Algoritmo	Dataset	Números	Comparações	Trocas	Complexidade	Tempo
Bubble Sort	Ordenado	n = 1000	999	0	O(n)	~0
		n = 10000	9999	0	O(n)	~0
	Reverso	n = 1000	499500	499500	O(n²)	~0
		n = 10000	49995000	49995000	O(n²)	0.385
	Aleatório	n = 1000	498939	256123	O(n²)	0.003
		n= 10000	49991597	24921506	O(n²)	0.359
Insertion Sort	Ordenado	n = 1000	999	999	O(n)	~0
		n=10000	9999	9999	O(n)	~0
	Reverso	n = 1000	499500	500499	O(n²)	~0
		n=10000	49995000	50004999	O(n²)	0.155
	Aleatório	n = 1000	257116	257122	O(n²)	~0
		n=10000	24931489	24931505	O(n²)	0.097
Selection Sort	Ordenado	n = 1000	499500	0	O(n²)	~0
		n=10000	49995000	0	O(n²)	0.111
	Reverso	n = 1000	499500	500	O(n²)	~0
		n=10000	49995000	5000	O(n²)	0.117000
	Aleatório	n = 1000	499500	991	O(n²)	0.001
		n=10000	49995000	9992	O(n²)	0.110000

1. **Escalabilidade (n=1000 x n=10000):**
   - **Ordenados (Melhor Caso):** para Bubble e Insertion Sort, o tempo de execução e as operações (comparações/movimentos) crescem linearmente (*O(n)*), o que é confirmado pelos resultados (aumento de ~10x nas operações, tempo quase constante devido à rapidez). Selection Sort permanece *O(n²)* nas comparações, mas *O(1)* ou *O(n)* nas trocas, resultando em um tempo *O(n²)*.
   - **Reversos e Aleatórios (Pior/Médio Caso)**: todos os três algoritmos exibem um comportamento quadrático (*O(n²)*). Aumentar N por um fator de 10 aumenta o número de comparações/trocas/movimentos por um fator de aproximadamente 100 (10^2). Isso se reflete drasticamente no tempo de execução, que aumenta significativamente (aproximadamente 100x, embora variações ocorram devido a fatores de hardware e otimizações do compilador).

2. **Melhor e Pior Cenário:**
   - **Melhor Caso (Ordenado):** Bubble Sort (com flag) e Insertion Sort são os mais rápidos (*O(n)*). Selection Sort é o mais lento *(O(n²))* devido às comparações.
   - **Pior Caso (Reverso):** Todos são *O(n²)*. Insertion Sort tende a ser ligeiramente mais rápido que Bubble Sort devido a menos movimentações totais de dados (menos atribuições por "movimento" vs. "troca"). Selection Sort tem um número de comparações semelhante, mas se destaca pelo número mínimo de trocas (*O(n)*), embora o tempo total ainda seja *O(n²)*.
   - **Caso Aleatório:** Todos são *O(n²)* em tempo. Insertion Sort geralmente supera Bubble Sort devido a menos comparações e movimentações de dados mais eficientes. Selection Sort permanece *O(n²)* em comparações e tempo, mas *O(n)* em trocas.

3. **Comparações vs. Trocas/Movimentos:**
   - **Bubble Sort:** O número de trocas é geralmente menor ou igual ao número de comparações, sendo zero no melhor caso e igual no pior caso.
   - **Insertion Sort:** O número de movimentos está fortemente correlacionado ao número de comparações (no while). No melhor caso, faz *O(n)* comparações e *O(n)* movimentos. No pior/médio, ambos são *O(n²)*.
   - **Selection Sort:** Realiza sempre *O(n²)* comparações, mas garante um número mínimo de trocas (*O(n)*). É o algoritmo com a menor quantidade de escritas na memória (trocas) entre os três.

Conclusão Final:

Para datasets pequenos ou quase ordenados, Insertion Sort e Bubble Sort (com flag) são eficientes. Para datasets maiores e não ordenados, todos os três algoritmos quadráticos (*O(n²)*) se tornam impraticáveis rapidamente, como visto na passagem de n=1000 para n=10000. Selection Sort é notável por minimizar trocas, enquanto Insertion Sort é frequentemente o mais rápido entre os três em casos médios devido a menos comparações e movimentações eficientes. Bubble Sort, mesmo com a otimização, geralmente é o menos eficiente nos casos médio e pior. Para N grande, algoritmos como Merge Sort ou Quick Sort (*O(n log n)*) são preferíveis.