Pontifícia Universidade Católica do Paraná - PUCPR

Aluno:

Lucas Fernando Assunção Cavalherie
Gabriel Maron Machado Lima
Renan Belem Biavati
Guilherme Chaves Pereira

TDE 4 - Projeto Colaborativo 2: Comparação de Algoritmos de Ordenação

CURITIBA - JUNHO

Introdução

Para o desenvolvimento do TDE, nós selecionamos os algoritmos de ordenação: QuickSort, ShellSort, HeapSort, InsertionSort, SelectionSort e MergeSort. Todos esses algoritmos foram implementados com fontes de foruns da internet. Para testarmos e tirarmos conclusões, nós testamos com diversos conjuntos diferentes, desde 1.000 até 1.000.000 de inteiros.

Configuração do computador que executa os algoritmos de ordenação:

Processador	Intel(R) Core(TM) i3-10100F CPU @ 3.60GHz
RAM instalada	16,0 GB
Tipo de sistema	Sistema operacional de 64 bits
Edição do sistema	Windows 11 Education
Versão do sistema	21H2

Caso de erro

Durante o desenvolvimento do projeto nos deparamos com um caso de erro com o quicksort. Por se tratar de um algoritmo recursivo o quicksort pode ter um estouro de pilha, o famoso *stacker overflow*, isso foi presenciado na execução do nosso algoritmo, mais especificamente ao tentar ordenar um conjunto decrescente, pois o quicksort busca recursivamente valores menor que o seu pivô, porém como o conjunto está em ordem decrescente, o pivô terá que realizar diversas chamadas recursivas, por conta disso, conjuntos decrescentes com tamanho superior a 15.000 números não foram possíveis de serem ordenados com o quicksort.

```
=================
Iniciando Sistema
=================
Informe o tamanho do array (Deve ser um inteiro):
20000
Exception in thread "main" java.lang.StackOverflowError Create breakpoint
    at QuickSort.quickSort(QuickSort.java:15)
    at QuickSort.quickSort(QuickSort.java:16)
```

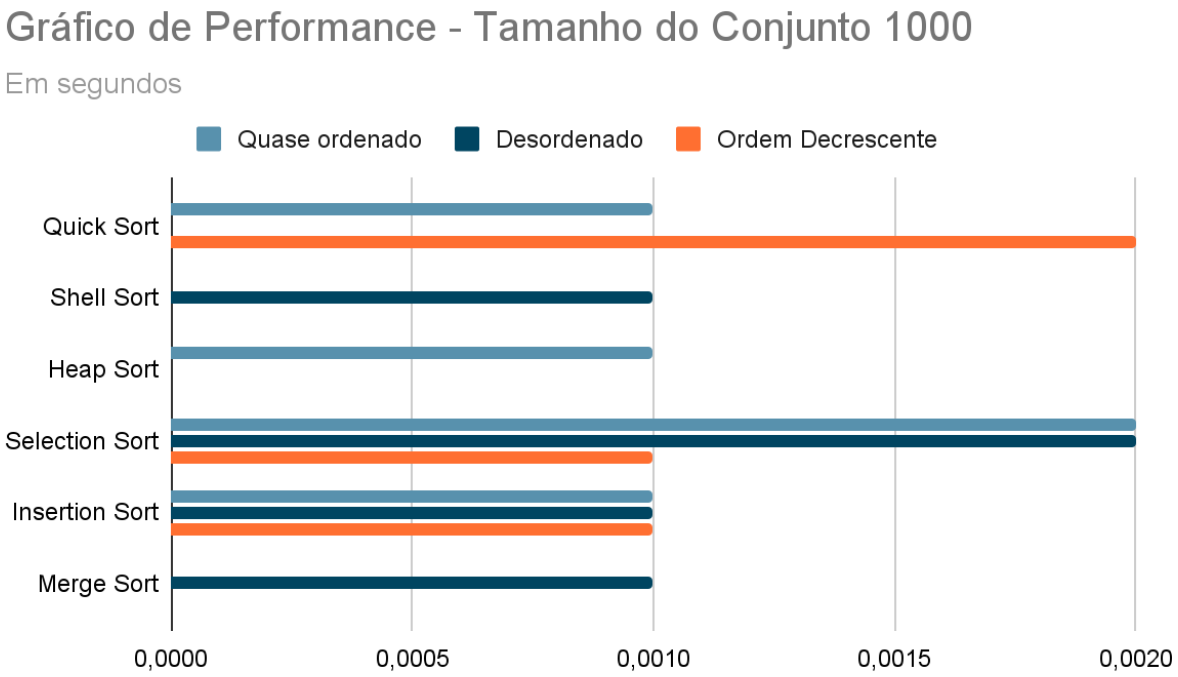
Por conta desse impeditivo a linha que executa essa ordenação foi comentada, dessa maneira o algoritmo não ficou limitado a somente conjuntos de 15.000 números.

```
QuickSort quickSort = new QuickSort();
quickSortTime[0] = quickSort.sort(conjuntoQuaseOrdenado.clone());
quickSortTime[1] = quickSort.sort(conjuntoDesordenado.clone());
//quickSortTime[2] = quickSort.sort(conjuntoDecrescente.clone());
```

Resultados finais

Tamanho do Conjunto 1000

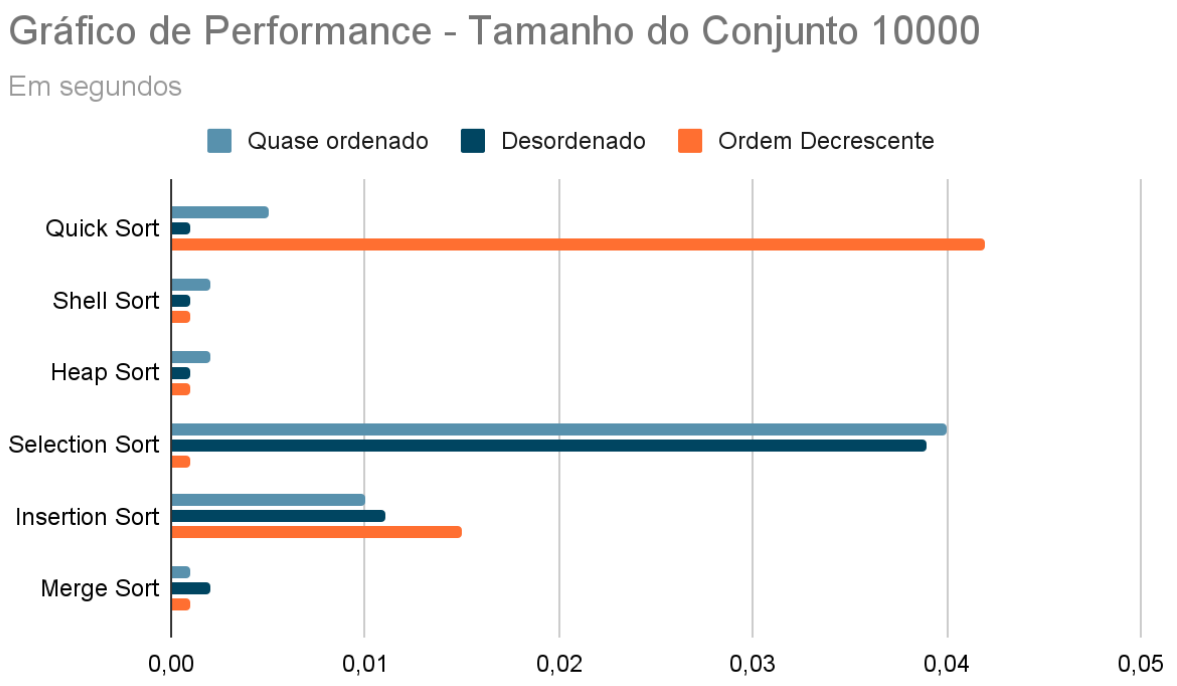
```
--------------------------------------------------------------------------------
Tamanho do Conjunto: 1000
--------------------------------------------------------------------------------
                                Tempos obtidos pelos algoritmos
--------------------------------------------------------------------------------
          Algoritmo |      Quase ordenado |         Desordenado |   Ordem decrescente |
         Quick Sort |                0.001s |                0.0s |               0.002s |
         Shell Sort |                  0.0s |              0.001s |                 0.0s |
          Heap Sort |                0.001s |                0.0s |                 0.0s |
     Selection Sort |                0.002s |              0.002s |               0.001s |
     Insertion Sort |                0.001s |              0.001s |               0.001s |
         Merge Sort |                  0.0s |              0.001s |                 0.0s |
--------------------------------------------------------------------------------
```



Todos algoritmos performaram rapidamente, levando em consideração também o pequeno tamanho do conjunto, Selection Sort e Quick Sort demoraram um pouco mais, o Quick pela necessidade de revirar o array em ordem contrária e o Select pela própria complexidade do mecanismo de busca foram os únicos que variaram, porém um valor muito pequeno e desprezível na prática.

Tamanho do Conjunto 10000

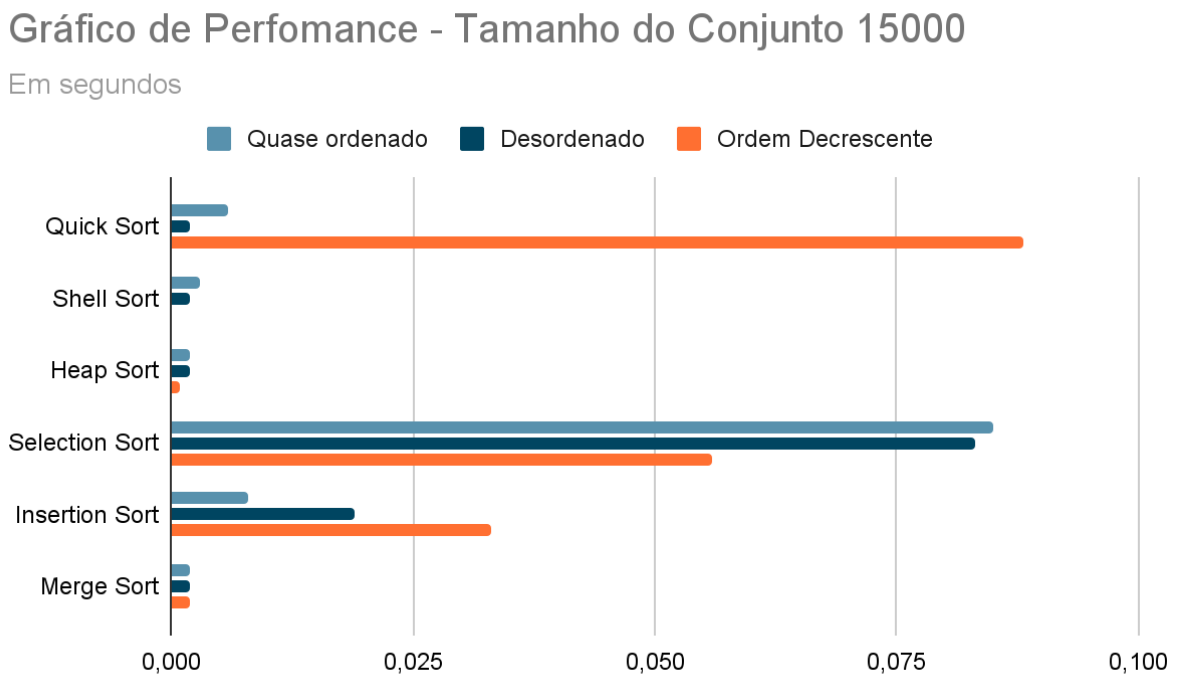
```
-------------------------------------------------------------------------------------
Tamanho do Conjunto: 10000
-------------------------------------------------------------------------------------
                                  Tempos obtidos pelos algoritmos
-------------------------------------------------------------------------------------
          Algoritmo |      Quase ordenado |         Desordenado |   Ordem decrescente |
         Quick Sort |                0.005s |              0.001s |              0.042s |
         Shell Sort |                0.002s |              0.001s |              0.001s |
          Heap Sort |                0.002s |              0.001s |              0.001s |
     Selection Sort |                 0.04s |              0.039s |              0.031s |
     Insertion Sort |                 0.01s |              0.011s |              0.015s |
         Merge Sort |                0.001s |              0.002s |              0.001s |
-------------------------------------------------------------------------------------
```



O mesmo da situação de 1000 é observável no conjunto de 10000 números, no Quick quando o array é quase ordenado e desordenado não é prejudicado o tempo de resposta, mas a ordem decrescente torna muito mais difícil a ordenação de números do Quick porque o último número (parâmetro do Quick Sort) é o menor, o Selection continua crescendo seus valores do tempo de conclusão, porque de fato é um algoritmo mais demorado pela procura de números e ordenação, já o insert também começa a aparecer como algoritmo lento, devido sua ordenação que varre número a número e os organiza crescentemente, os outros mantém um bom desempenho.

Tamanho do Conjunto 15000

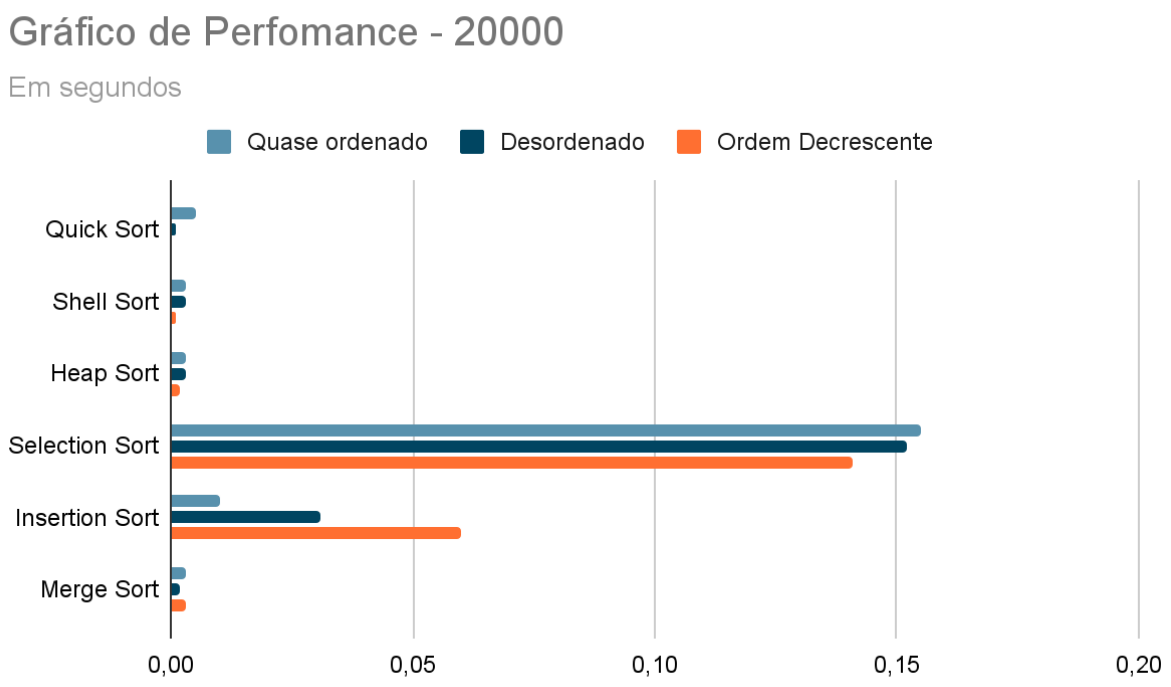
```
Informe o tamanho do array (Deve ser um inteiro):
15000

-------------------------------------------------------------------------------------------
Tamanho do Conjunto: 15000

-------------------------------------------------------------------------------------------
                                    Tempos obtidos pelos algoritmos

-------------------------------------------------------------------------------------------
          Algoritmo |      Quase ordenado |         Desordenado |   Ordem decrescente |
          Quick Sort |              0.006s |              0.002s |              0.088s |
          Shell Sort |              0.003s |              0.002s |                0.0s |
           Heap Sort |              0.002s |              0.002s |              0.001s |
      Selection Sort |              0.085s |              0.083s |              0.056s |
      Insertion Sort |              0.008s |              0.019s |              0.033s |
          Merge Sort |              0.002s |              0.002s |              0.002s |

-------------------------------------------------------------------------------------------

Process finished with exit code 0
```



No conjunto de 15000 números novamente a análise se mantém, porém, se intensifica, o Quick Sort é espantosamente lento para lidar com a ordem decrescente de números e o Selection continua tendo um padrão muito lento de desempenho também, principalmente quando o array está quase ordenado e desordenado, já que quando está em ordem decrescente ele não precisa revirar tanto o array como nos outros casos para buscar os números e organizá-los, o Insertion também aumenta consideravelmente o tempo em relação aos outros algoritmos por que sua ordenação depende da busca número a número.

Tamanho do Conjunto 20000

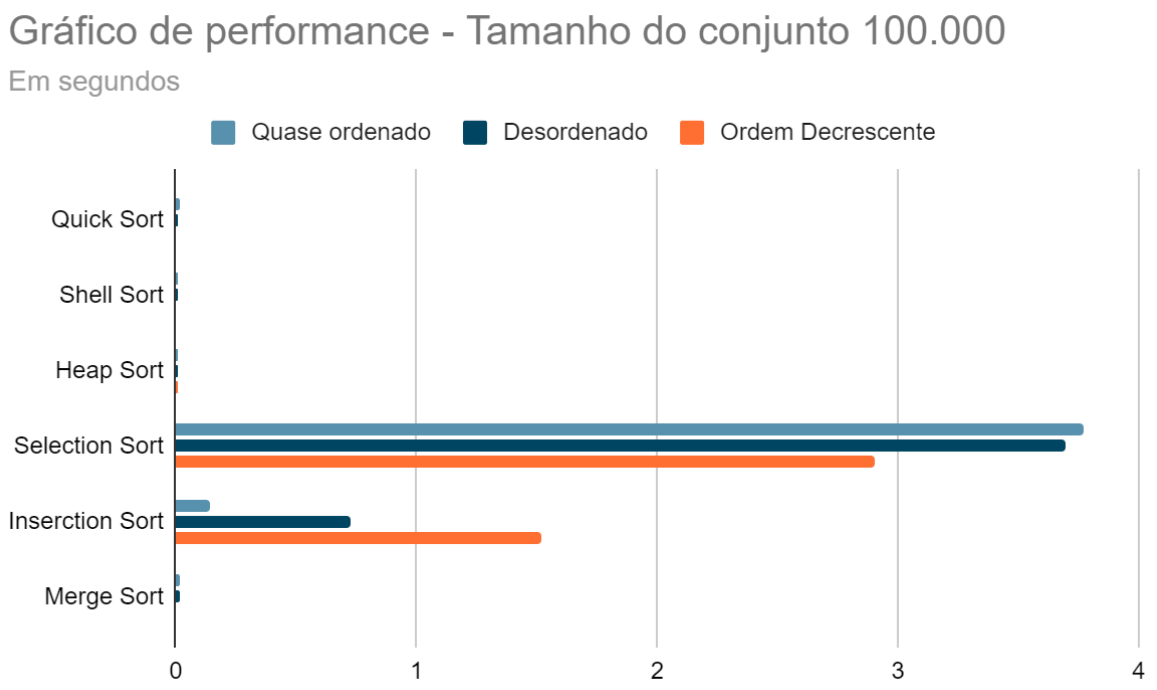
```
-------------------------------------------------------------------------------------------
Tamanho do Conjunto: 20000
-------------------------------------------------------------------------------------------
                                    Tempos obtidos pelos algoritmos
-------------------------------------------------------------------------------------------
          Algoritmo |      Quase ordenado |         Desordenado |   Ordem decrescente |
         Quick Sort |                0.005s |              0.001s |                null |
         Shell Sort |                0.003s |              0.003s |              0.001s |
          Heap Sort |                0.003s |              0.003s |              0.002s |
     Selection Sort |                0.155s |              0.152s |              0.141s |
     Insertion Sort |                 0.01s |              0.031s |               0.06s |
         Merge Sort |                0.003s |              0.002s |              0.003s |
-------------------------------------------------------------------------------------------
```



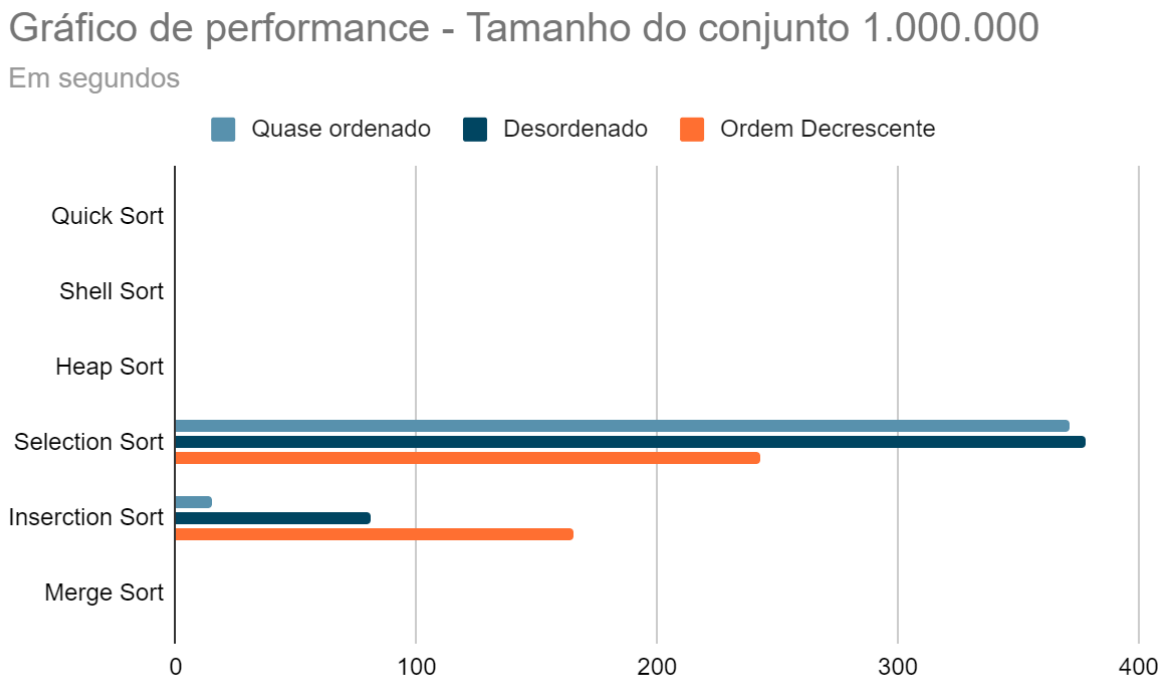
A partir deste gráfico a ordem decrescente do array no Quick Sort não é executado pelo erro mencionado anteriormente. O gráfico de 20.000 mostra como o Selection Sort é o algoritmo mais lento e demorado a ser ordenado devido sua necessidade de ir atrás dos menores valores e organizá-los, em sequência o Insertion Sort, se destaca no tempo de execução quando o array está em ordem decrescente, o que dificulta a sua ordenação que passa valor a valor com o array invertido, o conjunto desordenado demora mais que o conjunto quase ordenado, pois é mais lento ordenar valores distribuídos de maneira aleatória do que com cerca de 90% ordenado, os outros mantém uma média de resposta muito similar de tempo a serem concluídos.

Tamanho do Conjunto 100.000 e 1.000.000

```
--------------------------------------------------------------------------------
Tamanho do Conjunto: 100000
--------------------------------------------------------------------------------
                                Tempos obtidos pelos algoritmos
--------------------------------------------------------------------------------
          Algoritmo |      Quase ordenado |         Desordenado |   Ordem decrescente |
         Quick Sort |                0.015s |              0.007s |                null |
         Shell Sort |                 0.01s |               0.01s |              0.002s |
          Heap Sort |                0.011s |               0.01s |              0.006s |
     Selection Sort |                3.772s |              3.693s |              2.907s |
     Insertion Sort |                0.145s |               0.73s |               1.52s |
         Merge Sort |                0.016s |              0.014s |              0.004s |
--------------------------------------------------------------------------------
```




```
--------------------------------------------------------------------------------
Tamanho do Conjunto: 1000000
--------------------------------------------------------------------------------
                                Tempos obtidos pelos algoritmos
--------------------------------------------------------------------------------
          Algoritmo |      Quase ordenado |         Desordenado |   Ordem decrescente |
         Quick Sort |                0.148s |              0.081s |                null |
         Shell Sort |                0.106s |              0.147s |              0.026s |
          Heap Sort |                0.087s |              0.132s |               0.07s |
     Selection Sort |              370.947s |            378.322s |            243.043s |
     Insertion Sort |               15.343s |             80.579s |             165.37s |
         Merge Sort |                0.079s |              0.138s |              0.046s |
--------------------------------------------------------------------------------
```



No gráfico do conjunto de 100.000 e 1.000.000 de números, são muito pequenas as barras de tempo do Quick, Shell, Heap e Merge Sort, que são algoritmos de estrutura mais simples e consequentemente exigiram menos da máquina para serem concluídos, diferentemente do Selection e Insertion Sort, que possuem instruções definidamente mais “burocráticas” e lentas quando aplicadas principalmente em um grande array e por isso levaram um tempo maior para serem realizadas, principalmente o desordenado e quase ordenado do Selection Sort, porque sua ordenação é dependente da sequência crescente de valores que é prejudicada nas situações de array desordenado e quase ordenado, já que troca de posições valores que estão em posições aleatórias, e por isso a ordem decrescente se organiza mais rapidamente.

Referências Bibliográficas

LINK DO REPLIT:
https://www.devmedia.com.br/algoritmos-de-ordenacao-em-java/32693
https://www.baeldung.com/java-quicksort
https://www.baeldung.com/java-shell-sort
https://www.geeksforgeeks.org/heap-sort/
https://www.javatpoint.com/selection-sort-in-java
https://www.javatpoint.com/insertion-sort-in-java
https://www.javatpoint.com/merge-sort