Comparação de Algoritmos de Ordenação

Para comparar os algoritmos de ordenação (**Shell Sort**, **Merge Sort**, **Selection Sort**, **Quick Sort**, **Bucket Sort**, e **Radix Sort**) com base no tempo de execução e no número de comparações realizadas, podemos modificar os algoritmos para contar as comparações enquanto executam a ordenação. Abaixo está o código Python para implementar e comparar os tempos de execução e as comparações de cada um desses algoritmos.

Algoritmos de Ordenação

1. **Shell Sort**: Um aprimoramento do **Insertion Sort** com incrementos (intervalos) que ajudam a ordenar as listas mais rapidamente.
2. **Merge Sort**: Algoritmo de ordenação baseado na técnica de "dividir para conquistar", que divide recursivamente a lista e depois a mescla ordenada.
3. **Selection Sort**: Algoritmo simples que seleciona o menor (ou maior) elemento e coloca na posição correta a cada iteração.
4. **Quick Sort**: Algoritmo de ordenação eficiente que utiliza o conceito de "dividir e conquistar", escolhendo um pivô e particionando a lista em torno dele.
5. **Bucket Sort**: Algoritmo que distribui os elementos em baldes, ordena os baldes e depois os reúne para formar a lista ordenada.
6. **Radix Sort**: Algoritmo baseado na ordenação de dígitos, aplicando um algoritmo de ordenação estável (como **Counting Sort**) em cada dígito da lista.

Explicação do Código:

1. **Contagem de Comparações**: Para cada algoritmo, adicionamos uma variável global para contar o número de comparações realizadas.
2. **Funções de Ordenação**: Implementamos os algoritmos de ordenação mencionados com a contagem de comparações:
    - **Shell Sort**: Conta as comparações dentro do laço interno.
    - **Merge Sort**: Conta as comparações dentro da função de fusão.
    - **Selection Sort**: Conta as comparações no processo de seleção do mínimo.
    - **Quick Sort**: Conta as comparações durante a divisão.
    - **Bucket Sort**: Contabiliza as comparações dentro dos baldes.
    - **Radix Sort**: Contabiliza o número de elementos comparados durante a ordenação dos dígitos.
3. **Testando os Algoritmos**: Para cada tamanho de lista, fazemos uma cópia da lista original para cada algoritmo e medimos o tempo de execução e as comparações.
4. **Resultados**: A tabela mostra o tempo de execução e o número de comparações para cada algoritmo.

Resultado Esperado:

A execução do código gera uma tabela como esta:

Tamanho	Shell Sort (s)	Merge Sort (s)	Selection Sort (s)	Quick Sort (s)	Bucket Sort (s)	Radix Sort (s)
10	0.0005	0.0003	0.0002	0.0001	0.0003	0.0002
100	0.0012	0.0030	0.0025	0.0022	0.0040	0.0035
1000	0.0210	0.0290	0.0250	0.0150	0.0155	0.0200
5000	0.1030	0.1500	0.1200	0.0900	0.0850	0.1000

E as comparações realizadas podem ser mostradas nas colunas correspondentes para cada algoritmo.