Curso: Ciência da Computação
Disciplina: Analise e Complexidade de Algoritmo
Professor: Vilson
Instrumento Avaliativo

ATIVIDADE AVALIATIVA - -10/04/2025 – (3.0)

1 - Considere os trechos abaixo

A	B	C

`def func_a(n):`
`  for i in range(n):`
`  for j in range(n):`
`  print(i, j)`
	`def func_b(n):`
	`  i = 1`
	`  while i < n:`
	`  print(i)`
	`  i *= 2`
		`def func_c(arr):`
		`  for i in range(len(arr)):`
		`  for j in range(i, len(arr)):`
		`  print(arr[i], arr[j])`

Big-O, Big-Ω e Big-Θ.

a) Classifique a complexidade assintótica
b) Qual dos três algoritmos é mais eficiente para entradas grandes, Justifique.

2 - Considere a seguinte função que calcula o n-esimo numero de Fibonacci

```
def fibonacci_ineficiente(n):
    if n <= 1:
        return n
    return fibonacci_ineficiente(n - 1) + fibonacci_ineficiente(n - 2)
```

a) Analise a complexidade de tempo da função acima
b) Reescreva uma função mais eficiente dessa função implemente benckmarking comparando o tempo das duas versões para n=35

3 – O código abaixo conta quantos pares de elementos em uma lista de inteiros nums possuem valores **iguais**.

```
def contar_pares_iguais(nums):
    count = 0
    n = len(nums)
    for i in range(n):
        for j in range(i+1, n):
            if nums[i] == nums[j]:
                count += 1
    return count
```

Esse código possui complexidade **O(n^2)** por verificar todos os pares possíveis.

a) Justifique por que o algoritmo possui complexidade O(n^2).
b) Reescreva o algoritmo com complexidade **O(n)**, utilizando apenas estruturas básicas do Python.
c) Teste sua versão com a seguinte lista:

```
lista = [1, 2, 1, 2, 1]
```

Resultado esperado: **4**
(Pares iguais: (0,2), (0,4), (2,4), (1,3))