Análise de Casos: Exercícios Práticos

Prof. Me. Lucas Bruzzone

Aula 04

Exercício 1 - Soma de Array

Enunciado:
Faça uma análise de complexidade temporal do algoritmo abaixo que calcula a soma de todos os elementos de um array. Determine:

- A operação básica
- A função T(n) que conta as operações
- A notação assintótica (Θ, O, Ω)

Considere: Array de n elementos inteiros

Exercício 1 - Código

```
somaArray(arr, n):
    soma = 0
    for i = 0 to n-1:
        soma = soma + arr[i]
    return soma
```

Exercício 2 - Multiplicação de Matrizes

Enunciado:
Analise a complexidade temporal do algoritmo de multiplicação de duas matrizes quadradas $n \times n$. Determine:

- Qual a operação mais custosa
- Quantas vezes ela é executada
- A complexidade assintótica final

Considere: Matrizes A e B de dimensão $n \times n$, resultado em matriz C

Exercício 2 - Código

```
multiplicarMatrizes(A, B, n):
    for i = 0 to n-1:
        for j = 0 to n-1:
            C[i][j] = 0
            for k = 0 to n-1:
                C[i][j] += A[i][k] * B[k][j]
```

Exercício 3 - Encontrar Máximo

Enunciado:
Analise o algoritmo que encontra o maior elemento de um array.
Determine:

- O melhor caso possível
- O pior caso possível
- Se existe diferença entre os casos
- A complexidade geral do algoritmo

Considere: Diferentes organizações dos dados de entrada

Exercício 3 - Código

```
encontrarMaximo(arr, n):
    max = arr[0]
    for i = 1 to n-1:
        if arr[i] > max:
            max = arr[i]
    return max
```

Exercício 4 - Busca Condicional

Enunciado:
Analise o algoritmo que busca pela terceira ocorrência de um elemento x no array. O algoritmo para assim que encontra 3 elementos iguais a x. Determine:

- O melhor caso e sua complexidade
- O pior caso e sua complexidade
- Como o comportamento varia conforme a entrada

Considere: Diferentes distribuições do elemento x no array

Exercício 4 - Código

```
buscaCondicional(arr, n, x):
    count = 0
    for i = 0 to n-1:
        if arr[i] == x:
            count++
            if count >= 3:
                return i
    return -1
```

Exercício 5 - Potência Recursiva

Enunciado: Analise a complexidade do algoritmo recursivo para calcular potência ($base^{exp}$). Determine:

- A equação de recorrência T(n)
- O caso base da recursão
- A solução da recorrência
- A complexidade assintótica final

Considere: $exp \geq 0$ (expoente não negativo)

Exercício 5 - Código

```
potencia(base, exp):
    if exp == 0:
        return 1
    return base * potencia(base, exp-1)
```

Exercício 6 - Loop com Incremento Exponencial

Enunciado:
Analise a complexidade do algoritmo que usa um loop com incremento não-constante. A variável i dobra de valor a cada iteração. Determine:

- Quantas iterações o loop executa
- A relação entre n e o número de iterações
- A complexidade assintótica

Dica: Observe o padrão de crescimento da variável i

Exercício 6 - Código

```
algoritmoEspecial(n):
    count = 0
    i = 1
    while i <= n:
        count++
        i = i * 2
    return count
```

Exercício 7 - Análise Temporal e Espacial

Enunciado:
Analise tanto a complexidade temporal quanto espacial do algoritmo que cria e preenche uma matriz $n \times n$. Determine:

- A complexidade de tempo $T(n)$
- A complexidade de espaço $S(n)$
- Qual tipo de memória é utilizada

Considere: Alocação de matriz e preenchimento dos valores

Exercício 7 - Código

```
criarMatriz(n):
    matriz = new Array[n][n]
    for i = 0 to n-1:
        for j = 0 to n-1:
            matriz[i][j] = i + j
    return matriz
```

Exercício 8 - Fibonacci Iterativo

Enunciado:
Compare as complexidades temporal e espacial do algoritmo iterativo para calcular Fibonacci com a versão recursiva tradicional. Determine:

- A complexidade de tempo T(n)
- A complexidade de espaço S(n)
- As vantagens desta abordagem

Considere: Eficiência de memória e tempo de execução

Exercício 8 - Código

```
fibonaccilterativo(n):
    if n <= 1: return n

    a = 0
    b = 1
    for i = 2 to n:
        temp = a + b
        a = b
        b = temp
    return b
```

Exercício 9 - Busca em Matriz

Enunciado:
Analise o algoritmo de busca em matriz que tem dois parâmetros de entrada: m (linhas) e n (colunas). Determine:

- Como expressar a complexidade com dois parâmetros
- O melhor e pior caso
- A relação entre m, n e o desempenho

Considere: Matriz $m \times n$ e busca por elemento x

Exercício 9 - Código

```
buscaMatriz(matriz, m, n, x):
    for i = 0 to m-1:
        for j = 0 to n-1:
            if matriz[i][j] == x:
                return true
    return false
```

Exercício 10 - Loops Sequenciais

Enunciado:
Analise o algoritmo que possui múltiplos loops executados sequencialmente (não aninhados). Determine:

- A complexidade de cada seção separadamente
- Como combinar as complexidades
- A complexidade dominante final

Considere: Loops sequenciais com diferentes complexidades

Exercício 10 - Código

```
algoritmoComplexo(arr, n):
    // Primeira secao
    for i = 0 to n-1:
        print(arr[i])

    // Segunda secao
    for i = 0 to n-1:
        for j = 0 to n-1:
            if arr[i] > arr[j]:
                swap(arr[i], arr[j])

    // Terceira secao
    for i = 0 to n-1:
        arr[i] = arr[i] * 2
```