Klausur „Algorithmen und Datenstrukturen (AuD) I“
(Winter 2008/09)

Aufgabe 1 [Gesamtpunktzahl: 12]

Das Sortierverfahren ’Bubblesort’ arbeitet wie folgt:

- Beim Durchlaufen einer Liste werden jeweils die Paare unmittelbar aufeinanderfolgender Elemente miteinander verglichen und, falls sie nicht in der richtigen Reihenfolge bezüglich der verwendeten Ordnung sind, miteinander vertaucht.

- Das Durchlaufen wird solange wiederholt, bis die Liste vollständig geordnet ist.

a) Definieren Sie eine **Haskell-Funktion** `bubble` (inklusive Signatur), die *einmaliges* Durchlaufen einer Liste mit Vertauschen wie oben beschrieben realisiert.

Beispiel:
Der Aufruf
```
bubble [3,1,7,4,2]
```
soll liefern
```
[1,3,4,2,7]
```

b) Welches Element der (ursprünglichen) Liste befindet sich nach einmaliger Anwendung von `bubble` immer an der letzten Position der Liste?

Verallgemeinert: Welcher Teil der Liste ist nach der k-ten Anwendung von `bubble` auf jeden Fall bereits sortiert?

c) Definieren Sie eine **Haskell-Funktion** `bubbleK` (inklusive Signatur), die wie `bubble` arbeitet, aber abhängig vom Wert eines zusätzlichen Arguments k für die Anzahl bereits erfolgter Durchläufe den dann sicher bereits sortierten Teil der Liste **nicht mehr** erneut bearbeitet. Achten Sie auf eine möglichst effiziente Lösung.

d) Unter welcher Bedingung an die (ursprüngliche) Liste muss `bubbleK` maximal oft angewendet werden, um eine vollständig geordnete Liste zu erreichen) Wie viele Durchläufe sind dann insgesamt erforderlich?

e) Welcher Aufwand ($\mathcal{O}$-Notation) für das komplette Sortieren einer Liste durch wiederholte Anwendung von `bubbleK` – gemessen als Zahl der erforderlichen Vergleiche – ergibt sich damit im schlechtesten Fall für eine Liste der Länge n? Begründen Sie Ihr Ergebnis in nachvollziehbarer Weise.

Aufgabe 2 [Gesamtpunktzahl: 10]

Gegeben sei eine nichtleere Liste von Paaren vom Typ `(a,Int)`.

Beispiel: der Typ `a` könnte für Studenten (dagestellt z.B. durch `(Name, Vorname)`) Stehen und die Zahl für ihre erreichten Punkte bei der Klausur.

Ein kleines Beispiel einer solchen Liste könnte dann wie folgt aussehen:

```
[(("Mueller", "Heinrich"),38),(("Mueller", "Hans"),39),
 (("Mueller", "Hugo"),32),(("Meier","Paul",39)]
```

Uns interessiert nun die Liste all derjenigen Paare, bei denen der Int-Wert maximal ist.

a) Definieren Sie ein Haskell-Funktion `collectMax` (inklusive Signatur), welche die Liste all derjenigen Paare, bei denen der Int-Wert maximal ist, mit Hilfe vorhandener Funktionen höherer Ordnung (wie z.B. `map, filter, foldr,`…) bestimmt.
   Beachten Sie dabei, dass der maximale `Int`-Wert nicht vorab bekannt ist.

b) Definieren Sie dann direkt (d.h. ohne Verwendung von Funktionen höherer Ordnung) eine Haskell-Funkion `collectMax1`, die die gesuchte Liste *bei nur einem einzigen Durchlauf* durch die Liste von Paaren bestimmt.
   Beachten Sie auch hierbei, dass der maximale Int-Wert nicht vorab bekannt ist.

Aufgabe 3 [Gesamtpunktzahl: 8]

Gegeben seien die folgenden Definitionen der **Haskell-Funktionen** `map` und `foldr`.

```
map              :: (a -> b) -> [a] -> [b]

map  []           = []                              -- map.0
map f (x:xs)      = f x: (map f xs)                 -- map.1

foldr            :: (a -> b -> b) -> b -> [a] -> b

foldr _ z []      = z                               -- foldr.0
foldr g z (x:xs)  = g x (foldr g z xs)              -- foldr.1
```

Beweisen Sie durch strukturelle Induktion über die Listenlänge, dass **für alle endlichen Listen** `xs` und **für beliebige Funktionen** `h` (mit passender Signatur) gilt:

```
map h xs = foldr (\ e l -> h e : l) [] xs
```

[mit anderen Worten: jede Anwendung von `map` könnte durch die angegebene Anwendung von `foldr` ersetzt werden].

Wichtiger Hinweis: Geben Sie bitte bei allen Umformungen in Beweisschritten jeweils die verwendeteten Gleichungen oder sonstigen Begründungen an.

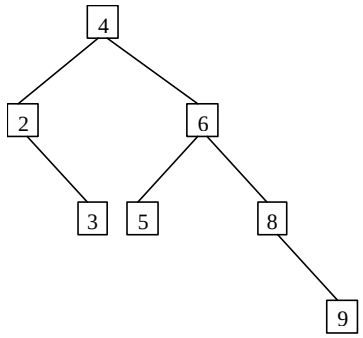

Abbildung 1: Ein binärer Suchbaum

Aufgabe 4 [Gesamtpunktzahl: 10]

a) Welche Liste von Zahlen ergibt sich bei einer *Postorder-Traversierung* des Baums aus Abb. 1?

b) Zur Erinnerung:
AVL-Bäume sind binäre Suchbäume mit der folgenden *Balanziertheitsbedingung*:

- *in jedem Knoten* gilt:
  - die Differenz zwischen der Höhe des linken Teilbaums und
  - der Höhe des rechtes Teilbaums ist nie grösser als 1

  Die Höhe eines Teilbaums ist dabei die maximale Länge eines Pfads von der jeweiligen Wurzel bis zu einem leeren Knoten.

Ist der Baum aus Abb. 1 ein AVL-Baum? Begründen Sie Ihre Antwort.

c) Pfade in einem Baum können als Listen mit den Werten der Knoten auf dem Weg von der Wurzel bis zu einem leeren Knoten dargestellt werden.

Definieren Sie ein **Haskell-Funktion** `paths` (inklusive Signatur), die aus einem Binärbaum alle **Pfade** von der Wurzel bis zu einem leeren Knoten in der Reihenfolge *von links nach rechts* ausliest und als Liste von Listen ausgibt.

Sie können hierbei verwenden, dass der Datentyp Binärbaum – wie in der Vorlesung – durch folgende Definition gegeben ist:

```
Data (Ord a) => BinTree a = EmptyBT
                          | NodeBT a (BinTree a) (BinTree a)
    deriving Show
```

Beispiel:
Für den Baum aus Abb. 1 ist das erste Element der von `paths` gelieferten Pfadliste die Liste `[4,2]` und das letzte Element die Liste `[4,6,8,9]`.
d) Welches sind die weiteren Elemente der für das voringe Beispiel (Baum aus Abb.1) von `paths` gelieferten Listenliste mit Pfaden?