Relatório dos algoritmos de ordenação

1. Implementação

Bubble Sort: percorre repetidamente o array comparando elementos vizinhos e trocando-os se estiverem fora de ordem. A cada passagem, o maior elemento é passado para o final. O processo se repete até que nenhuma troca seja necessária, o que o torna simples, mas bem ineficiente para grandes conjuntos de dados.

Selection Sort: divide o array em duas partes: a parte ordenada e a não ordenada. A cada iteração, encontra o menor elemento da parte não ordenada e o move para o final da parte ordenada. Apesar de fazer poucas trocas, o número de comparações é sempre o máximo possível.

Insertion Sort: constrói o array ordenado elemento por elemento, inserindo cada novo elemento na posição correta dentro da parte já ordenada. Funciona de forma semelhante a organizar cartas na mão: eficiente para arrays pequenos ou quase ordenados, mas lento para grandes entradas desordenadas.

Quick Sort: escolhe um elemento pivô e particiona o array em dois subgrupos — elementos menores e maiores que o pivô — e aplica o processo recursivamente em cada parte. Em geral, é muito eficiente, mas seu desempenho depende bastante da escolha do pivô.

Merge Sort: divide o array ao meio recursivamente até ter subarrays de um único elemento, e então os mescla de volta em ordem. Por dividir e conquistar de forma equilibrada, garante complexidade O(n log n) em todos os casos, no entanto, consome memória adicional.

2. Resultados obtidos

		Bubble sort	Selection sort	Insertion sort	Quick sort	Merge sort
10 números (ordem aleatória)	Tempo de execução (ms)	0.00	0.00	0.00	0.00	0.00
	Nº de trocas	17	5	17	27	42
	Nº de comparações	63	45	26	33	24
100 números (ordem aleatória)	Tempo de execução (ms)	0.02	0.01	0.01	0.01	0.01
	Nº de trocas	2479	91	2479	430	756
	Nº de comparações	9801	4950	2578	749	558
1000 números (ordem aleatória)	Tempo de execução (ms)	2.91	0.52	0.39	0.06	0.14
	Nº de trocas	248078	992	248078	5801	10722
	Nº de comparações	946053	499500	249077	10311	8724
10 números (ordem inversa)	Tempo de execução (ms)	0.00	0.01	0.00	0.00	0.00
	Nº de trocas	45	5	45	29	33
	Nº de comparações	81	45	54	45	15
100 números (ordem inversa)	Tempo de execução (ms)	0.04	0.01	0.02	0.02	0.01
	Nº de trocas	4950	50	4950	2549	514
	Nº de comparações	9801	4950	5049	4950	316
1000 números (ordem inversa)	Tempo de execução (ms)	2.33	0.77	1.06	1.26	0.09
	Nº de trocas	499500	500	499500	250499	6930
	Nº de comparações	998001	499500	500499	499500	4932

3. Comparação dos resultados

Os dados batem com a teoria de cada algoritmo. Para entradas aleatórias com 1000 elementos, o Bubble Sort e o Insertion Sort se mostram os mais lentos (2,91 ms e 0,39 ms respectivamente) e com o maior número de trocas (~248 mil), enquanto o Quick Sort se destaca com apenas 0,06 ms e 5801 trocas. O Merge Sort, apesar de realizar mais trocas que o Quick Sort na entrada aleatória, mantém um tempo consistente e muito baixo (0,14 ms). Nos arrays invertidos, o pior caso para a maioria dos algoritmos, o Bubble Sort piora bastante (2,33 ms, ~499 mil trocas), o Quick Sort também sofre (1,26 ms), mas o Merge Sort se destaca com apenas 0,09 ms e 6930 trocas para 1000 elementos, reforçando sua estabilidade independente da entrada. O Selection Sort apresenta comportamento intermediário, com poucas trocas mas muitas comparações em todos os cenários.