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:

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

- 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	0.5	1	Igual	0
2	0.25	2	ao método	2
3	0.125	3	1	3
4	0.0625	4		4
5	0.03125	5		4
6	0.015625	6		5
7	0.007813	7		5
8	0.003906	8		6
9	0.001953	9		5
10	0.000977	10		6
11	0.000488	11		6
12	0.000244	12		7
13	0.000122	13		6
14	0.000061	14		7
15	0.000031	15		7
16	0.000015	16		8
O(N)	N		$\log_2 N$	

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

- **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. Logaritmos (como diz na próxima página)

NOME: Pedro Velosa Teixeira Nº MEC: 84715

* 1 - o problema é decomposto em 2 mais pequenos)

- Cálculo da Potência

1º método

nº mínimo de multiplicações ←

$$M_1(N) = \begin{cases} 0, & \text{se } N = 0 \\ 1 + M(N-1), & \text{se } N > 0 \end{cases}$$

$\therefore M_1(N) = 1 + M_1(N-1) = 1 + 1 + M_1(N-2) = 2 + M_1(N-2) + \dots = K + M_1(N-K) = \dots = N + M_1(N-N) = N + M_1(0) = N$

↓ $O(N)$ Linear

———————————

2º método

$$M_2(N) = \begin{cases} 0, & \text{se } N = 0 \\ 1 + M(N/2), & \text{se } N \text{ par} \\ 2 + M(N/2), & \text{se } N \text{ ímpar} \end{cases}$$

Análise para N par

$\rightarrow M_2(N) = 1 + M(N/2) = 1 + 1 + M(N/4) = 1 + 1 + 1 + M(N/8) = \dots$

→ Reescrevendo,

$M_2(N) = 1 + M\left(\frac{N}{2^1}\right) = 2 + M\left(\frac{N}{2^2}\right) = 3 + M\left(\frac{N}{2^3}\right) = 4 + M\left(\frac{N}{2^4}\right)$

$= N + M\left(\frac{N}{2^N}\right) \longrightarrow \ell + M\left(\frac{N}{2^\ell}\right)$

Se $\ell = \log_2 N$, $\log_2 N + M\left(\frac{N}{2^{\log_2 N}}\right) = \log_2 N + M\left(\frac{N}{N}\right) =$

$= \log_2 N + M(1) = \log_2 N + 1 + M(1-1) = \log_2 N + 1 \longrightarrow O(N) = \log_2 N$

Logarítmico

Análise para N ímpar

→ Analogamente à análise para N par

$M_2(N) = 2 + M\left(\frac{N}{2^1}\right) = 2 + 2 + M\left(\frac{N}{2^2}\right) = \dots$

Logo $M_2(N) \approx 2N + M\left(\frac{N}{2^N}\right) \longrightarrow 2\ell + M\left(\frac{N}{2^\ell}\right)$

Se $\ell = \log_2 N$: $M_2(N) \approx \log_2(2N) + 1 \longrightarrow O(N) = \log_2(2N)$

Logarítmico

Ou utilizando o Master Theorem

→ Por análise empírica verifica-se que $M_2(N)$ é eventualmente não decrescente

$\rightarrow M(N) = 1 + M(N/2) \dashrightarrow a = 1,\ b = 2$ e $d = 0 \therefore a = b^d$

$\rightarrow M(N) = 2 + M(N/2) \dashrightarrow a = 1,\ b = 2$ e $d = 0$ ↗ $M(N) = O(N^0 \log N) = O(\log N)$

NOME: Pedro Veloso Teixeira   Nº MEC: 84715

- Verificação da Potência

→ Best case: $B_M(N) = 1$, o que ocorre quando verificamos se N é potência de N ou se 1 é potência de um dado número

→ Worst case: $W_M(N) = \log_b(N)$, analogamente à pesquisa numa Binary Search Tree (onde temos $O(N) = \log_2(N)$ porque em cada chamada recursiva * 1

- **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? Na tabela

- 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	0	0	Igual	0
1	1	0	a F.	0
2	3	2	Recursiva	2
3	11	4		4
4	39	8		6
5	139	14		8
6	495	24		10
7	1763	40		12
8	6279	66		14
9	22363	108		16
10	79647	176		18
11	283667	286		20
12	1010295	464		22

O(N)	Exponencial	Linear

NOME: Pedro Vitor Teixeira Nº MEC: 84715

• Usando Técnica de Programação Dinâmica

$r(N) = \sum_{i=2}^{N} 2 = 2 \sum_{i=2}^{N} = 2 \times (N-2+1) = 2 \times (N-1) = 2N-2$

$O(N)$ linear

• Versão recursiva

$r_2(N) = \begin{cases} N, & se\ N < 2 \\ 3 \times r_2(N-1) + 2 \times r(N-2) \end{cases}$

- Equação característica

$x^2 - 3x - 2 = 0 \Leftrightarrow x = \frac{3 \pm \sqrt{9 + 2 \times 4}}{2} \Rightarrow x = \frac{3 \pm \sqrt{17}}{2}$ → $x \approx 3.5$; $x \approx -0.5$

$r_2(N) = C_1 3.5^m + C_2 0.5^m \rightarrow O(3.5^m)$ Exponencial

$(C_1, C_2 \in \mathbb{R})$

• Versão recursiva (por indução)

- Vamos assumir que $T(N-1) = O(2^{N-1})$ Hipótese de indução
- Condição inicial

$N = 1$: $T(N-1) = O(2^0) = 1$ multiplicação ✓

- Tese da indução

$T(N-1) \Rightarrow T(N)$

$T(N) = T(N-1) + T(N-2) = O(2^{N-1}) + O(2^{N-2}) = O(2^N)$ qed

NOME: Pedro Valério Teixeira   Nº MEC: 84715