Documentação Trabalho Prático da disciplina de Estruturas de Dados

Ricardo Dias Avelar

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brazil

ricardodiasmode@hotmail.com

1. Introdução

Esta documentação tem como objetivo facilitar a compreensão do programa que lida com o problema de ordenar vetores que contém a distância dos robôs até as bases exoplanetárias.

Para resolver o problema foi utilizado conhecimentos adquiridos em aula para formular a lógica dos algoritmos de ordenação e foi utilizado o tipo string apenas para criar vetores de caracteres. Note que não foi utilizado NENHUM método pertencente a std::string.

A seção 2 trata da implementação do programa, onde é explicado detalhadamente as classes e métodos utilizados no programa.

2. Implementação

O código tem uma implementação objetiva, bem orientada e comentada para fácil compreensão. No “Main.cpp” temos como principal variável “SortedArray”, que é um vetor de string de tamanho argv[2]. Além disso, temos uma variável do tipo ifstream chamada InfoFile, utilizada para ler o arquivo. Todas as funções utilizadas no programa são pertencentes à classe “Ordenacoes”, sendo todas “static”, o que faz com que o objeto não precise ser instanciado.

O “Main.cpp” é um arquivo extremamente simples, onde no início abre o arquivo, em seguida se escolhe qual método de ordenação será utilizado e printa no console o vetor ordenado. No final o arquivo é fechado e a memória que aloca o vetor SortedArray é liberada.

class Ordenacoes

Nossa classe contém todos os métodos responsáveis pela ordenação do vetor. Esses métodos compartilham de alguns detalhes explicados abaixo que tornou a implementação bem fácil, diferindo apenas na lógica de ordenação do vetor.

No início de todo método de ordenação temos um “for” de n até NumbersToTest (igual a argv[2]), que nada mais faz que ler o arquivo, passar a linha a ser ordenada para um vetor de string chamado “lines” e passa seu número para um vetor chamado “numbers”. Em todo método pode-se perceber que utiliza-se “numbers” como referência para ordenação, e ao trocar os elementos “i” e “j” de “numbers”, troca-se a mesma posição “i” e “j” de “lines”.

Note que todo método ordena o vetor em ordem crescente, o que me fez utilizar a função “ReverterVetor” no final de todo método, o que não muda a complexidade assintótica de nenhum método, haja vista que a complexidade de “ReverterVetor” é O(n), e nenhum método tem complexidade menor que O(n).

- **static void Ordenacoes::ReverterVetor(string* lines, int tam)**

  string* lines: é o vetor a ser revertido.

  int tam: é o tamanho do vetor.

O método nada mais faz do que transformar um vetor em ordem crescente em o mesmo vetor em ordem decrescente, ou vice-versa.

- **static void Ordenacoes::InsertionSort(ifstream* FileToOrder, int NumbersToTest, string* lines)**

ifstream* FileToOrder: é o arquivo com todas as strings que serão lidas e ordenadas.

int NumbersToTest: é o número de linhas que serão lidas do arquivo.

string* lines: é a variável que segura as linhas que serão ordenadas.

- **static void Ordenacoes::MergeSort(ifstream* FileToOrder, int NumbersToTest, string* lines)**

Todas as variáveis seguem a mesma lógica de InsertionSort. Aqui chama-se o método MergeSort_Implementation, que é de fato a implementação do algoritmo.

- **static void Ordenacoes::Merge(string* lines, string* StringAux, int* saida, int* auxiliar, int inicio, int meio, int fim)**

Método utilizado dentro do MergeSort. Dispensa explicações pois foi explicado em sala de aula.

- **static void Ordenacoes::MergeSort_Implementation(string* lines, string* StringAux, int* saida, int* auxiliar, int inicio, int fim)**

De fato a implementação do MergeSort. Dispensa explicações pois foi explicado em sala de aula.

- **static void Ordenacoes::QuickSort(ifstream* FileToOrder, int NumbersToTest, string* lines)**

Todas as variáveis seguem a mesma lógica de InsertionSort. Aqui chama-se o método QuickSort_Implementation, que é de fato a implementação do algoritmo.

- **static void Ordenacoes::QuickSort_Implementation(string* lines, int values[], int began, int end)**

De fato a implementação do QuickSort. Dispensa explicações pois foi explicado em sala de aula.

- **static void Ordenacoes::QuickSortModificado(ifstream* FileToOrder, int NumbersToTest, string* lines)**

Todas as variáveis seguem a mesma lógica de InsertionSort. Aqui chama-se o método QuickSortModificado_Implementation, que é de fato a implementação do algoritmo.

- **static void Ordenacoes::QuickSortModificado_Implementation(string* lines, int values[], int began, int end)**

Escolhi fazer a melhoria explicada em sala de aula, que é pegar a mediana de 3 elementos no vetor. Fiz essa escolha pois me pareceu bem razoável, de fácil implementação, e gostaria de testá-la. No início do algoritmo há algumas condições para escolher o pivô, que faz com que ele seja o elemento n/2, (n/2)-1 ou (n/2)+1, dependendo de qual deles é a mediana.

- **static void Ordenacoes::GnomeSort(ifstream* FileToOrder, int NumbersToTest, string* lines);**

O método da minha escolha foi GnomeSort. Resumidamente, o algoritmo percorre o vetor em um loop de “i” até “n”, e quando acha um elemento[i] < elemento[i-1], percorre o vetor retornando com um loop de “j” até “0”, ou até encontrar um elemento[j] > elemento[i], e então coloca o elemento[i] na posição j-1. Escolhi esse método pois achei deveras interessante sua aplicação, além de ser de fácil implementação.

O programa foi testado no SO Windows 10, feito em C++ e compilado pelo G++. A máquina que testou possui um processador AMD Ryzen 5 3600x e 16GB de memória RAM.

3. Instruções de compilação e execução

Para facilitar a compilação e execução criei um arquivo.bat encontrado na pasta ..\bin\compile.bat, que quando escrito “compile” no terminal, enquanto neste diretório, o programa é compilado e sua saída é escrita automaticamente. Segue as instruções abaixo para executar o programa:

• Acesse o diretório ..\bin

• Edite a segunda linha do arquivo “compile.bat” para ser o diretório raiz do programa. Edite a quinta linha com os argumentos desejados para a execução do programa. Por default, o primeiro comando deve ser <nome do arquivo> e o

segundo <numeros a serem testados>.

A sua segunda linha, portanto, deve ser algo como:
C:\Users\Voce\Desktop\TP2\bin

A sua quinta linha, portanto, deve ser algo como:

run.out dados_crescente.txt 200000

• Coloque o arquivo .txt dos dados nessa pasta “bin”.
• Utilizando um terminal, execute o arquivo “compile.bat”.
• Com esse comando, o programa deve realizar a ordenação e irá printar no terminal o vetor ordenado em ordem decrescente. Também será printado no terminal o tempo de execução do algoritmo. Note que no Main.cpp há todos os métodos comentados, portanto é necessário retirar o comentário do método em que deseja-se realizar a ordenação.

4. Análise de complexidade e Comparação

4.1. Análise:

main - complexidade de tempo: Essa função tem um “for” que é um laço de 0 a n, onde n é argv[2]. Dessa forma, a complexidade assintótica de tempo dessa função é Θ(n).

main - complexidade de espaço: Essa função tem um vetor SortedArray de dimensão argv[2]. Portanto sua complexidade assintótica de espaço é Θ(n).

ReverterVetor - complexidade de tempo: Essa função realiza dois laços de 0 a **tam**, onde **tam** é o tamanho do vetor passado. Com isso, sua complexidade é Θ(n).

ReverterVetor - complexidade de espaço: Essa função declara uma string que assume tamanho igual a **tam**. Portanto sua complexidade assintótica de espaço é Θ(n).

*OBS: Todos os métodos abaixo têm complexidade de espaço no mínimo O(n) ao declarar o vetor “numbers” anteriormente citado. Tal fato foi desconsiderado para análise independente da implementação.

InsertionSort:

- **Complexidade de tempo:** O(n^2)
- **Complexidade de espaço:** O(1)
- **Vantagens:** Bom quando o arquivo está “quase” ordenado.
- **Desvantagens:** Alto custo de movimentações.
- **Estável?** Sim.

MergeSort:

- **Complexidade de tempo:** O(nlogn)

- **Complexidade de espaço:** O(1)
- **Vantagens:** Custo de pior caso é menor.
- **Desvantagens:** Requer espaço extra proporcional a n.
- **Estável?** Sim.

QuickSort/QuickSortModificado:

- **Complexidade de tempo:** O(n) para o melhor caso. O(nlogn) para o pior caso.
- **Complexidade de espaço:** O(n).
- **Vantagens:** Bom quando o arquivo está “quase” ordenado.
- **Desvantagens:** Implementação complexa.
- **Estável?** Não.

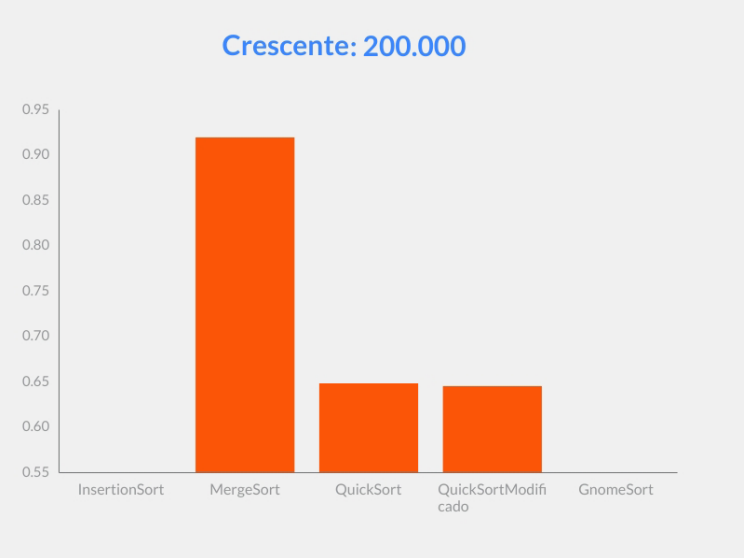
GnomeSort:

- **Complexidade de tempo:** O(n) para o melhor caso. O(n^2) para pior e médio caso.
- **Complexidade de espaço:** O(1).
- **Vantagens:** É estável.
- **Desvantagens:** Extremamente lento na maioria dos casos.
- **Estável?** Sim.
- **Funfact:** Originalmente nomeado como stupidsort.

4.2. Comparação:

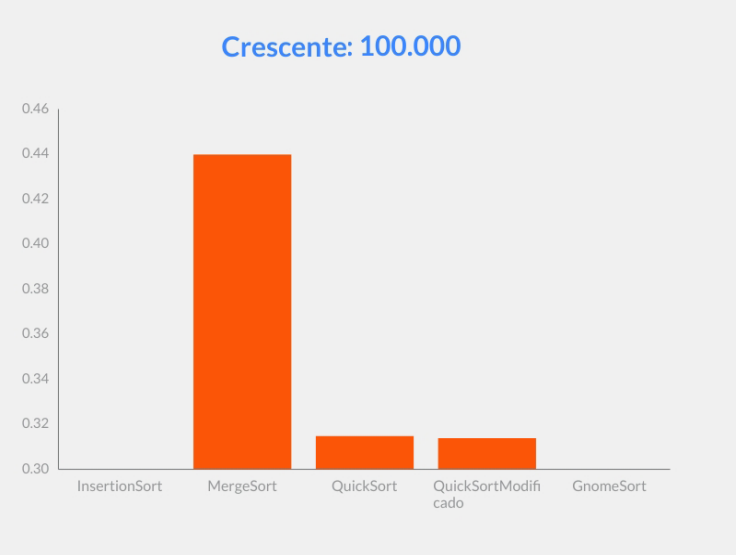
No título de cada gráfico temos a ordenação dos dados e o número de linhas consideradas. Note que alguns gráficos não há InsertionSort nem GnomeSort considerados pois seu número era tão grande que iria tornar os outros insignificantes. No entanto, quando isso aconteceu, foram colocados os seus valores abaixo do gráfico.

No eixo das ordenadas temos o tempo de execução em milissegundos e no eixo das abscissas temos o nome do método de ordenação.



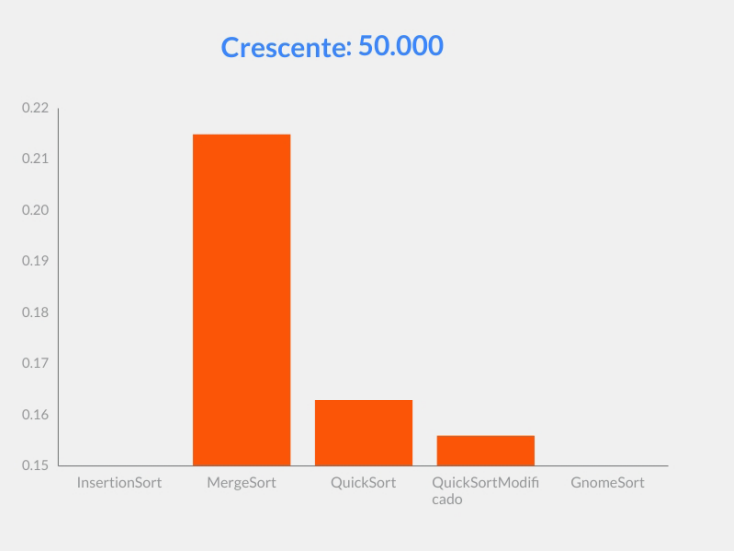
Tempo InsertionSort: 86.846

Tempo GnomeSort: 258.196



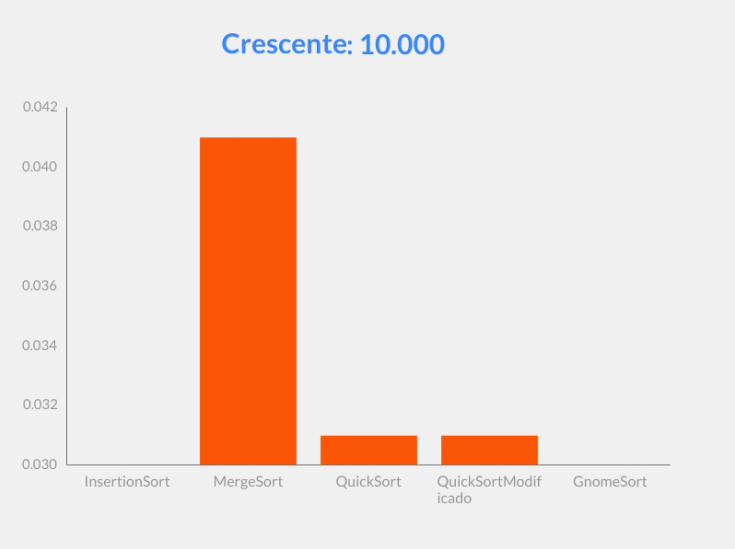
Tempo InsertionSort: 21.576

Tempo GnomeSort: 59.887



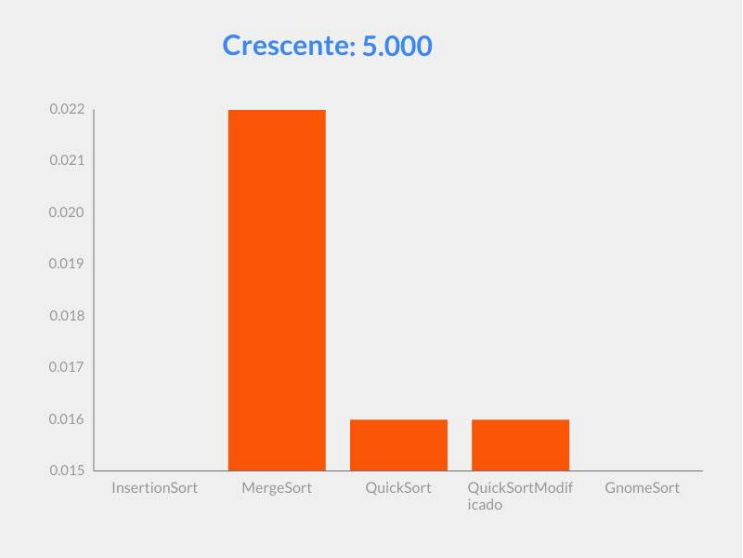
Tempo InsertionSort: 5.333

Tempo GnomeSort: 14.967



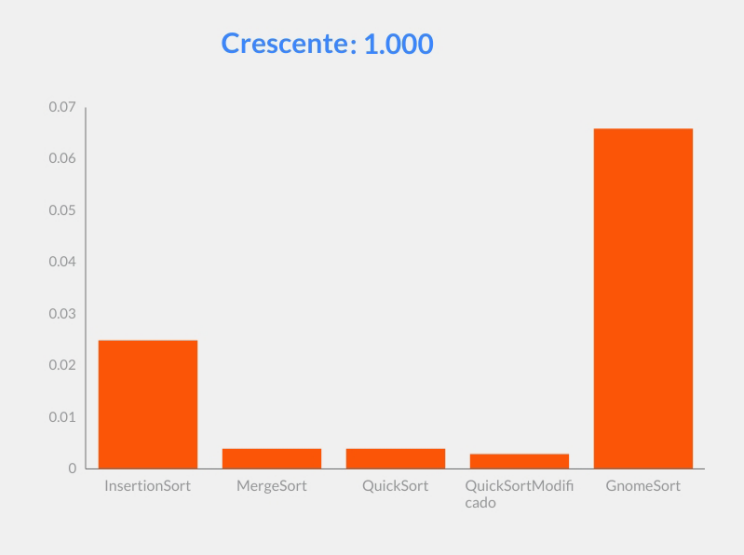
Tempo InsertionSort: 0.229

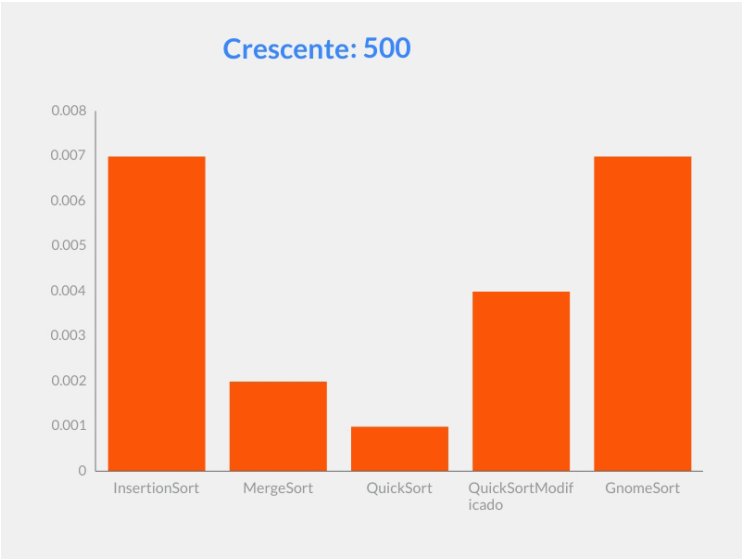
Tempo GnomeSort: 0.596

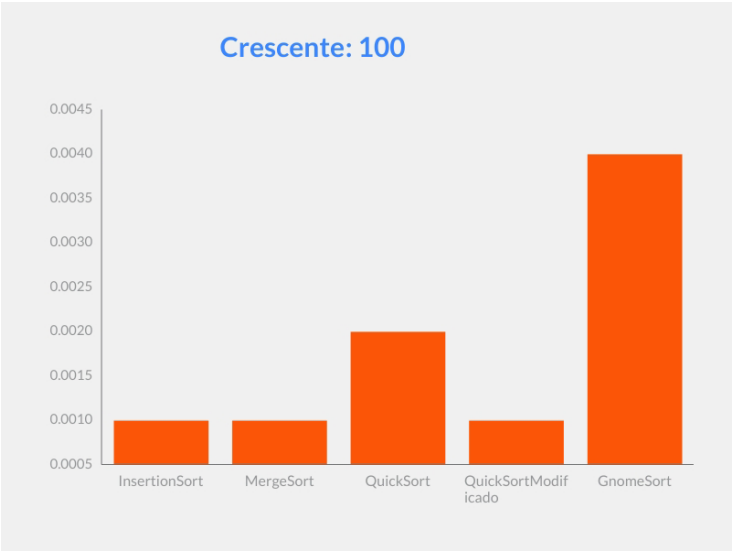


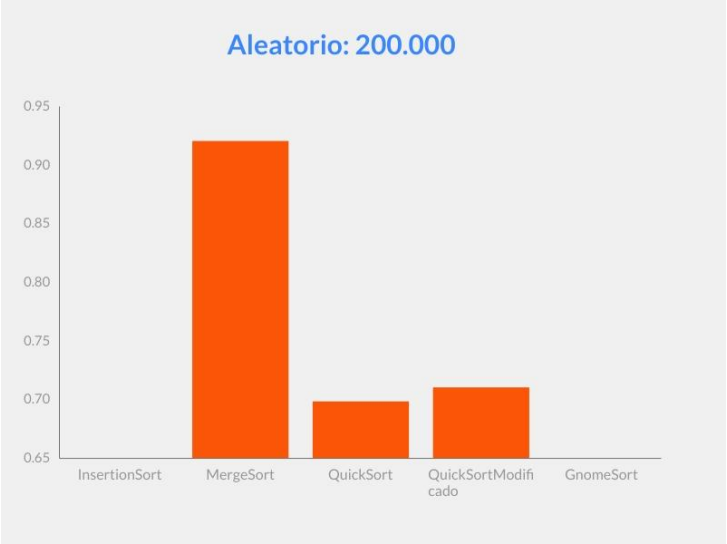
Tempo InsertionSort: 0.524

Tempo GnomeSort: 1.462



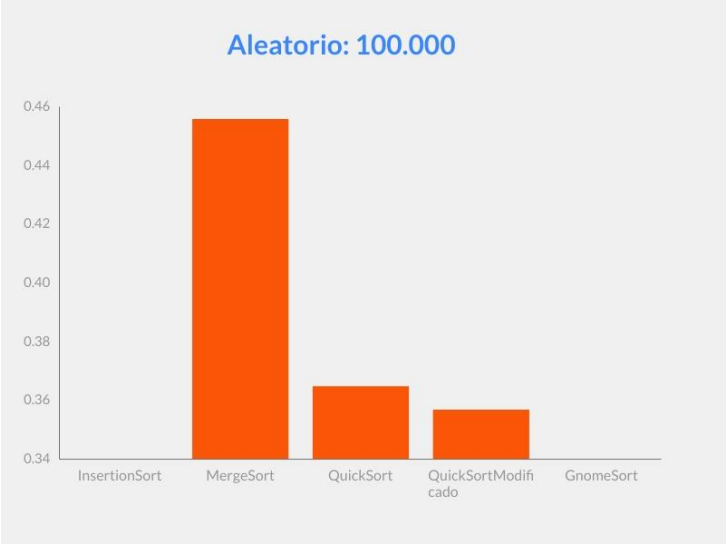

Crescente: 500
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
InsertionSort
MergeSort
QuickSort
QuickSortModificado
GnomeSort


Crescente: 100
0.0045
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
InsertionSort
MergeSort
QuickSort
QuickSortModificado
GnomeSort



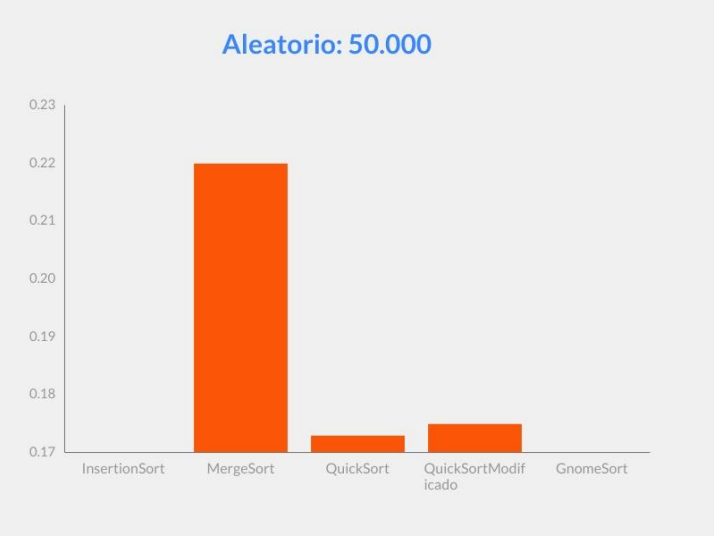
Tempo InsertionSort: 452.517

Tempo GnomeSort: 1359.42



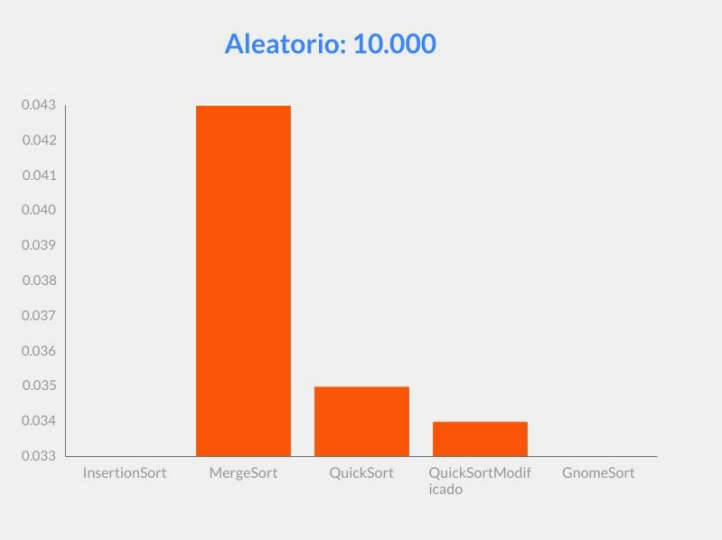
Tempo InsertionSort: 119.35

Tempo GnomeSort: 345.975



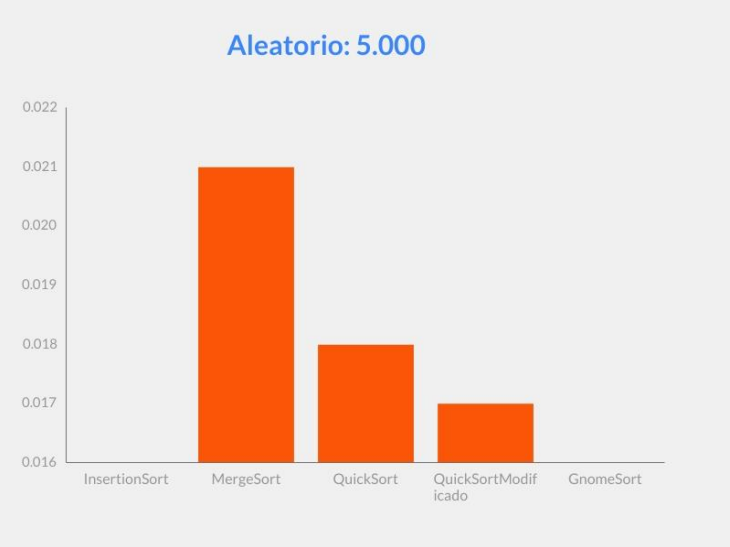
Tempo InsertionSort: 28.732

Tempo GnomeSort: 83.479



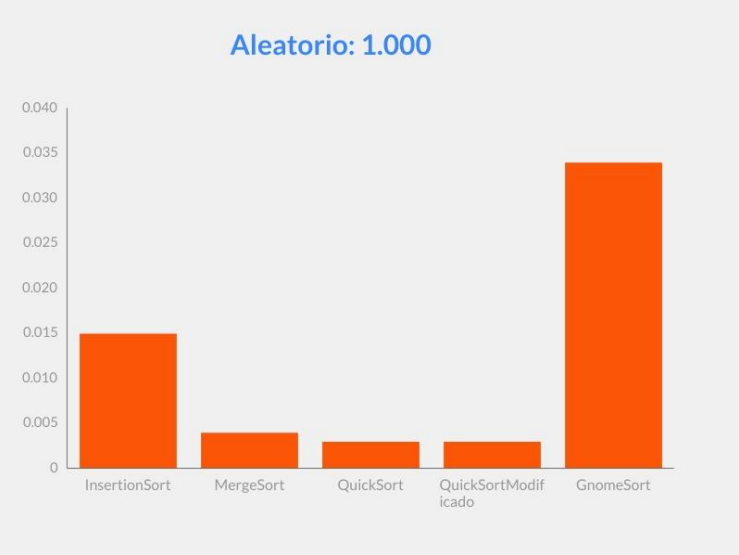
Tempo InsertionSort: 1.178

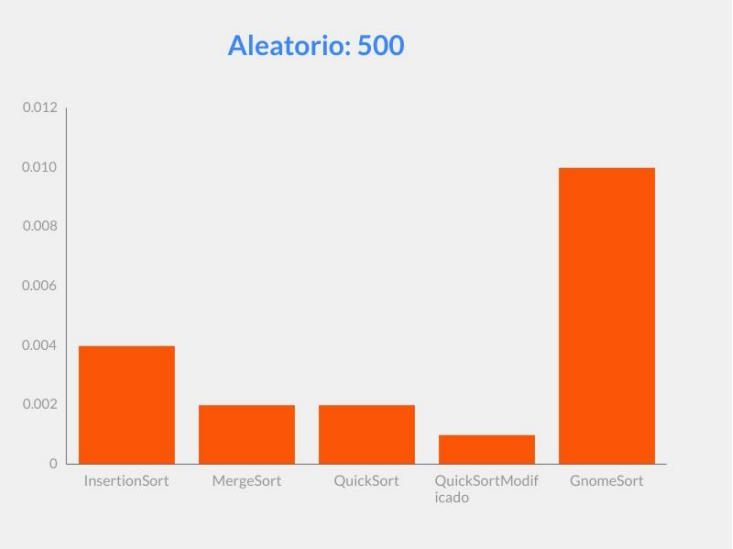
Tempo GnomeSort: 3.237

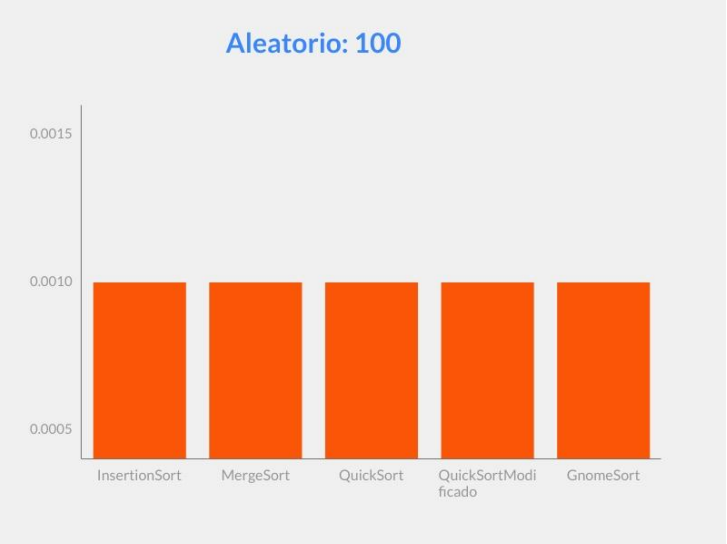


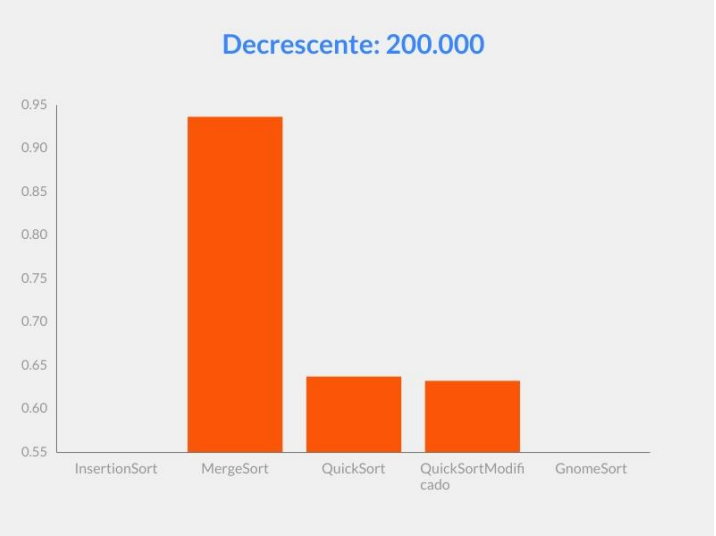
Tempo InsertionSort: 0.308

Tempo GnomeSort: 0.823



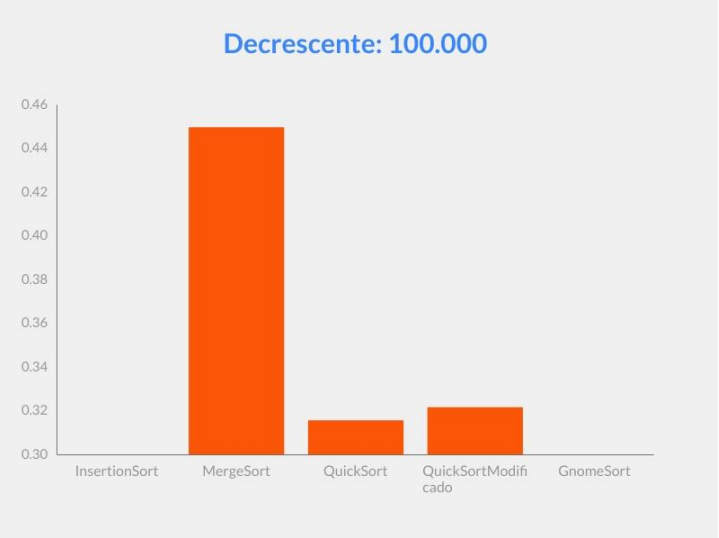

Aleatorio: 500
0.012
0.010
0.008
0.006
0.004
0.002
0
InsertionSort
MergeSort
QuickSort
QuickSortModificado
GnomeSort


Aleatorio: 100
0.0015
0.0010
0.0005
InsertionSort
MergeSort
QuickSort
QuickSortModificado
GnomeSort



Tempo InsertionSort: 985.385

Tempo GnomeSort: 2692.84

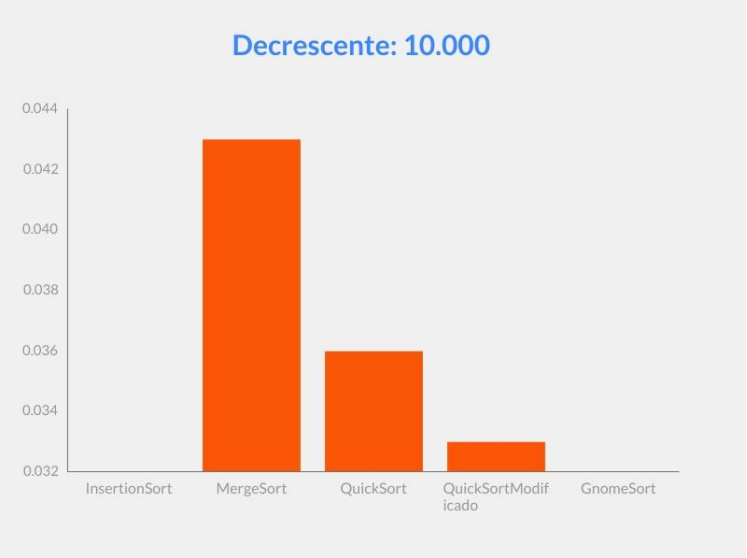


Tempo InsertionSort: 220.044

Tempo GnomeSort: 648.56

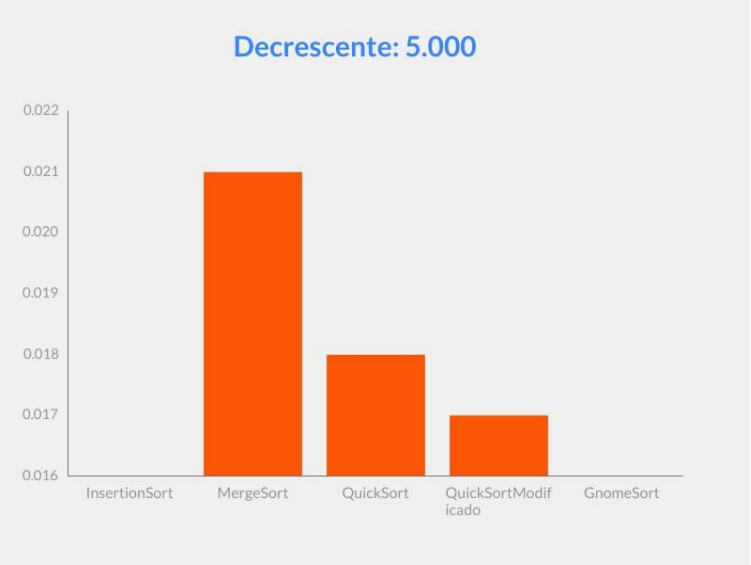
Tempo InsertionSort: 52.98

Tempo GnomeSort: 154.599



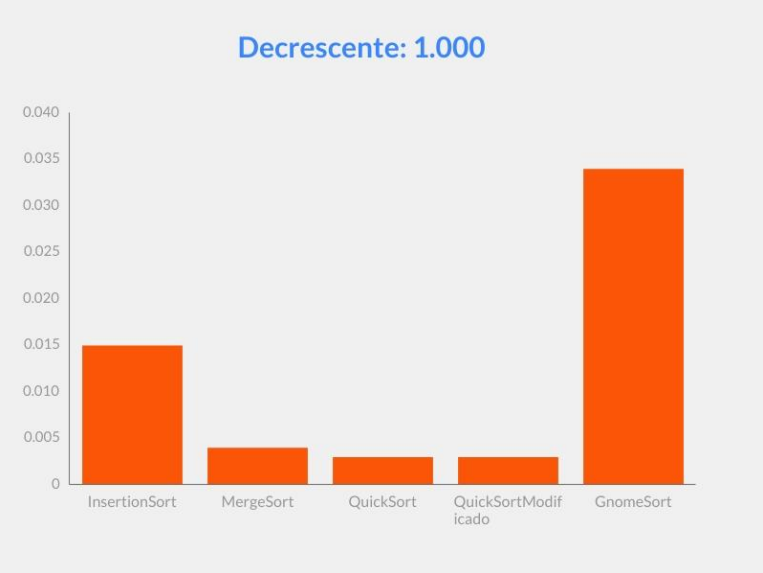
Tempo InsertionSort: 2.087

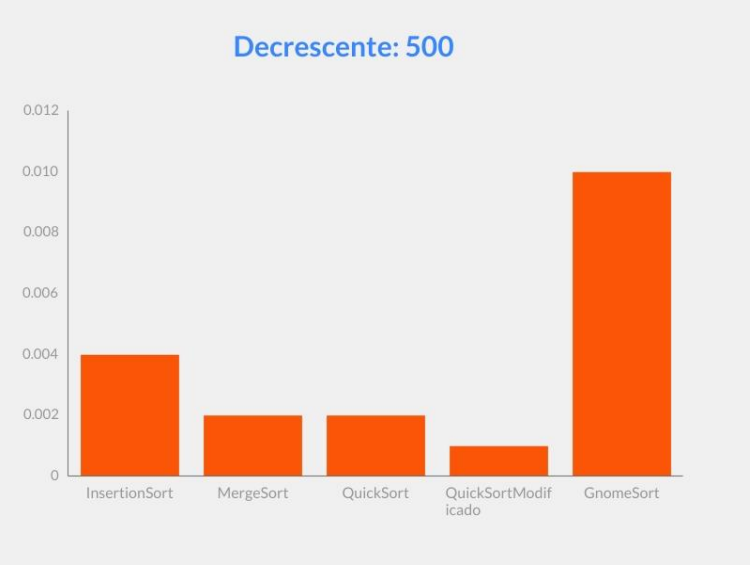
Tempo GnomeSort: 5.871



Tempo InsertionSort: 0.308

Tempo GnomeSort: 0.823





6. Conclusão

Esse trabalho lidou com o problema de ordenar vetores que contém a distância dos robôs até as bases exoplanetárias, na qual a abordagem utilizada para sua resolução foi a leitura dos dados de um .txt e sua ordenação por meio de métodos da classe Ordenacoes.

Com a solução adotada, da análise e dos gráficos construídos, pode-se verificar qual método é mais interessante para a ordenação dos dados na ordem crescente, decrescente ou aleatória.

Por meio da resolução desse trabalho consegui implementar alguns métodos de ordenação e entender como funcionam na prática, além de estudar cada um e compará-los.

Além dos clássicos desafios de programação, majoritariamente o trabalho com ponteiros, não houveram desafios para essa implementação. Após estudar os métodos nas aulas e, claro, com ajuda de fóruns de programação online, não houveram grandes desafios para implementar a solução. No entanto, um problema que tive para completar o trabalho foi rodar o programa com o método de ordenação **GnomeSort**, que se mostrou extremamente lento, quase o suficiente para que eu refizesse o código com um algoritmo mais rápido.

References

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.