Questões de Revisão
Complexidade de Algoritmos e Somatórios
AEDS II

1 Parte 1: Complexidade de Algoritmos

Questão 1 (Conceitual)

Explique a diferença entre complexidade de tempo no melhor caso, caso médio e pior caso. Por que geralmente focamos na análise de pior caso?

Questão 2 (Notação Big-O)

Ordene as seguintes funções de complexidade da mais eficiente para a menos eficiente:

- $O(n \log n)$
- $O(2^n)$
- $O(n^2)$
- $O(\log n)$
- $O(n!)$
- $O(1)$
- $O(n)$

Questão 3 (Análise de Código)

Determine a complexidade de tempo do seguinte algoritmo:

```
def exemplo(n):
    soma = 0
    for i in range(n):
        for j in range(i):
            soma += i * j
    return soma
```

Questão 4 (Comparação)

Considere dois algoritmos: Algoritmo A com complexidade $O(n^2)$ e Algoritmo B com $O(n \log n)$. Para qual tamanho aproximado de entrada o Algoritmo B se torna mais eficiente? Justifique.

Questão 5 (Aplicação)

Analise a complexidade do algoritmo de busca binária e explique por que ela é $O(\log n)$. Quais são os pré-requisitos para usar busca binária?

2 Parte 2: Somatórios

Questão 6 (Somatórios Básicos)

Calcule os seguintes somatórios:

a) $\sum_{i=1}^{100} i$

b) $\sum_{i=1}^{n} 2i$

c) $\sum_{i=0}^{n-1} 1$

Questão 7 (Propriedades)

Demonstre que: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Questão 8 (Somatórios Aninhados)

Calcule: $\sum_{i=1}^{n} \sum_{j=1}^{i} 1$

Relacione este resultado com a complexidade de algoritmos com loops aninhados.

Questão 9 (Progressão Geométrica)

Calcule o somatório da progressão geométrica:

$$\sum_{i=0}^{n} 2^i$$

Questão 10 (Análise Prática)

Dado o seguinte código, expresse o número total de operações usando somatórios e depois simplifique:

```
def processar(n):
    contador = 0
    for i in range(1, n+1):
        for j in range(1, i+1):
            contador += 1
    return contador
```

3 Parte 3: Integração dos Conceitos

Questão 11 (Conversão)

Converta o seguinte somatório em notação Big-O:

$$\sum_{i=1}^{n}(i^2 + 3i + 5)$$

Questão 12 (Análise Completa)

Analise o seguinte algoritmo, expressando primeiro o número de operações como somatório e depois determinando a complexidade Big-O:

```
def misterioso(arr):
    n = len(arr)
    for i in range(n):
        for j in range(i, n):
            for k in range(j, n):
                print(arr[i] + arr[j] + arr[k])
```

Questão 13 (Otimização)

Dois programadores implementaram o mesmo algoritmo. O primeiro tem complexidade expressa por: $\sum_{i=1}^{n} i$

O segundo tem: $\frac{n^2}{2}$

Essas implementações têm a mesma complexidade assintótica? Justifique.

Questão 14 (Problema Aplicado)

Você precisa processar uma matriz $n \times n$. Compare as complexidades:

- Processar apenas a diagonal principal
- Processar apenas o triângulo superior (incluindo diagonal)

- Processar a matriz inteira

Expresse cada uma usando somatórios e depois em Big-O.

Questão 15 (Desafio)

Prove que $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ e explique como isso é útil na análise de algoritmos com loops aninhados dependentes.

Gabarito resumido disponível mediante solicitação.