CÂMPUS CAMPO MOURÃO

Ministério da Educação

Universidade Tecnológica Federal do Paraná - Câmpus Campo Mourão

Algoritmos e Estruturas de Dados

Prof. Dr. Rodrigo Hübner

ERIK LOHAN DA SILVA ZAVAN

WAGNER SILVA COSTA

SUMÁRIO

Introdução

Este relatório apresenta os resultados da implementação e análise comparativa dos algoritmos de ordenação (*Selection Sort*, *Bubble Sort* padrão e otimizado, *Insertion Sort* padrão e otimizado) e busca (sequencial e binária), conforme solicitado na prova 02 da disciplina de Algoritmos e Estruturas de Dados I.

Metodologia

1. Geração de Dados

Foram criados três arquivos binários contendo valores inteiros aleatórios (entre 0 e 9999).

- ➔ pequeno.bin: 40.000 elementos (~ 1 segundo de ordenação)
- ➔ medio.bin: 100.000 elementos (~ 30 segundos de ordenação)
- ➔ grande.bin: 250.000 elementos (~3 minutos de ordenação)

2. Implementação

Foram implementados:

- ➔ 5 algoritmos de ordenação (incluindo versões otimizadas)
- ➔ 2 algoritmos de busca (sequencial e binária)
- ➔ Funções auxiliares para medição de tempo e geração de arquivos
- ➔ Script em python usando *matplotlib* para gerar os gráficos

3. Coleta de Dados

Os tempos de execução foram registrados no arquivo *tempos.csv*, que será utilizado para a análise.

Resultados e Análise

Dados de Ordenação (tempos.csv)

Algoritmo	Pequeno (s)	Médio (s)	Grande (s)
selection_sort	2.3340	11.2790	64.8720
bubble_sort	5.2450	34.8000	183.6910
bubble_sort_otimizado	5.4820	31.0750	188.9720
insertion_sort	1.4600	6.5790	37.0440
insertion_sort_otimizado	1.5570	6.5270	37.3590

Análise Crítica

1. Selection Sort vs Bubble Sort

O *Selection Sort* foi consistentemente mais rápido que o *Bubble Sort* em todos os tamanhos de entrada. Para o arquivo grande, *Selection Sort* foi quase **3x mais rápido** que *Bubble Sort.*

2. Versões Otimizadas

O *Bubble Sort* otimizado não apresentou melhoria significativa, o que já era esperado, pois estamos lidando com entradas aleatórias que não apresentam muitas sessões ordenadas e o mesmo pode ser concluído em relação ao *Insertion Sort*.

3. Insertion Sort

Foi o algoritmo mais rápido em todos os cenários, especialmente para entradas pequenas. Fica claro que esse *Insertion Sort* entrega a melhor performance mesmo com entradas aleatórias e com sua versão otimizada ele também é ideal para usos em aplicações que lidam com dados já ordenados.

Dados de Trocas (tempos.csv)

Algoritmo	Pequeno	Médio	Grande
selection_sort	3	9	24
bubble_sort	3,996	24,956	156,466
bubble_otimizado	3,996	24,956	156,466
insertion_sort	3,996	24,956	156,466
insertion_otimizado	3,996	24,956	156,466

1. SelectionSort

Mostra pouquíssimas trocas (3-24), confirmando sua característica de fazer O(n) trocas, isso explica seu tempo relativamente bom apesar da complexidade $O(n^2)$

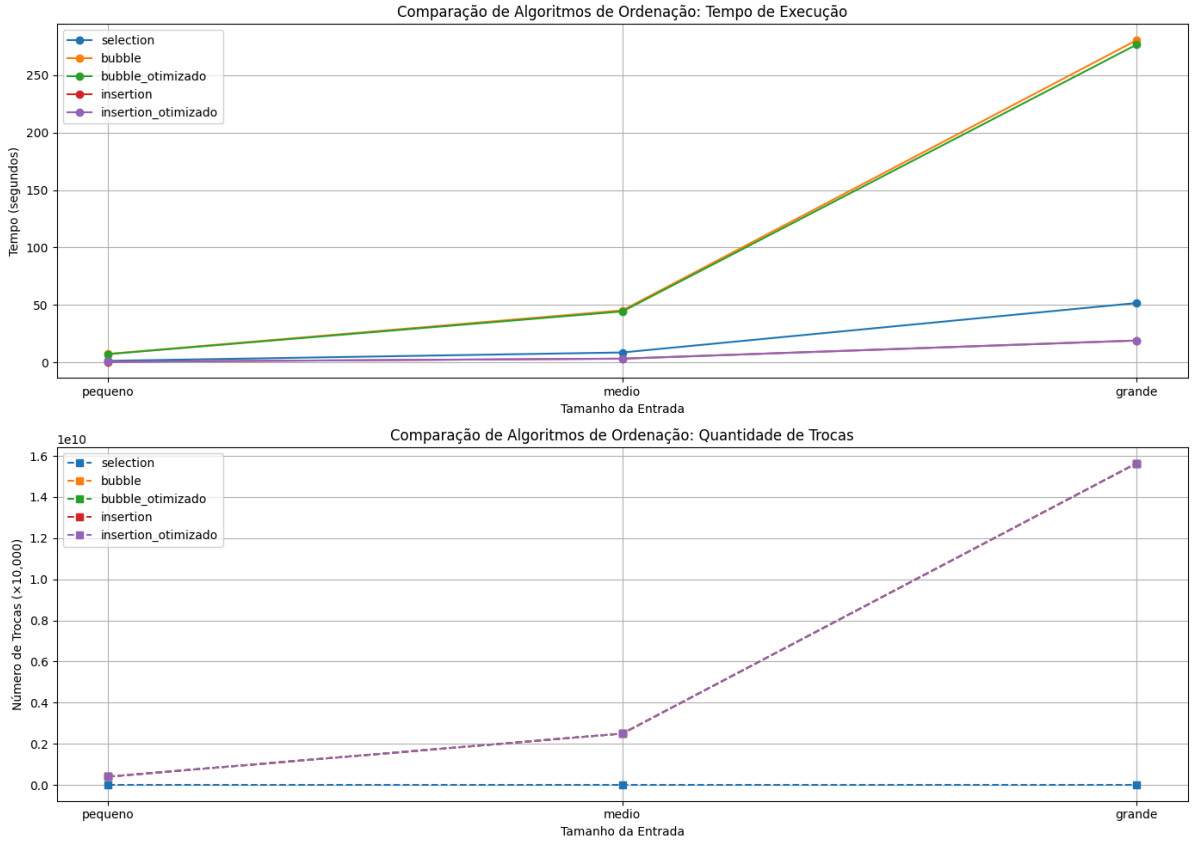
2. BubbleSort

Apresenta um alto número de trocas (~50% do total de elementos) justificando seu péssimo desempenho em tempo.

3. InsertionSort

Número de trocas similar ao BubbleSort, porém muito mais rápido, mostrando que trocas não são o único fator determinante

Visualização dos Resultados



Representação visual do tempo de execução / tamanho de entrada e números de trocas para cada um dos algoritmos.

Conclusão

Os resultados confirmam a teoria de complexidade de algoritmos:

- *Insertion Sort* foi o mais eficiente para os dados testados
- *Bubble Sort* apresentou o pior desempenho, mesmo na versão otimizada
- O crescimento do tempo foi aproximadamente quadrático, conforme esperado para algoritmos $O(n^2)$
- Os algoritmos de busca tem suas diferenças de performance, mas ao executar elas são minúsculas relativamente (milésimos de segundos)