Belo Horizonte, Junho/2019

UNIVERSIDADE FEDERAL DE MINAS GERAIS

Departamento de Ciência da Computação

Bacharelado em Sistemas de Informação

Biblioteca Digital de Arendelle

ESTRUTURA DE DADOS - TRABALHO PRÁTICO 2

ÍCARO HENRIQUE VIEIRA PINHEIRO

2018046556

INTRODUÇÃO

A Ideia desse trabalho prático é analisar os comportamentos, vantagens e desvantagens da utilização do algoritmo Quicksort considerando cada uma de suas variações conhecidas. O Quicksort em geral é baseado na premissa de “Dividir para conquistar”, a ideia principal é escolher um pivô(dividir), e ordenar os elementos contidos em cada lado dessa divisão(conquistar).

Inicialmente a proposta consistia em analisar sete diferentes implementações do Quicksort, sendo elas:

- Quicksort clássico
- Quicksort mediana de três
- Quicksort primeiro elemento
- Quicksort inserção 1%
- Quicksort inserção 5%
- Quicksort inserção 10%
- Quicksort não recursivo

O Algoritmo deveria ser implementado a fim de ser testado em vetores ordenados de forma crescente, decrescente e de maneira aleatória, possibilitando assim um maior cenário para testes. Além disso, cada uma das diferentes versões do Quicksort deveria ser testada em vetores de tamanhos diferentes, variando entre 50 mil e 500 mil elementos (em intervalos de 50 mil elementos).

Como objeto de análise nesse trabalho vamos utilizar o número de comparações feitas entre os elementos do vetor, o número de trocas de elementos de posição efetuadas durante a ordenação e o tempo gasto para que o algoritmo execute a ordenação com sucesso. Ademais, foi ressaltado a importância de realizar os testes repetidas vezes, a fim de conseguir obter uma medida de tendência central mais precisa em relação ao tempo gasto para o algoritmo para ordenação.

IMPLEMENTAÇÃO

Após a análise do requisitos, constatou-se que a linguagem de programação C++ seria a melhor escolha para que o algoritmo fosse implementado, levando em consideração as facilidades provenientes do C++ em relação à linguagem C, como a forma de declaração de tipos primitivos, alocação de memória, e a própria familiaridade com a linguagem em si. Optou-se pela não utilização de orientação a objetos em pontos onde sua utilização não era necessária, devido a esse programa não possuir necessidade de conter estruturas complexas.

Em busca de manter as boas práticas de programação, o sistema foi modularizado em diferentes arquivos, cada um deles contendo as funções específicas de cada umas das sete implementações do Quicksort, além de um arquivos separado somente para funções genéricas que são comuns a diferentes implementações do algoritmo. Vale ressaltar que nem todas possibilidades de reaproveitamento de código foram utilizadas, sendo essa uma ação intencional, a fim de preservar a didática do objeto de estudo, possibilitando no futuro que o programa compile sem todos os módulos, e somente os módulos de interesse.

As funções de cada implementação do Quicksort foram distribuídas da seguinte forma:

- **QC.hpp**

Funções necessárias para o algoritmo do **Quicksort Clássico**, sendo elas, uma para particionar o vetor em dois subvetores, com o nome de "*particao*" do tipo "void", de forma a aplicar o conceito "dividir para conquistar" e outra para ordenar os subvetores após ocorrer a partição com o nome de "ordena", também do tipo "void". Inicialmente é escolhido um pivô através da seleção do elemento central do vetor, a partir daí a função "particao" é chamada para dividir o vetor principal em dois subvetores, e então a função ordena é chamada, para ordenar o vetor, e dessa

forma acumulamos chamadas recursivas até que o vetor principal esteja completamente ordenado.

- **QM3.hpp**

Funções necessárias para o algoritmo do **Quicksort mediana de três**. Esse caso de implementação se aproxima da implementação do Quicksort clássico, o objetivo principal dela era evitar o pior caso do Quicksort, por isso o pivô é escolhido através da mediana entre o primeiro elemento do vetor, o elemento central e o último elemento.

- **QPE.hpp**

Funções necessárias para o algoritmo do **Quicksort utilizando o primeiro elemento** como pivô. Essa implementação não apresenta grande diferença da implementação do Quicksort clássico, a única alteração notável é a escolha do primeiro elemento como pivô, e não mais o elemento central.

- **QI.hpp**

Funções necessárias para o algoritmo do **Quicksort inserção**. Nesse caso em específico foi escolhido implementar de forma genérica para qualquer porcentagem, neste trabalho utilizaremos, 1%, 5% e 10%. A ideia inicial aproxima-se do Quicksort mediana de três, tendo como diferença que o processo de partição é interrompido quando o subvetor possuir menos que a porcentagem definida de chaves. Outra mudança é que o subvetor é ordenado utilizando uma implementação diferente do algoritmo de ordenação por inserção

- **QNR.hpp**

Funções necessárias para o algoritmo do Quicksort porém sem a utilização de recursividade, nomeado como **Quicksort não recursivo**. nesse caso em específico, como não era a ideia da variação utilizar a recursividade, foi necessário implementar uma pilha, a fim de simular as chamadas recursivas.

INSTRUÇÕES DE COMPILAÇÃO E EXECUÇÃO

- **PARA COMPILAR:**

  Acesse a pasta inicial dos arquivos e digite no terminal ~ make

  Ou, caso não possua o make instalado

  g++ main.cpp helpers.cpp qc.hpp qm3.hpp qpe.hpp qi.hpp qnr.hpp -Wall -o tp2

Nesse programa foi utilizado um arquivo Makefile, a fim de facilitar a compilação do programa, inicialmente rode o comando “make” no terminal, dentro da pasta inicial do programa. O arquivo Makefile vai se encarregar da compilação de todos os arquivos necessários ao programa.

- **PARA EXECUTAR:**

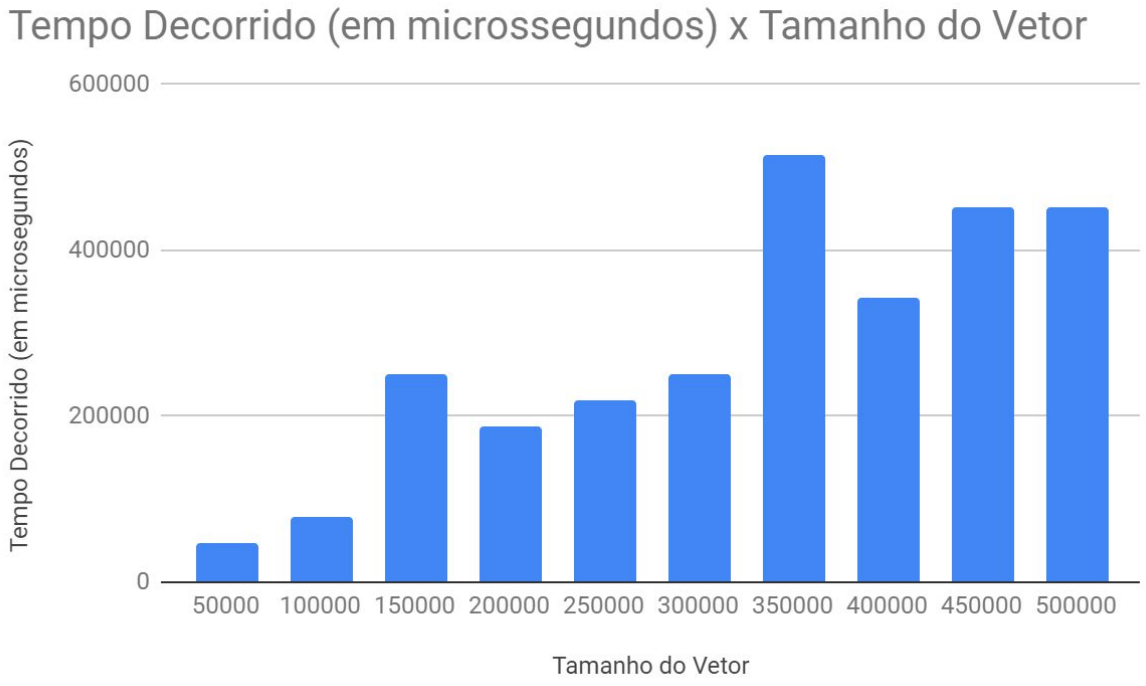
Acesse a pasta /bin e digite no terminal ~ /tp2 <variacao> <tipo> <tamanho> <-p>

O primeiro parâmetro se refere ao nome do executável gerado pelo compilador, o segundo se refere a variação do quicksort desejada, o terceiro ao tipo do vetor em que essa variação do quicksort deve ser aplicada, o terceiro parâmetro se refere ao tamanho do vetor desejado, sendo este entre 50 mil e 500 mil elementos, e o último parâmetro opcional, que tem como finalidade imprimir o vetor utilizado ou não.

ANÁLISE EXPERIMENTAL

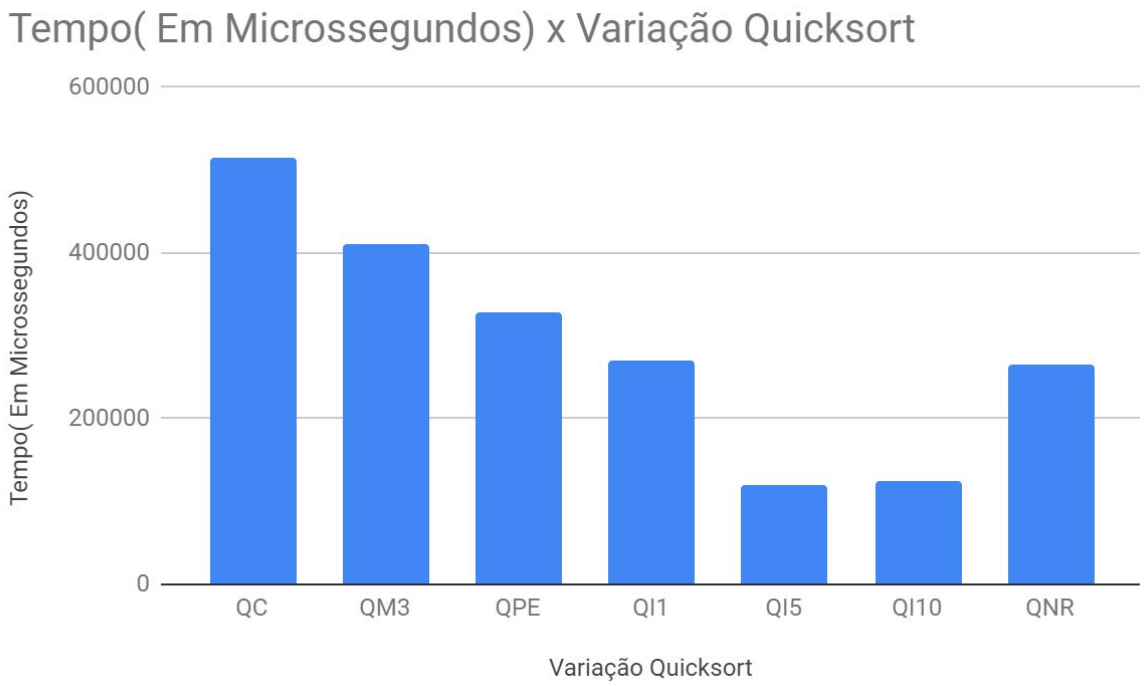
Vamos analisar inicialmente o comportamento de cada variação do quicksort, observando o número de comparações e número de trocas necessárias para ordenar os três tipos de vetores.

Inicialmente, durante a coleta de dados, observou-se que o tempo de execução, em geral, apresenta uma tendência de crescimento de acordo com o crescimento do vetor, sendo o vetor aleatório como o que demanda maior tempo de execução.



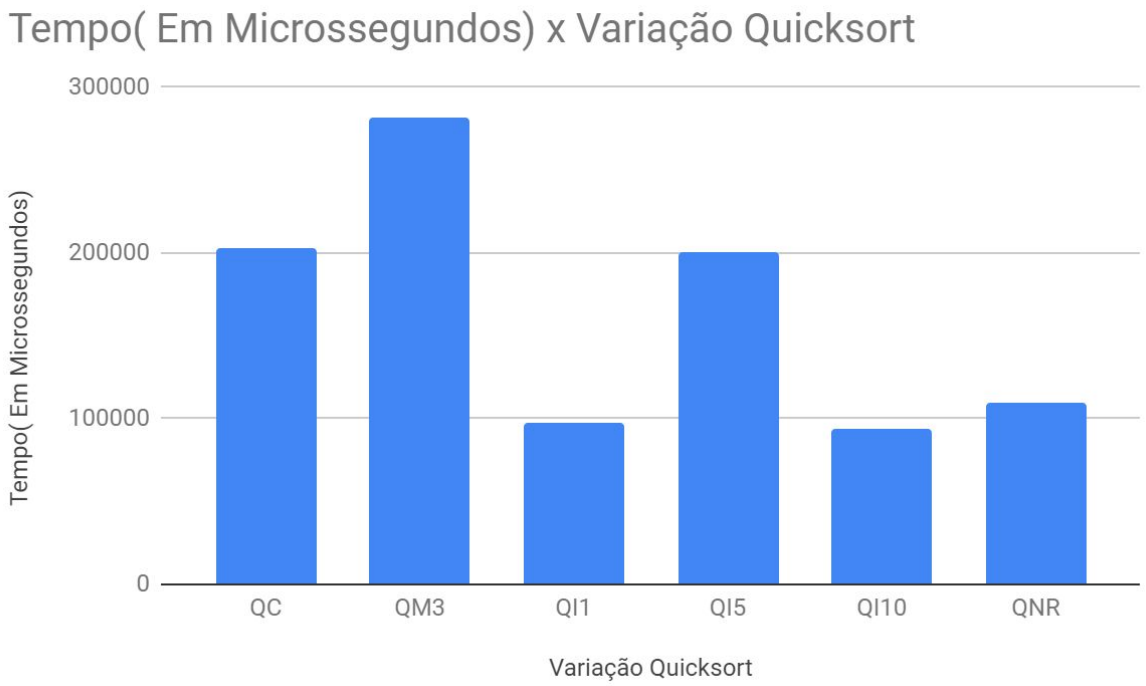
Dessa forma, será utilizado como padrão o vetor de tamanho 350.000 posições, a fim de observar qual variação do quicksort possui um menor tempo de execução em vetores aleatórios, ordenados de forma crescente e por fim de forma decrescente.

VETOR ALEATÓRIO



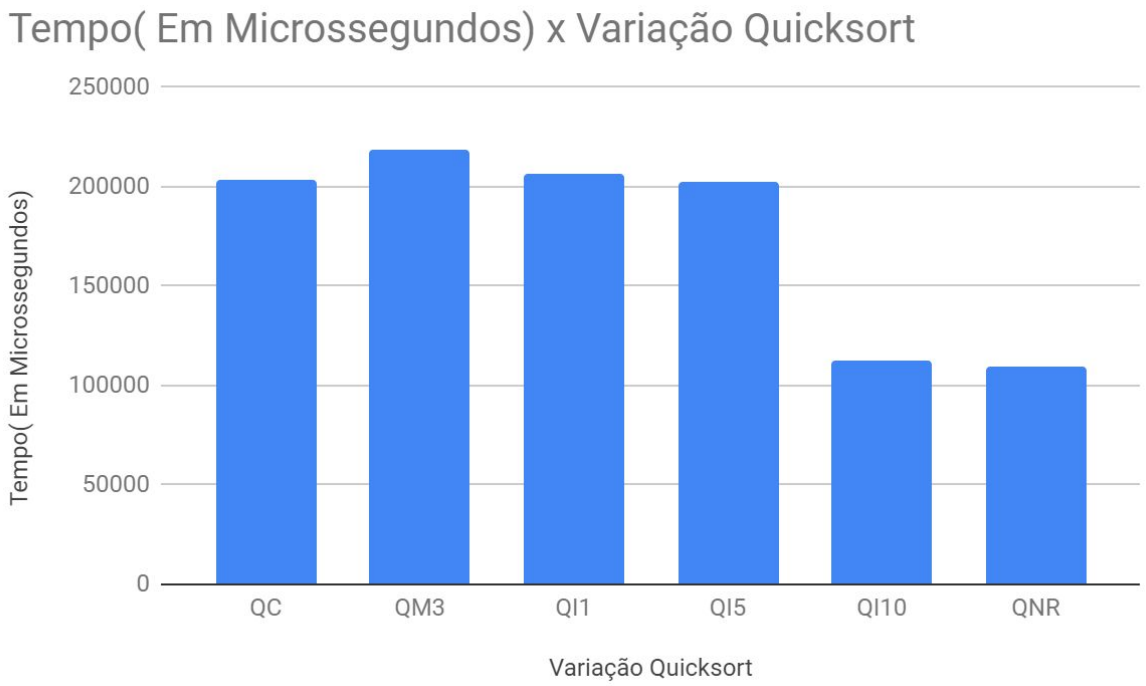
Dessa forma, observa-se que em um vetor aleatório, o Quicksort Inserção 5% e o Quicksort Inserção 10% apresentam os menores tempos gastos para ordenar um vetor aleatório, e é observado que o Quicksort Clássico se mostra a variação menos eficiente quando a proposta é ordenar um vetor aleatório.

VETOR ORDENADO DE FORMA CRESCENTE



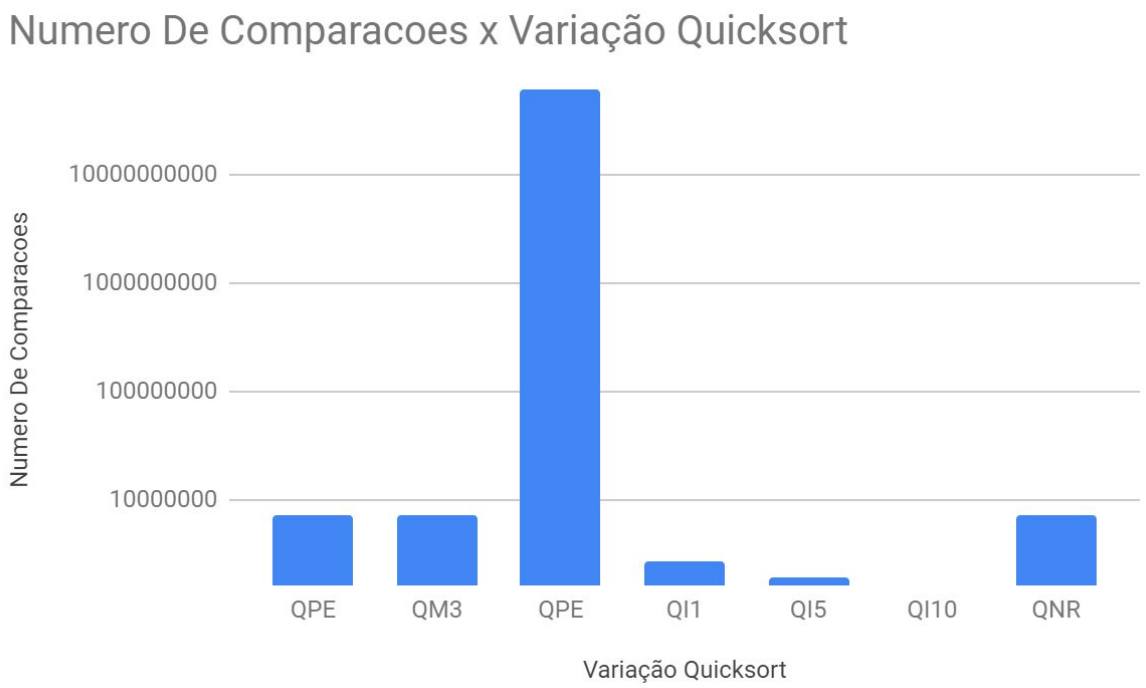
No caso de um vetor ordenado de forma crescente, observa-se que ainda sim o Quicksort Inserção se mostra eficiente, em relação aos demais, e o Quicksort do primeiro elemento apresenta a menor eficiência nesse tipo de caso, sendo que no último caso analisado acima, ele ganhava de outras duas versões do Quicksort

VETOR ORDENADO DE FORMA DECRESCENTE

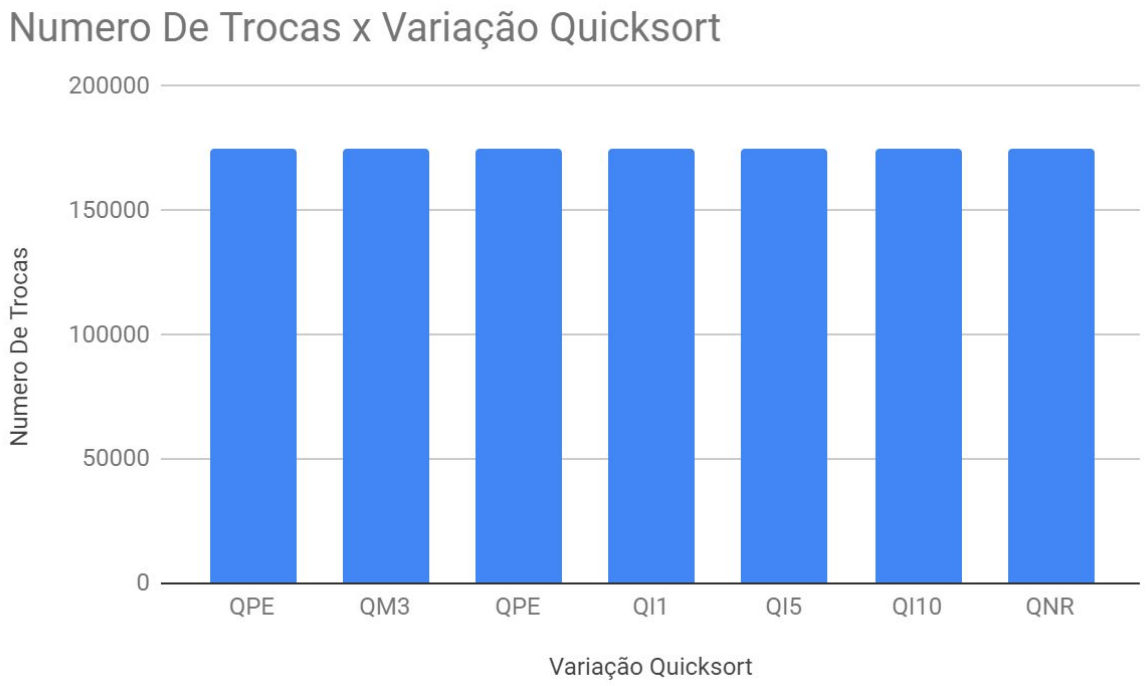


Observa-se que nesse caso, temos que o quicksort não recursivo apresenta o menor tempo para ordenar o vetor ordenado de forma decrescente, enquanto que o Quicksort com pivô definido para o primeiro elemento apresenta o maior tempo de ordenação.

Agora vamos analisar a relação entre o número de comparações e o número de trocas de cada variação do quicksort



Observa-se que o Quicksort primeiro elemento realiza um grande número de comparações para ordenar um vetor, enquanto que o Quicksort com inserção 10% realiza o menor número de comparações, isso é um parâmetro a ser analisado quando pensamos no custo operacional de ordenar um vetor, um fator que comprova o fato do Quicksort Inserção 10% apresentar o melhor desempenho é que ele apresenta o menor tempo gasto para ordenar um vetor aleatório e agora podemos afirmar que isso se deve ao algoritmo realizar um menor número de comparações.



Um outro parâmetro interessante, é que apesar da diferença no número de comparações feitas por cada variação do quicksort, em geral, eles realizam o mesmo número e trocas para ordenar um vetor.

CONCLUSÃO

Devido a versatilidade do Quicksort, das diferentes possibilidades de implementação de cada uma de suas variações, constata-se que as variações do Quicksort podem produzir bons resultados, devido ao fato de que algumas delas corrigem falhas de otimização do Quicksort, como por exemplo, a mediana de três, que evita o pior caso do Quicksort, entre outras variações que apresentam diferentes melhorias.

Nesse caso de estudo, constatamos que o Quicksort com o pivô como primeiro elemento apresenta o pior desempenho em geral, e que o Quicksort com inserção em geral apresenta o melhor desempenho dentre as outras implementações, uma das explicações para isso é que essa variação em específico utiliza o “Quicksort

mediana de três” como base, dessa forma evita o pior caso, dentre outras melhorias propostas por essa variação.

Acredita-se que uma das maiores contribuições deste trabalho foi a percepção de que um mesmo algoritmo pode ser melhorado de diversas formas quando é adaptado para um objetivo em específico, como por exemplo em um caso em que não seja possível implementar a recursividade, existe a possibilidade de implementar uma pilha. Não somente isso, mas estimular o desenvolvedor a pensar e analisar qual maneira é mais viável e produtiva para atingir um objetivo final é extremamente enriquecedor para a formação do profissional.

Por fim, o exercício na prática do conteúdo aprendido em sala trouxe uma nova visão a respeito da utilização dos algoritmos, além de ilustrar a realidade, a atividade proporcionou uma experiência das possibilidades que cada algoritmo traz.

REFERÊNCIAS BIBLIOGRÁFICAS

- CHAIMOWICZ, L; PRATES, Estrutura de Dados, UFMG. Disponivel em: <https://virtual.ufmg.br/20191/course/view.php?id=262> Acesso em: 10 jun 2019
- Geeks for Geeks. Disponível em: <https://www.geeksforgeeks.org> Acesso em 14 jun 2019
- FEOFILOFF, Paulo Quicksort. Disponível em <https://www.ime.usp.br/~pf/algoritmos/aulas/quick.html> Acesso em: 08 de Jun de 2019.