Aula Prática 9

Resumo:

- Complexidade algorítmica;
- Algoritmos de ordenação;
- Recursividade.

Exercício 9.1

Na classe `p2utils.Sorting`, implemente a função `mergeSort` com o algoritmo de ordenação por fusão para vetores de números inteiros. Utilize o programa `TestMergeSort` para testar a função. Podemos utilizar este programa de inúmeras formas. Por exemplo:

```
$ # input as output of command echo (piping):
$ echo 4 3 2 6 1 2 3 5 7 8 9 5 | java -ea TestMergeSort

$ # input as output of command cat (piping):
$ cat numbers.txt | java -ea TestMergeSort

$ # redirecting input from file (assuming file numbers.txt exists):
$ java -ea TestMergeSort < numbers.txt

$ # no piping nor redirecting (Ctrl-d to terminate the input):
$ java -ea TestMergeSort
4 5 2 3 2
2 3
<Ctrl-d>
```

Exercício 9.2

Na classe `p2utils.Sorting`, implemente a função `insertionSort` com o algoritmo de ordenação por inserção. Utilize o programa `TestInsertionSort` para testar a função.

Exercício 9.3

O programa `TestSorting` mede os tempos necessários para ordenar arrays de N inteiros usando diversos algoritmos. Experimente-o e analise-o para perceber o seu funcionamento.

a. A classe `p2utils.Sorting` inclui dois campos `static` para contabilizar o número de operações elementares requeridas pelos algorimtos de ordenação:

   `comparisonCount` para contar comparações entre elementos do array;

`assignmentCount` para contar atribuições de valor a elementos.

Acrescente instruções para incrementar esses contadores nos locais apropriados dos vários algoritmos. Teste o programa com $N = 100$ e $N = 1000$ e observe as contagens de operações nos vários casos.

b. Altere o programa para aceitar vários valores de N na linha de comando e produzir uma tabela com todos os resultados das diversas medições para todas essas dimensões, todos os algoritmos de ordenação, e para os três casos de teste (arrays aleatórios, ordenados crescentemente e ordenados decrescentemente). Para cada dimensão, todos os algoritmos devem ser aplicados exatamente aos mesmos dados.

Exercício 9.4

Em Java, além de classes genéricas, também é possível criar métodos genéricos. No fim da classe `p2utils.Sorting` está declarado um método genérico para ordenar arrays com elementos de tipo `E`. Complete esse método com uma cópia do algoritmo de ordenação por fusão adaptado para esse tipo de elementos. (Tem de usar o método `compareTo` para comparar elementos.) Teste a função num pequeno programa `TestGenericSort` que ordene os seus argumentos (strings).

```
$ java -ea TestGenericSort maria joao ana tomas antonio
[ana, antonio, joao, maria, tomas]
```

Exercício 9.5

O programa `ListSort` lê um ou mais ficheiros de texto e ordena as suas linhas. Para isso, armazena as linhas numa lista ligada e depois ordena-as usando uma versão do algoritmo QuickSort para listas genéricas. Preencha as lacunas na função de ordenação para completar o programa.

Exercício 9.6

No programa `GetWords`, a função `extractWords` extrai para um *array* todas as palavras existentes num ficheiro de texto. Altere o programa para extrair palavras de vários ficheiros de texto passados como argumentos e produzir uma listagem por ordem lexicográfica, sem palavras repetidas e indicar o número total de palavras listadas.

Exercício 9.7

Altere o programa anterior por forma a que a informação por ele gerada passe a ser persistente. Assim, no fim da execução do programa a lista de palavras (sem repetição) por ele gerada deverá ser registada num ficheiro de texto (`words.txt`), e a (eventual) lista de palavras existente nesse ficheiro deverá ser lida pelo programa no início da sua execução.