Implementando a versão paralelizada do Quicksort

Você recebeu um código Python que implementa o algoritmo de ordenação QuickSort . O programa gera uma lista de números aleatórios e os ordena usando o QuickSort. No entanto, para listas muito grandes, o processo pode ser lento, pois a ordenação é realizada de forma sequencial. Para melhorar o desempenho do programa, você deve paralelizar a execução do QuickSort utilizando threads. A ideia é dividir o trabalho de ordenação entre várias threads, permitindo que as sublistas (left e right) sejam ordenadas simultaneamente.

Objetivo

Modifique o código fornecido para implementar o QuickSort em threads separadas . Cada thread deve ser responsável por ordenar uma sublista específica (left ou right). Além disso, certifique-se de que as threads trabalhem de forma coordenada para evitar conflitos durante a ordenação.

Requisitos

1. Use a biblioteca threading do Python para criar threads.
2. Divida o trabalho de ordenação entre várias threads, onde cada thread processa uma sublista.
3. Garanta que as threads não conflitem ao acessar ou modificar estruturas de dados compartilhadas.
4. Combine os resultados das sublistas ordenadas corretamente para formar a lista final ordenada.
5. Mantenha a estrutura modular do código original, criando funções específicas para cada tarefa.
6. Adicione uma função principal (main) que inicie as threads e aguarde sua conclusão.
7. Exiba números antes e depois da ordenação, garantindo que a lista final esteja corretamente ordenada.
8. Meça o tempo de execução com e sem threads para comparar o desempenho.
9. Teste o programa com diferentes tamanhos de listas para verificar se a paralelização funciona corretamente.

Código a ser paralelizado

```python
import random

# Função principal do QuickSort

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[-1]
    left = [x for x in arr[:-1] if x <= pivot]  # Elementos menores ou iguais ao pivô
    right = [x for x in arr[:-1] if x > pivot]  # Elementos maiores que o pivô
    return quicksort(left) + [pivot] + quicksort(right)

# Função para gerar números aleatórios

def gerar_numeros_aleatorios(n=100, min_val=1, max_val=200):
    return [random.randint(min_val, max_val) for _ in range(n)]

# Função principal para testar o QuickSort

if __name__ == "__main__":
    numeros = gerar_numeros_aleatorios()

    print("Primeiros 10 números antes da ordenação:", numeros)
    numeros_ordenados = quicksort(numeros)
    print("Primeiros 10 números após a ordenação:", numeros_ordenados)
```