**Relatório Final: Análise Comparativa de Algoritmos de Ordenação**

**Grupo:** Cauã Rocha, Davi Bastos, Polyana Raquel

**Disciplina:**  Projeto e Análise de Algoritmos

**1. Introdução**

Este relatório apresenta uma análise de desempenho de seis algoritmos de ordenação: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort e Heap Sort. O objetivo foi implementar os algoritmos em Java, executá-los com dados de complexidade variável e analisar os resultados para determinar a ordem de complexidade e a eficiência de cada método. Os dados de entrada foram extraídos de redes de estradas dos EUA, conforme proposto na atividade.

**2. Metodologia**

Os algoritmos foram implementados em Java e executados a partir de um ambiente de linha de comando. Para a coleta de dados, foram utilizados os arquivos de redes de estradas de New York (NY), San Francisco Bay Area (BAY) e Colorado (COL). A partir desses arquivos, foram extraídos conjuntos de dados com tamanhos variando de 100.000 a 700.000 elementos. Cada algoritmo foi executado com um conjunto de dados de tamanho específico, e seu tempo de execução em milissegundos foi cronometrado e registrado.

**3. Resultados Experimentais**

**3.1. Tabela de Tempos de Execução**

A tabela abaixo mostra os tempos de execução medidos durante os testes. As células vazias indicam combinações que não foram executadas.

Aplicativo, Tabela, Excel

O conteúdo gerado por IA pode estar incorreto.

**3.2. Gráficos de Desempenho**

Os gráficos a seguir ilustram visualmente os dados coletados.

Este gráfico monstra a diferença de desempenho entre os algoritmos. A escala de tempo necessária para os algoritmos quadráticos torna os algoritmos log-lineares quase imperceptíveis, destacando a sua superioridade.

Interface gráfica do usuário, Gráfico, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

**Gráfico 2: Comparativo dos Algoritmos Eficientes (O(n log n))** Focando apenas nos algoritmos rápidos, este gráfico permite uma comparação mais detalhada do desempenho do Merge Sort, Quick Sort e Heap Sort. Fica claro que, mesmo para grandes volumes de dados, o tempo de execução é mais rápido.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

**4. Análise de Complexidade**

Os resultados experimentais confirmam a teoria de complexidade de algoritmos:

* **Bubble Sort, Insertion Sort, Selection:** O Gráfico 1 mostra claramente que o tempo de execução para estes algoritmos cresce de forma exponencial. Ao dobrar a entrada, o tempo tende a quadruplicar. Por exemplo, o Selection Sort saltou de ~8 segundos para 220.000 elementos para ~42 segundos para 460.000 elementos (aproximadamente o dobro da entrada), um aumento de mais de 5 vezes. Isso os torna impraticáveis para grandes conjuntos de dados.
* **Merge Sort, Quick Sort, Heap Sort:** O Gráfico 2 revela que estes algoritmos são ordens de magnitude mais eficientes. O tempo de execução cresce de forma muito lenta e controlada. Mesmo com 700.000 elementos, os tempos de execução permaneceram abaixo de 100 milissegundos, provando sua escalabilidade e adequação para aplicações do mundo real.

**5. Respostas Críticas**

**1. Qual algoritmo apresenta melhor desempenho com dados ordenados de forma crescente?** O Insertion Sort. Em um cenário com dados já ordenados, ele atinge seu melhor caso, percorrendo a lista apenas uma vez com complexidade O(n), sendo o mais rápido de todos.

**2. Qual é o mais eficiente com dados ordenados de forma decrescente?** O Heap Sort e o Merge Sort. Ambos mantêm sua performance de O(n log n) independentemente da ordem inicial dos dados. O Quick Sort, em implementações simples, pode degradar para O(n²) neste cenário, tornando-se tão lento quanto o Bubble Sort.

**3. Qual algoritmo é mais estável em relação ao tempo de execução, independentemente da organização dos dados?** O Merge Sort é o mais estável. Seu tempo de execução é consistentemente O(n log n) no melhor, médio e pior caso, pois ele sempre divide o array da mesma forma. O Heap Sort também é muito estável com O(n log n) em todos os casos. O Quick Sort, apesar de muitas vezes ser o mais rápido na prática (caso médio), é o menos estável dos três, pois seu pior caso é O(n²).

**6. Observações e Desafios Durante o Desenvolvimento**

Durante a execução deste trabalho, enfrentamos desafios que foram cruciais para o aprendizado:

* **Configuração do Ambiente Java:** Um dos primeiros obstáculos foi um erro de era compilado com uma versão mais nova UnsupportedClassVersionError. Isso ocorreu devido à existência de múltiplas versões do Java instaladas no sistema, onde o código (JDK 17) e executado com uma mais antiga (Java 8). A solução envolveu a correção das variáveis de ambiente do sistema operacional para garantir que a mesma versão do JDK fosse usada tanto para compilação (javac) quanto para execução (java).
* **Permissões do Sistema Operacional:** O desafio mais persistente foi um erro de FileNotFoundException: (Acesso negado). Mesmo com o arquivo de dados presente na pasta correta, o programa não conseguia lê-lo. Tentativas de mover o projeto para pastas diferentes (Documentos) não surtiram efeito. O problema só foi resolvido ao executar o terminal com **privilégios de Administrador** e mover o projeto para uma pasta na raiz do sistema de arquivos (C:\), contornando as políticas de segurança do Windows que protegem as pastas de usuário.
* **Análise Prática vs. Teórica:** A execução dos algoritmos lentos (Bubble, Insertion, Selection) com grandes volumes de dados se mostrou inviável, com tempos de execução na casa de minutos. Isso nos forçou a adotar uma estratégia de teste mista, usando tamanhos de entrada menores para esses algoritmos, o que na prática reforçou a conclusão teórica sobre sua ineficiência.

**7. Conclusão**

Este trabalho demonstrou de forma prática e conclusiva a diferença de performance entre classes de algoritmos de ordenação. A análise dos tempos de execução e dos gráficos gerados valida a teoria da complexidade, provando que a escolha do algoritmo correto é fundamental para a eficiência de uma aplicação. Enquanto algoritmos O(n²) são simples, sua aplicação é limitada a pequenos conjuntos de dados. Para qualquer aplicação que lide com volumes de informação significativos, algoritmos O(n log n) como o Merge Sort, Quick Sort e Heap Sort são a escolha indispensável.