UNIVERSIDADE FEDERAL DO MARANHÃO

CAMPUS DE SÃO LUÍS - CIDADE UNIVERSITÁRIA

ESTRUTURA DE DADOS II - CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DA ANÁLISE DOS RESULTADOS DOS ALGORITMOS : COMPARAÇÕES, ATRIBUIÇÕES E TEMPO DE EXECUÇÃO

ALUNO:

BRENO ROBERTO REIS VIDIGAL

PROFESSOR:

JOÃO DALLYSON

RESUMO:

O presente relatório tem como objetivos apresentar e analisar os principais algoritmos de ordenação sob critérios como: Número de atribuições, comparações feitas e tempo de execução dos algoritmos; assim como compará-los entre si para entradas de diferentes tamanhos, visando medir a eficiência de tempo e espaço ocupados por cada um.

SELECTION SORT

A ordenação por seleção é notada pela sua simplicidade e pode ser considerado um dos melhores algoritmos para ordenação de pequenos arquivos, se baseando em comparações entre o arquivo da posição "i" com os demais arquivos "j = i+1 de 1... n", percorrendo a entrada da esquerda pra direita e selecionando o índice com a menor chave. No final, é realizada uma troca entre arquivos, assim criando uma sub-lista ordenada que cresce da esquerda pra direita.

O selection sort não é indicado para ordenação de grandes arquivos devido à sua natureza de ordem quadrática, fazendo com que o tempo de execução cresça na ordem de $\mathbf{O(n^2)}$ em número de comparações.

INSERTION SORT

A ordenação por inserção também se trata de um procedimento simples que divide a entrada em duas sub-entradas, uma ordenada e outra não ordenada. O procedimento se baseia em comparar o primeiro elemento da lista não ordenada com os elementos da lista ordenada. Enquanto esse for menor, o loop continua procurando a posição certa para inserir o elemento, criando uma lista ordenada que cresce da esquerda pra direita.

Esse método é mais rápido que o Select, tanto no caso pior quanto no melhor caso.

SELECT-INSERT SORT

A implementação desse algoritmo testada nesse relatório seguiu a ideia simplificada de ordenar uma parte de X% do vetor, da esquerda pra direita, usando o Select Sort, e o restante do vetor usando Insert Sort.

QUICK SORT

O quick sort é um algoritmo eficiente e conhecido por ser rápido independentemente do tamanho da entrada. O algoritmo funciona escolhendo um pivô arbitrário e fazendo comparações para descobrir a posição certa do elemento pivô. Em seguida, o algoritmo é chamado recursivamente para fazer o mesmo para as partições esquerda e direita do pivô. Possui complexidade média de **O(nlogn) mas pode executar em $O(n^2)$ no pior caso.**

MERGE SORT

O merge sort é um algoritmo eficiente pautado na ideia de divisão e conquista. Ele divide o vetor em duas metades e chama o algoritmo recursivamente para ambas até que cada vetor tenha no máximo 1 elemento, e assim, eles são ordenados com ajuda de uma função auxiliar "merge" que também será responsável por juntar os pequenos vetores ordenados à solução final. Possui complexidade final de **O(nlogn), mas consome mais memória.**

HEAP SORT

HeapSort é o nome dado ao método de ordenação que utiliza a estrutura heap para ordenar coleções na ordem de **O(nlogn)**. O heap é uma árvore binária completa que pode ser representada num vetor com o auxílio de uma função maxheapify, que modifica o vetor de forma que cada nó pai tenha um valor maior que os nós filhos. Dessa forma, cada deleção do nó-raíz do heap resultará no maior valor ocupando a última posição do vetor, de forma a ordená-lo. Pode ser ruim para ordenação de pequenos arquivos, devido ao processo de criação do heap.

RESULTADOS

A análise dos resultados se dará por prints da execução e gráficos comparando o número de comparações, atribuições e tempo de execução. Os algoritmos foram testados usando vetores com 80000 registros.

Somente as comparações entre elementos do vetor foram consideradas.

Atribuições de índices, variáveis temporárias foram ignoradas, considerando somente a trocas de itens no vetor.

SELECTION SORT

```
Product{name='Toys', ID=461872, price=107.52}
Product{name='Toys', ID=286247, price=179.2}
Product{name='Toys', ID=454912, price=71.68}
Product{name='Toys', ID=174831, price=143.36}
Product{name='Toys', ID=252996, price=107.52}
Product{name='Toys', ID=312663, price=71.68}
Critério de ordenação: TEXTO
Duração em segundos: 79.429
Comparações feitas: 3199960000
Atribuições feitas: 240000
Total de operações simples: 3200200000

Process finished with exit code 0
```

```
Product{name='Technology', ID=197071, price=5250.0}
Product{name='Technology', ID=159677, price=5250.0}
Product{name='Technology', ID=944263, price=5250.0}
Product{name='Technology', ID=206354, price=5250.0}
Product{name='Technology', ID=174951, price=5250.0}
Product{name='Technology', ID=219992, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 31.5
Comparações feitas: 3199960000
Atribuições feitas: 240000
Total de operações simples: 3200200000

Process finished with exit code 0
```

INSERTION SORT

```
Product{name='Toys', ID=251400, price=35.84}
Product{name='Toys', ID=871117, price=35.84}
Product{name='Toys', ID=224865, price=143.36}
Product{name='Toys', ID=153324, price=143.36}
Product{name='Toys', ID=212867, price=179.2}
Product{name='Toys', ID=312663, price=71.68}
Critério de ordenação: TEXTO
Duração em segundos: 29.128
Comparações feitas: 1295197539
Atribuições feitas: 1295117541
Total de operações simples: 2590315080

Process finished with exit code 0
```

```
Product{name='Technology', ID=413656, price=5250.0}
Product{name='Technology', ID=428936, price=5250.0}
Product{name='Technology', ID=183059, price=5250.0}
Product{name='Technology', ID=270775, price=5250.0}
Product{name='Technology', ID=223470, price=5250.0}
Product{name='Technology', ID=197071, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 29.648
Comparações feitas: 1534971026
Atribuições feitas: 1534891032
Total de operações simples: 3069862058

Process finished with exit code 0
```

SELECT-INSERT SORT

```
Product{name='Toys', ID=251400, price=35.84}
Product{name='Toys', ID=871117, price=35.84}
Product{name='Toys', ID=224865, price=143.36}
Product{name='Toys', ID=153324, price=143.36}
Product{name='Toys', ID=212867, price=179.2}
Product{name='Toys', ID=312663, price=71.68}
Critério de ordenação: TEXTO
Duração em segundos: 23.367
Comparações feitas: 1121401217
Atribuições feitas: 120000
Total de operações simples: 1121521217

Process finished with exit code 0
```

```
Product{name='Technology', ID=413656, price=5250.0}
Product{name='Technology', ID=428936, price=5250.0}
Product{name='Technology', ID=183059, price=5250.0}
Product{name='Technology', ID=270775, price=5250.0}
Product{name='Technology', ID=223470, price=5250.0}
Product{name='Technology', ID=197071, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 17.513
Comparações feitas: 1561977342
Atribuições feitas: 120000
Total de operações simples: 1562097342

Process finished with exit code 0
```

QUICKSORT

```
Exception in thread "main" java.lang.StackOverflowError Create breakpoint
    at sorting.QuickSorter.quickSort(QuickSorter.java:17)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
    at sorting.QuickSorter.quickSort(QuickSorter.java:19)
```

É notada a presença de um Stack Overflow para a ordenação do texto dos registros. Como o arquivo possui muitos registros com chaves String equivalentes (já ordenadas) isso resulta no pior caso do quicksort e na extrapolação da Stack devido às chamadas recursivas.

```
Product{name='Technology', ID=995362, price=5250.0}
Product{name='Technology', ID=197071, price=5250.0}
Product{name='Technology', ID=158538, price=5250.0}
Product{name='Technology', ID=394903, price=5250.0}
Product{name='Technology', ID=199926, price=5250.0}
Product{name='Technology', ID=267842, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 1.395
Comparações feitas: 123868654
Atribuições feitas: 940137
Total de operações simples: 124808791

Process finished with exit code 0
```

MERGE SORT

```
Product{name='Toys', ID=260525, price=107.52}
Product{name='Toys', ID=181109, price=107.52}
Product{name='Toys', ID=258195, price=71.68}
Product{name='Toys', ID=161949, price=107.52}
Product{name='Toys', ID=195396, price=71.68}
Product{name='Toys', ID=992454, price=143.36}
Product{name='Toys', ID=640508, price=143.36}
Critério de ordenação: TEXTO
Duração em milissegundos: 329
Comparações feitas: 2712774
Atribuições feitas: 2617856

Process finished with exit code 0
```

```
Product{name='Technology', ID=318625, price=5250.0}
Product{name='Technology', ID=311849, price=5250.0}
Product{name='Technology', ID=252528, price=5250.0}
Product{name='Technology', ID=194017, price=5250.0}
Product{name='Technology', ID=883721, price=5250.0}
Product{name='Technology', ID=252275, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 0.355
Comparações feitas: 1200771
Atribuições feitas: 2617856
Total de operações simples: 3818627

Process finished with exit code 0
```

HEAP SORT

```
Product{name='Toys', ID=289707, price=143.36}
Product{name='Toys', ID=306192, price=179.2}
Product{name='Toys', ID=338732, price=107.52}
Product{name='Toys', ID=306980, price=35.84}
Product{name='Toys', ID=155310, price=71.68}
Product{name='Toys', ID=186663, price=35.84}
Critério de ordenação: TEXTO
Duração em segundos: 0.282
Comparações feitas: 1817739
Atribuições feitas: 2753400
Total de operações simples: 4571139

Process finished with exit code 0
```

```
Product{name='Technology', ID=127941, price=5250.0}
Product{name='Technology', ID=203708, price=5250.0}
Product{name='Technology', ID=263236, price=5250.0}
Product{name='Technology', ID=107407, price=5250.0}
Product{name='Technology', ID=237596, price=5250.0}
Product{name='Technology', ID=801128, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 0.293
Comparações feitas: 2314281
Atribuições feitas: 3604500
Total de operações simples: 5918781

Process finished with exit code 0
```

MÉTODO DA API COLLECTIONS (MERGE SORT)

```
Product{name='Toys', ID=338517, price=35.84}
Product{name='Toys', ID=671829, price=71.68}
Product{name='Toys', ID=468332, price=143.36}
Product{name='Toys', ID=251400, price=35.84}
Product{name='Toys', ID=871117, price=35.84}
Product{name='Toys', ID=224865, price=143.36}
Product{name='Toys', ID=153324, price=143.36}
Product{name='Toys', ID=212867, price=179.2}
Product{name='Toys', ID=312663, price=71.68}
Critério de ordenação: TEXTO
Duração em segundos: 0.255

Process finished with exit code 0
```

```
Product{name='Technology', ID=129821, price=5250.0}
Product{name='Technology', ID=911884, price=5250.0}
Product{name='Technology', ID=559261, price=5250.0}
Product{name='Technology', ID=413656, price=5250.0}
Product{name='Technology', ID=428936, price=5250.0}
Product{name='Technology', ID=183059, price=5250.0}
Product{name='Technology', ID=270775, price=5250.0}
Product{name='Technology', ID=223470, price=5250.0}
Product{name='Technology', ID=197071, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 0.241

Process finished with exit code 0
```

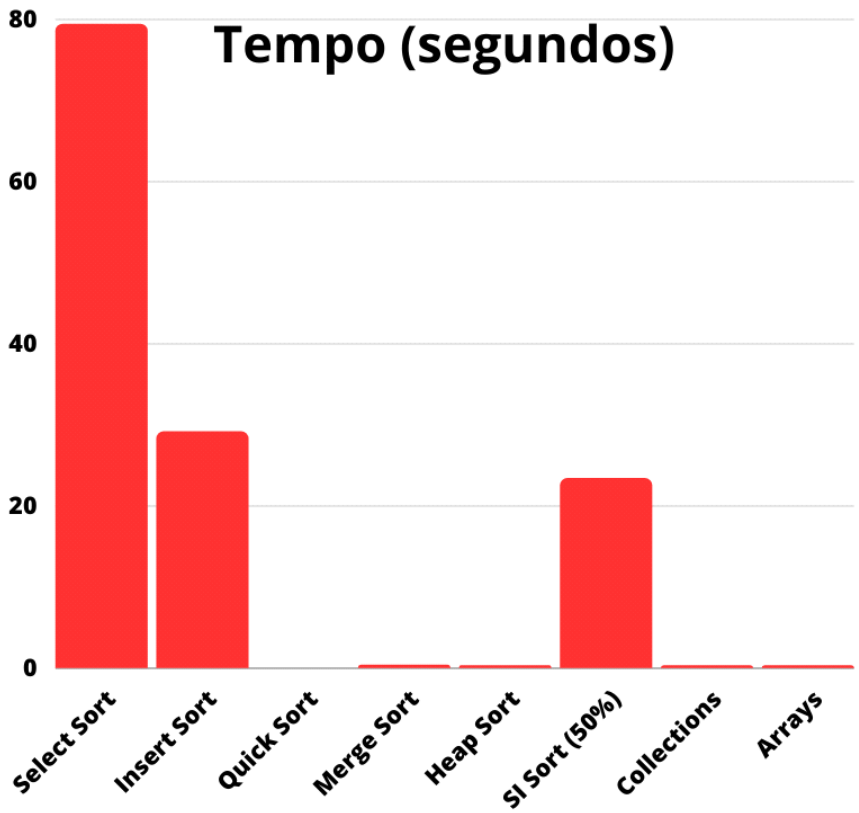
MÉTODO DA CLASSE ARRAYS(MERGE SORT ITERATIVO)

```
Product{name='Toys', ID=338517, price=35.84}
Product{name='Toys', ID=671829, price=71.68}
Product{name='Toys', ID=468332, price=143.36}
Product{name='Toys', ID=251400, price=35.84}
Product{name='Toys', ID=871117, price=35.84}
Product{name='Toys', ID=224865, price=143.36}
Product{name='Toys', ID=153324, price=143.36}
Product{name='Toys', ID=212867, price=179.2}
Product{name='Toys', ID=312663, price=71.68}
Critério de ordenação: TEXTO
Duração em segundos: 0.25

Process finished with exit code 0
```

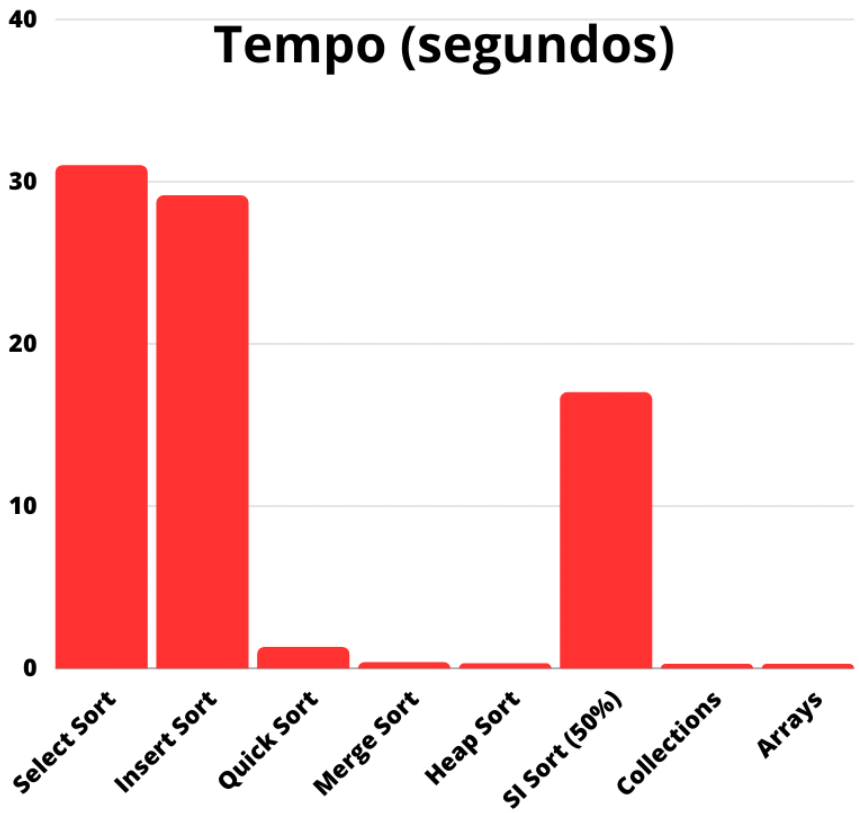
```
Product{name='Technology', ID=129821, price=5250.0}
Product{name='Technology', ID=911884, price=5250.0}
Product{name='Technology', ID=559261, price=5250.0}
Product{name='Technology', ID=413656, price=5250.0}
Product{name='Technology', ID=428936, price=5250.0}
Product{name='Technology', ID=183059, price=5250.0}
Product{name='Technology', ID=270775, price=5250.0}
Product{name='Technology', ID=223470, price=5250.0}
Product{name='Technology', ID=197071, price=5250.0}
Critério de ordenação: PREÇO
Duração em segundos: 0.252

Process finished with exit code 0
```



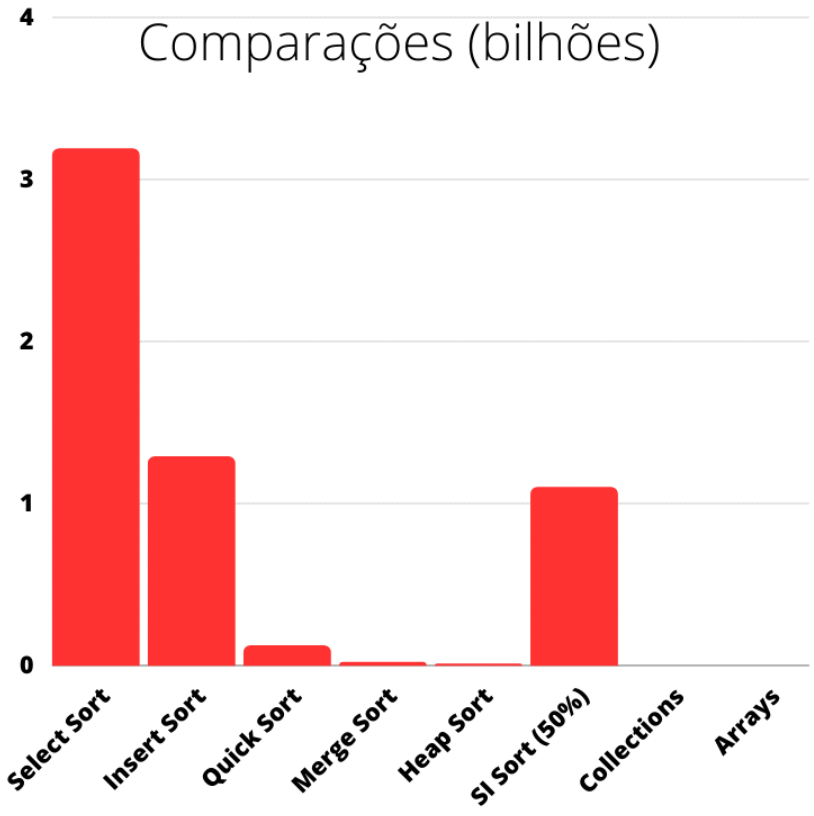
No quesito tempo de execução para ordenação do tipo **texto**, o Merge Sort e

Heap Sort apresentaram os melhores resultados, enquanto o Select Sort foi muito lento devido à necessidade de percorrer todo o vetor para selecionar o valor mínimo.

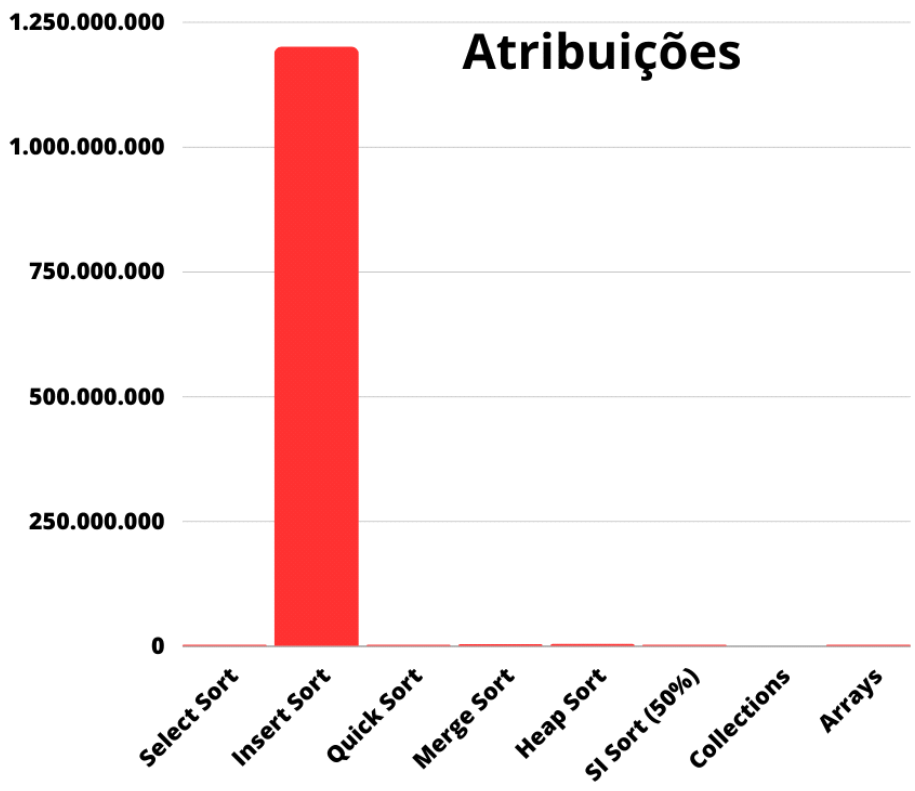


Já na ordenação do tipo **numérico**, Select e Insert apresentaram resultados bem parecidos.

Nota-se que o Heap Sort, junto com os métodos da API collections e da classe Arrays obtiveram as execuções mais rápidas



Pelo mesmo motivo anterior, o Select Sort apresentou o pior desempenho, com cerca de 3.19 bilhões de comparações feitas, enquanto o HeapSort realizou só cerca de 1.8 milhões, na ordenação tipo numérica.



Aqui podemos ver que o Insert Sort realiza mais de 1 bilhão de atribuições, devido à natureza das movimentações no algoritmo. Por outro lado, o Select-Insert Sort foi o que realizou menos atribuições, cerca de 120000, metade das atribuições realizadas pelo Select sort. Nota-se que esse resultado é devido à ordenação parcial pelo método de seleção, fazendo com que a inserção ocorra num vetor já quase ordenado, realizando menos operações. O gráfico de atribuições é praticamente idêntico para critérios texto e numérico.

CONCLUSÃO

Analisando os resultados com diferentes tamanhos de entrada, é notado que os métodos de seleção e inserção têm desempenho semelhante para valores pequenos e grandes de N. Porém, o método de inserção pode se destacar por realizar muito menos comparações quando o N é grande, além de realizar a ordenação de texto com mais eficiência.

Os demais métodos, quick, merge e heapsort tiveram desempenho muito similares em todos os aspectos, quando o tamanho da entrada é elevado, notando-se somente que o quicksort pode ter um desempenho arbitrário devido à escolha do pivô, que também é arbitrária e pode acarretar em um pior caso de

complexidade O(n^2), o que causou um erro na tentativa da ordenação de texto.

Considerando os testes realizados, foi notado que todos os algoritmos executam em tempos de execução quase idênticos para valores <= 1000, mostrando que para entradas pequenas até o algoritmo mais simples pode ser eficiente. Entretando, para valores grandes de N, os algoritmos que dividem o problema em problemas menores se destacam pela sua eficiência, devido à sua ordem de crescimento O(nlogn), fazendo com que o tempo de execução tenha variado no intervalo de 0.03s para 0.3s, considerando um vetor de 80000 elementos.

REFERÊNCIAS

https://docs.oracle.com/javase/tutorial/java/generics/types.html

https://refactoring.guru/design-patterns/template-method

https://refactoring.guru/design-patterns/strategy