TDE 4 – Algoritmos de Ordenação

Estudantes: Felipe Fumagalli, Nicole Fatuch, Jose Gabriel Kojo

Relatório de Resultados

A tabela representa o tempo (em nanossegundos) de execução de cada arquivo.

Arquivo	**BubbleSort**	**InsertionSort**	**QuickSort**
Aleatório 100	199300	57400	59300
Aleatório 1000	13718900	1746900	391700
Aleatório 10000	132943400	29174000	1167400
Crescente 100	7400	43300	24700
Crescente 1000	342600	61100	501200
Crescente 10000	12261800	134000	59839900
Decrescente 100	8500	8500	17700
Decrescente 1000	846000	104700	432300
Decrescente 10000	80162400	9549100	40524700

Lendo Dados em Ordem Aleatória:

- **QuickSort** foi o algoritmo mais rápido para todos os tamanhos de conjuntos aleatórios. Isso ocorre porque ele é projetado para lidar com grandes volumes de dados de forma eficiente, dividindo-os em partes menores e ordenando-as rapidamente. Já o **BubbleSort** e o **InsertionSort** ficaram mais lentos à medida que o tamanho dos dados aumentou, porque eles precisam fazer muitas comparações e trocas em cada etapa.

Lendo Dados em Ordem Crescente:

- Para 100 elementos, o **BubbleSort** teve um desempenho razoável, mas conforme os dados cresceram, **InsertionSort** se destacou como a melhor opção. **QuickSort** apresentou um desempenho pior com dados já ordenados porque sua abordagem para dividir os dados não é eficaz neste cenário específico, resultando em muitas operações desnecessárias.

Lendo Dados em Ordem Decrescente:

- **InsertionSort** foi o mais eficaz. Ele funciona bem nesses casos porque a quantidade de ajustes que precisa fazer é mais previsível. O **QuickSort** superou **BubbleSort** nos conjuntos de 1000 e 10000, mas teve uma queda de desempenho devido ao tipo de ordenação inicial, o que exigiu mais ajustes do que em cenários aleatórios.

Conclusão:

- **QuickSort** foi o mais eficiente para conjuntos grandes e aleatórios devido à sua capacidade de dividir rapidamente os dados e ordená-los.
- **InsertionSort** teve bons resultados para conjuntos pequenos ou quando os dados já estavam parcialmente ordenados, pois ele realiza menos movimentos nessas situações.

- **BubbleSort** foi consistentemente o menos eficiente, já que precisa realizar muitas operações repetitivas sem uma estratégia de divisão mais inteligente.

O tamanho dos dados e a ordem inicial foram os principais fatores que impactaram o desempenho dos algoritmos. Algoritmos como o QuickSort se beneficiam de divisões eficazes de dados, enquanto o InsertionSort é mais rápido para listas pequenas ou quase ordenadas.