Uma equipe de cientistas de dados está trabalhando na análise de um conjunto de dados contendo a taxa de precipitação diária (em milímetros) coletada por diversas estações meteorológicas em uma região. Para identificar padrões e outliers, é crucial que os dados estejam ordenados. Dois algoritmos de ordenação foram considerados para essa tarefa: Bubble Sort e Selection Sort.

Considere a seguinte lista de taxas de precipitação coletadas em um dia específico:

precipitacao = [5.2, 1.8, 8.5, 3.1, 7.0, 0.9]

Após aplicar os algoritmos Bubble Sort e Selection Sort separadamente a essa lista, qual das seguintes afirmações sobre o número de trocas (swaps) realizadas por cada algoritmo é **CORRETA**?

A) O Bubble Sort realiza um número menor de trocas que o Selection Sort.

B) O Selection Sort realiza um número menor de trocas que o Bubble Sort.

C) Ambos os algoritmos realizam o mesmo número de trocas.

D) O número de trocas realizadas pelo Bubble Sort é sempre constante para qualquer lista de 6 elementos.

E) O número de trocas realizadas pelo Selection Sort depende dos valores específicos da lista, mas nunca excede o número de trocas do Bubble Sort para o mesmo conjunto de dados.

**Desenvolvimento da Solução**

Para resolver esta questão, precisamos simular a execução de ambos os algoritmos de ordenação na lista fornecida e contar o número de trocas realizadas por cada um.

**1. Bubble Sort:**

O Bubble Sort compara elementos adjacentes e os troca se estiverem na ordem incorreta. Esse processo é repetido várias vezes até que a lista esteja completamente ordenada.

Vamos acompanhar as iterações do Bubble Sort na lista [5.2, 1.8, 8.5, 3.1, 7.0, 0.9]:

* **Primeira Passada:**
  + Compara 5.2 e 1.8 -> Troca: [1.8, 5.2, 8.5, 3.1, 7.0, 0.9] (1 troca)
  + Compara 5.2 e 8.5 -> Sem troca: [1.8, 5.2, 8.5, 3.1, 7.0, 0.9]
  + Compara 8.5 e 3.1 -> Troca: [1.8, 5.2, 3.1, 8.5, 7.0, 0.9] (2 trocas)
  + Compara 8.5 e 7.0 -> Troca: [1.8, 5.2, 3.1, 7.0, 8.5, 0.9] (3 trocas)
  + Compara 8.5 e 0.9 -> Troca: [1.8, 5.2, 3.1, 7.0, 0.9, 8.5] (4 trocas)
* **Segunda Passada:**
  + Compara 1.8 e 5.2 -> Sem troca: [1.8, 5.2, 3.1, 7.0, 0.9, 8.5]
  + Compara 5.2 e 3.1 -> Troca: [1.8, 3.1, 5.2, 7.0, 0.9, 8.5] (5 trocas)
  + Compara 5.2 e 7.0 -> Sem troca: [1.8, 3.1, 5.2, 7.0, 0.9, 8.5]
  + Compara 7.0 e 0.9 -> Troca: [1.8, 3.1, 5.2, 0.9, 7.0, 8.5] (6 trocas)
  + Compara 7.0 e 8.5 -> Sem troca: [1.8, 3.1, 5.2, 0.9, 7.0, 8.5]
* **Terceira Passada:**
  + Compara 1.8 e 3.1 -> Sem troca: [1.8, 3.1, 5.2, 0.9, 7.0, 8.5]
  + Compara 3.1 e 5.2 -> Sem troca: [1.8, 3.1, 5.2, 0.9, 7.0, 8.5]
  + Compara 5.2 e 0.9 -> Troca: [1.8, 3.1, 0.9, 5.2, 7.0, 8.5] (7 trocas)
  + Compara 5.2 e 7.0 -> Sem troca: [1.8, 3.1, 0.9, 5.2, 7.0, 8.5]
  + Compara 7.0 e 8.5 -> Sem troca: [1.8, 3.1, 0.9, 5.2, 7.0, 8.5]
* **Quarta Passada:**
  + Compara 1.8 e 3.1 -> Sem troca: [1.8, 3.1, 0.9, 5.2, 7.0, 8.5]
  + Compara 3.1 e 0.9 -> Troca: [1.8, 0.9, 3.1, 5.2, 7.0, 8.5] (8 trocas)
  + Compara 3.1 e 5.2 -> Sem troca: [1.8, 0.9, 3.1, 5.2, 7.0, 8.5]
  + Compara 5.2 e 7.0 -> Sem troca: [1.8, 0.9, 3.1, 5.2, 7.0, 8.5]
  + Compara 7.0 e 8.5 -> Sem troca: [1.8, 0.9, 3.1, 5.2, 7.0, 8.5]
* **Quinta Passada:**
  + Compara 1.8 e 0.9 -> Troca: [0.9, 1.8, 3.1, 5.2, 7.0, 8.5] (9 trocas)
  + Compara 1.8 e 3.1 -> Sem troca: [0.9, 1.8, 3.1, 5.2, 7.0, 8.5]
  + Compara 3.1 e 5.2 -> Sem troca: [0.9, 1.8, 3.1, 5.2, 7.0, 8.5]
  + Compara 5.2 e 7.0 -> Sem troca: [0.9, 1.8, 3.1, 5.2, 7.0, 8.5]
  + Compara 7.0 e 8.5 -> Sem troca: [0.9, 1.8, 3.1, 5.2, 7.0, 8.5]

O Bubble Sort realizou um total de **9 trocas**.

**2. Selection Sort:**

O Selection Sort encontra o menor elemento na lista não ordenada e o troca com o elemento na primeira posição não ordenada. Esse processo é repetido para o restante da lista não ordenada.

Vamos acompanhar as iterações do Selection Sort na lista [5.2, 1.8, 8.5, 3.1, 7.0, 0.9]:

* **Primeira Iteração:**
  + Menor elemento: 0.9 (índice 5)
  + Troca 5.2 com 0.9: [0.9, 1.8, 8.5, 3.1, 7.0, 5.2] (1 troca)
* **Segunda Iteração (considerando a sublista [1.8, 8.5, 3.1, 7.0, 5.2]):**
  + Menor elemento: 1.8 (índice 1)
  + Não há troca (já está na posição correta). (0 trocas nesta iteração)
* **Terceira Iteração (considerando a sublista [8.5, 3.1, 7.0, 5.2]):**
  + Menor elemento: 3.1 (índice 3)
  + Troca 8.5 com 3.1: [0.9, 1.8, 3.1, 8.5, 7.0, 5.2] (2 trocas no total)
* **Quarta Iteração (considerando a sublista [8.5, 7.0, 5.2]):**
  + Menor elemento: 5.2 (índice 5)
  + Troca 8.5 com 5.2: [0.9, 1.8, 3.1, 5.2, 7.0, 8.5] (3 trocas no total)
* **Quinta Iteração (considerando a sublista [7.0, 8.5]):**
  + Menor elemento: 7.0 (índice 4)
  + Não há troca (já está na posição correta). (3 trocas no total)

O Selection Sort realizou um total de **3 trocas**.

**Comparação:**

O Bubble Sort realizou 9 trocas, enquanto o Selection Sort realizou 3 trocas para a lista dada. Portanto, o Selection Sort realizou um número menor de trocas que o Bubble Sort.

**Análise das Alternativas:**

* A) Incorreta.
* B) Correta. O Selection Sort realizou um número menor de trocas (3) que o Bubble Sort (9).
* C) Incorreta.
* D) Incorreta. O número de trocas no Bubble Sort depende da ordem inicial dos elementos. No pior caso (lista inversamente ordenada), o número de trocas para uma lista de n elementos é 2n(n−1)​. Para 6 elementos, isso seria 26×5​=15 trocas.
* E) Incorreta. Embora para este exemplo específico o Selection Sort tenha realizado menos trocas, a afirmação de que ele nunca excede o número de trocas do Bubble Sort para o mesmo conjunto de dados não é sempre verdadeira. Existem casos onde o Bubble Sort pode realizar menos trocas, especialmente se a lista estiver parcialmente ordenada.

**Conclusão:**

A alternativa correta é a **B**. O Selection Sort realizou um número menor de trocas que o Bubble Sort para a lista de taxas de precipitação fornecida. Essa diferença no número de trocas pode ser um fator a ser considerado ao escolher um algoritmo de ordenação para grandes conjuntos de dados em ciência de dados, onde a eficiência é crucial. O Selection Sort tende a realizar menos trocas em comparação com o Bubble Sort, especialmente em listas com muitos elementos fora de ordem, pois ele move cada elemento para sua posição final no máximo uma vez.