Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Identificação do aluno,

Flávia Vitória Gonçalves de Queiroz - 212080172

Maria Luiza Almeida Leite - 212080334

1. Introdução

Esse relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de LEDA, que tem como fito implementar e analisar os algoritmos de ordenação, como Selection Sort, Insertion Sort, Merge Sort, Quicksort, QuickSort com Mediana de 3, Counting, e HeapSort,através da utilização de dados. Nesse projeto, foram analisados um conjunto de dados chamados de “passwords.csv”,que contém informações sobre as senhas,como tamanho,data e classificação.

Antes da análise dos algoritmos,foram realizadas algumas transformações nos arquivos.

- Classificação de senhas:

  O processo de classificação de senhas consistiu na análise dos dados presentes no arquivo “passwords.csv”, a partir da qual foi possível identificar e classificar cada senha de acordo com o tipo correspondente.Para essa classificação,foram considerados critérios como o tamanho da senha e a presença de diferentes tipos de caracteres.

  Muito Ruim: tamanho da string menor que 5, e só um tipo de caractere, por exemplo: só letra (letras minúsculas ou maiúsculas), só número ou só caractere especial.

  Ruim: tamanho da string menor ou igual a 5, e só um tipos de caracteres, por exemplo: letra (letras minúsculas ou maiúsculas), número ou caractere especial.

  Fraca: tamanho da string menor ou igual a 6, e só dois tipos de caracteres, por exemplo: letra(letras minúsculas e maiúsculas), número ou caractere especial.

  Boa: tamanho da string menor ou igual a 7, e só todos de caracteres, por exemplo: letra(letras minúsculas e maiúsculas), número ou caractere especial.

  Muito Boa: tamanho da string maior que 8, e só todos os tipos de caracteres, por exemplo: letra(letras minúsculas e maiúsculas), número ou caractere especial

  Sem Classificação: Senhas que não se qualificam com nenhuma das classificações acima.

  A partir dessas análises,foi criada uma nova coluna chamada “class” e os dados foram salvos em um novo arquivo chamado “password_classifier.csv”.

- Transformar data:

  A coluna que contém o campo de datas no arquivo “password_classifier.csv” apresenta variações em seu formato,sendo assim, necessário a mudança desse formato para realizar a análise dos dados.Dessa forma,foi efetuado um processo de tratamento que consistiu na identificação das diferentes variações de formato presente na coluna de datas, e na aplicação de métodos de manipulação nas datas para realizar a padronização das mesmas para um formato específico(DD/MM/AAAA).

- Filtrar senha pela categoria Boa e Muito Boa:

  Após a realização do processo de classificação de senhas,é preciso filtrar as senhas categorizadas como Boa e Muito Boa através da coluna “class” do arquivo “password_classifier.csv” e adicioná-las a um novo arquivo intitulado “passwords_classifier.csv”.

- Ordenações:

  Para as ordenações, são considerados sete algoritmos de ordenação, o Selection Sort, Insertion Sort, Merge Sort, Quicksort, Quicksort com Mediana de 3, Counting, e HeapSort. E o arquivo de entrada utilizado para ser ordenado é o “passwords_formated_data.csv” .

  Primeira ordenação: é uma ordenação decrescente, que ordena de acordo com o campo “length”, depois de ordenado ele vai gerar os arquivos de melhor, médio e pior caso para cada algoritmo.

  Segunda ordenação: é a ordenação crescente de acordo com o mês no campo data, depois de ordenado ele vai gerar os arquivos de melhor, médio e pior caso para cada algoritmo.

  Terceira ordenação: é a ordenação crescente do campo data, depois de ordenado ele vai gerar os arquivos de melhor, médio e pior caso para cada algoritmo.

Os principais resultados alcançados com relação a classificação de senhas e a filtragem,é o retorno de arquivos que mostram a classificação da senha,seguindo os critérios estabelecidos, e um novo arquivo que mostra a filtragem das senhas consideradas Boa e Muito Boa.Outro resultado encontrado em relação a

transformação do campo de datas é a obtenção de um arquivo em que a coluna datas se encontra formatada de acordo com as exigências estabelecidas.

Com relação aos algoritmos de ordenação, foi obtido o retorno de todos os arquivos ordenados de acordo com cada especificação e para cada caso de execução.

2. Descrição geral sobre o método utilizado

Os testes foram realizados na IDE Visual Studio Code utilizando a linguagem de programação Java.Com o fito de deixar o projeto mais organizado foi utilizado herança na produção dos códigos.

A priori, foram realizados os primeiros testes nos tópicos de classificação de senha,transformação de formatação das datas e na filtragem das senhas.Onde foi analisado o uso da CPU,memória RAM e tempo de execução com o fito de avaliar a eficiência dessas transformações.

Para os testes realizados a partir do uso de algoritmos de ordenação,foi realizado uma comparação de todos os algoritmos em relação ao que apresenta o melhor desempenho.Também foram analisados como cada algoritmo de ordenação se comportou em relação ao melhor,médio e pior caso.

Descrição geral do ambiente de testes

Os testes foram executados em um notebook Dell Inc. Inspiron 3576 com memória de 8,0 GBytes, com um o processador Intel(R) Core(™) i7-8550U CPU @ 1.80GHz × 8 e sistema operacional Ubuntu 22.04.

Os testes foram executados em um computador com memória de 16,0GB, com um processador Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz 3.70 GHz e sistema operacional Windows 10.

3. Resultados e Análise

Elaborar os resultados dos testes usando tabelas e gráficos

Para análise dos algoritmos consideramos o arquivo “passwords.csv” com apenas 22.330 linhas.

Os algoritmos de ordenação analisados mostram desempenhos diferentes em relação aos campos "length" e "mês".

Ordenação pelo campo length (em milésimos de segundos)

Algoritmos	Melhor caso	Médio caso	Pior caso
Selection Sort	40680 ms	29860 ms	42710 ms
Insertion Sort	15580 ms	2500 ms	6820 ms
Merge Sort	1900 ms	1450 ms	2100 ms
QuickSort	1980 ms	1930 ms	2380 ms
QuickSort com Mediana de três	2520 ms	2970 ms	3310 ms
Counting	2700 ms	1590 ms	1710 ms
HeapSort	1650 ms	2440 ms	1580 ms

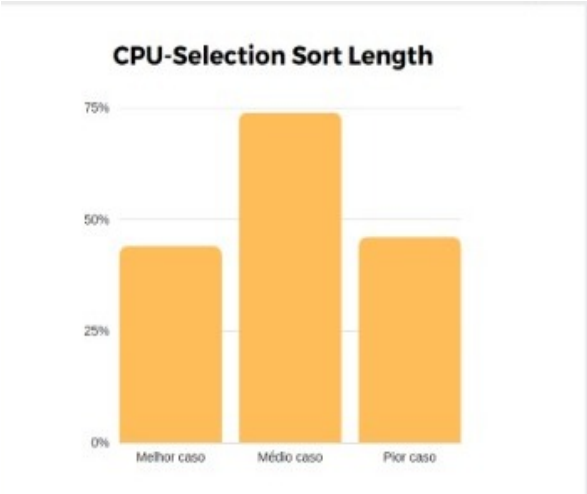
No caso do campo "length", o Selection Sort é o mais lento em todos os casos, enquanto o Insertion Sort têm um desempenho razoável. O Merge Sort, QuickSort e QuickSort com Mediana de Três têm desempenhos semelhantes, sendo mais eficientes que o Selection Sort e o Insertion Sort no pior caso. O Counting apresenta tempos de execução estáveis, e o HeapSort é rápido no melhor caso.

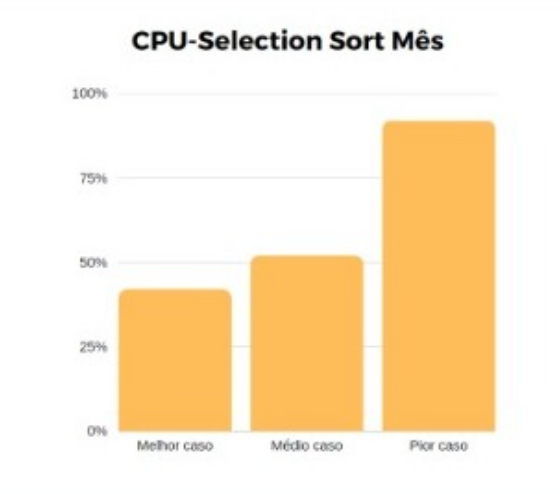
Ordenação pelo mês (em milésimos de segundos)

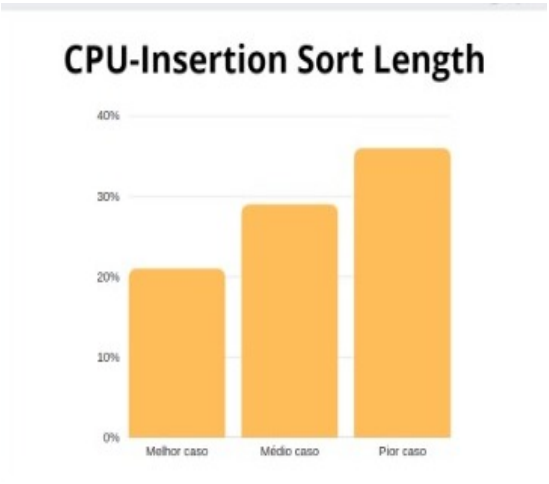
Algoritmos	Melhor caso	Médio caso	Pior caso
Selection Sort	53703 ms	50462 ms	65690 ms
Insertion Sort	2384 ms	0282 ms	3337 ms

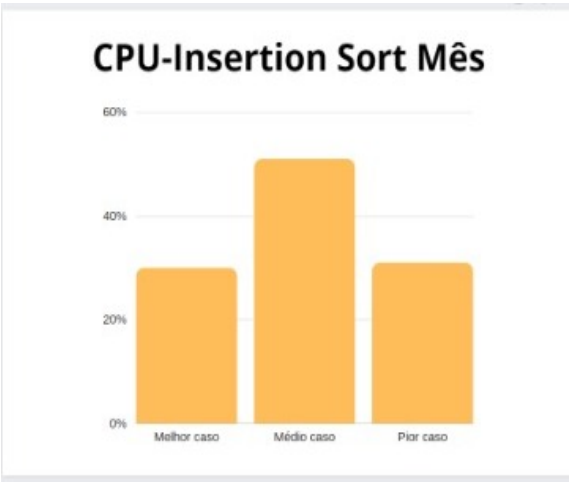
Merge Sort	210 ms	195 ms	78 ms
QuickSort	838 ms	753 ms	793 ms
QuickSort com Mediana de três	826 ms	774 ms	858 ms
Counting	146 ms	132 ms	237 ms
HeapSort	184 ms	237 ms	257 ms

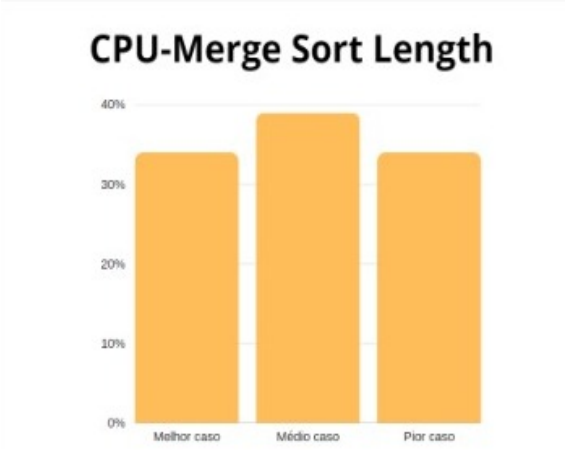
Já em relação ao campo "mês", o Selection Sort continua sendo o mais lento em todos os casos, com relação ao Insertion Sort ele é rápido no melhor caso. O Merge Sort se destaca como o mais rápido em todos os casos, seguido pelo QuickSort e QuickSort com Mediana de Três. O Counting também é rápido em todos os casos, enquanto o HeapSort é um pouco mais lento.

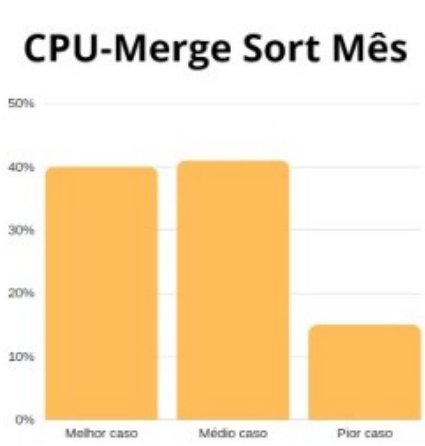


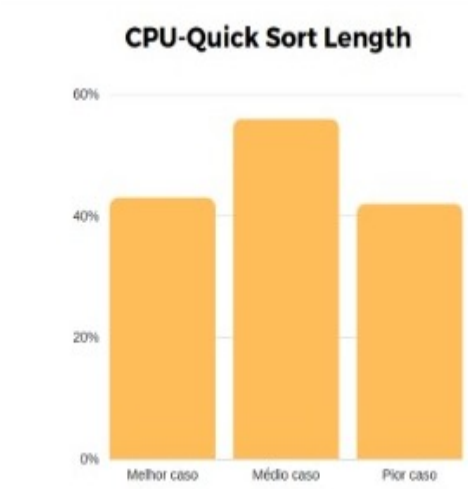


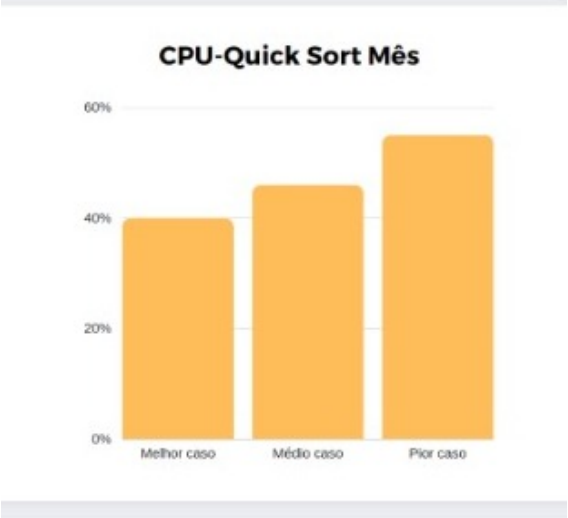

CPU-Insertion Sort Length
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso

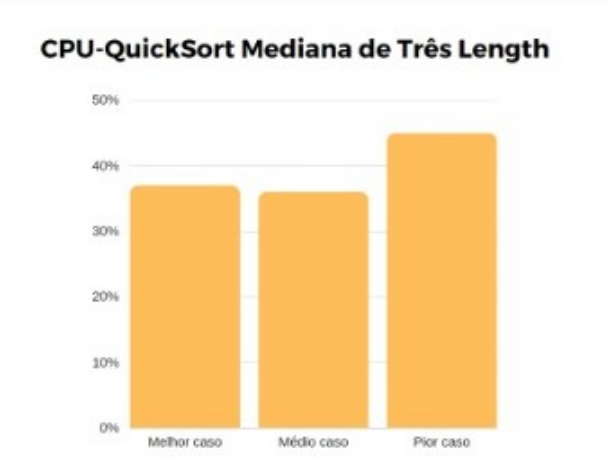

CPU-Insertion Sort Mês
60%
40%
20%
0%
Melhor caso
Médio caso
Pior caso

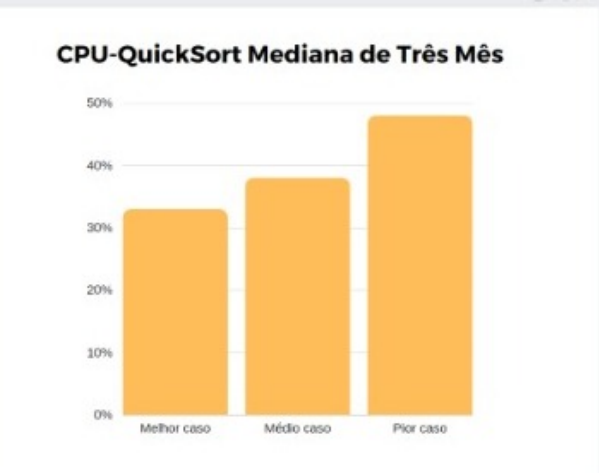

CPU-Merge Sort Length
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso

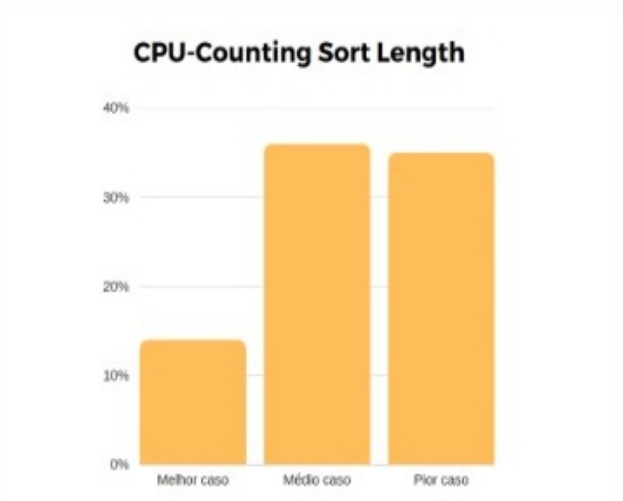

CPU-Merge Sort Mês
50%
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso

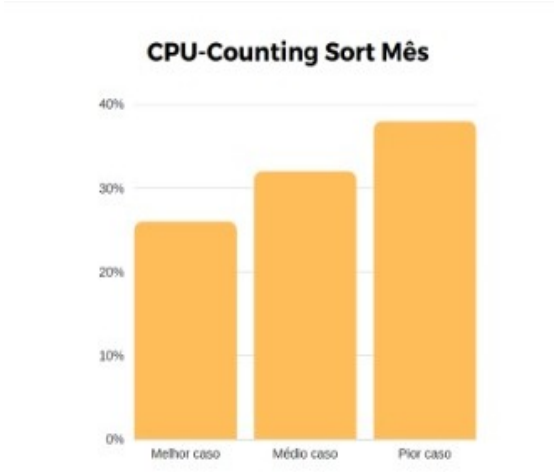

CPU-Quick Sort Length
60%
40%
20%
0%
Melhor caso
Médio caso
Pior caso

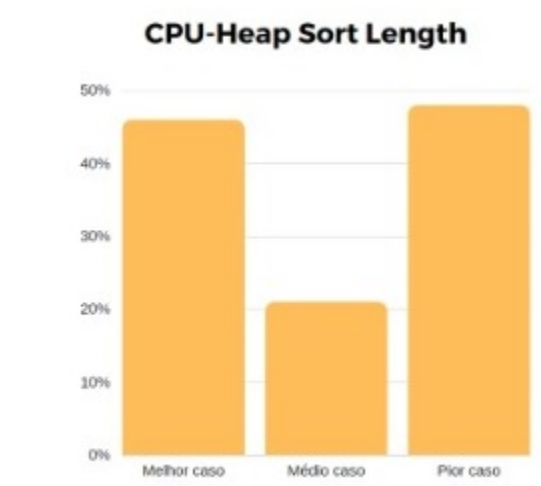

CPU-Quick Sort Mês
60%
40%
20%
0%
Melhor caso
Médio caso
Pior caso

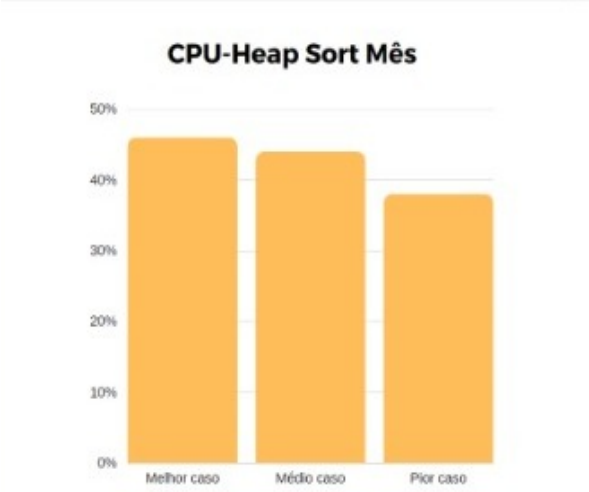

CPU-QuickSort Mediana de Três Length
50%
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso


CPU-QuickSort Mediana de Três Mês
50%
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso


CPU-Counting Sort Length
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso


CPU-Counting Sort Mês
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso


CPU-Heap Sort Length
50%
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso


CPU-Heap Sort Mês
50%
40%
30%
20%
10%
0%
Melhor caso
Médio caso
Pior caso

O Selection Sort é o algoritmo que mais exige uso de CPU, seguido pelo HeapSort, ambos com uso moderado. O Insertion Sort e o Merge Sort têm um uso moderado da CPU.O Quicksort e o Quicksort com Mediana de 3 têm um uso variável da CPU, mas em média são rápidos e eficientes. O Counting Sort tem um uso mínimo de CPU e é muito eficiente, enquanto não envolve comparações.

Algoritmos mais eficientes:

Entre os algoritmos analisados, os mais eficientes em termos de desempenho de tempo são o Merge Sort, QuickSort (com ou sem Mediana de três) e o Counting Sort. O Merge Sort divide a lista em sublistas menores e as combina de forma ordenada, com uma complexidade média e pior caso de O(n log n). O QuickSort realiza menos comparações e pode ser melhorado com a escolha da Mediana de três, evitando casos indesejáveis. O Counting Sort é altamente eficiente quando o intervalo de valores é pequeno, com uma complexidade linear de O(n+k), sendo ideal em cenários específicos.

Os algoritmos de ordenação diferem em seu uso da CPU durante a execução. Algoritmos como Selection Sort e Insertion Sort tendem a ter um uso mais intenso da CPU, enquanto algoritmos como Counting Sort têm um uso mínimo da CPU. Algoritmos como Merge Sort, Quicksort e HeapSort têm um uso moderado da CPU.

Os algoritmos de ordenação que utilizam menos memória são o Selection Sort, o Insertion Sort e o HeapSort. O Counting Sort também é eficiente no uso de memória, porém, requer uma quantidade de memória proporcional ao intervalo de valores a serem ordenados. O Merge Sort e o QuickSort usam mais memória, mas são eficientes em grandes conjuntos de dados. O Quicksort com Mediana de 3 é eficiente em termos de uso de memória, não exigindo uma quantidade significativa de memória adicional.

Análise geral dos resultados:

Entre os algoritmos analisados, Merge Sort, QuickSort (com ou sem Mediana de três) e Counting Sort são os mais eficientes em termos de desempenho de tempo. Algoritmos como Selection Sort e Insertion Sort têm um uso mais intenso da CPU, enquanto algoritmos como Counting Sort têm um uso mínimo da CPU. Os algoritmos que utilizam menos memória são Selection Sort, Insertion Sort e HeapSort, enquanto Merge Sort e QuickSort usam mais memória, mas são eficientes em grandes conjuntos de dados.