Centro universitário das Faculdades

Metropolitanas Unidas – FMU

Análise e Desenvolvimento de Sistemas

P.O.T.A

APS

André Bezerra Ribeiro RA: 7343674

Denilson Elias de Souza Junior RA: 3324643

Jéssica Adriana Feitosa RA: 2146934

Juliana dos Santos Lima RA: 3895943

Lucas Silva Rodrigues de Oliveira RA: 3851869

São Paulo

2021

P.OT.A

APS

Pesquisa apresentada no curso de graduação de

Análise e Desenvolvimento de sistemas das Faculdades

Metropolitanas Unidas. Orientador Orlando JR.

São Paulo

2021

Atividade proposta:

Analisar o tamanho da entrada de dados para um programa de computador é importante para a escolha dos melhores algoritmos e estruturas de dados que serão utilizadas no desenvolvimento. Por exemplo, se em um programa é necessário ordenar por nota um conjunto de alunos de uma turma com 30 alunos, qualquer algoritmo de ordenação poderá ser utilizado. Porém, se é necessário ordenar um vetor com 1 milhão de compras em um e-commerce, a escolha do algoritmo de ordenação terá um grande impacto no tempo de execução da aplicação.

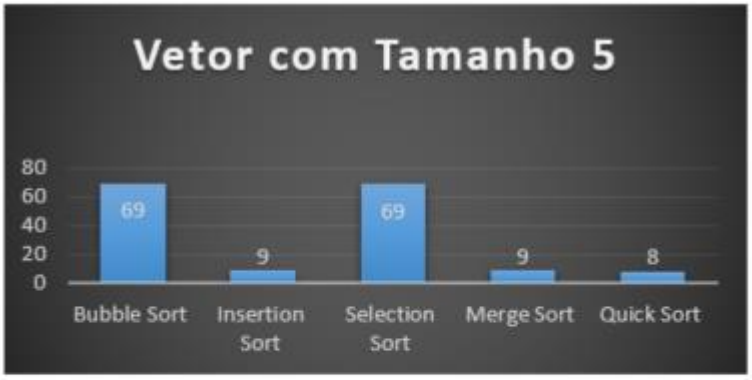
Baseado no problema descrito, realizar um estudo comparativo entre os algoritmos de ordenação Bubble Sort, Selection Sort, Insertion Sort, Merge Sort e Quick Sort utilizando como parâmetro o número de comparações realizadas por cada algoritmo. Deverão ser gerados 50 vetores de inteiros aleatórios com cada um dos seguintes tamanhos: 5, 10, 50, 100, 1.000 e 10.000. Em seguida, ordenar todos os vetores através de cada um dos métodos de ordenação propostos e contar o número de comparações entre os elementos em cada ordenação realizada.

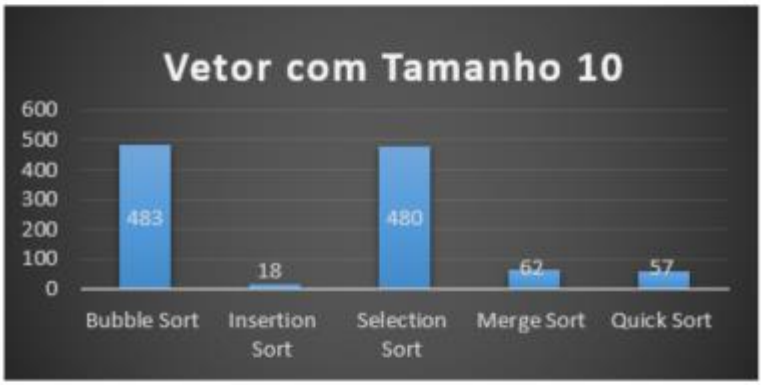
RESUMO

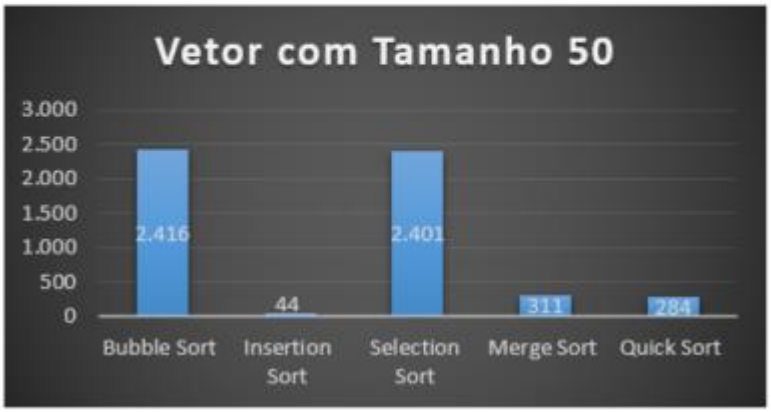
Este estudo, tem como objetivo, avaliar o tempo de execução de alguns algoritmos de ordenação. Eles são: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort e Quick Sort.

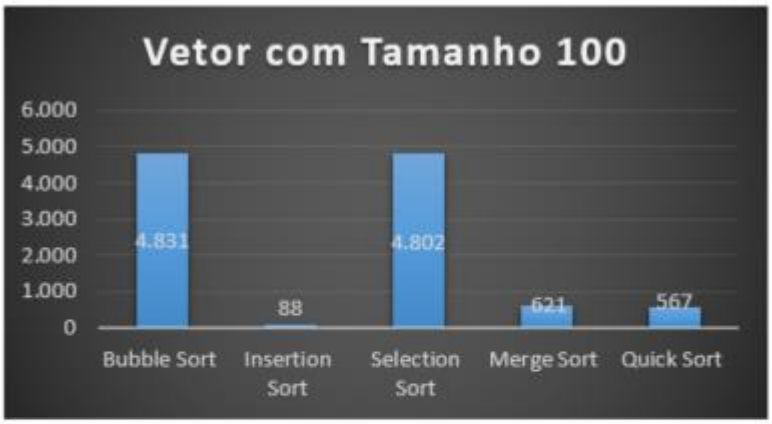
A extração foi realizada, utilizando como parâmetro o número de comparações realizadas por cada algoritmo.
Foram gerados 50 vetores de inteiros aleatórios, com cada um dos seguintes tamanhos: 5, 10, 50, 100, 1.000 e 10.000.

Quantidade de comparações, de acordo com cada algoritmo de ordenação.

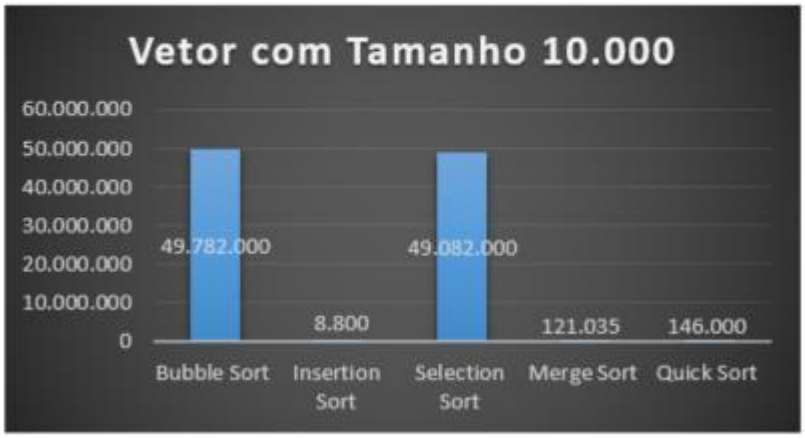







Vetor com Tamanho 100
6.000
5.000
4.000
3.000
2.000
1.000
0
4.831
88
4.802
621
567
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Quick Sort


Vetor com Tamanho 1.000
6.000.000
4.000.000
2.000.000
0
4.978.200
880
4.908.200
12.104
14.600
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Quick Sort


Vetor com Tamanho 10.000
60.000.000
50.000.000
40.000.000
30.000.000
20.000.000
10.000.000
0
49.782.000
8.800
49.082.000
121.035
146.000
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Quick Sort

Como foram gerados os vetores aleatórios?

Os vetores aleatórios foram gerados com números de 0 a 10.000 e com a Classe Random.

Como foram adaptados os algoritmos para realizar a contagem do número de comparações?

Foi feito um for, que vai até N (tamanho do vetor). Dentro dele temos outro for que vai até no máximo *N – 1(0)*.

Explicando os resultados dos experimentos:

Bubble Sort, apesar de ser o de mais fácil implementação, não apresenta resultados satisfatórios, principalmente no número de comparações. A ineficiência desse algoritmo pode ser traduzida como um grande consumo de processamento, o que, para máquinas com poucos (ou limitados) recursos computacionais, resulta em lentidão e longos períodos de espera.

Insertion Sort, é útil para estruturas lineares pequenas, tendo ainda, listas ordenadas de forma decrescente como pior caso, listas em ordem crescente como o melhor caso e, as demais ordens como sendo casos medianos. Sua principal vantagem é o pequeno número de comparações, e, o excessivo número de trocas, sua desvantagem.

Selection Sort torna-se útil em estruturas lineares similares ao do Insertion Sort, porém, com o número de elementos consideravelmente maior, já que, o número de trocas é muito inferior ao número de comparações, consumindo, assim, mais tempo de leitura e menos de escrita. Ambos os algoritmos (Insertion e Selection), apesar de suas diferentes características, são mais comumente utilizados em associação com outros algoritmos de ordenação, como os Merge Sort e o Quick Sort, que tendem a subdividir as listas a serem organizadas em listas menores, fazendo com que sejam mais eficientemente utilizados.

Merge Sort apresenta-se, em linhas gerais, como um algoritmo de ordenação mediano.

Devido a recursividade ser sua principal ferramenta, seu melhor resultado vem ao lidar com estruturas lineares aleatórias.

Esse algoritmo é indicado para quando se lida com estruturas lineares em que a divisão em estruturas menores sejam o objetivo, como, por exemplo, em filas para operações bancárias.

Quick Sort, ao subdividir o vetor e fazer inserções diretas utilizando um valor de referência (pivô), reduz seu tempo de execução, mas, as quantidades de comparações e, principalmente, trocas (escrita) ainda são muito altas. Apesar disso, o Quick Sort se apresenta uma boa opção para situações em que o objetivo é a execução em um menor tempo, mesmo que para isso haja um detrimento em recursos computacionais de processamento.