Módulo 4 - Laboratório prático – Exercícios

Em cada módulo, os exercícios assinalados com ♣ são mais importantes.

Exercícios:

1. Faça a análise da ordem de complexidade do produto de multiplicação de matrizes implementado no Trabalho 2:

$$\sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=1}^{n} a_{ik}b_{kj}$$

2. Considere o seguinte algoritmo de ordenação:

```
Algoritmo bubbleSort (s,n):
        //Input: uma sequência ordenável s e a quantidade n de valores a ordenar
        //Output: a sequência ordenada (crescente) da posição 1 à posição n
    for i = 1 ... n do
        for i = 1 ... n-1 do              // verificar se os elementos estão ordenados (crescente)
            if s[i] > s[i + 1] then       // trocar elementos de lugar
                trocar(s[i], s[i+1])
            end if
        end for
    end for
```

   a) Implemente este algoritmo em Pyhton.

   b) Analise (calcule) a ordem de complexidade, explicando quais os passos elementares importantes neste algoritmo. Indique, justificando se existe um melhor e pior dos casos para este algoritmo.

   c) Verifique que modo se poderia melhorar o algoritmo, ou seja, efetuar menos passos elementares.

   d) Implemente os melhoramentos que considere mais importantes e faça um teste empírico de comparação (ver enunciado da alínea 4.a).

   e) Estude a ordem de complexidade no pior dos casos do seu melhoramento e conclua sobre a "qualidade" do melhoramento relativamente ao algoritmo original.

3. Crie uma classe – `Ordena` - de funções de ordenação de valores em sequências (lists) que implemente os seguintes algoritmos:
   a) Bubblesort
   b) Selectionsort
   c) Insertionsort
   d) Mergesort
   e) Quicksort

4. Usando uma função para devolver uma `list` com valores inteiros gerados aleatoriamente a variar em {1, ..., 10000} e onde n será um parâmetro inteiro dado para a quantidade de inteiros na lista:

a) Teste os seus algoritmos em *lists* com n a variar em $\{100, 300, 500, 800, 1100, 1400, 1700, 2000\}$, **cronometrando** e guardando os tempos médios (gastos apenas na ordenação) para cada um dos algoritmos da classe `Ordena` e, ainda, usando o método sort() para ordenação de lists.

b) Utilizando o módulo para criar um gráfico de visualização indicado no primeiro exercício, crie um gráfico de linhas de modo a comparar os diversos tempos de execução de todos os algoritmos de ordenação da alínea (4.a).