**SIN110 Algoritmos e Grafos - Exercício E1**

**Aluna:** Caroline Lopes Resek

**Matrícula:** 2017010113

1. Considere a seguinte função g, definida no conjunto dos números naturais, da seguinte forma:

g(0) = 0, g(1) = 1 e,

g(n) = 5g(n-1) – 6g(n-2), para n ≥ 2.

* 1. Escreva um algoritmo que computa g;

|  |  |
| --- | --- |
| Linha | Algoritmo |
| 1 | int computaG(int n){ |
| 2 | if(n==1 || n==0) return n; |
| 3 | else return 5\*computaG(n-1)-6\*computaG(n-2); |
| 4 | } |

* 1. Escreva e resolva uma recorrência para o tempo T(n) consumido pela função g(n) , determine seu comportamento assintótico.



Análise da árvore:

|  |  |
| --- | --- |
| Nível | Soma do nível |
| 0 |  |
| 1 |  |
| 2 |  |
| ... | ... |
| n-1 |  |
| n |  |

K é uma constante, portanto:

\*Comportamento assintótico :

1. Considere a seguinte algoritmo recursivo para calcular o máximo de um vetor A[e..d]:

Max(A, e, d)

1 se e = d

2 então devolve A[e]

3 senão x ← (e+d)/2

4 a ← Max(A, e ,x)

5 b ← Max(A, x+1, d)

6 se a > b

7 então devolve a

8 senão devolve b

Seja C(n) o número de vezes que a comparação da linha 6 é executada em uma chamada de Max(A,e,d), onde n = d – e + 1. Escreva uma recorrência que define C(n) e, determine sua ordem de crescimento. Justifique sua resposta.

|  |  |  |
| --- | --- | --- |
| Linha | Algoritmo | Contagem |
| --- | Max(A, e, d) | ---- |
| 1 | se e = d | n + (n+1) |
| 2 | então devolve A[e] | n |
| 3 | senão x ← (e+d)/2 | (n-1) |
| 4 | a ← Max(A, e ,x) | (n-1) |
| 5 | b ← Max(A, x+1, d) | (n-1) |
| 6 | se a > b | (n-1) |
| 7 | então devolve a | 0 |
| 8 | senão devolve b | (n-1) |

\* Para a contagem de cada linha considerou-se que o último elemento do vetor era o maior, portanto, a linha não é executada

\*Obs.: A linha 6 é executada n-1 vezes, com n>1, uma vez que, se n=1, a execução do código terminaria na linha 2.

\*Ordem de crescimento :

1. No método de classificação por inserção, uma otimização possível seria uma pesquisa mais rápida do local de inserção de uma chave através de busca binária. Escreva uma versão recursiva do algoritmo de ordenação por inserção implementando esta otimização considerando que o algoritmo deverá colocar em ordem crescente um vetor A[e..d]. Determine e compare a eficiência dos dois algoritmos para um melhor e para um pior caso.

* Algoritmo iterativo:

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Algoritmo | Pior caso | Melhor caso |
| ---- | Ordenação-Por-Inserção (A, n) | ----- | ----- |
| 1 | para j crescendo de 2 até n faça | n | n |
| 2 | x ← A[j] | n-1 | n-1 |
| 3 | i ← j−1 | n-1 | n-1 |
| 4 | enquanto i>0 e A[i] > x faça | 2+3+...+n-1 | 1+1+1+....+1 = n-1 |
| 5 | A[i+1] ← A[i] | 1+2+3+...+n-1 | 0 |
| 6 | i ← i−1 | 1+2+3+...+n-1 | 0 |
| 7 | A[i+1] ← x | n-1 | n-1 |

\*Pior caso: T(n) =

Comportamento assintótico:

\*Melhor caso : T(n) =

Comportamento assintótico: Ω(n)

* Algoritmo recursivo:

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Algoritmo | Pior caso | Melhor caso |
| ---- | Inserção\_Recursiva (A, n) | ----- | ----- |
| 1 | se n > 1 então | 1 | 1 |
| 2 | Inserção\_Recursiva (A, n-1) | T(n-1) | T(n-1) |
| 3 | x ← A[n] | 1 | 1 |
| 4 | i ← n−1 | 1 | 1 |
| 5 | enquanto i > 0 e A[i] > x faça | n | n |
| 6 | A[i+1] ← A[i] | n-1 | 0 |
| 7 | i ← i−1 | n-1 | 0 |
| 8 | A[i+1] ← x | 1 | 1 |

\*Pior caso:

A partir do cálculo realizado na tabela acima, obtém-se que:

F(n) = 4.1 +2.(n-1) + n +F(n-1) = 4 + 2n – 2 + n + F(n-1)

F(n) = F(n-1) + 3n + 2

Sabendo-se que em todas as operações há, ao menos, uma execução:

F(1) = 1 , para n=1

F(n) = F(n-1) + 3n + 2 , para n > 2

Por recorrência:

F(n) = F(n-1) + 3n + 2 = F(n-2) + [3(n-1) + 2] + [3n + 2]

F(n) = F(n-3) + [3(n-2) + 2] + [3(n-1) + 2] + [3n + 2]

[...]

F(n) = F(1) + [3(2) + 2] + [3(3) + 2] + ... + [3(n-1) + 2] + [3n + 2]

F(n) = 1 + 3[2 + 3+ ... + (n-1) + n] + 2(n-1) = 1 + {3[(n+2)(n-1)]}/2 + 2n – 2 = 2n -1 + {3[(– n + 2n - 2)]}/2 F(n) = 2n -1 + {3[( + n - 2)]}/2 = 2n -1 + (3 + 3n – 6)/2) = (4n -2 + 3 + 3n – 6)/2

F(n) =

Comportamento assintótico:

\*Melhor caso :

A partir do cálculo realizado na tabela acima, obtém-se que:

F(n) = 4.1 + n +F(n-1)

F(n) = F(n-1) + n + 4

Sabendo-se que em todas as operações há, ao menos, uma execução:

F(1) = 1 , para n=1

F(n) = F(n-1) + n + 4 , para n > 2

Por recorrência:

F(n) = F(n-1) + n + 4 = F(n-2) + [(n-1) + 4] + [n + 4]= F(n-3) + [(n-2) + 4] + [(n-1) + 4] + [n + 4]

[...]

F(n) = F(1) + [2 + 4] + [3 + 4] + ... + [(n-1) + 4] + [n + 4]

F(n) = 1 + [2 + 3+ ... + (n-1) + n] + 4(n-1) = 1 + [(n+2)(n-1)]/2 + 4n – 4 = 4n - 3 + ( – n + 2n - 2)/2 = 4n - 3 + ( + n - 2)/2 = (8n - 6 + n 2 + n - 2)/2 = ( + 9n – 8)/2

F(n) =

Comportamento assintótico: Ω()

* Comparação:
* Pior caso: em ambos os algoritmos (recursivo e iterativo) o desempenho foi o mesmo: .
* Melhor caso: o caso iterativo possui melhor desempenho, uma vez que ele possui complexidade enquanto o algoritmo recursivo possui complexidade .