AULA 7 - ANÁLISE DA COMPLEXIDADE DE ALGORITMOS RECURSIVOS

***** Entregue, num ficheiro ZIP, este guião preenchido e o código desenvolvido *****

Considere a seguinte relação de recorrência:

$$F(n) = \begin{cases} 1, \text{ se } n = 0 \text{ ou } n = 1 \text{ ou } n = 2 \\ F(n-1) + F(n-2) + \sum_{k=0}^{n-3} F(k) \times F(n-3-k), \text{ se } n > 2 \end{cases}$$

Função Recursiva

- Implemente uma **função recursiva** que use diretamente a relação de recorrência acima, **sem qualquer simplificação**.
- Construa um programa para executar essa função para **sucessivos valores de n** e que permita **contar o número total de multiplicações efetuadas** para cada valor de n.
- **Preencha a as primeiras colunas tabela seguinte** com o resultado da função recursiva e o número de multiplicações efetuadas para os sucessivos valores de n.

n	F(n) – Versão Recursiva	Nº de Multiplicações	F(n) – Versão de Programação Dinâmica	Nº de Multiplicações
0	1	0	1	0
1	1	0	1	0
2	1	0	1	0
3	3	1	3	1
4	6	3	6	3
5	12	7	12	6
6	26	16	26	10
7	57	36	57	15
8	125	80	125	21
9	279	177	279	28
10	630	391	630	36
11	1433	863	1433	45
12	3285	1904	3285	55
13	7584	4200	7584	66
14	17611	9264	17611	78
15	41109	20433	41109	91
16	96416	45067	96416	105
17	227088	99399	227088	120
18	536896	219232	536896	136
19	1273763	483532	1273763	153
20	3031485	1066464	3031485	171
21	7235573	2352161	7235573	190
22	17315668	5187855	17315668	210
23	41539777	11442175	41539777	231
24	99877435	25236512	99877435	253
25	240645375	55660880	240645375	276

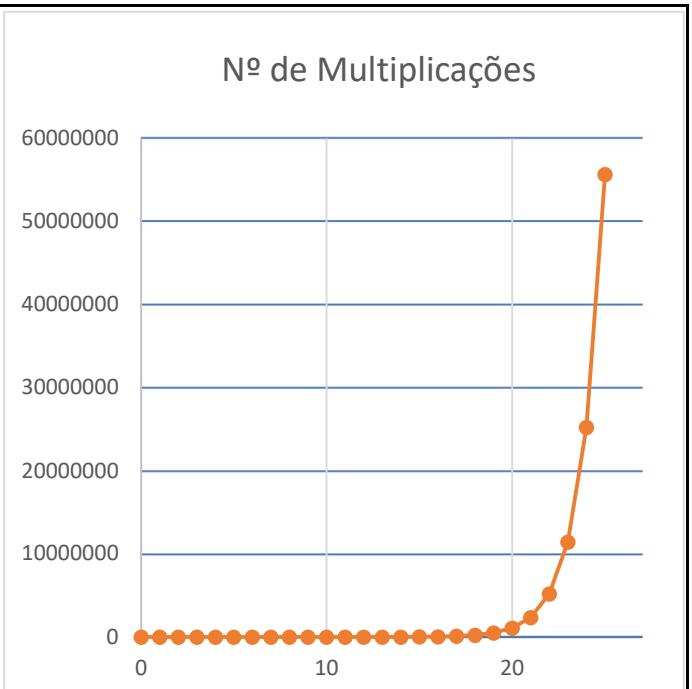
- Analisando os dados da tabela, estabeleça uma **ordem de complexidade** para a **função recursiva**.

Analisando o gráfico ao lado podemos fazer o seguinte cálculo:

$$\rightarrow \frac{número\ de\ multiplicações(n+1)}{número\ de\ multiplicações(n)} \approx 2{,}21$$

O número de multiplicações é feito pela multiplicação por 2,21 sucessivamente.

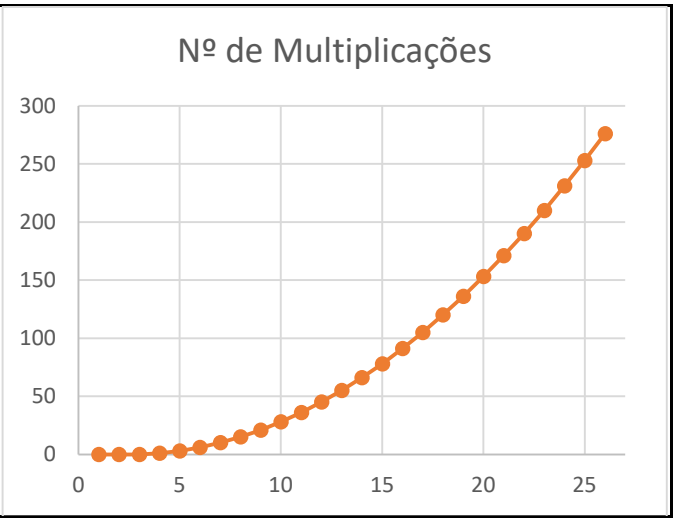
Logo, a ordem de complexidade é exponencial, ou seja, $O(2{,}21^n)$.



Programação Dinâmica

- Uma forma alternativa de resolver alguns problemas recursivos, para evitar o cálculo repetido de valores, consiste em efetuar esse cálculo de baixo para cima (*"bottom-up"*), ou seja, de **F(0)** para **F(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 os valores intermédios.
- Usando **programação dinâmica**, implemente uma **função iterativa** para calcular F(n). **Não utilize um array global.**
- Construa um programa para executar a função iterativa que desenvolveu para **sucessivos valores de n** e que permita **contar o número de multiplicações efetuadas** para cada valor de n.
- **Preencha as últimas colunas tabela anterior** com o resultado da função iterativa e o número de multiplicações efetuadas para os sucessivos valores de n.
- Analisando os dados da tabela, estabeleça uma **ordem de complexidade** para a **função iterativa**.

Analisando o gráfico ao lado com os dados da tabela acima relativamente ao número de multiplicações da função usando a programação dinâmica, verificamos que a sua ordem de complexidade é quadrática, ou seja, $O(n^2)$.
Chegamos a essa conclusão uma vez que, para os sucessivos valores de n, o algoritmo tem um comportamento geral de uma função quadrática.



Função Recursiva – Análise Formal da Complexidade

- Escreva uma **expressão recorrente** (direta) para o **número de multiplicações** efetuadas pela função recursiva F(n). Obtenha, depois, uma **expressão recorrente simplificada**. Note que $\sum_{k=0}^{n-3} \text{Mult}(k) = \sum_{k=0}^{n-3} \text{Mult}(n-3-k)$. **Sugestão:** efetue a subtração $\mathbf{Mult(n) - Mult(n-1)}$.

- Expressão recorrente direta do número de multiplicações:

$$Mult(n) = \begin{cases} 0, se\ n = 0, 1, 2 \\ Mult(n-1) + \sum_{k=0}^{n-3} (Mult(n-3-k) + Mult(k) + 1), se\ n \geq 3 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow Mult(n) = \begin{cases} 0, se\ n = 0, 1, 2 \\ Mult(n-1) + \sum_{k=0}^{n-3} Mult(n-3-k) + \sum_{k=0}^{n-3} Mult(k) + \sum_{k=0}^{n-3} 1, se\ n \geq 3 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow Mult(n) = \begin{cases} 0, se\ n = 0, 1, 2 \\ Mult(n-1) + (n-2) + 2 * \sum_{k=0}^{n-3} Mult(k), se\ n \geq 3 \end{cases}$$

- Recorrendo à expressão do 2º ramo de $Mult(n)$, podemos fazer a seguinte subtração:

$$Mult(n) - Mult(n-1) =$$

$$= \left(Mult(n-1) + (n-2) + 2 * \sum_{k=0}^{n-3} Mult(k)\right) - \left(Mult(n-3) + (n-3) + 2 * \sum_{k=0}^{n-4} Mult(k)\right)$$

$$= \left(Mult(n-1) + (n-2) + 2 * Mult(n-3) + 2 \sum_{k=0}^{n-4} Mult(k)\right) - \left(Mult(n-3) + (n-3) + 2 * \sum_{k=0}^{n-4} Mult(k)\right)$$

$= Mult(n-1) + 2 * Mult(n-3) - Mult(n-3) + (n-2) - (n-3)$

$= Mult(n-1) + Mult(n-3) + 1$

- Sendo assim, a expressão recorrente simplificada ficará na seguinte forma:

$Mult(n) - Mult(n-1) = Mult(n-1) + Mult(n-3) + 1 \Leftrightarrow$

$\Leftrightarrow Mult(n) = Mult(n-1) + Mult(n-1) + Mult(n-3) + 1 \Leftrightarrow$

$\Leftrightarrow Mult(n) = 2 * Mult(n-1) + Mult(n-3) + 1$

- A equação de recorrência obtida é uma **equação de recorrência linear não homogénea**. Considere a correspondente **equação de recorrência linear homogénea**. Determine as raízes do seu **polinómio característico** (Sugestão: use o **Wolfram Alpha**). Sem determinar as constantes associadas, escreva a **solução da equação de recorrência linear não homogénea**.

Equação de recorrência:

$$Mult(n) = 2 * Mult(n-1) + Mult(n-3) + 1 \Leftrightarrow$$
$$\Leftrightarrow Mult(n) - 2 * Mult(n-1) - Mult(n-3) = 1$$

, que representa uma equação linear não homogénea. Pelo que a sua solução será do tipo:

$$Mult(n) = Mult^1(n) + Mult^2(n)$$

- Cálculo de $Mult^1(n)$:
  1. Cálculo das raízes do polinómio característico da equação de recorrência linear não homogénea associada:

     $$Mult(n) - 2 * Mult(n-1) - Mult(n-3) = 0 \Leftrightarrow \Leftrightarrow x^3 - 2x^2 - 1 = 0$$

     Usando o Wolfram Alpha, chegamos ao resultado: $x \approx 2{,}21$

  2. Conclusão: $Mult^1(n) = 2{,}21^n * A$, sendo A uma constante associada.

- Cálculo de $Mult^2(n)$:
  1. $Mult^2(n)$ é calculado sabendo que: $f(n) = 1$

     $Mult^2(n) = B * n^r$, em que B é uma constante e r é a multiplicidade de grau 1 enquanto raíz característica da equação linear homogénea obtida acima. Sendo assim, neste caso, $r = 0$.

  2. Conclusão:
     $Mult^2(n) = C * n^0 = C$, sendo C uma constante associada

- Solução da equação de recorrência linear não homogénea:

$$Mult(n) = 2{,}21^n * A + C$$

- Usando a solução da equação de recorrência obtida acima, determine a **ordem de complexidade do número de multiplicações** efetuadas pela função recursiva. **Compare** a ordem de complexidade que acabou de obter com o resultado da **análise experimental**.

Através da expressão obtida na alínea anterior, conseguimos determinar a ordem de complexidade relativamente ao número de multiplicações efetuadas pela função recursiva.

Pela expressão $Mult(n) = 2{,}21^n * A + B$, podemos desprezar os termos de menor ordem, ficando assim apenas com $2{,}21^n$.

Por isso, chegamos logo à conclusão de que a ordem de complexidade do algoritmo é $O(2{,}21^n)$, o que representa uma complexidade exponencial.

A partir da análise experimental acima feita, chegamos a uma ordem de complexidade exponencial, $O(2{,}21^n)$, o que vai de acordo com o valor obtido na alínea anterior. Logo, podemos deduzir que, a partir das duas análises, a ordem de complexidade é exponencial, ou seja, $O(2{,}21^n)$.

Programação Dinâmica – Análise Formal da Complexidade

- Considerando o número de multiplicações efetuadas pela função iterativa, efetue a análise formal da sua complexidade. Obtenha uma **expressão exata e simplificada para o número de multiplicações** efetuadas.

Tendo em conta as condições iniciais $Mult(0) = Mult(1) = Mult(2) = 0$, temos que:

$$Mult(n) = \sum_{i=3}^{n}\left(\sum_{k=0}^{i-3} 1\right) = \sum_{i=3}^{n}(i-3+1) = \sum_{i=3}^{n}(i-2) = \sum_{i=1}^{n}(i-2) - \sum_{i=1}^{2}(i-2) =$$

$$= \sum_{i=1}^{n} i - \sum_{i=1}^{n} 2 - \sum_{i=1}^{2} i + \sum_{i=1}^{2} 2 = \frac{1}{2}n(n+1) - 2n - \frac{1}{2} * 2 * (2+1) + 2 * 2 =$$

$$= \frac{n^2+n}{2} - 2n + 1 = \frac{n^2-3n}{2} + 1$$

Através da expressão exata e simplificada obtida acima, podemos concluir que o número de multiplicações efetuadas tem ordem de complexidade quadrática, ou seja, $O(n^2)$.

- Usando a expressão obtida acima, determine a **ordem de complexidade do número de multiplicações** efetuadas pela função iterativa. **Compare** a ordem de complexidade que acabou de obter com o resultado da **análise experimental**.

Na alínea anterior, o resultado que obtivemos para saber a ordem de complexidade do número de multiplicações na função iterativa foi $Mult(n) = \frac{n^2-3n}{2} + 1$. Como na expressão o termo de maior ordem é $\frac{n^2}{2}$, facilmente descobrimos que a ordem de complexidade para o número de multiplicações é quadrática, ou seja, $O(n^2)$.

A ordem de complexidade descoberta a partir da análise experimental do algoritmo também foi quadrática, $O(n^2)$.

Podemos então concluir que ambas as ordens de complexidade em cada uma das análises estão de acordo com o que seria de esperar.