Comparação entre as estratégias pivô aleatório, posição central e mediana de três na implementação do algoritmo de ordenação *Quick Sort*

Gustavo Macedo[1], **Heloisa Aparecida Alves**[1], **Luiz Fernando Becher de Araujo**[1]

[1]Colegiado de Ciência da Computação
Universidade Estadual do Oeste do Paraná (Unioeste)
Caixa Postal 711 – 85.819-110 – Cascavel – PR – Brasil

{gustavo.macedo3,heloisa.alves,luiz.araujo2}@unioeste.br

Resumo. *Este artigo apresenta um estudo comparativo entre três abordagens para o algoritmo de ordenação Quick Sort: o pivô aleatório, o pivô central e o pivô escolhido pela mediana de três. A análise detalhou o impacto de cada estratégia no desempenho do algoritmo, considerando diferentes padrões de entrada e tamanhos de dados. Os resultados apontam que a abordagem com pivô aleatório apresentou o melhor desempenho médio, seguida pela escolha do pivô central, enquanto a mediana de três, embora eficiente em particionamentos equilibrados, mostrou-se menos vantajosa no contexto geral devido ao seu maior custo computacional. Este estudo reforça a importância de selecionar a estratégia mais adequada à natureza dos dados em problemas de ordenação.*

1. Introdução

Desenvolvido em 1960 por Charles Antony Richard Hoare, o *Quick Sort* é um algoritmo de ordenação que utiliza o paradigma de dividir para conquistar, dividindo o vetor em vetores menores e os ordenando de forma independente, mantendo os números menores à esquerda e os maiores à direita [1]. Seu pior caso possui tempo de execução $O(n^2)$, mas, apesar disso, geralmente é um algoritmo eficiente e possui um tempo de execução esperado de $O(n \log n)$ [2].

A principal característica do *Quick Sort* é a função de particionamento, responsável por dividir o vetor em duas partes: uma com os elementos menores que o pivô e outra com os elementos maiores que o pivô. O algoritmo começa com a escolha do pivô, que é utilizado para reorganizar os elementos do vetor. Em seguida, é feita uma troca entre os elementos que são menores e maiores que o pivô, garantindo que o pivô esteja em sua posição final [1]. Assim, esse algoritmo combina simplicidade e eficiência, tornando-se uma escolha popular para a ordenação de grandes volumes de dados em diversas aplicações.

```
1 void quicksort(int *V, int inicio, int fim) {
2     if (inicio < fim) {
3         int pivo = particiona(V, inicio, fim);
4         quicksort(V, inicio, pivo - 1);
5         quicksort(V, pivo + 1, fim);
6     }
7 }
```

Código 1. Função principal do Quick Sort

O Código 1 mostra a função principal do *Quick Sort* e suas chamadas recursivas, após a escolha do pivô através da função de particionamento. Ele organiza os elementos em torno de um pivô, separando os menores à esquerda e os maiores à direita, usando a função "particiona". Em seguida, chama a si mesmo para ordenar recursivamente as sublistas à esquerda e à direita do pivô, até que todas as partes estejam ordenadas.

Para uma melhor visualização e compreensão do tema, este artigo analisará o comportamento de três estratégias distintas do algoritmo *Quick Sort*. A primeira estratégia utiliza o algoritmo clássico, no qual o pivô é escolhido de forma aleatória. Na segunda abordagem, o pivô é selecionado na posição central do vetor a ser ordenado. Por fim, a terceira estratégia adota um método em que três elementos do vetor são sorteados, e o pivô é definido como o valor central entre os três sorteados.

Este estudo busca avaliar o desempenho dessas estratégias em conjuntos de dados com características variadas. Durante os testes, o critério de análise será o tempo cronológico necessário para a execução de cada método, permitindo uma compreensão mais profunda de seu comportamento em diferentes cenários.

O presente artigo está organizado da seguinte forma: a seção 2 descreve a implementação, detalhando a linguagem de programação utilizada, o ambiente de teste, as entradas do algoritmo, bem como as implementações propostas junto com os seus custos; a seção 3 apresenta os resultados obtidos; e a seção 4 contém as considerações finais.

2. Desenvolvimento

A modelagem e os testes dos algoritmos foram realizados na linguagem de programação C, utilizando o ambiente de desenvolvimento Visual Studio Code. As estratégias implementadas foram executadas em um dispositivo do tipo TVBox, apreendido pela Receita Federal e doado à Universidade Estadual do Oeste do Paraná para fins de descaracterização. O modelo utilizado, um $inXPlus$ com SoC da Rockchip, foi adaptado para rodar a distribuição Linux Armbian 24.11 como sistema operacional, substituindo o Android originalmente configurado para retransmissão de conteúdo pirata.

Graças ao seu hardware limitado, o dispositivo oferece um ambiente ideal para facilitar a comparação entre as estratégias implementadas. Além disso, para maximizar o desempenho, a compilação dos algoritmos foi realizada com o nível de otimização $O2$ do compilador GCC. A Tabela 1 mostra as especificações do dispositivo TVBox.

Especificação	Descrição
SoC	Rockchip RK3228A
CPU	Quad-Core ARM Cortex-A7 de 1.2 GHz
Cache L1	I-Cache/D-Cache de 32KB/32KB
Cache L2	256KB unificado
RAM	2 GB LPDDR2 de 800 MHz
Armazenamento	8 GB eMMC

Tabela 1. Especificações do in X plus

Os testes foram conduzidos com vetores de entrada lidos a partir de arquivos fornecidos pelo professor da disciplina via Google Drive. Esses arquivos foram organizados

em quatro categorias distintas para os diferentes cenários de teste: vetores aleatórios, decrescentes, crescentes e parcialmente ordenados.

Para avaliar o impacto no tempo de processamento do algoritmo de ordenação, os testes foram aplicados em vetores de tamanhos variados: 100, 200, 500, 1.000, 2.000, 5.000, 7.500, 10.000, 15.000, 30.000, 50.000, 75.000, 100.000, 200.000, 500.000, 750.000, 1.000.000, 1.250.000, 1.500.000 e 2.000.000. É importante destacar que os valores de cada arquivo foram gerados de forma aleatória para evitar qualquer análise preditiva por parte do algoritmo.

A leitura dos vetores de teste utilizou alocação dinâmica via *arrays*. Após essa etapa, uma função foi chamada para realizar a leitura dos arquivos e armazenar os valores no *array* correspondente. Em seguida, cada abordagem do *Quick Sort* foi aplicada, diferenciando-se na estratégia de escolha do pivô:

- Na primeira abordagem, o pivô foi escolhido de forma aleatória;
- Na segunda, o pivô foi selecionado na posição central do vetor;
- Na terceira, três elementos do vetor foram sorteados, e o pivô foi definido como o valor central entre esses três.

Com o pivô escolhido, o vetor de entrada foi particionado em duas partes, e os subvetores gerados foram ordenados de forma recursiva. Por fim, os subvetores foram mesclados em um único vetor ordenado. Os resultados dos testes foram apresentados em segundos, utilizando a função *clock()* para medir o tempo de execução. O cálculo do tempo foi realizado a partir do valor inicial do relógio do sistema operacional, retornando um valor de precisão dupla que representa o número de segundos desde o início da medição.

Diante disso, o presente capítulo apresenta os detalhes do desenvolvimento do método de ordenação proposto, organizado em três fases distintas: a análise dos algoritmos, que inclui os detalhes de sua implementação; a avaliação do custo assintótico de cada abordagem utilizada; e a análise do custo empírico associado à execução de cada uma delas.

2.1. Implementação das estratégias

A priori, partiu-se de uma abordagem padrão do algoritmo *Quick Sort*, utilizando esquemas clássicos de particionamento. O primeiro é o *Hoare's partition scheme*, no qual o pivô é sempre escolhido como o número mais à esquerda do *array*. O segundo é o *Lomuto's partition scheme*, que define o pivô como o elemento mais à direita do *array*) [3]. Ambos os métodos possuem vantagens e desvantagens dependendo da configuração dos dados de entrada, sendo escolhas populares para análises de desempenho e comparações entre implementações do *Quick Sort*. Essas abordagens clássicas serviram de base para variações mais sofisticadas, que buscam melhorar a eficiência em cenários específicos ou evitar o desempenho degradado no pior caso.

Dessa forma, as três estratégias descritas na Seção 2 serão detalhadas a seguir. A primeira consiste na escolha do pivô de forma aleatória, seguida pela seleção do pivô central, e, por fim, pela abordagem que utiliza a mediana de três elementos.

2.1.1. Aleatório

A abordagem apresentada para a escolha do pivô no Código 2 utiliza um índice aleatório dentro do intervalo delimitado pelos índices $inicio$ e $final$ do vetor. O índice do pivô é calculado com a expressão "pivo_idx = inicio + rand() % (final - inicio + 1)", garantindo que qualquer elemento do subvetor possa ser selecionado como pivô. Após determinar o pivô, ele é trocado com o elemento na posição inicial para facilitar o particionamento. Em seguida, o vetor é dividido em duas partes: os elementos menores ou iguais ao pivô são posicionados à esquerda, enquanto os maiores ficam à direita. A troca entre elementos é realizada sempre que as condições de ordenação não são atendidas, até que os ponteiros esq e dir se cruzem.

Escolher o pivô de forma aleatória não elimina a possibilidade de selecionar o menor ou maior valor do vetor, mas reduz a probabilidade de particionamentos desbalanceados, comuns em métodos determinísticos como a escolha do primeiro ou último elemento. Embora um pivô aleatório possa ocasionalmente levar ao pior caso $O(n^2)$, isso é menos provável do que em abordagens determinísticas. Na prática, a aleatoriedade mantém o desempenho médio do *Quick Sort* em $O(n \log n)$, tornando o pior desempenho um evento mais difícil de acontecer, o que torna essa estratégia vantajosa em alguns cenários.

```c
int particiona_aleatorio(int *V, int inicio, int final) {
    int esq, dir, pivo, aux;

    int pivo_idx = inicio + rand() % (final - inicio + 1);
    aux = V[inicio];
    V[inicio] = V[pivo_idx];
    V[pivo_idx] = aux;
    pivo = V[inicio];

    esq = inicio + 1;
    dir = final;

    while (esq <= dir) {
        while (esq <= final && V[esq] <= pivo)
            esq++;
        while (dir > inicio && V[dir] > pivo)
            dir--;

        if (esq < dir) {
            aux = V[esq];
            V[esq] = V[dir];
            V[dir] = aux;
            esq++;
            dir--;
        }
    }

    V[inicio] = V[dir];
    V[dir] = pivo;
    return dir;
}
```

Código 2. Estratégia do pivô aleatório

2.1.2. Central

A abordagem de escolha do pivô pela posição central no algoritmo *Quick Sort* no Código 3 visa melhorar a divisão dos elementos, selecionando o pivô como o elemento do meio do vetor, em vez de escolher o primeiro ou o último elemento. O índice do pivô é calculado com base na média entre os índices esquerdo e direito, e, em seguida, o valor do pivô é trocado com o último elemento do vetor. Essa estratégia busca reduzir a chance de o pivô ser um valor extremo, como o maior ou o menor do vetor, o que poderia resultar em particionamentos desbalanceados e pior desempenho no caso de dados já ordenados ou quase ordenados.

Após a escolha do pivô, o algoritmo realiza uma varredura no vetor, comparando os elementos com o pivô e movendo os menores para a esquerda e os maiores para a direita. Para isso, utiliza-se um índice i que é incrementado sempre que um elemento menor ou igual ao pivô é encontrado, e esses elementos são trocados de posição. Ao final, o pivô é colocado em sua posição final, entre os elementos menores e maiores que ele. Essa abordagem, ao reduzir a chance de particionamentos desbalanceados, tem um desempenho médio esperado de $O(n \log n)$, sendo mais eficiente do que o método de escolha aleatória do pivô em algumas situações.

```
1  void swap(int* a, int* b){
2      int aux = *a;
3      *a = *b;
4      *b = aux;
5  }
6
7  int partitiona_central(int *arr, int left, int right){
8      int mid = 0;
9      mid = left+(right-left)/2;
10
11     int pivot = arr[mid];
12     swap(&arr[mid], &arr[right]);
13
14     int i = left-1;
15     for(int j = left; j < right; j++) {
16         if (arr[j] <= pivot) {
17             i++;
18             swap(&arr[i], &arr[j]);
19         }
20     }
21
22     swap(&arr[i + 1], &arr[right]);
23     return i + 1;
24 }
```

Código 3. Estratégia do pivo central

2.1.3. Mediana de três

A estratégia mediana de três visa melhorar a estabilidade do particionamento ao selecionar o pivô com base em uma estimativa mais representativa dos dados. A seguir, descreve-se

as duas funções implementadas nessa estratégia: a função de particionamento (Código 4) e a função de cálculo da mediana (Código 5).

A função apresentada no Código 4, *particiona_mediana*, realiza o particionamento do vetor em torno do pivô escolhido. A diferença entre essa estratégia e a implementação base do *Hoare's partition scheme* é que, no início da função, realiza-se o cálculo do pivô pela mediana de três valores aleatórios e, então, a troca desse pivô com o primeiro elemento do subvetor atual, para manter a coerência da abordagem.

```
1  int particiona_mediana(int *V, int inicio, int final) {
2      int esq, dir, pivo, aux;
3
4      int pivo_idx = mediana_de_tres(V, inicio, final);
5      aux = V[inicio];
6      V[inicio] = V[pivo_idx];
7      V[pivo_idx] = aux;
8      pivo = V[inicio];
9
10     esq = inicio + 1;
11     dir = final;
12
13     while (esq <= dir) {
14         while (esq <= final && V[esq] <= pivo)
15             esq++;
16         while (dir > inicio && V[dir] > pivo)
17             dir--;
18
19         if (esq < dir) {
20             aux = V[esq];
21             V[esq] = V[dir];
22             V[dir] = aux;
23             esq++;
24             dir--;
25         }
26     }
27
28     V[inicio] = V[dir];
29     V[dir] = pivo;
30     return dir;
31 }
```

Código 4. Estratégia do pivo pela mediana de três – função de particionamento

O Código 5 apresenta o cálculo da mediana de três elementos aleatórios do subvetor. O objetivo é encontrar o valor intermediário entre os três, que serve como uma boa estimativa da mediana do conjunto. Para isso, três índices são escolhidos aleatoriamente dentro do intervalo de elementos, e os valores correspondentes a esses índices são comparados. A mediana dos três valores selecionados é então escolhida como pivô, o que tende a resultar em uma escolha mais representativa do vetor, evitando a seleção de valores extremos e, consequentemente, melhorando a divisão dos elementos em duas partes mais balanceadas.

A principal vantagem dessa abordagem é que ela diminui as chances de o pivô ser o menor ou maior valor do vetor, o que poderia levar a particionamentos desbalance-

ados e degradação de desempenho. Ao considerar três elementos aleatórios, a mediana oferece uma escolha mais robusta, minimizando o risco de ocorrer o pior caso (complexidade $O(n^2)$) em dados com padrões específicos, como vetores já ordenados ou quase ordenados. Em termos de desempenho, essa estratégia contribui para uma execução mais eficiente do *Quick Sort*, com o desempenho médio ainda permanecendo em $O(n \log n)$, mas com uma menor probabilidade de sofrer quedas significativas de desempenho.

```
int mediana_de_tres(int *V, int inicio, int fim) {
    int intervalo = fim - inicio + 1;
    int idx1 = inicio + rand() % intervalo;
    int idx2 = inicio + rand() % intervalo;
    int idx3 = inicio + rand() % intervalo;
    int val1 = V[idx1];
    int val2 = V[idx2];
    int val3 = V[idx3];

    if ((val1 <= val2 && val2 <= val3) || (val3 <= val2 && val2 <= val1
        ))
        return idx2;
    else if ((val2 <= val1 && val1 <= val3) || (val3 <= val1 && val1 <=
         val2))
        return idx1;
    else
        return idx3;
}
```

Código 5. Estratégia do pivo pela mediana de três – função de cálculo da mediana

2.2. Custo Assintótico

O cálculo do custo assintótico é uma ferramenta fundamental na análise de algoritmos, permitindo avaliar o comportamento de um algoritmo à medida que o tamanho da entrada cresce. Ele se concentra na descrição do tempo de execução ou do uso de memória em termos de funções matemáticas, que capturam o crescimento do algoritmo conforme o número de elementos processados aumenta.

Para o algoritmo usando o pivô aleatório (Código 2), temos a função "particiona_aleatorio" que emprega uma abordagem para particionamento de *arrays* utilizando um pivô escolhido aleatoriamente. A análise da complexidade assintótica revela que, no caso médio, o algoritmo apresenta uma complexidade de tempo de $O(n \log n)$, o que o torna bastante eficiente para uma ampla gama de aplicações. Esta eficiência é alcançada devido à escolha aleatória do pivô, que tende a balancear a divisão dos subarrays. No entanto, no pior caso, onde o pivô escolhido é consistentemente o maior ou menor elemento, a complexidade de tempo pode degradar para $O(n^2)$, semelhante ao *Quick Sort* degenerado.

Foi considerado que cada chamada de partição divide o *array* em duas subpartições. No caso médio, cada subpartição é aproximadamente metade do tamanho do array original. Seja $T(n)$ o tempo para particionar um array de tamanho n:

$$T(n) = T(k) + T(n - k - 1) + \mathcal{O}(n) \tag{1}$$

onde k é o tamanho da subpartição menor. No caso médio, $k \approx \frac{n}{2}$:

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n) \tag{2}$$

Aplicando o Mestre Teorema, temos que:

$$T(n) = \mathcal{O}(n \log n) \tag{3}$$

Portanto, a complexidade assintótica completa da função de particionamento aleatório é $\mathcal{O}(n \log n)$ no caso médio e $\mathcal{O}(n^2)$ no pior caso.

No Código 3, a função "partitiona_central" utiliza um pivô central para dividir um array em subarrays menores. A análise da complexidade assintótica mostra que, no caso médio, o algoritmo apresenta uma complexidade de tempo de $O(n \log n)$. Isso se deve ao fato de que o pivô central tende a dividir o array de maneira equilibrada, resultando em subarrays de tamanhos aproximadamente iguais e garantindo um desempenho eficiente. No entanto, no pior caso, onde a escolha do pivô não consegue balancear as divisões, a complexidade pode aumentar para $O(n^2)$.

Em uma análise mais detalhada, foi considerado que cada chamada de partição divide o array em duas subpartições. No caso médio, cada subpartição é aproximadamente metade do tamanho do array original. Seja $T(n)$ o tempo para particionar um array de tamanho n:

$$T(n) = T(k) + T(n - k - 1) + \mathcal{O}(n) \tag{4}$$

onde k é o tamanho da subpartição menor. No caso médio, $k \approx \frac{n}{2}$:

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n) \tag{5}$$

Aplicando o Mestre Teorema, temos que:

$$T(n) = \mathcal{O}(n \log n) \tag{6}$$

Portanto, a complexidade assintótica completa da função de particionamento central também é $\mathcal{O}(n \log n)$ no caso médio e $\mathcal{O}(n^2)$ no pior caso.

Agora, na função "particiona_mediana" do Código 4 utilizou o método da mediana de três para escolher um pivô, melhorando a eficiência da ordenação por meio de uma escolha mais equilibrada do pivô. A análise da complexidade assintótica revela que, no caso médio, o algoritmo apresenta uma complexidade de tempo de $O(n \log n)$, devido à escolha do pivô que tende a dividir o array em partes aproximadamente iguais. No entanto, no pior caso, onde o pivô escolhido não equilibra as divisões, a complexidade pode aumentar para $O(n^2)$.

Para uma análise mais detalhada, consideramos que cada chamada de partição divide o array em duas subpartições. No caso médio, cada subpartição é aproximadamente metade do tamanho do array original. Seja $T(n)$ o tempo para particionar um array de tamanho n:

$$T(n) = T(k) + T(n - k - 1) + \mathcal{O}(n) \tag{7}$$

onde k é o tamanho da subpartição menor. No caso médio, $k \approx \frac{n}{2}$:

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n) \tag{8}$$

Aplicando o Mestre Teorema, temos que:

$$T(n) = \mathcal{O}(n \log n) \tag{9}$$

Portanto, a complexidade assintótica completa da função de particionamento pela mediana de três é $\mathcal{O}(n \log n)$ no caso médio e $\mathcal{O}(n^2)$ no pior caso.

De maneira geral, os três algoritmos exibem uma complexidade de $O(n \log n)$ no melhor caso, o que indica um desempenho eficiente em cenários favoráveis, como quando os dados estão bem distribuídos. No entanto, no pior caso, a complexidade pode chegar a $O(n^2)$, o que ocorre quando o algoritmo enfrenta entradas desordenadas ou em casos extremos de desbalanceamento, resultando em um desempenho significativamente mais lento. Esses comportamentos ilustram a variabilidade dos algoritmos em diferentes condições de entrada.

2.3. Custo Empírico

Nesta seção, é apresentado o custo empírico dos três algoritmos implementados em segundos, ou seja, o custo de execução de cada uma das abordagens utilizadas: *Quick Sort* Aleatório, *Quick Sort* Central e *Quick Sort* Mediana de Três. Para realizar a comparação de desempenho entre as estratégias, cada um dos métodos foi executado utilizando arquivos de entrada com diferentes tamanhos, e cada arquivo foi processado seis vezes por cada código. A primeira execução de cada arquivo foi descartada para evitar qualquer interferência inicial no desempenho devido a fatores como cache de memória ou sobrecarga do sistema, e, em seguida, foi calculada a média das execuções restantes. Essas médias foram então utilizadas para a construção dos gráficos, proporcionando uma avaliação precisa e representativa do comportamento de cada algoritmo sob diferentes condições.

A avaliação do custo de execução dos algoritmos foi realizada de duas formas distintas, a fim de obter uma análise mais abrangente. A primeira abordagem, chamada de "sem leitura", considera apenas o custo de execução do algoritmo *Quick Sort*, focando no tempo gasto especificamente na ordenação dos dados. A segunda abordagem, denominada "com leitura", leva em conta não apenas o custo de execução do código de ordenação, mas também o tempo necessário para realizar a leitura dos arquivos de entrada, o que reflete o impacto total do processo de ordenação em um cenário mais realista.

A Figura 1 mostra o quadro com os resultados das médias dos tempos de execução para a estratégia do pivô aleatório. Já a Figura 2 mostra os resultados para a estratégia do pivô na posição central do vetor.

Pivô Aleatório								
	Aleatórios		Pré-ordenados		Ordenados		Decrescentes	
entradas	sem leitura	com leitura	sem leitura	com leitura	sem leitura	com leitura	sem leitura	com leitura
100	0,000029	0,0002948	0,0001084	0,0013406	0,0000704	0,0007584	0,0001376	0,00132
200	0,0001512	0,001002	0,000227	0,0023932	0,000112	0,0011686	0,0002662	0,0019576
500	0,0001318	0,0012276	0,0003048	0,0031594	0,0001392	0,0016566	0,0003088	0,0030724
1000	0,0005872	0,0046228	0,0005154	0,0048196	0,0003586	0,0042792	0,0004528	0,0045718
2000	0,0006056	0,005318	0,0007248	0,0074276	0,0005492	0,0058752	0,0005146	0,005802
5000	0,0014908	0,0121546	0,0012148	0,0118464	0,001292	0,0117672	0,0012438	0,0119768
7500	0,0022884	0,0174614	0,0018264	0,017055	0,0018194	0,0169134	0,0018588	0,0169228
10000	0,0031502	0,0233066	0,0025102	0,0225746	0,0025348	0,0225338	0,0025866	0,0226642
15000	0,0049976	0,0348648	0,003943	0,033861	0,0039762	0,0340416	0,004135	0,0344308
30000	0,0112506	0,0706928	0,0089206	0,068677	0,0089346	0,0683086	0,0092442	0,0686088
50000	0,0207562	0,1208642	0,0165278	0,1170326	0,0163972	0,1149672	0,0168506	0,1155634
75000	0,0334034	0,1832014	0,0261238	0,1742428	0,0263778	0,1750194	0,0271924	0,175051
100000	0,0457844	0,2423924	0,037808	0,2377996	0,0372176	0,235164	0,038553	0,2383916
200000	0,1045352	0,5014456	0,0857042	0,4876728	0,083804	0,4792148	0,0893144	0,4897632
500000	0,3358548	1,3250278	0,2839286	1,2879028	0,2780224	1,260455	0,2910718	1,2917128
750000	0,5993936	2,10107	0,5071602	2,011254	0,4989834	1,9756362	0,5091112	1,9851332
1000000	0,8951782	2,8736538	0,772967	2,7410856	0,7728122	2,7447346	0,7915522	2,7639736
1250000	1,27326	3,7769874	1,100053	3,5670774	1,1031736	3,566645	1,1205742	3,5775674
1500000	1,7027762	4,7030208	1,500193	4,5082794	1,4871604	4,4495466	1,507229	4,4778586
2000000	2,6655222	6,5984402	2,390428	6,3325782	2,4286388	6,415465	2,4337306	6,3713762

Figura 1. Médias dos tempos de execução para cada um dos cenários de teste da estratégias do pivô aleatório

Pivô Central								
	Aleatórios		Pré-ordenados		Ordenados		Decrescentes	
entradas	sem leitura	com leitura	sem leitura	com leitura	sem leitura	com leitura	sem leitura	com leitura
100	0,0000126	0,0003038	0,0000194	0,0006778	0,0000938	0,0011998	0,000024	0,0008754
200	0,000088	0,0015924	0,000061	0,0010486	0,0001458	0,0013644	0,0000406	0,000807
500	0,0000588	0,0011526	0,0000798	0,0015796	0,0001044	0,0028062	0,0000564	0,0014382
1000	0,000156	0,0042968	0,0001318	0,0042534	0,0001468	0,0040174	0,0001356	0,0041468
2000	0,0002938	0,0047592	0,000267	0,0052392	0,0002082	0,0049186	0,0002316	0,0050108
5000	0,0007616	0,0112516	0,0006318	0,0112994	0,0005542	0,0111374	0,0005996	0,0110522
7500	0,0012088	0,0162274	0,0009536	0,0162084	0,0008708	0,0159116	0,0009744	0,0159404
10000	0,0015384	0,0215754	0,0013568	0,0213936	0,0012656	0,0213004	0,0013112	0,0213776
15000	0,0024846	0,0323352	0,0021686	0,0320336	0,0020546	0,0319248	0,002157	0,0324182
30000	0,005721	0,0652168	0,0048954	0,0646674	0,0046892	0,0640662	0,0050036	0,0643938
50000	0,0112598	0,1113096	0,0093876	0,10991	0,0090552	0,1076136	0,009583	0,108088
75000	0,0195428	0,1693242	0,0156132	0,1635814	0,015673	0,1640888	0,0169606	0,164695
100000	0,0282126	0,2247472	0,0236138	0,2234218	0,02337	0,221207	0,0256468	0,2254026
200000	0,0755564	0,4726902	0,0657794	0,4678988	0,0647134	0,4603022	0,071759	0,4719872
500000	0,3282792	1,3174422	0,2980472	1,3020794	0,2944218	1,2769764	0,31901	1,3196554
750000	0,6834716	2,1854552	0,6115348	2,1157776	0,6048892	2,0814604	0,6376538	2,1133804
1000000	1,1145896	3,084942	1,0212334	2,9895372	1,0242552	2,9963854	1,0689328	3,0411466
1250000	1,6782188	4,1383104	1,5481774	4,014969	1,5492094	4,0124846	1,611116	4,067525
1500000	2,34288	5,3433002	2,2021478	5,2101848	2,1786594	5,1397178	2,2831822	5,288133
2000000	3,9282028	7,8609564	3,7628814	7,7046806	3,799689	7,7859034	3,867257	7,8048318

Figura 2. Médias dos tempos de execução para cada um dos cenários de teste da estratégias do pivô central

Por fim, a Figura 3 mostra os resultados da estratégia da mediana de três. Os valores foram apresentados de duas maneiras: considerando somente a execução das es-

tratégias (sem leitura) e considerando a execução das estratégias mais a leitura dos arquivos com as entradas (com leitura). A análise dos valores obtidos foi realizada na seção 3 (Resultados).

Pivô Mediana de Três								
	Aleatórios		Pré-ordenados		Ordenados		Decrescentes	
entradas	sem leitura	com leitura	sem leitura	com leitura	sem leitura	com leitura	sem leitura	com leitura
100	0,0000484	0,0003008	0,0000782	0,0007474	0,0000762	0,000567	0,0000806	0,0007926
200	0,0001438	0,0011538	0,0002326	0,001363	0,0001556	0,001184	0,0001532	0,000892
500	0,0005944	0,0031746	0,0004506	0,0020258	0,0002622	0,0014174	0,0006414	0,0032764
1000	0,000841	0,0045234	0,000769	0,0036774	0,0005222	0,0037784	0,0009926	0,0039538
2000	0,001029	0,0052684	0,000971	0,0058218	0,000974	0,0056352	0,000976	0,00565
5000	0,0027064	0,0129954	0,0023966	0,0127334	0,0023976	0,012626	0,0024246	0,0127472
7500	0,0041538	0,0192284	0,0036714	0,0187434	0,0036572	0,0186802	0,003723	0,0187342
10000	0,0057118	0,0256918	0,004997	0,0250202	0,0049946	0,0249502	0,0051272	0,0251556
15000	0,0090406	0,0388348	0,0079888	0,0378132	0,0080224	0,0378308	0,0082042	0,0384226
30000	0,0206224	0,0800122	0,0182012	0,077916	0,0182818	0,0775902	0,0186702	0,078053
50000	0,0381106	0,138043	0,0336524	0,1339538	0,0332846	0,1317822	0,0339262	0,1324238
75000	0,060695	0,2102958	0,052999	0,2010044	0,0532338	0,2017154	0,054334	0,2020506
100000	0,0829866	0,2794868	0,0745588	0,2743764	0,0738494	0,2716958	0,0762726	0,2759086
200000	0,1828348	0,5796862	0,163125	0,5650594	0,1611984	0,5566204	0,1670538	0,5671894
500000	0,5342504	1,5229358	0,4868064	1,4908306	0,4751572	1,4578096	0,4915098	1,4920366
750000	0,8973306	2,3990486	0,8048536	2,3094666	0,7932382	2,2696646	0,8100796	2,2848734
1000000	1,2920406	3,2622984	1,1643856	3,1324432	1,168787	3,1410048	1,1869768	3,1588306
1250000	1,7529716	4,2124324	1,5946014	4,061411	1,5934316	4,0573728	1,6158682	4,071811
1500000	2,2944026	5,2959578	2,0966836	5,104127	2,070374	5,0323196	2,1248164	5,1309738
2000000	3,4543606	7,3875702	3,2103718	7,1966556	3,2148246	7,2006178	3,220123	7,1576678

Figura 3. Médias dos tempos de execução para cada um dos cenários de teste da estratégias da mediana de três

3. Resultados

Nesta seção, serão apresentados e analisados os resultados obtidos a partir da execução das diferentes estratégias de escolha de pivô no algoritmo *Quick Sort*. Os dados coletados incluem o tempo de execução medido em segundos para vetores com características variadas, como ordenação aleatória, crescente, decrescente e parcialmente ordenada, com tamanhos progressivamente maiores. Para facilitar a compreensão e destacar as diferenças entre as abordagens de pivô aleatório, pivô central e pivô pela mediana de três, os resultados serão ilustrados por meio de gráficos. Esses gráficos permitirão uma análise visual clara do desempenho de cada estratégia, evidenciando os cenários de maior eficiência e os casos em que o custo computacional foi mais elevado.

Devido a escala de distanciamento do gráfico, para melhor visualização, fora divido todos gráficos em dois, sendo então, um gráfico para os arquivos de 100 à 7.500, com variação do eixo y de 0.001, e, outro gráfico para os arquivos de 10.000 á 2.000.000, com variação do eixo y de 0.1.

3.1. *Quick Sort* com pivô aleatório

Na Figura 4, a legenda está organizada para identificar os diferentes conjuntos de dados analisados: a cor azul representa os dados aleatórios, o amarelo corresponde aos dados

decrescentes, o verde aos dados parcialmente ordenados e o vermelho aos dados crescentes. No eixo y, é apresentado o tempo de execução em segundos, enquanto o eixo x ilustra os tamanhos dos arquivos de entrada, variando de 100 a 7.500 elementos.

Inicialmente, percebe-se uma oscilação nos valores médios registrados para os arquivos de tamanhos entre 100 e 5.000. Essa variação pode ser atribuída à natureza aleatória da escolha do pivô, que pode resultar em particionamentos mais ou menos balanceados dependendo da execução.

No entanto, à medida que o tamanho dos vetores aumenta, os valores médios tendem a se estabilizar, indicando um comportamento mais consistente do algoritmo. O padrão geral observado mostra que o conjunto de dados aleatório apresenta o maior custo computacional, devido à sua falta de padrão nos dados, o que dificulta o balanceamento das divisões. Em contrapartida, os demais conjuntos de dados, como os crescentes, decrescentes e parcialmente ordenados, apresentam tempos mais homogêneos, uma vez que possuem características mais previsíveis que influenciam positivamente no particionamento. Esse comportamento reforça a relação direta entre a aleatoriedade do pivô e a complexidade do algoritmo em diferentes contextos.

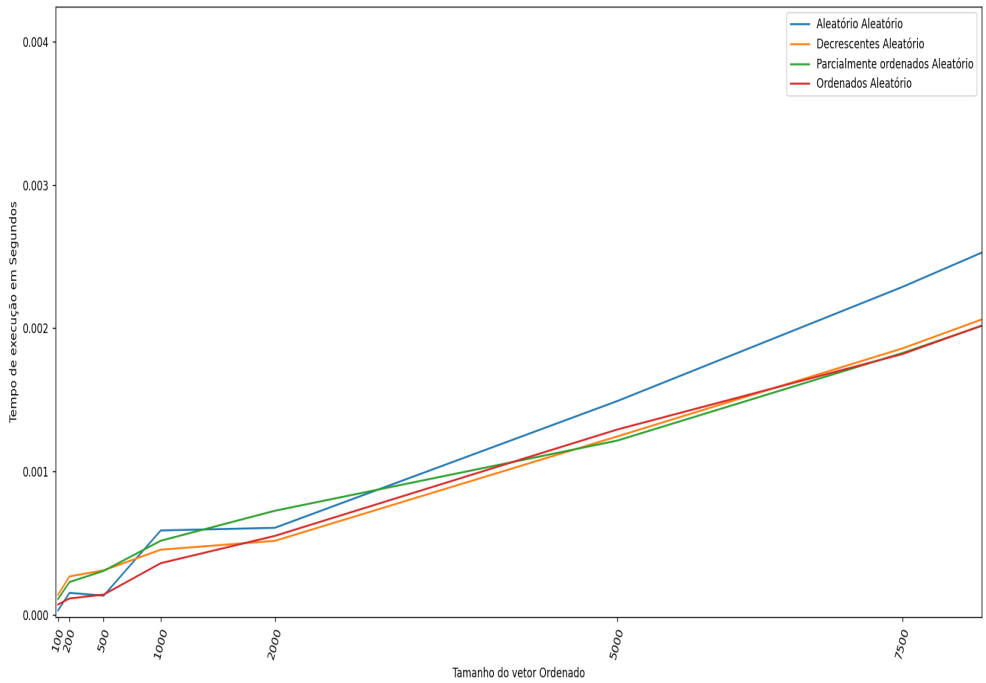

Figura 4. Todos arquivos para o método de pivô aleatório sem leitura, intervalo de 100 à 7,500.

Outro fator para o comportamento inicial mais instável do algoritmo pode estar relacionado à forma como o escalonador de tarefas do sistema operacional está lidando com o processo, trocando-o de contexto com certa frequência. Por conta do consumo, devido às entradas maiores, o processo pode ganhar mais prioridade na CPU, tornando-o mais estável. Além disso, a própria memória cache pode influenciar nesse comportamento, tornando a execução mais rápida em dado momento, por causa de *hits* mais frequentes, mas gerando um comportamento de "onda" no gráfico quando o aumento no número de

entradas passa a gerar mais *miss* do que *hit* na cache, impactando negativa e momentaneamente no tempo de execução. Após isso, o sistema se adapta a buscar os dados somente na RAM.

Seguindo, na Figura 5, apresenta-se a continuidade da análise gráfica, agora abrangendo as médias dos tempos de execução para arquivos com tamanhos variando de 10.000 a 2.000.000 elementos. Essa extensão permite observar o comportamento do algoritmo com pivô aleatório em conjuntos de dados significativamente maiores, complementando as observações realizadas anteriormente na Figura 4.

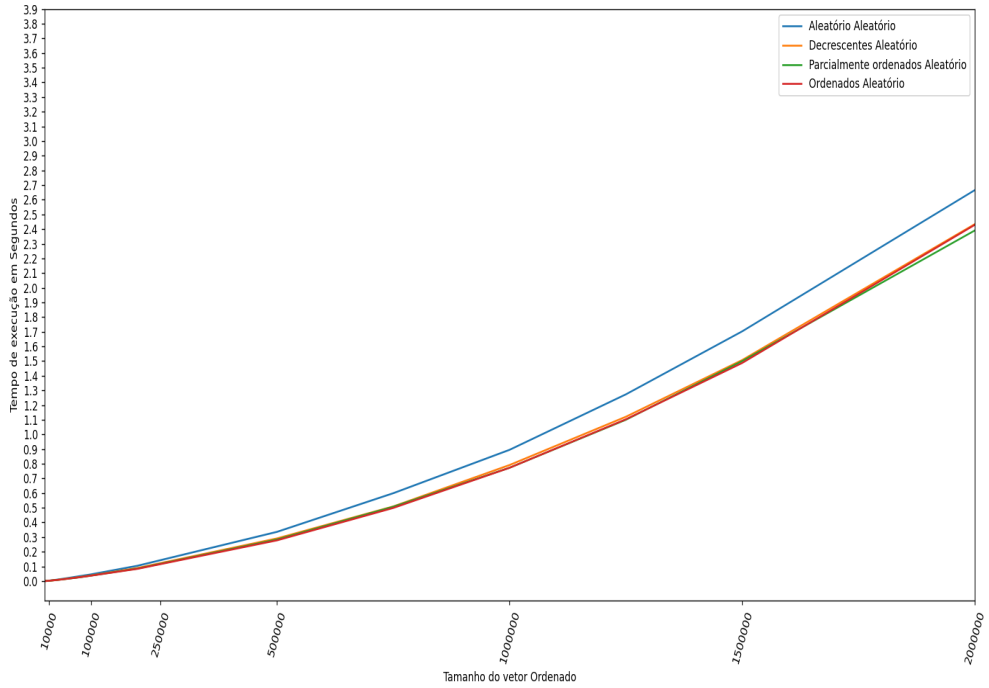

Figura 5. Todos arquivos para o método de pivô aleatório sem leitura, intervalo de 10,000 à 2,000,000.

Para os tempos coletados com a leitura de arquivos, verifica-se na Figura 6, acontece a mesma variação inicial entre os arquivos 100 e 5000, porém agora os valores de y são maiores, visto o gasto de leitura. Sobretudo, posteriormente, os valores padronizam, seguindo o mesmo esquema anterior, com o custo do conjunto de dados aleatório, se destoando com maior custo, seguido dos demais dados com valores aproximados.

Na sequência, a Figura 7 apresenta a continuidade do gráfico anterior, agora com os dados correspondentes aos arquivos variando de 10.000 a 2.000.000. Os resultados observados seguem a tendência estabelecida na análise dos arquivos anteriores, particularmente a partir da média dos tempos registrados para o arquivo de 5000. Essa consistência nos valores reflete a estabilidade do desempenho dos algoritmos à medida que o tamanho dos arquivos aumenta, mantendo o padrão identificado nas medições anteriores, com a variação inicial sendo suavizada à medida que os dados se estabilizam.

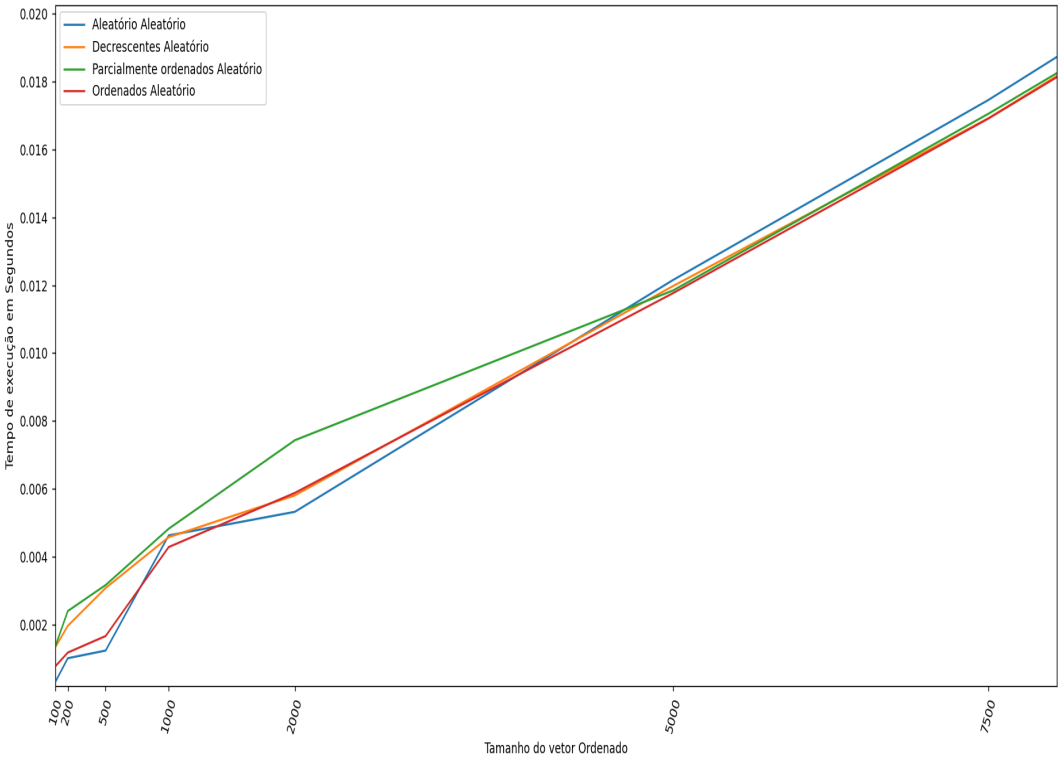

Figura 6. Todos arquivos para o método de pivô aleatório com leitura, intervalo de 100 à 7,500

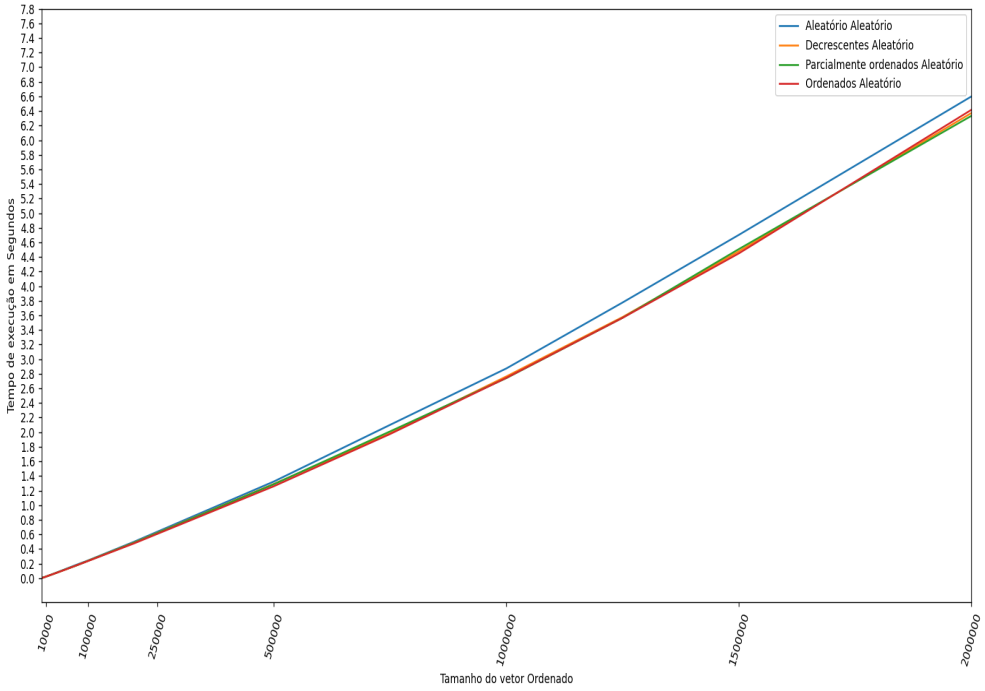

Figura 7. Todos arquivos para o método de pivô aleatório com leitura, intervalo de 10,000 à 2,000,000.

3.2. *Quick Sort* com pivô central

Como pode ser observado na Figura 8, a média dos tempos registrados para a abordagem com pivô central, sem a leitura dos arquivos, demonstra uma trajetória relativamente constante ao longo da análise. Desde os primeiros valores, os tempos permanecem próximos uns dos outros, indicando uma execução consistente do algoritmo. No entanto, a partir dos dados referentes ao arquivo de 5.000, começa a ser notado um pequeno destoamento, onde os valores começam a se afastar ligeiramente, especialmente em comparação com a abordagem de pivô aleatório. Esse desvio pode ser atribuído às características específicas da distribuição dos dados e à forma como o pivô central impacta o desempenho do algoritmo em arquivos maiores.

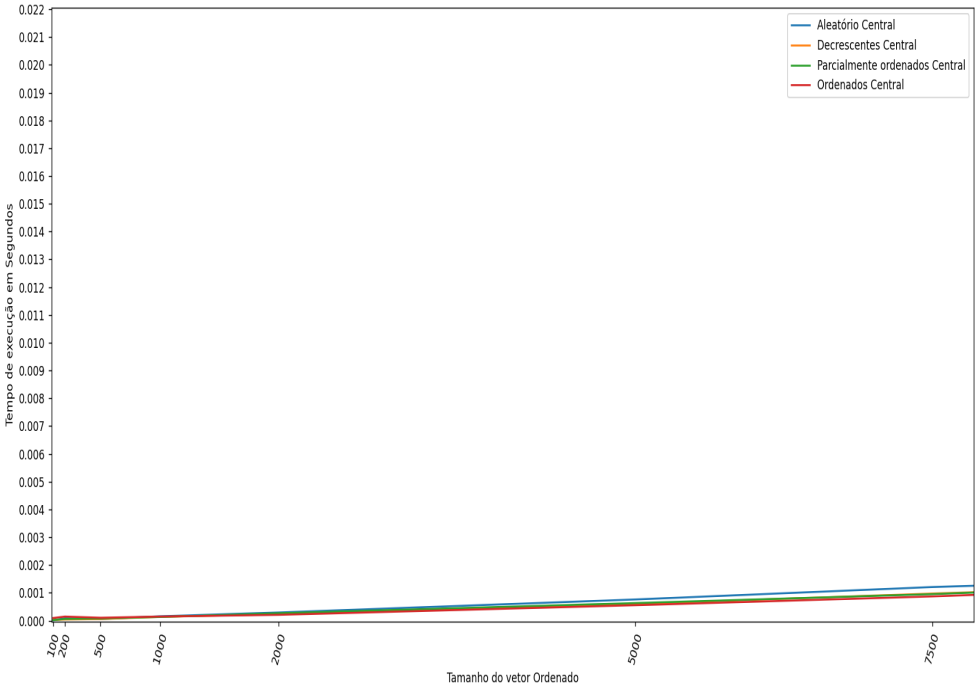

Figura 8. Todos arquivos para o método de pivô central sem leitura, intervalo de 100 à 7,500.

Seguindo, é possível observar na Figura 9 a continuação do gráfico anterior, agora com os dados dos arquivos entre 10.000 à 2.000.000. Os valores seguem o mesmo padrão pós 5.000 do gráfico anterior, sobretudo, isso até os dados do arquivo 50.000, onde se inicia um destoamento do dados Decrescentes junto aos dados Aleatórios, onde ambos apresentam valores maiores aos demais, ainda assim, o custo para os dados Aleatórios se mostra maior mesmo neste caso.

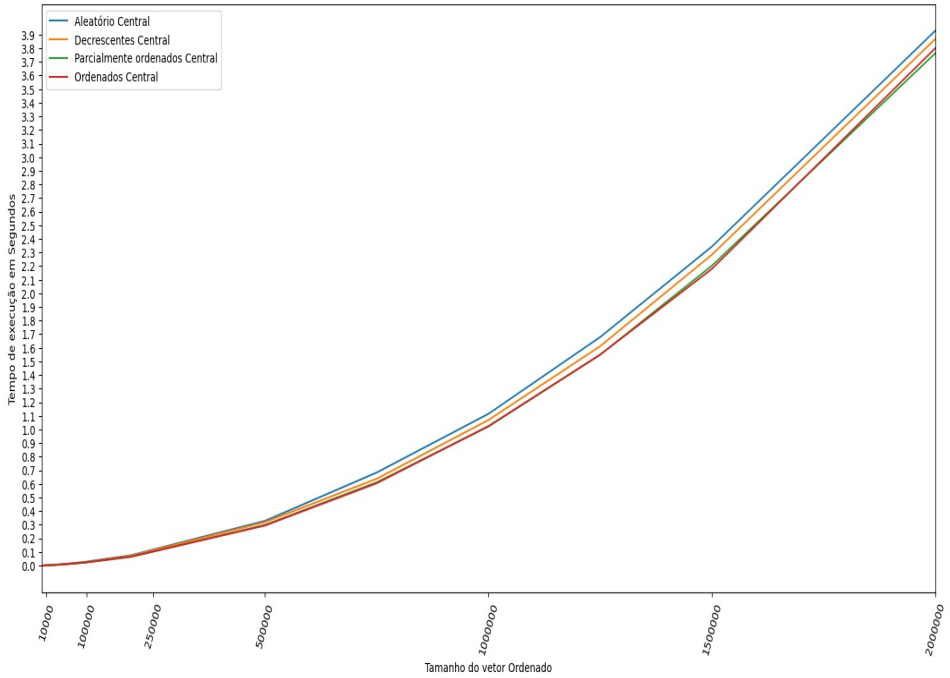

Figura 9. Todos arquivos para o método de pivô central sem leitura, intervalo de 10,000 à 2,000,000.

Nos testes realizados, considerando o tempo de leitura, é possível perceber, conforme ilustrado na Figura 10, que as médias dos tempos seguem um padrão semelhante entre as abordagens até o valor do arquivo de 5.000.

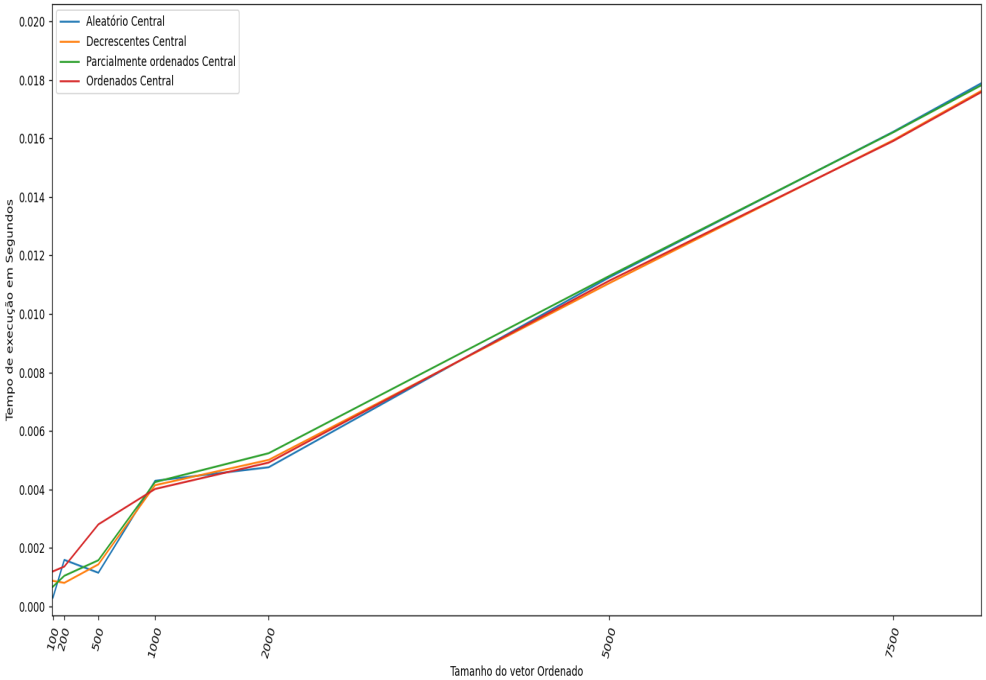

Figura 10. Todos arquivos para o método de pivô central com leitura, intervalo de 100 à 7,500.

Até esse ponto, os resultados das diferentes estratégias de pivô permanecem

próximos uns dos outros, indicando uma performance relativamente equilibrada no processo de leitura dos arquivos. No entanto, após o arquivo de 5.000, observa-se uma divergência, especialmente para a abordagem de pivô aleatório, que começa a se distanciar significativamente dos demais conjuntos de dados. Esse comportamento pode ser atribuído à aleatoriedade do processo de escolha do pivô, o que, embora contribua para uma maior dispersão no desempenho, pode também ter impactos no tempo de leitura quando lidamos com volumes maiores de dados. A diferença observada a partir desse ponto evidencia a variação de desempenho em função da estratégia utilizada, sugerindo que a aleatoriedade, embora vantajosa em certos cenários, pode acarretar custos mais elevados à medida que o tamanho do conjunto de dados aumenta.

Seguindo, tem-se a continuação do gráfico anterior na Figura 11, onde o padrão segue o mesmo, apresentando mudança significativa a partir dos valores do arquivo de 50.000, onde os valores dos dados decrescentes junto aos dados aleatórios, se destoam dos demais.

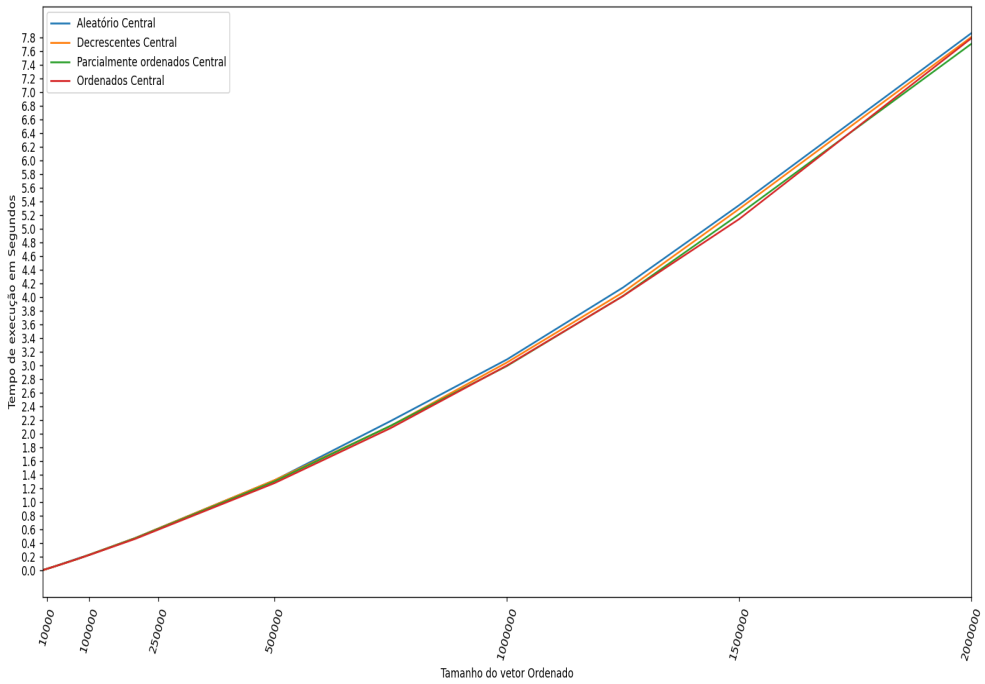

Figura 11. Todos arquivos para o método de pivô central com leitura, intervalo de 10,000 à 2,000,000.

3.3. *Quick Sort* com mediana de três

Na Figura 12, com as médias dos arquivos, sem a leitura, de 100 à 7.500, observa-se uma oscilação nos tempos de execução inciais. Tal como nos outros casos, essa oscilação inicial pode ser ocasionada pela. Porém, após o arquivo com 2.000 entradas, os tempos de execução convergiram para valores próximos, mas o tempo de execução em dados aleatórios permaneceu mais elevado que os outros.

Na Figura 13, apresenta-se a continuação do gráfico anterior, onde tem-se as médias entre os arquivos 10.000 e 2.000.000. Verifica-se que o padrão segue o mesmo do gráfico anterior, sendo evidente um aumento ainda mais significativo nos valores dos dados aleatórios, enquanto o restante dos dados ficaram próximos.

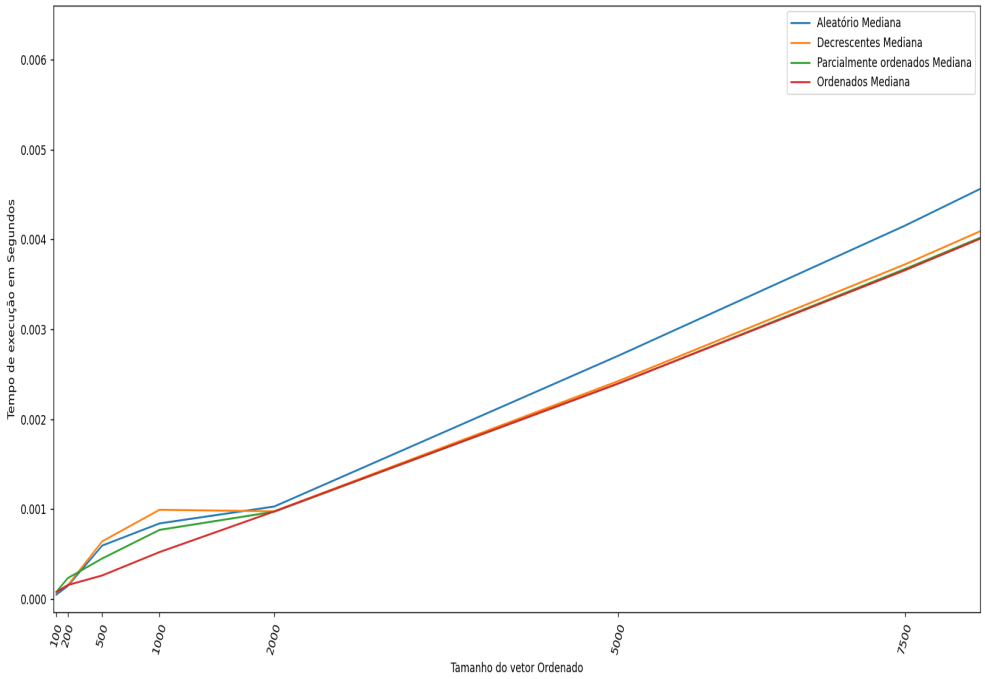

Figura 12. Todos arquivos para o método de pivô Mediana sem leitura, intervalo de 100 à 7,500.

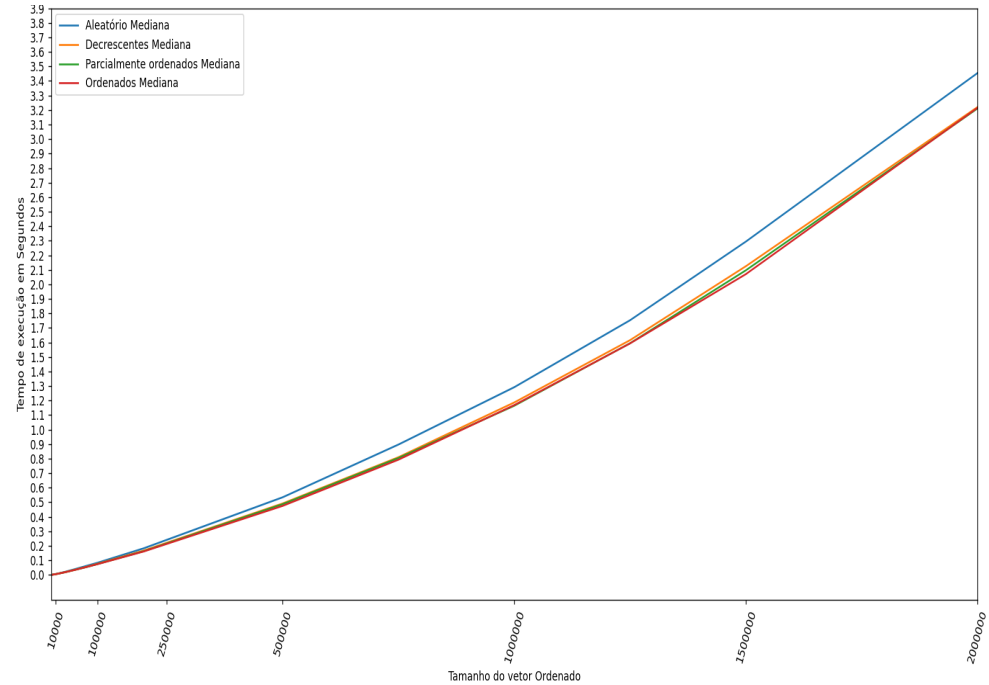

Figura 13. Todos arquivos para o método de pivô Mediana sem leitura, intervalo de 10,000 à 2,000,000.

Nos tempos de execução contando a leitura dos arquivos (Figura 14), nota-se que, inicialmente, os dados apresentam uma leve desorganização. Após o arquivo de 5.000, há um comportamento mais estável nos tempos de execução, sendo os dados aleatórios um pouco mais custosos que os demais.

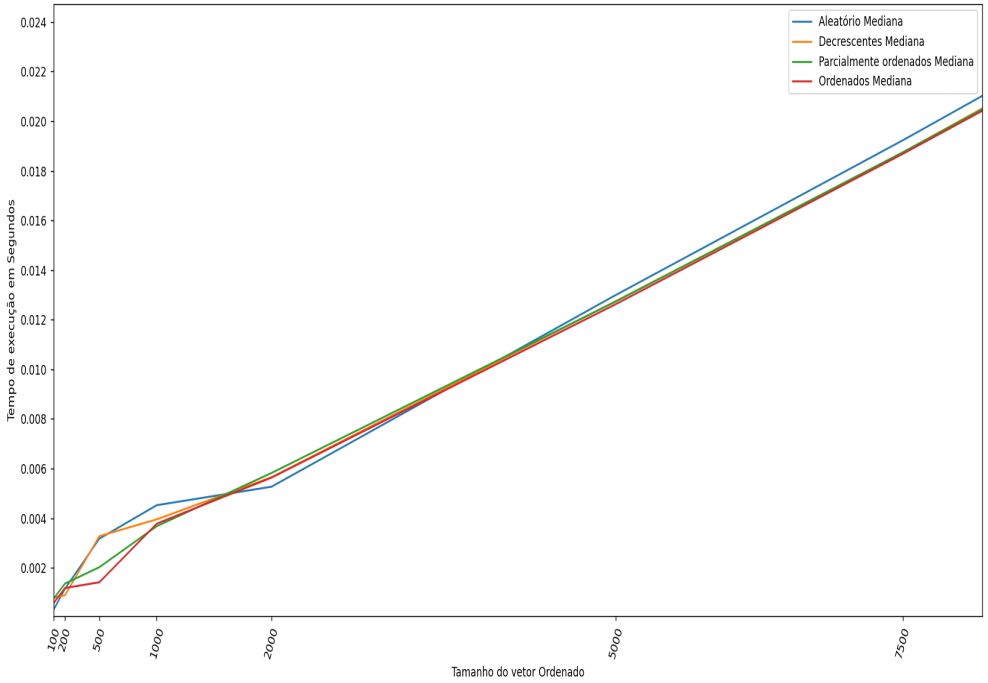

Figura 14. Todos arquivos para o método de pivô Mediana com leitura, intervalo de 100 à 7,500.

3.3.1. Teste com leitura de arquivo

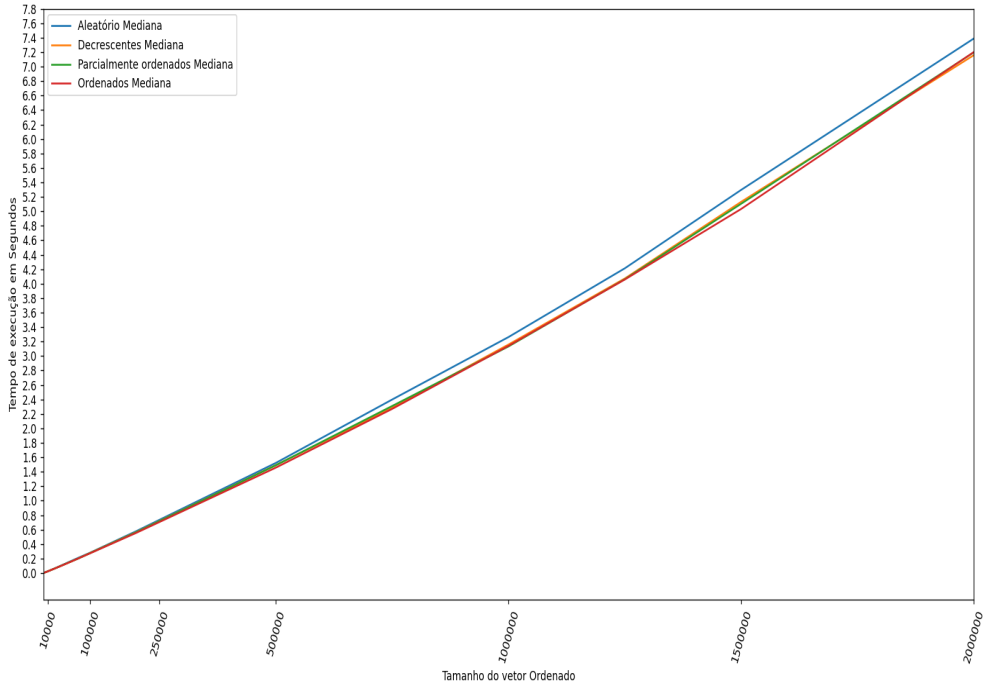

Figura 15. Todos arquivos para o método de pivô Mediana com leitura, intervalo de 10,000 à 2,000,000.

Seguindo, em continuidade ao gráfico anterior, na Figura 15 os dados seguem

o mesmo padrão já visto, somente aumentando recorrentemente o custo para os dados aleatórios, enquanto o custo dos demais se mostram muito próximos.

3.4. Comparação entre os métodos

A comparação entre os métodos abrange as três abordagens de pivô para cada cenário de teste, fornecendo uma análise abrangente do desempenho de cada uma em diferentes condições

3.4.1. Arquivos aleatório sem leitura

Verificando o desempenho de cada método com os dados aleatórios, entre 100 a 7.500, na Figura 16, é possível observar que o método Central tem um desempenho inicial melhor do que os demais métodos, seguido pelo método Aleatório e posteriormente o método Mediana. Este cenário ocorre visto que, inicialmente, o conjunto de dados contém poucos elementos, o que tende a aumentar o custo em métodos que necessitam de cálculos externos a mais, como é o caso da geração do número aleatório, e, do cálculo da mediana. Como o método Central depende apenas de uma divisão de inteiro, este cálculo não impacta significativamente seu desempenho, diferente do método Mediana, onde o ganho de desempenho usando a mediana não é o suficiente para cobrir seu cálculo.

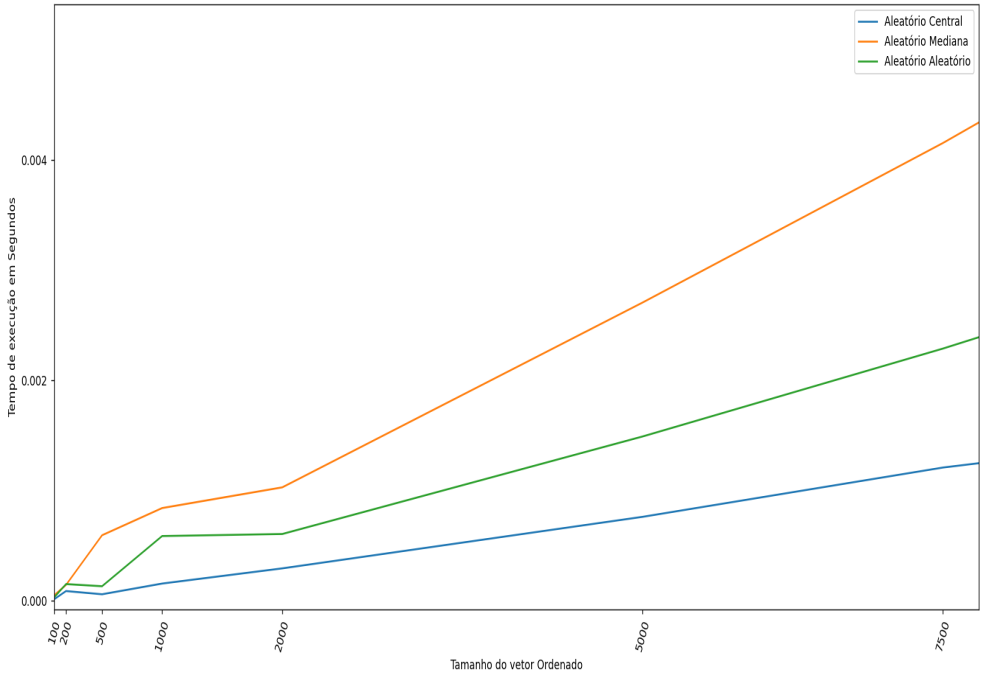

Figura 16. Todos métodos com arquivo de dados aleatório entre 100 à 7.500.

Em sequência, verifica-se os dados, entre 10.000 à 2.000.000, aleatórios. Onde, pela Figura 17, é possível observar que, conforme a quantidade de dados aumenta, melhor fica o desempenho usando o método Mediana, em contraste ao método Central, que, conforme aumento da quantidade de dados, seu desempenho cai, visto ser um método muito simples. No caso do método Aleatório, depende de sorte, onde pode ser escolhido

um bom ou ruim pivô. No caso dos testes feitos neste trabalho, a utilização do método Aleatório teve desempenho significativamente positivo para quantidade dados extensivos.

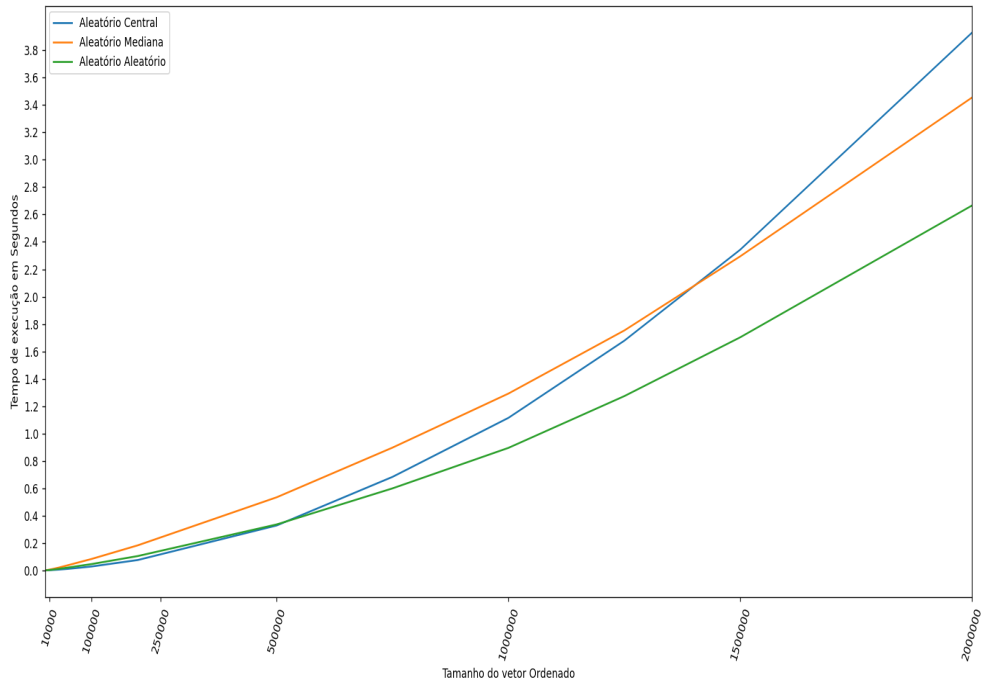

Figura 17. Todos métodos com arquivo de dados aleatório entre 10.000 à 2.000.000.

3.4.2. Arquivos aleatório com leitura

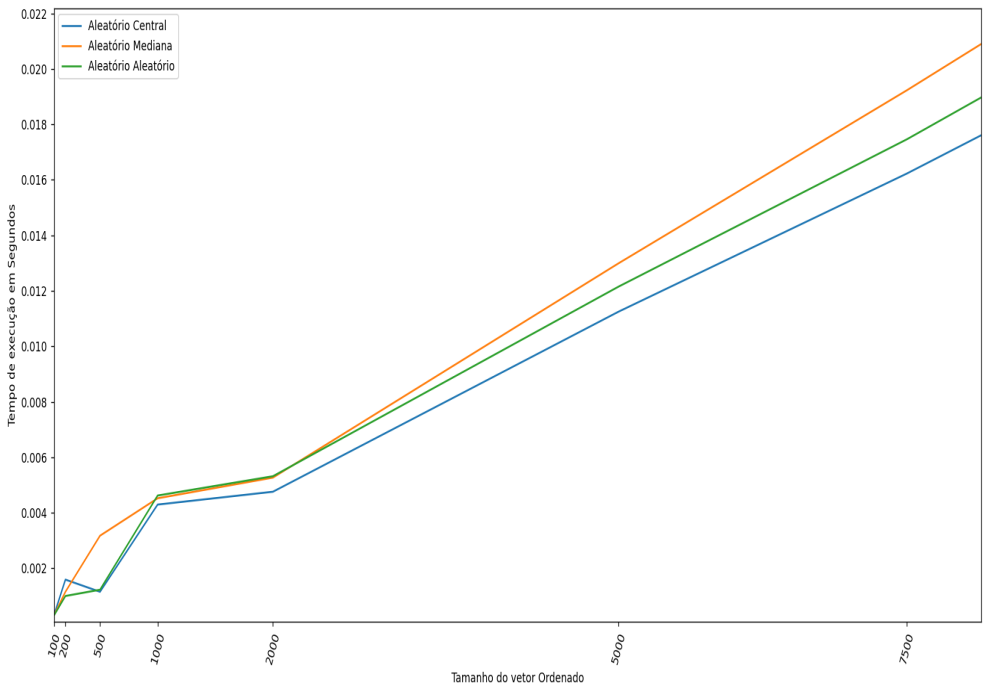

Figura 18. Todos métodos com arquivo de dados aleatório entre 100 à 7.500.

Para os testes realizados levando em conta o tempo de leitura, conforme ilustrado na Figura 18, observa-se que a tendência dos tempos de consumo permanece semelhante àquela identificada anteriormente sem considerar o tempo de leitura, mas agora com o acréscimo correspondente a esse intervalo.

Na sequência, na Figura 19, observa-se que o mesmo padrão é mantido na continuação do gráfico, preservando distribuições semelhantes às observadas anteriormente, sem levar em conta o custo de leitura.

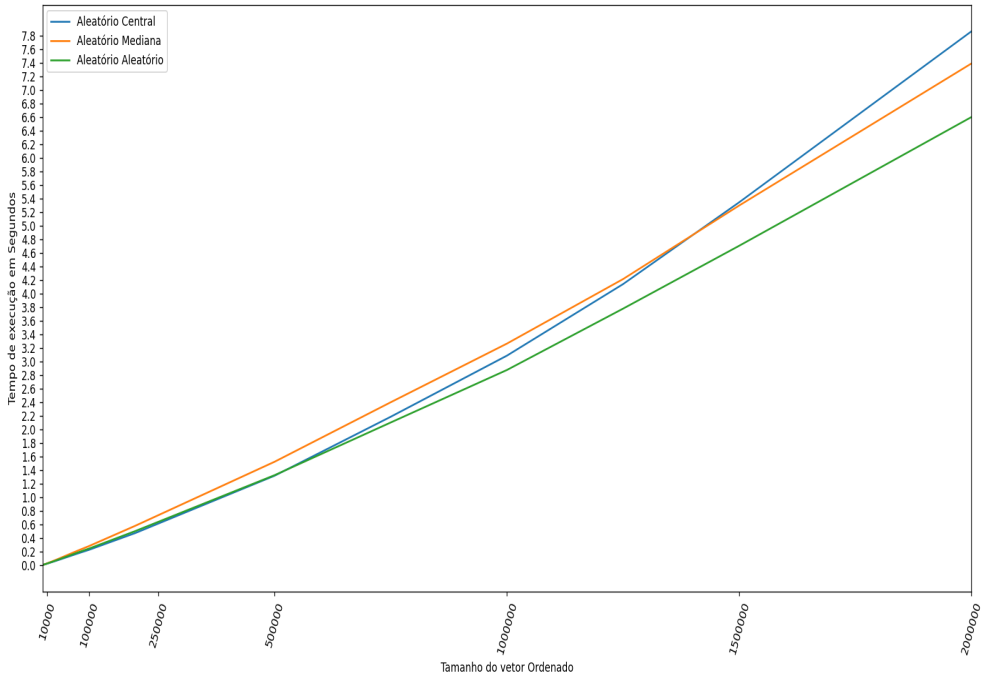

Figura 19. Todos métodos com arquivo de dados aleatório entre 10.000 à 2.000.000.

3.4.3. Arquivos decrescentes sem leitura

Verificando os dados organizados de forma decrescente, ente 100 à 7.500, observa-se pela Figura 20 que segue o mesmo padrão do visto na Figura 16 dos dados aleatórios. Onde, para os dados inicias, o método Central tem vantagem, seguido do método Aleatório, com o método Mediana sendo o menos eficiente.

Em sequência, verifica-se os dados, entre 10.000 à 2.000.000, decrescentes, na Figura 21. Onde ocorre o mesmo esquema já visualizado na Figura 17, seguindo o mesmo comportamento, o método Central se torna cada vez mais caro com o aumento da quantidade de dados, em contraposição, o método Mediana tende a melhorar o desempenho. De mesmo modo, o método Aleatório segue tendo o melhor desempenho, diante do fator de sorte.

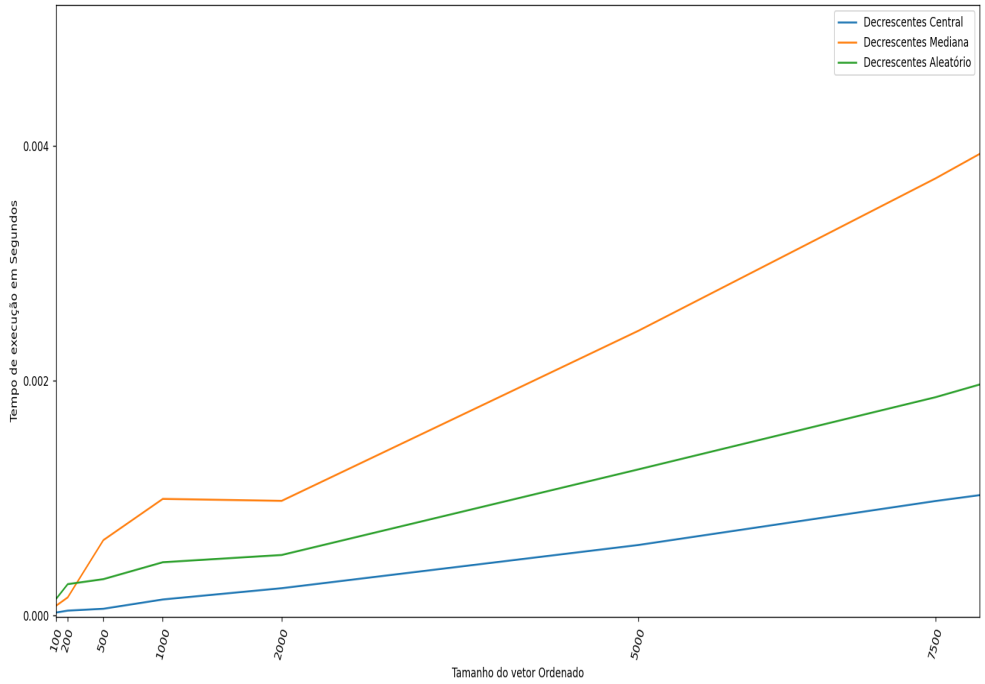

Figura 20. Todos métodos com arquivo de dados decrescentes entre 100 à 7.500.

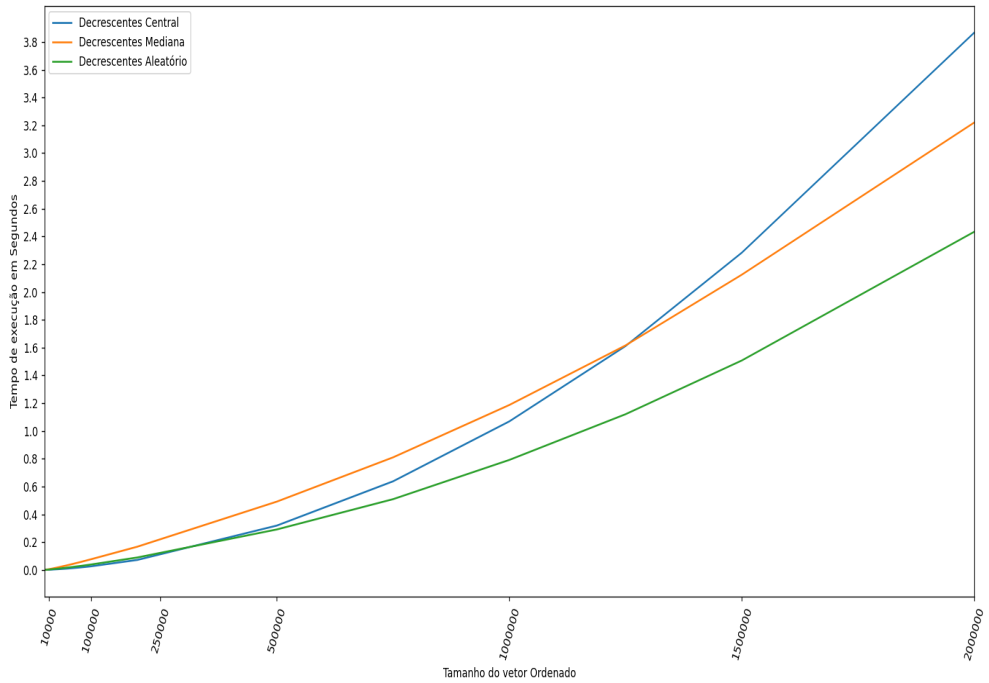

Figura 21. Todos métodos com arquivo de dados decrescentes entre 10.000 à 2.000.000.

3.4.4. Arquivos decrescentes com leitura

Para os testes feitos considerando tempo de leitura como se pode observar na Figura 22, a tendência dos tempos de consumo segue a mesma vista anteriormente sem considerar o

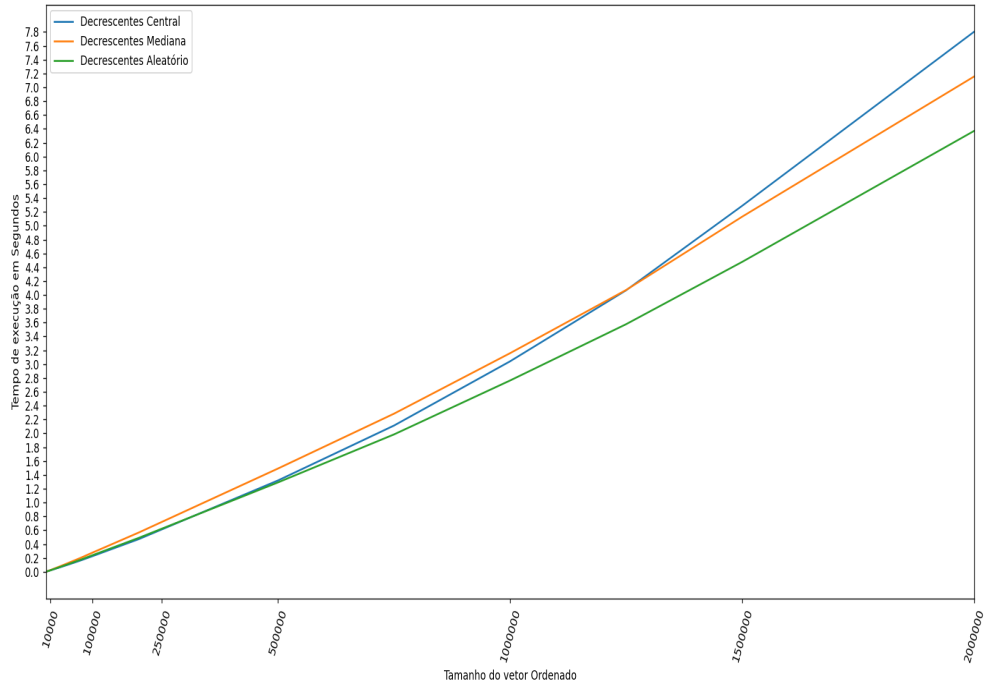

Figura 22. Todos métodos com arquivo de dados decrescentes entre 100 à 7.500.

tempo de leitura, porém agora apenas com um acréscimo deste valor de leitura.

Em sequência, na Figura 23, ainda temos o mesmo padrão para a continuação do gráfico, mantendo as distribuições semelhantes ao visto anteriormente sem considerar o custo de leitura.

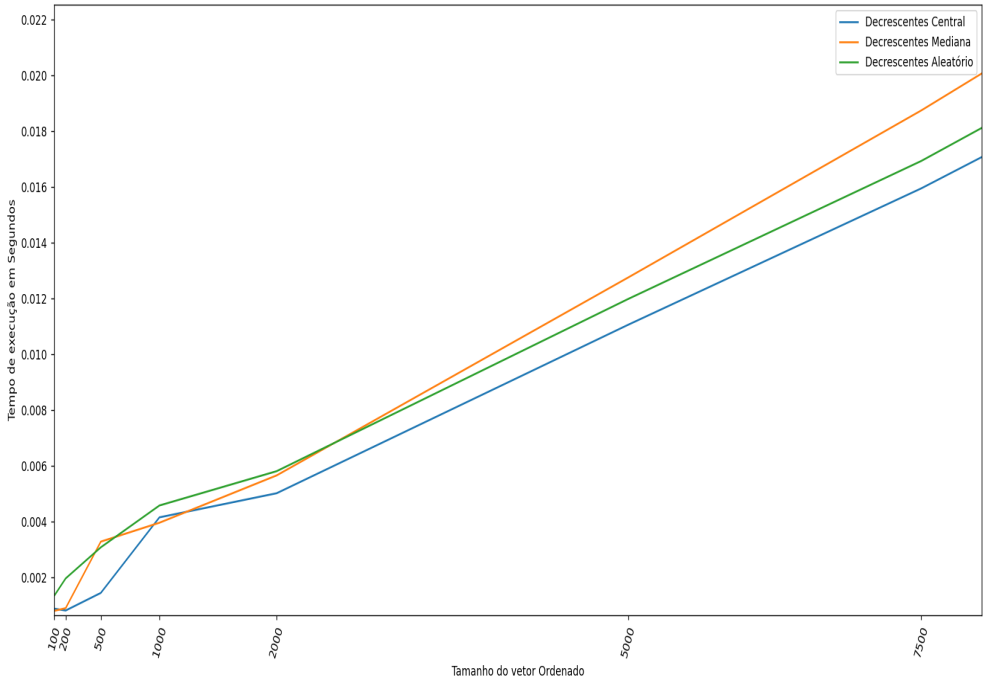

Figura 23. Todos métodos com arquivo de dados decrescentes entre 10.000 à 2.000.000.

3.4.5. Arquivos ordenados sem leitura

Para os arquivos ordenados, considerando o tempo de leitura, observa-se na Figura 24 que os tempos de consumo seguem a mesma tendência observada anteriormente nos testes realizados com arquivos ordenados, sem a consideração do tempo de leitura. No entanto, com a adição do tempo de leitura, todos os métodos analisados são igualmente impactados, resultando em uma influência uniforme sobre os tempos de execução. O método com pivô Central, como nas análises anteriores, continua a apresentar um desempenho superior nas primeiras iterações, destacando-se com tempos de execução mais baixos para entradas menores. No entanto, os métodos com pivô Mediana de Três e Aleatório, que inicialmente apresentam um custo mais elevado devido aos cálculos adicionais e à maior aleatoriedade, acabam se ajustando ao longo do processo. Estes métodos, embora iniciem com um custo de execução mais alto, demonstram um desempenho mais equilibrado à medida que o número de entradas aumenta, especialmente quando comparados com o método Central, que perde eficiência à medida que o tamanho dos dados cresce.

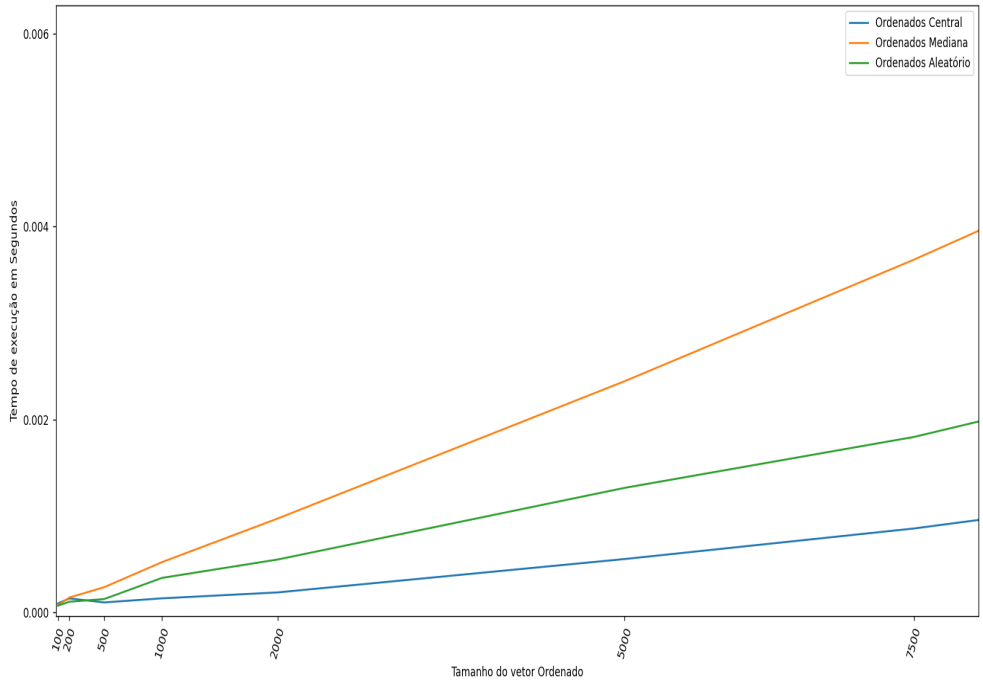

Figura 24. Todos métodos com arquivo de dados ordenados entre 100 à 7.500.

Conforme mostrado na Figura 25, para volumes maiores de dados ordenados, os três métodos seguem o mesmo padrão observado anteriormente, com algumas variações importantes no desempenho à medida que o tamanho dos dados aumenta. O método Central, que inicialmente apresenta um bom desempenho para entradas menores, começa a apresentar uma queda acentuada na sua eficiência conforme o volume de dados cresce, indicando que ele não é a solução mais escalável para grandes conjuntos de dados. Em contraste, o método Mediana de Três melhora progressivamente à medida que o tamanho dos dados aumenta, destacando-se como uma opção mais eficiente para conjuntos de dados maiores, devido à sua capacidade de balancear a aleatoriedade com a estrutura dos dados. Já o método Aleatório, embora continue apresentando uma certa dependência

da sorte, consegue manter um desempenho aceitável mesmo em arquivos extensos. Embora não seja tão consistente quanto a Mediana, o método Aleatório se destaca por sua capacidade de evitar piores casos de desbalanceamento, proporcionando uma solução razoavelmente eficiente em cenários com grandes volumes de dado

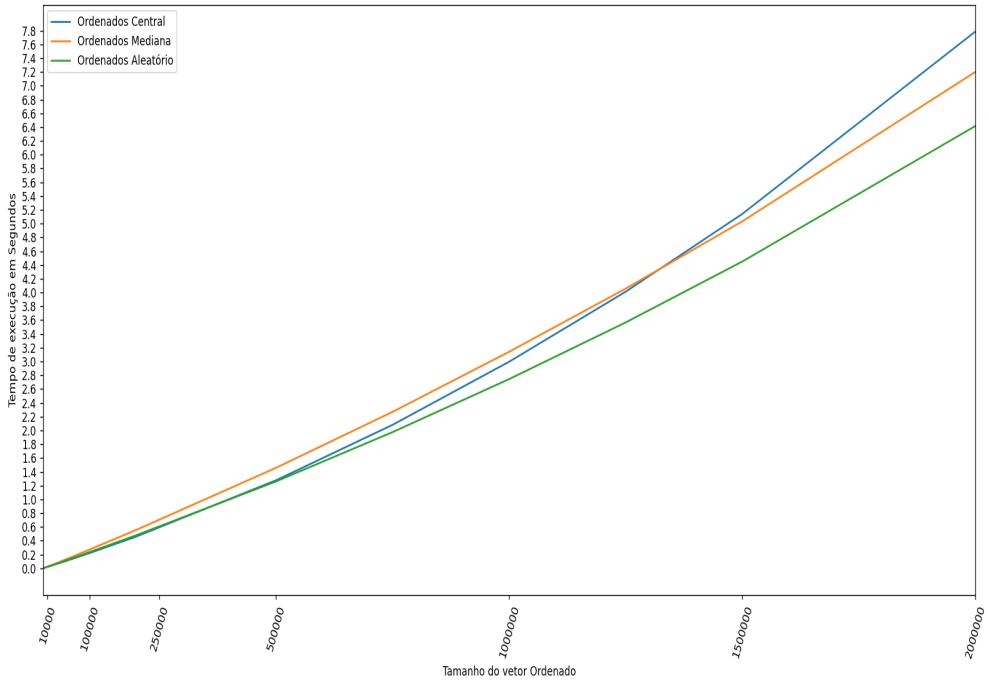

Figura 25. Todos métodos com arquivo de dados ordenados entre 10.000 à 2.000.000.

3.4.6. Arquivos ordenados com leitura

Para os arquivos parcialmente ordenados, sem considerar o tempo de leitura, observa-se na Figura 26 que o desempenho dos métodos segue um padrão intermediário entre os observados para os dados totalmente ordenados e os aleatórios. O método Central, por sua vez, mantém a vantagem inicial devido à simplicidade de seu cálculo, o que lhe confere um desempenho eficiente para volumes menores de dados, já que a escolha do pivô é realizada de forma direta e sem necessidade de cálculos complexos. No entanto, os métodos Aleatório e Mediana, que dependem de um processo mais envolvente de escolha de pivô, apresentam custos mais altos para volumes pequenos, sendo mais impactados pela desorganização parcial dos dados. Essa desorganização, embora não seja tão extrema quanto em dados totalmente aleatórios, dificulta a escolha de um pivô eficiente e afeta o desempenho desses métodos. O método Aleatório, em particular, continua a ser sensível às variações de sorte, enquanto o método Mediana tenta equilibrar essa aleatoriedade, mas não consegue superar o desempenho do método Central em dados de tamanho menor.

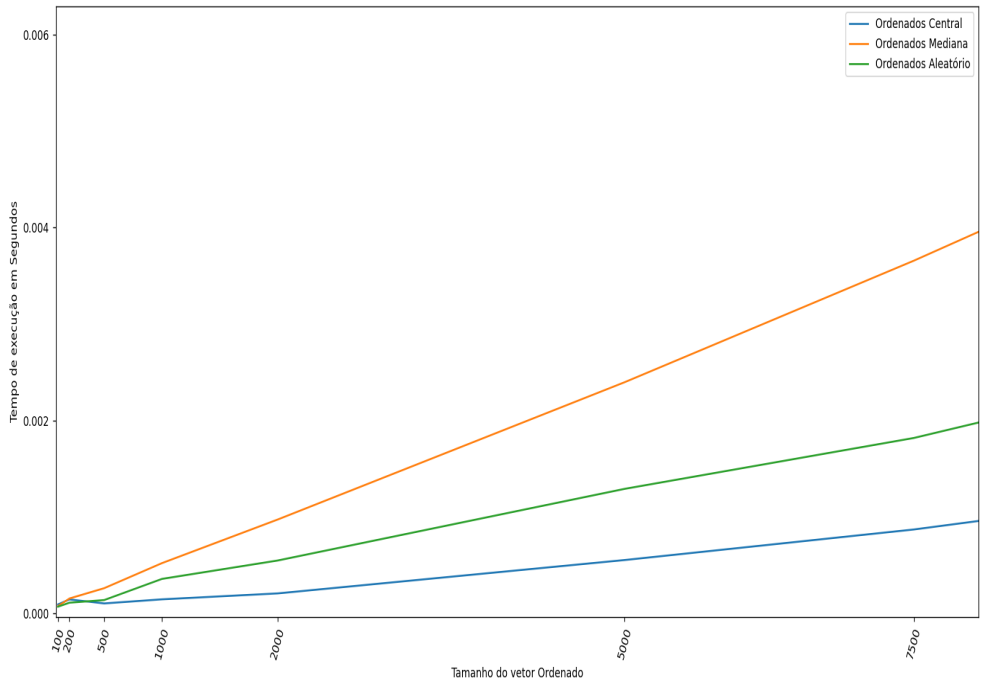

Figura 26. Todos métodos com arquivo de dados ordenados entre 100 à 7.500.

Na Figura 27, observa-se que, à medida que o volume de dados aumenta, o comportamento dos métodos se alinha ao dos gráficos para dados aleatórios e ordenados: o método Mediana apresenta melhor desempenho com grandes volumes, enquanto o método Central perde eficiência devido à simplicidade de sua abordagem. O método Aleatório continua a apresentar desempenho variado, dependente da escolha do pivô.

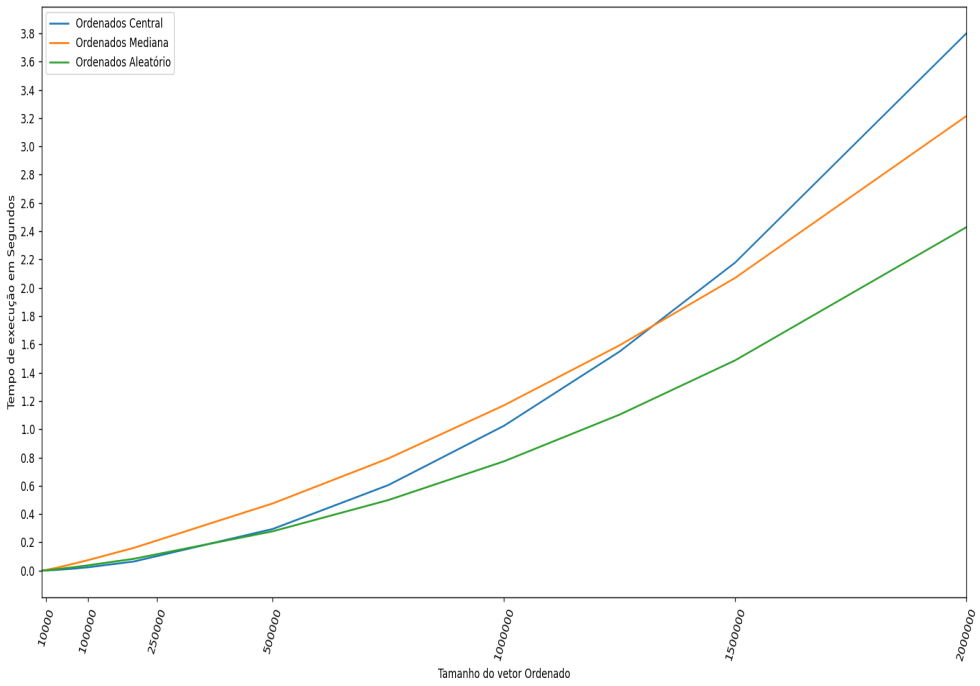

Figura 27. Todos métodos com arquivo de dados ordenados entre 10.000 à 2.000.000.

3.4.7. Arquivos parcialmente ordenados sem leitura leitura

No gráfico apresentado na Figura 28, são mostrados os tempos de execução das três estratégias de ordenação aplicadas a entradas parcialmente ordenadas, variando de 100 até 7.500 elementos. Assim como nos outros cenários analisados, observa-se que este gráfico exibe um comportamento oscilante nos tempos de execução durante a ordenação dos arquivos menores. No entanto, conforme o número de elementos aumenta, essa oscilação tende a diminuir, e os tempos de execução se estabilizam. Nesse contexto, a estratégia da mediana de três se destaca por apresentar o maior custo de execução entre os três métodos, especialmente para as entradas menores, devido à sua necessidade de realizar mais cálculos, como já discutido anteriormente. Por outro lado, a estratégia do pivô aleatório se sobressai ao apresentar o melhor desempenho médio em termos de tempo de execução, destacando-se como a mais eficiente na maior parte dos cenários.

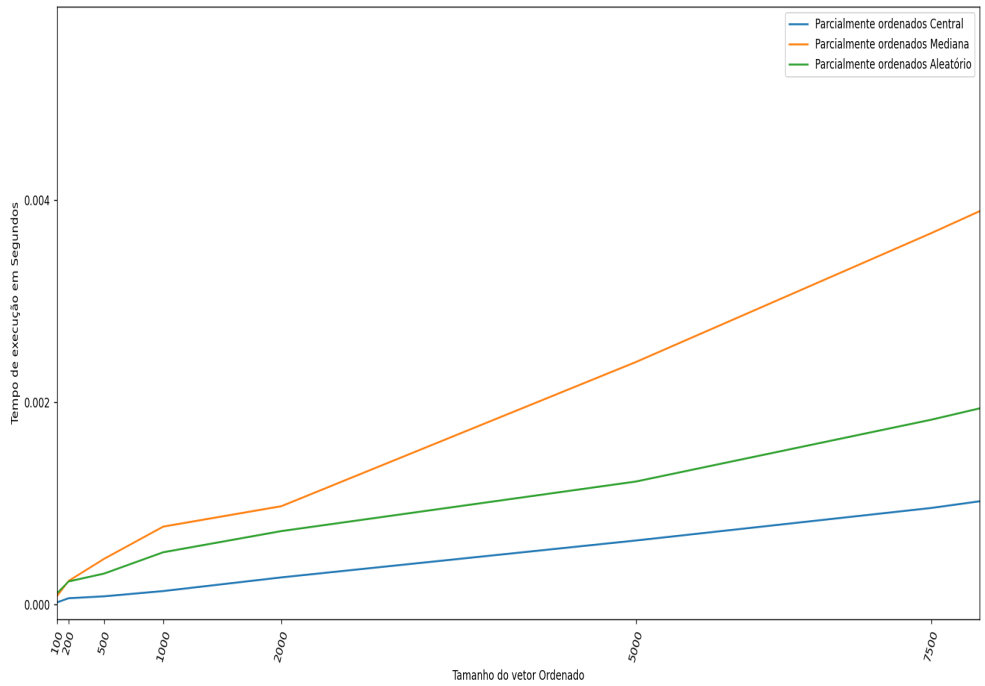

Figura 28. Todos métodos com arquivo de dados decrescentes entre 100 à 7.500.

Já no gráfico apresentado na Figura 29, é possível observar uma tendência semelhante à observada nos testes anteriores, com a estratégia da mediana de três superando a estratégia do pivô central à medida que o tamanho das entradas aumenta. Isso se deve ao fato de que, com volumes maiores de dados, a mediana de três consegue lidar melhor com as distribuições dos elementos, oferecendo um desempenho superior ao pivô central, que tende a piorar à medida que o número de entradas cresce. No entanto, apesar dessa melhoria da mediana de três em relação ao pivô central, a estratégia aleatória continua a se destacar, apresentando tempos de execução médios consistentemente mais baixos que as outras duas abordagens, independentemente do tamanho dos dados.

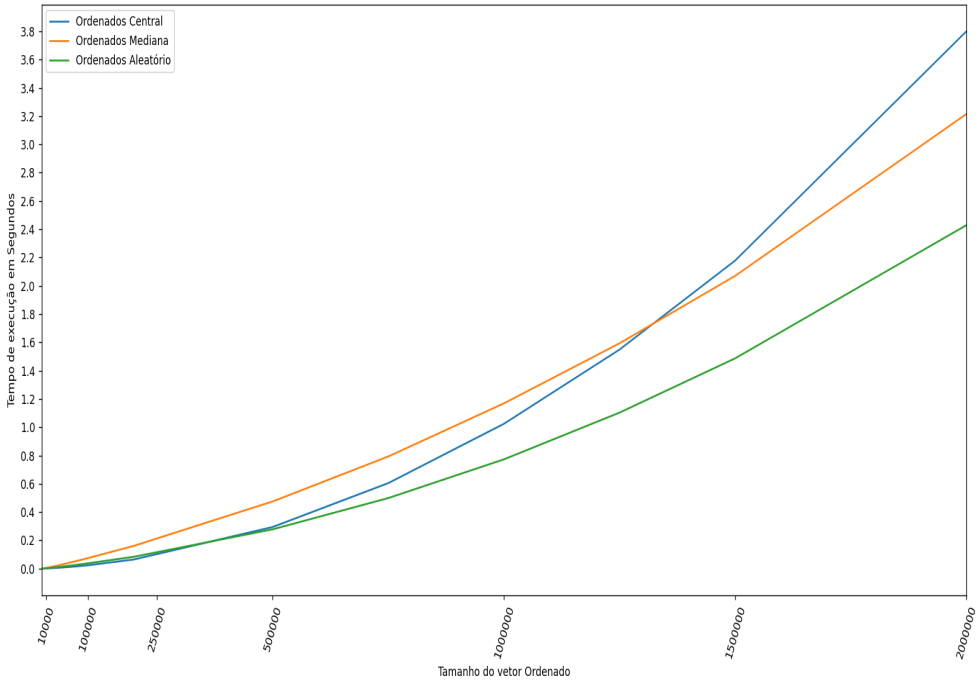

Figura 29. Todos métodos com arquivo de dados decrescentes entre 10.000 à 2.000.000.

3.4.8. Arquivos parcialmente ordenados com leitura leitura

Com o acréscimo do tempo de leitura, como ilustrado na Figura 30, os tempos de consumo para dados parcialmente ordenados mantêm a mesma tendência observada anteriormente nos gráficos sem o tempo de leitura. O impacto do tempo de leitura, embora tenha aumentado o custo total das execuções, se distribui de forma uniforme entre todos os métodos, não alterando significativamente as diferenças de desempenho entre eles. Os métodos, embora afetados de maneira geral pelo aumento no tempo de leitura, continuam a apresentar padrões de desempenho semelhantes aos observados nas execuções anteriores. O método Central, com sua vantagem inicial devido à simplicidade de cálculo, continua a ter um desempenho superior em volumes menores, enquanto os métodos Mediana e Aleatório ainda são mais impactados pela desorganização dos dados, especialmente em volumes pequenos. Apesar do acréscimo do tempo de leitura, as relações de desempenho entre os métodos se mantêm, com a eficiência do método Aleatório dependendo ainda da sorte e o método Mediana mostrando vantagens para volumes maiores de dados.

Na Figura 31, para volumes maiores de dados, o método Mediana se destaca, confirmando seu desempenho superior em relação aos outros métodos. Isso ocorre devido à sua capacidade de lidar de maneira mais eficiente com a desorganização parcial dos dados, o que lhe confere uma vantagem notável em cenários com entradas parcialmente ordenadas. Por outro lado, o método Central, que apresenta um bom desempenho com volumes pequenos, começa a se tornar menos eficiente à medida que o volume de dados aumenta, pois a estratégia determinística se torna menos eficaz para grandes conjuntos de dados. O método Aleatório, embora ainda dependa em parte da sorte, mantém resultados positivos em situações nas quais a escolha do pivô favorece a ordenação do conjunto de

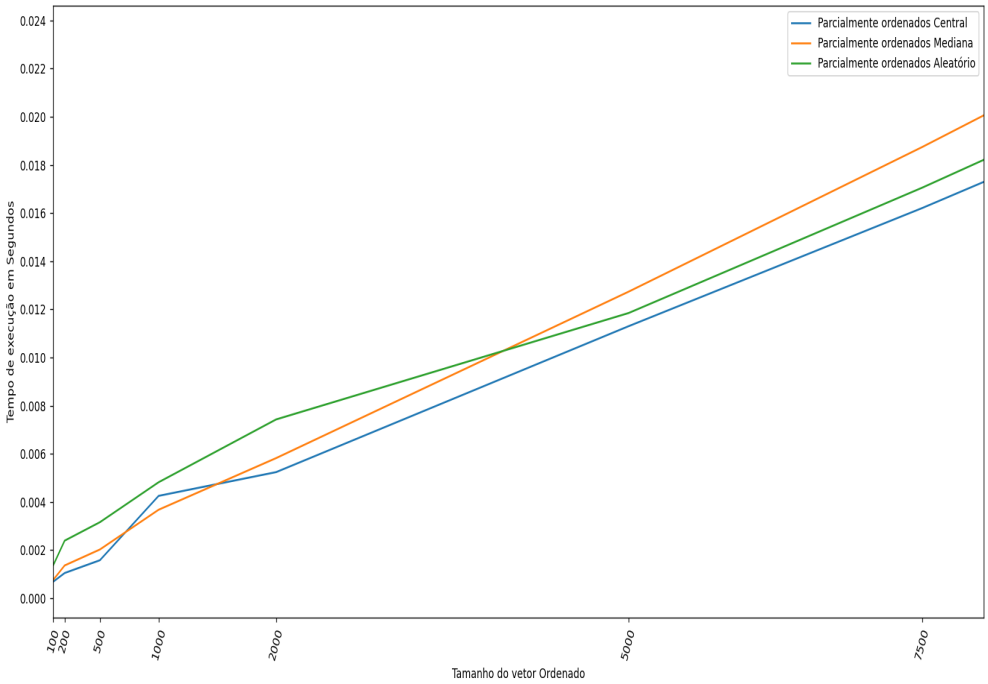

Figura 30. Todos métodos com arquivo de dados ordenados entre 100 à 7.500.

dados. Quando o pivô sorteado se alinha com uma boa escolha, o desempenho do método Aleatório se aproxima dos melhores resultados, especialmente em arquivos maiores, onde o impacto das desorganizações diminui.

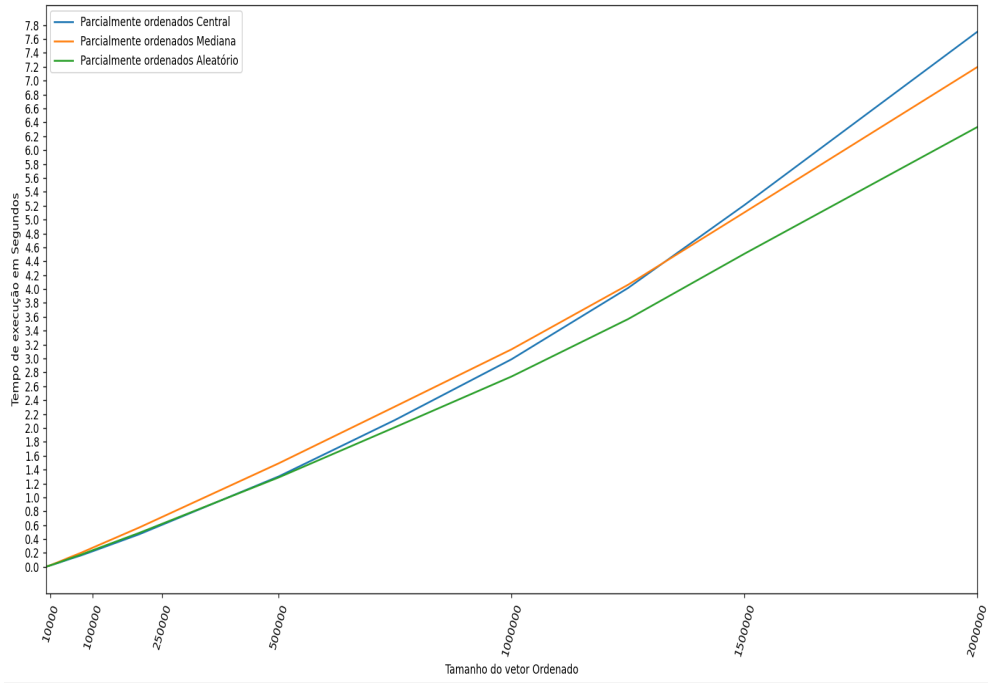

Figura 31. Todos métodos com arquivo de dados ordenados entre 10.000 à 2.000.000.

4. Conclusões

Com base nos resultados obtidos tanto nos cálculos assintóticos quanto nos testes de desempenho, concluímos que a escolha da estratégia de pivô no algoritmo *Quick Sort* tem um impacto significativo no desempenho geral. Ao analisarmos as médias dos tempos de execução das três estratégias de ordenação, observamos que a abordagem do pivô aleatório se destaca como a mais eficiente, apresentando consistentemente os menores tempos de execução em uma ampla gama de casos. Esse comportamento pode ser explicado pelo fator sorte na hora da aleatorização, que ajuda a evitar os piores cenários de desbalanceamento. Além disso, a implementação do pivô aleatório exigiu um número menor de chamadas de função em relação às outras abordagens, o que pode ter contribuído para a redução do tempo de execução, tornando-o mais rápido em cenários de dados variados.

É importante destacar que, ao calcular a média de todas as 5 últimas execuções de cada cenário, os valores piores (resultantes de execuções com pivôs desfavoráveis) foram suavizados, levando a uma representação mais equilibrada do desempenho geral. Esse fator de suavização contribuiu para que o método de pivô aleatório, que inicialmente apresenta variabilidade devido à aleatorização, se mostrasse consistentemente superior na média, especialmente em cenários com dados mais complexos. Assim, o pivô aleatório demonstrou ser a estratégia mais eficiente, embora sua performance dependa do comportamento aleatório das escolhas dos pivôs.

A estratégia da mediana de três acompanha o desempenho do pivô aleatório em muitos cenários, principalmente devido à semelhança nas implementações de ambos os métodos. Nos gráficos, observamos que a curvatura de tempo de execução da mediana de três é muito semelhante à do pivô aleatório, o que sugere que ambos os métodos têm comportamentos de desempenho próximos. No entanto, é importante destacar que a mediana de três se mostra mais custosa, pois utiliza três valores aleatórios para calcular o pivô, ao contrário do pivô aleatório, que utiliza apenas um valor aleatório. Esse acréscimo de cálculos na mediana de três, embora contribua para um desempenho estável em cenários de dados maiores, resulta em um maior custo computacional, o que a torna um pouco menos eficiente em comparação com o pivô aleatório.

Por outro lado, a estratégia do pivô central se mostrou eficiente em casos com entradas menores, alcançando bons tempos de execução e mantendo um desempenho razoável em cenários com dados mais simples. Contudo, à medida que o tamanho dos dados aumenta, o tempo de execução cresce de forma considerável, o que indica que essa estratégia não é a mais escalável para grandes volumes de dados. Como o pivô central sempre escolhe o valor médio, o risco de desequilíbrio aumenta à medida que o tamanho dos dados cresce, fazendo com que o algoritmo seja menos eficiente em grandes entradas. A estratégia da mediana de três, embora tenha apresentado um desempenho inferior ao do pivô aleatório na maioria dos cenários, mostrou-se vantajosa em conjuntos de dados maiores, oferecendo um equilíbrio entre a aleatoriedade do pivô e a estrutura dos dados. Nesse caso, a mediana de três pode fornecer uma escolha de pivô mais representativa, reduzindo os casos extremos. Contudo, mesmo para grandes volumes de dados, ela não conseguiu superar a eficiência do pivô aleatório, que se manteve superior em termos de tempo de execução geral.

Esses resultados ressaltam a importância de uma escolha cuidadosa da estratégia de pivô, levando em consideração o tamanho, a distribuição e as características dos dados,

para otimizar o desempenho do algoritmo *Quick Sort* e garantir uma execução eficiente em diferentes cenários.

Referências

[1] A. R. Backes, *Algoritmos e Estruturas de Dados em Linguagem C*. Rio de Janeiro: LTC, 1 ed., 2023.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Algoritmos: Teoria e Prática*. Rio de Janeiro: Elsevier, 2 ed., 2002. Tradução de: Introduction to Algorithms, 2ª edição americana. Tradução por Vandenberg D. de Souza. 6ª Reimpressão.

[3] S. Chhabra, "Hoare's vs lomuto partition scheme in quicksort," 2024.