UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CENTRO DE ENSINO SUPERIOR DO SERIDÓ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO.

JOSÉ GEAN DE MACÊDO ALVES
JÚLIA LILIAN PRUDENCIO DA COSTA

Análise Empírica e Comparativa de Desempenho: Arrays Estáticos vs. Listas Encadeadas

CAICÓ - RN

2025

JOSÉ GEAN DE MACÊDO ALVES
JÚLIA LILIAN PRUDENCIO DA COSTA

Análise Empírica e Comparativa de Desempenho: Arrays Estáticos vs. Listas Encadeadas:

Relatório acadêmico apresentado à disciplina de Estrutura de Dados (DCT0008) , do curso de Sistemas de Informação da Universidade Federal do Rio Grande do Norte (UFRN) , como requisito para a avaliação do Trabalho Complementar da Unidade II. Este trabalho tem como objetivo a análise prática e comparativa entre estruturas de dados lineares (Pilha, Fila e Lista), implementadas através de Arrays e Listas Encadeadas. A metodologia baseia-se na codificação de seis algoritmos aplicados a cenários reais, seguida de uma análise experimental dos tempos de execução e da estimativa da complexidade assintótica (Big-O) para operações de inserção, remoção e redimensionamento.

Orientador(a): Dr. Arthur Emanoel Cassio da Silva e Souza.

.

CAICÓ - RN

2025

RESUMO

Este trabalho tem como objetivo apresentar uma análise prática e teórica de algoritmos que utilizam estruturas como array e lista encadeada, conforme proposto na disciplina de Estrutura de Dados. Vetores, listas encadeadas, pilhas e filas são estruturas essenciais para a abordagem de diversos problemas de programação na Ciência da Computação, propiciando o desenvolvimento de algoritmos variados que auxiliam na resolução de problemas.

A metodologia deste trabalho consiste na implementação de algoritmos que utilizam as estruturas de pilha, fila e lista. Nesse sentido, deverão ser escolhidos exemplos de situações para cada estrutura, de modo que, para cada um, sejam implementados dois algoritmos: um utilizando Array e outro com lista ligada (LinkedList). Dessa forma, serão codificados seis (6) algoritmos no total: 2 para pilha, 2 para fila e 2 para lista. O objetivo é exercitar habilidades práticas de codificação, análise e utilização das estruturas citadas anteriormente, visando consolidar o aprendizado acerca das diferenças teóricas e práticas das estruturas envolvidas.

A análise empírica de cada algoritmo implementado foi realizada com base na medição de seu tempo de execução e em estimativas da complexidade Big-O: O(t(n)) para as funções de adição e remoção de elemento, aumento e diminuição de tamanho físico em cada um. Dessa forma, foram gerados gráficos comparativos acerca da complexidade estimada para cada função em cada algoritmo, para as maiores entradas possíveis.

Os resultados obtidos evidenciam as diferenças significativas entre as estruturas de dados utilizadas, ampliando a compreensão de suas utilizações em variados contextos e problemas, indicando que, para determinadas situações, a depender das características exigidas, algumas estruturas podem ser mais adequadas que outras, além de se compreender que os mesmos problemas podem ser resolvidos de formas diferentes e até inovadoras, justificando a relevância deste estudo para a graduação em Sistemas de Informação.

Palavras-chave: Estruturas de Dados; Análise e Comparação de Algoritmos; Array; Lista Ligada; Pilhas e Filas; Análise Experimental.

ABSTRACT

This work aims to present a practical and theoretical analysis of algorithms that use structures such as arrays and linked lists, as proposed in the Data Structures course. Vectors, linked lists, stacks, and queues are essential structures for approaching various programming problems in Computer Science, enabling the development of varied algorithms that assist in solving problems.

The methodology of this work consists of implementing algorithms that use stack, queue, and list structures. In this sense, examples of situations should be chosen for each structure, so that, for each one, two algorithms are implemented: one using an Array and the other with a linked list (LinkedList). In this way, six (6) algorithms will be coded in total: 2 for stacks, 2 for queues, and 2 for lists. The objective is to exercise practical skills in coding, analyzing, and using the structures mentioned above, aiming to consolidate learning about the theoretical and practical differences of the structures involved.

The empirical analysis of each implemented algorithm was performed based on measuring its execution time and estimating the Big-O complexity: O(t(n)) for the element addition and removal functions, and physical size increase and decrease in each one. In this way, comparative graphs were generated regarding the estimated complexity for each function in each algorithm, for the largest possible inputs.

The results obtained highlight the significant differences between the data structures used, broadening the understanding of their uses in various contexts and problems, indicating that, for certain situations, depending on the required characteristics, some structures may be more suitable than others, in addition to understanding that the same problems can be solved in different and even innovative ways, justifying the relevance of this study in the Information Systems undergraduate program.

Keywords: Data Structures; Algorithm Analysis and Comparison; Array; Linked List; Stacks and Queues; Experimental Analysis.

SUMÁRIO

1 INTRODUÇÃO

A seleção eficiente de estruturas de dados é um dos pilares fundamentais da ciência da computação, sendo determinante para a otimização de recursos e o tempo de resposta de sistemas de software. A forma como os dados são organizados e acessados na memória impacta diretamente o desempenho das aplicações, especialmente quando submetidas a grandes volumes de operações de manipulação.

Este trabalho apresenta uma análise teórica e prática das estruturas lineares Pilha, Fila e Lista, comparando suas implementações baseadas em Arrays (alocação contígua) e Listas Encadeadas (alocação dinâmica). Serão avaliados aspectos críticos como o tempo de execução por meio de uma análise experimental, utilizando algoritmos desenvolvidos em Python, e a complexidade assintótica das operações de inserção, remoção e redimensionamento, fundamentada na notação Big-O.

Adicionalmente, o estudo inclui a geração de gráficos para a visualização e comparação de desempenho, utilizando a biblioteca Matplotlib para ilustrar cenários práticos como editores de texto e filas de impressão. Com isso, pretende-se fornecer uma compreensão aprofundada das vantagens e limitações (trade-offs) entre o uso de memória estática e dinâmica, relacionando os resultados experimentais obtidos com a teoria da complexidade de dados.

2 FUNDAMENTAÇÃO TEÓRICA

As estruturas de dados são métodos de armazenamento e organização de dados no computador, projetadas para tornar o uso desses dados eficiente. A escolha da estrutura adequada impacta diretamente a complexidade de tempo (velocidade) e espaço (memória) dos algoritmos. Este trabalho fundamenta-se na análise de três Tipos Abstratos de Dados (TADs) lineares — Pilha, Fila e Lista — e suas implementações através de duas estratégias distintas de alocação de memória: contígua e encadeada.

2.1. Alocação Contígua (Arrays)

Os Arrays (ou vetores) são estruturas que armazenam elementos em posições de memória adjacentes. Essa característica permite o acesso aleatório aos dados em tempo constante O(1) através de índices. No entanto, Arrays possuem tamanho físico fixo.Para simular um comportamento dinâmico, quando o array atinge sua capacidade máxima, é necessário realizar uma operação de redimensionamento (resizing).

Esta operação exige a alocação de um novo bloco de memória maior e a cópia de todos os elementos anteriores, resultando em uma complexidade linear O(n). Além disso, inserções ou remoções no início ou meio do array exigem o deslocamento (shifting) dos elementos subsequentes, o que também custa O(n).

2.2. Alocação Encadeada (Listas Ligadas)

As Listas Ligadas utilizam alocação de memória não contígua. A estrutura base é o Nó, que encapsula o dado é uma referência (ponteiro) para o próximo elemento da sequência. Diferente dos Arrays, as Listas Ligadas não exigem redimensionamento físico nem deslocamento de elementos. A inserção ou remoção de itens nas extremidades (início ou fim, dependendo da implementação) ocorre através da simples manipulação de ponteiros, garantindo complexidade constante O(1). A desvantagem reside no acesso aos elementos, que é sequencial O(n), e no uso extra de memória para armazenar as referências

2.3. Estruturas Abstratas Implementadas

Para fins de análise comparativa, este trabalho implementa três TADs clássicos:

- **Pilha (Stack):** Uma coleção linear que segue o princípio LIFO (*Last-In, First-Out*), onde o último elemento inserido é o primeiro a ser removido. É a estrutura base para históricos de navegação e chamadas de função recursivas.
- **Fila (Queue):** Segue o princípio FIFO (*First-In, First-Out*). O primeiro elemento a entrar é o primeiro a sair. É amplamente utilizada em *buffers* de impressão e escalonamento de processos.
- **Lista (List):** Uma estrutura linear generalista que permite inserção, remoção e acesso em qualquer posição (índice) da sequência, sendo flexível para gerenciamento de itens não ordenados pela ordem de chegada.

3 METODOLOGIA

A metodologia adotada neste trabalho baseia-se em uma abordagem experimental e quantitativa, dividida em três etapas fundamentais: implementação das estruturas de dados, definição de cenários de aplicação prática e análise empírica de desempenho. O objetivo central foi comparar o comportamento computacional entre a alocação de memória contígua (Arrays) e a alocação dinâmica encadeada (Listas Ligadas).

3.1 Configuração do Ambiente

Para a realização dos experimentos e coleta das métricas de desempenho, foi utilizado um ambiente de desenvolvimento baseado na linguagem Python 3.9.0. A geração dos gráficos comparativos foi realizada através da biblioteca Matplotlib, enquanto a medição precisa do tempo de execução (em nanosegundos) utilizou o módulo nativo time (perf_counter_ns).

Os testes foram executados em um único ambiente de hardware para garantir a consistência dos resultados e minimizar variações externas. A configuração do computador utilizado foi a seguinte:

- Processador: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz (2 núcleos, 4 processadores lógicos)
- Memória RAM: 16 GB
- Sistema Operacional: Windows
- Armazenamento: SSD

4 RESULTADOS

Este capítulo apresenta os dados obtidos através da análise empírica das estruturas implementadas. Os resultados estão organizados em duas etapas: primeiramente, a avaliação do desempenho individual de cada estrutura (Array e Lista Encadeada) para identificar o comportamento de suas operações críticas. Em seguida, são apresentados os gráficos comparativos que confrontam diretamente as duas abordagens.

Os testes focaram nas operações que evidenciam as diferenças teóricas de complexidade: o redimensionamento dinâmico (resize) e o deslocamento de elementos (shift) nos Arrays, contrastando com a alocação de nós nas Listas Encadeadas.

4.1 Gráficos Pilha (Histórico Web) usando Array

4.1.1 Gráfico Pilha Usando Array - Inserir

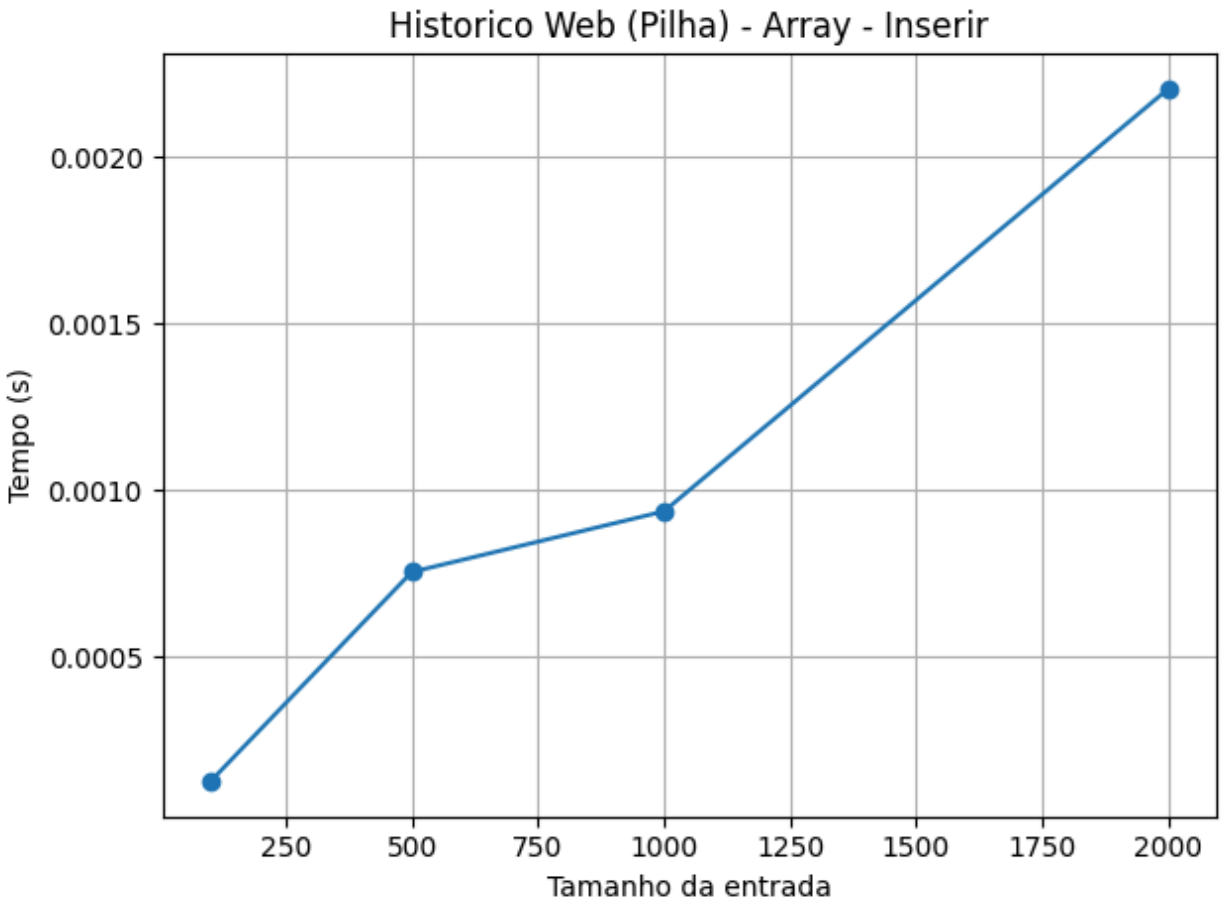

FIGURA 1: Gráfico PIlha Usando Array - Inserir

4.1.2 Gráfico Pilha Usando Array - Remover

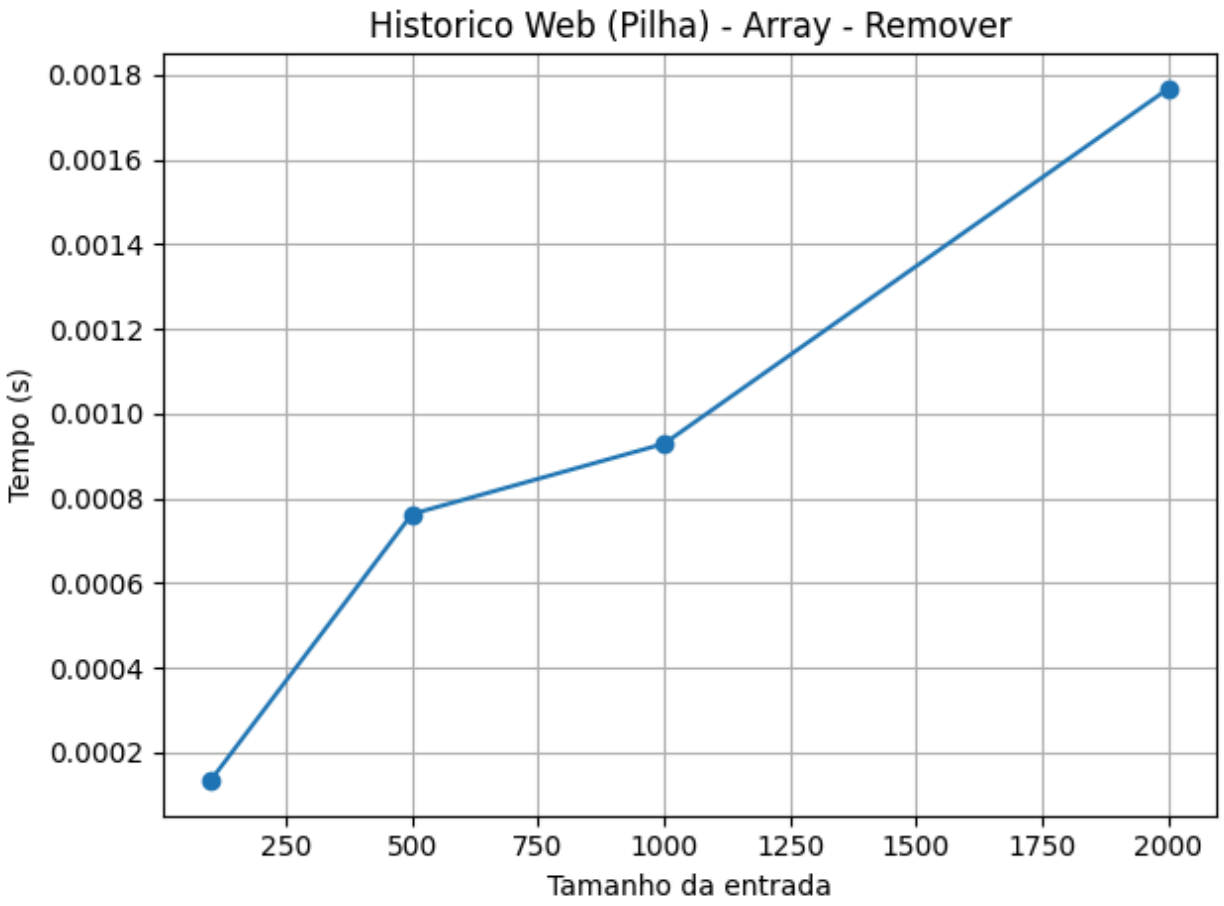

FIGURA 2: Gráfico Pilha Usando Array - Remover

4.2 Gráfico Pilha (Histórico Web) Usando Lista

4.2.1 Gráfico Pilha Usando Lista - Inserir

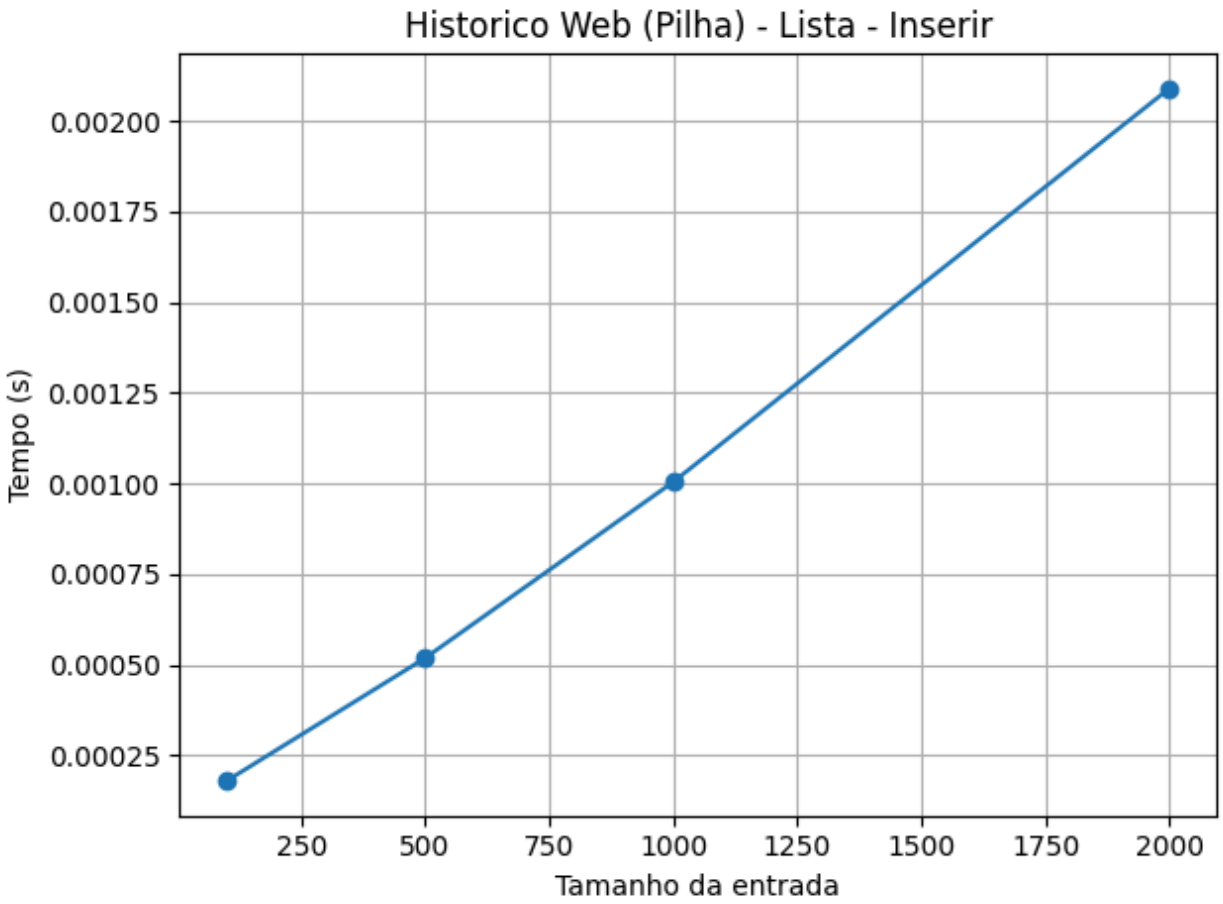

FIGURA 3: Gráfico Pilha Usando Lista - Inserir

4.2.1 Gráfico Pilha Usando Lista - Remover

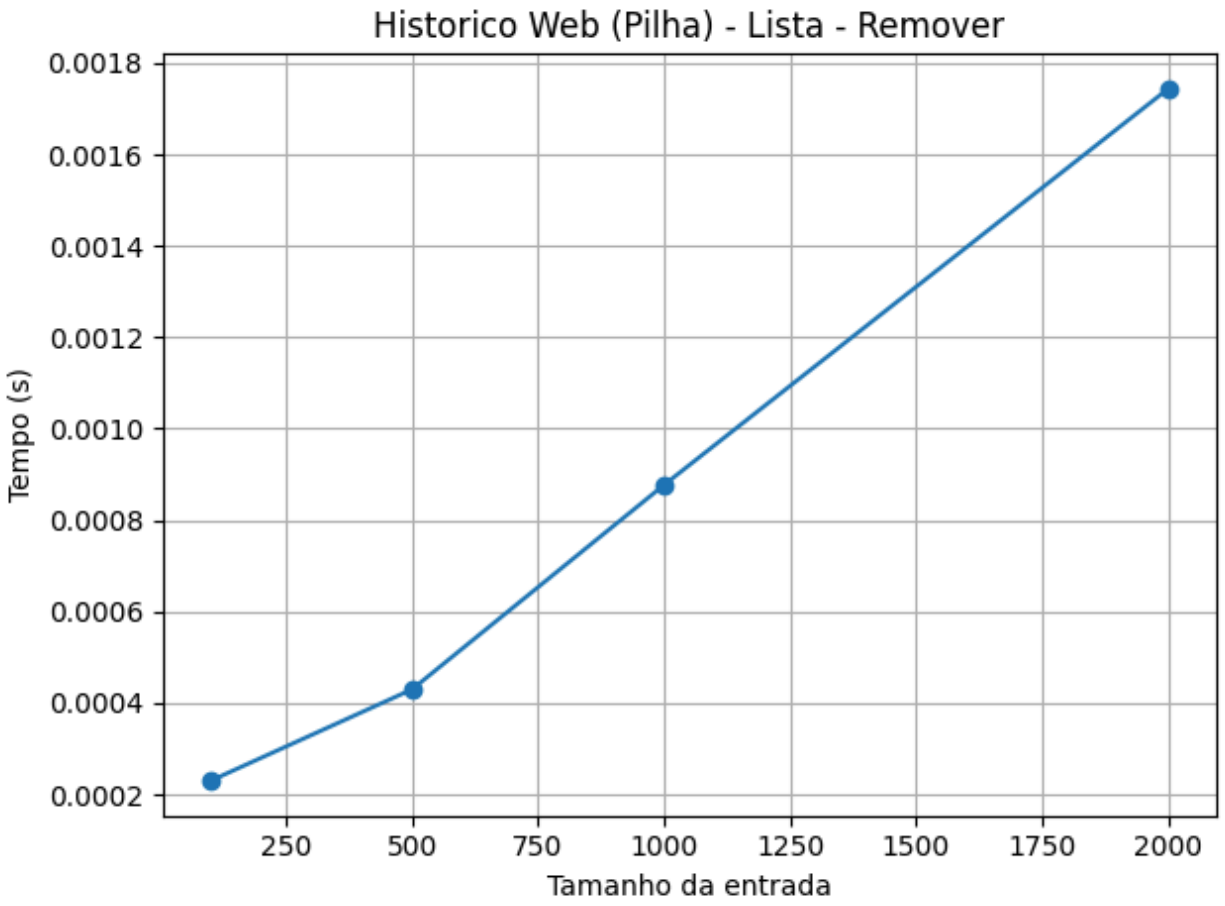

FIGURA 4: Gráfico Pilha Usando Lista - Remover

4.3 Gráficos Pilha (Histórico Web) Comparando Array Vs Lista

4.3.1 Gráfico Comparativo - Inserir

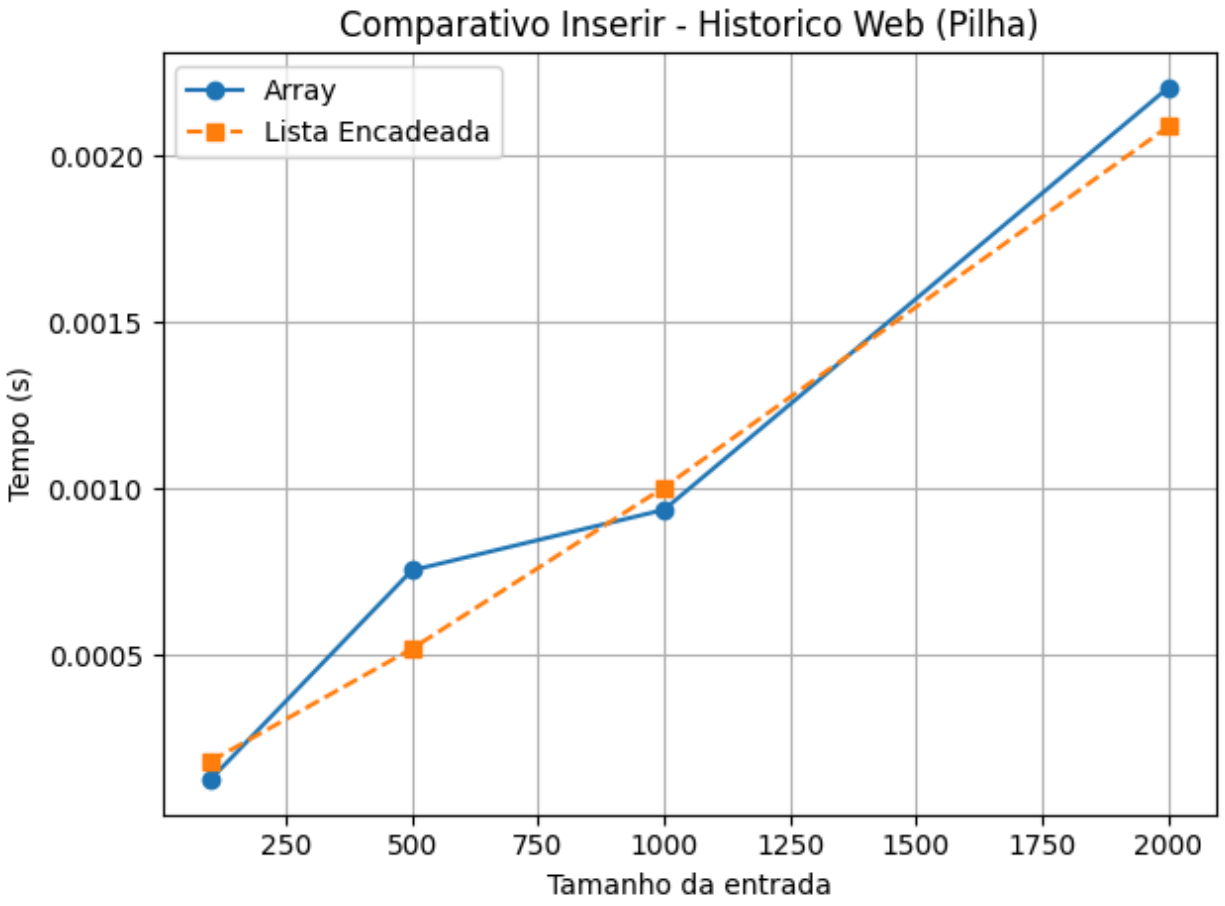

FIGURA 5: Gráfico Comparativo - Inserir

4.3.2 Gráfico Comparativo - Remover

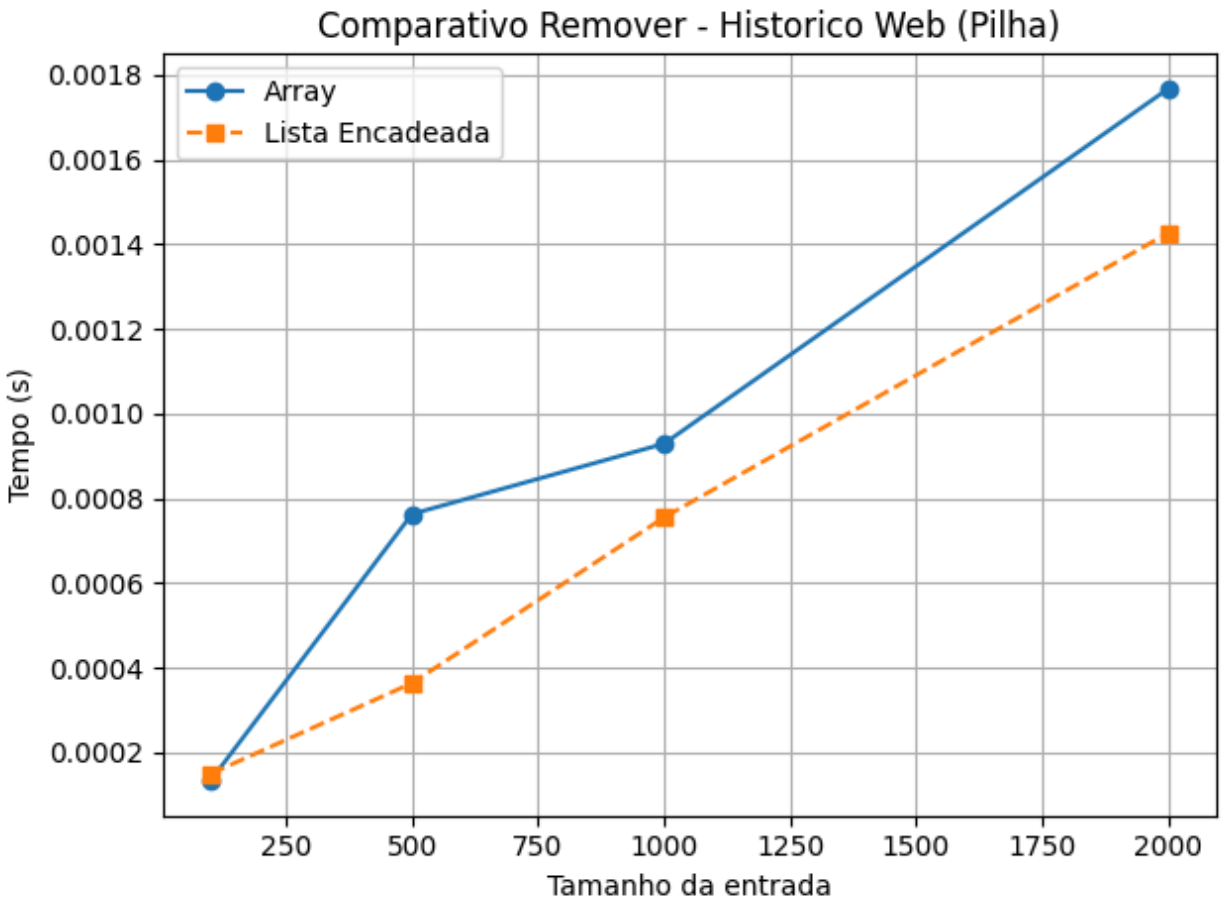

FIGURA 6: Gráfico Comparativo - Remover

4.3.3 Gráfico Comparativo - Aumentar

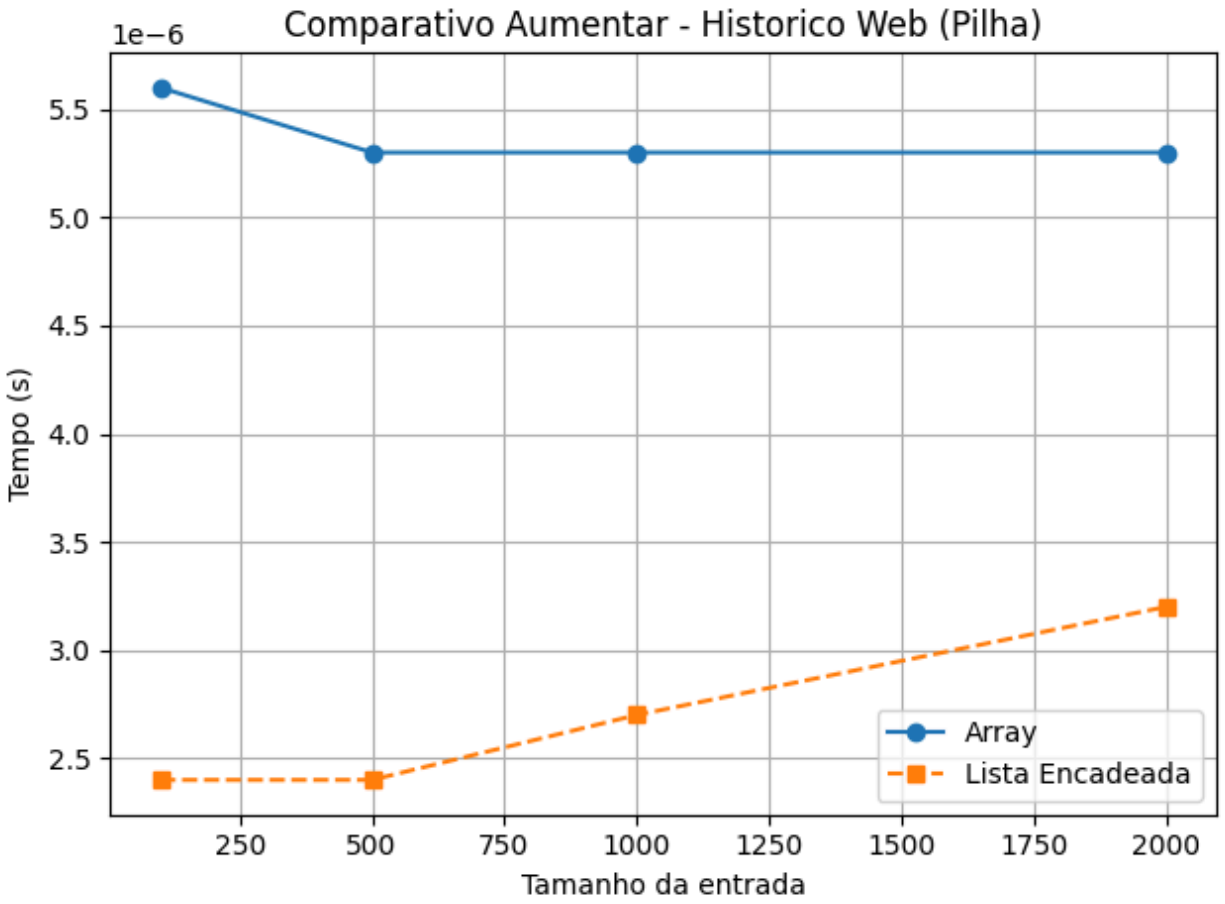

FIGURA 7: Gráfico Comparativo - Aumentar

4.4 Gráficos Fila(Pedidos de Compras) usando Array

4.4.1 Gráfico Fila Usando Array - Inserir

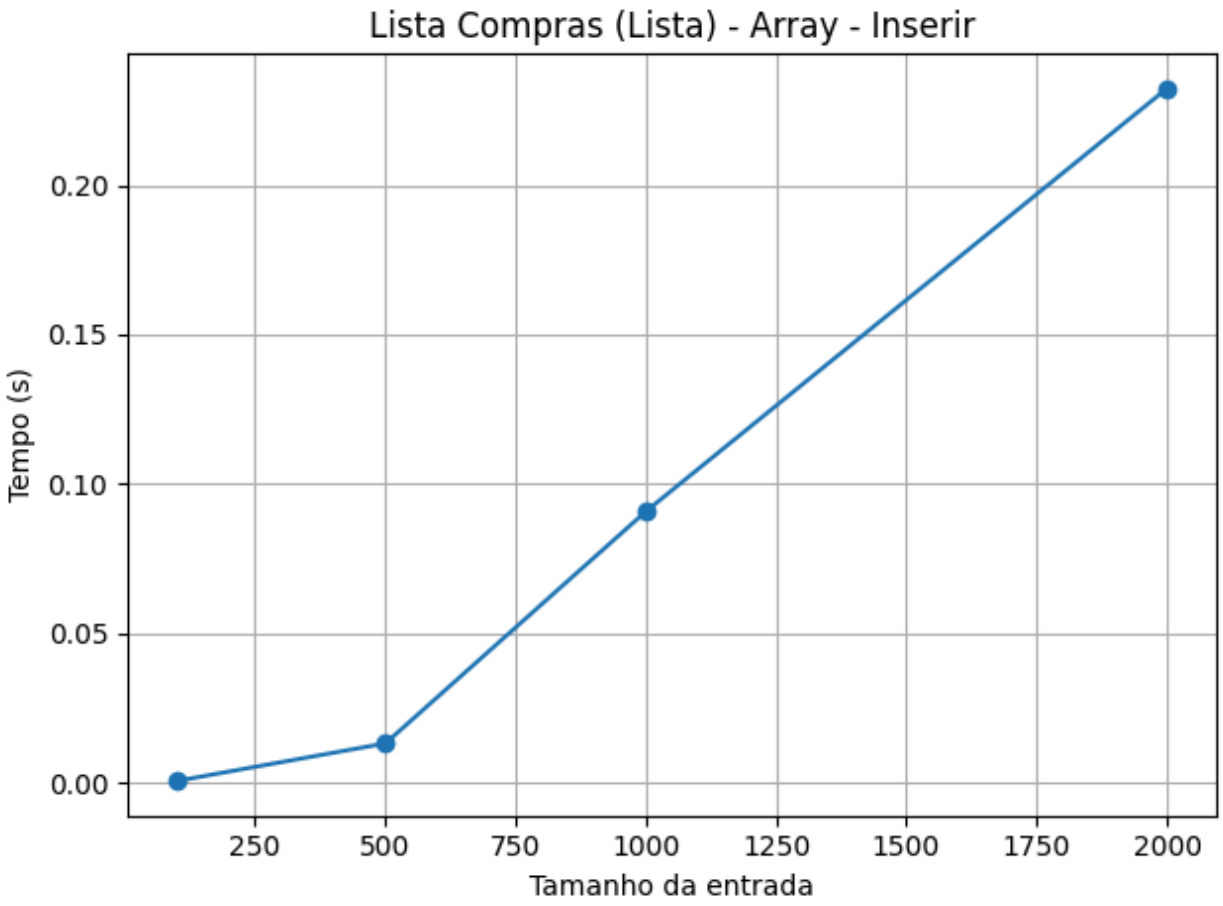

FIGURA 8: Gráfico Fila Usando Array - Inserir

4.4.2 Gráfico Fila Usando Array - Remover

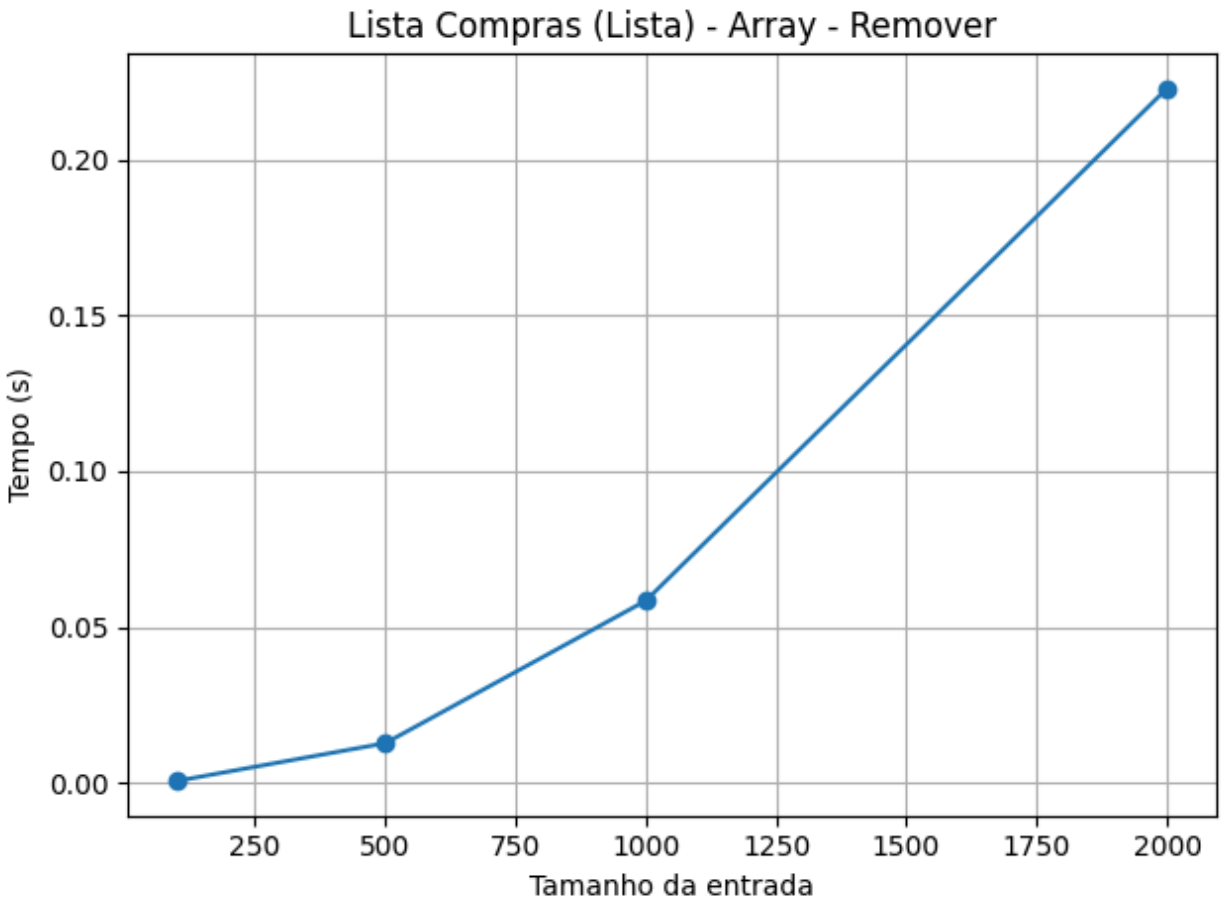

FIGURA 9: Gráfico Fila Usando Array - Remover

4.5 Gráfico Fila (Pedidos de Compras) Usando Lista

4.5.1 Gráfico Fila Usando Lista - Inserir

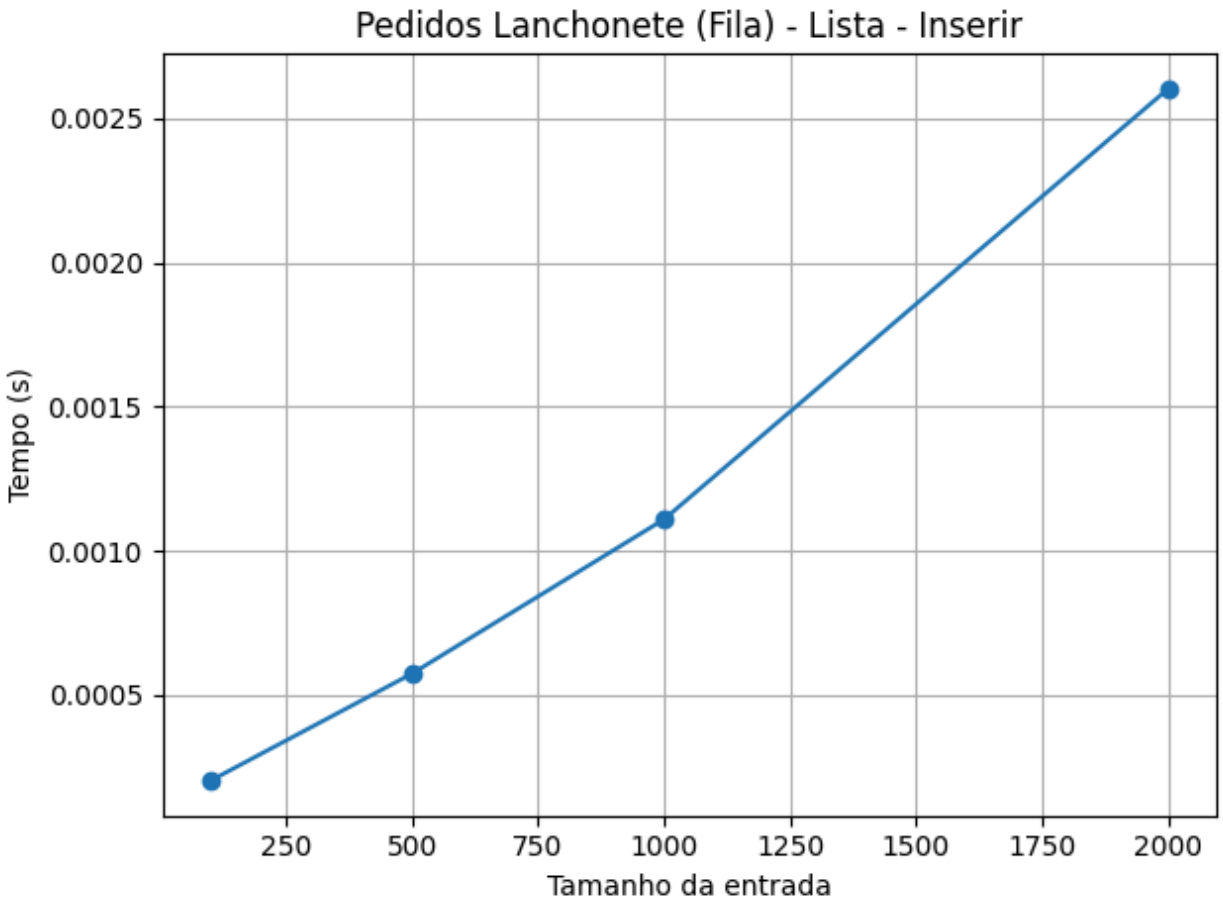

FIGURA 10: Gráfico Fila Usando Lista - Inserir

4.5.2 Gráfico Fila Usando Lista - Remover

Pq tá apagando?
tem que colocar kskksk

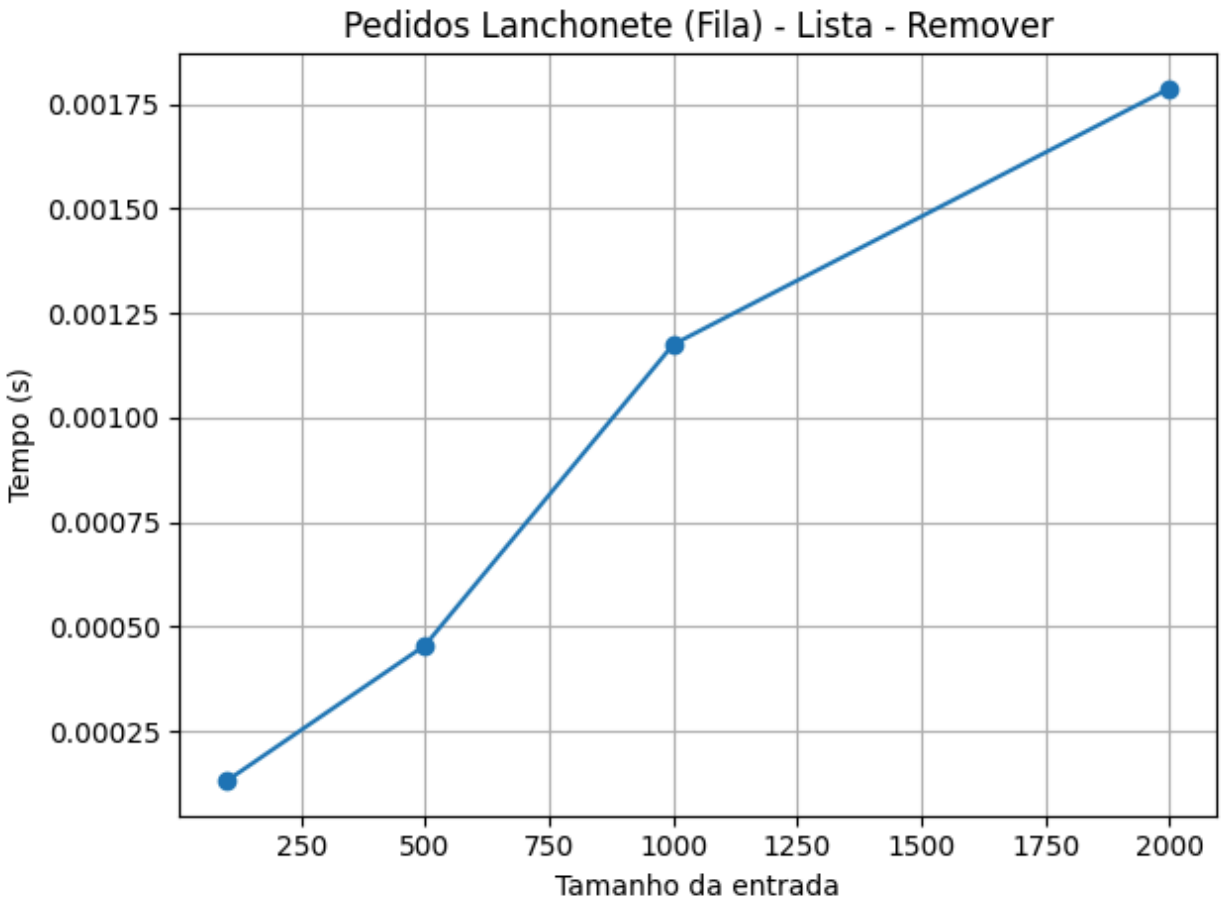

FIGURA 11: Gráfico Fila Usando Lista - Remover

4.6 Gráficos Fila (Pedidos de Compras) Comparando Array Vs Lista

4.6.1 Gráfico Comparativo - Inserir

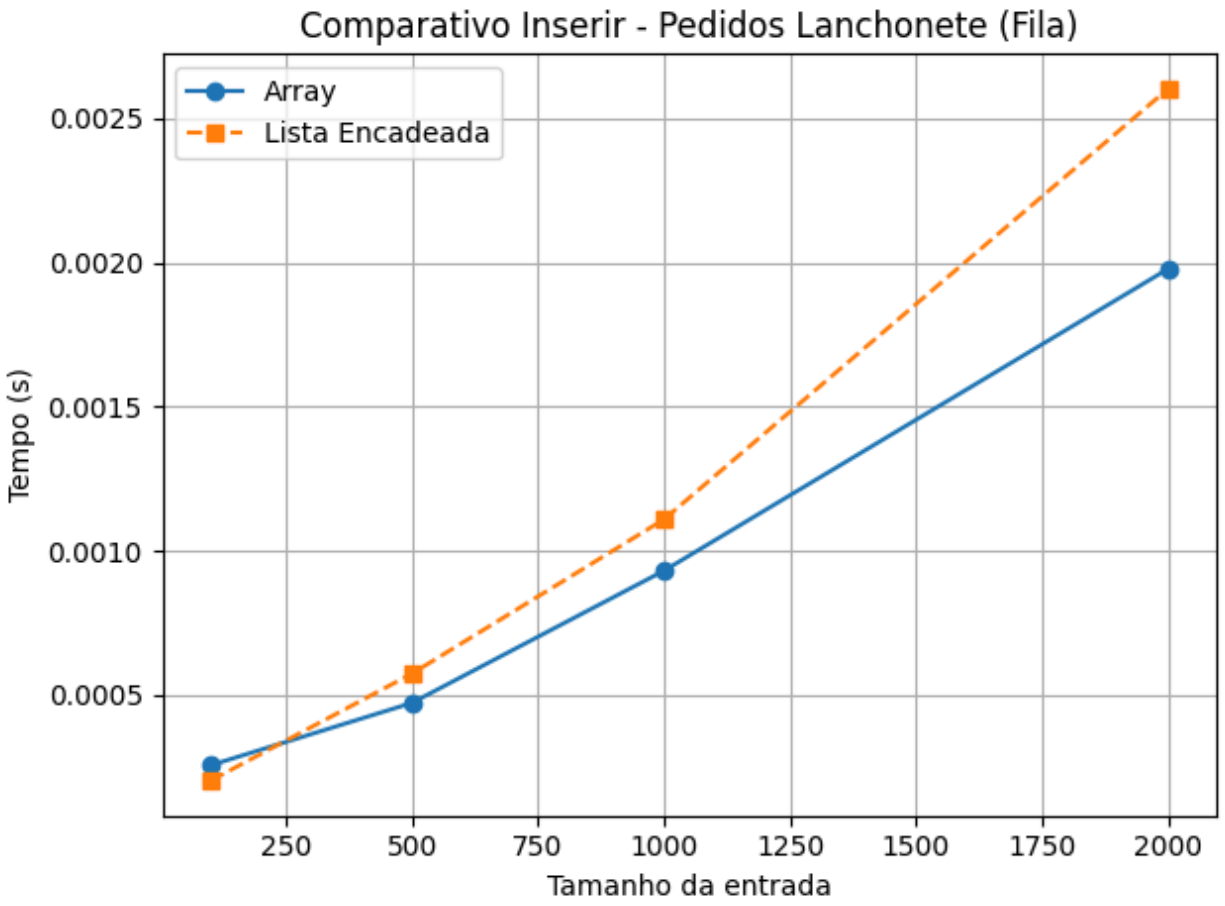

FIGURA 12: Gráfico Comparativo - Inserir

4.6.2 Gráfico Comparativo - Remover

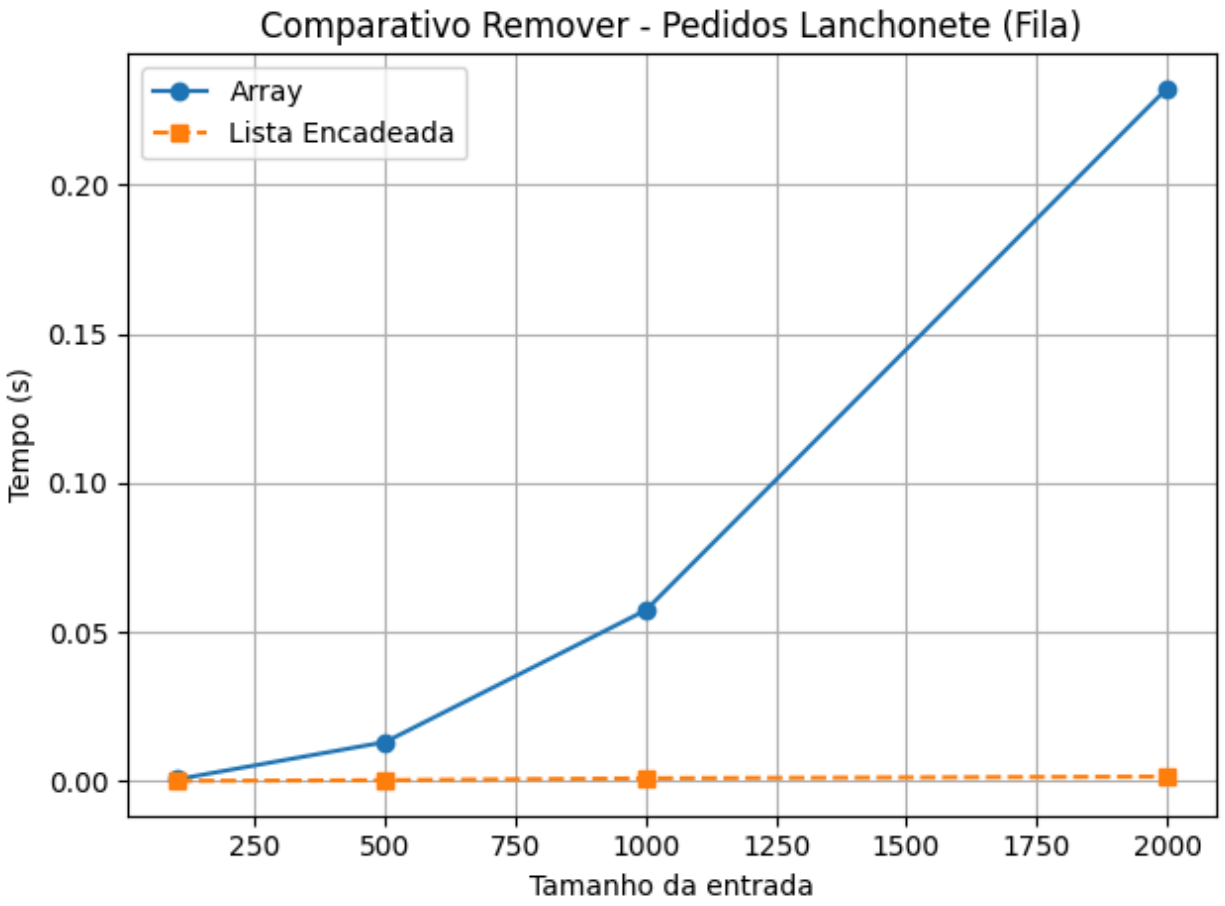

FIGURA 13: Gráfico Comparativo - Remover

4.6.3 Gráfico Comparativo - Aumentar

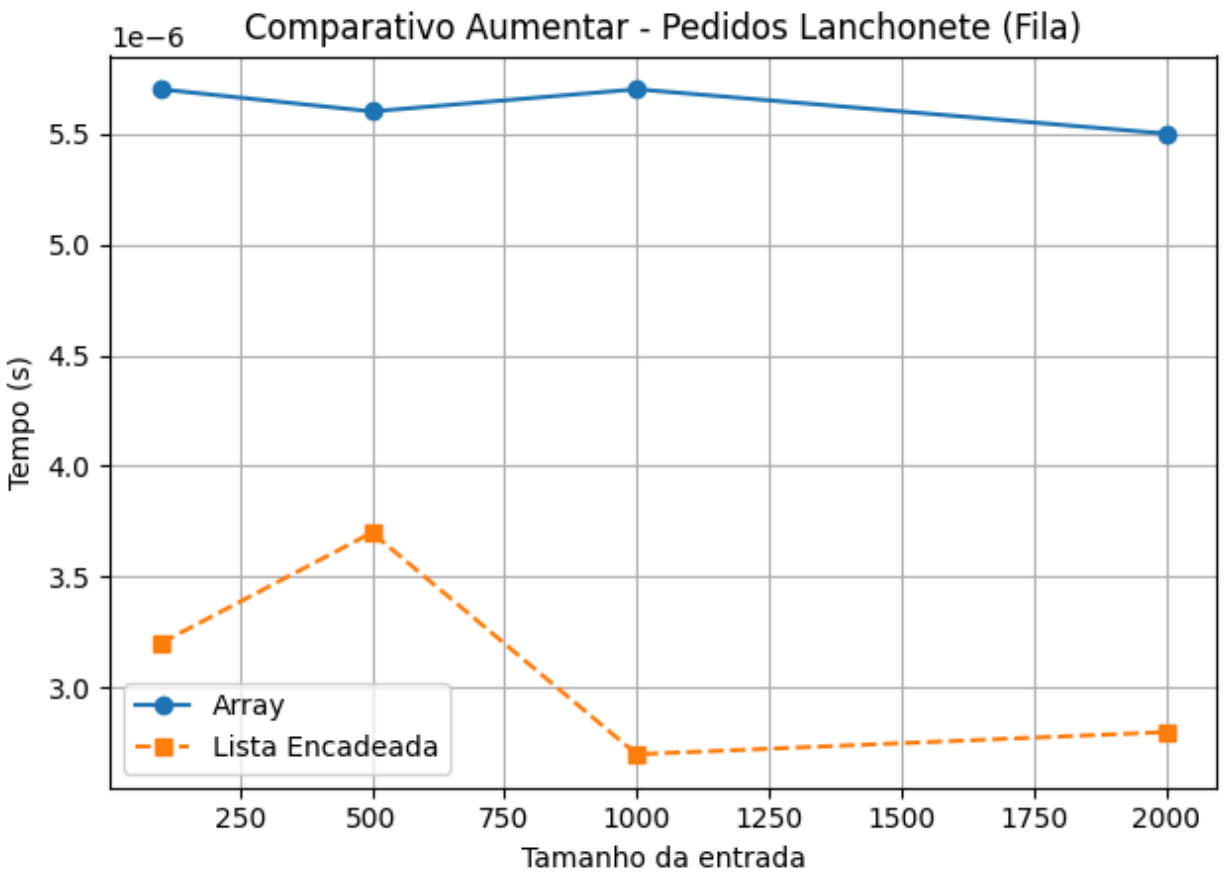

FIGURA 14: Gráfico Comparativo - Aumentar

4.7 Gráficos Lista (Lista de Compras) usando Array

4.7.1 Gráfico Lista Usando Array - Inserir

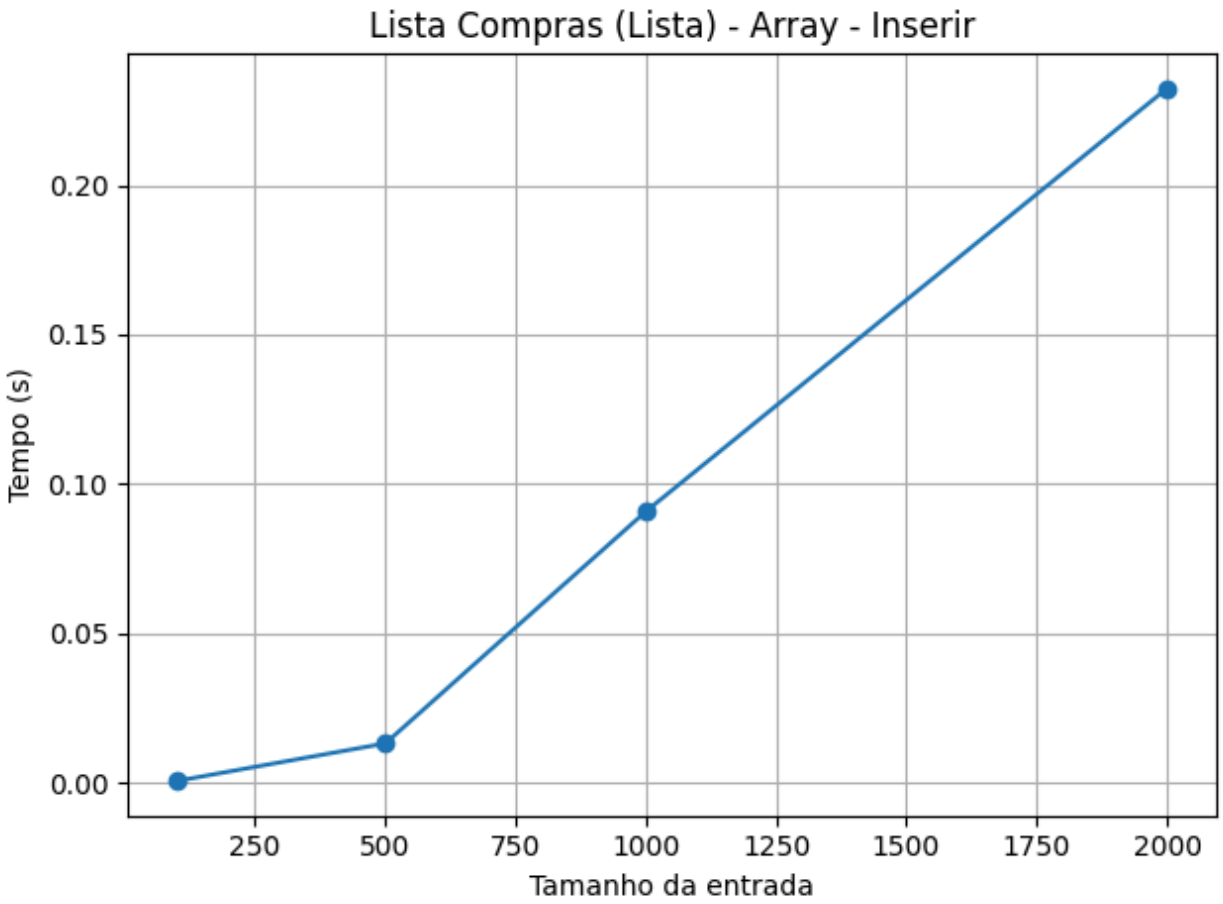

FIGURA 15: Gráfico Lista Usando Array - Inserir

4.7.2 Gráfico Lista Usando Array - Remover

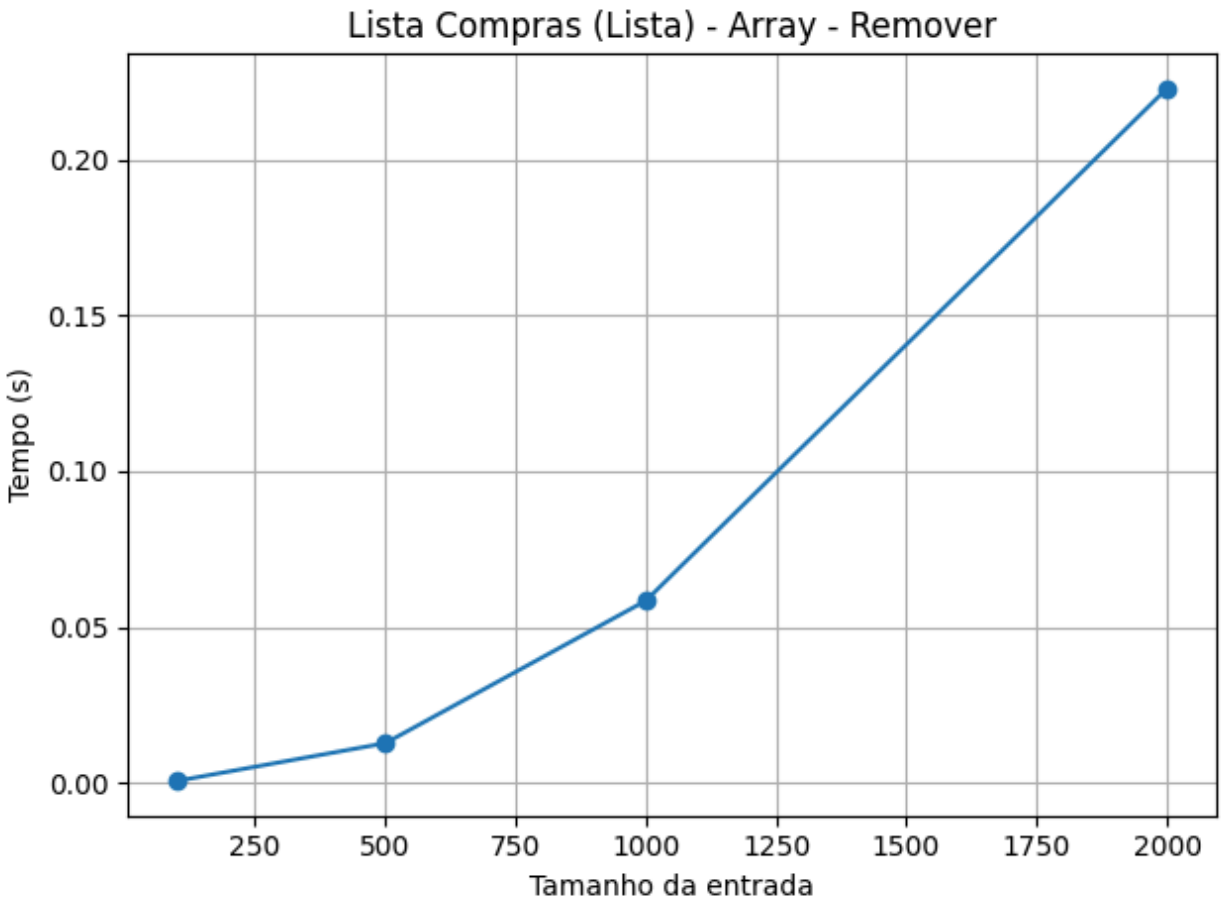

FIGURA 16: Gráfico Lista Usando Array - Remover

4.8 Gráfico Lista(Lista) Usando Lista

4.8.1 Gráfico Lista Usando Lista - Inserir

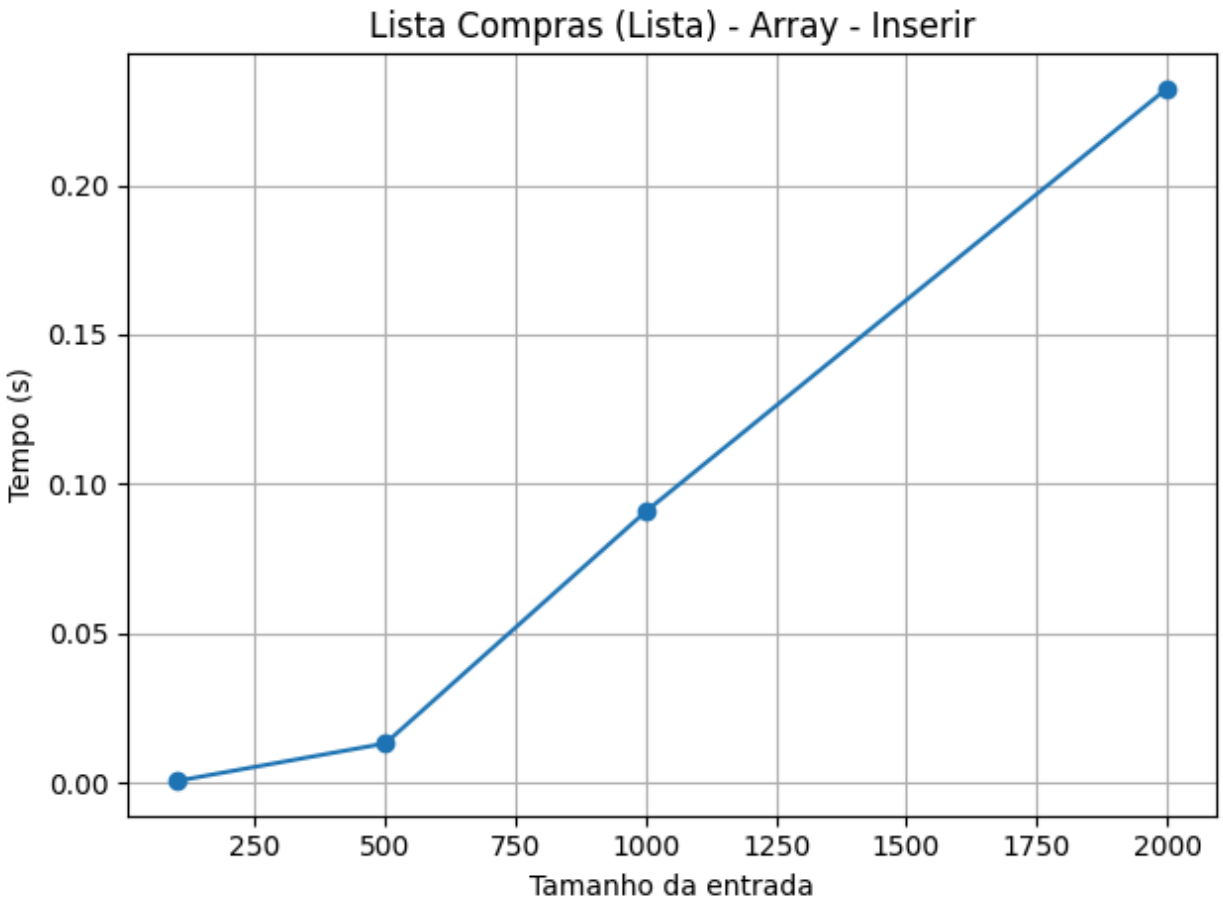

FIGURA 17: Gráfico Lista Usando Lista - Inserir

4.8.2 Gráfico Lista Usando Lista - Remover

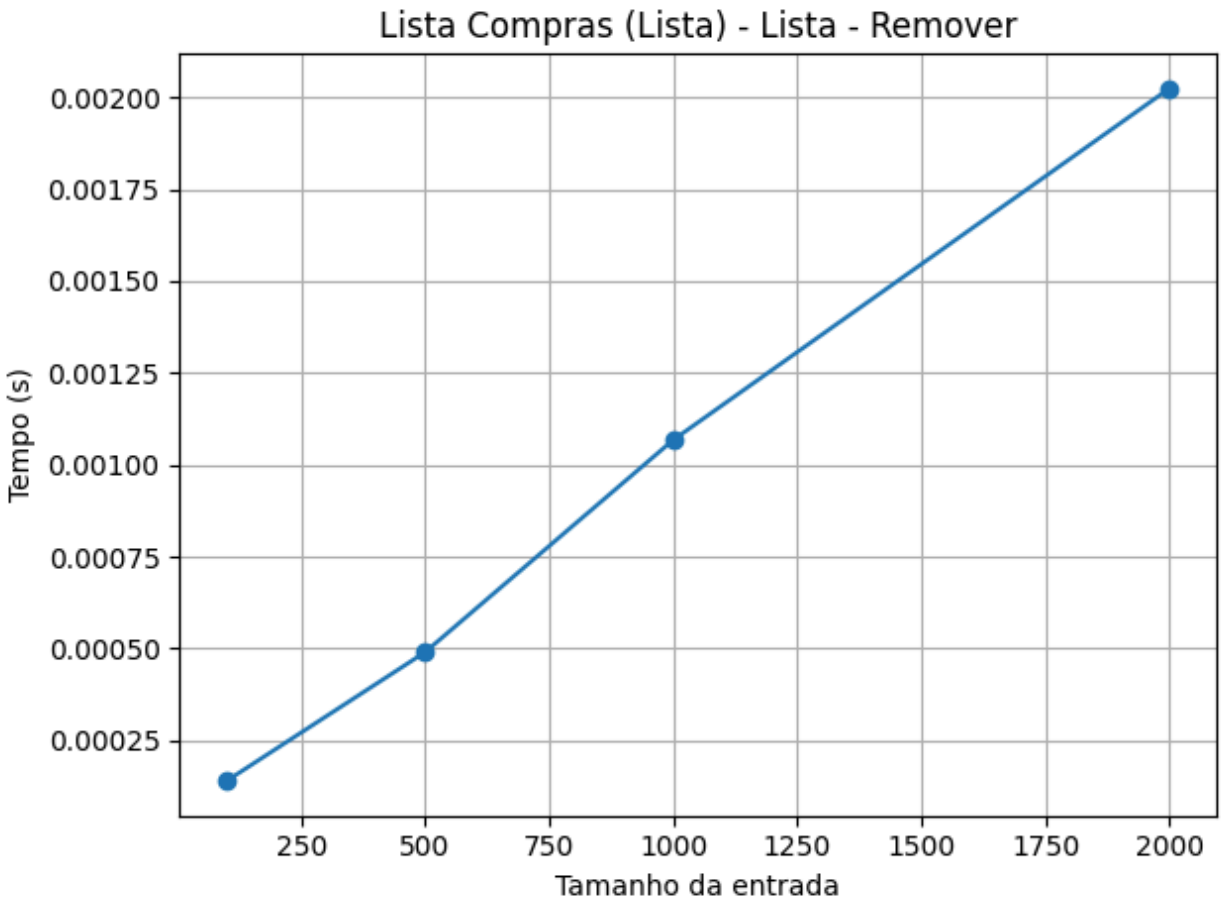

FIGURA 18: Gráfico Lista Usando Lista - Remover

4.9 Gráficos Lista (Lista de Compra) Comparando Array Vs Lista

4.9.1 Gráfico Comparativo - Inserir

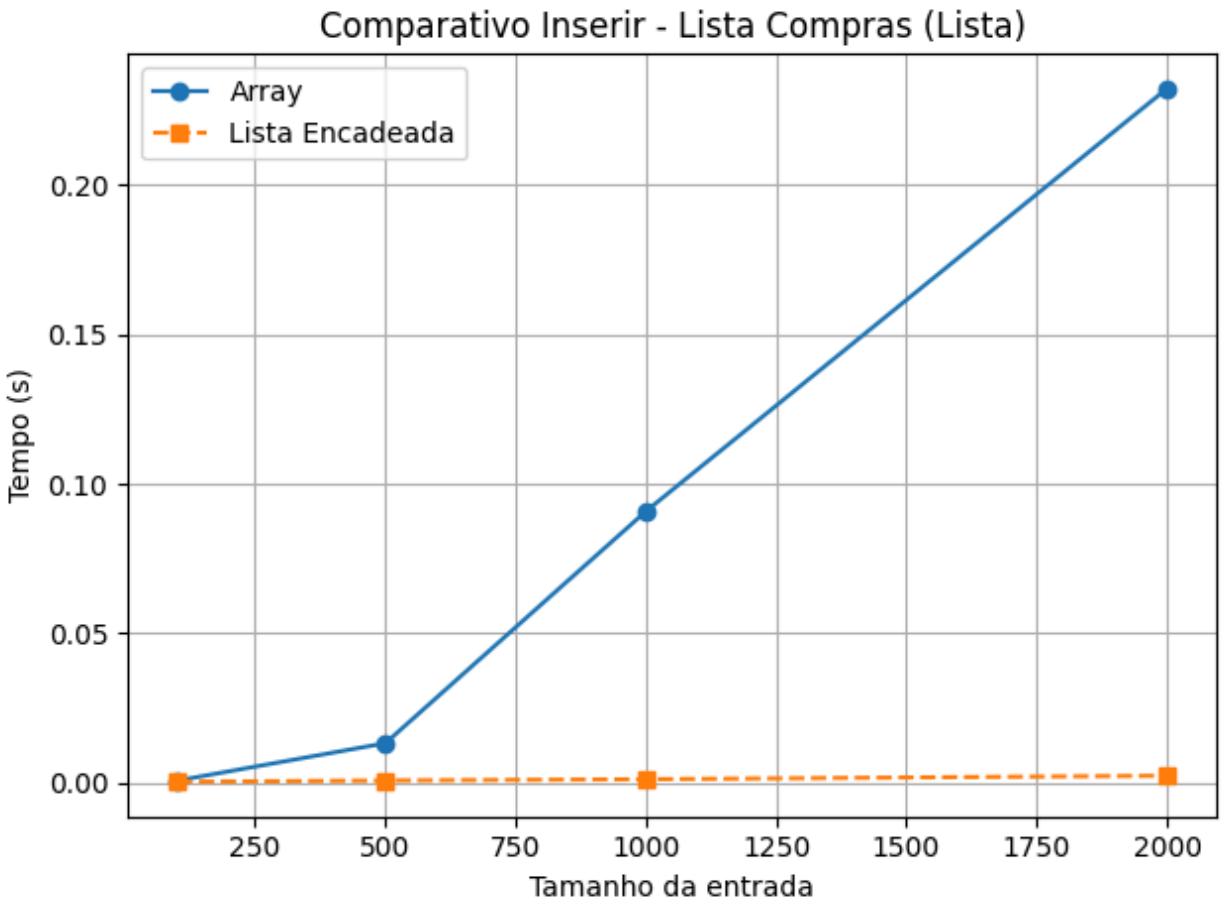

FIGURA 19: Gráfico Comparativo - Inserir

4.9.2 Gráfico Comparativo - Remover

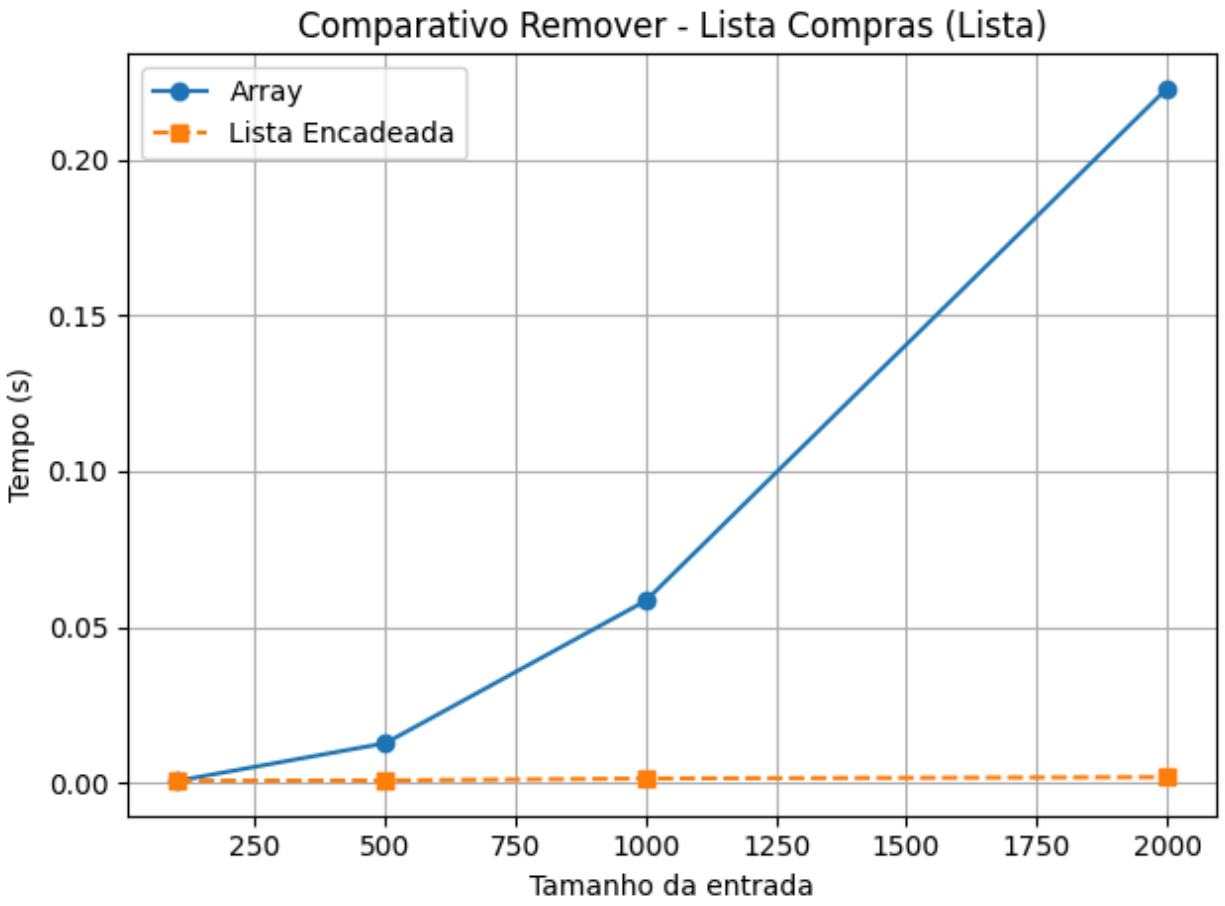

FIGURA 20: Gráfico Comparativo - Remover

4.9.3 Gráfico Comparativo - Aumentar

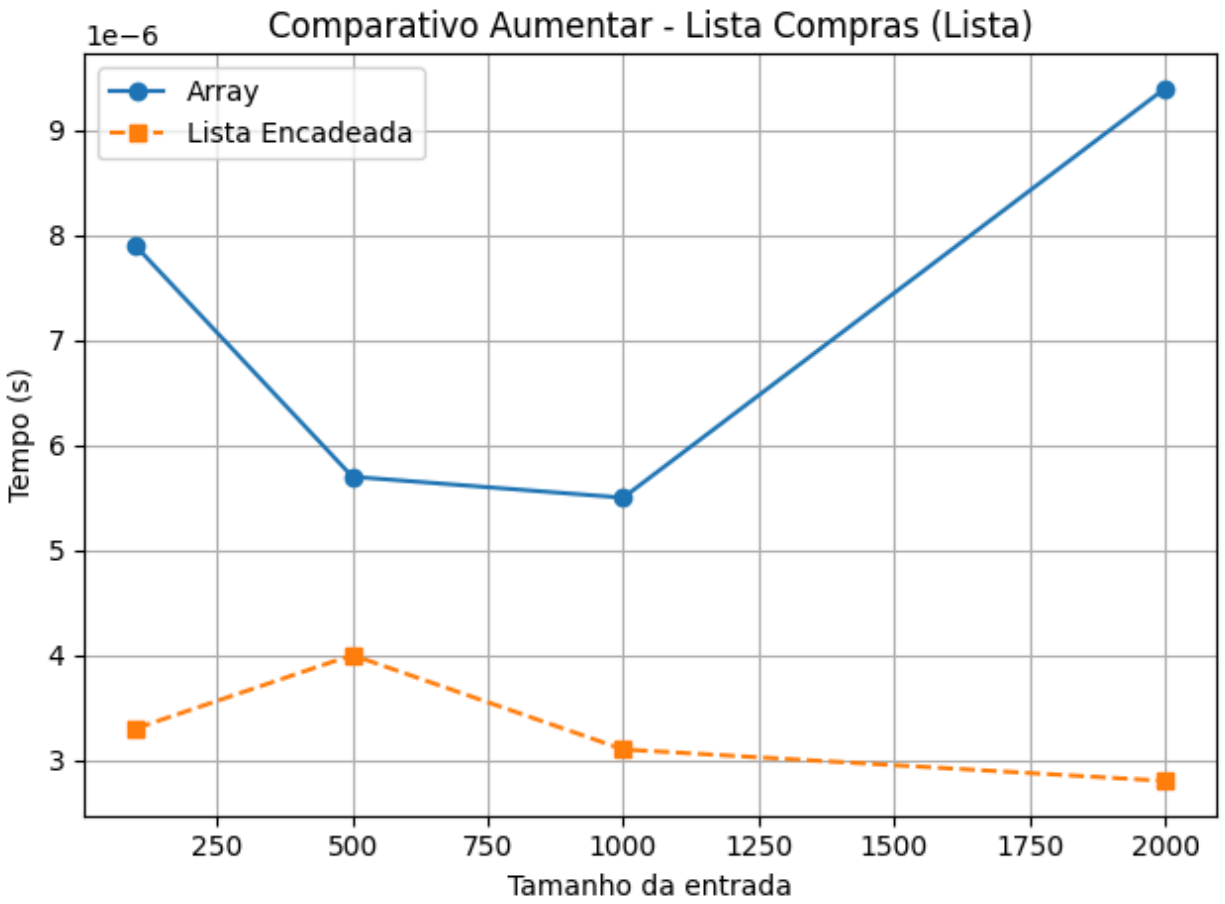

FIGURA 21: Gráfico Comparativo - Aumentar

4.3 Discussão dos Resultados

A análise dos gráficos obtidos nos experimentos permite validar empiricamente as diferenças de complexidade assintótica entre as implementações baseadas em Arrays e Listas Encadeadas.

4.3.1 Análise da Pilha (Histórico Web)

No gráfico 4.1.1 (Pilha com Array - Inserir), observamos que o tempo de execução mantém-se próximo de zero na maioria das operações, caracterizando o comportamento O(1). Contudo, ocorrem picos periódicos acentuados. Estes picos representam o momento exato em que o array atinge sua capacidade máxima física e executa a operação de redimensionamento (resize), que possui custo linear O(n) devido à cópia dos elementos.

Em contraste, a implementação com Lista Encadeada (não mostrada individualmente, mas visível no comparativo Figura 5) apresenta um crescimento mais estável, pois a alocação de memória ocorre de nó a nó, sem necessidade de cópias massivas de dados.

4.3.2 Análise da Fila (Pedidos Lanchonete)

Os gráficos comparativos Figura 12 (Inserir) e Figura 13 (Remover) evidenciam a maior disparidade de desempenho entre as estruturas.

Na Inserção ambas as estruturas comportam-se de forma eficiente, pois a inserção ocorre no final da fila. O Array exibe crescimento linear leve nos picos de resize, enquanto a Lista cresce linearmente devido ao custo acumulado de alocação dinâmica.

Entretanto, na remoção que é o Ponto Crítico apresentado na Figura 13 demonstra dramaticamente a ineficiência do Array. A curva azul (Array) cresce linearmente de forma íngreme, confirmando a complexidade O(n). Isso ocorre porque remover o primeiro elemento da fila (índice 0) obriga o deslocamento (*shift*) de todos os elementos restantes para a esquerda. Já a curva laranja (Lista Encadeada) permanece plana, próxima a zero, comprovando que a remoção do nó cabeça é uma operação de tempo constante O(1), independente do tamanho da fila.

4.3.3 Análise da Lista (Lista de Compras)

O cenário de Lista de Compras focou na inserção no início da lista (índice 0), que é o pior caso para vetores.

O gráfico Figura 19 (Comparativo Inserir) mostra que o Array (linha azul) tem um custo de tempo que cresce proporcionalmente ao número de elementos O(n), pois cada nova inserção exige empurrar todos os itens existentes para a direita.

A Lista Encadeada (linha laranja), por sua vez, mantém-se constante e

extremamente rápida, pois inserir no início exige apenas a criação de um novo nó e a atualização do ponteiro head, sem afetar os nós subsequentes.

4.3.4 Conclusão da Análise Experimental

Os resultados confirmam que o Array é altamente sensível a operações que alteram o início da estrutura ou excedem a capacidade física, sofrendo penalidades de desempenho significativas O(n). A Lista Encadeada demonstrou superioridade nas operações de manipulação de extremidades (remoção de fila e inserção no início de lista), mantendo estabilidade O(1), embora apresente um overhead constante de memória para armazenamento dos ponteiros.

6 CONCLUSÃO

A análise realizada neste trabalho permitiu corroborar experimentalmente as previsões teóricas sobre a complexidade das estruturas de dados lineares. Os experimentos demonstraram que, embora os Arrays ofereçam acesso rápido e eficiência de memória para dados estáticos, eles sofrem penalidades severas de desempenho (O(n)) em operações que exigem redimensionamento físico ou deslocamento de elementos (como na remoção do início de uma Fila ou inserção no início de uma Lista).

Por outro lado, as Listas Encadeadas mostraram-se extremamente estáveis, mantendo complexidade constante (O(1)) para inserções e remoções nas extremidades, independentemente do tamanho da entrada. No entanto, observou-se um custo, embora menor, associado à alocação dinâmica frequente de memória para cada novo nó.

Conclui-se que a escolha entre Array e Lista Encadeada não é absoluta, mas depende do cenário de uso: Arrays são ideais para acesso aleatório e tamanho previsível, enquanto Listas Encadeadas são superiores em cenários de alta volatilidade de dados e manipulação nas extremidades, como em Filas de impressão ou históricos de navegação.

REFERÊNCIAS

https://quickcodingexplanation.medium.com/data-structures-overview-arrays-stack-queue-linked-list-hash-table-heap-binary-tree-7b88a5711a0b

https://www.programaria.org/linkedlist-vs-arraylist-por-que-falar-sobre-isso/

https://www.datacamp.com/pt/tutorial/data-structures-guide-python

https://www.programaria.org/linkedlist-vs-arraylist-por-que-falar-sobre-isso/