AULA 7 - ANÁLISE DA COMPLEXIDADE DE ALGORITMOS RECURSIVOS (ALGORITMOS SIMPLES)

Implemente os seguintes **algoritmos recursivos** e calcule o **número de operações** aritméticas (multiplicações ou divisões) executadas por cada algoritmo:

- **Cálculo da potência** x^n usando os seguintes métodos:

$$1^{o}\text{ método} \rightarrow x^n = x \times x^{n-1} \qquad 2^{o}\text{ método} \rightarrow x^n = \begin{cases} 1, \text{ se } n = 0 \\ \left(x^{n/2}\right)^2, \text{ se } n \text{ é par} \\ x \times \left(x^{n/2}\right)^2, \text{ se } n \text{ é ímpar} \end{cases}$$

  - Preencha a tabela com o valor da função (para x = 0.5) e o número de multiplicações para os sucessivos valores de n.

N	1º método (N)	Nº de Multiplicações	2º método (N)	Nº de Multiplicações
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
O(N)				

  - Analisando os dados da tabela qual é a ordem de complexidade de cada algoritmo?
  - Determine formalmente a ordem de complexidade de cada algoritmo, obtendo uma expressão que corresponda aos valores obtidos experimentalmente.

- **Verificação de potência** $a = b^n$

Implemente uma **função recursiva** eficiente para determinar se um número inteiro positivo é uma potência de outro número inteiro positivo.

O número a é uma potência do número b se a for múltiplo de b e o quociente da divisão inteira de a por b também for uma potência de b. Tenha em consideração que a unidade e o próprio b são potências de b ($1 = b^0$ e $b = b^1$).

  - Determine a ordem de complexidade do algoritmo desenvolvido.

NOME: **Nº MEC:**

- **Uma generalização dos Números de Fibonacci**

Implemente uma função recursiva para calcular uma generalização dos Números de Fibonacci usando a definição recorrente:

$$P(0) = 0$$

$$P(1) = 1$$

$$P(n) = 3 \times P(n-1) + 2 \times P(n-2), \text{ para } n > 1$$

Implemente um programa para executar a função para sucessivos valores de n e que permita determinar experimentalmente a **ordem de complexidade das operações de multiplicação** do seu algoritmo. Efetue a análise empírica da complexidade construindo uma tabela com o número de operações efetuadas para diferentes valores de n. Qual é a ordem de complexidade da função recursiva?

Uma forma de resolver problemas recursivos de maneira a evitar o cálculo repetido de valores, consiste em calcular os valores de baixo para cima, ou seja, de P(0) para P(n) e utilizar um *array* para manter os valores entretanto calculados. Este método designa-se por **programação dinâmica** e reduz o tempo de cálculo à custa da utilização de mais memória para armazenar valores intermédios.

- Usando a técnica de programação dinâmica, implemente uma função repetitiva alternativa e efetue a análise empírica da sua complexidade. Qual é a ordem de complexidade da função repetitiva?

- Faça a análise formal (no verso da folha) da complexidade de cada uma das funções e confirme as ordens de complexidade obtidas experimentalmente.

N	F. Recursiva	Nº de Multiplicações	P. Dinâmica	Nº de Multiplicações
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

O(N)		

NOME: **Nº MEC:**