Albert-Ludwigs-Universität Freiburg
Informatik 1
Wintersemester 2011/12

Prof. Dr. Wolfram Burgard
Dipl. Ing. Jörg Röwekämper
Dipl. Inf. Markus Kuderer

Zusatzübungsblatt

Abgabe in Absprache mit Tutor

Aufgabe 15.1

Betrachten Sie folgende Haskell-Implementierungen für die Berechnung des größten gemeinsamen Teilers zweier Eingabezahlen:

```
ggt1 a b  | b == 0 = a
          | a == 0 = b
          |otherwise = last [x| x<-[1..a*b], ((mod (a*b) x) == 0)]

ggt2 a b | b == 0 = a
         | a == 0 = b
         | a >= b = ggt2 (mod a b) b
         | otherwise = ggt2 a (mod b a)
```

Benutzen Sie die Haskell-Library `QuickCheck`, um zu testen, ob die Funktionen tatsächlich die geforderte Funktionalität erfüllen. Implementieren Sie dafür Test-Funktionen, die folgende Eigenschaften testen:

1. `ggt a b` teilt `a` und `b`
2. Es gibt keine Zahl `x > (ggt a b)`, die `a` und `b` teilt.

Aufgabe 15.2

1. Implementieren Sie eine Haskell-Funktion `insert` mit folgender Signatur:
   `insert:: (a->a->Bool) -> a -> [a] -> [a]`,
   die ein Element in eine sortierte Liste einfügt. Der Eingabeparameter `(a->a->Bool)` definiert dabei die Ordnungsrelation, bezüglich der die Liste sortiert sein soll. Beispiel:
   `insert:: (<=) 4 [1,3,7,11]` ergibt `[1,3,4,7,11]`
2. Implementieren Sie eine Haskell-Funktion `isort` mit folgender Signatur:
   `isort:: (a->a->Bool) -> [a] -> [a]`,
   die eine Liste bezüglich der angegebenen Ordnungsrelation sortiert. Verwenden Sie dafür `insert` und die Haskell-Funktion `foldr`.
3. Beschreiben Sie mit eigenen Worten, welche Operation folgende Funktion ausführt:

```
myfunc:: [Int] -> [Int]
myfunc l = isort (\ x y -> ((length [a| a<-[1..x], mod x a == 0])
         < (length [a| a<-[1..y], mod y a == 0]))) l
```

Aufgabe 15.3

Eine Prioritätswarteschlange wird dazu verwendet, Elemente zu speichern und diese später in einer bestimmten Reihenfolge wieder auszugeben. Dabei wird jedem Element ein Schlüssel `priority` mitgegeben der festlegt, in welcher Reihenfolge die Elemente zurückgeliefert werden. Dabei gilt, je kleiner der Wert von `priority` für ein Element ist, desto eher wird das Element ausgegeben. Die Methode zum Hinzufügen von Elementen sei `push()` und zum Herausnehmen `pop()`. Betrachten Sie die folgenden Klassendefinitionen, bei denen zur Realisierung der Prioritätswarteschlange eine einfach verkettete Liste als interne Datenstruktur verwendet wird.

```
class PriorityQueue {
    public PriorityQueue() { ... }
    public Object pop() { ... }
    public void push(Object o, int priority) { ... }
    ...
    protected Node head;
}

class Node {
    ...
    private Object content;
    ...
}
```

1. Geben Sie die vollständige Klassendefinition eines Speicherelements `Node` mit geeigneten Zugriffsmethoden sowie Konstruktor an und implementieren Sie diese.

2. Implementieren Sie den Konstruktor sowie die Methoden `push()` und `pop()` der Klasse `PriorityQueue`. Die Methode `push()` fügt ein Element hinzu und die Methode `pop()` liefert das nächste abzuarbeitende Element zurück und löscht dieses.

3. Bestimmen Sie die `best-case` und `worst-case`- Laufzeit Ihrer `push()` und `pop()` Methoden in Abhängigkeit der Größe der Warteschlange.

4. Implementieren Sie eine Methode, die eine `PriorityQueue` verwendet, um Elemente eines Integer-Vektors zu sortieren. Bestimmen Sie die Laufzeit der Methode `sort` in Abhängigkeit der Größe des zu sortierenden Vektors.