Programação Funcional/Paradigmas da Programação I
1º Ano – LEI/LCC/LESI

Exame da 2ª Chamada

30 de Janeiro de 2007 – Duração: 2 horas

Parte II

1. Apresente uma definição para a função *merge* que funde os elementos de duas listas, ordenadas de forma crescente, numa única lista também ordenada de forma crescente:

```
merge :: (Ord a) => [a] -> [a] -> [a]
merge [1,3,5] [2,3,6]
=> [1,2,3,3,5,6]
```

2. Defina uma função `minsep` que recebe uma lista l e calcula um tuplo com o menor elemento da lista, e duas listas que contêm divididos entre elas os *restantes elementos* de l, por qualquer ordem. Os comprimentos das duas listas devem diferir no máximo numa unidade. A função deverá efectuar uma única travessia da lista.

```
minsep :: (Ord a) => [a] -> (a,[a],[a])
minsep [90,30,60,40,50]
=> (30,[90,60,],[40,50])
minsep [40,50]
=> (40,[50],[])
```

3. Considere o tipo indutivo de árvores binárias:

```
data BTree a = Empty | Node (a, BTree a, BTree a)
```

e a seguinte função que usa a função da alínea anterior:

```
list2btree :: (Ord a) => [a] -> BTree a
list2btree [] = Empty
list2btree l  = let (y,e,d) = minsep l
                in Node(y, list2btree e, list2btree d)
```

Que propriedades (ou *invariantes*) se pode afirmar que possuem todas as árvores construídas por esta função?

4. Defina a função `btree2list` que produz uma lista *ordenada* de forma crescente a partir de uma árvore construída pela função da alínea anterior.

```
btree2list :: (Ord a) => BTree a -> [a]
```

5. Utilizando como funções auxiliares apenas funções definidas nas alíneas anteriores, defina uma função de *ordenação* de listas

```
hsort :: (Ord a) => [a] -> [a]
```