kind, dann das rechte Kind und schließlich die Wurzel.
- **Levelorder**: Besuche die Knoten nach aufsteigender Tiefe und alle Knoten mit der gleichen Tiefe von links nach rechts.

Bearbeiten Sie die folgenden Aufgaben:

(a) Schreiben Sie Funktionen `preorder`, `inorder`, `postorder`, `levelorder:: Ord k => SBaum k v -> [(k,v)]`, welche die vier Traversierungsarten implementieren.

(b) Beweisen Sie durch Induktion: Wenn wir die Einträge eines binären Suchbaums in Inorder ausgeben, dann sind die Schlüssel in den Einträgen aufsteigend sortiert.

(c) Die Preorder-Traversierung eines binären Suchbaums mit 10 Einträgen ergibt die folgende Schlüsselfolge: M B A F D P N O Z R. Die Postorder-Traversierung liefert A D F B O N R Z P M. Wie sieht der Baum aus?

**Aufgabe 2** Algorithmische Probleme *10 Punkte*

Betrachten Sie das algorithmische Problem AlleVerschieden:

- **Eingabe**: Eine Liste `xs::[t]`, wobei `t` in der Typklasse `Ord` enthalten ist.
- **Ausgabe**: `True`, falls alle Elemente in `xs` paarweise verschieden sind, und `False`, sonst.

Im folgenden sehen Sie drei Algorithmen, die AlleVerschieden lösen sollen. Welche Algorithmen sind korrekt? Wenn ein Algorithmus nicht korrekt ist, geben Sie ein Gegenbeispiel. Wenn ein Algorithmus korrekt ist, bewerten Sie seine Effizienz, indem Sie die Anzahl von Vergleichen abschätzen, die er benötigt (in Abhängigkeit von der Listenlänge $n$).

```
-- Erster Algorithmus
av1::Ord t => [t] -> Bool
av1 []  = True
av1 [x] = True
av1 (x1:(x2:xs))
  | x1 /= x2   = av1 (x2:xs)
  | otherwise = False

-- Zweiter Algorithmus
av2::Ord t => [t] -> Bool
av2 []     = True
av2 (x:xs) = xunique && av2 xs
  where
    xunique = (filter (==x) xs) == []

-- Dritter Algorithmus
av3::Ord t => [t] -> Bool
av3 = av1 . msort

-- Merge Sort
msort::Ord t => [t] -> [t]
msort []  = []
msort [x] = [x]
msort xs  = merge ys zs
  where
    ys = msort (take half xs)
    zs = msort (drop half xs)
    half = (length xs) `div` 2
merge::Ord t => [t] -> [t] -> [t]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys)
  | x <= y    = x:merge xs (y:ys)
  | otherwise = y:merge (x:xs) ys
```

**Aufgabe 3** Sortieren *10 Punkte*

(a) Geben Sie eine Permutation der Ziffern 1 bis 8 an, bei deren Eingabe Merge Sort die maximale Anzahl von Vergleichen tätigt. Wieviele sind das?

(b) Analysieren Sie die Anzahl der Vergleiche beim Standardquicksort (Pivot ist der Kopf der Liste), wenn die Eingabefolge der Länge $n$ absteigend sortiert ist.

(c) Wie viele Vergleiche benötigt der Insertion Sort Algorithmus aus der Vorlesung, wenn die Eingabefolge der Länge $n$ aufsteigend sortiert ist?

Geben Sie immer alle Signaturen an und machen Sie geeignete Testläufe. Versehen Sie Ihr Skript mit geeigneten Kommentaren.