Segundo Projeto de ATP-2
(Algoritmos e Técnicas de Programação)

Discente: Luis Miguel Gomes Nascimento
Docente: Lucas Correia Ribas

Conteúdo

1 Objetivo

Este projeto tem como objetivo analisar e comparar o desempenho de diferentes algoritmos de ordenação (Selection, Insertion, Merge, Quick e Shell Sort) no contexto da classificação de clientes bom pagadores. Os algoritmos são aplicados para ordenar as distâncias calculadas entre os dados de referência e os dados de avaliação.

2 Detalhes de Implementação

Esta seção apresenta as etapas essenciais da implementação e suas funções no sistema.

2.1 Leitura de Dados

A leitura dos dados foi feita a partir do código-base fornecido pelo professor. Foram lidos os dados dos clientes de avaliação e seu perfil de bom pagador; além disso, também foram lidos os dados de referência usados para verificar a taxa de acerto na aprovação do crédito.

2.2 Algoritmos de Ordenação

Todos os algoritmos de ordenação usados tiveram pequenas modificações em relação aos vistos em sala de aula (RIBAS, 2025), sendo elas a organização da matriz p (pagadores). Isso foi necessário para que, quando uma distância mudasse de lugar, a informação de bom pagador daquele cliente de referência fosse alterada da mesma forma, garantindo que nenhuma informação fosse perdida ou modificada e que tudo permanecesse estável.

2.3 Cálculo da Matriz Distância

Para o cálculo da matriz de distâncias, foi utilizada a fórmula da distância euclidiana. Nesse processo, os dados de cada cliente em avaliação são subtraídos dos dados dos clientes de referência e, em seguida, cada diferença é elevada ao quadrado. Depois, soma-se todos esses valores e, por fim, calcula-se a raiz quadrada do resultado.

2.4 Funções de Alocação

No código, temos duas funções de alocação de memória, sendo elas matriz-d e matriz-p. Elas fazem referência à criação da matriz de distância (distância de cada cliente de referência em relação ao de avaliação) e à matriz de pagadores (onde ficam todos os índices de bom pagador de referência em relação a cada cliente de avaliação). O código para a alocação da matriz de distância pode ser visto na Figura 1 e o da matriz de pagadores na Figura 2.

Figura 1: Código de Alocação da Matriz de Distância (`matriz-d`)

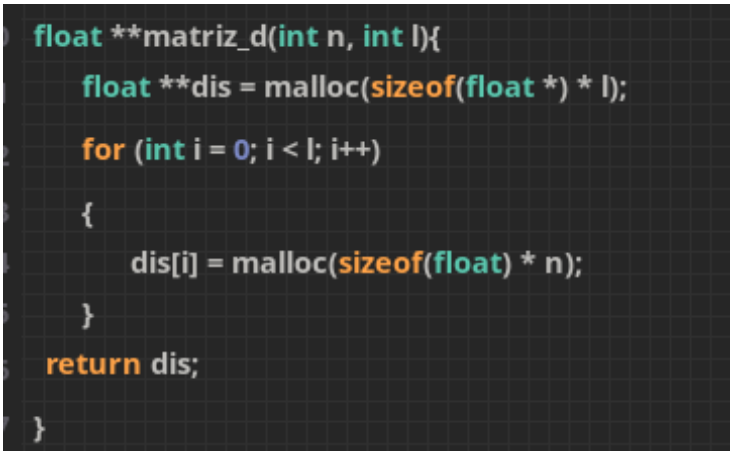
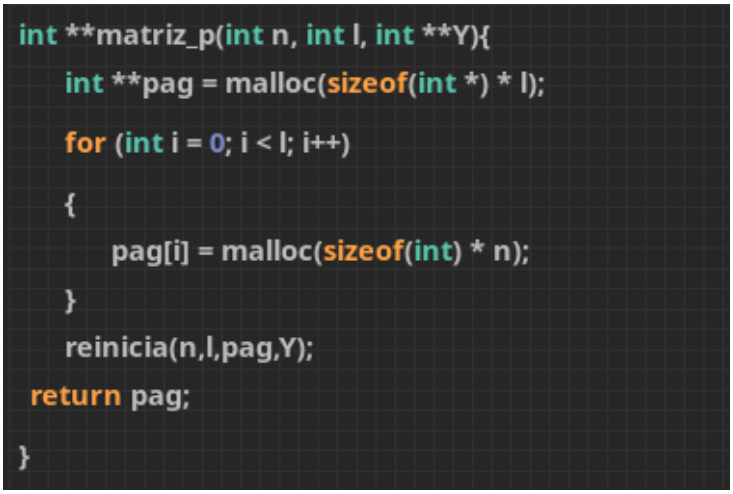


Figura 2: Código de Alocação da Matriz de Pagadores (`matriz-p`)



2.5 Medição do Tempo de Execução

Para a medição do tempo, foi utilizada uma função da biblioteca `time.h`, que reserva duas variáveis para armazenar o tempo de início e de fim. Em seguida, é feito um cálculo subtraindo o tempo de início do tempo de fim (fim – início), permitindo encontrar um valor em segundos. O código responsável por esta medição é apresentado na Figura 3.

Figura 3: Exemplo do Código usado para Medição de Tempo (`time.h`)

```
clock_t inicio, fim;
double tempo;

inicio = clock();
for(int k=0;k<rows_Xa;k++){
 selection(rows_X, dis[k], pag[k]);
}
fim = clock();
```

3 Sistema e Hardware

Ubuntu 24.04.3 LTS

Placa Mãe: GA-A320M-H
Processador: AMD Ryzen 3 3100 4-Core Processor — 3.59 GHz
RAM: 8,00 GB
Armazenamento: 447 GB SSD Kingston SA400S37 480G
Placa de Vídeo: NVIDIA GeForce GT 1030 (2 GB)

4 Dificuldades

Algumas dificuldades encontradas na construção do projeto foram: garantir que todos os algoritmos de ordenação estivessem estáveis (Insertion Sort e Merge Sort), encontrar a melhor forma de organizar os dados para permitir a ordenação e assegurar uma boa otimização do código nas chamadas de função e, por fim, garantir um baixo tempo de execução.

5 Tabelas de Resultados

Tabela 1: Desempenho do Algoritmo: Selection Sort

Referência (N)	**k=1**	**k=3**	**k=5**	**k=7**	**k=9**	**k=11**	**Tempo (s)**
100	74.50	80.00	82.50	84.50	85.00	83.00	0.000
1000	76.00	83.00	83.00	84.00	83.50	84.00	0.164
2000	77.00	85.00	85.50	84.00	84.50	85.50	0.624
5000	77.50	87.00	84.00	86.00	85.50	85.00	3.760
10000	79.00	85.50	87.00	86.50	87.00	86.50	14.638

Tabela 2: Desempenho do Algoritmo: Insertion Sort

Referência (N)	**k=1**	**k=3**	**k=5**	**k=7**	**k=9**	**k=11**	**Tempo (s)**
100	74.50	80.00	82.50	84.50	85.00	83.00	0.000
1000	76.00	83.00	83.00	84.00	83.50	84.00	0.174
2000	77.00	85.00	85.50	84.00	84.50	85.50	0.708
5000	77.50	87.00	84.00	86.00	85.50	85.00	4.244
10000	79.00	85.50	87.00	86.50	87.00	86.50	16.502

Tabela 3: Desempenho do Algoritmo: Merge Sort

Referência (N)	**k=1**	**k=3**	**k=5**	**k=7**	**k=9**	**k=11**	**Tempo (s)**
100	74.50	80.00	82.50	84.50	85.00	83.00	0.000
1000	76.00	83.00	83.00	84.00	83.50	84.00	0.030
2000	77.00	85.00	85.50	84.00	84.50	85.50	0.060
5000	77.50	87.00	84.00	86.00	85.50	85.00	0.170
10000	79.00	85.50	87.00	86.50	87.00	86.50	0.360

Tabela 4: Desempenho do Algoritmo: Quick Sort

Referência (N)	**k=1**	**k=3**	**k=5**	**k=7**	**k=9**	**k=11**	**Tempo (s)**
100	76.50	82.50	83.50	85.00	84.50	83.00	0.000
1000	80.00	83.00	83.00	84.00	83.50	84.00	0.040
2000	80.50	85.00	85.50	84.00	84.50	85.50	0.104
5000	80.00	86.00	84.00	86.00	85.50	85.00	0.450
10000	81.50	84.00	87.00	87.00	87.00	86.50	1.530

Tabela 5: Desempenho do Algoritmo: Shell Sort

Referência (N)	**k=1**	**k=3**	**k=5**	**k=7**	**k=9**	**k=11**	**Tempo (s)**
100	74.50	82.50	83.00	84.50	85.00	83.50	0.000
1000	78.50	83.00	83.00	84.00	83.50	84.00	0.030
2000	80.50	85.00	85.50	84.00	84.50	85.50	0.060
5000	81.00	85.00	83.50	86.00	85.50	85.00	0.172
10000	81.00	82.50	87.00	86.50	87.00	86.50	0.382

5.1 Redução do Vetor D e Análise dos Resultados

Para avaliar como o tamanho do vetor de distâncias impacta o desempenho dos algoritmos de ordenação e a taxa de acerto do classificador, foram realizados testes reduzindo o vetor D para diferentes valores de $s = \{100, 1000, 2000, 5000, 10000\}$.

Essa redução foi aplicada **antes** da etapa de ordenação, conforme solicitado, preservando apenas as primeiras s posições do vetor original. Após essa etapa, cada algoritmo foi executado normalmente, registrando-se o tempo de processamento e as taxas de acerto para cada valor de k.

A Tabela 6 apresenta os resultados obtidos considerando a redução do vetor D para os diferentes valores de s.

Tabela 6: Resultados dos Algoritmos com Variação no Tamanho do Vetor D

Tamanho N	**Algoritmo**	**Tempo (s)**	**Taxa de Acerto Média (%)**
100	Merge Sort	0.000	79.30
1000	Merge Sort	0.030	82.30
2000	Merge Sort	0.060	83.20
5000	Merge Sort	0.170	84.30
10000	Merge Sort	0.360	85.30

Observa-se que, conforme o tamanho de s aumenta, os tempos de execução crescem de forma proporcional ao custo computacional previsto pelos algoritmos. As taxas de acerto também tendem a melhorar conforme o conjunto de referência se aproxima do tamanho total original, o que era esperado devido à maior disponibilidade de informações na classificação.

6 Conclusão

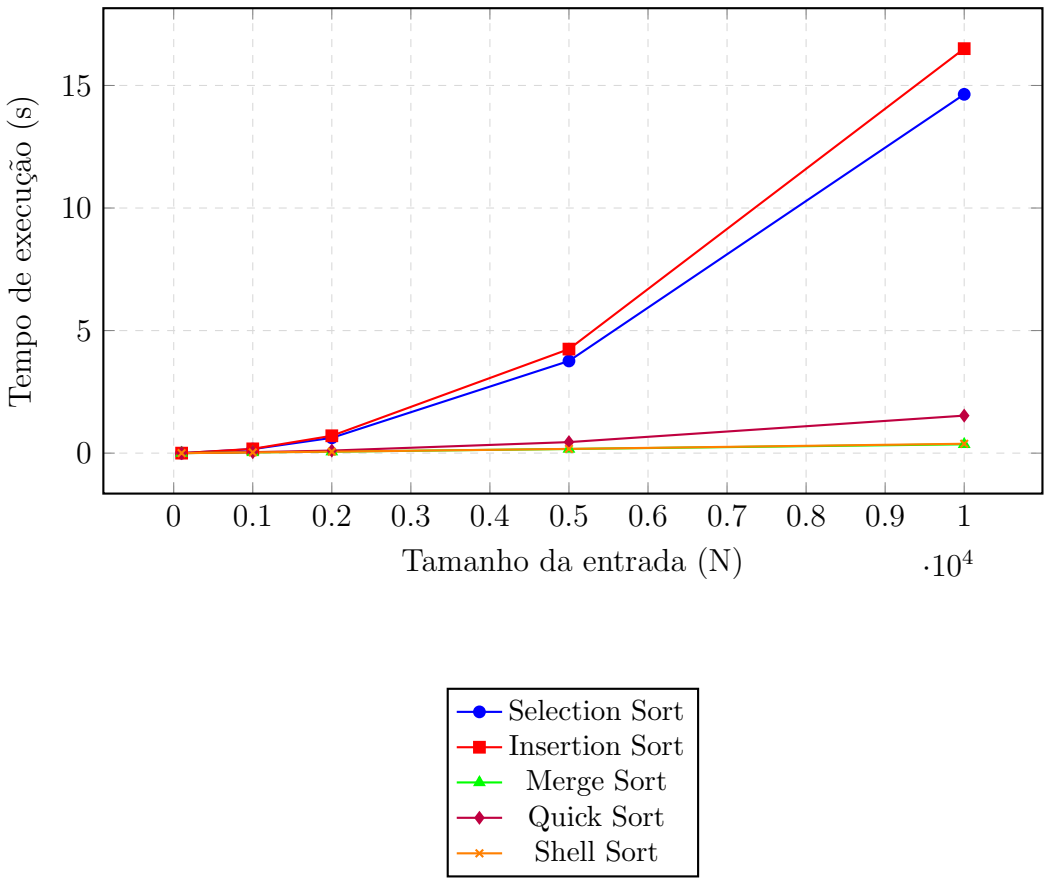
Após todos os dados serem coletados, organizados e verificados, é de extrema importância comparar os tipos de sorts e analisar, a partir dos resultados, as propriedades de cada algoritmo de ordenação.

Por exemplo, no gráfico a seguir, verifica-se a constância nos resultados do Merge Sort e do Shell Sort. O Merge Sort apresenta um tempo de execução totalmente previsível, pois sua complexidade é sempre O(n log(n)), independentemente da distribuição dos dados. Já o Shell Sort é apenas parcialmente previsível, pois seu desempenho pode variar dependendo da distribuição dos dados, fazendo com que o tempo de execução sofra pequenas variações; neste caso, não houve nenhuma alteração significativa.

Em contrapartida, o Selection Sort e o Insertion Sort apresentam um desempenho inferior de modo geral. A causa desse problema vem da grande quantidade de comparações e operações realizadas por esses algoritmos, o que ocasiona um aumento drástico no tempo de execução conforme o tamanho da amostra cresce.

Por fim, o Quick Sort apresentou um tempo de execução razoável, o que pode ser considerado esperado devido à sua complexidade: no pior caso, próximo de $O(n^2)$, e no melhor caso (e também no caso médio), O(n log(n)). Entretanto, caso os dados estivessem organizados de outra forma, poderiam ocorrer alterações significativas em seu desempenho, especialmente dependendo da escolha do pivô.

Figura 4: Comparação de tempo de execução entre algoritmos



A análise do gráfico mostra com clareza que os algoritmos Merge Sort, Shell Sort e Quick Sort são mais adequados para conjuntos maiores de dados e apresentam um desempenho significativamente superior aos algoritmos quadráticos.

Por fim, algumas possíveis melhorias poderiam aumentar o desempenho de determinados algoritmos. No caso do Shell Sort, a escolha de uma sequência de gaps mais eficiente como a sequência proposta por Knuth (1973), pode tornar a ordenação até 20% mais rápida (FEOFILOFF, 2009). O Quick Sort também poderia ser otimizado com o uso de dois pivôs, dividindo o vetor em mais partes e reduzindo a quantidade de comparações, o que tende a melhorar seu desempenho. Já o Insertion Sort e o Selection Sort não apresentam melhorias significativas capazes de alterar de forma relevante sua complexidade ou eficiência.

7 Referências

RIBAS, Lucas. *aula10.pdf (Algoritmos de Ordenação)*. 2025.
FEOFILOFF, Paulo. *Algoritmos em Linguagem C*. 2009.