-Introdução

Esta atividade teve como objetivo implementar e comparar os principais algoritmos clássicos de ordenação:

Bubble Sort

Selection Sort

Insertion Sort

Quick Sort

Merge Sort

Os algoritmos foram desenvolvidos na linguagem C, utilizando vetores de diferentes tamanhos (100, 1000 e 10000 elementos), com medição de tempo por meio da função `clock()` da biblioteca `<time.h>`.

-Implementação

2.1 Bubble Sort

Implementado com dois laços aninhados, comparando elementos adjacentes e realizando trocas quando necessário.

Características:

Estratégia: Troca sucessiva

Complexidade: $O(n^2)$

Algoritmo simples, porém, ineficiente para grandes volumes

2.2 Selection Sort

Seleciona o menor elemento da parte não ordenada e o posiciona corretamente.

Características:

Estratégia: Seleção do menor valor

Complexidade: $O(n^2)$

Menos trocas que o Bubble, mas mesmo número de comparações

2.3 Insertion Sort

Insere cada elemento na posição correta dentro da parte já ordenada.

Características:

Estratégia: Inserção ordenada progressiva

Complexidade: $O(n^2)$

Melhor desempenho quando o vetor já está parcialmente ordenado

2.4 Quick Sort

Implementado utilizando estratégia de divisão e conquista usando escolha de pivô, particionamento e chamadas recursivas

Características:

Complexidade média: $O(n \log n)$, muito eficiente na prática e pode ter pior caso $O(n^2)$, dependendo do pivô

2.5 Merge Sort

Ele Também baseado em divisão e conquista, nesse caso divide o vetor em duas partes e ordena recursivamente e realiza intercalação (merge)

Características:

Complexidade: $O(n \log n)$, estável usa memória auxiliar

3. Resultados Obtidos

Os testes foram realizados com vetores 100, 1000 e 10000.

Tempo médio observado:

Algoritmo	100	1000	10000
Bubble Sort	~ 0.000033 s	~ 0.002317 s	~ 0.254993 s
Selection Sort	~ 0.000025 s	~ 0.001408 s	~ 0.147937 s
Insertion Sort	~ 0.000011 s	~ 0.000892 s	~ 0.079727 s
Quick Sort	~ 0.000019 s	~ 0.000133 s	~ 0.001261 s
Merge Sort	~ 0.000017 s	~ 0.000180 s	~ 0.002107 s

Obs: Os valores podem variar conforme o hardware

4. Comparação de Desempenho

4.1 Para 100 elementos

-Todos os algoritmos apresentaram desempenho semelhante.

-Diferença pouco perceptível.

-Algoritmos $O(n^2)$ ainda são viáveis para pequenos volumes.

4.2 Para 1000 elementos

-Algoritmos $O(n^2)$ começaram a apresentar aumento significativo de tempo.

-Quick e Merge mostraram desempenho muito superior.

4.3 Para 10000 elementos

-Bubble, Selection e Insertion tornaram-se significativamente mais lentos.

-Quick Sort foi o mais rápido.

-Merge Sort apresentou desempenho muito próximo ao Quick.

5. Análise Comparativa

Algoritmos $O(n^2)$

-Bubble Sort

-Selection Sort

-Insertion Sort

Crescimento quadrático, o tempo aumenta drasticamente conforme o tamanho do vetor cresce. Para 10.000 elementos, tornam-se pouco eficientes.

Algoritmos O(n log n)

-Quick Sort

-Merge Sort

Crescimento log-linear, eles escalam muito melhor para grandes volumes de dados, onde ficam até 200 vezes mais rápidos que os algoritmos quadráticos para 10.000 elementos.

6. Conclusão

A implementação permitiu observar, na prática, a diferença entre algoritmos de complexidade quadrática e log-linear.

Principais conclusões:

-Para vetores pequenos, qualquer algoritmo é aceitável.

-Para grandes volumes, algoritmos $O(n^2)$ tornam-se inviáveis.

-Quick Sort apresentou o melhor desempenho geral.

-Merge Sort mostrou-se estável e consistente.