Tema da aula: Ordenação

Objetivo: Compreender e comparar o desempenho prático de diferentes algoritmos de ordenação (sorting) em Python, aplicando-os a listas de tamanhos variados. Você deve implementar ou utilizar diferentes métodos de ordenação, medir seus tempos de execução e analisar os resultados.

Descrição da Atividade

1. **Implemente ou reutilize** as funções de ordenação estudadas em aula:
    - `bubble_sort(lista)`
    - `insertion_sort(lista)`
    - `selection_sort(lista)`
    - `merge_sort(lista)`
    - `quick_sort(lista)`
    - `sorted(lista)` (Python — Timsort)
2. **Gere listas aleatórias** de tamanhos:
    - 100
    - 1.000
    - 10.000
    - 100.000
      (opcional: 1.000.000 se a máquina permitir)
3. **Meça o tempo de execução** de cada algoritmo para cada tamanho de lista, usando o módulo `time` ou `timeit`.
4. **Apresente os resultados** em forma de tabela e/ou gráfico (usando `matplotlib`, opcional).
5. **Analise criticamente os resultados**:
    - Quais algoritmos se comportam melhor para listas pequenas?
    - Quais escalam melhor para listas grandes?
    - Os resultados experimentais confirmam a complexidade teórica?

```
import random
import time

# ---------- Implementações Simples ----------

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
```

```
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivo = arr[len(arr)//2]
        esquerda = [x for x in arr if x < pivo]
        meio = [x for x in arr if x == pivo]
        direita = [x for x in arr if x > pivo]
        return quick_sort(esquerda) + meio + quick_sort(direita)

# ---------- Benchmark ----------

def medir_tempo(func, lista):
    inicio = time.time()
    func(lista)
    return time.time() - inicio

# ---------- Testes ----------

tamanhos = [100, 1000, 10000]
algoritmos = {
    "Bubble Sort": bubble_sort,
    "Insertion Sort": insertion_sort,
    "Selection Sort": selection_sort,
    "Merge Sort": merge_sort,
    "Quick Sort": lambda l: quick_sort(l.copy()),
    "Timsort (sorted)": lambda l: sorted(l)
}

for n in tamanhos:
    lista_base = [random.randint(0, 100000) for _ in range(n)]
    print(f"\nTamanho da lista: {n}")
    for nome, funcao in algoritmos.items():
        lista_teste = lista_base.copy()
        tempo = medir_tempo(funcao, lista_teste)
        print(f"{nome:<20} -> {tempo:.4f} segundos")
```