Sistemas de Informação
6º Período
19/01/2021

IFMG - Ouro Branco
Diego Santos Seabra
0040251

Tarefa 2

Projeto e Análise de Algoritmos

Hospedado no Github

Contents

1) (1 pt) Calcule a ordem de complexidade para a função abaixo.

```
funcao1(A, n, x):
y ← 0
for k ← n to 1 do
    y ← A[k] + y * x
end for
return y
```

Solução:

Ordem de Complexidade

código	***custo***
$y \leftarrow 0$	1
$for(k \leftarrow n$ to $1)$	$n+1$
$y \leftarrow A[k] + y * x$	n
return y	1

$$
\begin{aligned}
O(n) &= 1 + (n+1) + n + 1 \\
&= \cancel{2}n + \cancel{3} \\
O(n) &= n
\end{aligned}
$$

Código Implementado

```c
////////////////////////////////////////
//                                    //
// Autor: Diego S. Seabra             //
// Matricula: 0040251                 //
//                                    //
////////////////////////////////////////

#include <stdio.h>

int funcao1(int A[], int n, int x)
{
    int y = 0;                          // custo: 1
    for (int k = n; k > 0; k--)         // custo: n+1
    {
        y = A[k] + y * x;               // custo: n
    }
    return y;                           // custo: 1
}

int main()
{
    int A[5] = { 3, 2, 3, 4, 5};
    int res = funcao1(A,2,5);
    printf("%i\n",res);
}
```

2) (2pts) Calcule a ordem de complexidade para a função abaixo

```
funcao2(n):
soma ← 0
for i ← 0 to n − 1 do
   for j ← 0 to n − 1 do
      for x ← 0 to n − 1 do
         soma ← soma + n − 1
      end for
   end for
end for
return soma
```

Solução:

Ordem de Complexidade

código	***custo***
soma $\leftarrow 0$	1
$for(i \leftarrow 0 \text{ to } n-1)$	$n+1$
$for(j \leftarrow 0 \text{ to } n-1)$	$n \times (n+1)$
$for(x \leftarrow 0 \text{ to } n-1)$	$n \times n \times (n+1)$
soma $\leftarrow$ soma $+ n - 1$	$n \times n \times n$
return soma	1

$$
\begin{aligned}
O(n) &= 1 + (n+1) + (n \times (n+1) + (n \times n \times (n+1)) + (n \times n \times n)) \\
&= 1 + (n+1) + (n^2 + n) + (n^3 + n^2) + n^3 + 1 \\
&= \cancel{2n} + \cancel{2n^2} + \cancel{2}n^3 + \cancel{3} \\
O(n) &= n^3
\end{aligned}
$$

Código Implementado

```
///////////////////////////////////////
//                                  //
// Autor: Diego S. Seabra           //
// Matricula: 0040251               //
//                                  //
///////////////////////////////////////

#include <stdio.h>

int funcao2(n)
{
    int soma = 0;                                  // custo: 1
    for (int i = 0; i ≤ n - 1; i++)               // custo: n+1
    {
        // printf("for1\n");
        for (int j = 0; j ≤ n - 1; j++)           // custo: n * n+1
        {
            // printf("for2\n");
            for (int x = 0; x ≤ n - 1; x++)       // custo: n * n * n+1
            {
                // printf("for3\n");
                soma = soma + (n - 1);             // custo: n * n * n
            }
        }
    }

    return soma;                                   // custo: 1
}

int main()
{
    int soma = funcao2(2);
    printf("%i\n", soma);
}
```

3) (3pts) Calcule a ordem de complexidade para a função abaixo

```
funcao3(A, n):
for j ← 3 to n do
    aux ← A[j]
    i ← j − 1
    while i > 1 && A[i] > aux do
        A[i + 1] ← A[i]
        i ← i − 1
    end while
    A[i + 1] ← aux
end for
```

Solução:

Ordem de Complexidade

código	***custo***
$for(j \leftarrow 3 \text{ to } n)$	n
$aux \leftarrow A[j]$	$n-1$
$i \leftarrow j-1$	$n-1$
$while(i > 1\&\&A[i] > aux$	$\sum_{j=3}^{n-1} j$
$A[i+1] \leftarrow A[i]$	$\sum_{j=3}^{n-1} j-1$
$i \leftarrow i-1$	$\sum_{j=3}^{n-1} j-1$
$A[i+1] \leftarrow aux$	$n-1$

$$
\begin{aligned}
O(n) &= n + (3 \times (n-1)) + (\textstyle\sum_{j=3}^{n-1} j) + (2 \times (\textstyle\sum_{j=3}^{n-1} j - 1)) \\
&= n + 3n - 3 + \left(\frac{n(n-1)}{2} - 3\right) + 2 \times \left(\frac{n(n-1)}{2} - n\right) \\
&= 4n - 3 + \frac{n^2 - n}{2} - 3 + 2 \times \frac{n^2 - n}{2} - n \\
&= \cancel{3n} - \cancel{6} + \cancel{3} \times \frac{n^2 - \cancel{n}}{\cancel{2}} \\
O(n) &= n^2
\end{aligned}
$$

Código Implementado

```c
//////////////////////////////////////
//                                  //
// Autor: Diego S. Seabra           //
// Matricula: 0040251               //
//                                  //
//////////////////////////////////////

#include <stdio.h>

#define n 6

// Mesma ideia do Insertion Sort
void funcao3(int A[n])
{
    int aux, i;                         // custo: 1
    for (int j = 2; j < n; j++)         // custo: n
    {
        aux = A[j];                     // custo: n-1
        i = j - 1;                      // custo: n-1

        while (i > 0 && A[i] > aux)     // custo: somatorio (j=3) (n-1) (j)
        {
            A[i + 1] = A[i];            // custo: somatorio (j=3) (n-1) (j-1)
            i = i - 1;                  // custo: somatorio (j=3) (n-1) (j-1)
        }
        A[i + 1] = aux;                 // custo: n-1
    }
}

void imprimeArranjo(int A[n])
{
    for (int i = 0; i < n; i++)
    {
        printf("%i ", A[i]);
    }
    printf("\n");
}

int main()
{
    int A[n] = {31, 41, 59, 26, 41, 58};
    printf("Antes: ");
    imprimeArranjo(A);
    funcao3(A);
    printf("Depois: ");
    imprimeArranjo(A);
}
```

4) (4 pts) Dadas n variáveis booleanas, escreva um algoritmo que gere todas as combinações possíveis. Por exemplo, para três variáveis deverá ser gerado: 000 - 001 - 010 - 011 - 100 - 101 - 110 - 111. Em seguida análise a complexidade deste algoritmo.

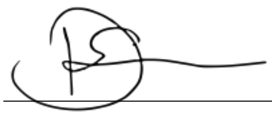
Solução:

Código Implementado

```c
////////////////////////////////////////
//                                    //
// Autor: Diego S. Seabra             //
// Matricula: 0040251                 //
//                                    //
////////////////////////////////////////

#include <stdio.h>
#include <math.h>

#define n 5

int main()
{
    int potenciaBinaria = pow(2, n);          // custo: 1
    int k;                                    // custo: 1

    printf("%i\n", potenciaBinaria);

    for (int i = 0; i < potenciaBinaria; i++)   // custo: 2^n + 1
    {
        for (int j = n - 1; j ≥ 0; j--)        // custo: 2^n * (n+1)
        {
            k = i >> j;                       // custo: 2^n * n

            if (k & 1)                        // custo: 2^n * n
                printf("1");                  // custo: 2^n * n
            else                              // custo: 2^n * n
                printf("0");                  // custo: 2^n * n
        }

        if (i != potenciaBinaria - 1)         // custo: 2^n
        {
            printf(" - ");                    // custo: 2^n
        }
    }
    printf("\n");                             // custo: 1
}
```

Ordem de Complexidade

$$
\begin{aligned}
O(n) &= 1 + 1 + (2^n + 1) + ((2^n + 1) \times (n + 1)) + (2^n + 1 \times n) \\
&\quad + ((2^n + 1) \times n) + ((2^n + 1) \times n) \\
&\quad + ((2^n + 1) \times n) + ((2^n + 1) \times n) + (2^n) + (2^n) + 1 \\
&= 3 + 2^n + 1 + ((2^n + 1) \times (n + 1)) + 5 \times ((2^n + 1) \times n) + 2 \times 2^n \\
&= \cancel{3} \times \cancel{2^{n+1}} n + \cancel{6n} + 2^{n+2} + \cancel{5} \\
O(n) &= 2^{n+2}
\end{aligned}
$$



Diego Santos Seabra
0040251