



SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO  
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV  
CAMPUS FLORESTAL

**TRABALHO PRÁTICO 3 - AEDS I**  
**SISTEMA PARA ORDENAÇÃO DE ROCHAS**

ANA CLARA OLIVEIRA SOARES [5896]  
LARA GEOVANA DE ALMEIDA [5897]  
MARIA EDUARDA DUARTE LACERDA [5920]

FLORESTAL - MG  
2025

## SUMÁRIO

<b>1. INTRODUÇÃO</b>	<b>2</b>
<b>2. ORGANIZAÇÃO</b>	<b>3</b>
<b>3. DESENVOLVIMENTO</b>	<b>4</b>
3.1 TIPOS ABSTRATOS DE DADOS(TADs)	4
3.2 INSERÇÃO	6
3.3 QUICKSORT	7
<b>4. COMPILAÇÃO E EXECUÇÃO</b>	<b>10</b>
<b>5. RESULTADOS</b>	<b>11</b>
<b>6.CONCLUSÃO</b>	<b>15</b>
<b>7. REFERÊNCIAS</b>	<b>16</b>

## **1. INTRODUÇÃO**

O trabalho prático 3 da disciplina de Algoritmos e Estruturas de Dados I tinha como objetivo colocar em prática os conceitos aprendidos ao longo de toda a matéria, mas especialmente com foco nos Algoritmos de Ordenação, especialmente observando o comportamento e diferença dos algoritmos simples e sofisticados em relação ao tamanho da lista a ser ordenada.

No desenvolvimento foi utilizada os Tipos Abstratos de Dados(TADs) desenvolvidos no trabalho prático 1 da mesma disciplina, apenas realizando algumas alterações para que a implementação ficasse condizente com a especificação do trabalho.

## 2. ORGANIZAÇÃO

Na organização do projeto foi utilizado o GitHub[1], criando um repositório denominado “OrdenaRochasMinerais” onde todos os integrantes do trio puderam ter acesso e contribuir para a realização do trabalho. Na Figura 1 é possível visualizar a organização do repositório, na pasta Testes estão todos os arquivos utilizados para testar o nosso projeto.

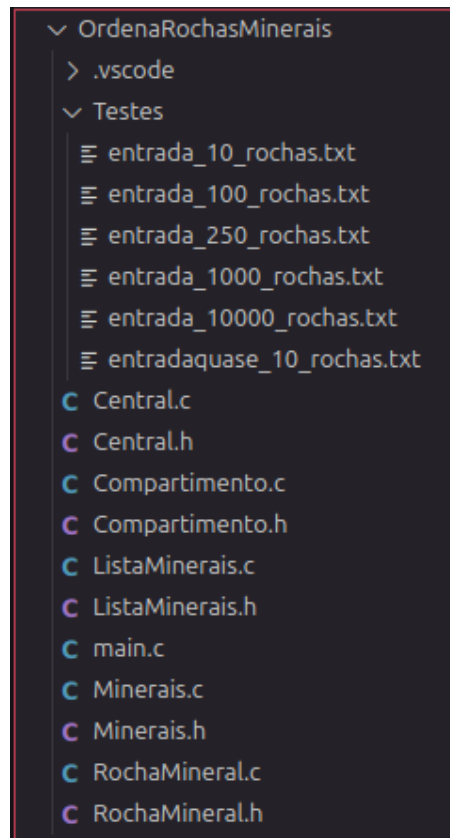


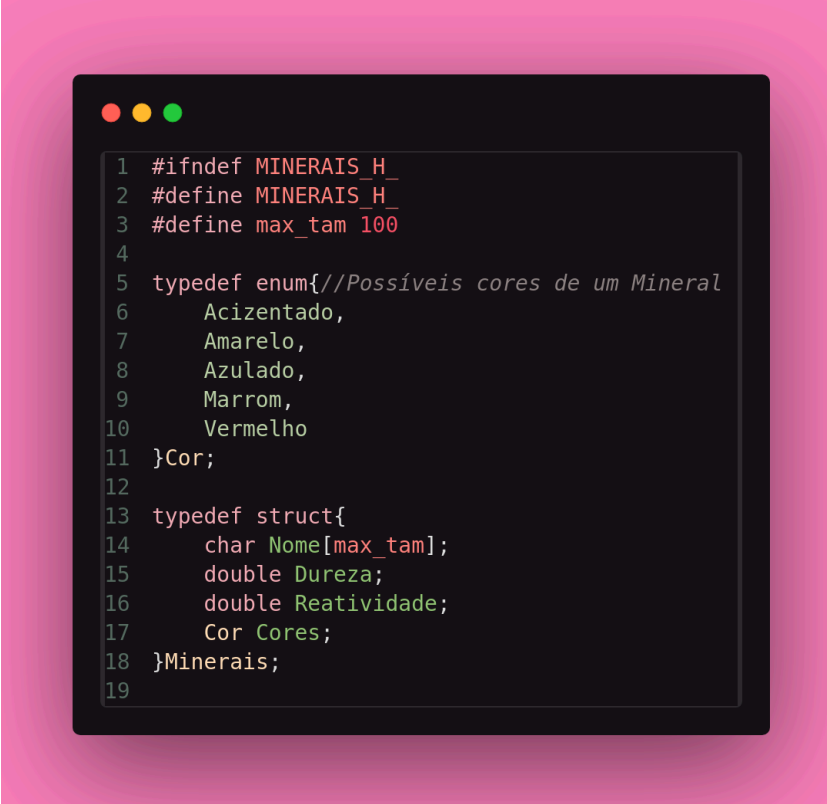
Figura 1 - Repositório do projeto.

### 3. DESENVOLVIMENTO

O primeiro passo para iniciar o desenvolvimento do trabalho prático foi a organização dos Tipos Abstratos de Dados(TADs) necessários.

#### 3.1 TIPOS ABSTRATOS DE DADOS(TADs)

Os TADs utilizados nesta implementação foram os TADs Minerais, Lista de Minerais, Rocha Mineral e Compartimento, no entanto apenas o TAD Compartimento sofreu alterações, as Figuras 2, 3, 4 mostram a estrutura destes inalterados.



```
1 #ifndef MINERAIS_H
2 #define MINERAIS_H
3 #define max_tam 100
4
5 typedef enum{//Possíveis cores de um Mineral
6     Acizentado,
7     Amarelo,
8     Azulado,
9     Marrom,
10    Vermelho
11 }Cor;
12
13 typedef struct{
14     char Nome[max_tam];
15     double Dureza;
16     double Reatividade;
17     Cor Cores;
18 }Minerais;
19
```

Figura 2 - Minerais.h

```

1  #ifndef LISTAMINERAIS_H_
2  #define LISTAMINERAIS_H_
3  #define InicioArranjo 0
4  #define MaxTam 3
5  #include "Minerais.h"
6
7  typedef struct {
8
9      Minerais listaminerais[MaxTam];
10     int Primeiro, Ultimo;
11
12 } ListaMinerais;

```

Figura 3 - ListaMinerais.h

```

1  #ifndef ROCHAMINERAL_H_INCLUDED
2  #define ROCHAMINERAL_H_INCLUDED
3  #include "ListaMinerais.h"
4
5  typedef struct{
6      int Identificador;
7      double Peso;
8      char Categoria[max_tam];
9      ListaMinerais ListaM;
10     double Latitude, Longitude;
11 } RochaMineral;

```

Figura 4 - RochaMineral.h

Em relação, ao Tipo Abstrato de Dados(TADs) Compartimento houve uma mudança significativa, anteriormente ele era uma lista encadeada no entanto para esta implementação a lista linear feita com array seria compatível. Além disso, outras funções foram modificadas e criadas para seguir a especificação do Sistema de Ordenação, na Figura 5 mostra como ficou a estrutura do Compartimento após as alterações.



```
1  #ifndef COMPARTIMENTO_H_
2  #define COMPARTIMENTO_H_
3  #include "RochaMineral.h"
4  #define TamMax 10000
5
6
7  typedef struct {
8
9      RochaMineral ListaRochas[TamMax];
10     int Primeiro, Ultimo;
11 }
12 Compartimento;
13
```

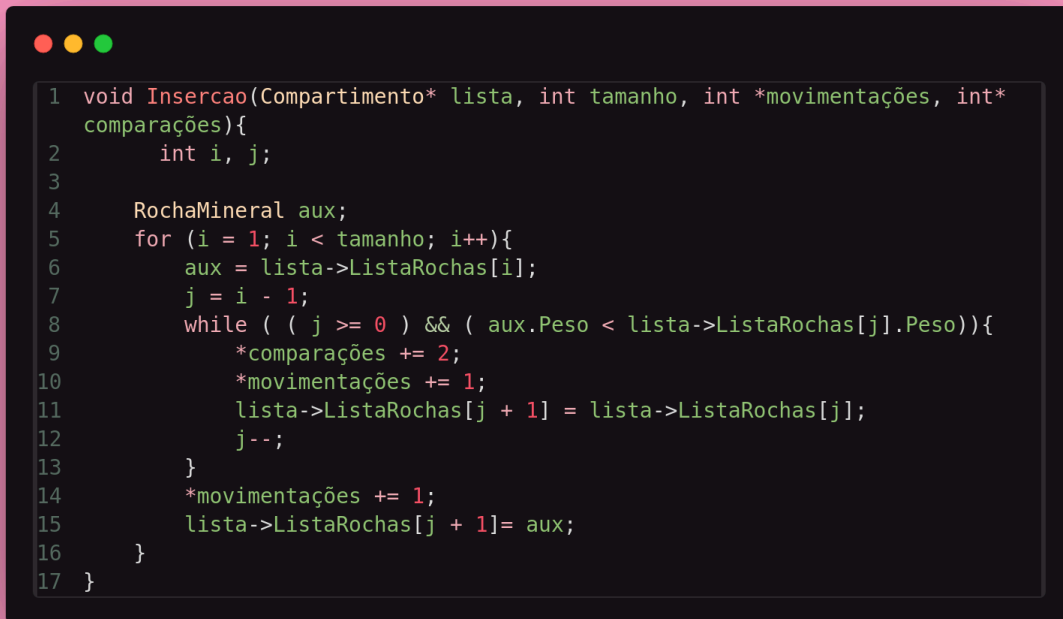
Figura 5 - Compartimento.h

### 3.2 INSERÇÃO

Seguindo a especificação do trabalho, escolhemos um algoritmo de ordenação simples para ordenar a lista de rochas de acordo com seu peso, o algoritmo escolhido foi o inserção, ele funciona comparando os itens da esquerda para a direita,

O inserção possui pior caso quando a lista está de forma decrescente e melhor caso quando já está completamente ordenado. O seu comportamento assintótico é de  $O(n^2)$ , uma de suas vantagens é que ele é um algoritmo estável.

A função de ordenação utilizando Inserção foi criada no TAD Compartimento, assim como o solicitado na especificação, na Figura 6 é possível visualizar como ele foi estruturado. As variáveis auxiliares para a contagem das movimentações e comparações foram passadas como parâmetro para que pudessem ser utilizadas após a ordenação pela função de impressão do compartimento, que será abordada posteriormente.



```
1 void Insercao(Compartimento* lista, int tamanho, int *movimentações, int*
   comparações){
2     int i, j;
3
4     RochaMineral aux;
5     for (i = 1; i < tamanho; i++){
6         aux = lista->ListaRochas[i];
7         j = i - 1;
8         while ( ( j >= 0 ) && ( aux.Peso < lista->ListaRochas[j].Peso)){
9             *comparações += 2;
10            *movimentações += 1;
11            lista->ListaRochas[j + 1] = lista->ListaRochas[j];
12            j--;
13        }
14        *movimentações += 1;
15        lista->ListaRochas[j + 1] = aux;
16    }
17 }
```

**Figura 6 - Função Inserção**

### 3.3 QUICKSORT

Em relação ao algoritmo de ordenação sofisticado o escolhido foi o QuickSort, ele é o algoritmo mais rápido e eficiente na maioria das situações e o mais utilizado. A lógica principal dele é de dividir para conquistar, ou seja, ele ordena partes menores até tudo estar ordenado.

O QuickSort foi dividido em três funções criadas no TAD Compartimento, sendo elas denominadas QuickSort, Ordena e Partição, sendo entre elas a mais importante a função Partição pois ela calcula qual será o pivô e a partir disso as partições para ordenação são definidas. O QuickSort possui o pior caso quando é escolhido um pivô ruim, o que implica em uma complexidade de  $O(n^2)$ , no entanto



em média o tempo de execução é  $O(n \log n)$ . Nas Figuras 7 e 8 é possível visualizar as funções utilizadas no algoritmo de ordenação QuickSort, e da mesma forma que no Inserção as variáveis para contagem das movimentações e comparações foram passadas como parâmetro para serem utilizadas posteriormente.

```
1 void QuickSort(Compartimento *lista, int n, int *movimentacoes, int *
  comparacoes){
2     Ordena(0, n-1, lista, movimentacoes, comparacoes);
3 }
4 void Ordena(int Esq, int Dir, Compartimento *lista, int *movimentacoes, int
  *comparacoes){
5     int i,j;
6     Particao(Esq, Dir, &i, &j, lista->ListaRochas, movimentacoes,
  comparacoes);
7
8     if (Esq < j){
9         *comparacoes += 1;
10        Ordena(Esq, j, lista, movimentacoes, comparacoes);
11    }
12    if (i < Dir) {
13        *comparacoes += 1;
14        Ordena(i, Dir, lista, movimentacoes, comparacoes);
15    }
16 }
```

Figura 7 - QuickSort e Ordena

```
1 void Particao(int Esq, int Dir, int *i, int *j, RochaMineral* rocha, int *
  movimentacoes, int *comparacoes){
2     RochaMineral pivo, aux;
3     *i = Esq; *j = Dir;
4     pivo = rocha[(*i + *j)/2];
5     do{
6         while (pivo.Peso > rocha[*i].Peso){
7             *comparacoes += 1;
8             (*i)++;
9         }
10        while (pivo.Peso < rocha[*j].Peso){
11            *comparacoes += 1;
12            (*j)--;
13        }
14        if (*i <= *j){
15            *comparacoes += 1;
16            *movimentacoes += 1;
17
18            aux = rocha[*i];
19            rocha[*i] = rocha[*j];
20            rocha[*j] = aux;
21            (*i)++;
22            (*j)--;
23        }
24    } while (*i <= *j);
25    *comparacoes += 1;
26 }
27 }
```

Outro ponto importante do desenvolvimento do trabalho foi a criação de uma central para realizar a entrada de dados por arquivo e leitura, além da inserção das rochas no compartimento e escolha do algoritmo de ordenação a ser utilizado. Além disso, para realizar a impressão do compartimento junto com as outras informações desejadas foi utilizada a função de impressão do TAD Compartimento, na Figura 9 mostra como ficou a configuração da função para atender a saída desejada.



```
1 void ImprimeComp(Compartimento *ListaR, int movimentacoes, int comparacoes,
2 int escolha, double tempo){
3     for(int i = 0; i < ListaR->Ultimo; i++){
4         printf("%s %.1lf\n", ListaR->ListaRochas[i].Categoria, ListaR->
5         ListaRochas[i].Peso);
6     }
7     printf("\n");
8     printf("Comparacoes: %d\n", comparacoes);
9     printf("Movimentacoes: %d\n", movimentacoes);
10    printf("Tempo de execucao: %lf \n", tempo);
11    if(escolha == 1){
12        printf("Algoritmo: Insercao\n");
13    }
14    if(escolha == 2){
15        printf("Algoritmo: QuickSort");
16    }
17 }
```

**Figura 9 - Função ImprimeComp**

#### 4. COMPILAÇÃO E EXECUÇÃO

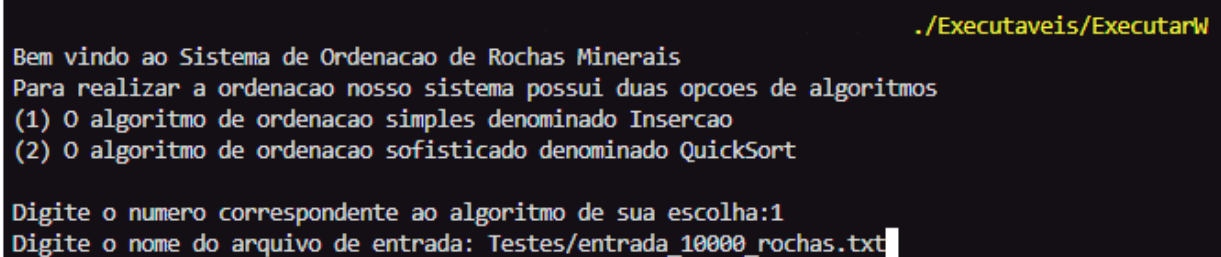
Para a compilação e execução do código, o grupo optou por apenas usar o comando de compilação no terminal e encaminhar junto aos arquivos de código os executáveis gerados por Linux e por Windows em uma pasta destinada para os dois.

Para a compilação no Windows utilizamos o comando “gcc -o Executaveis/ExecutarW main.c Central.c Compartimento.c RochaMineral.c ListaMinerais.c Minerais.c”.

Para o Linux usamos o mesmo comando, renomeando o executável para ExecutarL, então o comando foi “gcc -o Executaveis/ExecutarL main.c Central.c Compartimento.c RochaMineral.c ListaMinerais.c Minerais.c”

Para executar o código, usamos os arquivos já gerados digitando no terminal “./Executaveis/ExecutarW” ou “./Executaveis/ExecutarW”.

Como os códigos usam de leitura de arquivo para funcionar, deixamos os arquivos teste disponíveis em uma pasta chamada “Testes”, então para conferir os resultados de cada entrada, ao executar o código deve-se escolher qual o algoritmo de ordenação deseja usar e digitar o nome do arquivo no formato “Testes/nomedesejado”, como mostrado na figura abaixo.



```
./Executaveis/ExecutarW
Bem vindo ao Sistema de Ordenacao de Rochas Minerais
Para realizar a ordenacao nosso sistema possui duas opcoes de algoritmos
(1) O algoritmo de ordenacao simples denominado Insercao
(2) O algoritmo de ordenacao sofisticado denominado QuickSort

Digite o numero correspondente ao algoritmo de sua escolha:1
Digite o nome do arquivo de entrada: Testes/entrada_10000_rochas.txt
```

Figura 10 - Exemplo de como usar as entradas disponíveis

## 5. RESULTADOS

Os resultados obtidos com os algoritmos de ordenação com cada número de rochas foram colocados na seguinte tabela mostrada na Figura 10, com essa tabela é possível observar a diferença gigantesca entre os dois algoritmos.

<b>ALGORITMOS DE ORDENAÇÃO</b>						
TAMANHO DA ENTRADA	INSERÇÃO			QUICKSORT		
	MOVIMENTAÇÕES M(n)	COMPARAÇÕES C(n)	TEMPO DE EXECUÇÃO	MOVIMENTAÇÕES M(n)	COMPARAÇÕES C(n)	TEMPO DE EXECUÇÃO
10	35	52	0.014000	11	66	0.014000
250	15620	30742	1.460000	650	2359	0.243000
1000	241898	481798	30.377000	3452	9821	1.717000
10000	24121227	48222456	1232.372000	50219	118297	20.085000

Figura 11 - Tabela

Além disso, outra forma de visualizar é a partir de gráficos, logo a partir da tabela criamos um gráfico que representasse individualmente os resultados de número de Comparações e Movimentações, nas Figuras 11 e 12 mostram respectivamente os gráficos de Movimentações e Comparações.

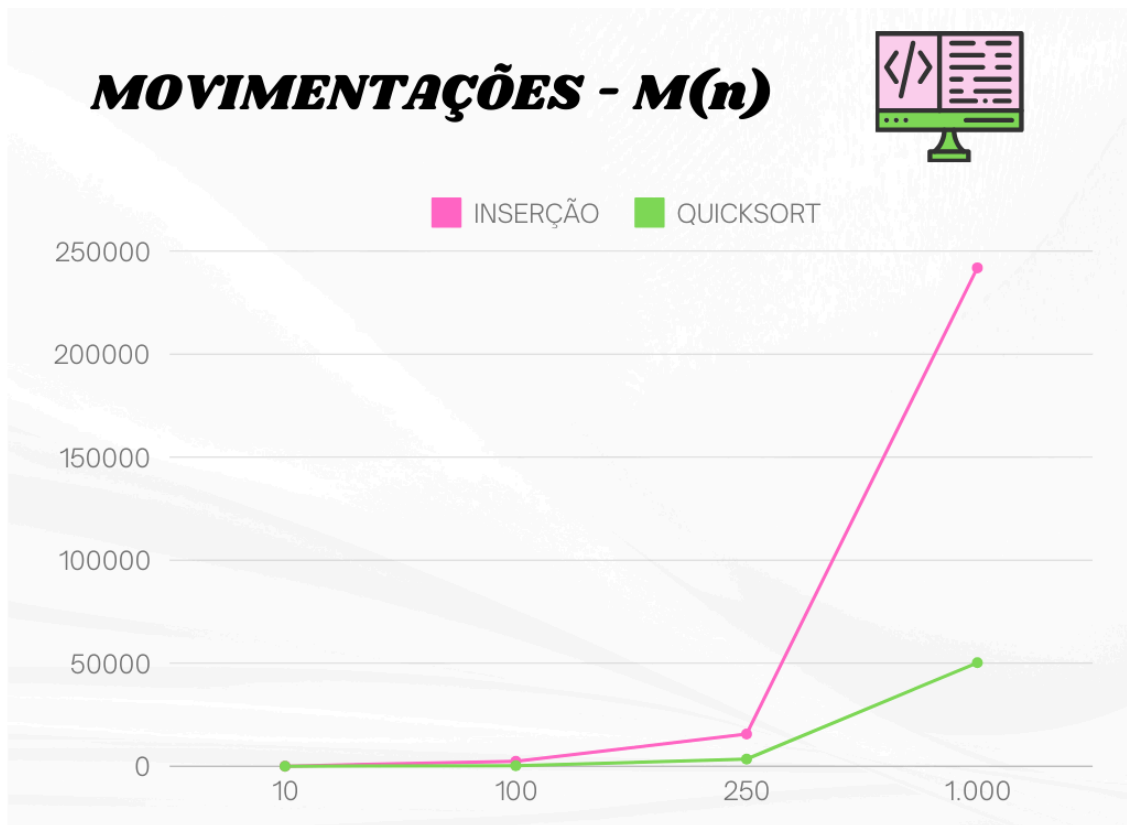


Figura 12 - Gráfico de Movimentações

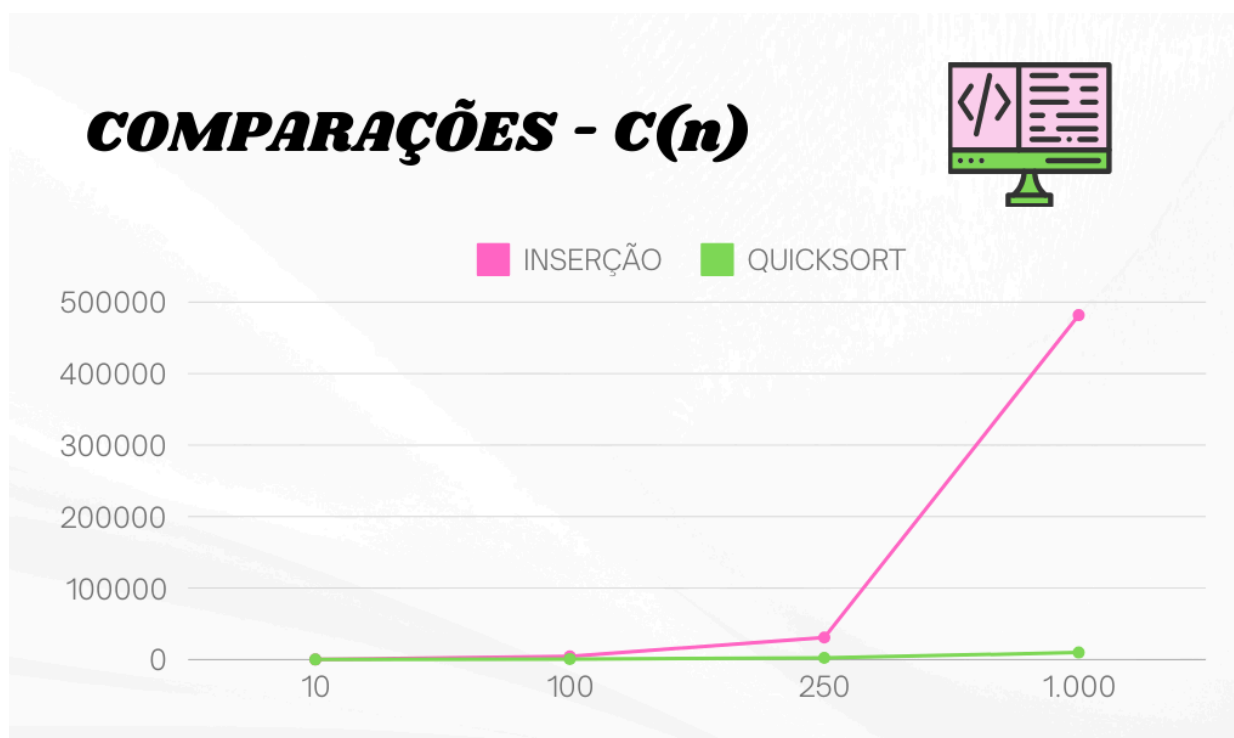


Figura 13 - Gráfico de Comparação

Ao analisar a tabela e os gráficos é possível perceber a diferença entre os dois algoritmos de ordenação, com arquivos menores os dois possuem um desempenho muito parecido, podendo até mesmo o Inserção ter um comportamento consideravelmente melhor se o arquivo estiver quase ordenado, um exemplo disso foi feito utilizando o arquivo de teste entradaquase\_10\_rochas.txt, e os resultados mostraram que o QuickSort realizou um número maior de comparações que o Inserção. Nas Figuras 13 e 14 é possível observar esta diferença.

```
Digite o número correspondente ao algoritmo de sua escolha:1
Digite o nome do arquivo de entrada: Testes/entradaquase_10_rochas.txt
Aquacalis 6.9
Terrasol 7.1
Solaris 8.2
Terrolis 9.8
Solarisfer 10.2
Calquer 11.4
Ferrom 12.5
Terralis 13.6
Aquaterra 14.0
Aquaterra 15.3

Comparacoes: 18
Movimentacoes: 18
Tempo de execucao: 0.011000
Algoritmo: Insercao
```

**Figura 14 - Inserção**

```
Digite o número correspondente ao algoritmo de sua escolha:2
Digite o nome do arquivo de entrada: Testes/entradaquase_10_rochas.txt
Aquacalis 6.9
Terrasol 7.1
Solaris 8.2
Terrolis 9.8
Solarisfer 10.2
Calquer 11.4
Ferrom 12.5
Terralis 13.6
Aquaterra 14.0
Aquaterra 15.3

Comparacoes: 58
Movimentacoes: 10
Tempo de execucao: 0.014000
```

**Figura 15 - QuickSort**

No entanto, ao tratarmos de entradas relativamente grandes este cenário muda, e os gráficos conseguem mostrar isso excepcionalmente bem, pois o número de comparações e movimentações do Inserção sofre um crescimento

gigantesco comparado com o do QuickSort que é quase linear, pois não sofre nenhum crescimento brusco em mesmo com o aumento do arquivo de entrada.

Logo, com essas informações fica claro quais são as melhores ocasiões para o uso de cada algoritmo de ordenação, o Inserção funciona bem com arquivos pequenos e quase ordenados e já o QuickSort funciona bem na maioria das situações, mas tendo um cuidado ao calcular o pivô e tendo em vista que seu desempenho com arquivos pequenos não é o melhor.

## **6.CONCLUSÃO**

Em conclusão, a partir deste trabalho foi possível observar na prática o desempenho e diferenças entre os algoritmos de ordenação escolhidos, a implementação do código foi descomplicada uma vez que já havíamos praticado isto nos trabalhos anteriores, a novidade foi os algoritmos de ordenação que para não haver erro utilizamos os códigos introduzidos na disciplina de Algoritmos e Estruturas de Dados I.

Além disso, o trabalho adicionou em nossa bagagem mais conhecimentos sobre as situações onde estes algoritmos são mais compatíveis e isto é indispensável para nossa formação e conseqüentemente para nosso futuro trabalho.



## 7. REFERÊNCIAS

[1] Github. Disponível em: <<https://github.com/>> Último acesso em: 26 de janeiro de 2025.

[2] Materiais da Disciplina QuickSort. Disponível em: <[https://ava.ufv.br/pluginfile.php/901788/mod\\_resource/content/2/Aula11-ORDQuickSort-2023.pdf](https://ava.ufv.br/pluginfile.php/901788/mod_resource/content/2/Aula11-ORDQuickSort-2023.pdf)> Último acesso em: 26 de janeiro de 2025.

[3] Materiais da Disciplina Inserção. Disponível em: <<https://ava.ufv.br/mod/resource/view.php?id=472913>> Último acesso em: 26 de janeiro de 2025.