Lista de Complexidade de Algoritmos — Respostas

1. Considere o seguinte algoritmo para encontrar o número máximo em uma lista de tamanho n:

```
max = lista[0]
for i in range(1, n):
    if lista[i] > max:
        max = lista[i]
```

   Qual é a complexidade de tempo deste algoritmo?

   $O(n)$

2. Considere o seguinte algoritmo para ordenar uma lista de tamanho n:

```
for i in range(n):
    for j in range(i, n):
        if lista[j] < lista[i]:
            temp = lista[i]
            lista[i] = lista[j]
            lista[j] = temp
```

   Qual é a complexidade de tempo deste algoritmo?

   $O(n^2)$

3. Considere o seguinte algoritmo para multiplicar duas matrizes $n * n$:

```
def matrix_multiplication(A, B):
    n = len(A)
    C = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

Qual é a complexidade de tempo deste algoritmo?

$O(n^3)$

4. Considere o seguinte algoritmo para encontrar o número de pares distintos de elementos em uma lista de tamanho n:

```
def count_pairs(lista):
    count = 0
    for i in range(n):
        for j in range(i+1, n):
            if lista[i] != lista[j]:
                count += 1
    return count
```

Qual é a complexidade de tempo deste algoritmo?

$O(n^2)$

5. Suponha que você tenha duas listas ordenadas de tamanho m e n, respectivamente, e deseja combinar essas duas listas em uma única lista ordenada. Considere o seguinte algoritmo para realizar esta combinação:

```
def merge(lista1, lista2):
    i = 0
    j = 0
    resultado = []
    while i < len(lista1) and j < len(lista2):
        if lista1[i] < lista2[j]:
            resultado.append(lista1[i])
            i += 1
        else:
            resultado.append(lista2[j])
            j += 1
    resultado += lista1[i:]
    resultado += lista2[j:]
    return resultado
```

Qual a complexidade deste algoritmo?

$O(m + n)$

7. Avalie a complexidade de tempo e espaço para ambos os algoritmos:

```
def fat(n):
    res = 1
    while n > 1:
        res *= n
        n -= 1
    return res

def fat_recursivo(n):
    if n <= 1:
        return 1
    return n * fat_recursivo(n - 1)
```

fat: $O(n)$ tempo e $O(1)$ espaço

fat_recursive: $O(n)$ tempo e $O(n)$ espaço

8. Dois algoritmos A e B possuem complexidade n^2 e 500n, respectivamente. Você utilizaria o algoritmo B ao invés do A? Em qual caso? Exemplifique

   B se estiver apenas analisando complexidade de tempo, contudo, sem saber a complexidade de espaço, não é possível tomar uma decisão fundamentada. Além disso, para pequenos valores de n, o primeiro algoritmo é mais interessante se tratando de complexidade de tempo.

9. Considere que o tempo de execução de um algoritmo A é $f(n) = n^2 + 400n + 50000n$ e de um algoritmo B é $g(n) = 3000n + 200n + 1000 + 2n$. Qual dos dois algoritmos possui a maior complexidade assintótica?

   O primeiro

10. Considere que o tempo de execução de um determinado algoritmo é: $f(n) = n^3 + 10n^2 + 2^n + 5^n + n * log(2n)$. A complexidade assintótica desse algoritmo é constante, logarítmica, linear, quadrática ou cúbica?

    Exponencial, devemos sempre considerar a maior.