Universidade Federal do Rio Grande do Norte

Centro de Ensino Superior do Seridó – CERES

Departamento de Computação e Tecnologia – DCT

Bacharelado em Sistemas de Informação – BSI

Estrutura de Dados - Relatório 2

Flávio Glaydson Guimarães Lopes

Orientador: Prof. Dr. João Paulo de Souza Medeiros

Caicó-RN, 02 de Julho de 2023

Resumo

Este relatório tem como objetivo principal apresentar uma análise relacionada a tempos de execução de alguns algoritmos de busca, usando como base os seguintes algoritmos: Árvore binária, árvore AVL e tabela hash. No relatório foram desenvolvidas duas atividades principais: A geração de gráficos dos tempos de execução de cada algoritmo utilizando a ferramenta Gnuplot e a comparação de desempenho de tempo entre as estruturas de buscas citadas e seus respectivos casos de melhor, pior e caso médio na busca, caso existam.

Abstract

The main objective of this report is to present an analysis related to the execution times of some search algorithms, based on the following algorithms: Binary tree, AVL tree and hash table. Two main activities were developed in the report: The generation of graphs of the execution times of each algorithm using the Gnuplot tool and the comparison of time performance between the cited search structures and their respective cases of best, worst and average case in the search, if any.

Sumário

1. Introdução

O problema de busca, assim como o de ordenação, é um desafio fundamental em ciência da computação, e existem várias estruturas de dados e algoritmos eficientes desenvolvidos para resolver esse problema. Neste relatório, abordaremos três dessas estruturas: árvore binária de busca, árvore AVL e tabela hash.

2. Árvore binária

Uma árvore binária de busca é uma estrutura de dados hierárquica em que cada nó possui um valor e dois filhos (subárvores): um filho à esquerda, cujos valores são menores que o valor do nó, e um filho à direita, cujos valores são maiores. Essa propriedade permite a busca eficiente na árvore, pois podemos descartar metade dos valores a cada comparação.

A operação de busca em uma árvore binária de busca consiste em comparar o valor procurado com o valor do nó atual e, em seguida, seguir pela subárvore esquerda ou direita, dependendo do resultado da comparação. A busca termina quando o valor procurado é encontrado ou quando chegamos a um nó folha, indicando que o valor não está presente na árvore.

2.1. Gráficos dos tempos de execução

2.1.1. Caso médio

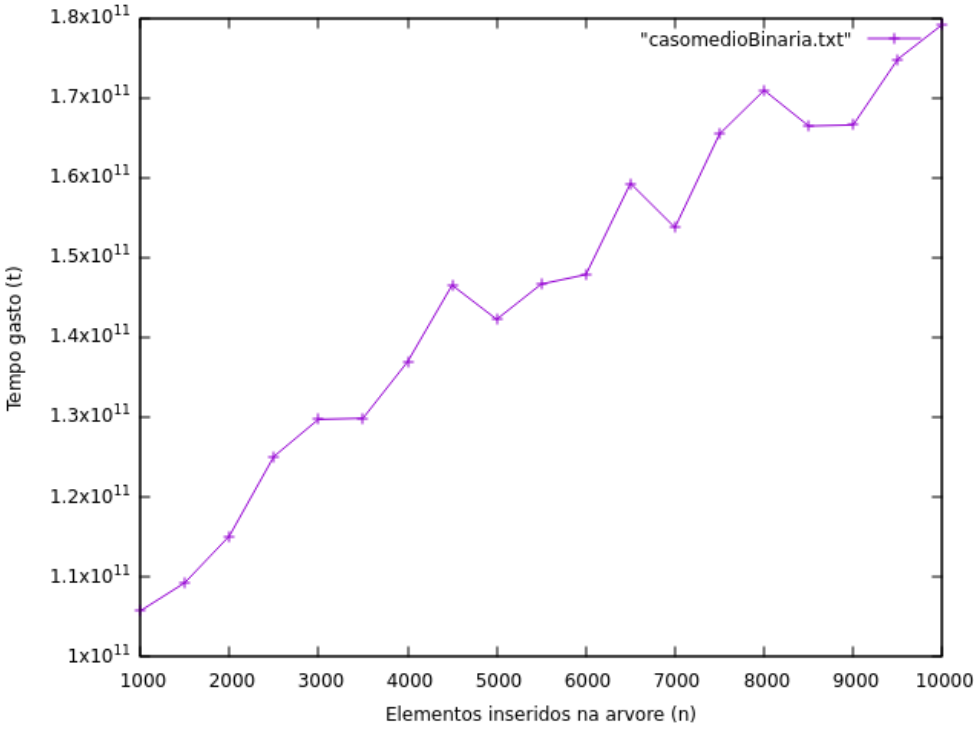

Figura 2.1: Árvore binária - Caso médio

A árvore binária de busca em seu caso médio possui uma complexidade de tempo de execução em $O(log_2 n)$. Nesse caso, o objeto da busca é procurar determinado elemento escolhido de forma aleatória, que poderá estar entre os elementos inseridos na árvore, também de forma aleatória.

2.1.2. Melhor caso

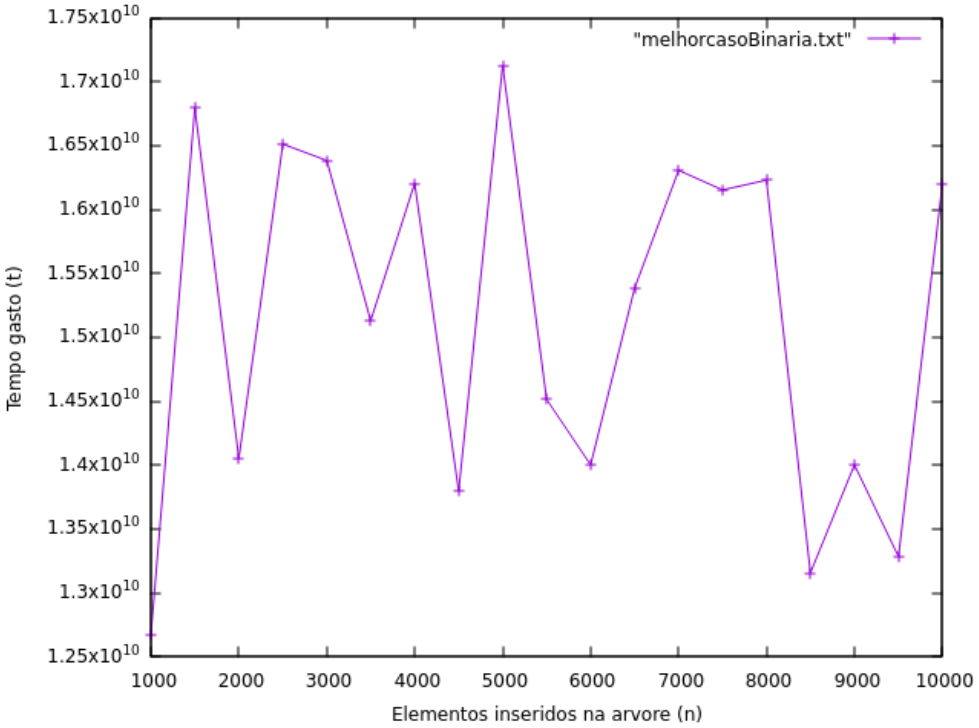

Figura 2.2: Árvore binária - Melhor médio

A árvore binária de busca em seu melhor caso possui uma complexidade de tempo de execução em $O(1)$. Nesse caso, o objeto da busca é procurar determinado na raiz da árvore, ou seja, o elemento procurado sempre vai ser encontrado na primeira verificação que poderá estar entre os elementos inseridos na árvore de forma aleatória.

2.1.3. Pior caso

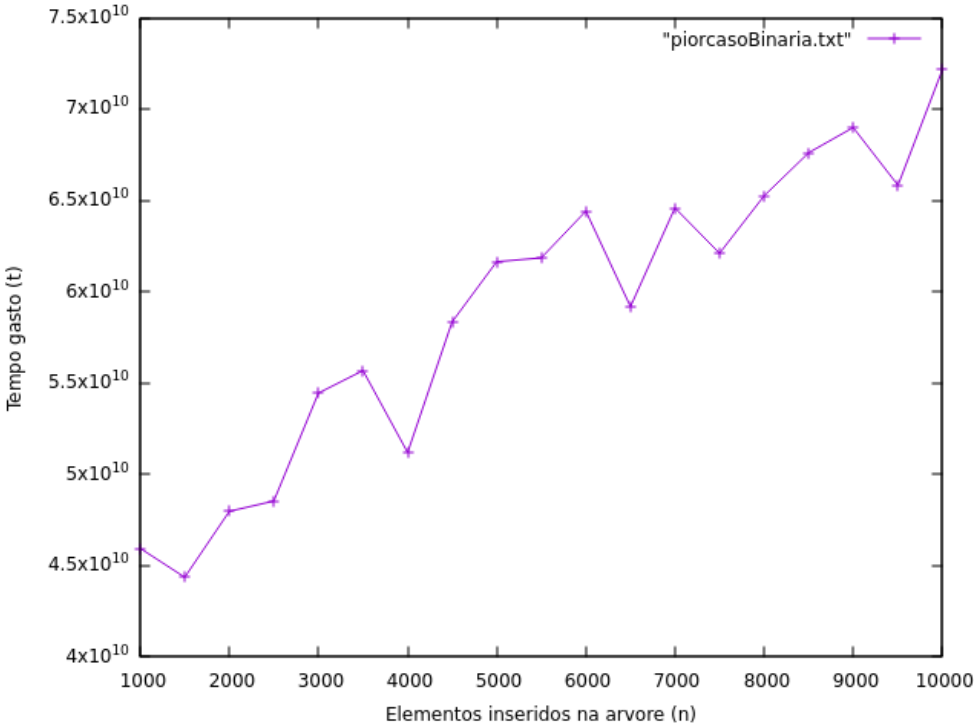

Figura 2.3: Árvore binária - Pior caso

A árvore binária de busca em seu pior caso possui uma complexidade de tempo de execução em $O(n)$. Nesse caso, ocorre quando o elemento buscado não é encontrado.

3. Árvore AVL

Uma árvore AVL é uma árvore binária de busca balanceada, onde a diferença de altura entre as subárvores esquerda e direita de cada nó é no máximo 1. Esse balanceamento é mantido através de rotações realizadas após inserções ou remoções que podem desequilibrar a árvore. A principal vantagem de uma árvore AVL é que ela garante um tempo de busca e outras operações eficientes, pois sua altura é mantida baixa e balanceada. Isso significa que as operações de busca têm complexidade O(log n), onde n é o número de nós na árvore.

3.1. Gráficos dos tempos de execução

3.1.1. Caso médio

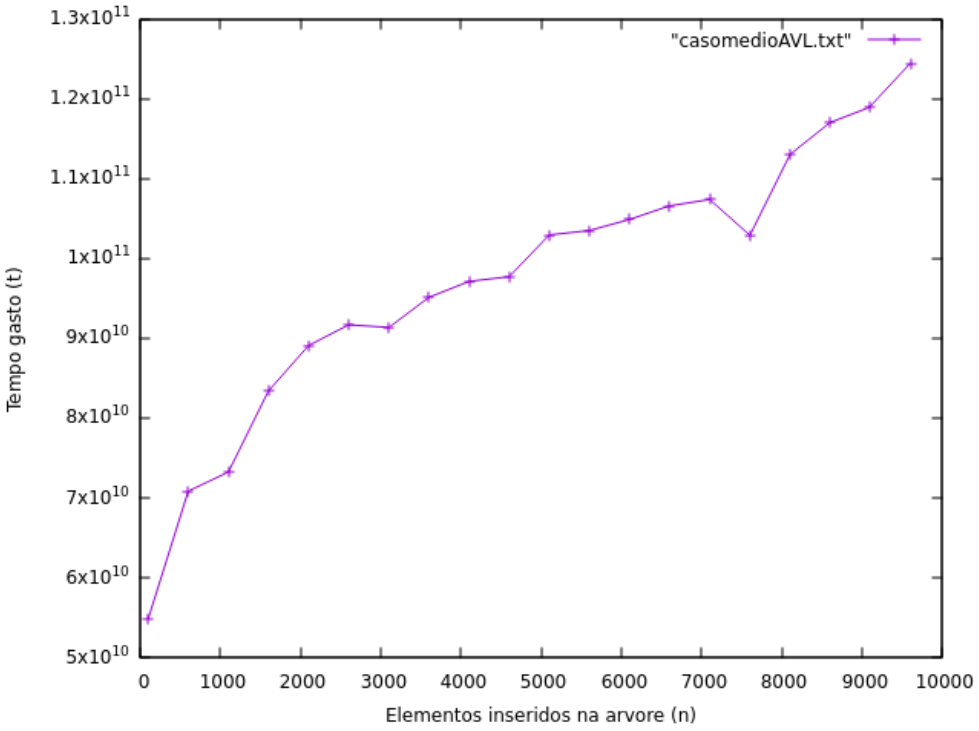

Figura 3.1: Árvore AVL: Caso Médio

A árvore AVL de busca em seu caso médio possui uma complexidade de tempo de execução em $O(log_2 n)$. Nesse caso, o objeto da busca é procurar determinado elemento escolhido de forma aleatória, que poderá estar entre os elementos inseridos na árvore, também de forma aleatória.

3.1.2. Melhor caso

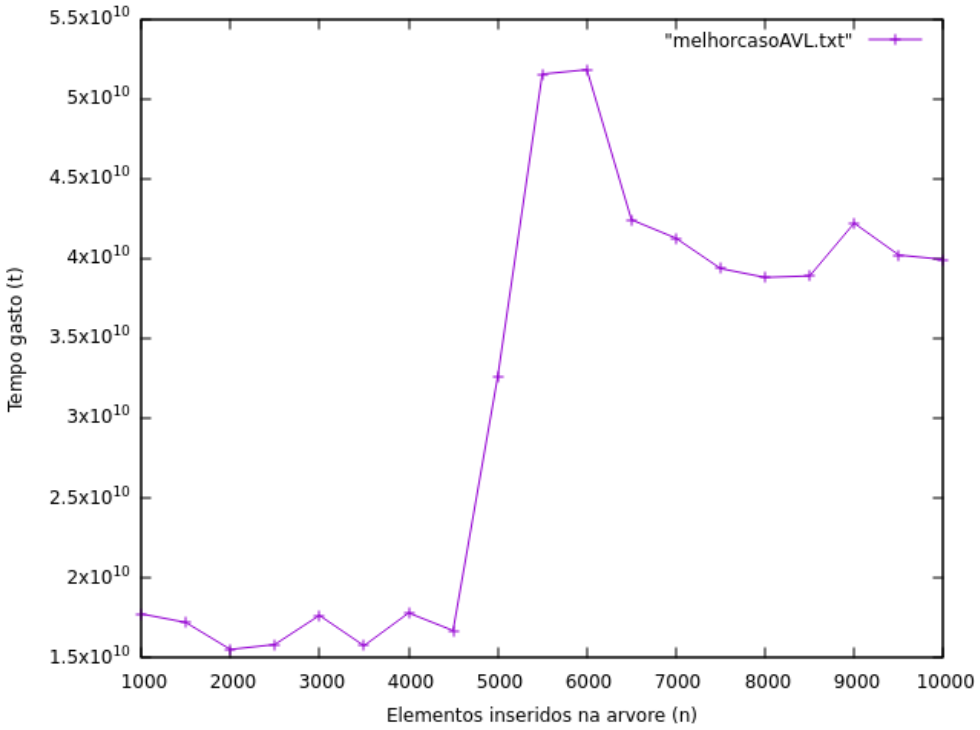

Figura 3.2: Árvore AVL: Melhor caso

A árvore AVL de busca em seu melhor caso possui uma complexidade de tempo de execução em $O(1)$. Nesse caso, o objeto da busca é procurar determinado na raiz da árvore, ou seja, o elemento procurado sempre vai ser encontrado na primeira verificação que poderá estar entre os elementos inseridos na árvore de forma aleatória.

3.2.3. Pior caso

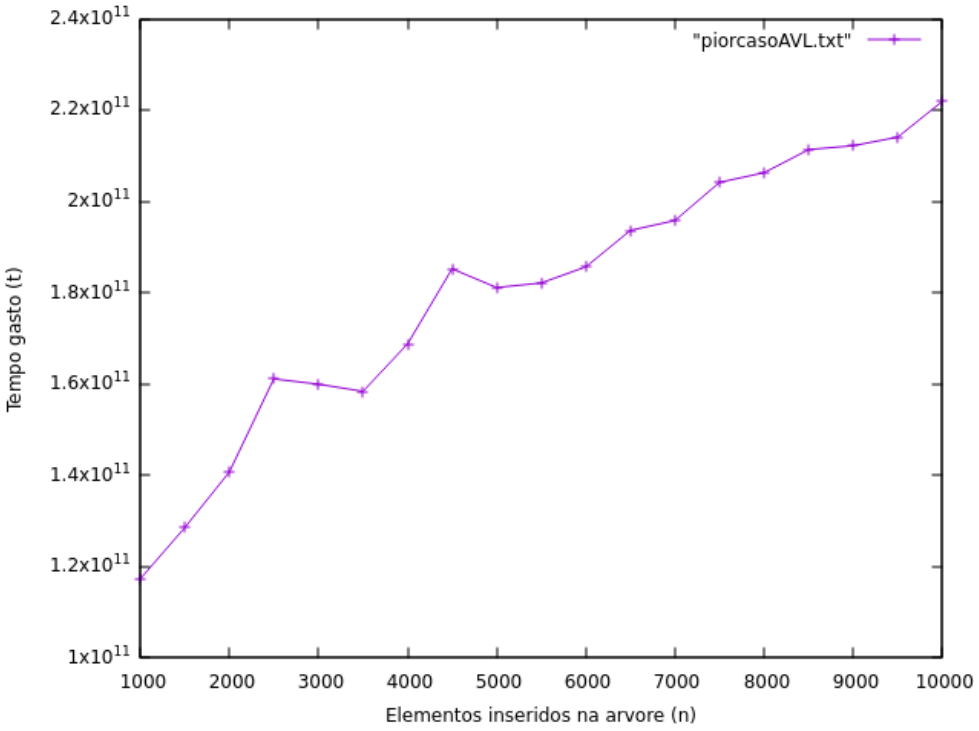

Figura 3.3: Árvore AVL: Pior caso

A árvore binária de busca em seu pior caso possui uma complexidade de tempo de execução em $O(log_2 n)$. Nesse caso, ocorre quando o elemento buscado não é encontrado.

3.2. Análise análitica

$$T_b(n) = C_1 + C_2 + C_3$$

$$T_w(n) = C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{2}\right)$$

$$T(1) = C_1$$

$$T_w\left(\frac{n}{2}\right) = C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{4}\right)$$

$$T_w(n) = C_1 + C_2 + C_{4,6} + C_{5,7} + \left[C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{4}\right)\right]$$

$$= 2\left(C_1 + C_2 + C_{4,6} + C_{5,7}\right) + T_w\left(\frac{n}{2^x}\right)$$

$$\frac{n}{2^x} = 0 \quad n = 2^x \quad \therefore \log_2 n = x$$

$$T_w(n) = \log_2 n\left(C_1 + C_2 + C_{4,6} + C_{5,7}\right) + T_w\left(\frac{\log_2 n}{\log_2 n}\right) =$$

$$\log_2 n\left(C_1 + C_2 + C_{4,6} + C_{5,7}\right) + C_1$$

4. Tabela Hash

Uma tabela hash é uma estrutura de dados que permite o armazenamento e recuperação eficiente de pares chave-valor. Ela utiliza uma função de hash para mapear as chaves para posições na tabela, onde os valores correspondentes são armazenados. Uma boa função de hash distribui as chaves uniformemente pela tabela, minimizando as colisões (quando duas chaves são mapeadas para a mesma posição). Para lidar com colisões, diferentes técnicas podem ser empregadas, como encadeamento (armazenando uma lista de valores para cada posição) ou endereçamento aberto (tentando outras posições na tabela quando ocorre uma colisão).

A busca em uma tabela hash é extremamente rápida, geralmente com complexidade O(1), pois a função de hash é direcionada diretamente para a posição onde o valor é armazenado. No entanto, em casos de muitas colisões, o desempenho pode ser reduzido.

4.1. Gráficos dos tempos de execução

4.1.1. Caso base

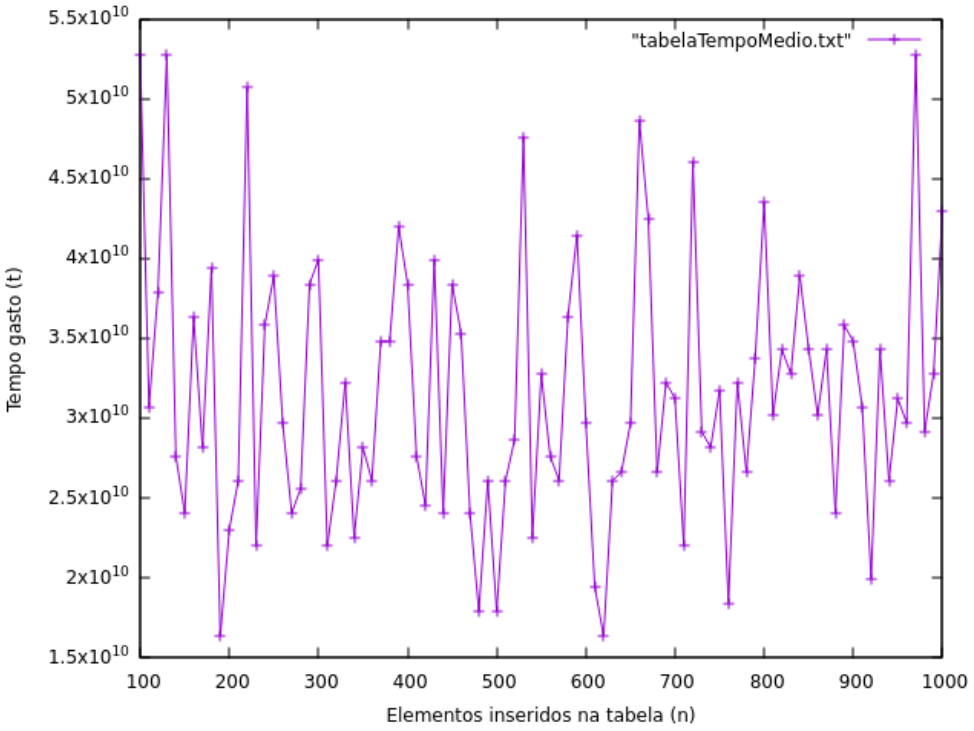

Figura 4.1: Tabela Hash: Caso base

. A tabela de dispersão tem tempo de execução O (1) em seu melhor e médio caso e, pode chegar a ter tempo de execução até O (n) em seu pior caso.

4.2. Análise análitica

$T_b(n) = c$

$T_w(n) = an + b$

$T_a(n) = ?$

$T_a(n) = \frac{1}{4}c + \frac{1}{4}c + \frac{1}{4}c + \frac{1}{4} \cdot 4c$

$T_a(4) = \frac{1}{4}(7c)$

$T_a(n) = \underbrace{\frac{1}{n}c + \frac{1}{n}c + \ldots}_{(n-1)\frac{1}{n}c} + \frac{1}{n}nc$

$T_a(n) = (n-1)\frac{1}{n}c + \frac{1}{n}nc$

$= \frac{1}{n}\left[nc - c + nc\right]$

$= \frac{1}{n}\left[2nc - c\right]$

$T_a(n) = \frac{c}{n}(2n-1)$

$= c\left(2 - \frac{1}{n}\right)$

$T_a(n) \in (H)(1)$

5. Comparações

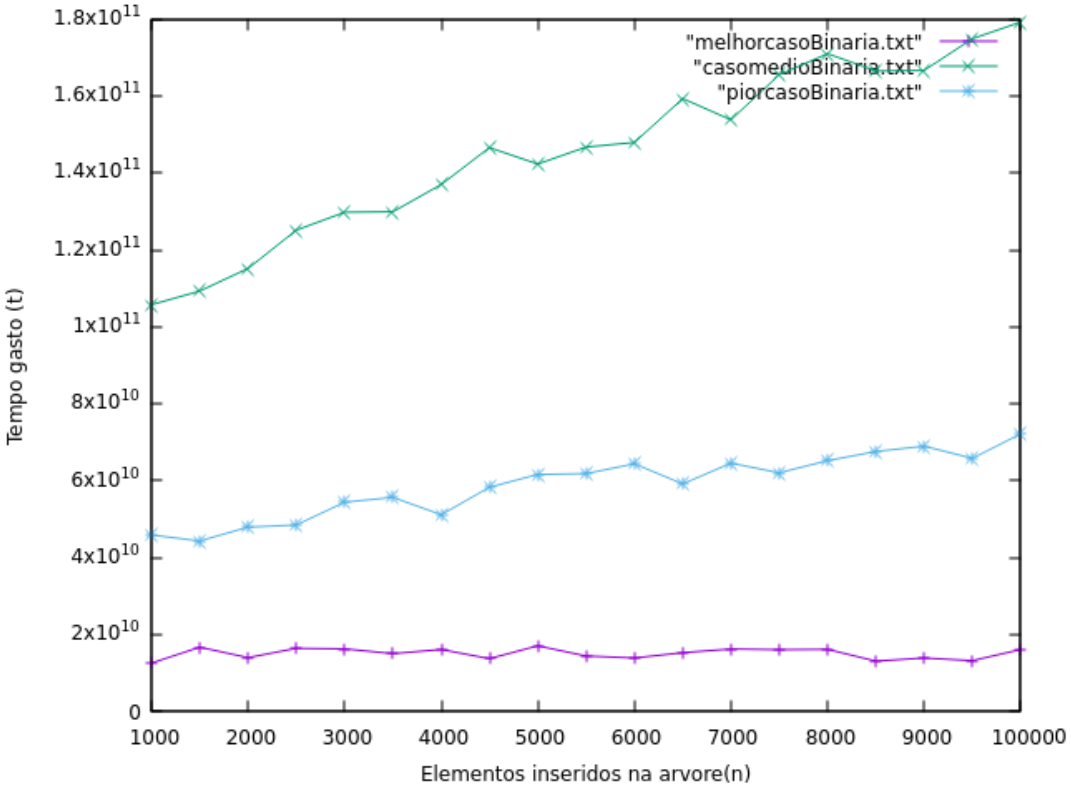

Figura 5.1: Comparação da árvore binária(melhor, pior e caso médio)

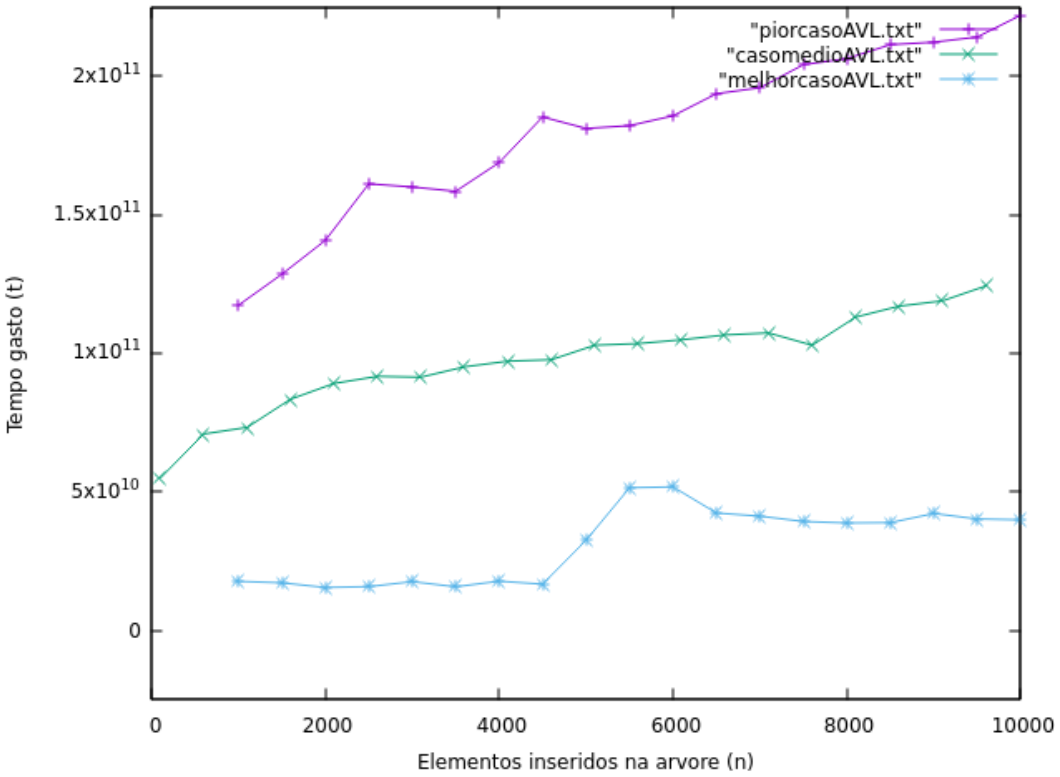

Figura 5.2: Comparação da árvore AVL(melhor, pior e caso médio)

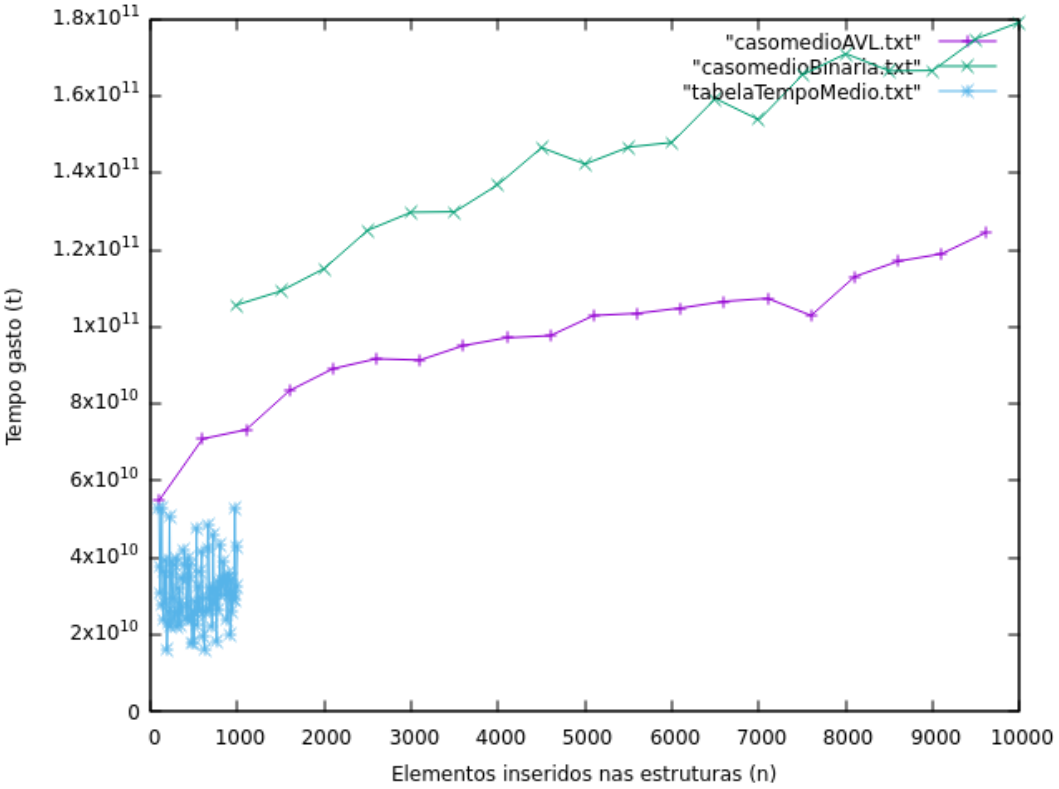

Figura 5.2: Comparação da árvore AVL, binária e tabela hash