|  |  |  |  |
| --- | --- | --- | --- |
| **Nome:** | **Raquel Resende Milheiro Pinto** | **N.º Mec:** | **92948** |

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?

|  |
| --- |
| Para efetuar uma analise melhor e mais facilmente, fiz três gráficos, tendo então estabelecido para a função ordem de complexidade logarítmica , para a função ordem de complexidade linear O(n) e para a função ordem de complexidade logarítmica .  Isto quer dizer que tanto como pertencem quando muito à ordem de complexidade e que pertence quando muito à ordem de complexidade O(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**.

|  |
| --- |
| C(n) = número de invocações sucessivas de  C(0) = 0 → caso base  Expressão recorrente e dada por:  Para o caso particular de    Se escolher-mos um k para que  , logo concluímos que o algoritmo é de complexidade logarítmica, .  Este resultado pode ser confirmado pelo Teorema Mestre, pois a = 1, b=3, c=1, f(n)=1 → d=0, logo . Assim verificamos que a complexidade é logarítmica, .  Através da expressão podemos obter o número de chamadas recursivas do algoritmo podemos ver que para:  n = 0 → C(0) = 0 n = 1 → C(1) = 1 n = 2 → C(2) = 1 n = 3 → C(3) = 2  n = 4 → C(4) = 2 n = 9 → C(9) = 3 n = 18 → C(18) = 3 n = 27 → C(27) = 4  Comparando a ordem de complexidade estabelecida anteriormente com a ordem de complexidade obtida , podemos dizer que esta complexidade é mais específica. A complexidade que obtive aqui não podia ser superior à estabelecia anteriormente, tendo sido verificado. |

**Tabela:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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**.

|  |
| --- |
| C(n) = número de invocações sucessivas de  C(0) = C(1) = C(2) = 0 → casos base  Triângulo de Pascal → e  Para o caso particular  A expressão recorrente é dada por:  Para este caso particular sabemos que então:  ou seja:  Usando o desenvolvimento telescópico temos:  =  =  , logo este algoritmo tem ordem de complexidade polinomial .  Através do Teorema Mestre podemos confirmar este resultado, pois a = 2, b = 3,  c = 0, f(n) = 2 → d = 0, logo o que significa que a ordem de complexidade é polinomial .  Através da expressão podemos obter o número de chamadas recursivas do algoritmo podemos ver que para:  n = 0,1,2 → C(0) = C(1) = C(2) = 0 n = 3 → C(3) = 2 n = 4 → C(4) = 2 n = 7 → C(7) = 4 n = 9 → C(9) = 6 n = 27 → C(27) = 14  n = 18 → C(18) = , isto acontece porque 18 não é potência de 3 e porque a expressão calculada é só para valores que sejam potências de 3, pois só ai é que .  Comparando a ordem de complexidade estabelecida anteriormente com a ordem de complexidade obtida , podemos dizer que esta complexidade é mais especifica. A complexidade que obtive aqui não podia ser superior à estabelecia anteriormente, tendo sido verificado. |

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

|  |
| --- |
| Como , é uma função suave ( é eventualmente decrescente e também pertence a ), como C(n) é uma função eventualmente não decrescente e como a função tem complexidade polinomial para os valores de n potências de 3 (b = 3) onde , então pela regra da suavidade, concluímos que é possível generalizar a ordem de complexidade que obtivemos anteriormente para todo o n. |

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

|  |
| --- |
| C(n) = número de invocações sucessivas de  Expressão recorrente é dada por: |

* **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**.

|  |
| --- |
| C(0) = C(1) = C(2) = 0 → casos base  Para o caso particular usamos a equação , pois todas as potências de 3 são múltiplos de 3.  , logo concluímos que o algoritmo é de complexidade logarítmica, .  Usando o Teorema Mestre, sabemos que a = 1, b = 3, c = 0 , f(n) = 1 → d = 0, logo . Assim segundo este teorema, concluímos que algoritmo é de complexidade logarítmica, , tal como quando usamos o desenvolvimento telescópico.  Através da expressão podemos obter o número de chamadas recursivas mais um (Nº de chamadas +1) do algoritmo podemos ver que para:  n = 0,1,2 → C(0) = C(1) = C(2) = 0 n = 3 → C(3) = 2 (1+1)  n = 9 → C(9) = 3 (2+1) n = 27 → C(27) = 4 (3+1)  Podemos ver também que a expressão não é valida para números que não sejam múltiplos de 3 (pois para isso teria de ser usado outro ramo):  n = 17 → C(17)=3 quando devia de ser 5 n = 22 → C(22)=3 quando devia de ser 8  Comparando a ordem de complexidade estabelecida anteriormente com a ordem de complexidade obtida , podemos dizer que esta complexidade é mais especifica. A complexidade que obtive aqui não podia ser superior à estabelecia anteriormente, tendo sido verificado. |

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

|  |
| --- |
| A ordem de complexidade anterior não se pode generalizar para todo o n, pois para o caso particular usamos a equação . Esta equação só e usada quando n for múltiplo de 3. Para obter a ordem de complexidade para outros casos (n não múltiplo de 3 e n diferente de 0, 1 ou 2) teríamos de usar a equação , prosseguindo com o desenvolvimento telescópico ou o Teorema Mestre. |

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

|  |
| --- |
| Atendendo às semelhanças entre e , podemos observar em relação ao esforço computacional que émajorante. Isto significa que a ordem de complexidade de não pode ser superior que à de uma vez que a ordem de complexidade de está limitada pela ordem de complexidade de . |