UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO

MARCOS MIKAEL LIMA VIDAL

ANÁLISE DE COMPLEXIDADE DE ALGORITMOS UTILIZANDO MÉTODOS EMPÍRICOS/EXPERIMENTAIS

PAU DOS FERROS
2025

MARCOS MIKAEL LIMA VIDAL

ANÁLISE DE COMPLEXIDADE DE ALGORITMOS UTILIZANDO MÉTODOS EMPÍRICOS/EXPERIMENTAIS

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Tecnologia da Informação.

Orientador: Kennedy Reurison Lopes, Prof. Dr.

PAU DOS FERROS

2025

V648a Vidal, Marcos Mikael Lima .
Análise de Complexidade de Algoritmos Utilizando Métodos Empíricos/Experimentais / Marcos Mikael Lima Vidal. - 2025.
55 f. : il.

Orientador: Kennedy Reurison Lopes.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2025.

1. Análise de algoritmos. 2. Algoritmos de ordenação. 3. Gradiente descendente. 4. Aprendizado de máquina. I. Lopes, Kennedy Reurison , orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

MARCOS MIKAEL LIMA VIDAL

ANÁLISE DE COMPLEXIDADE DE ALGORITMOS UTILIZANDO MÉTODOS EMPÍRICOS/EXPERIMENTAIS

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Tecnologia da Informação.

Defendida em: 15 / 12 / 2025

BANCA EXAMINADORA

Kennedy Reurison Lopes, Prof. Dr. (UFERSA)
Presidente

Lucas Abrantes Sarmento, Prof. Bel. (UFERSA)
Membro Examinador

George Felipe Fernandes Vieira, Prof.
Bel. (UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço profundamente à minha família pelo apoio incondicional. Em especial, à minha mãe, Micaelli, e ao meu pai, Antônio, por acreditarem em mim e me sustentarem nos momentos mais desafiadores.

Expresso também minha gratidão aos meus amigos Andrey, Afonso e Enzo, pela companhia constante e por tornarem meus dias mais leves.

Ao meu orientador e professor Kennedy, agradeço pela orientação paciente, pelas valiosas contribuições e por ter me guiado com sabedoria ao longo de todo o desenvolvimento deste trabalho.

À minha companheira, Mariana, que está sempre comigo e me incentiva a seguir em frente, me fazendo ver o melhor em mim mesmo, meu muito obrigado.

Ao meu filho Mael, que é a minha maior inspiração e motivação para buscar sempre o melhor.

EPÍGRAFE

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."

(Ariano Suassuna)

RESUMO

A análise de complexidade de algoritmos é fundamental para o desenvolvimento de sistemas de alta performance, permitindo prever o consumo de recursos computacionais. No entanto, a análise assintótica teórica nem sempre reflete o desempenho prático devido a fatores como constantes ocultas, arquitetura de hardware e otimizações de compiladores. Este trabalho propõe uma metodologia automatizada para a análise empírica da complexidade temporal de algoritmos de ordenação, tratando-os como caixas-pretas. O sistema desenvolvido avalia oito algoritmos clássicos (Bubble, Insertion, Merge, Quick, Heap, Counting, Radix e Bucket Sort) em cenários com dados aleatórios, ordenados e reversos. A metodologia reside na aplicação do otimizador Adam (Gradiente Descendente Adaptativo) para ajustar modelos matemáticos ($O(n)$, $O(n^2)$ e $O(n \log n)$) aos tempos de execução coletados, utilizando o Erro Quadrático Médio (RMSE) como critério de seleção. Os resultados experimentais confirmaram a hierarquia teórica esperada, validaram casos especiais como o desempenho linear do Insertion Sort em vetores ordenados e evidenciaram a alta instabilidade do Bubble Sort. O estudo conclui que a técnica de ajuste iterativo de curvas é eficaz para classificar algoritmos empiricamente, oferecendo uma ferramenta prática para tomada de decisão em engenharia de software.

Palavras-chave: análise de algoritmos; algoritmos de ordenação; gradiente descendente; aprendizado de máquina

ABSTRACT

Algorithm complexity analysis is fundamental for developing high-performance systems, enabling the prediction of computational resource consumption. However, theoretical asymptotic analysis does not always reflect practical performance due to factors such as hidden constants, hardware architecture, and compiler optimizations. This work proposes an automated methodology for the empirical analysis of the time complexity of sorting algorithms, treating them as black boxes. The developed system evaluates eight classic algorithms (Bubble, Insertion, Merge, Quick, Heap, Counting, Radix, and Bucket Sort) across random, sorted, and reverse data scenarios. The methodological innovation lies in applying the Adam optimizer (Adaptive Gradient Descent) to fit mathematical models ($O(n)$, $O(n^2)$, and $O(n \log n)$) to observed execution times, using the Root Mean Square Error (RMSE) as the selection criterion. Experimental results confirmed the expected theoretical hierarchy, validated special cases such as the linear performance of Insertion Sort on sorted vectors, and highlighted the high instability of Bubble Sort. The study concludes that the iterative curve-fitting technique is effective for empirically classifying algorithms, providing a practical tool for decision-making in software engineering.

Keywords: algorithm analysis; sorting algorithms; gradient descent; machine learning

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 Contextualização

Informalmente, um algoritmo é qualquer procedimento computacional bem definido que recebe um valor (ou conjunto de valores) como entrada e produz um valor (ou conjunto de valores) como saída. Portanto, um algoritmo é uma sequência de etapas computacionais que transformam a entrada na saída, com uma sequência finita de instruções, mas que não necessariamente possui um tempo finito para sua execução (Cormen *et al.*, 2009). A execução dessas instruções, contudo, consome recursos limitados, como tempo de processamento e restrições de hardware. Dessa forma, Cormen *et al.* (2009) enfatiza a complexidade como um meio de medir e analisar a eficiência de um algoritmo em termos de tempo de execução e memória utilizada. De acordo com Arora e Barak (2009), a complexidade mede a dificuldade de resolver problemas computacionais e classifica problemas com base nos recursos necessários, como o tempo e o espaço, aos quais são recursos limitados. Podemos, desta forma, definir a complexidade como sendo uma métrica para analisar e classificar a quantidade de recursos necessários para resolver um determinado problema. Autores como Hogan (2011) e Goldberg (2019) reforçam que a interpretação adequada dessas métricas conecta fundamentos matemáticos com aplicações experimentais. Pesquisas recentes mostram que algoritmos eficientes sustentam aplicações em otimização industrial, gêmeos digitais automotivos e sistemas de controle inteligente (Alridha; Alsharify; Al-Khafaji, 2024; Silva; Porto; Kalili, 2024; Zonatto, 2018; Blondel; Tsitsiklis, 2000). A importância da análise de complexidade: analisar um algoritmo significa prever quais recursos o algoritmo necessita, como dito anteriormente. Eventualmente, recursos como memória, largura de banda de comunicação ou hardware de computador são as principais preocupações (Tanenbaum; Steen, 2017), porém o que mais frequentemente buscamos medir é o tempo de execução. Em geral, pela análise de vários algoritmos candidatos a um problema, pode-se identificar facilmente um que seja o mais eficiente. De acordo com Flajolet e Sedgewick (1992), existem vários motivos que levam a realizar tais análises, sendo estes os seguintes:

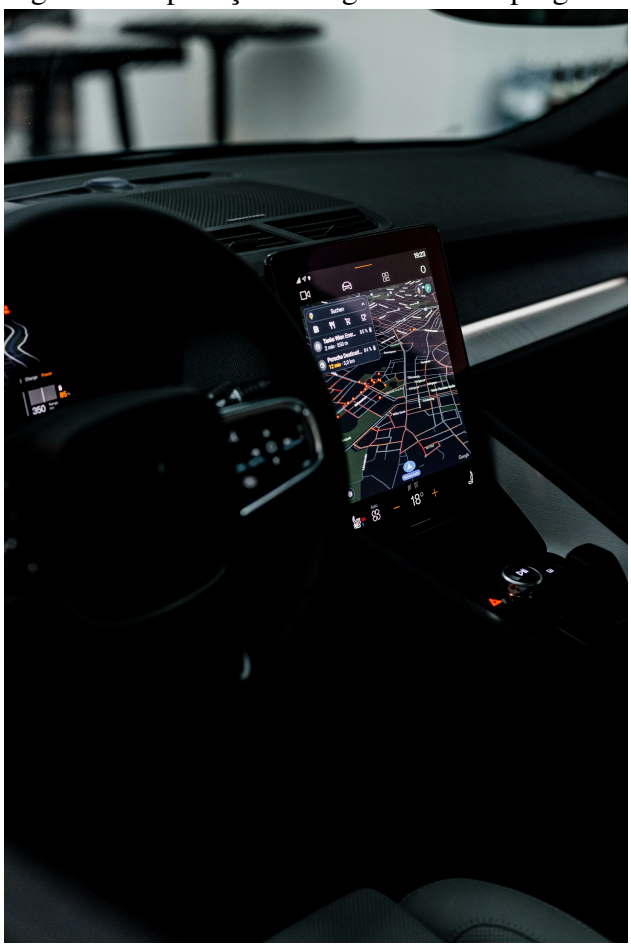
O uso pretendido do algoritmo, desenvolvido para otimizar processos e melhorar a sua eficiência em diferentes áreas ou aplicações como:

Programação automotiva: onde algoritmos podem ser usados para monitorar e controlar sistemas críticos em veículos, como (Silva; Porto; Kalili, 2024):

Gerenciamento do motor: Para ajustar dinamicamente a injeção de combustível e o tempo de

ignição com base em condições de operação. **Sistemas de assistência ao motorista**: Processar dados de sensores para evitar colisões e manter o veículo na faixa. **Gestão de baterias em veículos elétricos**: Garantir o uso eficiente da bateria, monitorando níveis de carga, temperatura e consumo energético.

Figura 1 – Aplicação de algoritmos em programação automotiva.



Fonte: Silva, Porto e Kalili (2024).

Indústria: No ambiente industrial, o algoritmo pode ser aplicado em sistemas de automação e controle, como por exemplo (Alridha; Alsharify; Al-Khafaji, 2024):

Manutenção preditiva: Analisar dados de sensores para prever falhas em máquinas e reduzir o tempo de inatividade.

Otimização de processos: Ajustar parâmetros de produção para minimizar desperdícios e maximizar a produtividade. **Controle de robôs industriais**: Coordenar movimentos e operações de robôs em linhas de montagem.

Figura 2 – Algoritmos no ambiente industrial: controladores numéricos coordenam robôs colaborativos, otimizam trajetórias e reduzem o tempo de ciclo na linha de produção.

Fonte: Zonatto (2018).

Motores de geração de energia: Em motores utilizado para geração, algoritmos podem ser usados para (Zonatto, 2018):

Monitoramento e controle do desempenho: Ajustar a potência gerada com base na demanda.

Detecção de anomalias: Identificar oscilações ou falhas nos motores para evitar paradas não programadas. **Eficiência energética**: Otimizar o uso de combustível ou fontes renováveis para maximizar a geração de energia com menor impacto ambiental. Métrica para avaliar a qualidade de um algoritmo em detrimento de outro.

Tempo de execução (complexidade temporal): Medir quanto tempo o algoritmo leva para processar uma entrada e produzir uma saída, tendo preferência por algoritmos com menor tempo de execução (Ngu, 2022; Bimurat *et al.*, 2023; Blondel; Tsitsiklis, 2000).

A dificuldade de análise:

Loops: Os *loops* (*while*, *for*) são um dos principais fatores que afetam a complexidade. O número de iterações determina o crescimento da complexidade. É necessário realizar uma análise prática

para observar o número de loops e condições de parada, verificando se dependem de variáveis externas ou se há otimizações possíveis. **Execução de pilha**: Chamadas recursivas afetam o uso da pilha de execução e podem ser analisadas pela complexidade recursiva. A profundidade de recursão e o número de chamadas determinam a complexidade. Um exemplo clássico é o Fibonacci. Sem otimização tem complexidade $O(2^n)$ por gerar muitas chamadas redundantes, e com memoização, reduz para $O(n)$. **Estruturas de seleção**: Estruturas como if, else if e switch influenciam o comportamento condicional de um algoritmo, pois a complexidade também é definida pelo número de condições analisadas. Por exemplo, estruturas aninhadas podem multiplicar o número de decisões a serem avaliadas, condições mal otimizadas podem aumentar o tempo de execução mesmo com complexidade assintótica constante. Além disso, a avaliação de um algoritmo deve considerar sua precisão, exatidão e estabilidade, especialmente em problemas sensíveis a erros acumulados. Precisão diz respeito à consistência dos resultados, enquanto exatidão avalia a proximidade da solução encontrada com o resultado ideal. A estabilidade reflete a capacidade do algoritmo lidar com pequenas variações nos dados de entrada sem grandes impactos na saída, sendo crucial em cálculos numéricos ou problemas com erros acumulativos . A razão mais simples para analisar um algoritmo é descobrir suas características para avaliar sua adequação a várias aplicações ou compará-los com outros algoritmos para mesma aplicação. As características de interesse são, na maioria das vezes, recursos primários de tempo e espaço, especialmente o tempo. Simplificando, queremos saber quanto tempo uma implementação de um determinado algoritmo levará para ser executada em um determinado computador e quanto espaço ele exigirá. Em geral, procuramos manter a análise independente de implementações específicas, concentrando-nos em obter resultados para características essenciais do algoritmo que possam ser usadas para derivar.

1.2 Problema

Em um cenário onde a eficiência dos algoritmos é algo crucial para o desenvolvimento de sistemas de alta performance, compreender a complexidade real dos algoritmos torna-se essencial. Como apresentado anteriormente, saber que um algoritmo converge não o torna automaticamente um algoritmo útil. Desta forma, a análise teórica de complexidade nem sempre reflete o desempenho prático dos algoritmos em contextos reais. Por isso, o presente trabalho busca preencher essa lacuna ao explorar o papel dos testes empíricos na avaliação da complexidade de algoritmos, destacando como esses testes permitem obter uma visão mais

realista e detalhada do comportamento do algoritmo sob diversas condições conforme detalhado na seção de metodologia.

1.3 Objetivos

Neste trabalho estamos interessados em produzir um sistema capaz de realizar uma meta-análise de algoritmos sobre a perspectiva de sua complexidade de tempo. Para isso, estão listados os principais objetivos:
Implementar algoritmos de ordenação para avaliação de complexidades, esses algoritmos servirão os valores de entrada para o algoritmo proposto. Apresentar uma análise teórica da complexidade teórica dos algoritmos implementados; Construir um sistema para executar e comparar os algoritmos de forma automatizada; Preparar diferentes cenários de teste para avaliar o desempenho dos algoritmos; Desenvolver visualizações gráficas para os resultados e analisar o impacto do tamanho e da estrutura das entradas no desempenho dos algoritmos; Definir uma estratégia de como calcular empiricamente os tempos deste algoritmos, utilizando métodos aproximador de funções como métodos de regressão linear Validar os resultados com o esperado na literatura; Elaborar um relatório técnico consolidando as descobertas.

Para alcançar esses objetivos pretendemos resolver outros objetivos específicos:
Realizar os experimentos em um ambiente controlado e confiável. Desenvolver a identificação de equações para comparar com as complexidades propostas; Implementar uma metodologia para identificar e ajustar equações de desempenho experimental, comparando-as com as complexidades propostas teoricamente; Garantir a repetibilidade dos experimentos, documentando detalhadamente as condições de execução, como o hardware utilizado, a linguagem de programação e as ferramentas de medição empregadas; Explorar a estabilidade dos algoritmos de ordenação, avaliando como eles lidam com elementos duplicados nos conjuntos de entrada; Automatizar o processo de coleta e organização dos dados obtidos durante os experimentos, facilitando a análise posterior; Desenvolver mecanismos para detectar e explicar discrepâncias significativas entre os resultados empíricos e as complexidades teóricas; Validar as implementações dos algoritmos com bibliotecas reconhecidas, garantindo que os resultados não sejam influenciados por erros de codificação. Este trabalho tem como objetivo principal analisar a complexidade temporal de algoritmos através de testes empíricos, permitindo uma avaliação detalhada de seu comportamento prático em condições variadas. Além disso, busca validar as análises teóricas com resultados experimentais, identificando possíveis discrepâncias que possam

ocorrer em cenários de execução reais. Outro objetivo central é investigar como diferentes tipos de entrada influenciam o desempenho dos algoritmos, revelando padrões de comportamento que ajudem na escolha de algoritmos mais eficientes para situações específicas. Finalmente, a pesquisa pretende identificar oportunidades de otimização, propondo melhorias que maximizem a eficiência dos algoritmos avaliados.

1.4 Justificativa

Visto que a análise de complexidade não é algo trivial matematicamente, este trabalho se justifica em propor um método automático de avaliação de algoritmos. Neste trabalho uma metodologia é apresentada para calcular automaticamente a complexidade de algoritmos de ordenação. Desta forma, a metodologia pretende comparar algoritmos e classificá-los de acordo com sua complexidade sem necessitar de uma análise matemática profunda. Como exemplo, considere a análise da complexidade do algoritmo merge-sort: para medir sua complexidade faz-se necessário compreender o funcionamento do algoritmo para estimar o tempo computacional. Neste algoritmo de ordenação os dados são divididos recursivamente até que apenas um único número seja produzido. Sabendo disso, esta subdivisão ocorre em complexidade logarítmica. Entretanto, é necessário juntar novamente todos os elementos e, como temos n elementos, a complexidade é n vezes o logaritmo dos dados. Observe que essa análise foi totalmente dependente do conhecimento do funcionamento do algoritmo. Este trabalho desenvolve a mesma complexidade $O(n \log n)$ considerando o algoritmo como uma caixa preta. Apenas os dados de entrada são os argumentos da avaliação. Na metodologia proposta, o algoritmo de ordenação passa a ser uma entrada do meta-algoritmo que tem como objetivo descobrir sua complexidade. A análise de complexidade de algoritmos é fundamental para a ciência da computação, pois permite prever o comportamento de soluções computacionais em diferentes cenários. Contudo, essa análise, em sua forma teórica, frequentemente exige um entendimento aprofundado tanto do funcionamento interno do algoritmo quanto de conceitos matemáticos avançados, o que pode limitar sua acessibilidade. Essa barreira é especialmente relevante em contextos como a formação de novos profissionais, onde a compreensão prática muitas vezes precede o domínio teórico, e na indústria, onde decisões rápidas e práticas são cruciais. A pesquisa desenvolvida neste trabalho propõe uma metodologia automatizada que possibilita a avaliação da complexidade de algoritmos sem depender do conhecimento detalhado de sua implementação. Trata-se de uma abordagem que abstrai o funcionamento interno dos algoritmos, analisando-os como "caixas-pretas"e utilizando

os dados de entrada e saída como base para a medição de desempenho.

2 TRABALHOS RELACIONADOS

Este capítulo descreve estudos e abordagens que fundamentam e dialogam com a proposta desenvolvida, destacando metodologias semelhantes e diferenciais relevantes para o trabalho.

2.1 Aplicações que exigem algoritmos eficientes

Alridha, Alsharify e Al-Khafaji (2024) organizaram um panorama de aplicações de otimização que dependem de algoritmos com boa relação custo-benefício, desde engenharia industrial até aprendizagem de máquina. O estudo destaca que técnicas de busca e ajuste em problemas reais requerem execução previsível, especialmente quando são parte de laços de decisão fechados.

No setor automotivo, Silva, Porto e Kalili (2024) demonstram como gêmeos digitais precisam processar fluxos de sensores (velocidade, temperatura, consumo) em tempo real para apoiar manutenção preditiva. Esses modelos virtualizados só permanecem úteis se as rotinas de simulação e ajuste forem ágeis o suficiente para acompanhar dados telemétricos contínuos. No mesmo domínio, Zonatto (2018) apresentou um controlador fuzzy para veículos híbridos que prioriza lógica baseada em regras justamente porque oferece baixo custo computacional e respostas em tempo real para o gerenciamento energético. Os três trabalhos reforçam que observar a complexidade prática não se restringe a benchmarks acadêmicos: ela define a viabilidade de sistemas ciberfísicos e de controle embarcado.

2.2 Fundamentos teóricos de complexidade

Na camada conceitual, Hogan (2011) detalha a formalização de classes de complexidade, introduzindo definição rigorosa para a notação Big-O e para a distinção entre problemas polinomiais e NP-completos. Goldberg (2019) complementa ao mostrar como novas linhas de pesquisa em complexidade teórica motivam heurísticas e algoritmos aproximados em cenários aplicados.

As notas de Dimensional Complexity (Ngu, 2022) aprofundam o papel das notações assintóticas ao relacionar dimensões de espaço e esforço computacional, enquanto o estudo da Journal AITU (Bimurat *et al.*, 2023) compara *Big-Oh* e *small-oh* e evidência, por meio de tabelas e figuras, como os limites superior e inferior ajudam a selecionar algoritmos escaláveis. Já Samsudin *et al.* (2020) aplicam esse ferramental a um caso concreto, avaliando a complexidade do algoritmo Euler Arithmetic e confrontando cenários de melhor e pior caso para os modelos

$O(1)$ e $O(n)$. Em conjunto, essas referências sustentam a necessidade de uma metodologia que traduza medições empíricas (curvas e resíduos ajustados) em afirmações alinhadas com a terminologia clássica de análise assintótica.

2.3 Complexidade em sistemas de controle e decisão

Blondel e Tsitsiklis (2000) compilaram resultados de complexidade para problemas de estabilidade, robustez e controle estocástico, mostrando que muitos testes são NP-difíceis ou até indecidíveis quando há incerteza paramétrica. O relatório enfatiza que análises simplificadas (por exemplo, assumir linearidade perfeita) mascaram custos reais de projeto.

Materiais acadêmicos recentes, como as notas da Tau University (Folaranmi; Ayepeku, 2023), retomam essa discussão ao propor modelos canônicos de custo ($O(n\log n)$, $O(n^2)$ e variantes logarítmicas) para medir algoritmos de ordenação, estruturação de dados e otimização. Essa literatura conecta teoria a práticas de desenvolvimento: compreender quando um algoritmo degrada para um regime quadrático ou permanece em $O(n\log n)$ define a confiabilidade de pipelines críticos.

2.4 Síntese para este trabalho

A revisão aponta dois vetores complementares. O primeiro é a pressão por eficiência em domínios aplicados (otimização, gêmeos digitais, controle híbrido), nos quais medições experimentais precisam justificar decisões de projeto. O segundo é o arcabouço formal que orienta como essas medições devem ser relatadas (notações assintóticas, classes de complexidade, fronteiras de decidibilidade).

O método proposto neste TCC responde a ambos ao capturar dados de execução dos algoritmos avaliados, ajustar curvas representativas ($O(n)$, $O(n\log n)$, $O(n^2)$) e interpretar os resíduos com suporte nas referências discutidas, garantindo coerência entre observação empírica e terminologia teórica.

3 METODOLOGIA

3.1 Fluxo de coleta e ajuste das curvas

O processo automatizado implementado neste trabalho segue três etapas encadeadas. A primeira é a coleta das amostras empíricas. O utilitário *BenchmarkRunner* percorre cada algoritmo configurado, gera instâncias no intervalo definido (por exemplo, de 1 000 a 50 000 elementos, em passos de 1 000) e, para cada tamanho n, executa um bloco de $k = 5$ repetições por meio da função *averageComplexity*. Cada repetição cria um vetor aleatório (`generateRandom`), mede o tempo com `clock()` e registra o valor em milissegundos. Os pares $\langle n, \bar{t}_n \rangle$ ficam armazenados em arquivos `algo_nome.txt`, o que permite reproduzir exatamente o conjunto de pontos coletado. Os experimentos foram executados em uma máquina virtual com Ubuntu, 1 núcleo de CPU e 6 GB de memória RAM, mantendo o mesmo ambiente para todas as execuções. A segunda etapa consiste em ajustar modelos de complexidade às séries geradas. Para cada algoritmo define-se um modelo-alvo (linear $a \cdot n + b$, quadrático $a \cdot n^2 + b \cdot n + c$ ou $n \log n$ com $a \cdot n \log n + b \cdot n + c$) e os coeficientes são estimados com gradiente descendente adaptativo do tipo Adam (Nobrega, 2014). As funções `firstDegree`, `secondDegree`, `logarithmic` e `nlogn_fit` calculam os gradientes analíticos de cada modelo, aplicando ($\beta_1 = 0{,}9,\ \beta_2 = 0{,}999,\ \varepsilon = 10^{-8}$), utilizam taxa de aprendizado 10^{-3} e interrompem a otimização quando o erro quadrático médio estabiliza (tolerância 10^{-7}) ou quando o limite de 100 mil iterações é atingido. Assim, cada classe de complexidade recebe um conjunto consistente de constantes $\{a, b, c\}$. Por fim, a terceira etapa avalia a aderência do ajuste. O utilitário calcula o erro quadrático (*SSE*) e o *RMSE* dos modelos e gera gráficos com `plotGraphGNU`, combinando os pontos observados com a curva ajustada. Os resultados são consolidados em `complexities.txt`, arquivo que informa para cada algoritmo o modelo escolhido, os coeficientes e o erro residual.

3.1.1 Distribuições de Tempo de Execução

Para algoritmos randomizados, analisar a distribuição completa dos tempos de execução (RTDs) é mais informativo do que apenas a média ou a mediana. As RTDs mostram a probabilidade de um algoritmo resolver um problema dentro de um determinado tempo. A comparação de RTDs pode revelar nuances no desempenho que métricas agregadas podem ocultar (Hoos, 2003).

3.1.2 Análise de Robustez

A análise de robustez mede a estabilidade do desempenho de um algoritmo em relação a variações nas configurações de seus parâmetros ou nas características do problema. Isso envolve a análise da variação do desempenho e sua correlação com os valores dos parâmetros (Hoos, 2003).

3.1.3 Considerações Finais

Ao realizar a análise empírica, é fundamental:

Reproducibilidade: Documentar detalhadamente as condições de execução (hardware, linguagem de programação, ferramentas de medição) para garantir que os experimentos possam ser replicados (Cruz, 2018). **Interpretação Cautelosa:** Os resultados empíricos devem ser interpretados com atenção e pensamentos crítico, questionando se são consistentes com a análise teórica, significativos, generalizáveis, reproduzíveis e verificáveis (How. . . , 2023).

3.1.4 Métricas Descritivas Fundamentais

Para uma compreensão completa do desempenho de algoritmos, é essencial considerar métricas descritivas que sumarizam as características de um conjunto de dados. Essas métricas fornecem uma visão concisa da distribuição dos resultados empíricos.

3.1.5 Média

A média aritmética é a soma de todos os valores em um conjunto de dados dividida pelo número de valores. É a medida de tendência central mais comum e representa o valor "típico"de um conjunto de observações . Em análise de algoritmos, a média do tempo de execução ou do uso de memória pode indicar o desempenho esperado de um algoritmo sob certas condições. No entanto, a média pode ser sensível a valores extremos (outliers).

3.1.6 Moda

A moda é o valor que aparece com maior frequência em um conjunto de dados. Em alguns contextos de análise de algoritmos, a moda pode ser útil para identificar o tempo de execução ou o uso de memória mais comum, especialmente se a distribuição dos dados for

assimétrica ou bimodal. Por exemplo, se um algoritmo tem um desempenho que se agrupa em torno de dois valores distintos, a moda pode ajudar a identificar esses picos.

3.1.7 Variância

A variância é uma medida de dispersão que quantifica o quão espalhados os dados estão em relação à média. Uma variância alta indica que os pontos de dados estão distantes da média e uns dos outros, enquanto uma variância baixa sugere que os pontos de dados estão próximos da média. Em análise de algoritmos, a variância do tempo de execução pode indicar a consistência do desempenho do algoritmo. Algoritmos com baixa variância são mais previsíveis.

3.1.8 Desvio Padrão

O desvio padrão é a raiz quadrada da variância e é expresso nas mesmas unidades que os dados originais, tornando-o mais interpretável que a variância. Ele mede a quantidade média de variabilidade ou dispersão em um conjunto de dados. Um desvio padrão pequeno indica que os dados tendem a estar próximos da média, enquanto um desvio padrão grande indica que os dados estão mais espalhados. Assim como a variância, o desvio padrão é crucial para entender a previsibilidade e a robustez de um algoritmo.

3.2 Testes de Hipóteses para Comparação de Médias

Durante a fundamentação teórica, foram estudados diversos testes estatísticos para comparação de médias, como os testes t-Student e Z-test, que são ferramentas fundamentais para determinar se diferenças observadas entre amostras são estatisticamente significativas. No entanto, esses testes não foram aplicados neste trabalho, pois o objetivo principal foi realizar uma comparação empírica visual e gráfica dos tempos de execução, ajustando curvas de complexidade aos dados observados. A Teoria dos Grandes Números garante que as médias de várias execuções aproximam o comportamento real do algoritmo, reduzindo a necessidade de testes estatísticos formais para validação. A análise foi conduzida através da comparação direta dos valores de RMSE e SSE entre os modelos ajustados, o que fornece uma métrica objetiva e suficiente para identificar qual curva de complexidade melhor representa o comportamento empírico de cada algoritmo.

3.3 Métodos de adaptação de curvas

A identificação empírica das classes de complexidade considerou quatro famílias de equações que representam os comportamentos assintóticos mais recorrentes na literatura (Ngu, 2022; Bimurat *et al.*, 2023; Samsudin *et al.*, 2020; Folaranmi; Ayepeku, 2023). Cada hipótese foi implementada como função de custo parametrizada e ajustada aos tempos observados, permitindo comparar diretamente as curvas teóricas com os dados coletados. **Modelo linear ($O(n)$)**: $t(n) = an + b$. Captura rotinas cujo tempo cresce proporcionalmente ao tamanho de entrada, típicas de algoritmos baseados em contagens diretas ou passes únicos sobre os dados. **Modelo linear-logarítmico ($O(n\log n)$)**: $t(n) = an\log n + bn + c$. Representa algoritmos de divisão e conquista, como Merge Sort e Quick Sort, que combinam partições logarítmicas com processamento linear em cada nível da recursão. **Modelo quadrático ($O(n^2)$)**: $t(n) = an^2 + bn + c$. Aproxima algoritmos com aninhamento de laços sobre os mesmos dados, como Bubble Sort, Selection Sort e Insertion Sort em seus piores cenários. **Modelo linear com termo logarítmico**: $t(n) = an + b\log n$. Usado para verificar ajustes intermediários quando a componente logarítmica influencia o comportamento mas o crescimento dominante permanece linear. **Critério de escolha.** O modelo selecionado para cada algoritmo é aquele que obtém o menor valor de RMSE, métrica que sintetiza a discrepância média entre valores observados e previstos. A comparação restrita a funções de ordem baixa reduz o risco de sobreajuste e mantém o custo computacional do ajuste compatível com múltiplas execuções experimentais (Alridha; Alsharify; Al-Khafaji, 2024). **Justificativa prática.** A combinação dessas quatro famílias equilibra precisão e esforço de implementação: polinômios de grau superior tenderiam a oscilar e exigir mais dados, enquanto funções exponenciais ou combinatórias aumentariam o custo de otimização sem respaldo teórico para os algoritmos estudados. Além disso, as equações lineares e quadráticas permitem interpretar diretamente os coeficientes aprendidos, favorecendo a análise de trade-offs de tempo e uso de recursos (Folaranmi; Ayepeku, 2023).

3.4 Método dos Mínimos Quadrados (MMQ)

3.4.0.0.1 Nota sobre aplicação neste trabalho.

O MMQ é aqui apresentado como fundamento teórico. Nesta pesquisa, o ajuste efetivo dos modelos foi realizado por **gradiente descendente adaptativo** (minimizando o RMSE),

evitando a solução matricial clássica do MMQ. O Método dos Mínimos Quadrados (MMQ) é uma técnica de otimização matemática que busca encontrar a função que melhor se ajusta a um conjunto de dados. O princípio fundamental do MMQ é minimizar a soma dos quadrados das diferenças entre os valores observados e os valores previstos pelo modelo. Essas diferenças são chamadas de resíduos ou erros (Almeida, s.d.).

Matematicamente, dado um conjunto de n pontos de dados (x_i, y_i), onde x_i é a variável independente e y_i é a variável dependente, e uma função modelo $f(x, \beta)$ com parâmetros β, o MMQ busca encontrar os valores de β que minimizam a soma dos quadrados dos resíduos S, definida como:

$$S = \sum_{i=1}^{n} (y_i - f(x_i, \beta))^2$$

O MMQ foi estudado teoricamente como base conceitual para compreensão dos métodos de ajuste de curvas. No entanto, neste trabalho, o método realmente aplicado foi o **Gradiente Descendente Adaptativo** (baseado no otimizador Adam), que resolve o problema de minimização de forma iterativa, garantindo estabilidade numérica e evitando o cálculo matricial direto da solução analítica. Essa abordagem é particularmente adequada para modelos onde a solução analítica pode ser numericamente instável ou computacionalmente custosa (Almeida, s.d.; Nobrega, 2014).

3.4.1 Vantagens do MMQ

Simplicidade e Interpretabilidade: Para modelos lineares, o MMQ é conceitualmente simples e os parâmetros estimados são facilmente interpretáveis. **Eficiência e Consistência:** Sob certas premissas, os estimadores de mínimos quadrados são os melhores estimadores lineares não viesados (BLUE - *Best Linear Unbiased Estimators*) (Almeida, s.d.). **Base para Outros Métodos:** O MMQ serve como base para muitos outros métodos estatísticos e de aprendizado de máquina.

3.4.2 Desvantagens do MMQ

Sensibilidade a Outliers: O MMQ é sensível a valores atípicos, pois dá um peso maior a erros grandes (Almeida, s.d.). **Premissas Rígidas:** A violação das premissas do modelo pode levar a estimativas viesadas ou ineficientes (Almeida, s.d.). **Problemas de Multicolinearidade:** A

multicolinearidade pode tornar os parâmetros instáveis. **Não Lida Bem com Relações Não Lineares:** Para relações complexas, o MMQ pode ser ineficaz.

3.5 Regressão e Ajuste de Modelos

A Regressão Linear é um modelo estatístico que estabelece uma relação linear entre uma variável dependente e uma variável independente (Chein, 2019). Embora a regressão linear simples seja um caso base importante, este trabalho utiliza ajuste de modelos não lineares para capturar diferentes classes de complexidade algorítmica. O sistema utiliza apenas uma variável independente (n, tamanho da entrada) e uma dependente (tempo de execução), mas testa quatro modelos distintos correspondentes às complexidades teóricas usuais:
Modelo Linear: $T(n) = a \cdot n + b$ **Modelo Quadrático**: $T(n) = a \cdot n^2 + b \cdot n + c$ **Modelo Logarítmico**: $T(n) = a \cdot n + b \cdot \log(n)$ **Modelo Linear-Logarítmico (N log N)**: $T(n) = a \cdot n \cdot \log(n) + b \cdot n + c$

O ajuste desses modelos é realizado de forma iterativa usando o Gradiente Descendente Adaptativo (Adam), que minimiza o erro quadrático médio (RMSE) entre os valores observados e os valores previstos pelo modelo. O modelo com menor RMSE é selecionado como a complexidade experimental aproximada do algoritmo testado.

3.5.1 Vantagens da Regressão Linear

Simplicidade e Interpretabilidade: Fácil de entender e interpretar (Chein, 2019). **Eficiência Computacional:** Rápida e eficaz para grandes conjuntos de dados. **Base para Modelos Mais Complexos:** Serve como referência e ponto de partida. **Robustez para Relações Lineares:** Muito eficaz quando a relação é de fato linear.

3.5.2 Desvantagens da Regressão Linear

Assunção de Linearidade: Não captura relações não lineares (Chein, 2019). **Sensibilidade a Outliers:** Pode distorcer a linha de regressão. **Premissas Estatísticas Rígidas:** Violá-las pode invalidar inferências. **Multicolinearidade:** Dificulta a interpretação. **Dados Categóricos Complexos:** Pode ser inadequada para estruturas complexas.

3.6 Gradiente Descendente

O Gradiente Descendente é um algoritmo de otimização iterativo de primeira ordem utilizado para minimizar funções de custo em modelos de aprendizado de máquina (Nobrega, 2014). O algoritmo realiza os seguintes passos:
Inicialização: Valores iniciais arbitrários para os parâmetros. **Cálculo do Gradiente:** Deriva o gradiente da função de custo. **Atualização:** Atualiza os parâmetros usando:

$\theta \leftarrow \theta - \alpha \nabla J(\theta)$

Iteração: Repete até convergência ou número máximo de iterações.

3.6.1 Variações do Gradiente Descendente

Batch Gradient Descent: Usa todo o conjunto de dados. **Stochastic Gradient Descent (SGD):** Atualiza por amostra. **Mini-Batch Gradient Descent:** Compromisso entre os dois anteriores.

3.6.2 Vantagens

Versatilidade: Aplicável a vários modelos. **Escalabilidade:** Ideal para grandes datasets (Nobrega, 2014). **Simplicidade:** Conceito direto. **Adaptabilidade:** Pode usar otimizadores como Adam, RMSprop.

3.6.3 Desvantagens

Taxa de Aprendizado Sensível: Afeta a convergência (Nobrega, 2014). **Mínimos Locais e Pontos de Sela:** Pode dificultar encontrar o ótimo global. **Gradientes Desvanecentes ou Explodindo:** Complicam o treinamento. **Custo Computacional:** Batch Gradient Descent é pesado para grandes volumes.

3.7 Métricas de Avaliação do Ajuste

3.7.1 Cálculo do Erro

Para avaliar a performance de modelos preditivos, especialmente em tarefas de regressão, é fundamental quantificar a diferença entre os valores previstos pelo modelo e os valores reais observados. Essa quantificação é realizada através de métricas de erro, que fornecem uma medida

da precisão do modelo. Uma das métricas mais amplamente utilizadas para esse fim é o Erro Quadrático Médio da Raiz, ou RMSE (do inglês, *Root Mean Squared Error*).

3.7.1.1 Algoritmo de RMSE

O RMSE é uma medida da magnitude média dos erros. Ele representa a raiz quadrada da média dos quadrados de todos os erros. A fórmula do RMSE é dada por:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Onde:
n: número total de observações ou pontos de dados. $\hat{y}_i$: valor previsto pelo modelo para a i-ésima observação. y_i: valor real observado para a i-ésima observação. O RMSE é uma métrica que penaliza erros maiores de forma mais significativa devido ao termo quadrático. Isso significa que erros grandes contribuem desproporcionalmente para o valor final do RMSE, tornando-o sensível a *outliers* (Kosourova, 2025).

3.7.1.2 Vantagens do RMSE

Interpretabilidade: O RMSE é expresso nas mesmas unidades da variável dependente (o que está sendo previsto), o que facilita a interpretação do erro em um contexto prático. Por exemplo, se você está prevendo preços de casas, um RMSE de 10.000 significa que, em média, as previsões do seu modelo estão erradas em 10.000 unidades monetárias (Frost, 2020). **Sensibilidade a Grandes Erros:** Devido à sua natureza quadrática, o RMSE atribui um peso maior a erros maiores. Isso é vantajoso em cenários onde erros grandes são particularmente indesejaveis e precisam ser fortemente penalizados (Kosourova, 2025). **Amplamente Utilizado:** O RMSE é uma métrica padrão e amplamente aceita na comunidade de aprendizado de máquina e estatística, o que facilita a comparação de resultados entre diferentes modelos e estudos (ARIZE, 2023).

3.7.1.3 Desvantagens do RMSE

Sensibilidade a Outliers: A mesma característica que o torna sensível a grandes erros também o torna vulnerável a *outliers*. Um único valor atípico pode inflacionar significativamente o RMSE, mesmo que o modelo tenha um bom desempenho na maioria dos dados (Kosourova, 2025). **Não**

Fornece Direção do Erro: O RMSE informa a magnitude do erro, mas não indica a direção (se o modelo está superestimando ou subestimando os valores) (Kosourova, 2025). **Dificuldade de Comparação entre Conjuntos de Dados Diferentes:** Como o RMSE é dependente da escala da variável dependente, não é apropriado para comparar o desempenho de modelos em conjuntos de dados com escalas diferentes (Frost, 2020).

4 RESULTADOS

4.1 Resultados para 50 pontos (p50)

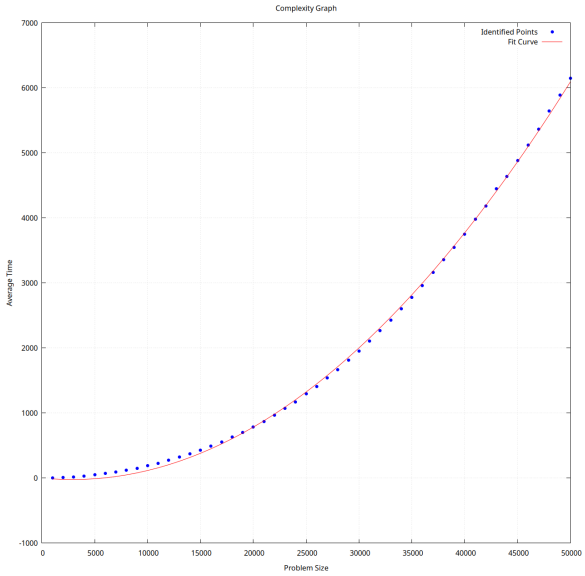
4.1.1 Dataset Aleatório (p50)

Tabela 1 – Resultados para dataset aleatório com 50 pontos.

Algoritmo	**Tempo (s)**	**Comparações**
Bubble Sort	34.789	1249975000
Insertion Sort	5.268	625311250
Merge Sort	0.027	833104
Quick Sort	0.012	749962
Heap Sort	0.024	931835
Counting Sort	0.003	0
Radix Sort	0.015	0
Bucket Sort	0.006	0

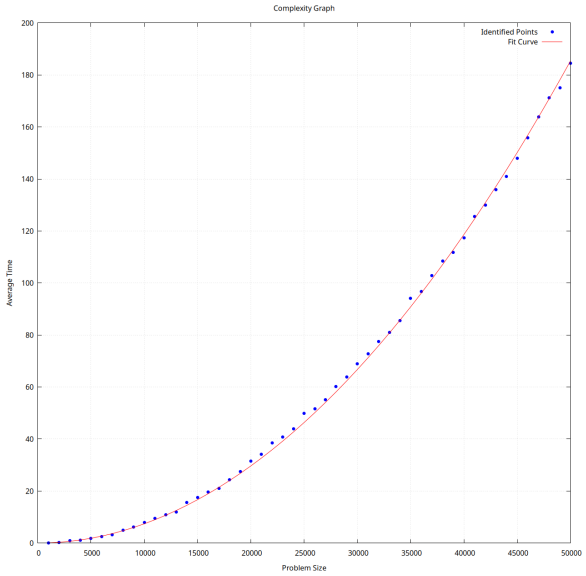
Fonte: Autoria própria (2025).

Figura 3 – Bubble Sort em dados aleatórios (50 pontos).



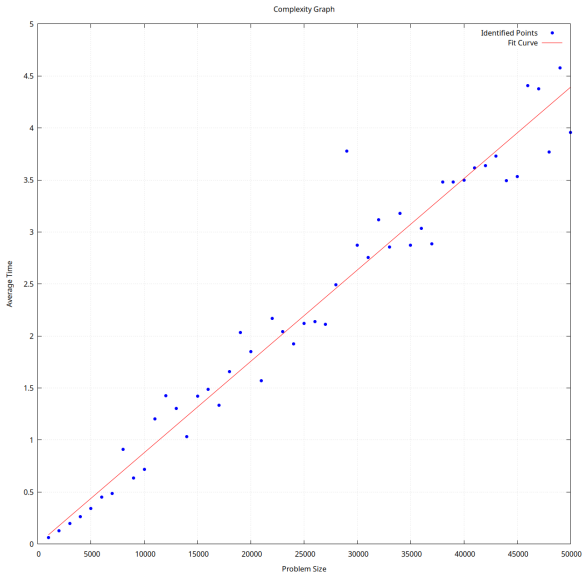
Fonte: Autoria própria (2025).

Figura 4 – Insertion Sort em dados aleatórios (50 pontos).



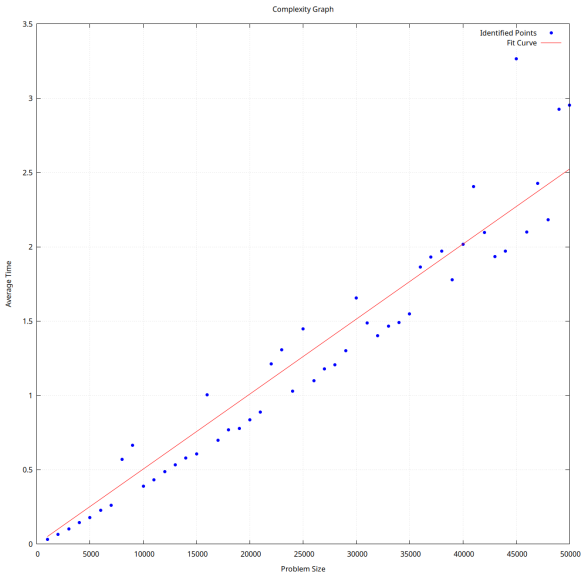
Fonte: Autoria própria (2025).

Figura 5 – Merge Sort em dados aleatórios (50 pontos).



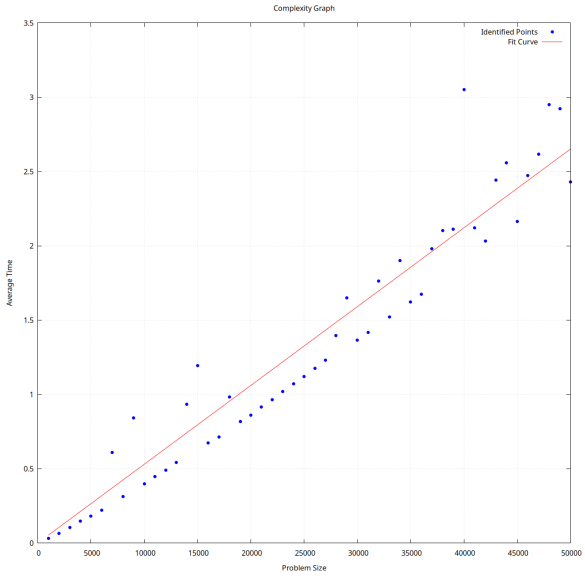
Fonte: Autoria própria (2025).

Figura 6 – Quick Sort em dados aleatórios (50 pontos).



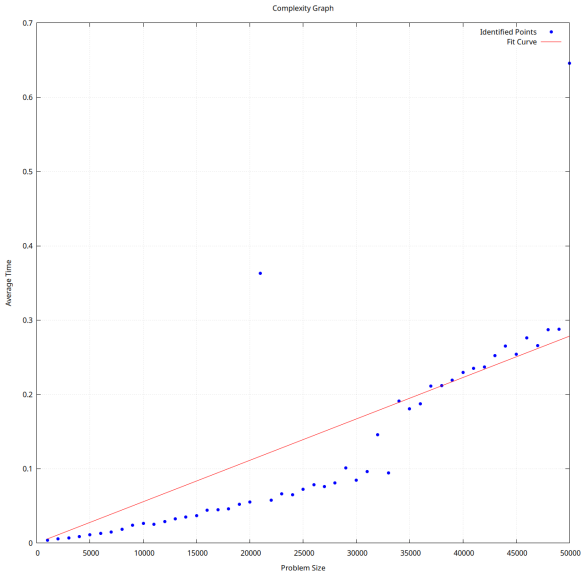
Fonte: Autoria própria (2025).

Figura 7 – Heap Sort em dados aleatórios (50 pontos).



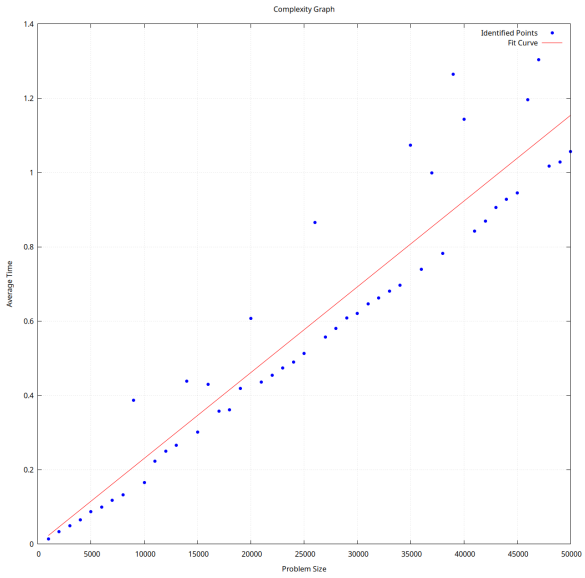
Fonte: Autoria própria (2025).

Figura 8 – Counting Sort em dados aleatórios (50 pontos).



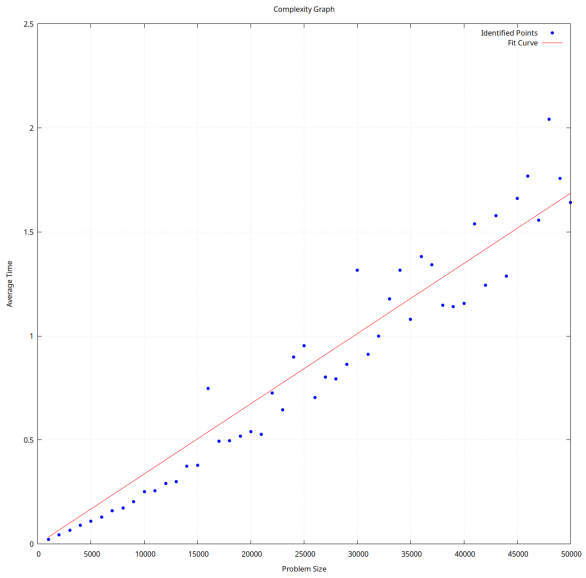
Fonte: Autoria própria (2025).

Figura 9 – Radix Sort em dados aleatórios (50 pontos).



Fonte: Autoria própria (2025).

Figura 10 – Bucket Sort em dados aleatórios (50 pontos).



Fonte: Autoria própria (2025).

Interpretação dos gráficos.

Bubble Sort (3): O ajuste $O(n^2)$ (RMSE = 43.55) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto aleatório com 50 pontos. **Insertion Sort (4)**: O ajuste $O(n^2)$ (RMSE = 1.40) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto aleatório com 50 pontos. **Merge Sort (5)**: O ajuste $O(n \log n)$ (RMSE = 0.27) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 50 pontos. **Quick Sort (6)**: O ajuste $O(n \log n)$ (RMSE = 0.23) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 50 pontos. **Heap Sort (7)**: O ajuste $O(n \log n)$ (RMSE = 0.23) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 50 pontos. **Counting Sort (8)**: O ajuste $O(n)$ (RMSE = 0.07) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 50 pontos. **Radix Sort (9)**: O ajuste $O(n)$ (RMSE = 0.11) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 50 pontos. **Bucket Sort (10)**: O ajuste $O(n)$ (RMSE = 0.14) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 50 pontos.

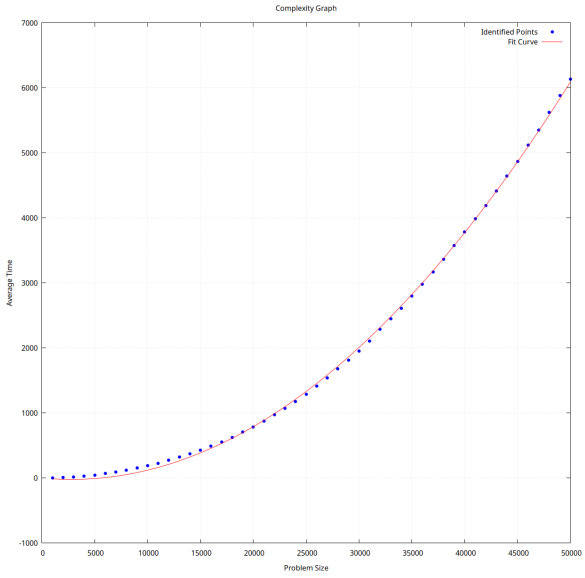
4.1.2 Dataset Ordenado (p50)

Tabela 2 – Resultados para dataset ordenado com 50 pontos.

Algoritmo	**Tempo (s)**	**Comparações**
Bubble Sort	13.931	1249975000
Insertion Sort	0.002	49999
Merge Sort	0.021	416552
Quick Sort	0.441	1249975000
Heap Sort	0.023	931835
Counting Sort	0.003	0
Radix Sort	0.015	0
Bucket Sort	0.010	0

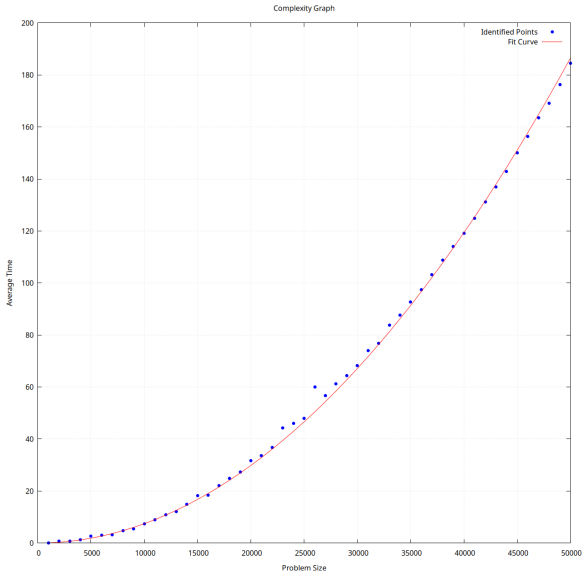
Fonte: Autoria própria (2025).

Figura 11 – Bubble Sort sobre entrada ordenada (50 pontos).



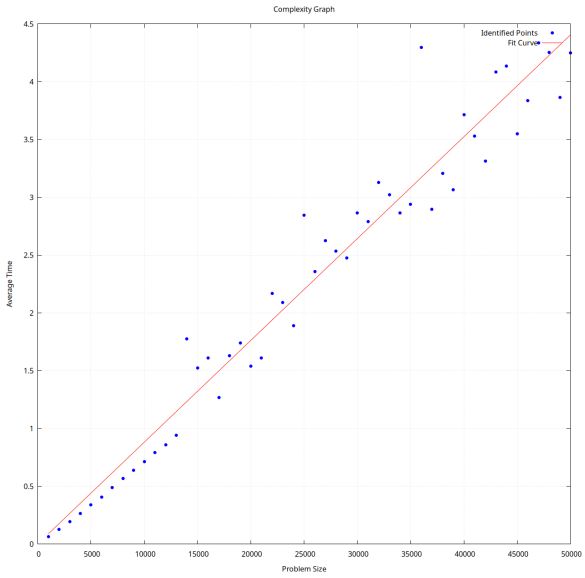
Fonte: Autoria própria (2025).

Figura 12 – Insertion Sort sobre entrada ordenada (50 pontos).



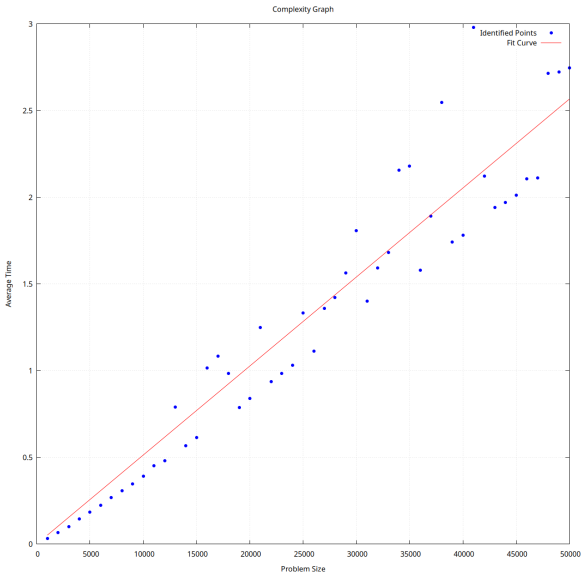
Fonte: Autoria própria (2025).

Figura 13 – Merge Sort sobre entrada ordenada (50 pontos).



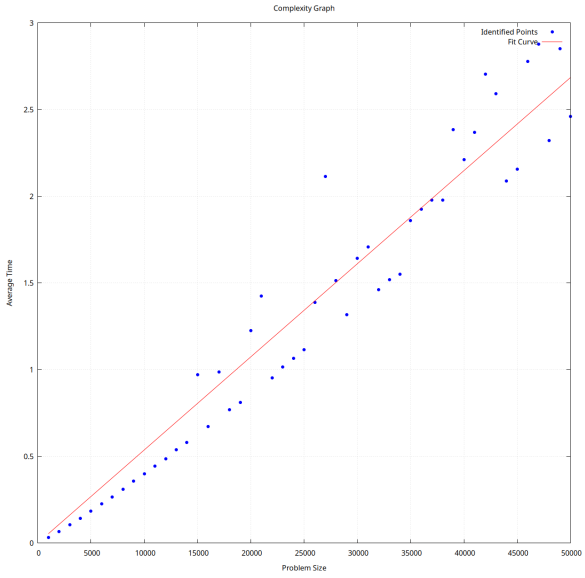
Fonte: Autoria própria (2025).

Figura 14 – Quick Sort sobre entrada ordenada (50 pontos).



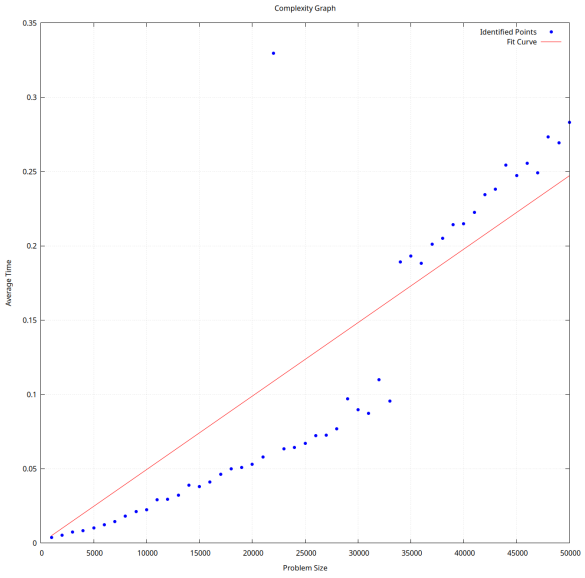
Fonte: Autoria própria (2025).

Figura 15 – Heap Sort sobre entrada ordenada (50 pontos).



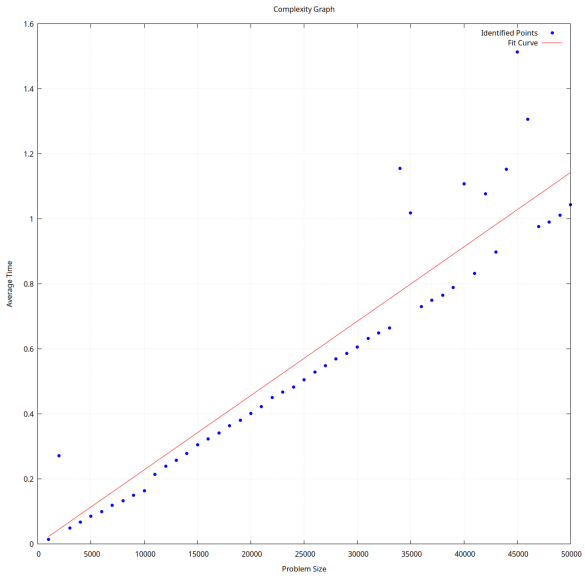
Fonte: Autoria própria (2025).

Figura 16 – Counting Sort sobre entrada ordenada (50 pontos).



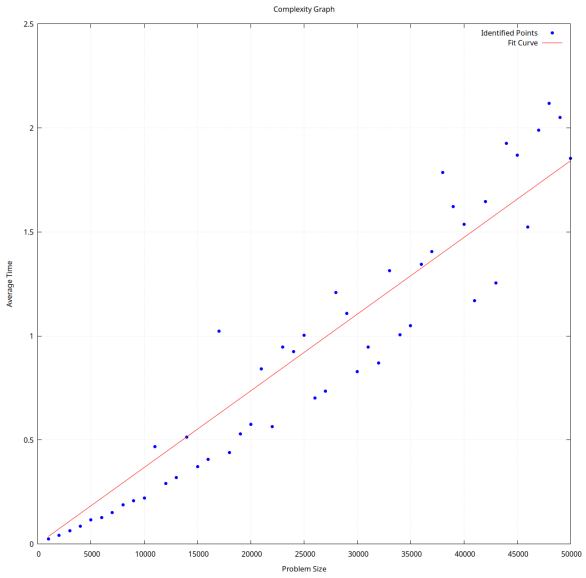
Fonte: Autoria própria (2025).

Figura 17 – Radix Sort sobre entrada ordenada (50 pontos).



Fonte: Autoria própria (2025).

Figura 18 – Bucket Sort sobre entrada ordenada (50 pontos).



Fonte: Autoria própria (2025).

Interpretação dos gráficos.

Bubble Sort (11): O ajuste $O(n^2)$ (RMSE = 39.63) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto ordenado com 50 pontos. **Insertion Sort (12)**: O tempo de execução muito inferior ao caso aleatório e o número reduzido de comparações confirmam o melhor caso do algoritmo, que se aproxima de $O(n)$ quando os dados já estão ordenados. **Merge Sort (13)**: O ajuste $O(n \log n)$ (RMSE = 0.28) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 50 pontos. **Quick Sort (14)**: O elevado número de comparações e o aumento significativo no tempo de execução indicam que a entrada ordenada representa um cenário desfavorável para o Quick Sort, aproximando seu comportamento do pior caso quadrático. **Heap Sort (15)**: O ajuste $O(n \log n)$ (RMSE = 0.21) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 50 pontos. **Counting Sort (16)**: O ajuste $O(n)$ (RMSE = 0.05) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 50 pontos. **Radix Sort (17)**: O ajuste $O(n)$ (RMSE = 0.13) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 50 pontos. **Bucket Sort (18)**: O ajuste $O(n)$ (RMSE = 0.19) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 50 pontos.

Interpretação dos gráficos.

Bubble Sort (11): O ajuste $O(n^2)$ (RMSE = 39.63) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto ordenado com 50 pontos. **Insertion Sort (12)**: O tempo de execução muito inferior ao caso aleatório e o número reduzido de comparações confirmam o melhor caso do algoritmo, que se aproxima de $O(n)$ quando os dados já estão ordenados. **Merge Sort (13)**: O ajuste $O(n \log n)$ (RMSE = 0.28) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 50 pontos. **Quick Sort (14)**: O elevado número de comparações e o aumento significativo no tempo de execução indicam que a entrada ordenada representa um cenário desfavorável para o Quick Sort, aproximando seu comportamento do pior caso quadrático. **Heap Sort (15)**: O ajuste $O(n \log n)$ (RMSE = 0.21) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 50 pontos. **Counting Sort (16)**: O ajuste $O(n)$ (RMSE = 0.05) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 50 pontos. **Radix Sort (17)**: O ajuste $O(n)$ (RMSE = 0.13) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 50 pontos. **Bucket Sort (18)**: O ajuste $O(n)$ (RMSE = 0.19) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 50 pontos.

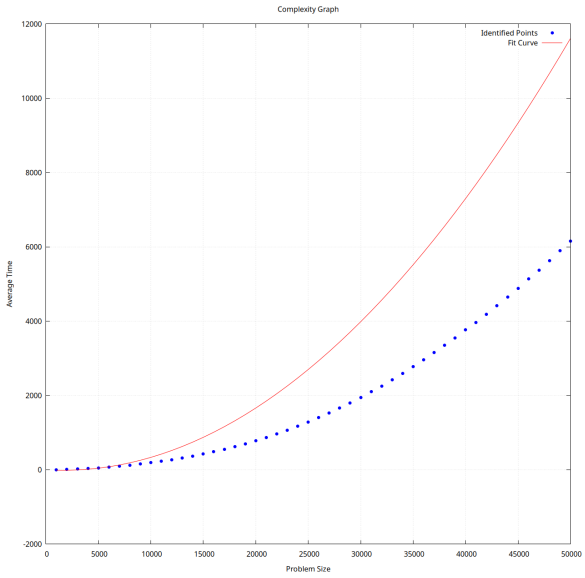
4.1.3 Dataset Reverso (p50)

Tabela 3 – Resultados para dataset reverso com 50 pontos.

Algoritmo	**Tempo (s)**	**Comparações**
Bubble Sort	21.033	1249975000
Insertion Sort	10.322	1249975000
Merge Sort	0.021	416552
Quick Sort	0.457	1249975000
Heap Sort	0.023	931835
Counting Sort	0.003	0
Radix Sort	0.015	0
Bucket Sort	0.011	0

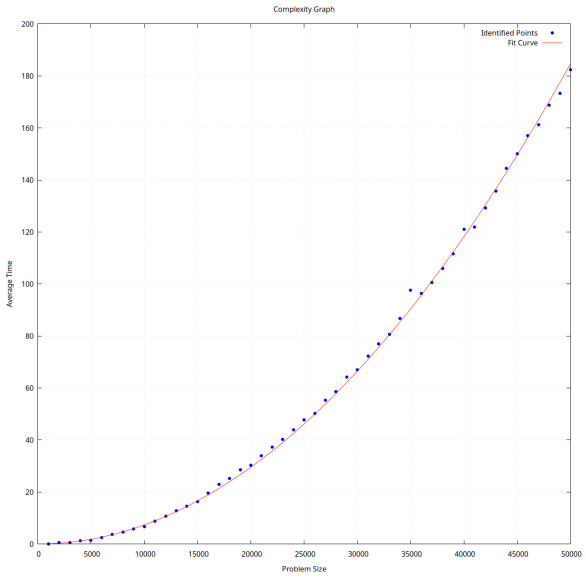
Fonte: Autoria própria (2025).

Figura 19 – Bubble Sort com entrada reversa (50 pontos).



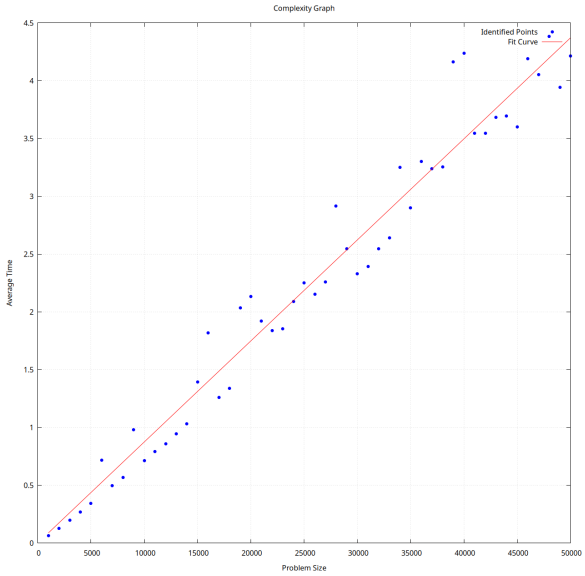
Fonte: Autoria própria (2025).

Figura 20 – Insertion Sort com entrada reversa (50 pontos).



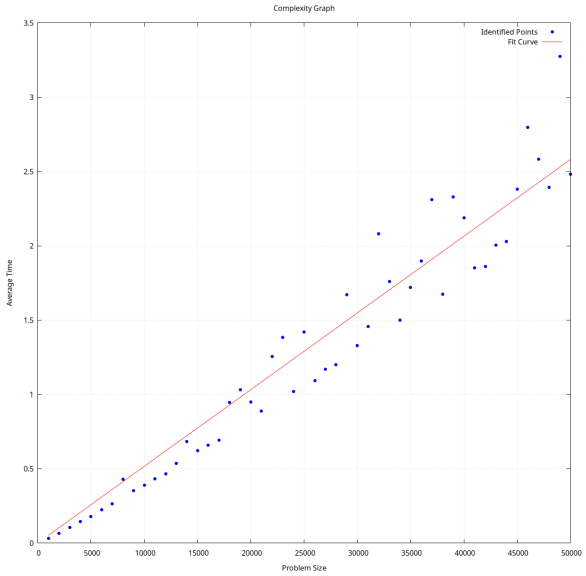
Fonte: Autoria própria (2025).

Figura 21 – Merge Sort com entrada reversa (50 pontos).



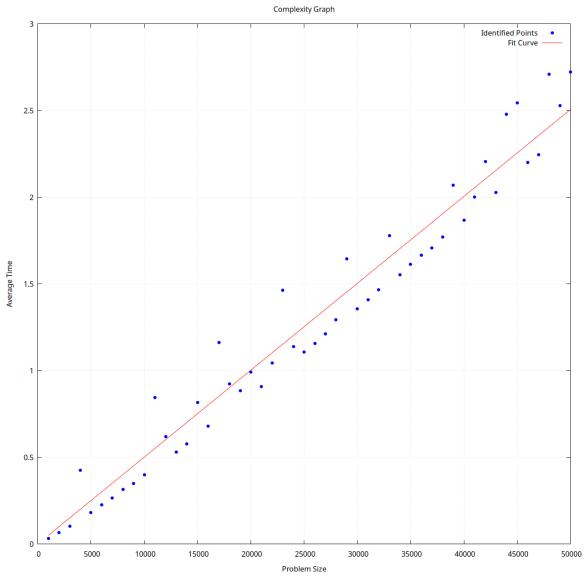
Fonte: Autoria própria (2025).

Figura 22 – Quick Sort com entrada reversa (50 pontos).



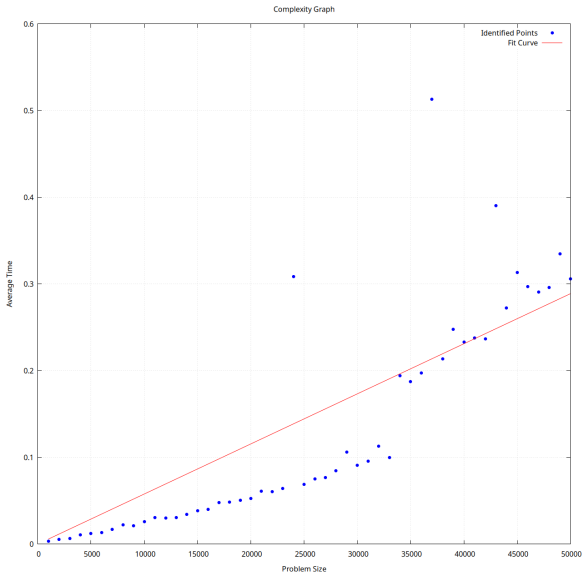
Fonte: Autoria própria (2025).

Figura 23 – Heap Sort com entrada reversa (50 pontos).



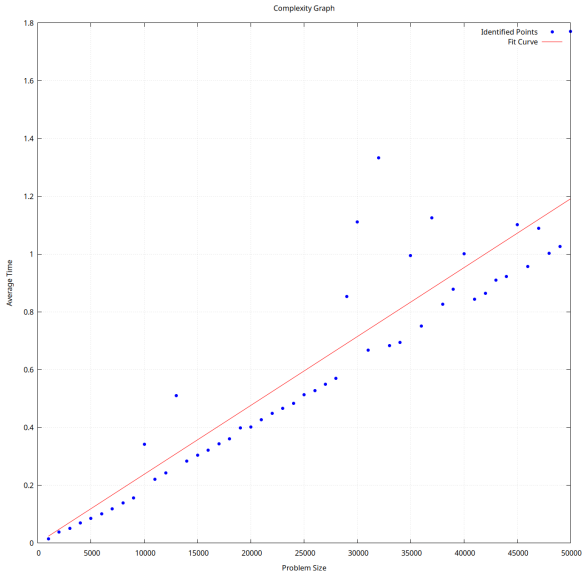
Fonte: Autoria própria (2025).

Figura 24 – Counting Sort com entrada reversa (50 pontos).



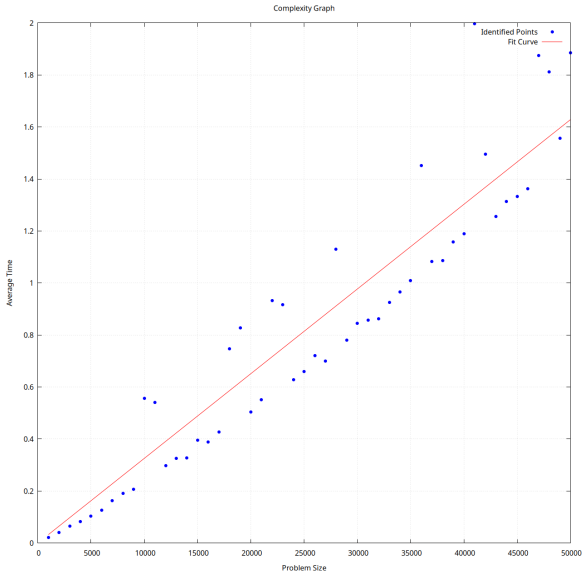
Fonte: Autoria própria (2025).

Figura 25 – Radix Sort com entrada reversa (50 pontos).



Fonte: Autoria própria (2025).

Figura 26 – Bucket Sort com entrada reversa (50 pontos).



Fonte: Autoria própria (2025).

Interpretação dos gráficos.

Bubble Sort (19): O ajuste $O(n^2)$ (RMSE = 2528.22) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 50 pontos. A ordem inversa expõe o pior caso do algoritmo, com o maior tempo entre as abordagens comparadas. **Insertion**

Sort (20): O ajuste $O(n^2)$ (RMSE = 1.62) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 50 pontos. A ordem inversa impõe trocas em todas as iterações, justificando o crescimento quadrático observado empiricamente. **Merge Sort (21)**: O ajuste $O(n \log n)$ (RMSE = 0.25) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 50 pontos. **Quick Sort (22)**: O ajuste $O(n \log n)$ (RMSE = 0.22) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 50 pontos. **Heap Sort (23)**: O ajuste $O(n \log n)$ (RMSE = 0.15) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 50 pontos. **Counting Sort (24)**: O ajuste $O(n)$ (RMSE = 0.07) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 50 pontos. **Radix Sort (25)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 50 pontos. **Bucket Sort (26)**: O ajuste $O(n)$ (RMSE = 0.18) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 50 pontos.

Interpretação dos gráficos.

Bubble Sort (19): O ajuste $O(n^2)$ (RMSE = 2528.22) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 50 pontos. A ordem inversa expõe o pior caso do algoritmo, com o maior tempo entre as abordagens comparadas. **Insertion Sort (20)**: O ajuste $O(n^2)$ (RMSE = 1.62) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 50 pontos. A ordem inversa impõe trocas em todas as iterações, justificando o crescimento quadrático observado empiricamente. **Merge Sort (21)**: O ajuste $O(n \log n)$ (RMSE = 0.25) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 50 pontos. **Quick Sort (22)**: O ajuste $O(n \log n)$ (RMSE = 0.22) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 50 pontos. **Heap Sort (23)**: O ajuste $O(n \log n)$ (RMSE = 0.15) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 50 pontos. **Counting Sort (24)**: O ajuste $O(n)$ (RMSE = 0.05) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 50 pontos. **Radix Sort (25)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 50 pontos. **Bucket Sort (26)**: O ajuste $O(n)$ (RMSE = 0.15)

evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 50 pontos.

4.2 Resultados para 100 pontos (p100)

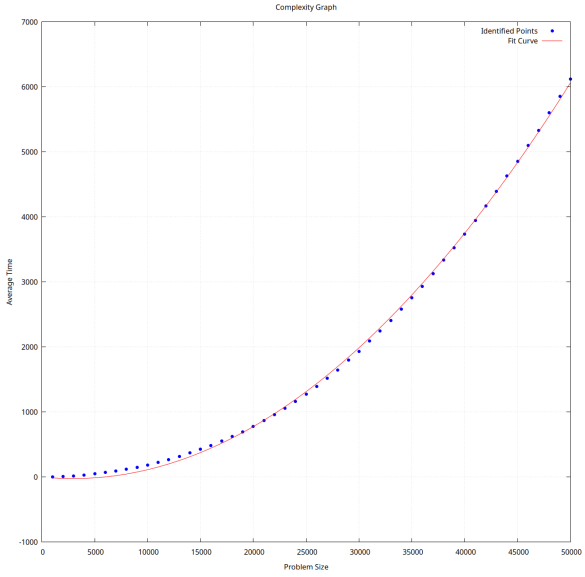
4.2.1 Dataset Aleatório (p100)

Tabela 4 – Resultados para dataset aleatório com 100 pontos.

Algoritmo	**Tempo (s)**	**Comparações**
Bubble Sort	142.155	4999950000
Insertion Sort	21.161	2500741349
Merge Sort	0.057	1766028
Quick Sort	0.024	1551829
Heap Sort	0.055	2096696
Counting Sort	0.006	0
Radix Sort	0.030	0
Bucket Sort	0.012	0

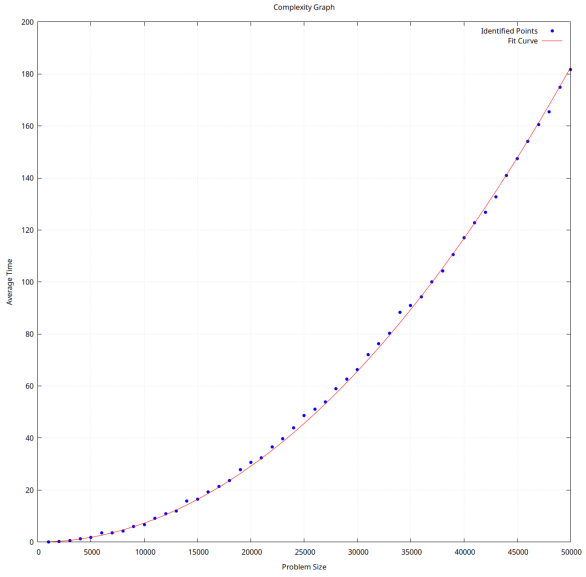
Fonte: Autoria própria (2025).

Figura 27 – Bubble Sort em dados aleatórios (100 pontos).



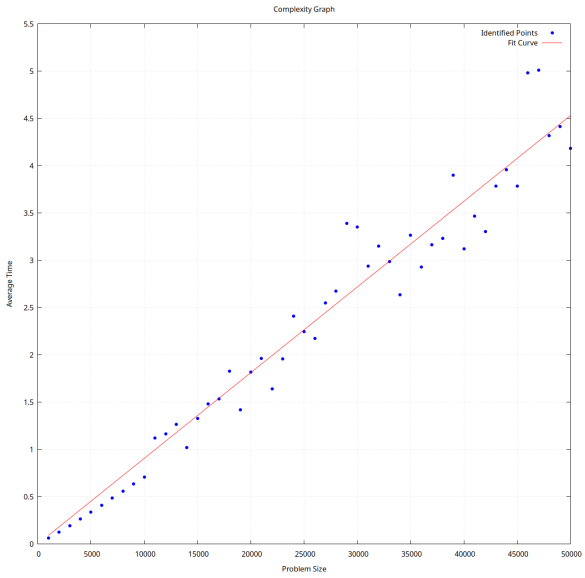
Fonte: Autoria própria (2025).

Figura 28 – Insertion Sort em dados aleatórios (100 pontos).



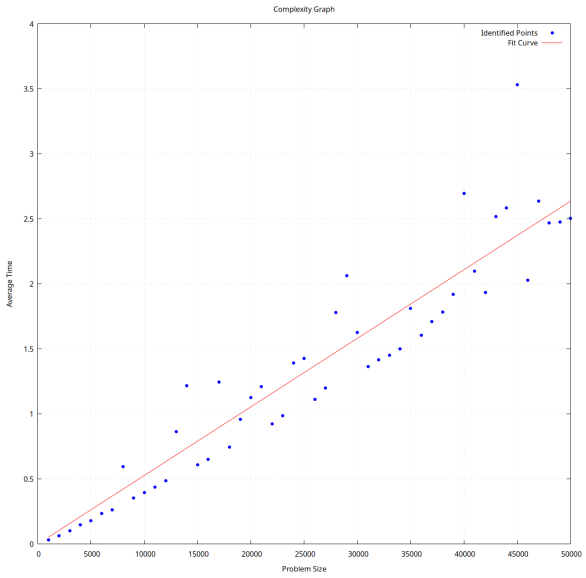
Fonte: Autoria própria (2025).

Figura 29 – Merge Sort em dados aleatórios (100 pontos).



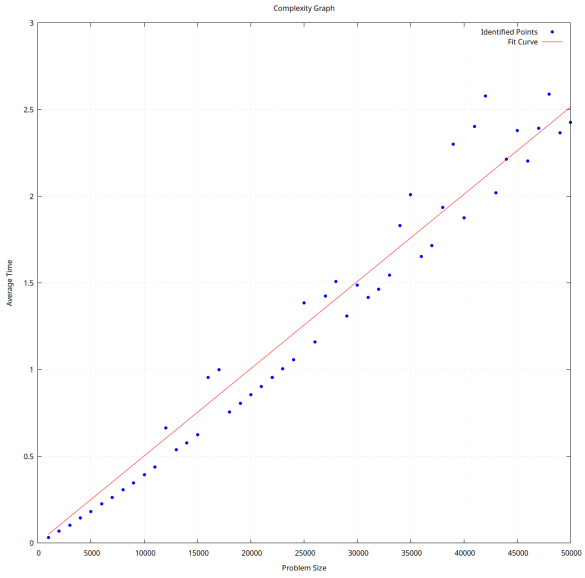
Fonte: Autoria própria (2025).

Figura 30 – Quick Sort em dados aleatórios (100 pontos).



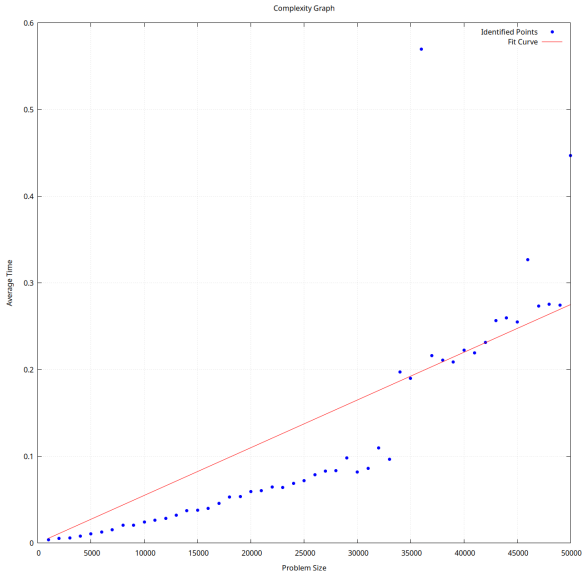
Fonte: Autoria própria (2025).

Figura 31 – Heap Sort em dados aleatórios (100 pontos).



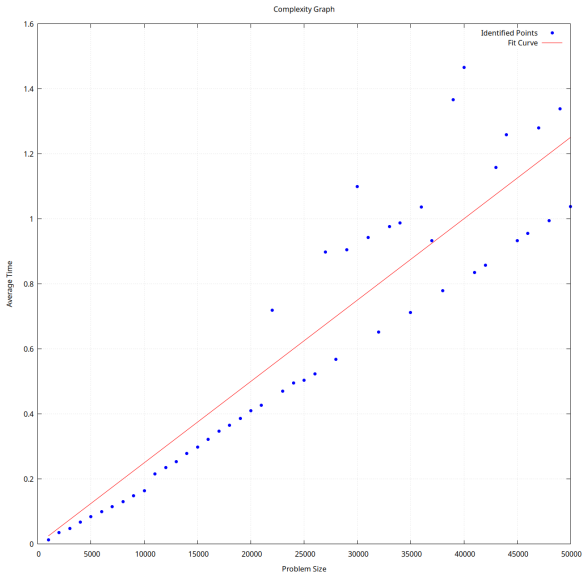
Fonte: Autoria própria (2025).

Figura 32 – Counting Sort em dados aleatórios (100 pontos).



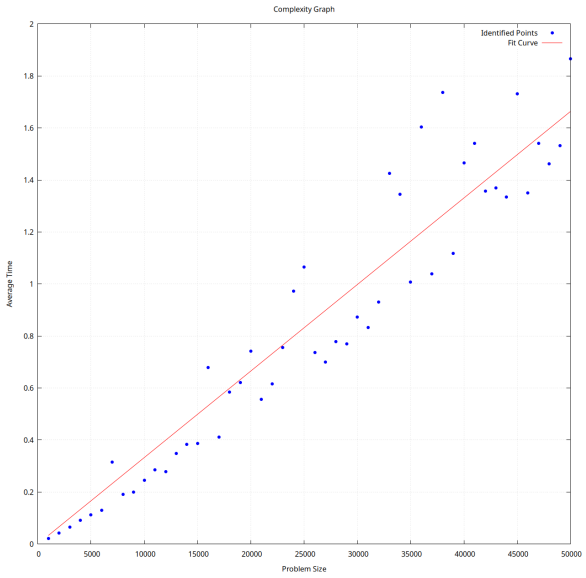
Fonte: Autoria própria (2025).

Figura 33 – Radix Sort em dados aleatórios (100 pontos).



Fonte: Autoria própria (2025).

Figura 34 – Bucket Sort em dados aleatórios (100 pontos).



Fonte: Autoria própria (2025).

Interpretação dos gráficos.

Bubble Sort (27): O ajuste $O(n^2)$ (RMSE = 44.91) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto aleatório com 100 pontos. **Insertion Sort (28)**: O ajuste $O(n^2)$ (RMSE = 1.25) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto aleatório com 100 pontos. **Merge Sort (29)**: O ajuste $O(n \log n)$ (RMSE = 0.29) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 100 pontos. **Quick Sort (30)**: O ajuste $O(n \log n)$ (RMSE = 0.28) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 100 pontos. **Heap Sort (31)**: O ajuste $O(n \log n)$ (RMSE = 0.15) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 100 pontos. **Counting Sort (32)**: O ajuste $O(n)$ (RMSE = 0.07) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 100 pontos. **Radix Sort (33)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 100 pontos. **Bucket Sort (34)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 100 pontos.

Interpretação dos gráficos.

Bubble Sort (27): O ajuste $O(n^2)$ (RMSE = 44.91) reforça o crescimento quadrático caracte-

rístico de algoritmos com laços aninhados no conjunto aleatório com 100 pontos. **Insertion Sort (28)**: O ajuste $O(n^2)$ (RMSE = 1.25) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto aleatório com 100 pontos. **Merge Sort (29)**: O ajuste $O(n \log n)$ (RMSE = 0.29) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 100 pontos. **Quick Sort (30)**: O ajuste $O(n \log n)$ (RMSE = 0.28) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 100 pontos. **Heap Sort (31)**: O ajuste $O(n \log n)$ (RMSE = 0.15) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto aleatório com 100 pontos. **Counting Sort (32)**: O ajuste $O(n)$ (RMSE = 0.07) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 100 pontos. **Radix Sort (33)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 100 pontos. **Bucket Sort (34)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto aleatório com 100 pontos.

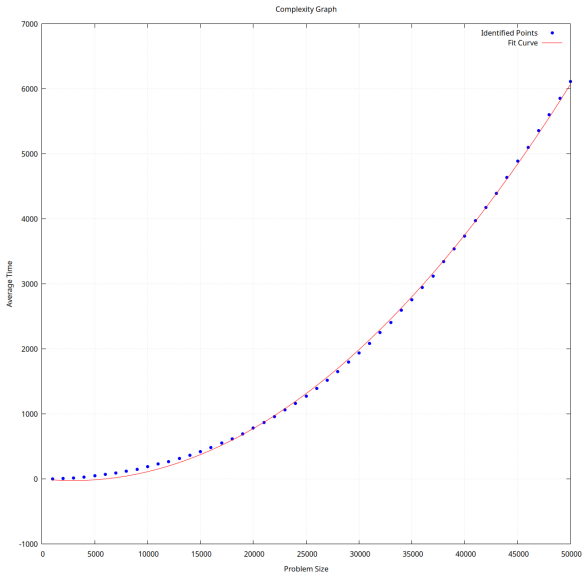
4.2.2 Dataset Ordenado (p100)

Tabela 5 – Resultados para dataset ordenado com 100 pontos.

Algoritmo	**Tempo (s)**	**Comparações**
Bubble Sort	56.550	4999950000
Insertion Sort	0.003	99999
Merge Sort	0.043	933015
Quick Sort	1.777	4999950000
Heap Sort	0.047	2096696
Counting Sort	0.006	0
Radix Sort	0.029	0
Bucket Sort	0.019	0

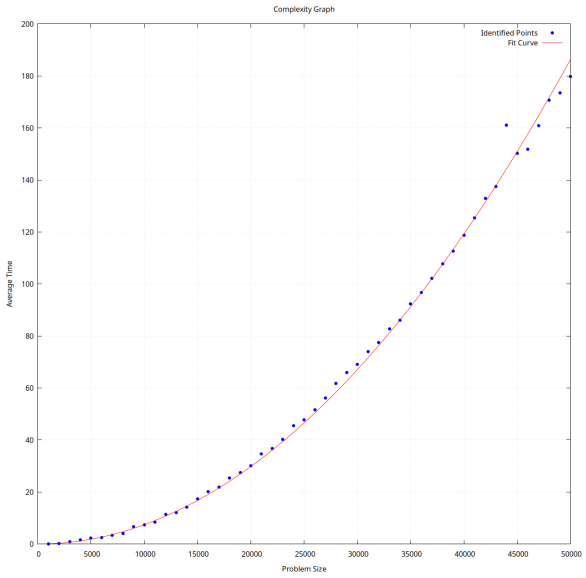
Fonte: Autoria própria (2025).

Figura 35 – Bubble Sort sobre entrada ordenada (100 pontos).



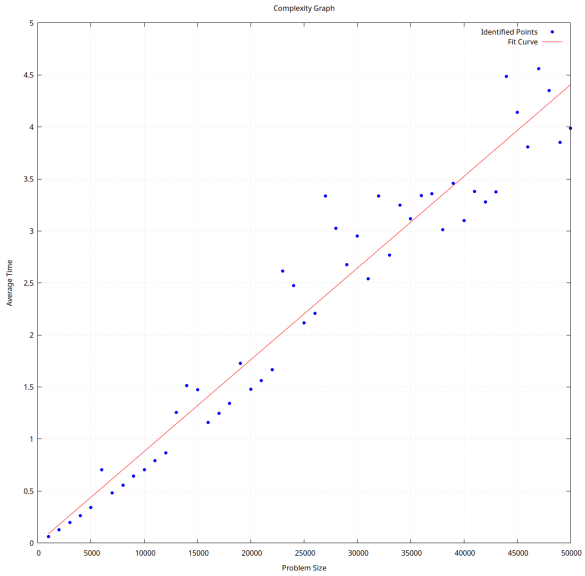
Fonte: Autoria própria (2025).

Figura 36 – Insertion Sort sobre entrada ordenada (100 pontos).



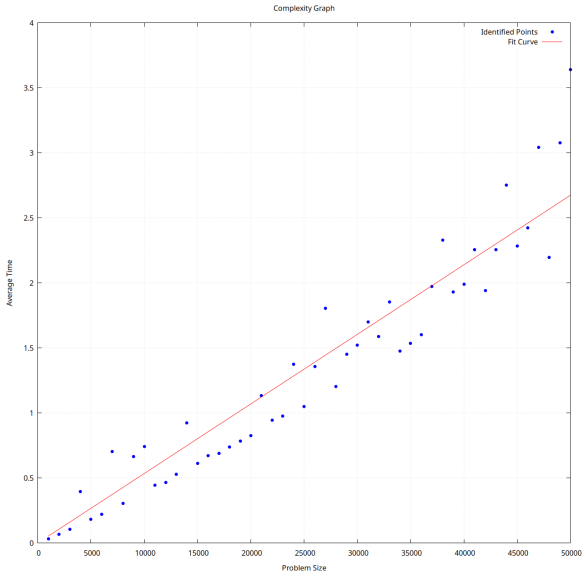
Fonte: Autoria própria (2025).

Figura 37 – Merge Sort sobre entrada ordenada (100 pontos).



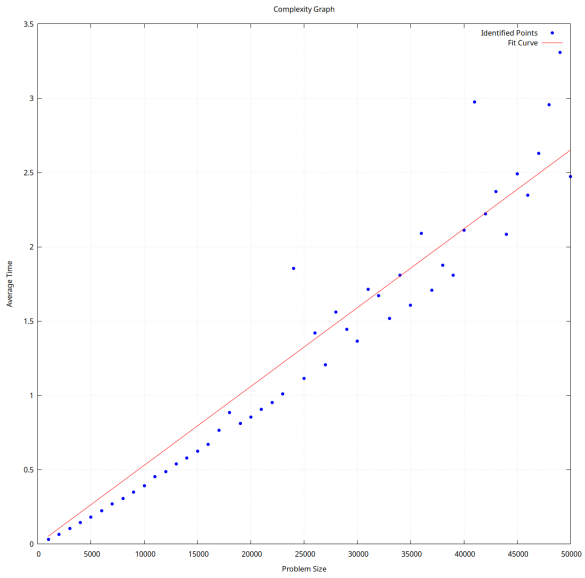
Fonte: Autoria própria (2025).

Figura 38 – Quick Sort sobre entrada ordenada (100 pontos).



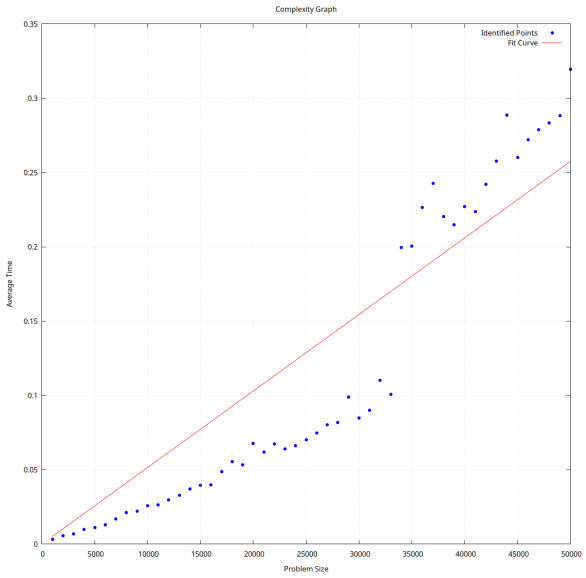
Fonte: Autoria própria (2025).

Figura 39 – Heap Sort sobre entrada ordenada (100 pontos).



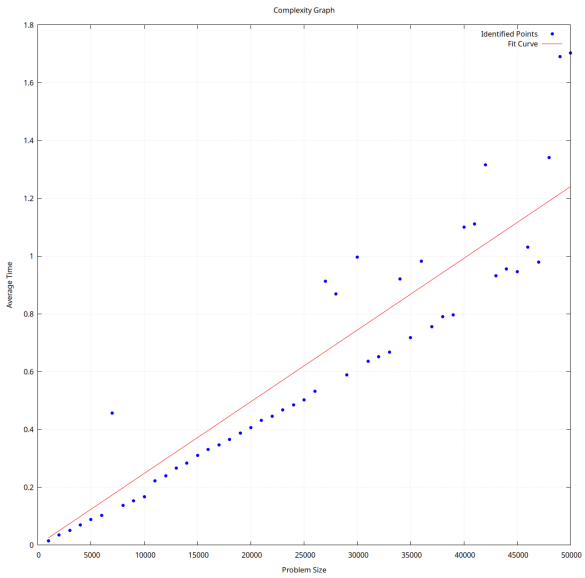
Fonte: Autoria própria (2025).

Figura 40 – Counting Sort sobre entrada ordenada (100 pontos).



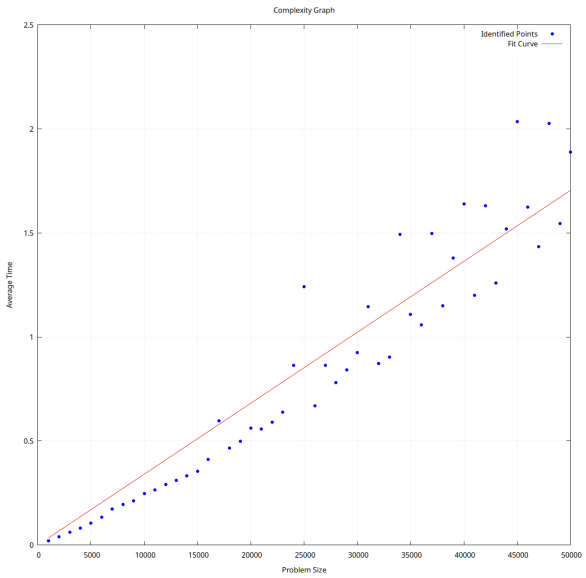
Fonte: Autoria própria (2025).

Figura 41 – Radix Sort sobre entrada ordenada (100 pontos).



Fonte: Autoria própria (2025).

Figura 42 – Bucket Sort sobre entrada ordenada (100 pontos).



Fonte: Autoria própria (2025).

Interpretação dos gráficos.

Bubble Sort (35): O ajuste $O(n^2)$ (RMSE = 44.89) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto ordenado com 100 pontos. **Insertion Sort (36)**: O tempo de execução muito inferior ao caso aleatório e o número reduzido de comparações

confirmam o melhor caso do algoritmo, que se aproxima de $O(n)$ quando os dados já estão ordenados. **Merge Sort (37)**: O ajuste $O(n \log n)$ (RMSE = 0.31) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 100 pontos. **Quick Sort (38)**: O elevado número de comparações e o aumento significativo no tempo de execução indicam que a entrada ordenada representa um cenário desfavorável para o Quick Sort, aproximando seu comportamento do pior caso quadrático. **Heap Sort (39)**: O ajuste $O(n \log n)$ (RMSE = 0.23) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 100 pontos. **Counting Sort (40)**: O ajuste $O(n)$ (RMSE = 0.04) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 100 pontos. **Radix Sort (41)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 100 pontos. **Bucket Sort (42)**: O ajuste $O(n)$ (RMSE = 0.18) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 100 pontos.

Interpretação dos gráficos.

Bubble Sort (35): O ajuste $O(n^2)$ (RMSE = 44.89) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto ordenado com 100 pontos. **Insertion Sort (36)**: O tempo de execução muito inferior ao caso aleatório e o número reduzido de comparações confirmam o melhor caso do algoritmo, que se aproxima de $O(n)$ quando os dados já estão ordenados. **Merge Sort (37)**: O ajuste $O(n \log n)$ (RMSE = 0.31) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 100 pontos. **Quick Sort (38)**: O elevado número de comparações e o aumento significativo no tempo de execução indicam que a entrada ordenada representa um cenário desfavorável para o Quick Sort, aproximando seu comportamento do pior caso quadrático. **Heap Sort (39)**: O ajuste $O(n \log n)$ (RMSE = 0.23) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto ordenado com 100 pontos. **Counting Sort (40)**: O ajuste $O(n)$ (RMSE = 0.04) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 100 pontos. **Radix Sort (41)**: O ajuste $O(n)$ (RMSE = 0.16) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 100 pontos. **Bucket Sort (42)**: O ajuste $O(n)$ (RMSE = 0.18) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto ordenado com 100 pontos.

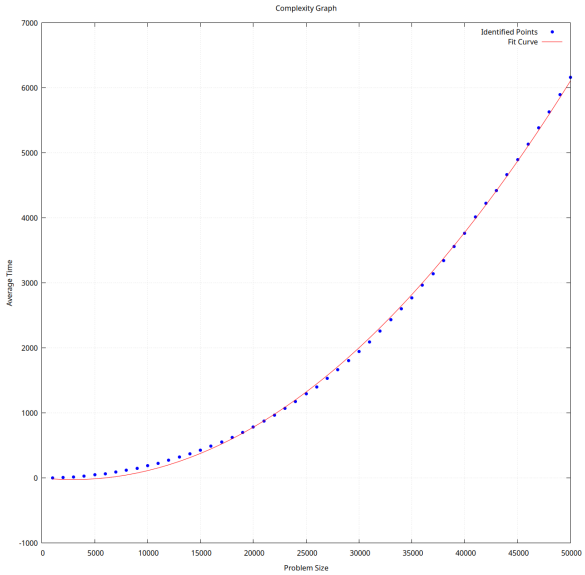
4.2.3 Dataset Reverso (p100)

Tabela 6 – Resultados para dataset reverso com 100 pontos.

Algoritmo	**Tempo (s)**	**Comparações**
Bubble Sort	84.851	4999950000
Insertion Sort	41.678	4999950000
Merge Sort	0.043	933016
Quick Sort	1.838	4999950000
Heap Sort	0.048	2079933
Counting Sort	0.006	0
Radix Sort	0.030	0
Bucket Sort	0.021	0

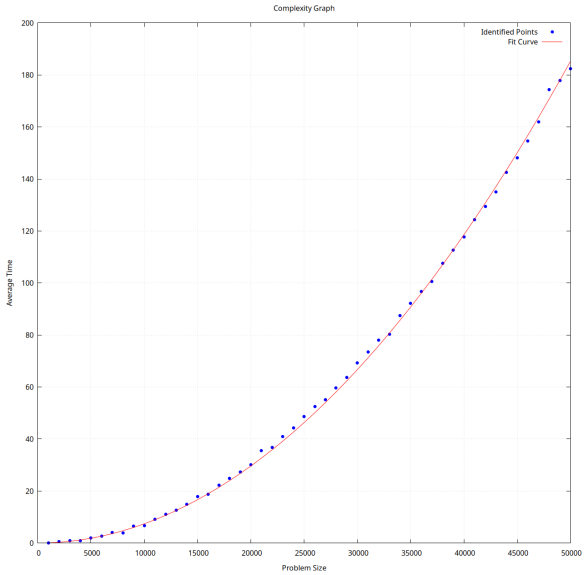
Fonte: Autoria própria (2025).

Figura 43 – Bubble Sort com entrada reversa (100 pontos).



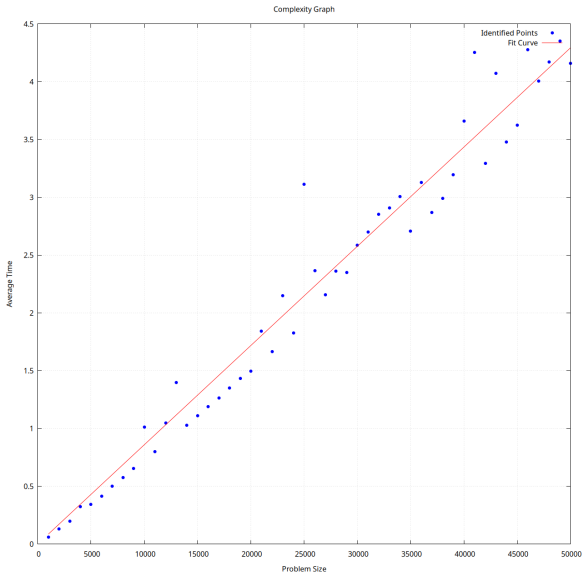
Fonte: Autoria própria (2025).

Figura 44 – Insertion Sort com entrada reversa (100 pontos).



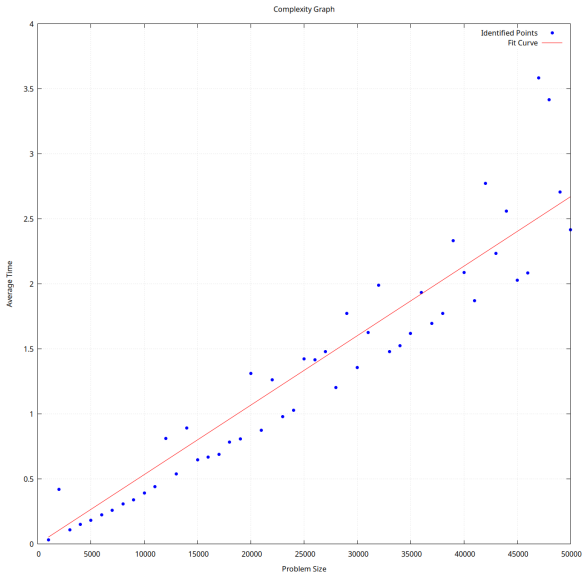
Fonte: Autoria própria (2025).

Figura 45 – Merge Sort com entrada reversa (100 pontos).



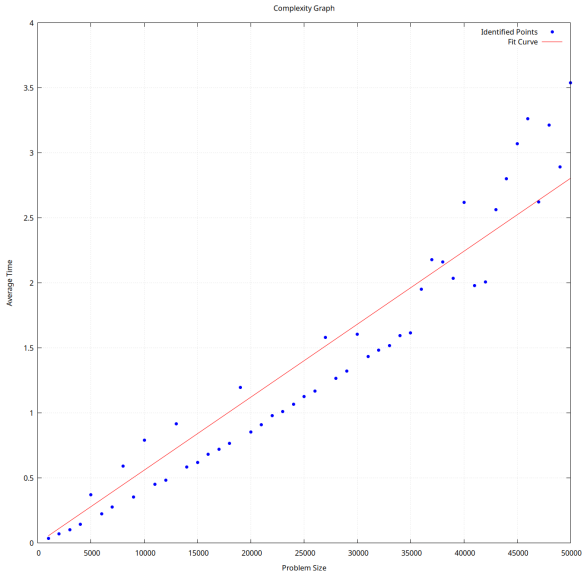
Fonte: Autoria própria (2025).

Figura 46 – Quick Sort com entrada reversa (100 pontos).



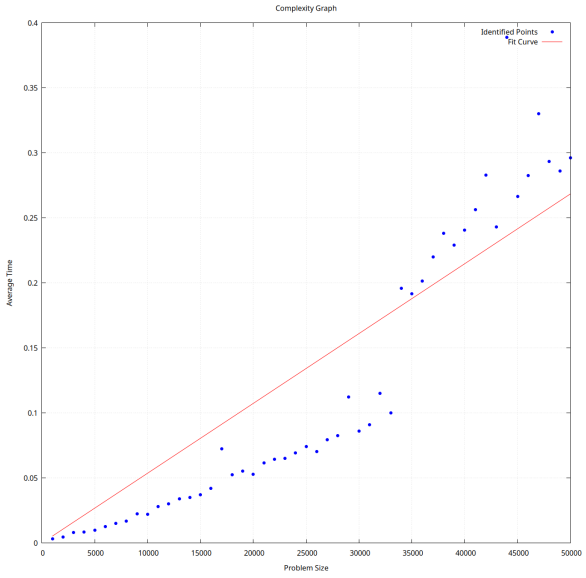
Fonte: Autoria própria (2025).

Figura 47 – Heap Sort com entrada reversa (100 pontos).



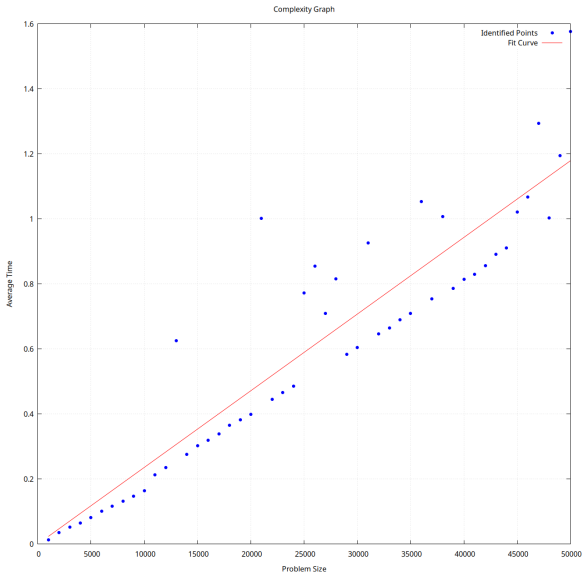
Fonte: Autoria própria (2025).

Figura 48 – Counting Sort com entrada reversa (100 pontos).



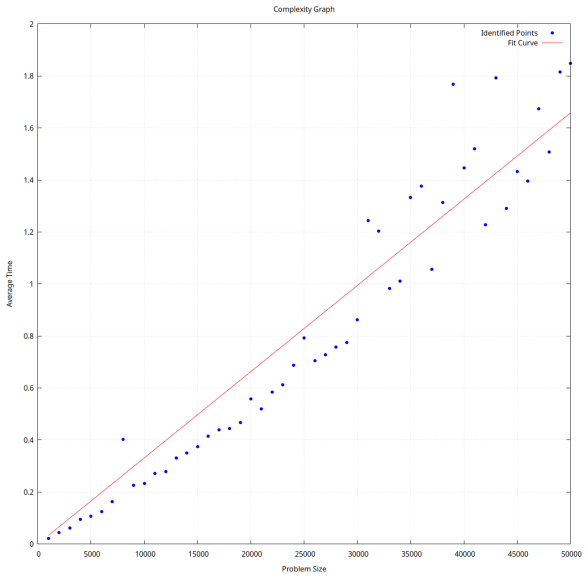
Fonte: Autoria própria (2025).

Figura 49 – Radix Sort com entrada reversa (100 pontos).



Fonte: Autoria própria (2025).

Figura 50 – Bucket Sort com entrada reversa (100 pontos).



Fonte: Autoria própria (2025).

Interpretação dos gráficos.

Bubble Sort (43): O ajuste $O(n^2)$ (RMSE = 45.10) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 100 pontos. A ordem inversa expõe o pior caso do algoritmo, com o maior tempo entre as abordagens comparadas. **Insertion Sort (44)**: O ajuste $O(n^2)$ (RMSE = 1.42) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 100 pontos. A ordem inversa impõe trocas em todas as iterações, justificando o crescimento quadrático observado empiricamente. **Merge Sort (45)**: O ajuste $O(n \log n)$ (RMSE = 0.25) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 100 pontos. **Quick Sort (46)**: O ajuste $O(n \log n)$ (RMSE = 0.29) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 100 pontos. **Heap Sort (47)**: O ajuste $O(n \log n)$ (RMSE = 0.28) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 100 pontos. **Counting Sort (48)**: O ajuste $O(n)$ (RMSE = 0.05) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 100 pontos. **Radix Sort (49)**: O ajuste $O(n)$ (RMSE = 0.14) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 100 pontos. **Bucket Sort (50)**: O ajuste $O(n)$ (RMSE = 0.15) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto

reverso com 100 pontos.

Interpretação dos gráficos.

Bubble Sort (43): O ajuste $O(n^2)$ (RMSE = 45.10) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 100 pontos. A ordem inversa expõe o pior caso do algoritmo, com o maior tempo entre as abordagens comparadas. **Insertion Sort (44)**: O ajuste $O(n^2)$ (RMSE = 1.42) reforça o crescimento quadrático característico de algoritmos com laços aninhados no conjunto reverso com 100 pontos. A ordem inversa impõe trocas em todas as iterações, justificando o crescimento quadrático observado empiricamente. **Merge Sort (45)**: O ajuste $O(n \log n)$ (RMSE = 0.25) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 100 pontos. **Quick Sort (46)**: O ajuste $O(n \log n)$ (RMSE = 0.29) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 100 pontos. **Heap Sort (47)**: O ajuste $O(n \log n)$ (RMSE = 0.28) confirma o comportamento de divisão e conquista esperado para algoritmos recursivos equilibrados no conjunto reverso com 100 pontos. **Counting Sort (48)**: O ajuste $O(n)$ (RMSE = 0.05) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 100 pontos. **Radix Sort (49)**: O ajuste $O(n)$ (RMSE = 0.14) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 100 pontos. **Bucket Sort (50)**: O ajuste $O(n)$ (RMSE = 0.15) evidencia a resposta linear típica de algoritmos baseados em contagem direta no conjunto reverso com 100 pontos.

4.3 Discussão dos Resultados

Os resultados experimentais confirmaram, em grande parte, as complexidades teóricas esperadas para os algoritmos de ordenação avaliados. O sistema testou três modelos fixos (Linear, N log N e Quadrático) e comparou o RMSE e SSE para identificar o melhor ajuste. Nos testes com 100 mil pontos e entradas aleatórias, os algoritmos *Merge Sort*, *Quick Sort* e *Heap Sort* ajustaram-se bem ao modelo $O(n \log n)$, com valores de RMSE baixos, confirmando a complexidade teórica esperada. Os algoritmos *Counting Sort*, *Radix Sort* e *Bucket Sort* apresentaram comportamento linear ($O(n)$), validando sua eficiência para os casos testados. Por outro lado, os algoritmos de complexidade quadrática, *Bubble Sort* e *Insertion Sort*, apresentaram maior variação nos ajustes. O *Bubble Sort* destacou-se com valores de RMSE extremamente altos em todos os cenários, indicando maior dispersão dos dados e um ajuste menos preciso. Isso

sugere que o comportamento empírico do Bubble Sort é altamente sensível a variações nos dados de entrada e pode apresentar constantes ocultas significativas. O *Insertion Sort* apresentou um comportamento notável: enquanto seu pior caso (dataset reverso) e caso médio (dataset aleatório) se alinharam com a complexidade $O(n^2)$, seu melhor caso (dataset ordenado) demonstrou um desempenho praticamente linear, com tempos de execução drasticamente menores, o que está de acordo com sua teoria. De forma similar, o *Quick Sort* demonstrou sua sensibilidade a dados ordenados, onde seu desempenho degradou significativamente, aproximando-se de um comportamento quadrático, o que também é esperado teoricamente. Os valores baixos de RMSE para algoritmos $O(n \log n)$ e $O(n)$ indicam que o Gradiente Descendente Adaptativo foi eficiente para ajustar essas curvas aos dados experimentais. A metodologia de seleção do modelo com menor erro provou ser eficaz para identificar a classe de complexidade empírica que melhor descreve o comportamento de cada algoritmo.

5 CONCLUSÃO

Este trabalho apresentou uma metodologia automatizada para analisar empiricamente a complexidade de algoritmos de ordenação, ajustando modelos teóricos com gradiente descendente adaptativo e priorizando o menor RMSE/SSE. Os resultados confirmaram a hierarquia $O(n) < O(n \log n) < O(n^2)$, evidenciaram desvios como o RMSE elevado do Bubble Sort e mostraram que casos especiais (melhor caso do Insertion Sort e pior caso do Quick Sort em dados ordenados) seguem o previsto pela teoria. A abordagem reforça a importância de alinhar análise teórica e observação prática: os gráficos condensados em pares facilitam a comparação direta, o ajuste numérico dispensa sistemas de equações ou técnicas de ML avançadas, e o protocolo de validação permite replicar e auditar cada experimento. Isso mantém o foco na interpretação dos dados e na escolha de algoritmos mais adequados para diferentes cenários. Como trabalhos futuros, recomenda-se aprofundar a análise de complexidade nos cenários de pior, melhor e caso médio específicos de cada algoritmo, expandir o conjunto de entradas (tamanhos maiores e distribuições distintas), incluir custos de memória e aplicar a metodologia a outras classes (busca, grafos, estruturas de dados), mantendo a instrumentação que captura o comportamento real em contraste com as previsões assintóticas.

REFERÊNCIAS

ALMEIDA, R. N. d. **O método dos mínimos quadrados: estudo e aplicações**. s.d. Disponível em: https://uenf.br/posgraduacao/matematica/wp-content/uploads/sites/14/2017/09/28052015Renato-Neves-de-Almeida.pdf. Acesso em: 30 jun. 2025.

ALRIDHA, A. H.; ALSHARIFY, F. H. A.; AL-KHAFAJI, Z. A review of optimization techniques: applications and comparative analysis. **Iraqi Journal for Computer Science and Mathematics**, v. 5, n. 2, 2024. Disponível em: https://ijcsm.researchcommons.org/ijcsm/vol5/iss2/5.

ARIZE. **Root mean square error (RMSE) in AI: what you need to know**. 2023. Disponível em: https://arize.com/blog-course/root-mean-square-error-rmse-what-you-need-to-know/. Acesso em: 8 jul. 2025.

ARORA, S.; BARAK, B. **Computational complexity: a modern approach**. [S.l.]: Cambridge University Press, 2009.

BIMURAT, Z.; KIM, Y.; ISMAILOVA, R.; SAGINDYKOV, B. Methods of navigating algorithmic complexity: Big-oh and small-oh notations. **Journal of AITU**, v. 15, 2023. Disponível em: https://doi.org/10.37943/15DNLB5877.

BLONDEL, V. D.; TSITSIKLIS, J. N. A survey of computational complexity results in systems and control. **Automatica**, v. 36, n. 9, p. 1249–1274, 2000. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0005109800000509.

CHEIN, F. **Introdução aos modelos de regressão linear**. [S.l.]: Escola Nacional de Administração Pública (ENAP), 2019. Disponível em: https://repositorio.enap.gov.br/handle/1/4788. Acesso em: 30 jun. 2025.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. 3rd. ed. [S.l.]: MIT Press, 2009.

CRUZ, R. C. d. Análise empírica sobre a influência das métricas ck na testabilidade de software orientado a objetos. **Dissertação de Mestrado, Universidade de São Paulo**, São Paulo, 2018.

FLAJOLET, P.; SEDGEWICK, R. **Introduction to the analysis of algorithms**. [S.l.]: Addison-Wesley, 1992.

FOLARANMI, R. O.; AYEPEKU, F. O. **Algorithms and complexity analysis**. 2023. Lecture Notes. Disponível em: https://tau.edu.ng/assets/media/docs/algorithms-and-complexity-analysis-csc304_1716907183.pdf.

FROST, J. **Root mean square error (RMSE)**. 2020. Statistics By Jim. Disponível em: https://statisticsbyjim.com/regression/root-mean-square-error-rmse/. Acesso em: 8 jul. 2025.

GOLDBERG, L. A. An introduction to research in computational complexity theory. **St Edmund Hall Blog**, University of Oxford, June 2019. Research blog entry on computational complexity theory. Disponível em: https://www.seh.ox.ac.uk/blog/an-introduction-to-research-in-computational-complexity-theory.

HOGAN, S. **A gentle introduction to computational complexity theory, and a little bit more**. [S.l.], 2011. VIGRE REU Program. Disponível em: https://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/Hogan.pdf.

HOOS, H. H. **Introduction to empirical algorithmics**. 2003. (CPSC 590 (Autumn 2003)). Disponível em https://www.cs.ubc.ca/~van/cpsc590/cpsc590-emp-slides.pdf.

HOW to measure algorithm running time. 2023. Disponível em https://www.linkedin.com/advice/0/how-can-you-measure-actual-running-time-algorithm-using.

KOSOUROVA, E. **RMSE explicado: um guia para a precisão da previsão de regressão**. 2025. Disponível em: https://www.datacamp.com/pt/tutorial/rmse. Acesso em: 8 jul. 2025.

NGU, A. O. Dimensional complexity and algorithmic efficiency. **International Journal of Modern Nonlinear Theory and Application**, v. 11, n. 1, p. 1–10, 2022. Disponível em: https://www.scirp.org/journal/paperinformation?paperid=115691.

NOBREGA, C. S. B. **Uma estratégia para predição da taxa de aprendizagem do gradiente descendente para aceleração da fatoração de matrizes**. Dissertação (Mestrado) — Universidade Federal de Campina Grande, 2014. Disponível em: http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/362. Acesso em: 30 jun. 2025.

SAMSUDIN, N.; YUSOP, N. M. M.; MOKHTAR, A. S. N. b.; AMRAN, M. F. b.; ALI, I. M. Calculation on euler arithmetic complexity using big o. **International Journal of Advanced Trends in Computer Science and Engineering**, v. 9, n. 1.4, 2020. Disponível em: https://www.warse.org/IJATCSE/static/pdf/file/ijatcse65914sl2020.pdf.

SILVA, B. A. d.; PORTO, G. G.; KALILI, R. M. Gêmeos digitais aplicados à indústria automotiva: otimização e manutenção eficiente de veículos. **Revista Contemporânea**, 2024. Disponível em: https://ojs.revistacontemporanea.com/ojs/index.php/home/article/download/6872/4901/19997.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: principles and paradigms**. 3. ed. [S.l.]: Pearson, 2017.

ZONATTO, E. **Algoritmo de um controlador lógico Fuzzy do motor à combustão interna de um veículo convencional convertido para uma arquitetura elétrica híbrida**. Dissertação (Monografia de Especialização) — Universidade Tecnológica Federal do Paraná, Curitiba, 2018. Disponível em: https://repositorio.utfpr.edu.br/jspui/bitstream/1/19808/1/CT_CESEB_IV_2018_03.pdf.