Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Apucarana
Bacharelado em Engenharia da Computação

TRABALHO DE TÓPICOS E ANÁLISE DE ALGORITMOS

TRABALHO 1

Algoritmos de ordenação

Maria Eduarda Pedroso[1]

APUCARANA
JULHO / 2024

[1] mpedroso@alunos.utfpr.edu.br

SUMÁRIO

1. INTRODUÇÃO

A ordenação é um problema fundamental em ciência da computação, com uma variedade de algoritmos disponíveis para resolver essa tarefa de maneira eficiente. Neste relatório, abordamos quatro algoritmos de ordenação amplamente utilizados: Insertion Sort, Merge Sort, Quick Sort e Counting Sort. Cada um desses algoritmos possui suas próprias características, vantagens e limitações, que serão exploradas em profundidade ao longo desta análise comparativa.

Uma das principais métricas para avaliar o desempenho de um algoritmo de ordenação é sua complexidade temporal em diferentes cenários: melhor caso, caso médio e pior caso. No melhor caso, o algoritmo apresenta seu desempenho mais eficiente, geralmente quando o conjunto de entrada já está parcial ou totalmente ordenado. Por outro lado, o pior caso representa a situação na qual o algoritmo demanda o máximo de recursos e tempo para realizar a ordenação, frequentemente quando o conjunto de entrada está em uma ordem específica que desafia a estratégia de ordenação utilizada. O caso médio, por sua vez, representa uma média ponderada dos diferentes casos de entrada possíveis.

Neste contexto, cada algoritmo possui suas próprias complexidades temporais para o melhor, médio e pior caso. Compreender essas complexidades é fundamental para selecionar o algoritmo mais apropriado para uma determinada tarefa de ordenação, levando em consideração fatores como o tamanho do conjunto de dados, a distribuição dos elementos e as restrições de tempo e espaço.

Ao analisar e comparar o desempenho desses algoritmos em diferentes cenários, buscamos fornecer insights valiosos que podem orientar a escolha do algoritmo mais adequado para uma variedade de aplicações práticas. A partir dessa compreensão, os profissionais de ciência da computação podem tomar decisões informadas e otimizar o desempenho de seus sistemas em relação à ordenação de dados.

2. DESCRIÇÃO DO PROBLEMA

Ordenação é uma operação fundamental em ciência da computação, amplamente utilizada em uma variedade de contextos, desde a organização de dados em bancos de dados até a classificação de resultados em algoritmos de busca. O problema consiste em rearranjar uma coleção de itens de acordo com uma ordem específica, geralmente crescente ou decrescente.

Para resolver esse problema, diversos algoritmos foram desenvolvidos ao longo do tempo, cada um com suas próprias estratégias e eficiências. Neste relatório, focamos em quatro desses algoritmos:

- Insertion Sort: Um algoritmo simples que percorre a lista de itens e, para cada elemento, o insere na posição correta em uma sublista ordenada.
- Merge Sort: Um algoritmo de ordenação dividir para conquistar, que divide repetidamente a lista em sublistas menores, as ordena e as funde para produzir a lista ordenada final.
- Quick Sort: Outro algoritmo dividir para conquistar que seleciona um elemento como pivô, rearranja os elementos ao redor do pivô de forma que os menores fiquem antes e os maiores depois, e então recursivamente ordena as sublistas resultantes.
- Counting Sort: Um algoritmo de ordenação não-comparativo adequado para ordenar números inteiros dentro de um intervalo específico.

Cada um desses algoritmos aborda o problema de ordenação de maneira diferente, e cada um possui suas próprias complexidades e eficiências. Ao analisar e comparar o desempenho desses algoritmos, podemos obter insights valiosos sobre como selecionar a abordagem mais adequada para diferentes conjuntos de dados e requisitos de desempenho.

3. DESENVOLVIMENTO

Para desenvolver esse projeto inicialmente foi definido um código base paralelo, para apenas trocar a função de ordenação e caso necessário poucas modificações, mantendo sempre as comparações de melhor caso, caso médio e pior caso. Vamos analisar o código para entender como ele funciona e como os experimentos são conduzidos.

O código começa importando as bibliotecas necessárias, incluindo time para medir o tempo de execução, random para gerar números aleatórios, matplotlib.pyplot para visualização de dados e multiprocessing.Pool para processamento paralelo.

```python
import time
import random
import matplotlib.pyplot as plt
from multiprocessing import Pool
```

A variável tamanhos define os diferentes tamanhos de vetores que serão testados durante o experimento.

```python
tamanhos = [50000, 100000, 150000, 200000, 250000, 300000]
```

A função executar_experimento recebe como entrada um caso (melhor, médio ou pior) e um tamanho de vetor. Ela executa o algoritmo de ordenação várias vezes para o mesmo caso e tamanho, registrando o tempo de execução de cada execução. Os tempos de execução são armazenados em um arquivo de texto chamado execussao.txt.

```python
def executar_experimento(caso_tamanho):
    caso, tamanho = caso_tamanho
    tempos = []
    with open('execussao.txt', 'a+') as f:
        for _ in range(10):
            if caso == 'melhor':
                arr = gerar_arranjo_ordenado(tamanho)
            elif caso == 'medio':
                arr = gerar_arranjo_aleatorio(tamanho)
            elif caso == 'pior':
                arr = gerar_arranjo_ordenado(tamanho, decrescente=True)
            tempo_inicio = time.time()
            metodo_sort(arr)
            tempo_fim = time.time()
            tempo_execucao = tempo_fim - tempo_inicio
            tempos.append(tempo_execucao)
             print(f"Tamanho: {tamanho}, Caso {caso} Tempo médio de execução: {tempo_execucao} s")
                  f.write(f"Tamanho: {tamanho}, Caso {caso} Tempo medio de execucao: {tempo_execucao} s\n")
    return caso, tamanho, sum(tempos) / len(tempos)
```

A função tempo_execucao_medio calcula o tempo médio de execução para cada caso e tamanho de vetor, com base nos resultados dos experimentos.

```python
def tempo_execucao_medio(resultados):
    medias = {}
    for caso, tamanhos in resultados.items():
        medias[caso] = {}
        for tamanho, tempos in tamanhos.items():
            medias[caso][tamanho] = sum(tempos) / len(tempos)
```

```
    return medias
```

As funções plotar_resultado e plotar_resultados são responsáveis por visualizar os resultados dos experimentos. Elas utilizam a biblioteca matplotlib para gerar gráficos mostrando o tempo médio de execução em função do tamanho do vetor, para cada caso e para todos os casos, respectivamente.

A função principal orquestra todo o experimento. Ela utiliza processamento paralelo para executar os experimentos para cada caso e tamanho de vetor de forma eficiente. Os resultados são armazenados em um arquivo de texto chamado output.txt, que contém o tempo médio de execução para cada caso e tamanho de vetor.

```python
def principal():
    casos = ['melhor', 'medio', 'pior']
    pool = Pool()
    resultados = {}
    for caso in casos:
        resultados[caso] = {}
        resultados_temp = pool.map(executar_experimento, [(caso, tamanho) for tamanho in tamanhos])
        for r in resultados_temp:
            resultados[r[0]][r[1]] = [r[2]]  # Guardar apenas o tempo médio de execução
    with open('output.txt', 'w') as f:
        f.write("Resultados do experimento:\n")
        for caso, data in tempo_execucao_medio(resultados).items():
            f.write(f"Caso: {caso}\n")
            for tamanho, tempo in data.items():
                f.write(f"Tamanho: {tamanho}, Tempo medio de execucao: {tempo:.6f} s\n")
```

```
for caso, dados in tempo_execucao_medio(resultados).items():
    plotar_resultado(caso, dados)
plotar_resultados(tempo_execucao_medio(resultados))
```

Por fim, a condição if __name__ == "__main__": garante que o código seja executado apenas quando o script Python for executado diretamente, não quando for importado como um módulo. A função principal é então chamada para iniciar o experimento.

3.1. Insertion sort

O algoritmo Insertion Sort é um método de ordenação simples e intuitivo que opera construindo uma lista ordenada um item de cada vez, movendo cada item para a posição correta no momento em que é inserido. Abaixo temos uma explicação de como ele funciona:

- Iteração sobre a lista: O algoritmo começa iterando sobre a lista a ser ordenada, começando pelo segundo elemento (índice 1) até o último elemento.
- Seleção do elemento atual: Para cada elemento na lista, o algoritmo seleciona o elemento atual e o armazena em uma variável temporária chamada de key.
- Comparação e movimentação: Em seguida, o algoritmo começa a comparar o elemento atual com os elementos anteriores na lista, começando pelo elemento imediatamente anterior (índice j). Enquanto o elemento anterior for maior que o elemento atual (key) e j for maior ou igual a zero, o algoritmo move o elemento anterior uma posição à frente na lista, abrindo espaço para inserir o elemento atual.

- Inserção do elemento atual: Após encontrar a posição correta para o elemento atual, o algoritmo insere o elemento atual na posição j + 1, finalizando a iteração atual.

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
```

Para analisar o melhor, pior e caso médio utilizamos apenas a ordenação dos vetores, sendo o melhor caso ele ordenado, o pior ordenado porém decrescente e para o médio apenas geramos um vetor aleatório.

```python
            if caso == 'melhor':
                arr = gerar_arranjo_ordenado(tamanho)
            elif caso == 'medio':
                arr = gerar_arranjo_aleatorio(tamanho)
            elif caso == 'pior':
                arr = gerar_arranjo_ordenado(tamanho, decrescente=True)
```

3.2. Merge sort

O algoritmo Merge Sort é um método de ordenação baseado na estratégia de "dividir para conquistar". Ele funciona da seguinte forma:

- Divisão da lista: O algoritmo começa dividindo a lista original em duas metades aproximadamente iguais, até que cada sublista tenha apenas um elemento. Isso é feito recursivamente até que não seja possível mais dividir a lista.
- Ordenação das sublistas: Em seguida, o algoritmo começa a mesclar as sublistas ordenadas em uma única lista ordenada. Ele faz isso comparando os elementos das sublistas e selecionando o menor (ou maior) elemento em cada iteração, movendo-o para a lista de saída.
- Fusão (Merge): O processo de fusão (merge) envolve comparar os elementos das duas sublistas e selecionar o menor (ou maior) elemento. Esse elemento selecionado é então adicionado à lista de saída. Esse processo continua até que todas as sublistas tenham sido completamente mescladas.
- Combinando as sublistas: Finalmente, o algoritmo combina todas as sublistas mescladas em uma única lista ordenada.

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
```

```
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

Para definir os casos basicamente seguimos a lógica do algoritmo anterior.

3.3. Quick Sort

O algoritmo QuickSort é uma técnica de ordenação baseada na divisão e conquista, que organiza os elementos de uma lista ao dividir a lista em sublistas menores e ordenar essas sublistas de forma recursiva. A seguir, está uma explicação detalhada de como o QuickSort funciona.

- Escolha do Pivô: Seleciona um elemento da lista como pivô. Este elemento pode ser o primeiro, o último, o elemento do meio ou qualquer outro.
- Particionamento: Reorganiza a lista de modo que todos os elementos menores que o pivô fiquem à esquerda e todos os elementos maiores fiquem à direita.
- Recursão: Aplica recursivamente o mesmo processo às sublistas à esquerda e à direita do pivô.

```python
def QuickSort(X, IniVet, FimVet, caso):
    i = IniVet
    j = FimVet

    if caso == 'pior':
        pivo = X[IniVet]
    else:
        pivo = X[(IniVet + FimVet) // 2]

    while(i <= j):
        while (X[i] < pivo):
            i += 1
        while (X[j] > pivo):
            j -= 1
        if (i <= j):
            aux = X[i]
            X[i] = X[j]
            X[j] = aux
            i += 1
            j -= 1
    if (IniVet < j):
        QuickSort(X, IniVet, j, caso)
    if (i < FimVet):
        QuickSort(X, i, FimVet, caso)
```

Para os casos devemos lidar principalmente com o pivô, para o pior caso usamos um pivô no início do vetor, sendo esse ordenado crescente. Para o caso médio e o melhor usamos o pivô no meio do array, mas o vetor do caso médio é um vetor aleatório.

3.4. Counting Sort

O algoritmo Counting Sort é uma técnica de ordenação não-comparativa que opera contando o número de ocorrências de cada elemento em um intervalo específico e, em seguida, usando essas contagens para determinar a posição correta de cada elemento na lista ordenada. Aqui está uma breve explicação de como ele funciona:

- Inicialização: O algoritmo começa inicializando uma lista de contagem C com zeros e uma lista de saída B também com zeros. Ele determina o valor máximo (k) presente na lista de entrada A.
- Contagem de ocorrências: O algoritmo percorre a lista de entrada A e conta o número de ocorrências de cada elemento, armazenando essas contagens na lista de contagem C.
- Atualização das contagens: Em seguida, o algoritmo atualiza a lista de contagem C para indicar a posição correta de cada elemento na lista ordenada B. Isso é feito somando os valores acumulados na lista de contagem C.
- Construção da lista ordenada: Por fim, o algoritmo percorre a lista de entrada A da direita para a esquerda. Para cada elemento, ele encontra sua posição correta na lista ordenada B usando as contagens na lista de contagem C e, em seguida, decrementa a contagem desse elemento em C.
- Retorno da lista ordenada: Após concluir o processo de construção da lista ordenada B, o algoritmo retorna essa lista como resultado.

```python
def countingSort(A):
    n = len(A)
    k = max(A)  # descobre o valor máximo no vetor A
```

```
    C = [0] * (k + 1)  # Inicializa a lista de contagem com zeros
    B = [0] * n  # Inicializa a lista de saída com zeros

    # Conta a ocorrência de cada elemento em A
    for j in range(n):
        C[A[j]] += 1

    # Atualiza C para indicar a posição correta de cada elemento na saída
    for i in range(1, k + 1):
        C[i] += C[i - 1]

    # Monta o array ordenado B
    for j in range(n - 1, -1, -1):
        B[C[A[j]] - 1] = A[j]
        C[A[j]] -= 1

    return B
```

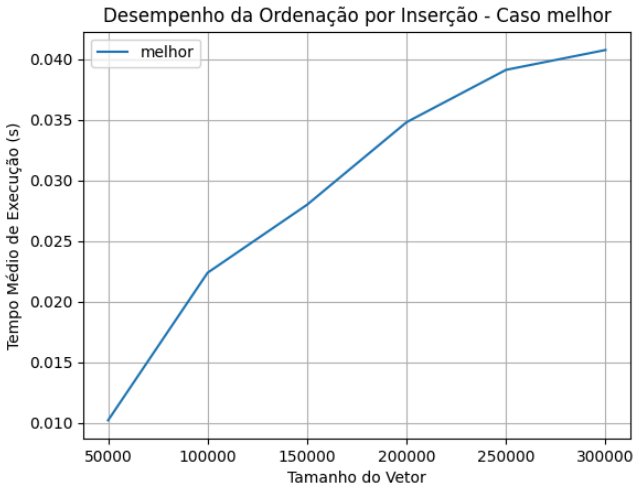
Os casos funcionam igualmente aos dois primeiros, sendo o melhor um vetor ordenado, o pior ordenado decrescente e o médio um vetor aleatório.

4. RESULTADOS

4.1. Insertion sort

No melhor caso, o Insertion Sort apresenta um crescimento linear no tempo de execução com o aumento do tamanho da entrada. Este cenário corresponde a uma lista que já está ordenada, resultando em uma complexidade de O(n). O tempo de execução é o mais eficiente neste caso, refletindo o mínimo número de operações necessárias para verificar a ordem dos elementos.

Figura 4.1.1 - Melhor Caso



Fonte: Autoria própria.

No caso médio, o tempo de execução aumenta de forma quadrática, refletindo a complexidade O(n²). Este cenário representa uma lista com elementos dispostos aleatoriamente. O crescimento quadrático é evidenciado pela rápida elevação dos tempos de execução com o aumento do tamanho da entrada.

Figura 4.1.2 - Caso Médio

Fonte: Autoria própria.

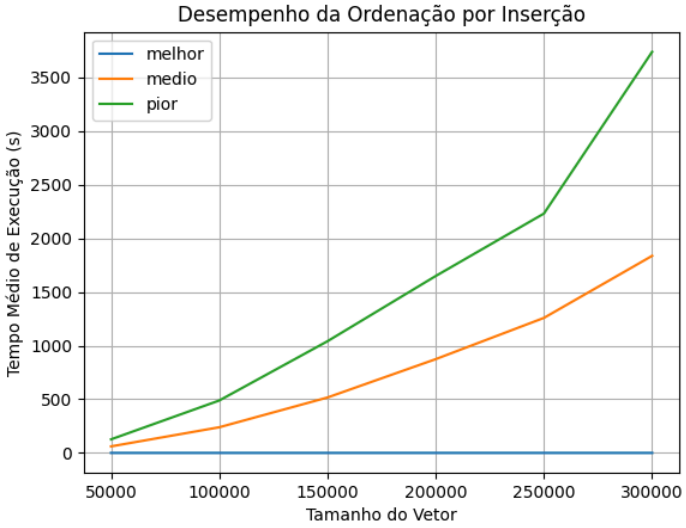
No pior caso, que corresponde a uma lista ordenada em ordem inversa, o tempo de execução também segue uma tendência quadrática ($O(n^2)$). Este cenário requer o máximo número de trocas e comparações, resultando nos tempos de execução mais altos entre os três casos.

Figura 4.1.3 - Pior Caso

Fonte: Autoria própria.

Os resultados do experimento confirmam a natureza dual do Insertion Sort: excelente desempenho em cenários quase ordenados (melhor caso) com complexidade $O(n)$, e desempenho pobre em listas desordenadas ou inversamente ordenadas (caso médio e pior caso) com complexidade $O(n^2)$. Esta análise destaca que o Insertion Sort pode ser útil para pequenas listas ou listas quase ordenadas, mas não é adequado para grandes listas desordenadas onde outros algoritmos, como Merge Sort ou Quick Sort, seriam mais eficientes.

Figura 4.1.4 - Todos os Casos

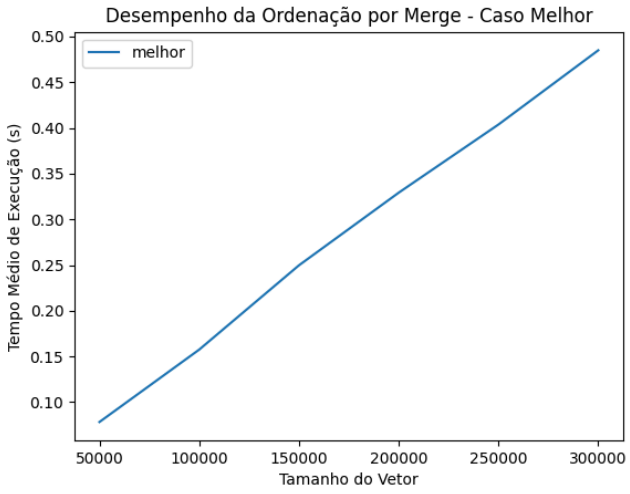


Fonte: Autoria própria.

4.2. Merge sort

Para o melhor caso, o tempo de execução aumenta de forma consistente com o aumento do tamanho da entrada. Isso é esperado, pois mesmo no melhor caso, o Merge Sort mantém sua complexidade de tempo O(n log n). O crescimento parece linear em relação ao número de elementos, mas isso é devido ao fator logarítmico ser menos pronunciado para entradas maiores.

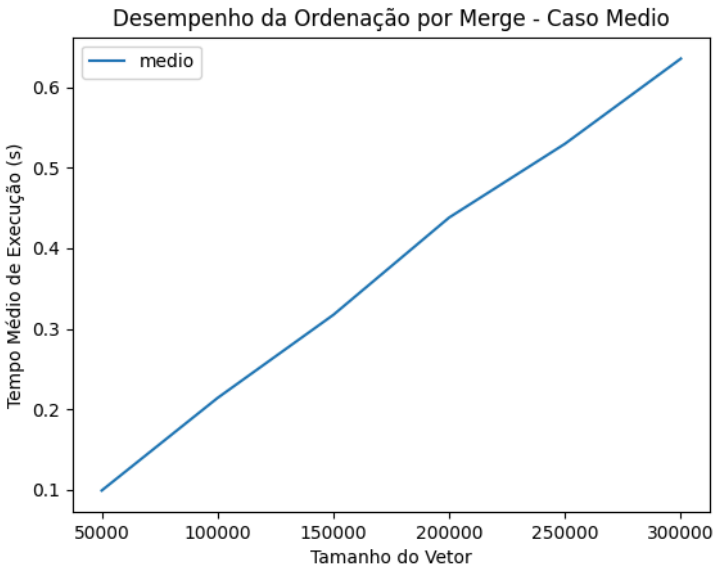
Figura 4.2.1 - Melhor Caso



Fonte: Autoria própria.

Os tempos de execução no caso médio também aumentam de forma consistente, refletindo a mesma complexidade O(n log n). Comparado ao melhor caso, os tempos são ligeiramente maiores, mas ainda mostram uma tendência de crescimento semelhante.

Figura 4.2.2 - Caso Médio



Fonte: Autoria própria.

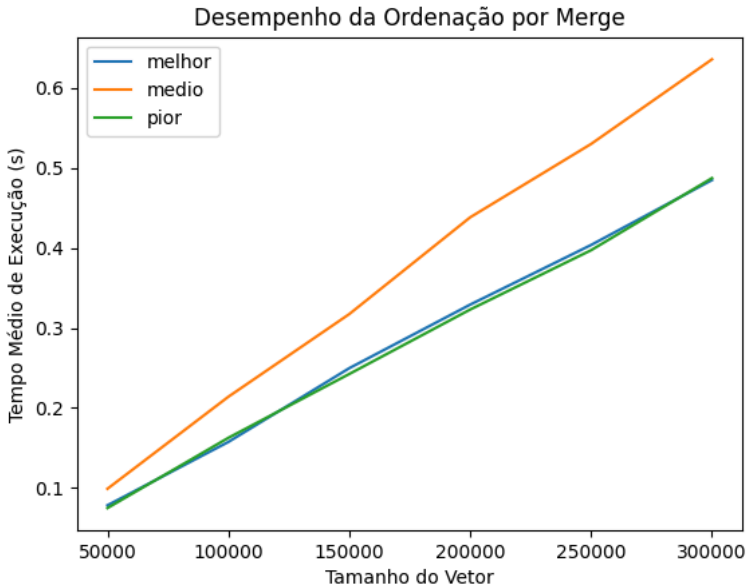
No pior caso, os tempos de execução são ligeiramente maiores que no melhor e médio caso para as menores entradas, mas a diferença tende a diminuir para tamanhos maiores. Isso é consistente com a complexidade O(n log n), pois o Merge Sort é eficiente em lidar com qualquer tipo de entrada, incluindo a mais desordenada.

Figura 4.2.3 - Pior Caso

Fonte: Autoria própria.

Os resultados do experimento confirmam a teoria de que o Merge Sort possui uma complexidade de tempo O(n log n) em todos os casos (melhor, médio e pior). As variações observadas entre os diferentes casos são pequenas, o que destaca a eficiência e a estabilidade do algoritmo. Essa propriedade faz do Merge Sort uma excelente escolha para aplicações que exigem um comportamento de tempo previsível e eficiente, independentemente da ordenação inicial dos dados.

Figura 4.2.4 - Todos os Casos

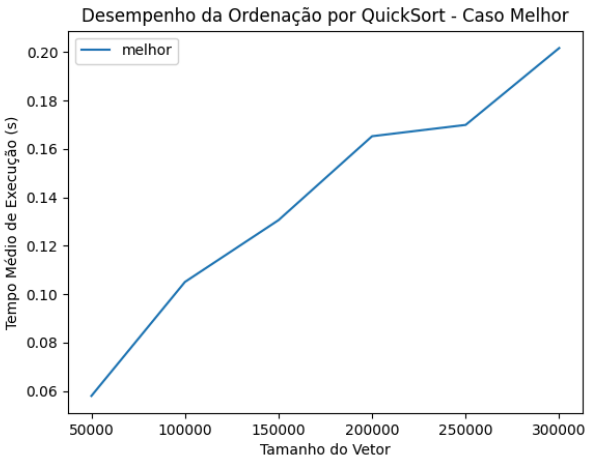


Fonte: Autoria própria.

4.3. Quick Sort

No melhor caso, o tempo de execução cresce de forma sub-linear conforme o tamanho da entrada aumenta. Isso está de acordo com a complexidade esperada de O(n log n) para o melhor caso do Quick Sort, onde a escolha dos pivôs é ótima e divide o conjunto de dados de forma equilibrada.

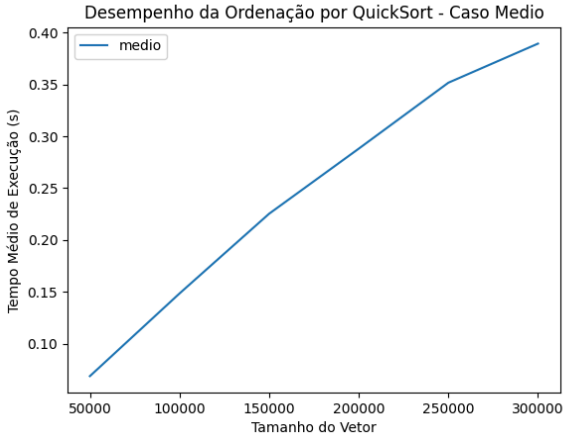
Figura 4.3.1 - Melhor Caso



Fonte: Autoria própria.

No caso médio, o tempo de execução também segue um crescimento sub-linear, refletindo a complexidade O(n log n). Os tempos são maiores que no melhor caso, mas a diferença não é significativa, mostrando a eficiência do Quick Sort em cenários comuns.

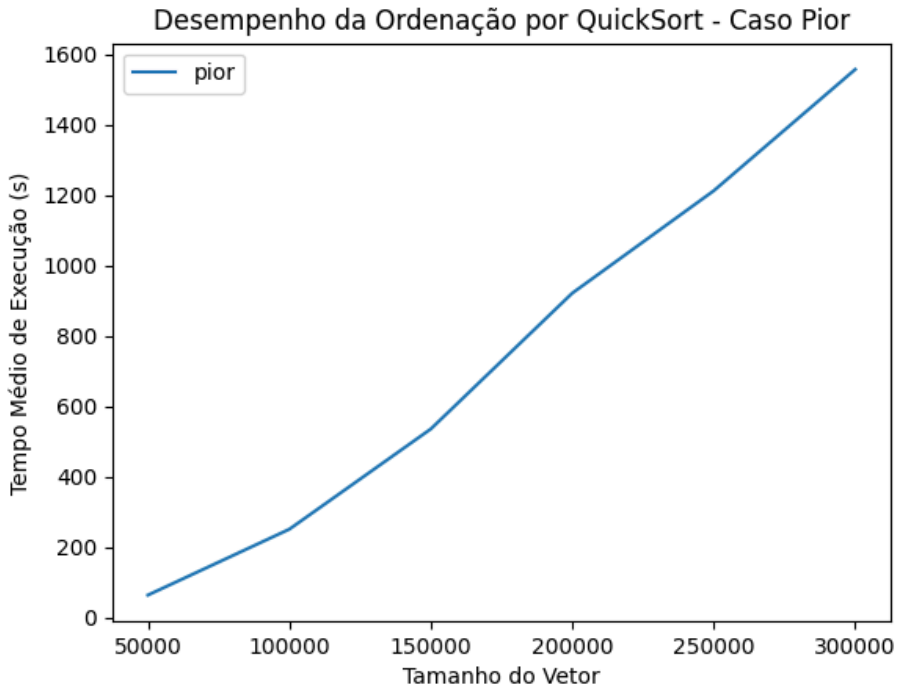
Figura 4.3.2 - Caso Médio



Fonte: Autoria própria.

No pior caso, o tempo de execução cresce drasticamente, seguindo uma complexidade de O(n²). Isso ocorre quando os pivôs escolhidos são os menores ou maiores elementos do conjunto, resultando em partições altamente desequilibradas e degradando o desempenho do algoritmo.

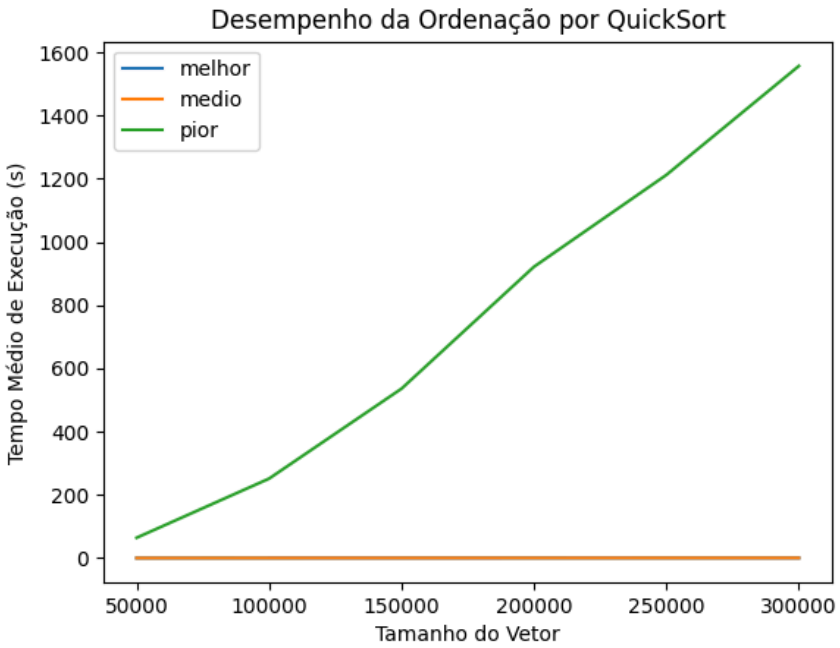
Figura 4.3.3 - Pior Caso



Fonte: Autoria própria.

Os resultados do experimento demonstram a dualidade do Quick Sort: sua alta eficiência e rapidez no melhor e no caso médio, com complexidade O(n log n), contrastando com sua potencial ineficiência no pior caso, com complexidade O(n²). Para manter a eficiência e evitar o pior caso, é crucial implementar estratégias de escolha de pivô robustas. Assim, o Quick Sort se confirma como um dos algoritmos de ordenação mais rápidos e práticos na maioria dos cenários, com a ressalva de cuidados na escolha do pivô para garantir o desempenho ótimo.

Figura 4.3.4 - Todos os Casos

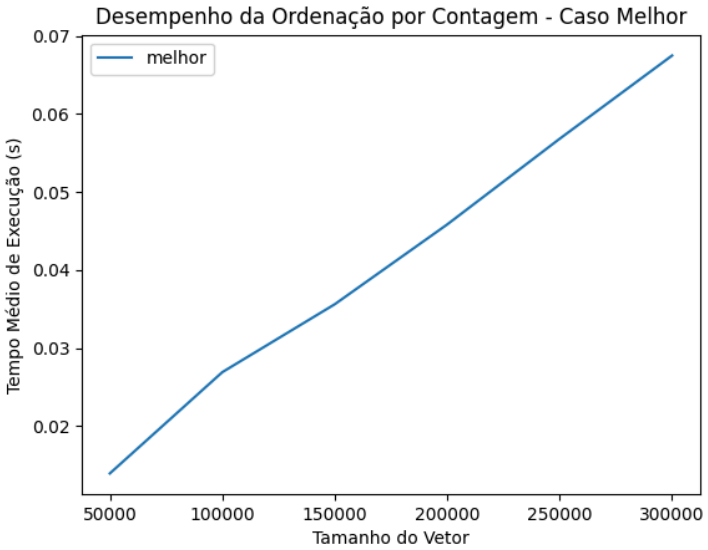


Fonte: Autoria própria.

4.4. Counting Sort

Nesse primeiro caso, o tempo de execução aumenta de maneira linear com o aumento do tamanho da entrada. Isso está de acordo com a complexidade O(n + k) do Counting Sort, onde n é o número de elementos e k é o valor máximo no intervalo de entrada. O crescimento linear reflete a eficiência do algoritmo quando k é relativamente pequeno em comparação a n.

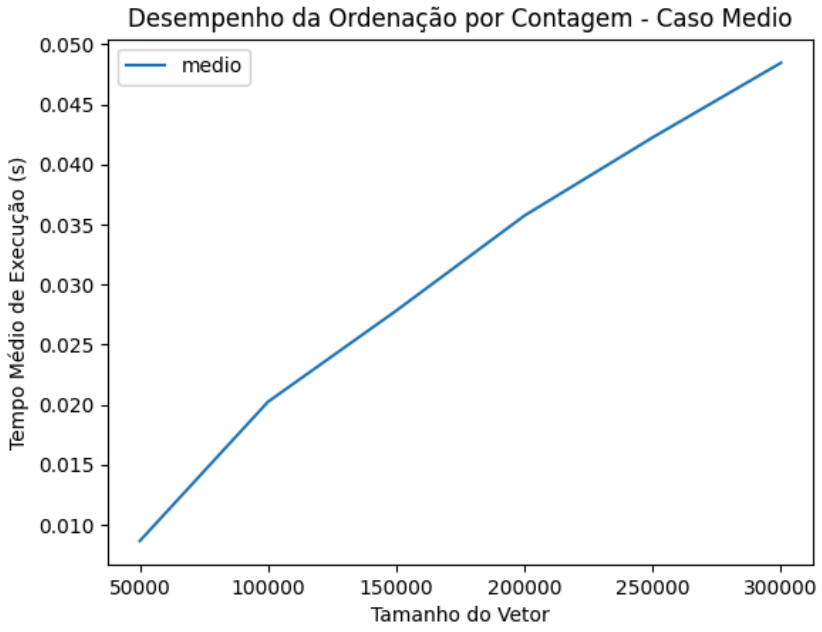
Figura 4.4.1 - Melhor Caso



Fonte: Autoria própria.

No caso médio, os tempos de execução também aumentam linearmente com o tamanho da entrada. Os tempos são ligeiramente menores que no melhor caso, o que pode ser atribuído a variações nas condições experimentais, mas seguem a mesma tendência linear, confirmando a eficiência do algoritmo para entradas típicas.

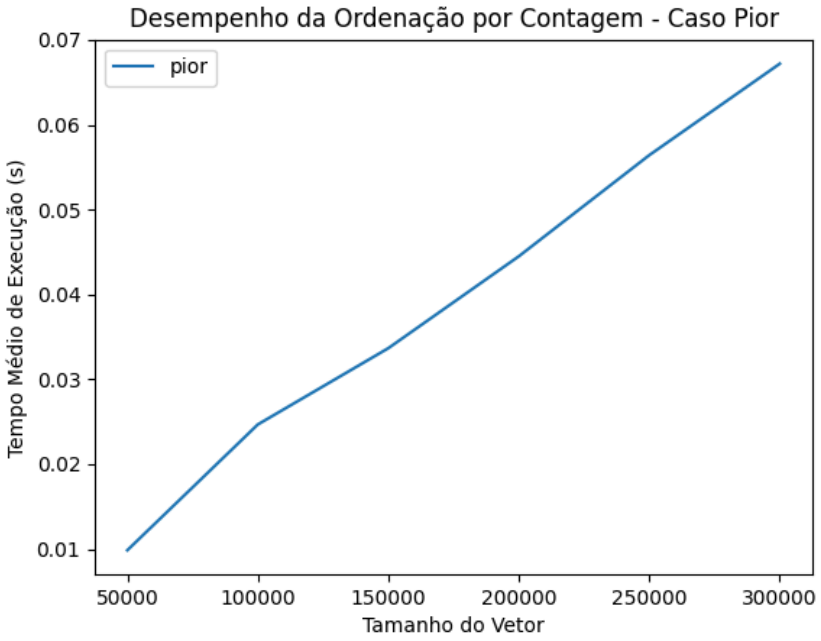
Figura 4.4.2 - Caso Médio



Fonte: Autoria própria.

No pior caso, o crescimento do tempo de execução também é linear, o que demonstra a robustez do Counting Sort em manter sua complexidade O(n + k) independentemente da distribuição dos dados. Os tempos de execução são similares aos dos outros casos, indicando que o algoritmo é eficaz mesmo em condições desfavoráveis.

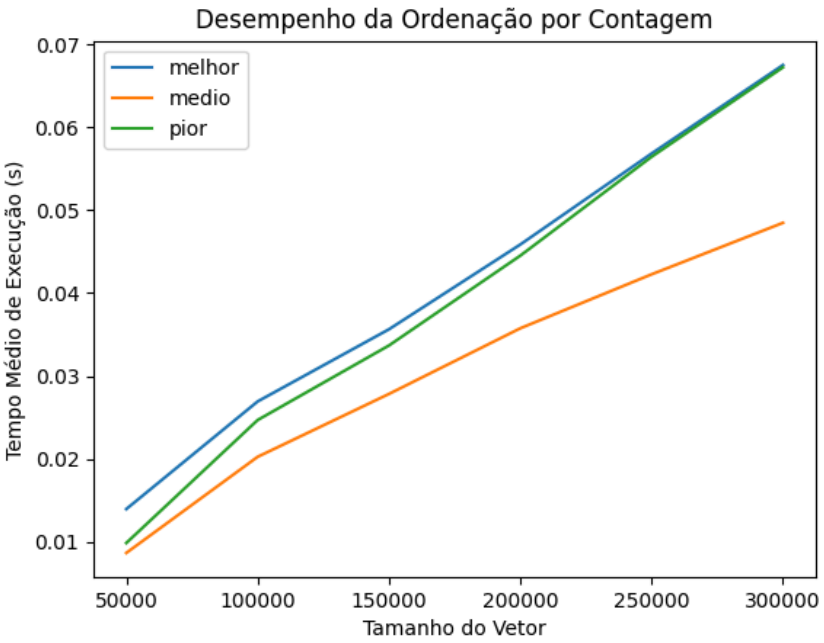
Figura 4.4.3 - Pior Caso



Fonte: Autoria própria.

Os resultados do experimento confirmam a eficiência do Counting Sort, que mantém uma complexidade de tempo O(n + k) em todos os casos (melhor, médio e pior). O comportamento linear observado nos tempos de execução reforça a eficácia do algoritmo para grandes volumes de dados, desde que o intervalo de valores seja limitado. A estabilidade dos tempos de execução entre os diferentes casos destaca a robustez do Counting Sort, tornando-o uma opção ideal para aplicações que envolvem uma faixa conhecida e limitada de valores.

Figura 4.4.4 - Todos os Casos



Fonte: Autoria própria.

5. CONCLUSÃO

A análise comparativa dos algoritmos de ordenação Insertion Sort, Merge Sort, Quick Sort e Counting Sort proporcionou uma visão abrangente sobre suas respectivas eficiências e aplicações. Ao avaliar o desempenho desses algoritmos em diferentes cenários, podemos tirar algumas conclusões importantes.

Em primeiro lugar, observamos que a escolha do algoritmo de ordenação mais adequado depende significativamente das características do conjunto de dados a ser ordenado. Para conjuntos de dados pequenos ou quase ordenados, algoritmos simples como Insertion Sort podem ser eficazes devido à sua simplicidade e baixa sobrecarga. Por outro lado, para conjuntos de dados grandes ou aleatórios, algoritmos de divisão e conquista como Merge Sort e Quick Sort tendem a oferecer melhor desempenho devido à sua eficiência em lidar com conjuntos de dados grandes e não ordenados.

Além disso, a análise dos casos de melhor, médio e pior desempenho de cada algoritmo destacou a importância de entender as complexidades temporais de cada algoritmo. Enquanto alguns algoritmos, como o Counting Sort, apresentam desempenho consistente independentemente do cenário de entrada, outros, como o Quick Sort, podem variar significativamente em termos de tempo de execução dependendo da distribuição dos dados.

Em suma, a seleção do algoritmo de ordenação mais adequado requer uma compreensão cuidadosa das características do conjunto de dados e dos requisitos de desempenho específicos da aplicação. Ao considerar as complexidades temporais e as características de implementação de cada algoritmo, os profissionais de ciência da computação podem tomar decisões informadas para otimizar a eficiência e o desempenho de seus sistemas em relação à ordenação de dados.

6. REFERÊNCIAS

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Algoritmos: Teoria e Prática (3ª ed.). Campus.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Python. John Wiley & Sons.

Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2008). Algorithms. McGraw-Hill.

ANEXO I - INSERTION SORT

- **Código**

```python
import time
import random
import matplotlib.pyplot as plt
from multiprocessing import Pool
tamanhos = [50000, 100000, 150000, 200000, 250000, 300000]

def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key

def gerar_arranjo_aleatorio(tamanho):
    return [random.randint(0, 1000) for _ in range(tamanho)]

def gerar_arranjo_ordenado(tamanho, decrescente=False):
    if decrescente:
        return [i for i in range(tamanho, 0, -1)]
    else:
        return [i for i in range(1, tamanho + 1)]

def executar_experimento(caso_tamanho):
    caso, tamanho = caso_tamanho
    tempos = []
    with open('execussao.txt', 'a+') as f:
        for _ in range(10):
            if caso == 'melhor':
                arr = gerar_arranjo_ordenado(tamanho)
```

```
            elif caso == 'medio':
                arr = gerar_arranjo_aleatorio(tamanho)
            elif caso == 'pior':
                arr = gerar_arranjo_ordenado(tamanho, decrescente=True)
            tempo_inicio = time.time()
            insertion_sort(arr)
            tempo_fim = time.time()
            tempo_execucao = tempo_fim - tempo_inicio
            tempos.append(tempo_execucao)
             print(f"Tamanho: {tamanho}, Caso {caso} Tempo médio de execução:
{tempo_execucao} s")
                f.write(f"Tamanho: {tamanho}, Caso {caso} Tempo medio de
execucao: {tempo_execucao} s\n")
    return caso, tamanho, sum(tempos) / len(tempos)

def tempo_execucao_medio(resultados):
    medias = {}
    for caso, tamanhos in resultados.items():
        medias[caso] = {}
        for tamanho, tempos in tamanhos.items():
            medias[caso][tamanho] = sum(tempos) / len(tempos)
    return medias

def plotar_resultado(caso, dados):
    plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos], label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
           plt.title(f'Desempenho da Ordenação por Inserção - Caso
{caso.capitalize()}')
    plt.legend()
    plt.show()

def plotar_resultados(medias):
    for caso, dados in medias.items():
              plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos],
label=caso)
```

```
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
    plt.title('Desempenho da Ordenação por Inserção')
    plt.legend()
    plt.show()

def principal():
    casos = ['melhor', 'medio', 'pior']
    pool = Pool()
    resultados = {}
    for caso in casos:
        resultados[caso] = {}
        resultados_temp = pool.map(executar_experimento, [(caso, tamanho) for
tamanho in tamanhos])
        for r in resultados_temp:
            resultados[r[0]][r[1]] = [r[2]]  # Guardar apenas o tempo médio
de execução
    with open('output.txt', 'w') as f:
        f.write("Resultados do experimento:\n")
        for caso, data in tempo_execucao_medio(resultados).items():
            f.write(f"Caso: {caso}\n")
            for tamanho, tempo in data.items():
                f.write(f"Tamanho: {tamanho}, Tempo medio de execucao:
{tempo:.6f} s\n")
    for caso, dados in tempo_execucao_medio(resultados).items():
        plotar_resultado(caso, dados)
    plotar_resultados(tempo_execucao_medio(resultados))

if __name__ == "__main__":
    principal()
```

- **Execução**

```
Unset
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.006991386413574219 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.004744291305541992 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.007985115051269531 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.006997108459472656 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.008005142211914062 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.009192943572998047 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.008937835693359375 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.017004966735839844 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.006992816925048828 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.025241374969482422 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.015743017196655273 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.01499485969543457 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.04680490493774414 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.014992237091064453 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.015981197357177734 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.016533613204956055 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.023946046829223633 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.03788566589355469 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.01600050926208496 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.021001577377319336 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.024314403533935547 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.03488016128540039 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.031003713607788086 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.024250507354736328 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.027588605880737305 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.02699756622314453 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.04200172424316406 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.023532629013061523 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.02303147315979004 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.022417306900024414 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.03337979316711426 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.04053068161010742 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.04625248908996582 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.0320279598236084 s
```

```
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.03473258018493652 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.03317618370056152 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.032001495361328125 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.03194546699523926 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.03359580039978027 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.030484676361083984 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.04199814796447754 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.04495739936828613 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05275535583496094 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.03715205192565918 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.04120492935180664 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.03976869583129883 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.03867745399475098 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.03617668151855469 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.032949209213256836 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.02571272850036621 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.05100107192993164 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.05498766899108887 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.0465693473815918 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.054235219955444336 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.046604156494140625 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.04534912109375 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.03202486038208008 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.02970600128173828 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.023321866989135742 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.023932695388793945 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 52.810635566711426 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 58.5576229095459 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 63.89446234703064 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 60.02028465270996 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 62.31683540344238 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 61.42255759239197 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 64.16565299034119 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 60.804691314697266 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 64.56303071975708 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 59.65249800682068 s
```

```
Tamanho: 100000, Caso medio Tempo medio de execucao: 257.5498688220978 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 273.08284735679626 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 241.92514753341675 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 244.68617820739746 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 237.3584384918213 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 234.00426125526428 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 227.05779314041138 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 226.16388273239136 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 224.3856077194214 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 225.1936638355255 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 645.1898782253265 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 582.7039453983307 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 547.7990646362305 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 540.3099267482758 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 493.5882565975189 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 477.0474808216095 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 479.6069486141205 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 472.38689613342285 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 469.2719886302948 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 459.4259374141693 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 1197.7011334896088 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 1051.3497474193573 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 903.5608911514282 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 912.6601271629333 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 911.041764497757 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 770.8842353820801 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 752.5672962665558 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 741.1031754016876 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 746.7678003311157 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 755.3121106624603 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 1857.6350848674774 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 1555.4907011985779 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 1516.2593059539795 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 1267.878839969635 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 1345.7099359035492 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 1233.494042634964 s
```

```
Tamanho: 250000, Caso medio Tempo medio de execucao: 950.4452831745148 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 910.1250741481781 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 955.1616773605347 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 996.6944267749786 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 2801.2463047504425 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 2269.782615661621 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1863.1458427906036 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1872.221016407013 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1393.4648733139038 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1393.447698354721 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1499.6578888893127 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1794.1710302829742 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1661.186806678772 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1827.2432022094727 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 116.27917504310608 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 122.48363041877747 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 129.37923431396484 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 134.12832164764404 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 129.33435559272766 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 125.12519860267639 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 130.594096660614 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 121.76850080490112 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 126.1129322052002 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 129.7463150024414 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 501.4500296115875 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 518.9128723144531 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 505.44960474967957 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 485.9290134906769 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 484.04425859451294 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 482.6715278625488 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 467.04030871391296 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 486.6989941596985 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 479.97135376930237 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 474.764657497406 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 1165.949336528778 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 1085.5792036056519 s
```

```
Tamanho: 150000, Caso pior Tempo medio de execucao: 1055.8641664981842 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 1079.2132496833801 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 1054.507978439331 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 988.9266557693481 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 961.6468584537506 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 1023.512439250946 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 991.7291843891144 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 1006.3751788139343 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 2029.0505952835083 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1934.3108444213867 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1855.2311115264893 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1792.1259942054749 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1733.9627902507782 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1715.6636736392975 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1556.2628931999207 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1333.653264760971 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1274.4311609268188 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1270.1736688613892 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 3117.4744822978973 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 2894.816066503525 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 2813.6384756565094 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 2717.729781150818 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 2317.6017804145813 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1998.9625797271729 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1716.1012876033783 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1579.3601446151733 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1574.7944133281708 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1586.9952523708344 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3740.4744822978973 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3706.816066503525 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3750.6384756565094 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3797.729781150818 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3123.6017804145813 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3456.9625797271729 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3667.1012876033783 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3756.3601446151733 s
```

```
Tamanho: 300000, Caso pior Tempo medio de execucao: 3455.7944133281708 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 3745.9952523708344 s
```

- **Tempo Médio**

```
Unset
Resultados do experimento:
Caso: melhor
Tamanho: 50000, Tempo medio de execucao: 0.010209 s
Tamanho: 100000, Tempo medio de execucao: 0.022388 s
Tamanho: 150000, Tempo medio de execucao: 0.028002 s
Tamanho: 200000, Tempo medio de execucao: 0.034813 s
Tamanho: 250000, Tempo medio de execucao: 0.039135 s
Tamanho: 300000, Tempo medio de execucao: 0.040773 s
Caso: medio
Tamanho: 50000, Tempo medio de execucao: 60.820827 s
Tamanho: 100000, Tempo medio de execucao: 239.140769 s
Tamanho: 150000, Tempo medio de execucao: 516.733032 s
Tamanho: 200000, Tempo medio de execucao: 874.294828 s
Tamanho: 250000, Tempo medio de execucao: 1258.889437 s
Tamanho: 300000, Tempo medio de execucao: 1837.556728 s
Caso: pior
Tamanho: 50000, Tempo medio de execucao: 126.495176 s
Tamanho: 100000, Tempo medio de execucao: 488.693262 s
Tamanho: 150000, Tempo medio de execucao: 1041.330425 s
Tamanho: 200000, Tempo medio de execucao: 1649.486600 s
Tamanho: 250000, Tempo medio de execucao: 2231.747426 s
Tamanho: 300000, Tempo medio de execucao: 3740.000000 s
```

ANEXO II - MERGE SORT

- **Código**

```python
import time
import random
import matplotlib.pyplot as plt
from multiprocessing import Pool
tamanhos = [50000, 100000, 150000, 200000, 250000, 300000]

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
```

```
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

def gerar_arranjo_aleatorio(tamanho):
    return [random.randint(0, 1000) for _ in range(tamanho)]

def gerar_arranjo_ordenado(tamanho, decrescente=False):
    if decrescente:
        return [i for i in range(tamanho, 0, -1)]
    else:
        return [i for i in range(1, tamanho + 1)]

def executar_experimento(caso_tamanho):
    caso, tamanho = caso_tamanho
    tempos = []
    with open('execussao.txt', 'a+') as f:
        for _ in range(10):
            if caso == 'melhor':
                arr = gerar_arranjo_ordenado(tamanho)
            elif caso == 'medio':
                arr = gerar_arranjo_aleatorio(tamanho)
            elif caso == 'pior':
                arr = gerar_arranjo_ordenado(tamanho, decrescente=True)
            tempo_inicio = time.time()
            merge_sort(arr)
            tempo_fim = time.time()
            tempo_execucao = tempo_fim - tempo_inicio
            tempos.append(tempo_execucao)
             print(f"Tamanho: {tamanho}, Caso {caso} Tempo médio de execução:
{tempo_execucao} s")
                f.write(f"Tamanho: {tamanho}, Caso {caso} Tempo medio de
execucao: {tempo_execucao} s\n")
    return caso, tamanho, sum(tempos) / len(tempos)
```

```
def tempo_execucao_medio(resultados):
    medias = {}
    for caso, tamanhos in resultados.items():
        medias[caso] = {}
        for tamanho, tempos in tamanhos.items():
            medias[caso][tamanho] = sum(tempos) / len(tempos)
    return medias

def plotar_resultado(caso, dados):
    plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos], label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
           plt.title(f'Desempenho da Ordenação por Inserção - Caso
{caso.capitalize()}')
    plt.legend()
    plt.show()

def plotar_resultados(medias):
    for caso, dados in medias.items():
              plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos],
label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
    plt.title('Desempenho da Ordenação por Inserção')
    plt.legend()
    plt.show()

def principal():
    casos = ['melhor', 'medio', 'pior']
    pool = Pool()
    resultados = {}
    for caso in casos:
        resultados[caso] = {}
        resultados_temp = pool.map(executar_experimento, [(caso, tamanho) for
tamanho in tamanhos])
        for r in resultados_temp:
```

```
            resultados[r[0]][r[1]] = [r[2]]  # Guardar apenas o tempo médio
de execução
    with open('output.txt', 'w') as f:
        f.write("Resultados do experimento:\n")
        for caso, data in tempo_execucao_medio(resultados).items():
            f.write(f"Caso: {caso}\n")
            for tamanho, tempo in data.items():
                    f.write(f"Tamanho: {tamanho}, Tempo medio de execucao:
{tempo:.6f} s\n")
    for caso, dados in tempo_execucao_medio(resultados).items():
        plotar_resultado(caso, dados)
    plotar_resultados(tempo_execucao_medio(resultados))

if __name__ == "__main__":
    principal()
```

- **Execução**

```
Unset
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.12348818778991699 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.13800811767578125 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.15848517417907715 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.1415259838104248 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.15148448944091797 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.13890695571899414 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.1382427215576172 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.13796472549438477 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.14021682739257812 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.1234748363494873 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.2994048595428467 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.2988312244415283 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.28934407234191895 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.30617761611938477 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.27876877784729004 s
```

```
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.2747213840484619 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.2727992534637451 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.2838311195373535 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.30663347244262695 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.2757875919342041 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.46729421615600586 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.46184778213500977 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.46332263946533203 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.44579148292541504 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.4346790313720703 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.4422621726989746 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.44724488258361816 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.4526963233947754 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.462660551071167 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.46303439140319824 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.611781120300293 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.623551607131958 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5912525653839111 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5949170589447021 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5723435878753662 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5745906829833984 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5954828262329102 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5562374591827393 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5662293434143066 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.5461671352386475 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.7907466888427734 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.7741749286651611 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.7722790241241455 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.753788948059082 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.7330722808837891 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.7245965003967285 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.6731219291687012 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.6737058162689209 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.6892733573913574 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.6759607791900635 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.9586191177368164 s
```

```
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.910865068435669 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8777077198028564 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8629989624023438 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8318676948547363 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8397397994995117 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8235688209533691 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8017711639404297 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.7995226383209229 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.8015592098236084 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.14698219299316406 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.15219855308532715 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.1610095500946045 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.15830373764038086 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.16668152809143066 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.16908597946166992 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.16395020484924316 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.15622282028198242 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.16943883895874023 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.14825201034545898 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.34012556076049805 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.32346320152282715 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.34068799018859863 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.333864688873291 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.3534698486328125 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.314373254776001 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.3320128917694092 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.33471155166625977 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.3315451145172119 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.3306102752685547 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.5315647125244141 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.5586230754852295 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.524010419845581 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.523529052734375 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.5257084369659424 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.5291907787322998 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.5266389846801758 s
```

```
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.512833833694458 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.54779052734375 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.5445849895477295 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.7667713165283203 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.7261178493499756 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.7094683647155762 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.725926399230957 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.7182838916778564 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.7039763927459717 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.7096292972564697 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.750483512878418 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.6821527481079102 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.6823179721832275 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.9252109527587891 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.9177772998809814 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.9003205299377441 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.8965373039245605 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.8805732727050781 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.8706722259521484 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.9257287979125977 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.8909828662872314 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.8164515495300293 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.8588180541992188 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.1201465129852295 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.1228246688842773 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.063647747039795 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.094273567199707 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.158020257949829 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.017970085144043 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.065539836883545 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.0208241939544678 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.037048101425171 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 1.0602929592132568 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.14630675315856934 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.2366030216217041 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.19210243225097656 s
```

```
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.25437211990356445 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.16689038276672363 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.13094329833984375 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.14492392539978027 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.13779854774475098 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.1452028751373291 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.1560804843902588 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.4302501678466797 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.4505343437194824 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.3109109401702881 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.3086814880371094 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.31398725509643555 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.29003405570983887 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.4045250415802002 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.3890674114227295 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.29857563972473145 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.2770555019378662 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.6419162750244141 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.5235273838043213 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.49335360527038574 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.48995471000671387 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.6572229862213135 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.46120667457580566 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.4879941940307617 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.45004701614379883 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.4471602439880371 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.4537534713745117 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.9160704612731934 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.645172119140625 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.6422414779663086 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.7627873420715332 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.5950298309326172 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.5959200859069824 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.5821206569671631 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.5611300468444824 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.558011531829834 s
```

```
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.5645105838775635 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.9768519401550293 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.8118512630462646 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.762099027633667 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.7371113300323486 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.7171244621276855 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.7276487350463867 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.6985528469085693 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.6860227584838867 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.733351469039917 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.6979899406433105 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.9837920665740967 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.919642448425293 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.867048978805542 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.8740177154541016 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.842146635055542 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.9079830646514893 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.841418981552124 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.8040816783905029 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.8355472087860107 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.8320620059967041 s
```

- **Tempo Médio**

```
Unset
Resultados do experimento:
Caso: melhor
Tamanho: 50000, Tempo medio de execucao: 0.139180 s
Tamanho: 100000, Tempo medio de execucao: 0.288630 s
Tamanho: 150000, Tempo medio de execucao: 0.454083 s
Tamanho: 200000, Tempo medio de execucao: 0.583255 s
Tamanho: 250000, Tempo medio de execucao: 0.726072 s
Tamanho: 300000, Tempo medio de execucao: 0.850822 s
Caso: medio
```

```
Tamanho: 50000, Tempo medio de execucao: 0.159213 s
Tamanho: 100000, Tempo medio de execucao: 0.333486 s
Tamanho: 150000, Tempo medio de execucao: 0.532447 s
Tamanho: 200000, Tempo medio de execucao: 0.717513 s
Tamanho: 250000, Tempo medio de execucao: 0.888307 s
Tamanho: 300000, Tempo medio de execucao: 1.076059 s
Caso: pior
Tamanho: 50000, Tempo medio de execucao: 0.171122 s
Tamanho: 100000, Tempo medio de execucao: 0.347362 s
Tamanho: 150000, Tempo medio de execucao: 0.510614 s
Tamanho: 200000, Tempo medio de execucao: 0.642299 s
Tamanho: 250000, Tempo medio de execucao: 0.754860 s
Tamanho: 300000, Tempo medio de execucao: 0.870774 s
```

ANEXO III - QUICK SORT

- **Código**

```python
import time
import random
import matplotlib.pyplot as plt
from multiprocessing import Pool
import sys
sys.setrecursionlimit(300000000) #configura a 'profundidade' da pilha para 10000

tamanhos = [50000, 100000, 150000, 200000, 250000, 300000]

def QuickSort(X, IniVet, FimVet, caso):
    i = IniVet
    j = FimVet

    if caso == 'pior':
        pivo = X[IniVet]
    else:
        pivo = X[(IniVet + FimVet) // 2]

    while(i <= j):
        while (X[i] < pivo):
            i += 1
        while (X[j] > pivo):
            j -= 1
        if (i <= j):
            aux = X[i]
            X[i] = X[j]
            X[j] = aux
            i += 1
            j -= 1
    if (IniVet < j):
```

```
        QuickSort(X, IniVet, j, caso)
    if (i < FimVet):
        QuickSort(X, i, FimVet, caso)



def gerar_arranjo_aleatorio(tamanho):
    return [random.randint(0, 1000) for _ in range(tamanho)]

def gerar_arranjo_ordenado(tamanho):
    return [i for i in range(1, tamanho + 1)]

def gerar_arranjo_pior_caso(tamanho):
    return [tamanho - i for i in range(tamanho)]


def executar_experimento(caso_tamanho):
    caso, tamanho = caso_tamanho
    tempos = []
    with open('execussao.txt', 'a+') as f:
        for _ in range(10):
            if caso == 'melhor':
                arr = gerar_arranjo_ordenado(tamanho)
            elif caso == 'medio':
                arr = gerar_arranjo_aleatorio(tamanho)
            elif caso == 'pior':
                arr = gerar_arranjo_ordenado(tamanho)
            tempo_inicio = time.time()
            QuickSort(arr, 0, len(arr) - 1, caso)
            tempo_fim = time.time()
            tempo_execucao = tempo_fim - tempo_inicio
            tempos.append(tempo_execucao)
             print(f"Tamanho: {tamanho}, Caso {caso} Tempo médio de execução:
{tempo_execucao} s")
                  f.write(f"Tamanho: {tamanho}, Caso {caso} Tempo medio de
execucao: {tempo_execucao} s\n")
```

```
    return caso, tamanho, sum(tempos) / len(tempos)

def tempo_execucao_medio(resultados):
    medias = {}
    for caso, tamanhos in resultados.items():
        medias[caso] = {}
        for tamanho, tempos in tamanhos.items():
            medias[caso][tamanho] = sum(tempos) / len(tempos)
    return medias

def plotar_resultado(caso, dados):
    plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos], label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
           plt.title(f'Desempenho da Ordenação por QuickSort - Caso
{caso.capitalize()}')
    plt.legend()
    plt.show()

def plotar_resultados(medias):
    for caso, dados in medias.items():
              plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos],
label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
    plt.title('Desempenho da Ordenação por QuickSort')
    plt.legend()
    plt.show()

def principal():
    casos = ['melhor', 'medio', 'pior']
    pool = Pool()
    resultados = {}
    for caso in casos:
        resultados[caso] = {}
```

```
        resultados_temp = pool.map(executar_experimento, [(caso, tamanho) for
tamanho in tamanhos])
        for r in resultados_temp:
             resultados[r[0]][r[1]] = [r[2]]  # Guardar apenas o tempo médio
de execução
    with open('output.txt', 'w') as f:
        f.write("Resultados do experimento:\n")
        for caso, data in tempo_execucao_medio(resultados).items():
            f.write(f"Caso: {caso}\n")
            for tamanho, tempo in data.items():
                    f.write(f"Tamanho: {tamanho}, Tempo medio de execucao:
{tempo:.6f} s\n")
    for caso, dados in tempo_execucao_medio(resultados).items():
        plotar_resultado(caso, dados)
    plotar_resultados(tempo_execucao_medio(resultados))

if __name__ == "__main__":
    principal()
```

- **Execução**

```
Unset
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.04688763618469238 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.055016517639160156 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.05932784080505371 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.05257534980773926 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.061578989028930664 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.07203388214111328 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.06343722343444824 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.06154131889343262 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.04997754096984863 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.057030677795410156 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.11987829208374023 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.12476110458374023 s
```

```
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.12097501754760742 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.11271166801452637 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.11810088157653809 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.10246849060058594 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.10024356842041016 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.09126472473144531 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.07062768936157227 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.0896146297454834 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.19802618026733398 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.19267725944519043 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.17280125617980957 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.14581012725830078 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.09510135650634766 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.10597920417785645 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.14068102836608887 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.05793571472167969 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.07407259941101074 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.12294912338256836 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.24432921409606934 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.23192095756530762 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.20515131950378418 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.19012904167175293 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.1646578311920166 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.13714003562927246 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.13500189781188965 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.10939335823059082 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.14200091361999512 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.09320592880249023 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.30937886238098145 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.24942803382873535 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.19471073150634766 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.15735435485839844 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.14339900016784668 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.14977383613586426 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.16208100318908691 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.11677336692810059 s
```

```
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.10680103302001953 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.10992980003356934 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.3824892044067383 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.24178481101989746 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.27959585189819336 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.21347999572753906 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.1496882438659668 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.15198087692260742 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.17657184600830078 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.14006662368774414 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.1406857967376709 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.14105844497680664 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.0471646785736084 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.047579288482666016 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.06978368759155273 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.05587124824523926 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.08765935897827148 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.0718069076538086 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.07979178428649902 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.06821537017822266 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.08493328094482422 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.07303833961486816 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.15468859672546387 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.17350435256958008 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.11234235763549805 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.16348004341125488 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.15689992904663086 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.1622154712677002 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.1400749683380127 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.1678328514099121 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.16209149360656738 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.09062719345092773 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.26673030853271484 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.2144453525543213 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.2370145320892334 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.1649324893951416 s
```

```
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.29292845726013184 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.24468684196472168 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.15498018264770508 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.26328563690185547 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.2415330410003662 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.17290019989013672 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.3200569152832031 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.30293846130371094 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.2569003105163574 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.39781999588012695 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.2994832992553711 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.35770606994628906 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.2712516784667969 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.27015209197998047 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.20312285423278809 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.20200514793395996 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.4187467098236084 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.3411262035369873 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.4570119380950928 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.43683385848999023 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.4510056972503662 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.34168243408203125 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.2790212631225586 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.2927734851837158 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.24321961402893066 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.25665831565856934 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.5430502891540527 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.5042121410369873 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.5292465686798096 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.4730565547943115 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.3365345001220703 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.40739870071411133 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.29285526275634766 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.29463839530944824 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.26423001289367676 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.2500002384185791 s
```

```
Tamanho: 50000, Caso pior Tempo medio de execucao: 46.46762180328369 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 57.58586072921753 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 67.88958644866943 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 66.33979868888855 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 70.50097441673279 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 68.44169235229492 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 63.090420961380005 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 65.90812587738037 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 65.87700057029724 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 70.70154333114624 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 234.66713762283325 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 266.9723541736603 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 252.00116896629333 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 238.2658200263977 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 245.68389630317688 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 249.28850317001343 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 253.20843148231506 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 253.37357473373413 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 252.14495134353638 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 267.1870584487915 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 574.4630305767059 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 536.6735746860504 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 569.0795140266418 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 558.292322397232 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 558.3309705257416 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 516.6214957237244 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 485.439820766449 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 506.58397006988525 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 459.10995054244995 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 593.5309863090515 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 991.8664255142212 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 1004.7626578807831 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 979.0922083854675 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 900.1265137195587 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 853.3089106082916 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 979.5206186771393 s
```

```
Tamanho: 200000, Caso pior Tempo medio de execucao: 879.0797040462494 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 871.4610278606415 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 879.0783808231354 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 875.660893201828 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1577.0387456417084 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1517.4757933616638 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1403.775952577591 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1461.3461413383484 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1378.5415058135986 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1354.820238351822 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 1205.8433566093445 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 749.3580281734467 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 733.2853901386261 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 738.9567558765411 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 2258.166032552719 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 2053.5957429409027 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 2064.505531311035 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1947.3368470668793 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1711.061942577362 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1070.7347376346588 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1051.9691307544708 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1035.151426076889 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1191.0946357250214 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 1181.8287127017975 s
```

- **Tempo Médio**

```
Unset
Resultados do experimento:
Caso: melhor
Tamanho: 50000, Tempo medio de execucao: 0.057941 s
Tamanho: 100000, Tempo medio de execucao: 0.105065 s
Tamanho: 150000, Tempo medio de execucao: 0.130603 s
Tamanho: 200000, Tempo medio de execucao: 0.165293 s
```

```
Tamanho: 250000, Tempo medio de execucao: 0.169963 s
Tamanho: 300000, Tempo medio de execucao: 0.201740 s
Caso: medio
Tamanho: 50000, Tempo medio de execucao: 0.068584 s
Tamanho: 100000, Tempo medio de execucao: 0.148376 s
Tamanho: 150000, Tempo medio de execucao: 0.225344 s
Tamanho: 200000, Tempo medio de execucao: 0.288144 s
Tamanho: 250000, Tempo medio de execucao: 0.351808 s
Tamanho: 300000, Tempo medio de execucao: 0.389522 s
Caso: pior
Tamanho: 50000, Tempo medio de execucao: 64.280263 s
Tamanho: 100000, Tempo medio de execucao: 251.279290 s
Tamanho: 150000, Tempo medio de execucao: 535.812564 s
Tamanho: 200000, Tempo medio de execucao: 921.395734 s
Tamanho: 250000, Tempo medio de execucao: 1212.044191 s
Tamanho: 300000, Tempo medio de execucao: 1556.544474 s
```

ANEXO IV - COUNTING SORT

- **Código**

```python
import time
import random
import matplotlib.pyplot as plt
from multiprocessing import Pool
tamanhos = [50000, 100000, 150000, 200000, 250000, 300000]

def countingSort(A):
    n = len(A)
    k = max(A)  # descobre o valor máximo no vetor A

    C = [0] * (k + 1)  # Inicializa a lista de contagem com zeros
    B = [0] * n  # Inicializa a lista de saída com zeros

    # Conta a ocorrência de cada elemento em A
    for j in range(n):
        C[A[j]] += 1

    # Atualiza C para indicar a posição correta de cada elemento na saída
    for i in range(1, k + 1):
        C[i] += C[i - 1]

    # Monta o array ordenado B
    for j in range(n - 1, -1, -1):
        B[C[A[j]] - 1] = A[j]
        C[A[j]] -= 1

    return B

def gerar_arranjo_aleatorio(tamanho):
    return [random.randint(0, 1000) for _ in range(tamanho)]
```

```
def gerar_arranjo_ordenado(tamanho, decrescente=False):
    if decrescente:
        return [i for i in range(tamanho, 0, -1)]
    else:
        return [i for i in range(1, tamanho + 1)]

def executar_experimento(caso_tamanho):
    caso, tamanho = caso_tamanho
    tempos = []
    with open('execussao.txt', 'a+') as f:
        for _ in range(10):
            if caso == 'melhor':
                arr = gerar_arranjo_ordenado(tamanho)
            elif caso == 'medio':
                arr = gerar_arranjo_aleatorio(tamanho)
            elif caso == 'pior':
                arr = gerar_arranjo_ordenado(tamanho, decrescente=True)
            tempo_inicio = time.time()
            countingSort(arr)
            tempo_fim = time.time()
            tempo_execucao = tempo_fim - tempo_inicio
            tempos.append(tempo_execucao)
             print(f"Tamanho: {tamanho}, Caso {caso} Tempo médio de execução:
{tempo_execucao} s")
                  f.write(f"Tamanho: {tamanho}, Caso {caso} Tempo medio de
execucao: {tempo_execucao} s\n")
    return caso, tamanho, sum(tempos) / len(tempos)

def tempo_execucao_medio(resultados):
    medias = {}
    for caso, tamanhos in resultados.items():
        medias[caso] = {}
        for tamanho, tempos in tamanhos.items():
            medias[caso][tamanho] = sum(tempos) / len(tempos)
    return medias
```

```
def plotar_resultado(caso, dados):
    plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos], label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
           plt.title(f'Desempenho da Ordenação por Contagem - Caso
{caso.capitalize()}')
    plt.legend()
    plt.show()

def plotar_resultados(medias):
    for caso, dados in medias.items():
             plt.plot(tamanhos, [dados[tamanho] for tamanho in tamanhos],
label=caso)
    plt.xlabel('Tamanho do Vetor')
    plt.ylabel('Tempo Médio de Execução (s)')
    plt.title('Desempenho da Ordenação por Contagem')
    plt.legend()
    plt.show()

def principal():
    casos = ['melhor', 'medio', 'pior']
    pool = Pool()
    resultados = {}
    for caso in casos:
        resultados[caso] = {}
        resultados_temp = pool.map(executar_experimento, [(caso, tamanho) for
tamanho in tamanhos])
        for r in resultados_temp:
             resultados[r[0]][r[1]] = [r[2]]  # Guardar apenas o tempo médio
de execução
    with open('output.txt', 'w') as f:
        f.write("Resultados do experimento:\n")
        for caso, data in tempo_execucao_medio(resultados).items():
            f.write(f"Caso: {caso}\n")
            for tamanho, tempo in data.items():
```

```
                    f.write(f"Tamanho: {tamanho}, Tempo medio de execucao:
{tempo:.6f} s\n")
    for caso, dados in tempo_execucao_medio(resultados).items():
        plotar_resultado(caso, dados)
    plotar_resultados(tempo_execucao_medio(resultados))

if __name__ == "__main__":
    principal()
```

- **Execução**

Unset

```
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.027836084365844727 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.015965700149536133 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.017978906631469727 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.010897159576416016 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.008008956909179688 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.015346050262451172 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.010803461074829102 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.010378122329711914 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.011453866958618164 s
Tamanho: 50000, Caso melhor Tempo medio de execucao: 0.010869264602661133 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.04278731346130371 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.029922962188720703 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.021979808807373047 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.02395009994506836 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.028322219848632812 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.029233217239379883 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.023379087448120117 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.025574684143066406 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.026149272918701172 s
Tamanho: 100000, Caso melhor Tempo medio de execucao: 0.0179293155670166 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.05760502815246582 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.03312802314758301 s
```

```
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.034558773040771484 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.03203129768371582 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.033892154693603516 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.03286862373352051 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.02812027931213379 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.04108381271362305 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.031148195266723633 s
Tamanho: 150000, Caso melhor Tempo medio de execucao: 0.03179526329040527 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.06680011749267578 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.04475522041320801 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.040355682373046875 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.0443117618560791 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.0508120059967041 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.037973642349243164 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.04126882553100586 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.04055333137512207 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.043709993336242676 s
Tamanho: 200000, Caso melhor Tempo medio de execucao: 0.047890663146972656 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.08087468147277832 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05395960807800293 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05262303352355957 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.06682538986206055 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05589461326599121 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05140376091003418 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05118298530578613 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05393266677856445 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.05108308792114258 s
Tamanho: 250000, Caso melhor Tempo medio de execucao: 0.050377607345581055 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.09127330780029297 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.07551765441894531 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.0625612735748291 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.0671849250793457 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.06240129470825195 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.06633496284484863 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.056122541427612305 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.06073927879333496 s
```

```
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.06660604476928711 s
Tamanho: 300000, Caso melhor Tempo medio de execucao: 0.06606173515319824 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.007539510726928711 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.008011341094970703 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.014075040817260742 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.00014853477478027344 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.011623859405517578 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.009890317916870117 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.016582250595092773 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.002801656723022461 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.007523059844970703 s
Tamanho: 50000, Caso medio Tempo medio de execucao: 0.008420705795288086 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.017965316772460938 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.015113353729248047 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.019327640533447266 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.03425931930541992 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.01525568962097168 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.027600526809692383 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.01481485366821289 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.019174814224243164 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.014795064926147461 s
Tamanho: 100000, Caso medio Tempo medio de execucao: 0.024165868759155273 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.0314333438873291 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.027296781539916992 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.027871370315551758 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.03478240966796875 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.027739763259887695 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.03302574157714844 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.020300626754760742 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.024150848388671875 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.025497913360595703 s
Tamanho: 150000, Caso medio Tempo medio de execucao: 0.026200532913208008 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.0412440299987793 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.03258156776428223 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.0355830192565918 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.034699440002441406 s
```

```
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.0499730110168457 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.03660225868225098 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.03232240676879883 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.032926321029663086 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.03346395492553711 s
Tamanho: 200000, Caso medio Tempo medio de execucao: 0.02803182601928711 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.0456085205078125 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.0444188117980957 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.04576420783996582 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.0469968318939209 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.04145097732543945 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.042903900146484375 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.04494619369506836 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.043721675872802734 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.03638195991516113 s
Tamanho: 250000, Caso medio Tempo medio de execucao: 0.030315876007080078 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.058382272720336914 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.04901766777038574 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.05010509490966797 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.05622720718383789 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.04989337921142578 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.050871849060058594 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.04244065284729004 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.040313005447387695 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.03690743446350098 s
Tamanho: 300000, Caso medio Tempo medio de execucao: 0.050417184829711914 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.02176976203918457 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.009365081787109375 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.0067293643951416016 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.006433010101318359 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.00699615478515625 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.008094549179077148 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.011278390884399414 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.010186910629272461 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.004077434539794922 s
Tamanho: 50000, Caso pior Tempo medio de execucao: 0.013678312301635742 s
```

```
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.03315234184265137 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.03348708152770996 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.031124353408813477 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.0259549617767334 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.019690513610839844 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.021440505981445312 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.014389991760253906 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.027386188507080078 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.019455909729003906 s
Tamanho: 100000, Caso pior Tempo medio de execucao: 0.020734786987304688 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.0329136848449707 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.05175137519836426 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.03284025192260742 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.020205020904541016 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.036370038986206055 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.038579463958740234 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.033983469009399414 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.030295372009277344 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.030222177505493164 s
Tamanho: 150000, Caso pior Tempo medio de execucao: 0.02954888343811035 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.0498354434967041 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.06337881088256836 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.048665761947631836 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.05266141891479492 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.04769396781921387 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.03497147560119629 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.04694986343383789 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.02763509750366211 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.039335012435913086 s
Tamanho: 200000, Caso pior Tempo medio de execucao: 0.033881187438964844 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.07065725326538086 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.06234002113342285 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.06053280830383301 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.0521395206451416 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.0671236515045166 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.04895830154418945 s
```

```
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.050922393798828125 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.04758715629577637 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.05182456970214844 s
Tamanho: 250000, Caso pior Tempo medio de execucao: 0.05213594436645508 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.08723044395446777 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.06609964370727539 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.07279825210571289 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.0644235610961914 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.0658717155456543 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.06350302696228027 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.06352424621582031 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.06513500213623047 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.05881690979003906 s
Tamanho: 300000, Caso pior Tempo medio de execucao: 0.06457686424255371 s
```

- **Tempo Médio**

```
Unset
Resultados do experimento:
Caso: melhor
Tamanho: 50000, Tempo medio de execucao: 0.013954 s
Tamanho: 100000, Tempo medio de execucao: 0.026923 s
Tamanho: 150000, Tempo medio de execucao: 0.035623 s
Tamanho: 200000, Tempo medio de execucao: 0.045843 s
Tamanho: 250000, Tempo medio de execucao: 0.056816 s
Tamanho: 300000, Tempo medio de execucao: 0.067480 s
Caso: medio
Tamanho: 50000, Tempo medio de execucao: 0.008662 s
Tamanho: 100000, Tempo medio de execucao: 0.020247 s
Tamanho: 150000, Tempo medio de execucao: 0.027830 s
Tamanho: 200000, Tempo medio de execucao: 0.035743 s
Tamanho: 250000, Tempo medio de execucao: 0.042251 s
Tamanho: 300000, Tempo medio de execucao: 0.048458 s
Caso: pior
```

```
Tamanho: 50000, Tempo medio de execucao: 0.009861 s
Tamanho: 100000, Tempo medio de execucao: 0.024682 s
Tamanho: 150000, Tempo medio de execucao: 0.033671 s
Tamanho: 200000, Tempo medio de execucao: 0.044501 s
Tamanho: 250000, Tempo medio de execucao: 0.056422 s
Tamanho: 300000, Tempo medio de execucao: 0.067198 s
```