Módulo 3 - Laboratório prático – Exercícios

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

Exercícios:

1. ♣ Considere as seguintes funções: $y1 = 100x + x^2$ e $y2 = x^2$.

a) Implemente estas funções em código Python e, de. modo a gerar gráficos para ambas as funções, obtenha listas de valores de y1 e y2 para x a variar nos seguintes intervalos:

$x \in [0, 100]$; $x \in [0, 300]$; $x \in [0, 500]$; $x \in [0, 1000]$; $x \in [0, 10000]$; $x \in [0, 100000]$

b) Usando o módulo de visualização. Matplotlib (https://matplotlib.org/) através do comando `import matplotlib.pyplot as plt` e as funções `plt.plot(x,y1)`, `plt.plot(x,y2)` e `plt.show()`.

c) Após observação dos gráficos, que conclusões pode estabelecer?

Nota: *Quando necessário calcular o tempo de execução, como já sabe, pode usar a biblioteca de módulos `timeit` do Python. Em alternativa ao módulo `timeit`, pode utilizar as funções do módulo `time`, criando um segmento de código como o seguinte:*

```
import time
total_time = 0
for n in range(0, MAX):   #Max é um parâmetro a indicar (deve ser superior a 30)
    time_start = time.time()
    #colocar aqui o código a cronometrar
    total_time += time.time() - time_start
average_runtime = total_time/MAX
```

A vantagem de usar a o módulo `timeit` consiste na facilidade de especificar o número de vezes que queremos executar a instrução ou instruções e o facto de certas funções autónomas serem desativadas (por exemplo, a limpeza de memória para objetos que perdem referência (para mais detalhes, pesquise "Python garbage collection"), o que poderia desvirtuar os resultados. Mas se preferir, pode recorrer ao módulo como `time` indicado.

2. ♣

a) Implemente o algoritmo de **Pesquisa linear,** que permite pesquisar/encontrar um dado valor numa coleção de valores .

b) Implemente o **algoritmo de Pesquisa binária**, que permite pesquisar/encontrar um dado valor numa coleção de valores ordenados, neste caso, por ordem decrescente:

Algoritmo
Input: *uma sequência ordenada por ordem decrescente – seq - e um elemento - k.*
Output: *devolve True, se o elemento existe na sequência, ou False, caso contrário.*

- *Calcular o índice do elemento ao meio da sequência - meio;*
- *Se k =seq[meio], a resposta é True;*
- *Se k > seq[meio], procurar k na metade esquerda de seq;*
  - *Senão, procurar k na metade direita de seq.*

c) Usando sequências ordenada, criada por si, execute e cronometre os tempos médios de 100 execuções de cada pedaço de código para os testes necessários, preencha/crie uma tabela com os dados pedidos:

Tamanho da coleção	Número a pesquisar	Tempo médio usando 2.a)	Tempo médio usando 2.b)	Tempo médio usando "**in**"
10000	7777			
20000	15554			
30000	23331			
40000	31108			
50000	38885			
60000	46662			
70000	54439			
80000	62216			
90000	69993			
100000	77770			

Utilizando o módulo para criar gráficos de visualização indicado no primeiro exercício, crie um gráfico de pontos ou de linhas de modo a comparar os diversos tempos de execução entre as 3 possibilidades de pesquisa.

i. Observe o gráfico e tire conclusões sobre qual cresce mais rapidamente e porquê.
ii. Verifique se as ordens de crescimento destas pesquisas concordam com os valores teóricos indicados para as ordem de complexidade tipícas (O(n) para a pesquisa linear, O(n log n) para a pesquisa binária).

Notas:

- Para uma visualização com maior qualidade, deverá reescalar os valores finais de tempos para a escala de variação 1 a 10.
- Para um gráfico de pontos pode usar as seguintes instruções:

```
plt.scatter(x, c1, label= 'sequencial')
plt.scatter(x, c2, label= 'binaria')
plt.scatter(x, c2, label= 'binaria')
plt.legend()
```

3. Preencha a tabela seguinte:

Expressão	**Termo dominante**	**Notação Big $\mathcal{O}$**
5		
$5\,n + 15$		
$5 + 0.001\,n^2 + 0.025\,n$		
$200\,n + 0.001n^2$		
$0.01\,n + 200\,n^2$		
$n + n^{1.5}$		
$n + n^{0.5}$		
$\log_{10} n + \mathrm{n}\log_5 n$		
$n^3 \log_2 n + n(\log_2 n)^3$		

4. Determine π pelo método de *Monte Carlo*. Para isso, o método de Monte Carlo gera n pontos x e y aleatórios, com $x, y \in [0.0, 1.0[$, e os pontos m que se encontram dentro de $\sqrt{1-x^2}$ são contabilizados. Seja m o número de pontos contabilizados no interior de $\sqrt{1-x^2}$. Então, o valor de π será (aproximadamente) igual a $4 \times \frac{m}{n}$.

a) Justifique a igualdade $\pi = 4 \times \frac{m}{n}$.
Note que $f(x) = \sqrt{1-x^2}$ representa um quarto de círculo.

b) Calcule π para valores de n a variar entre 100 e 10.000, em passos de variação de 100.
Nota: Use o módulo `random` e função `random.random()` para gerar um valor aleatório em $[0.0, 1.0[$.

Compare os sucessivos valores de π calculados com este teste. O que consegue concluir?
Se a função `random.random` fosse uma distribuição uniforme perfeita, qual seria o resultado no limite assintótico, i.e, quando $n \to \infty$?