```
module HeapSort
  where
```

Definitionen: Sei H ein Binärbaum, dann sei $\#H$ die Zahl seiner inneren Knoten. Ein *B-Heap* ist ein Binärbaum mit Elementen in den inneren Knoten, in dem gilt:

Balance An jedem inneren Knoten (Kn a h1 h2) gilt für die Sub-Heaps: $\#h1 - \#h2 \in \{0, 1\}$, d.h. ihre Knotenzahl unterscheidet sich maximal um 1 und der linke Teilbaum ist größer.

Ordnung Das enthaltene Element in einem Knoten ist stets kleiner (oder allenfalls gleich) als die Elemente in beiden Sub-Heaps des Knotens.

```
data BHeap a = Empty
             | Kn a (BHeap a) (BHeap a)
        deriving Show -- Anzeigefunktion automatisch erstellen lassen
```

Nun können wir mit Hilfe des Heaps eine Liste sortieren: alle Elemente der Liste werden in einen leeren BHeap eingefügt und (sortiert) wieder herausgenommen.

```
-- sortieren mit Heap: alles einfügen und sortiert wieder herausnehmen
heapsort :: Ord a => [a] -> [a]
heapsort [] = []
heapsort xs = fromHeap ( toHeap xs)

-- Herausnehmen
fromHeap :: Ord a => BHeap a -> [a]
fromHeap Empty = []
fromHeap heap  = a:(fromHeap h')
        where (a,h') = removeTop heap

-- Einfügen
toHeap :: Ord a => [a] -> BHeap a
toHeap [] = Empty
toHeap xs = let insertAll :: Ord a => BHeap a -> [a] -> BHeap a
                insertAll h   []  = h
                insertAll h (a:as)= insertAll (insertHeap h a) as
            in insertAll Empty xs
-- dies ist eine einfache Faltungsfunktion (von links falten, Start mit leerem BHeap:
      --  = foldl insertHeap Empty xs

-- Spitze des Heap (kleinstes Element) entfernen, Heap-Eigenschaft muss erhalten bleiben
removeTop :: Ord a => BHeap a -> (a,BHeap a)
removeTop Empty = error "removeTop: Heap empty"
removeTop (Kn a Empty Empty) = (a,Empty)
removeTop (Kn a   lh    rh ) = (a, mergeHeaps lh rh)

-- neues Element in einen Heap einfügen, Heap-Eigenschaft erhalten
-- Trick für Balance-Bedingung: immer in rechten Teilbaum einfügen, austauschen
insertHeap :: Ord a => BHeap a -> a -> BHeap a
insertHeap   Empty a          = Kn a Empty Empty
insertHeap (Kn x left right) a | a < x     = Kn a (insertHeap right x) left
                               | otherwise = Kn x (insertHeap right a) left
```

Hier war ein Denkfehler drin: wir benutzen `mergeHeaps` ja immer nur für den Fall, dass beide Heaps (fast!) gleichgroß sind (in `removeTop` und rekursiv). Daher wesentlich einfacher als die ursprüngliche Lösung

Ursprünglich als Lösung angegeben:

```
mergeHeaps :: Ord a => BHeap a -> BHeap a -> BHeap a
mergeHeaps Empty x = x
mergeHeaps x Empty = x
mergeHeaps (Kn l l1 l2) (Kn r r1 r2)
        -- es gilt: #l1 + #r1 >= #l2 + #r2 >= #l1 + #r1 - 2
        --      ein Element nach oben, anderes in "kleineren" einfügen, auswählen
         | countleft heap2 > countright heap1  = Kn mi heap2 heap1
         | otherwise                           = Kn mi heap1 heap2
      where (mi, ma) = (min l r, max l r)
            heap2      = insertHeap (mergeHeaps l2 r2) ma -- #l2 + #r2 + 1
            heap1      = (mergeHeaps l1 r1)               -- #l1 + #r1

countleft,countright :: Ord a => BHeap a -> Int
countleft Empty          = 0
countleft (Kn _ Empty _) = 1
countleft (Kn _ l _)     = 1 + countleft l

countright Empty          = 0
countright (Kn _ _ Empty) = 1
countright (Kn _ _ r)     = 1 + countright r
```

Einfachere Lösung für diesen Fall:

```
mergeHeaps :: Ord a => BHeap a -> BHeap a -> BHeap a
mergeHeaps Empty x = x
mergeHeaps x Empty = x
mergeHeaps (Kn l l1 l2) (Kn r r1 r2)
        -- es gilt: #l1 + #r1 >= #l2 + #r2 >= #l1 + #r1 - 2
        --     UND: #l1 + #l2 >= #r1 + #r2 >= #l1 + #l2 -1 !!!
        --      ein Element nach oben, anderes in "rechten Teil" einfügen, Seiten tauschen
           = Kn mi (insertHeap right ma) left
    where mi   = min l r
          ma   = max l r
          left = mergeHeaps l1 l2
          right= mergeHeaps r1 r2
```

Ein einfaches Quicksort ist langsamer und verbraucht mehr Speicher (testen!), lässt sich aber wesentlich kürzer (mit "List-Comprehensions") formulieren:

```
quicksort :: Ord a => [a] -> [a]
quicksort []     = []
quicksort (x:xs) = quicksort [l | l <- xs, l <= x]
                        ++ [x] ++
                   quicksort [h | h <- xs, h > x]
```