|  |  |  |  |
| --- | --- | --- | --- |
| **Nome:** | **Roberto Oliveira Graça** | **N.º Mec:** | **93020** |

Aula 5 - Análise da Complexidade de Algoritmos Recursivos

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

Implemente os seguintes **algoritmos recursivos** – **sem recorrer a funções de arredondamento** (floor e ceil) – e analise o **número de chamadas recursivas** executadas por cada algoritmo.

Deve utilizar **aritmética inteira**: n/3 é igual a e (n+2)/3 é igual a .

* **Preencha a tabela da página seguinte** com o resultado de cada função e o número de chamadas recursivas para os sucessivos valores de n.
* Analisando os dados da tabela, estabeleça uma ordem de complexidade para cada algoritmo?

|  |
| --- |
| T1(n)→ O(log n)  T2(n)→ O(n^log n)  T3(n)→ O(log n) |

* Escreva uma **expressão recorrente** para o **número de chamadas recursivas** efetuadas pela função **.** Obtenha, depois, uma **expressão exata e simplificada;** determine a sua **ordem de complexidade**. Compare a expressão obtida com a os dados da **tabela**. Sugestão: use o **desenvolvimento telescópico**.

|  |
| --- |
| F(n) = número de chamadas recursivas  F(n) = F(n/3) + 1  F(0) = 0  F(1) = 1 + 0 = 1  F(n) = F(n/3)+1 = [F(n/3^2) + 1] + 1 = … = F(n/3^k) + k, para (n/3^k)=1 →  F(n) = k + F(1)  → n = 3^k <=> k = log\_3(n)  F(n) = 1 + log\_3(n), n>0 → complexidade logarítmica  F(n) = aF(n/b) + f(n), a = 1, b = 3, f(n) = 1  F(1) = 0  Como f(n) pertence a O(n^d), onde d=0, verificando que a = b^d, ou seja, 1 = 3^0, pode-se concluir que a complexidade de F(n) é O((n^d)\*log(n)) → O(log n) |

**­­**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **n** | **T1(n)** | **Nº de Chamadas Recursivas** | **T2(n)** | **Nº de Chamadas Recursivas** | **T3(n)** | **Nº de Chamadas Recursivas** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 2 | 2 | 1 | 2 | 0 | 2 | 0 |
| 3 | 4 | 2 | 5 | 2 | 5 | 1 |
| 4 | 5 | 2 | 7 | 2 | 7 | 2 |
| 5 | 6 | 2 | 8 | 2 | 8 | 2 |
| 6 | 8 | 2 | 10 | 2 | 10 | 1 |
| 7 | 9 | 2 | 14 | 4 | 14 | 3 |
| 8 | 10 | 2 | 15 | 4 | 15 | 3 |
| 9 | 13 | 3 | 19 | 6 | 19 | 2 |
| 10 | 14 | 3 | 22 | 6 | 22 | 5 |
| 11 | 15 | 3 | 23 | 6 | 23 | 5 |
| 12 | 17 | 3 | 26 | 6 | 26 | 3 |
| 13 | 18 | 3 | 28 | 6 | 28 | 6 |
| 14 | 19 | 3 | 29 | 6 | 29 | 6 |
| 15 | 21 | 3 | 31 | 6 | 31 | 3 |
| 16 | 22 | 3 | 34 | 6 | 34 | 5 |
| 17 | 23 | 3 | 35 | 6 | 35 | 5 |
| 18 | 26 | 3 | 38 | 6 | 38 | 2 |
| 19 | 27 | 3 | 43 | 8 | 43 | 6 |
| 20 | 28 | 3 | 44 | 8 | 44 | 6 |
| 21 | 30 | 3 | 49 | 10 | 49 | 4 |
| 22 | 31 | 3 | 51 | 10 | 51 | 8 |
| 23 | 32 | 3 | 52 | 10 | 52 | 8 |
| 24 | 34 | 3 | 54 | 10 | 54 | 4 |
| 25 | 35 | 3 | 59 | 12 | 59 | 7 |
| 26 | 36 | 3 | 60 | 12 | 60 | 7 |
| 27 | 40 | 4 | 65 | 14 | 65 | 3 |
| 28 | 41 | 4 | 69 | 14 | 69 | 9 |

* Escreva uma **expressão recorrente** para o **número de chamadas recursivas** efetuadas pela função **. Considere o caso particular e** obtenha uma **expressão exata e simplificada;** determine a **ordem de complexidade** para esse caso particular. Compare a expressão obtida com a os dados da **tabela**. Sugestão: use o **desenvolvimento telescópico** e confirme o resultado obtido usando o **Teorema Mestre**.

|  |
| --- |
| F(n) = número de chamadas recursivas  F(n) = 2\*F(n/3) + 2  F(0) = 0;F(1) = 0; F(2) = 0;  F(n) = 2\*F(n/3) + 2 = 2\*[2\*F(n/3^2) + 2] +2 = … = (2^k)\*[F(n/3^k)]+[2^(k+1)]-2  → n/3^k = 1, logo F(n) = (2^k)\*[F(1)] + [2^(k+1)] -2  → k=log\_3(n)  F(n) = (2^k)\*0 + [2^(log\_3(n)+1)]-2 = [2^(log\_3(n)+1)]-2 = [2 \* 2^(log\_3(n))] – 2=  = [2 \* n^(log\_3(2))] -2  F(n) = aF(n/b) + f(n), a = 2, b = 3, f(n) = 2  F(1) = 1  Como f(n) pertence a O(n^d), onde d=0, verificando que a > b^d, ou seja, 2 > 3^0, pode-se concluir que a complexidade de F(n) é O(n^log\_b(a)) → O(n^log n)) |

* Pode **generalizar a ordem de complexidade** que acabou de obter para todo o n? **Justifique.**

|  |
| --- |
| Não, porque o arredondamento do logaritmo dos números não divisíveis por 3 é apenas uma aproximação. |

* Obtenha uma **expressão recorrente** para o **número de chamadas recursivas** efetuadas pela função

|  |
| --- |
| F(n) = número de chamadas recursivas  { 0 , se n<=2  F(n) = { F(n/3) + 1 , se n é múltiplo de 3  { 2\*F(n/3) + 2 , se n não é múltiplo de 3 |

* **Considere o caso particular e** obtenha uma **expressão exata e simplificada;** determine a **ordem de complexidade** para esse caso particular. Compare a expressão obtida com a os dados da **tabela**. Sugestão: use o **desenvolvimento telescópico** e confirme o resultado obtido usando o **Teorema Mestre**.

|  |
| --- |
| F(n) = F(n/3) + 1 = F(n/3^2) + 1 + 1 = … = F(n/3^k) + k  → k = log\_3(n)  F(n) = F(1) + k = 0 + log\_3(n) = log\_3(n)  F(n) = aF(n/b) + f(n), a = 1, b = 3, f(n) = 1  F(1) = 1  Como f(n) pertence a O(n^d), onde d=0, verificando que a = b^d, ou seja, 1 = 3^0, pode-se concluir que a complexidade de F(n) é O((n^d)\*log(n)) → O(log n) |

* Pode **generalizar a ordem de complexidade** que acabou de obter para todo o n? **Justifique.**

|  |
| --- |
| Não, porque o algoritmo desenvolvido é orientado a múltiplos de 3. Números fora desse conjunto não apresentam resultados tão precisos. |

* Atendendo às **semelhanças entre e**  estabeleça uma **ordem de complexidade para . Justifique.**

|  |
| --- |
| T3 é uma versão otimizada de T2, uma vez que, para múltiplos de 3, necessita de menos passos para chegar a um resultado. A regressão linear do número de chamadas de T3 é majorada pela regressão linear das chamadas recursivas de T2. |