UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Algoritmos e Estruturas de Dados I

Prova 2 - Trabalho Prático

Prof. Rodrigo Hübner

João Gabriel Wolf

Campo Mourão - PR

Junho de 2025

1. Introdução

Esse trabalho prático tem como objetivo implementar, comparar e analisar o desempenho dos algoritmos de ordenação SelectionSort (padrão e otimizado), BubbleSort (padrão e otimizado), InsertionSort e dos algoritmos de busca sequencial e binária, utilizando dados armazenados em arquivos binários. O guia de compilação e execução se encontra no arquivo "*README*". A linguagem escolhida para a execução do trabalho foi C.

2. Geração de Dados

Os dados gerados são valores inteiros aleatoriamente gerados do intervalo [0, 1.000.000[a partir do uso da função rand(), e inclui valores repetidos. Os tamanhos de arquivos escolhidos foram:

- 18.000 valores para arquivos pequenos (~1 segundo);
- 90.000 para arquivos médios (~30 segundos);
- 210.000 para arquivos grandes (~3 minutos).

O método de escolha dos valores foi empírico, baseado em testes de aproximação com resultados de ordenação bubble sort não otimizada com o hardware do aluno. É esperado que computadores com hardware diferentes tenham resultados diferentes.

Após o uso do arquivo *"gerador_dados.c"*, serão gerados os arquivos *"pequeno.bin"*, *"medio.bin"* e *"grande.bin"* localizados dentro do diretório *"dados"*. Esses arquivos serão utilizados por códigos de ordenação.

3. Ordenação de Arquivos

Todos os algoritmos de ordenação se localizam no arquivo *"ordenacao.c"*, dentro do diretório *"codigo"*. Para fatores de pesquisa, serão contabilizadas todas as trocas de valores de vetores (*swaps*) e comparações entre valores de vetores (*comparisons*), além do tempo necessário para cada ordenação de arquivos pequenos, médios e grandes.

3.1. Bubble Sort e Optimized Bubble Sort

O método de ordenação Bubble Sort possui complexidade *O(n^2)* em pior e médio caso, e complexidade *O(n)* em melhor caso. Se baseia na comparação e troca (*swap*) de elementos adjacentes, com iterações suficientes para que o vetor inteiro esteja ordenado. Na sua ordenação otimizada, possui uma verificação de trocas (*swaps*), que caso seja falsa, significa que o arquivo já está ordenado e não necessita de mais verificações.

3.2. Selection Sort e Optimized Selection Sort

O método de ordenação Selection Sort possui complexidade *O(n^2)* em pior, médio e melhor caso. Se baseia em iterações de encontro do menor valor e trocas com o primeiro elemento não ordenado. O método de otimização escolhido foi o Bidirectional Selection Sort, que além de

possuir um valor mínimo (*min*) procurado ao longo da ordenação, também possui um valor máximo (*max*), o que diminui o tempo de ordenação significativamente.

3.3. Insertion Sort

O método de ordenação Insertion Sort possui complexidade *O(n^2)* em pior e médio caso, e complexidade *O(n)* em melhor caso. Se baseia em iterações que dividem os valores em dois grupos: ordenados e não ordenados. O primeiro valor é assumido como ordenado, e gradualmente os valores não ordenados são inseridos no grupo ordenado.

3.4. Resultados de Ordenação

3.4.1. Arquivos Pequenos (18.000 valores inteiros)

Método de Ordenação	Tempo	Comparações	Trocas (*swaps*)
Bubble Sort	1,250 s	161.991.000	80.682.759
Optimized Bubble Sort	1,257 s	161.987.840	80.682.759
Selection Sort	0,409 s	161.991.000	17.990
Optimized Selection Sort	0,255 s	162.018.000	17.992
Insertion Sort	0,248 s	80.700.754	80.682.759 *(deslocamentos)*

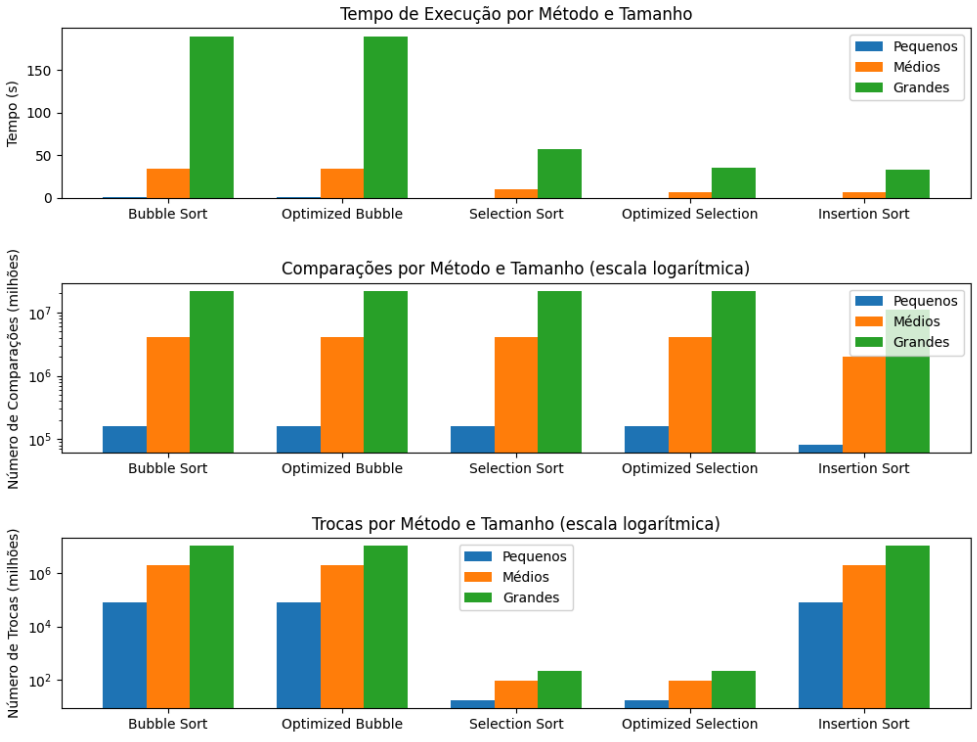
3.4.2. Arquivos Médios (90.000 valores inteiros)

Método de Ordenação	Tempo	Comparações	Trocas (*swaps*)
Bubble Sort	34,344 s	4.049.955.000	2.014.926.783
Optimized Bubble Sort	34,351 s	4.049.874.800	2.014.926.783
Selection Sort	10,473 s	4.049.955.000	89.991
Optimized Selection Sort	6,497 s	4.050.090.000	89.996
Insertion Sort	6,174 s	2.015.016.777	2.014.926.783 *(deslocamentos)*

3.4.3. Arquivos Grandes (210.000 valores inteiros)

Método de Ordenação	Tempo	Comparações	Trocas (*swaps*)
Bubble Sort	189,579 s	22.049.895.000	11.036.422.619
Optimized Bubble Sort	189,892 s	22.049.753.754	11.036.422.619
Selection Sort	56,717 s	22.049.895.000	209.987
Optimized Selection Sort	35,424 s	22.050.210.000	209.987
Insertion Sort	32,991 s	11.036.632.613	11.036.422.619 *(deslocamentos)*

3.4.4. Gráfico



4. Busca em Arquivos

Todos os algoritmos de busca se localizam no arquivo *"busca.c"*, dentro do diretório *"codigo"*. Para fatores de pesquisa, serão contabilizadas todas as comparações entre valores de vetores (*comparisons*), além do tempo necessário para 100.000 repetições de buscas de arquivos pequenos, médios e grandes. Cada busca tem um tempo extremamente curto, fazendo com que 100.000 repetições demonstrem resultados mais compreensíveis. O método para encontrar o alvo de busca é o maior valor possível+1, ou seja, um valor que não existe no vetor para medir a pior situação possível..

Todas as buscas são feitas no diretório *dados/ordenado*, e se assume que os arquivos buscados já estão ordenados.

4.1. Busca Sequencial (Linear Search)

O método de Busca Sequencial tem complexidade *O(N)* em pior e médio caso, e complexidade *O(1)* em melhor caso. itera sobre todos os elementos do vetor e os compara, não precisando ter arquivos ordenados para leitura. Ele é mais lento, ineficiente e mais simples comparado a Busca Binária.

4.2. Busca Binária (Binary Search)

O método de Busca Binária tem complexidade *O(log N)* em pior e médio caso, e complexidade *O(1)* em melhor caso. Necessita de arquivos ordenados para leitura, e se baseia em diversas iterações com a divisão do vetor com metades (*center*) e comparações entre maiores (*right*) e menores (*left*) valores das metades. Caso metade seja maior ou menor que o valor buscado, se descarta metade dos valores de busca sucessivamente até que se encontre o valor desejado.

4.3. Resultados de Busca

4.3.1. Arquivos Pequenos (18.000 valores inteiros)

Método de Busca	Tempo (1000 repetições)	Tempo (1 busca)	Comparações (1000 repetições)	Comparações (1 busca)
Busca Sequencial	4,724 s	47,240 μs	1.800.000.000	18.000
Busca Binária	0,008 s	0,08 μs	1.500.000	15

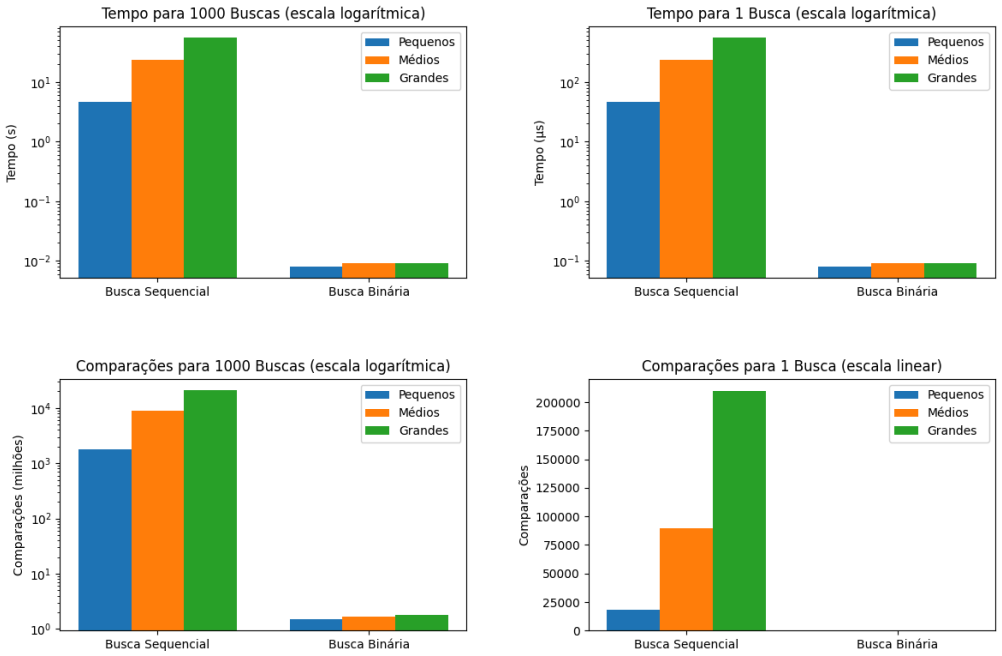
4.3.2. Arquivos Médios (90.000 valores inteiros)

Método de Busca	Tempo (1000 repetições)	Tempo (1 busca)	Comparações (1000 repetições)	Comparações (1 busca)
Busca Sequencial	23,687 s	236,870 µs	9.000.000.000	90.000
Busca Binária	0,009 s	0,09 µs	1.700.000	17

4.3.3. Arquivos Grandes (210.000 valores inteiros)

Método de Busca	Tempo (1000 repetições)	Tempo (1 busca)	Comparações (1000 repetições)	Comparações (1 busca)
Busca Sequencial	55,675 s	556,750 µs	21.000.000.000	210.000
Busca Binária	0,009 s	0,09 µs	1.800.000	18

4.3.4. Gráfico



5. Conclusão

5.1. Ordenação

Ao comparar os resultados da ordenação com Bubble Sort e Optimized Bubble Sort (*OBS*), é perceptível que embora o OBS deveria ser, em geral, mais eficiente e com menos tempo necessário para ordenação quando comparado a Bubble Sort, ele teve resultados minimamente piores que até mesmo sua versão não otimizada. Isso ocorre porque o OBS apenas contribui positivamente para dados que possuem certo grau de ordenação. Caso a geração dos dados seja semi-aleatória, o OBS contribui negativamente, com mais verificações do que o Bubble Sort padrão.

A diferença entre Optimized Selection Sort (*Bidirectional*) e Selection Sort foi positiva, com a otimização tendo uma redução de ~40% no tempo necessário para ordenação, e aumento extremamente mínimo entre comparações (~0.001% maior), demonstrando que utilizar tanto *min* quanto *max* foi produtivo para a ordenação.

O Insertion Sort, comparado a todos os outros métodos de ordenação pesquisados (Bubble Sort, Optimized Bubble Sort, Selection Sort, Optimized Selection Sort) foi o algoritmo que apresentou menor tempo de ordenação, mas continua ineficiente para ordenações de grande escala, demorando ~30 segundos para uma ordenação de ~200.000 valores inteiros no hardware utilizado para testes. O motivo de Insertion Sort ter sido o método de ordenação mais rápido testado foi por não utilizar trocas (*swaps*) e sim deslocamentos, sendo mais eficiente no gerenciamento de memória.

5.2. Buscas

Os métodos de busca analisados possuem diferenças extremas e limitações específicas. A Busca Sequencial (*Linear Search*) possui tempo diretamente proporcional à quantidade de valores lidos no pior caso, sendo muito ineficiente em buscas grandes. Comparativamente, a Busca Binária (*Binary Search*), tem uma quantidade extremamente inferior de comparações, fazendo buscas em tempos menores que 0,01 segundos. Entretanto, a Busca Binária necessita de dados previamente ordenados, o que significa que a Busca Sequencial apenas é viável em dados não ordenados e em pequenas quantidades.