Segundo teste de **Algoritmos e Estruturas de Dados**

22 de Novembro de 2017 **Duração máxima: 1 hora 30 minutos**

Justifique todas as suas respostas.

Nome:

N. Mec.:

Nas perguntas sobre árvores binárias, cada nó da árvore usa a seguinte estrutura de dados:

```
typedef struct tree_node
{
  struct tree_node *left;   // pointer to the left branch (a sub-tree)
  struct tree_node *right;  // pointer to the right branch (a sub-tree)
  struct tree_node *parent; // pointer to the parent node (NULL for the root node)
  int data;                 // the data (only one node can have a given data value)
}
tree_node;
```

2.0 **1:** Escreva uma função recursiva que, dada a raíz de uma árvore binária **não ordenada**, e um valor `v`, conta o número de nós da árvore que armazenam valores menores ou iguais a `v`. Qual é a complexidade computacional da sua função?

3.0 **2:** Escreva uma função recursiva **eficiente** que, dada a raíz de uma árvore binária **ordenada**, e um valor `v`, conta o número de nós da árvore que armazenam valores menores ou iguais a `v`. Qual é a complexidade computacional da sua função?

2.0 **3:** Explique como está organizada a informação num *min-heap*. Ilustre a sua exposição inserindo os números, por esta ordem, num *min-heap* inicialmente vazio: **7**, **3**, **9**, **1**, **2**.

3.0 **4:** É possível implementar eficientemente uma fila (*queue*) usando uma lista simplesmente ligada. Como?

2.0 **5:** Dos vários algoritmos de ordenação que conhece, existem alguns que funcionam naturalmente de uma forma recursiva. Explique o funcionamento de um deles (recursivo!).

2.0 **6:** Compare dois algoritmos de ordenação à sua escolha no que diz respeito a i) complexidade computacional, ii) melhor caso, iii) pior caso.

3.0 **7:** Explique como pode procurar informação numa lista biligada. Explique também como pode tornar a procura mais eficiente quando a informação de que se está à procura não estiver uniformemente distribuida.

3.0 **8:** Explique como funciona uma *hash table*. Indique as vantagens e desvantagens das implementações de *hash tables* usando *open-addressing* e *chaining*.