



## ALGORITMOS E ESTRUTURAS DE DADOS AULA DE LABORATÓRIO #05 - ORDENAÇÃO

### Objectivos

Este laboratório versa o tema da ordenação de dados. Como exemplo de trabalho, voltamos ao problema de contar palavras distintas num ficheiro de texto, problema que foi usado no Laboratório #03.

### Plano da Aula

Neste trabalho utilizaremos dois módulos em C que no seu conjunto permitirão ilustrar o problema da ordenação. Os módulos, denominados `sortwords.c` e `word.c` contêm código incompleto em que a organização dos dados é feita numa tabela como discutido no 1º problema do Laboratório #03. Cada elemento da tabela aponta para uma estrutura (tipo `Sword`), que contém um inteiro e um ponteiro para uma cadeia de caracteres.

Pretende-se agora não só a contabilização do número de palavras diferentes mas também a sua ordenação, que poderá ser segundo um dos seguintes critérios:

- A. ordem alfabética das palavras
- B. número de ocorrências
- C. comprimento das palavras

O critério de ordenação é especificado pelo utilizador na linha de comando. Para ordenação por ordem alfabética deverá ser usado o comando

```
./sortwords ficheiro.txt -alfabetica {-a|-d},
```

onde os parâmetros `a` e `d` indicam, respectivamente, ordenação ascendente e ordenação descendente. Para ordenação por número de ocorrências deverá ser usado o comando

```
./sortwords ficheiro.txt -ocorrencias {-a|-d}
```

e para ordenação pelo comprimento de palavras deverá ser usado o comando

```
./sortwords ficheiro.txt -comprimento {-a|-d}.
```

Por omissão, a ordenação deverá ser alfabética e ascendente, usando o algoritmo implementado na função `sort()` dada.

Estude o código fornecido, para compreender como está organizado, qual o algoritmo subjacente e de que forma os dados são guardados. Verifique os cabeçalhos das funções de interface, e note que o último argumento da função `sort()` é um ponteiro para função:

```
void sort(Item a[], int l, int r, int (*compare) (Item, Item));
```

Use o comando `make` e o ficheiro `Makefile`, com as alterações que entender necessárias, para compilar o programa e fazer testes. A pasta `test` tem alguns exemplos de textos, mas pode usar outros de maior dimensão, inclusivamente documentos ou livros disponíveis na internet ou criados para o efeito.

1. Identifique o algoritmo de ordenação implementado pela função `sort()` fornecida, e implemente a sua chamada no programa principal.

2. Escreva as restantes funções de comparação para implementar todos os critérios de ordenação. Liberte também a memória alocada.
3. Teste o programa com os diferentes critérios de ordenação e diferentes ficheiros de entrada.
4. Altere o programa de modo a avaliar as operações de acesso efectuadas. Obtenha dados experimentais usando ficheiros de teste, crie tabela com regressão mostrando que o algoritmo é  $\mathcal{O}(n^2)$ .
5. Crie uma versão `qsortwords.c`, em que o conteúdo da função `sort()` é substituído por uma implementação sua do algoritmo Quicksort.
6. Obtenha dados experimentais de teste que lhe permitam comparações com o programa anterior e tirar conclusões sobre a complexidade do Quicksort.
7. Considere o ponto anterior em casos de dados típicos e de pior caso.