UNIVERSIDADE FEDERAL DE VIÇOSA - CAMPUS FLORESTAL

DISCIPLINA: ALGORITMOS E ESTRUTURA DE DADOS I - CCF221

Professora: Thais Regina de Moura Braga Silva

Trabalho Prático 3

Sistema de Ordenação de Vetores com Algoritmos de Quicksort

INTEGRANTES: 01- Arthur Fernandes Bastos. [EF04679]

02- Ângelo Cupertino Machado. [EF04695]

03- Jéssica Cristina Carvalho. [EF04686]

Sumário

1 Introdução

Algoritmos de ordenação são algoritmos para manipulação de dados que visam a organização dos mesmos, geralmente para facilitar a recuperação deles em uma lista. Existem várias técnicas de ordenação, desde as mais simples até as mais sofisticadas. Não há uma técnica que seja a mais adequada para todas as situações possíveis, é necessário analisar cada caso específico para dizer qual é o algoritmo de ordenação mais adequado, e em alguns deles é mais interessante usar a junção de dois ou mais algoritmos.

O algoritmo quicksort e suas variações utilizam o paradigma de programação “dividir para conquistar”, que consiste em ramificar a entrada múltiplas vezes a fim de dividir o problema em problemas menores. Isso é feito através do uso de um método particionador, onde é escolhido um elemento chamado pivô que será comparado com todos os demais elementos, fazendo com que os maiores que ele fiquem à sua direita e os menores à sua esquerda. A forma de escolher o pivô, a quantidade de partições e a forma específica de ordenar os elementos dentro delas geram subdivisões

Este trabalho prático visa analisar o desempenho de alguns tipos de quicksort na ordenação de vetores. Foi proposto que os alunos usem esses algoritmos para ordenar vetores gerados aleatoriamente de diferentes tamanhos e comparar os seguintes dados: número de comparações, quantidade de movimentações e tempo gasto para ordenar. Os resultados são apresentados por meio de tabelas num arquivo, comparando em cada caso os seguintes algoritmos:

- Quicksort Recursivo: este é o Quicksort recursivo apresentado em sala de aula
- Quicksort Mediana(k): esta variação do Quicksort recursivo escolhe o pivô para partição como sendo a mediana de k elementos do vetor, aleatoriamente escolhidos. Utilize k = 3 e k = 5.
- Quicksort Inserção(m): esta variação modifica o Quicksort Recursivo para utilizar o algoritmo de inserção para ordenar partições (isto é, pedaços do vetor) com tamanho menor ou igual a m. Experimente com m = 10 e m = 100.
- Quicksort Empilha Inteligente(): esta variação otimizada do Quicksort Recursivo processa primeiro o lado menor da partição.
- Quicksort Iterativo: esta variação escolhe o pivô como o elemento do meio (como apresentado em sala de aula), mas não é recursiva. Em outras palavras, esta é uma versão iterativa do Quicksort apresentado em sala de aula.

2 Desenvolvimento do Código

Primeiramente, foram implementados os métodos de Quicksort Recursivo e o de Iteração e uma base para a main, nos possibilitando ver se esses dois métodos estavam ordenando corretamente com um vetor feito manualmente.

Após conferir os vetores e ver que estavam corretos, seguimos para a implementação do quicksort Mediana, utilizando k = 3 primeiramente.Utilizando o primeiro, o último e o elemento no meio do vetor e assim fazendo a mediana entre os três.

Para ter uma ideia melhor de como o processo estava indo, foi criado a parte do código onde seria criado o vetor de entrada com ordem aleatória e também implementado o sistema de medição de tempo utilizado no último trabalho.

Em seguida, foi implementado o Quicksort Inserção (m = 10 e m = 100) e após realizar pesquisas conseguimos entender como funcionaria o método de Empilha Inteligente e conseguimos adequá-lo ao sistema. Completando assim, 6 dos 7 métodos que eram necessários para a comparação dos códigos

Por último e não menos importante, foi feito o método de Quicksort Mediana utilizando a variável k = 5 e adequado ao sistema de medição de tempo, comparações e movimentações.Finalizando assim os métodos pedidos pelo trabalho.

O próximo passo foi finalizar a parte de entrada, saída e leitura de arquivos para comparar os resultados de forma correta.

As variáveis para cálculo de tempo gasto na ordenação em cada algoritmo foram declaradas na função main, bem como as de contagem de movimentações e contagem de comparações. No caso das duas últimas, o endereço das variáveis é passado como parâmetro das funções para serem iteradas dentro delas. Depois de executado cada algoritmo, o valor dessas variáveis é incluído no arquivo de saída, e logo em seguida zerado para ser novamente iterado no próximo algoritmo de ordenação.

Decidimos contabilizar 3 cópias a cada troca realizada (função swap) nos algoritmos. No Quicksort Mediana (de 3 e de 5), optamos por não contabilizar como cópias as atribuições das variáveis dentre as quais seria calculada a mediana para ser utilizada como pivô, já que ali não se tratava da operação principal.

3 Resultados e Discussão

Os resultados de desempenho dos métodos de ordenação podem ser vistos nas figuras a seguir:

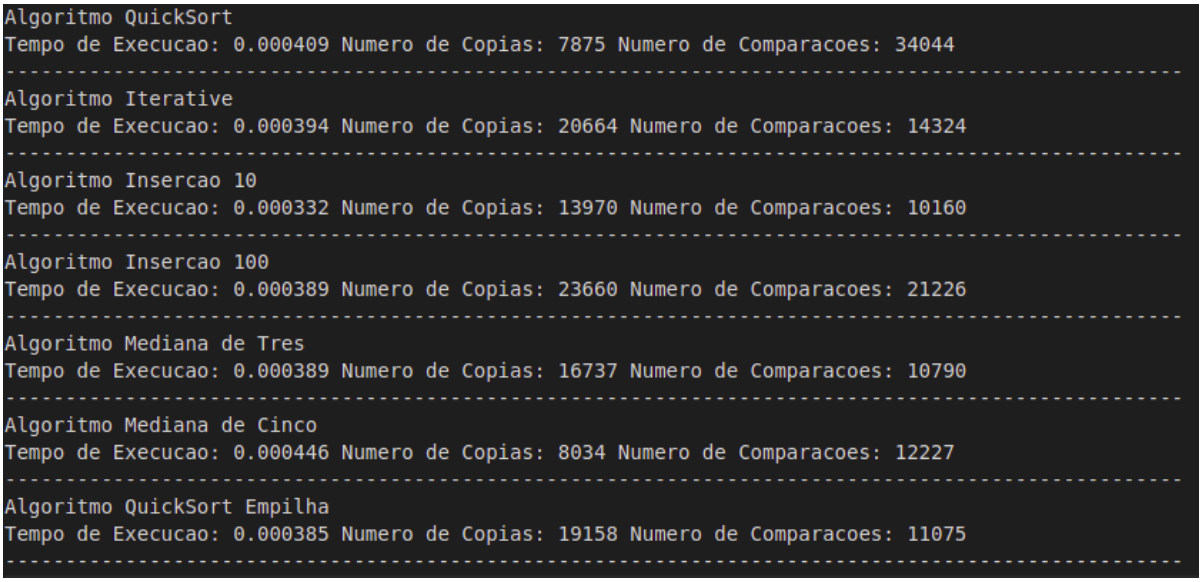


Figura 1: Desempenho na ordenação de vetores de tamanho 1000

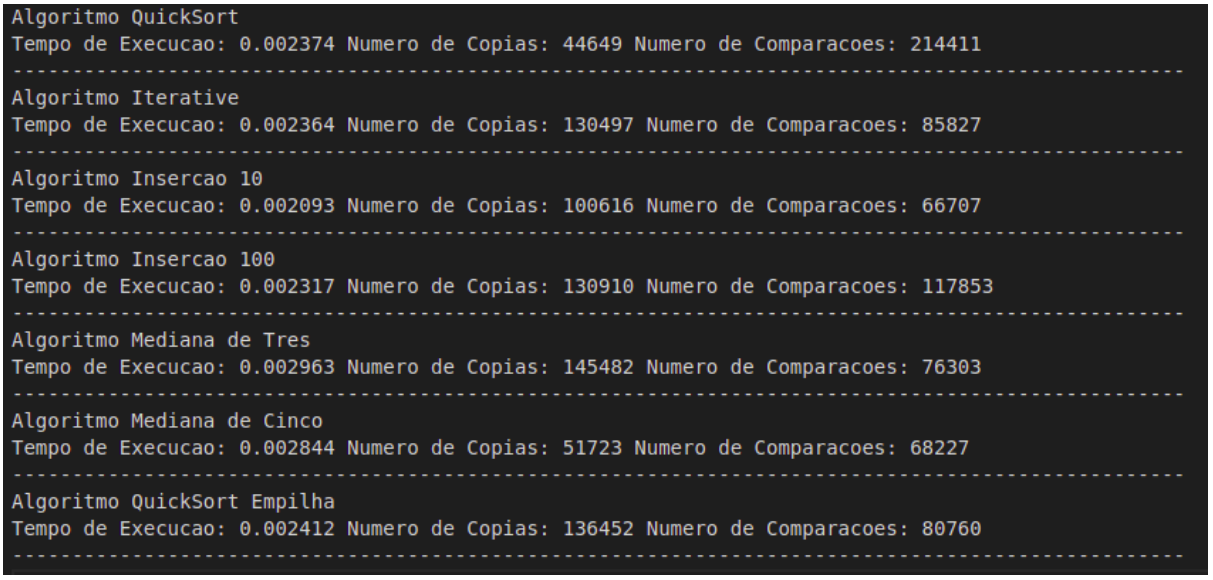


Figura 2: Desempenho na ordenação de vetores de tamanho 5000

```
Algoritmo QuickSort
Tempo de Execucao: 0.004840 Numero de Copias: 91536 Numero de Comparacoes: 482695
-----------------------------------------------------------------------------------------------
Algoritmo Iterative
Tempo de Execucao: 0.004473 Numero de Copias: 363009 Numero de Comparacoes: 213654
-----------------------------------------------------------------------------------------------
Algoritmo Insercao 10
Tempo de Execucao: 0.003091 Numero de Copias: 258355 Numero de Comparacoes: 162178
-----------------------------------------------------------------------------------------------
Algoritmo Insercao 100
Tempo de Execucao: 0.002592 Numero de Copias: 282659 Numero de Comparacoes: 237353
-----------------------------------------------------------------------------------------------
Algoritmo Mediana de Tres
Tempo de Execucao: 0.002746 Numero de Copias: 359835 Numero de Comparacoes: 189782
-----------------------------------------------------------------------------------------------
Algoritmo Mediana de Cinco
Tempo de Execucao: 0.002054 Numero de Copias: 116184 Numero de Comparacoes: 133631
-----------------------------------------------------------------------------------------------
Algoritmo QuickSort Empilha
Tempo de Execucao: 0.002365 Numero de Copias: 363147 Numero de Comparacoes: 179032
-----------------------------------------------------------------------------------------------
```

Figura 3: Desempenho na ordenação de vetores de tamanho 10000

```
Algoritmo QuickSort
Tempo de Execucao: 0.017690 Numero de Copias: 464700 Numero de Comparacoes: 4456289
-----------------------------------------------------------------------------------------------
Algoritmo Iterative
Tempo de Execucao: 0.018032 Numero de Copias: 4986501 Numero de Comparacoes: 2025335
-----------------------------------------------------------------------------------------------
Algoritmo Insercao 10
Tempo de Execucao: 0.013501 Numero de Copias: 4721886 Numero de Comparacoes: 1921911
-----------------------------------------------------------------------------------------------
Algoritmo Insercao 100
Tempo de Execucao: 0.005122 Numero de Copias: 1266015 Numero de Comparacoes: 878107
-----------------------------------------------------------------------------------------------
Algoritmo Mediana de Tres
Tempo de Execucao: 0.011518 Numero de Copias: 4848453 Numero de Comparacoes: 1937833
-----------------------------------------------------------------------------------------------
Algoritmo Mediana de Cinco
Tempo de Execucao: 0.005558 Numero de Copias: 736890 Numero de Comparacoes: 735983
-----------------------------------------------------------------------------------------------
Algoritmo QuickSort Empilha
Tempo de Execucao: 0.010862 Numero de Copias: 4810137 Numero de Comparacoes: 1914762
-----------------------------------------------------------------------------------------------
```

Figura 4: Desempenho na ordenação de vetores de tamanho 50000

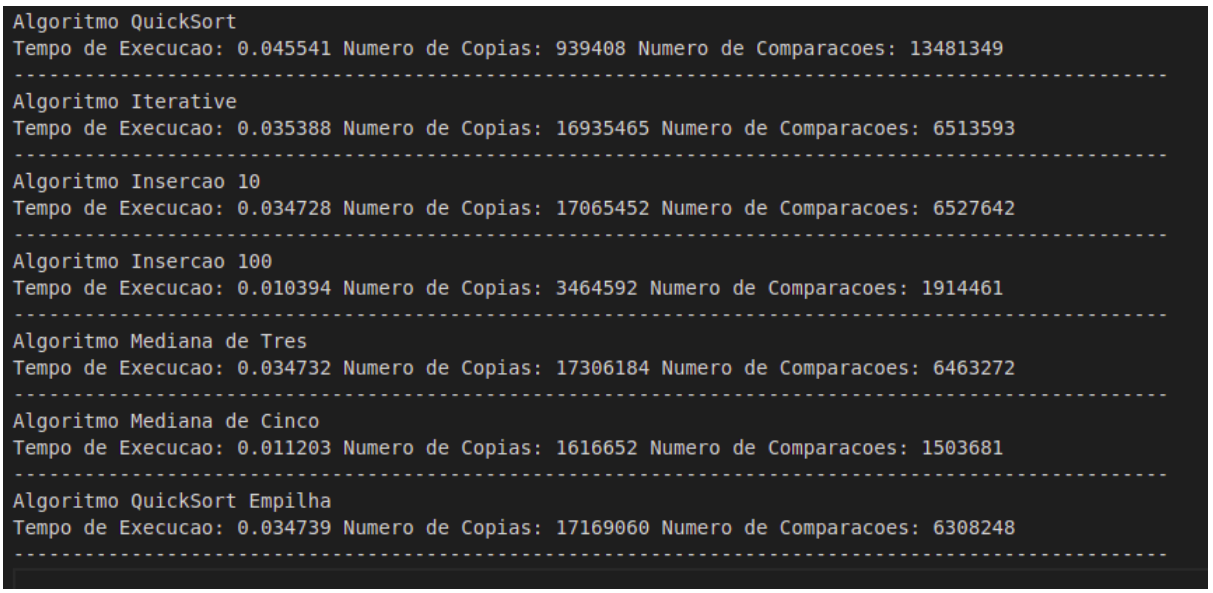


Figura 5: Desempenho na ordenação de vetores de tamanho 100000

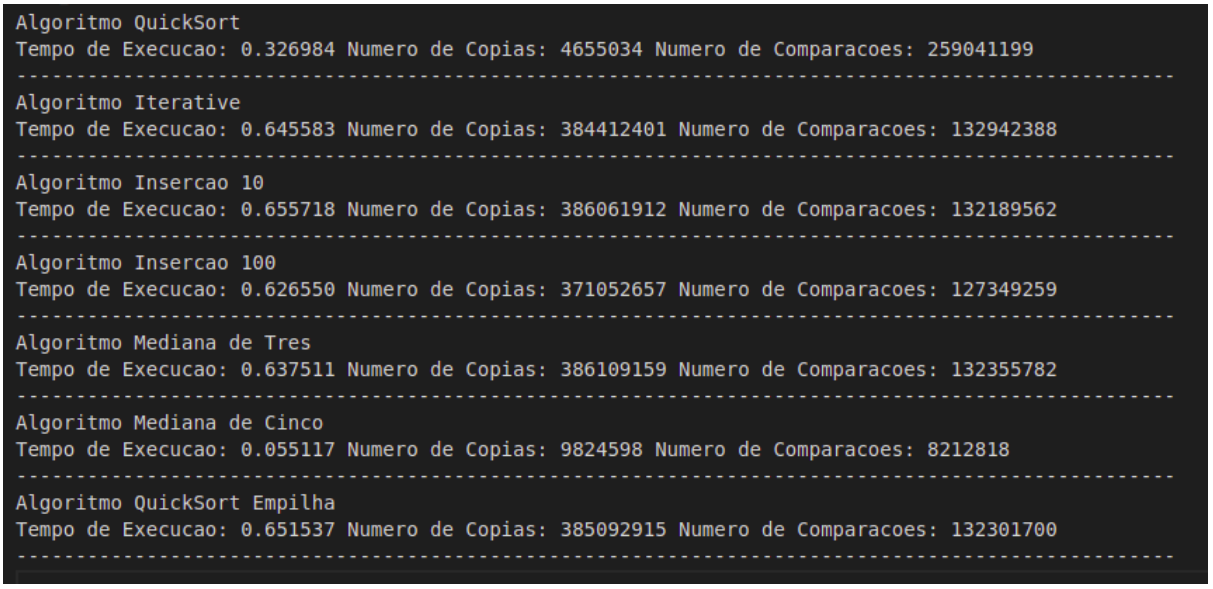


Figura 6: Desempenho na ordenação de vetores de tamanho 500000

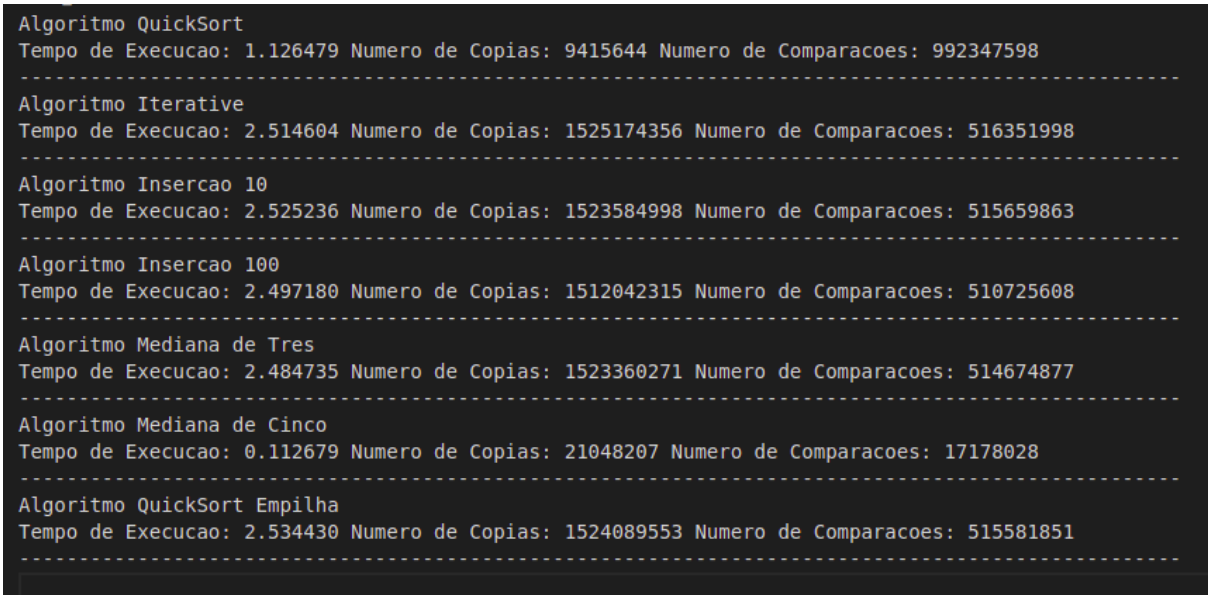


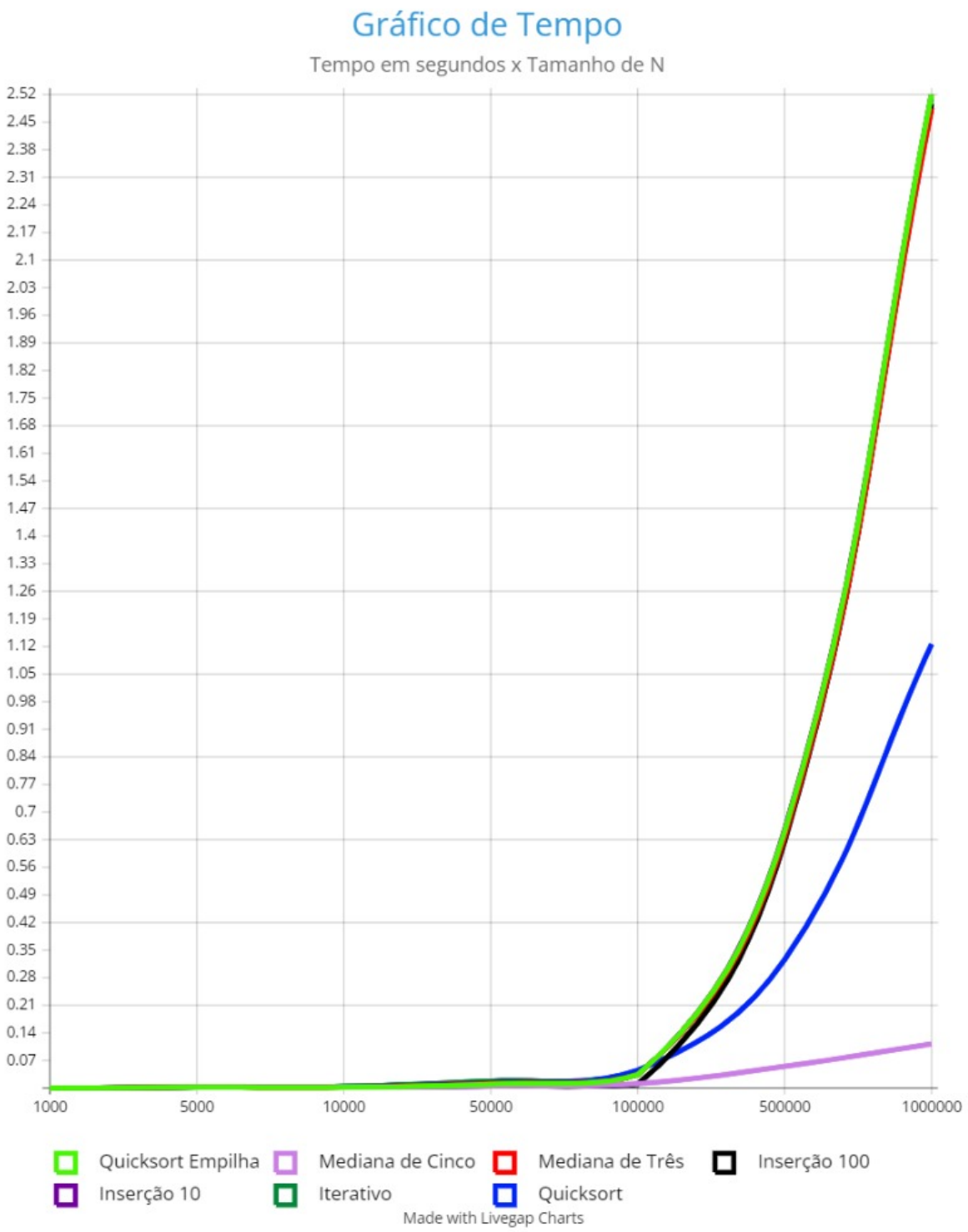
Figura 7: Desempenho na ordenação de vetores de tamanho 1000000

Foi possível observar que, para as menores entradas, o quicksort Mediana de Cinco teve um tempo de execução ligeiramente maior que os demais. Isso porque ele cria um novo vetor de cinco elementos e precisa ordená-lo. No entanto, para entradas com tamanhos maiores que 10000 ele passou a ser o que ordena mais rapidamente, chegando a ser cerca de 20 vezes mais rápido que os demais para o maior tamanho de vetor. Isso provavelmente se deve ao fato de a seleção do pivô desse método ser muito eficiente, embora inicialmente custosa. A quantidade de movimentações nesse método também foi bem menor que as demais. Acreditamos que o fato de termos escolhido não contabilizar as atribuições no momento de escolha do pivô influiu muito pouco nesse resultado.

Quando comparamos o desempenho dos algoritmos de QuickSort Mediana(k) com k=3 e k=5, percebemos que até um certo número de entradas é mais vantajoso utilizar o de k=3, mas a partir desse valor, o desempenho (em questão de velocidade de processamento) do com k=5 passa a superar expressivamente o outro. No caso dos métodos QuickSort Inserção(m) com m = 10 e m = 100, em um dado momento parece que esse mesmo comportamento será observado. No entanto, nas maiores entradas eles apresentaram velocidades de processamento similares.

No caso do QuickSort que utiliza recursividade e o iterativo, nota-se que o primeiro também passa a ser mais veloz que o segundo a partir de um certo tamanho de entradas. No geral, algoritmos que se utilizam de recursividade são evitados, mas nesse caso, a partir de uma certa quantidade de dados, ele parece ser mais adequado.

Nos gráficos a seguir essas informações podem ser mais facilmente visualizadas:



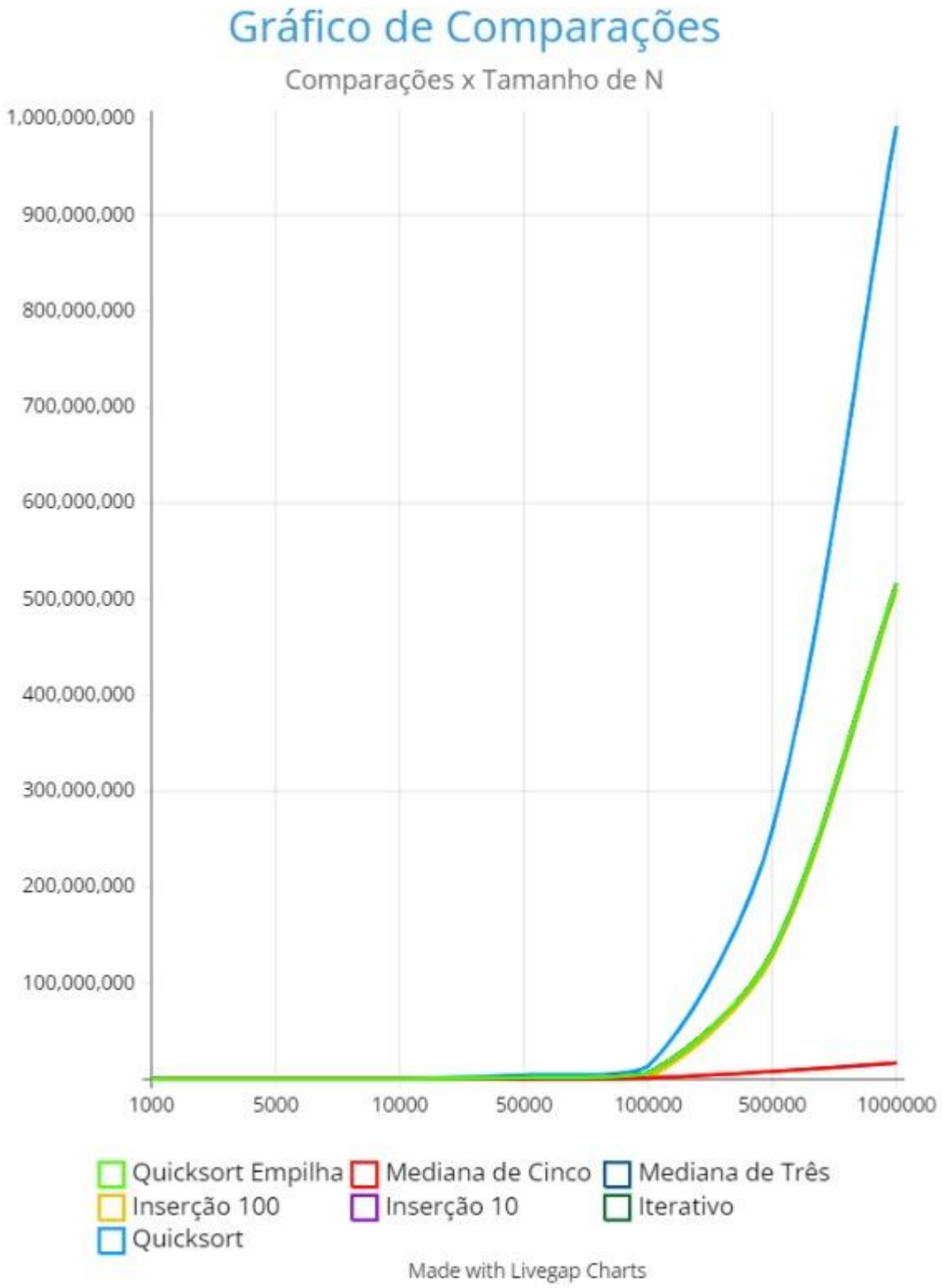
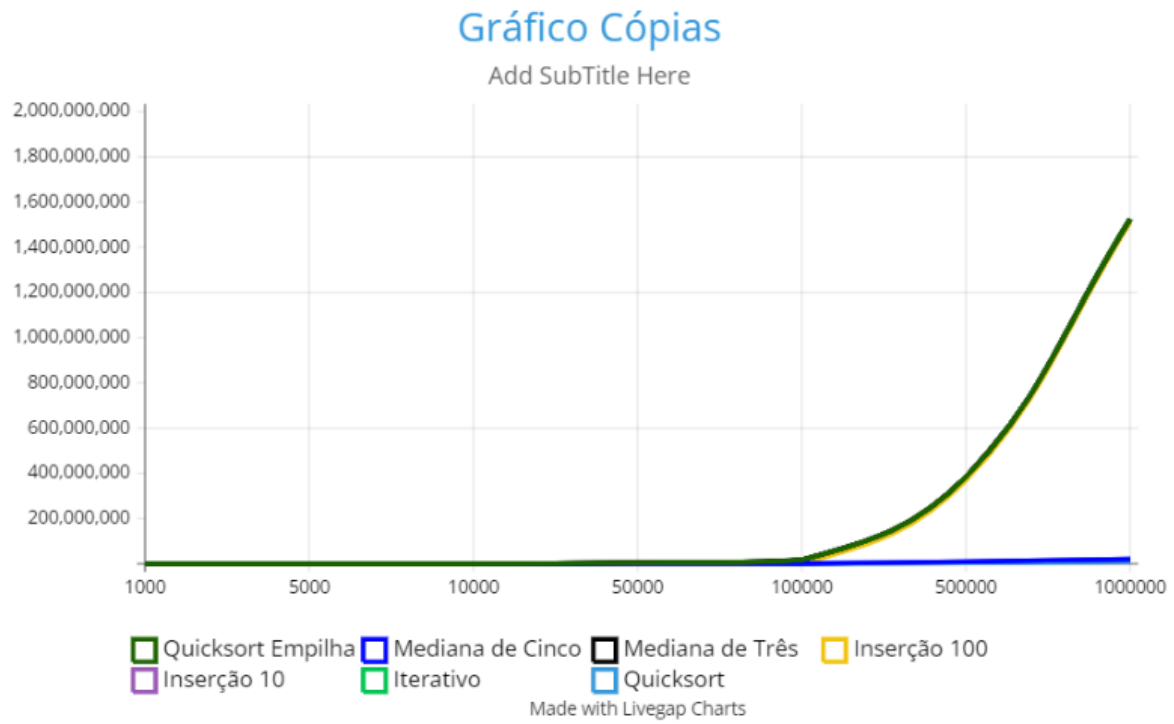


Gráfico de Comparações
Comparações x Tamanho de N
1,000,000,000
900,000,000
800,000,000
700,000,000
600,000,000
500,000,000
400,000,000
300,000,000
200,000,000
100,000,000
1000
5000
10000
50000
100000
500000
1000000
Quicksort Empilha
Mediana de Cinco
Mediana de Três
Inserção 100
Inserção 10
Iterativo
Quicksort
Made with Livegap Charts



4 Conclusão

Neste trabalho foi possível ver na prática que, para cada situação, existe um algoritmo que mais se adequa, e que apresentará melhor desempenho. Embora o método QuickSort Mediana k = 5 tenha tido o melhor desempenho para os maiores valores de entrada, nenhum dos algoritmos foi o mais veloz em todas as situações dadas. Outro fato observado foi a eficiência que os algoritmos de ordenação disponíveis podem apresentar, e o quanto eles podem facilitar a solução de problemas em computação. Os métodos utilizados puderam ordenar vetores de tamanho de até um milhão em questão de menos de três segundos, o que é algo realmente interessante.