##### 

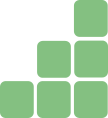
**TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**RELATÓRIO PARA A DISCIPLINA DE ALGORITMOS**

**SOBRE ORDENAÇÃO**

# DISCENTE: BERNARDO MEDEIROS

Natal – RN

2025

# RESUMO DO PROJETO

O projeto consiste na implementação de algoritmos de ordenação, que são procedimentos fundamentais na organização de elementos em uma lista ou array em uma ordem específica, normalmente crescente ou decrescente. A ordenação é uma operação básica e amplamente utilizada, sendo essencial em diversas aplicações, como busca binária, sistemas de banco de dados e análise de dados.

No meu projeto, estou aplicando e comparando algoritmos de ordenação em duas linguagens de programação: C e Java.

.

# Sumário

1. **METODOLOGIA 5**
2. **ALGORITMOS 5**
   1. **BUBBLE SORT 5**
      1. **C 6** 
         1. **Implementação 6**
         2. **Gráfico 7**
      2. **Java 8** 
         1. **Implementação 8**
         2. **Gráfico 9**
      3. **Análise 10**

* 1. **INSERTION SORT 11**
     1. **C 12**
        1. **Implementação 12**
        2. **Gráfico 13**
     2. **Java 14**
        1. **Implementação 14**
        2. **Gráfico 15**
     3. **Análise 16**
  2. **SELECTION SORT 17**
     1. **C 18** 
        1. **Implementação 18**
        2. **Gráfico 19**
     2. **Java 20**
        1. **Implementação 20**
        2. **Gráfico 21**
     3. **Análise 22**
  3. **MERGE SORT 23**
     1. **C 24**
        1. **Implementação 24**
        2. **Gráfico 25**
     2. **Java 26**
        1. **Implementação 26**
        2. **Gráfico 27**
     3. **Análise 28**
  4. **QUICK SORT 29**
     1. **C 30**
        1. **Implementação 30**
        2. **Gráfico 31**
     2. **Python 32**
        1. **Implementação 32**
        2. **Gráfico 33**
     3. **Análise 34**

## 1. METODOLOGIA

A metodologia utilizada consiste na implementação dos algoritmos de ordenação nas linguagens C e Java, contendo variadas quantidades de dados e valores aleatórios desordenados para os testes, a quantidade de dados utilizados variam entre 100, 1000, 10000, 100000 e 1000000. Afim de coletar dados do comportamento do algoritmo com relação ao tamanho de entrada exigida.

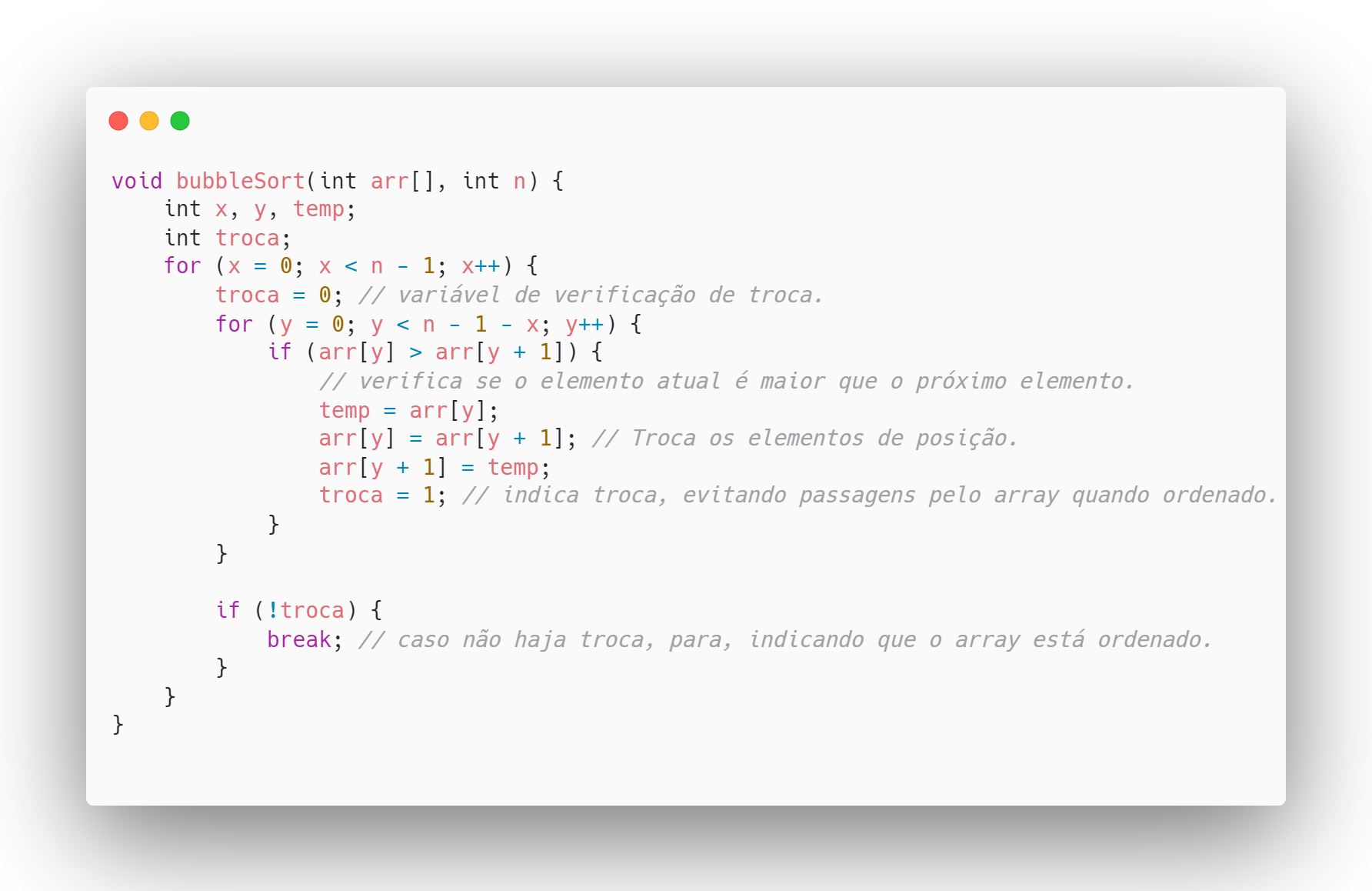
## 2. ALGORITMOS

### 2.1. BUBBLE SORT

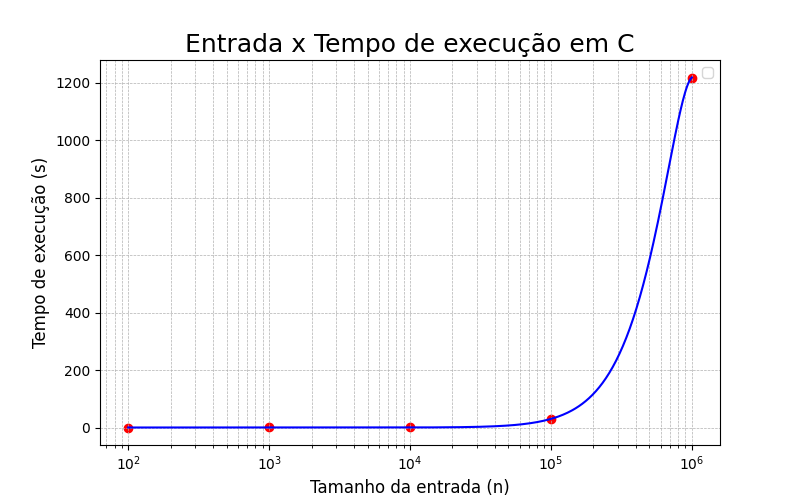
É um dos algoritmos de ordenação mais simples e ineficientes. Funciona comparando repetidamente pares consecutivos de elementos em uma lista e trocando-os caso estiverem na ordem errada. De modo que depois de passar pela lista toda, o maior elemento estará no final da lista (como uma bolha que sobe para o topo). Esse processo continua até que não sejam necessárias mais trocas, quando a lista estará completamente ordenada.

##### 2.1.1. LINGUAGEM - C

##### 2.1.1.1. Implementação

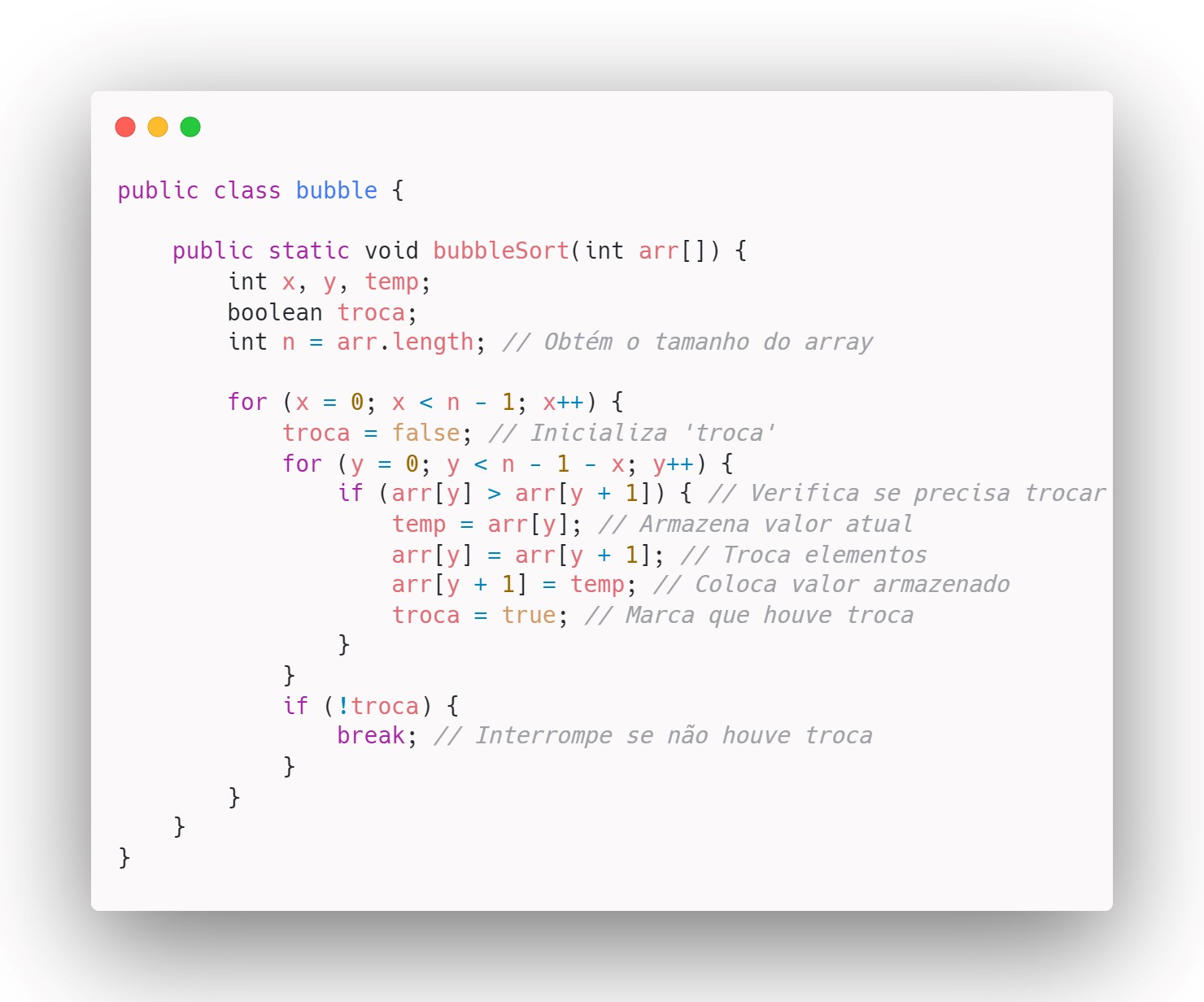


##### 2.1.1.2. Gráfico

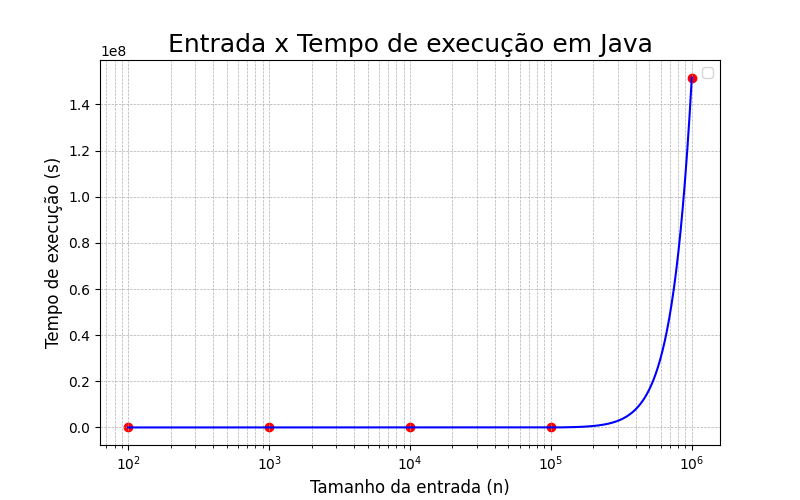
****

#### 2.1.2. LINGUAGEM - JAVA

##### 2.1.2.1. Implementação



**2.1.2.2. Gráfico**



##### 2.1.3. Análise Crítica

O algoritmo Bubble Sort, é objetivamente ineficiente e limitado, tendo a complexidade de tempo O(n²), sendo parcialmente adequado apenas para listas pequenas ou já quase ordenadas, mas há opções melhores de sistemas de ordenação. Tem um desempenho aprimorado na linguagem C quando comparado à implementação em Java, porém, o algoritmo em geral tem o desempenho defasado e extremamente lento, mesmo quando a lista já está ordenada ou parcialmente ordenada, o algoritmo ainda pode continuar fazendo comparações desnecessárias e trocas, o que aumenta o tempo de execução, então, mesmo em seu melhor caso de complexidade O(n), para a maioria das situações, ele não é tão eficiente quanto outros algoritmos.

### 2.2. INSERTION SORT

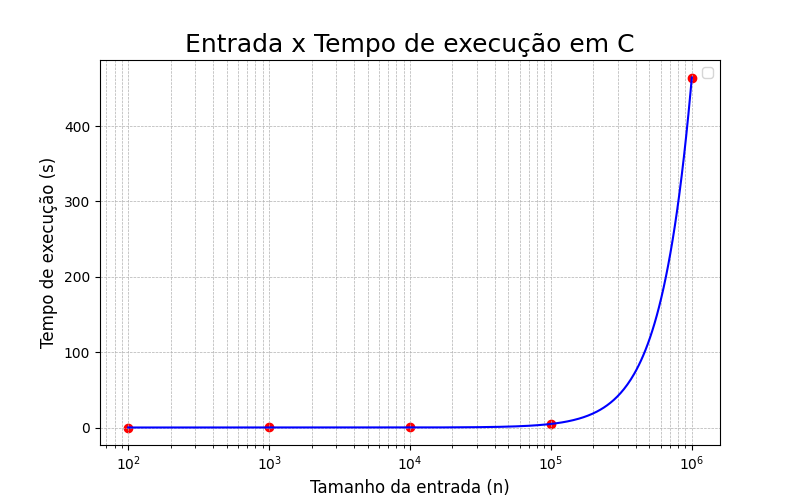
É um algoritmo de ordenação que constrói a lista ordenada, ordenando um elemento de cada vez. Iniciando no segundo elemento, considerando o primeiro já é ordenado por padrão, e em seguida, insere elemento por elemento na posição adequada, comparando com os elementos da parte já ordenada (à esquerda). Repetindo esse processo até que, todos os elementos estejam corretamente posicionados, onde ele não é maior do que o anterior nem menor do que o próximo.

#### 2.2.1. LINGUAGEM - C

##### 2.2.1.1. Implementação



##### 2.2.1.2. Gráfico

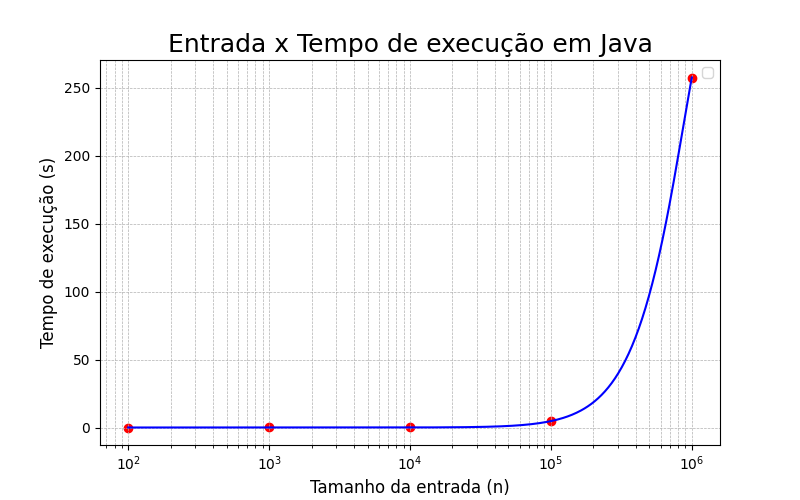


#### 2.2.2. LINGUAGEM - JAVA

##### 2.2.2.1. Implementação



##### 2.3.2.2. Gráfico



##### 2.2.3. Análise Crítica

O algoritmo Insertion Sort é eficiente para listas pequenas ou quase ordenadas. Sua complexidade no melhor caso é O(n), o que é bastante eficiente, porém, para listas grandes, é ineficiente devido à sua complexidade de tempo O(n²) no pior caso, sendo um problema quando lidamos com grandes volumes de dados, em listas de dados completamente aleatórios, tende a ser lento, pois requer muitas comparações e movimentações de elementos. Em conclusão, o alto nível de esforço comparativo impacta o desempenho.

### 

### 2.3. SELECTION SORT

É um algoritmo de ordenação que constrói a lista ordenada selecionando e colocando o menor elemento em sua posição correta, um elemento de cada vez. A cada iteração, o algoritmo encontra o menor elemento na sublista não classificada e o troca com o primeiro elemento da sublista não classificada. Começando no primeiro elemento, considerando que ele é o menor por padrão, o algoritmo repete esse processo de seleção e troca para cada posição subsequente.

#### 2.3.1. LINGUAGEM - C

##### 2.3.1.1. Implementação



##### 2.3.1.2. Gráfico

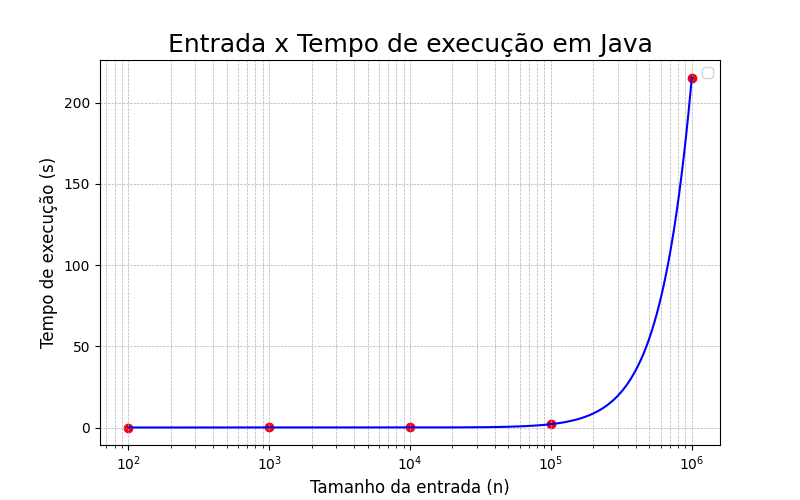


#### 2.3.2. LINGUAGEM - JAVA

##### 2.3.2.1. Implementação



##### 2.3.2.2. Gráfico



##### 2.3.3. Análise Crítica

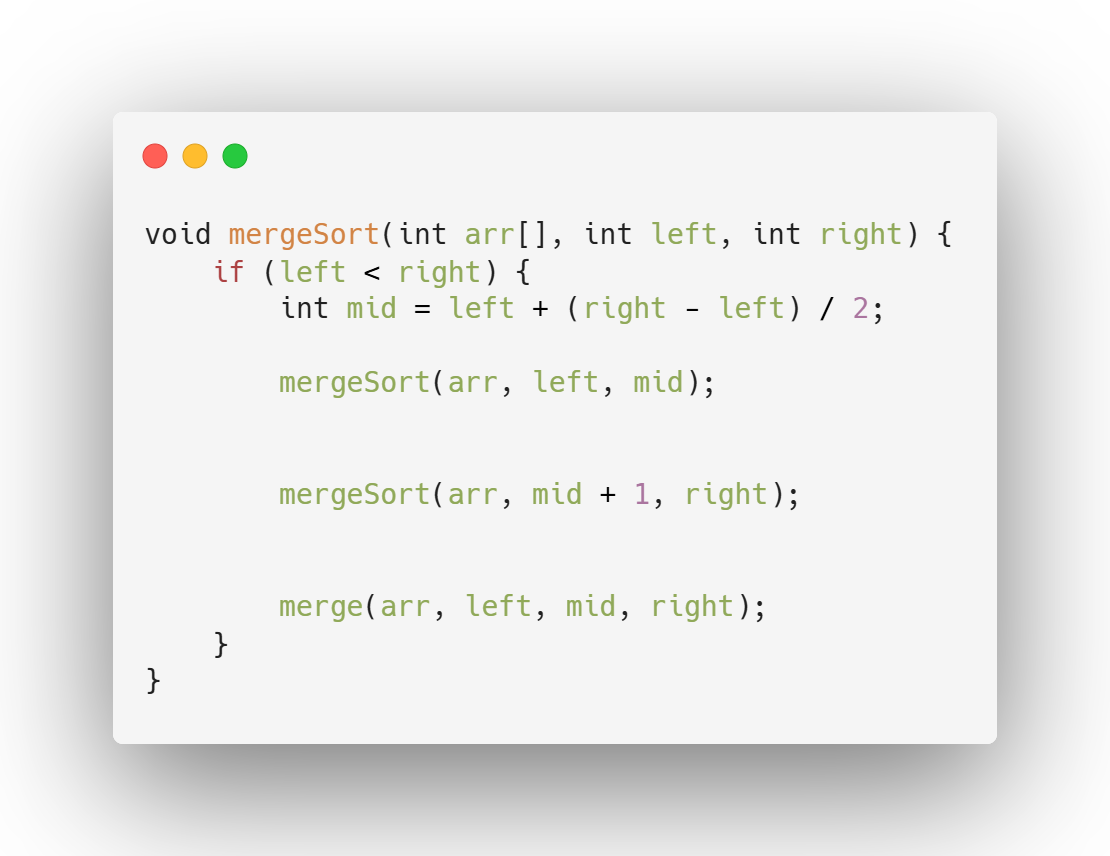
O algoritmo é eficiente para listas pequenas devido à sua simplicidade, mas torna-se ineficiente para grandes volumes de dados por sua complexidade de tempo O(n2) Ele ordena a lista selecionando repetidamente o menor elemento da sublista não classificada e movendo-o para a posição correta. No entanto, realiza um número fixo de comparações e muitas trocas desnecessárias, mesmo em listas quase ordenadas, o que impacta negativamente seu desempenho. Embora utilize pouca memória adicional, sua ineficiência para listas maiores faz com que seja menos desejável em aplicações práticas, onde algoritmos como Merge Sort ou Quick Sort são mais adequados. Em resumo, o Selection Sort tem limitações significativas devido ao esforço comparativo e ao número excessivo de trocas.

### 2.4. MERGE SORT

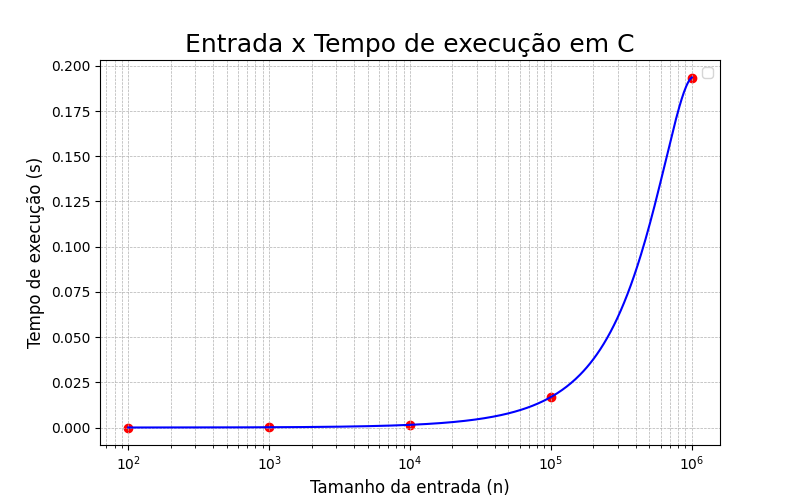
O Merge Sort é um algoritmo eficiente que utiliza o método de divisão e conquista. Seu funcionamento se dá através da divisão repetida da lista em duas metades, até que cada sublista contenha apenas um único elemento. Em seguida, essas sublistas são mescladas de forma ordenada para produzir sublistas maiores, até que a lista completa esteja ordenada. O algoritmo começa dividindo a lista original na metade, repetindo o processo até que o caso base seja atingido, ou seja, até que todas as sublistas contenham apenas um elemento. A partir disso, as sublistas são combinadas de forma ordenada, resultando na lista final completamente ordenada.

#### 2.4.1. LINGUAGEM - C

##### 2.4.1.1. Implementação



##### 2.4.1.2. Gráfico

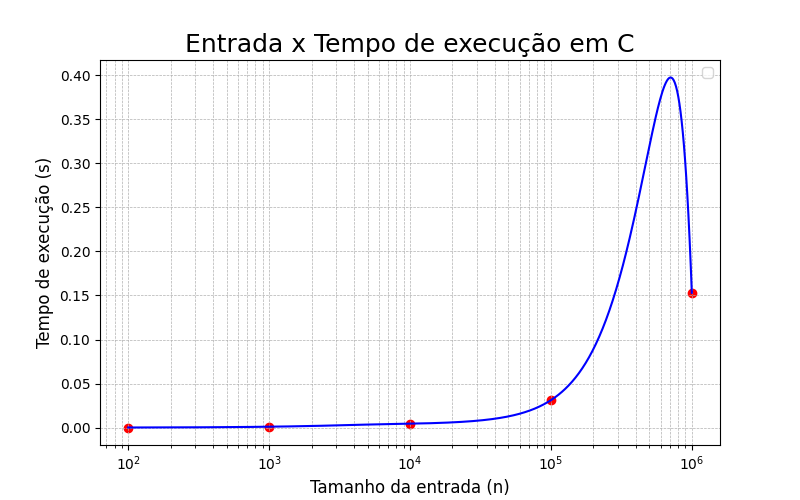


#### 2.4.2. LINGUAGEM - JAVA

##### 2.4.2.1. Implementação



##### 2.4.2.2. Gráfico



##### 2.4.3. Análise Crítica

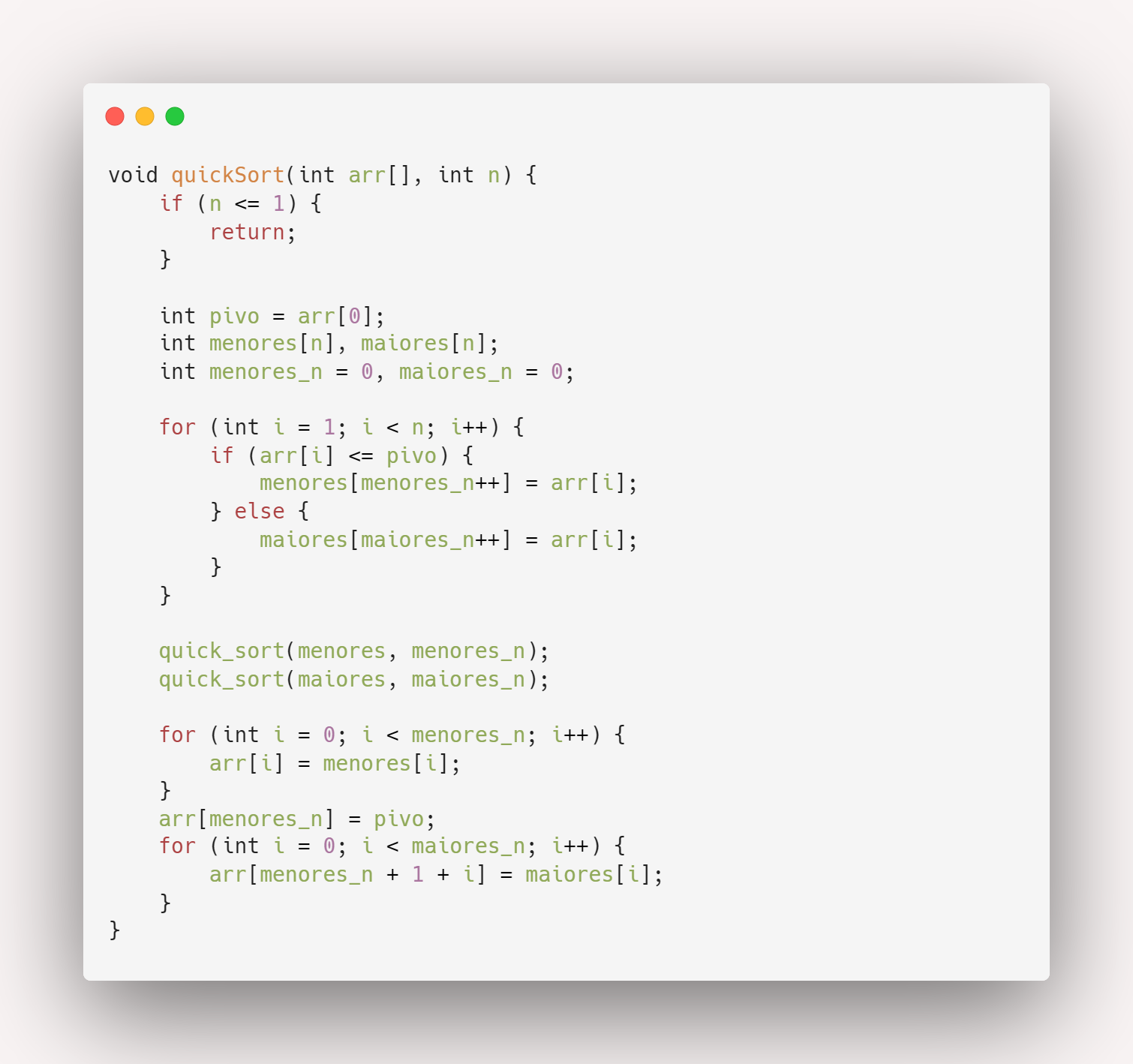
É um algoritmo de ordenação eficiente com uma complexidade média de tempo de O (n log n). É geralmente mais rápido que outros algoritmos de ordenação, especialmente em listas grandes. O Merge Sort é um algoritmo estável, o que significa que ele mantém a ordem relativa dos elementos iguais e consequentemente, a complexidade se mantêm. A estabilidade da ordenação é importante em aplicações onde a ordem dos elementos é crucial. Outra vantagem é que ele oferece um desempenho previsível, pois não é afetado pela escolha de elementos pivô ou partições desequilibradas, proporcionando um desempenho consistente.

### 2.5. QUICK SORT

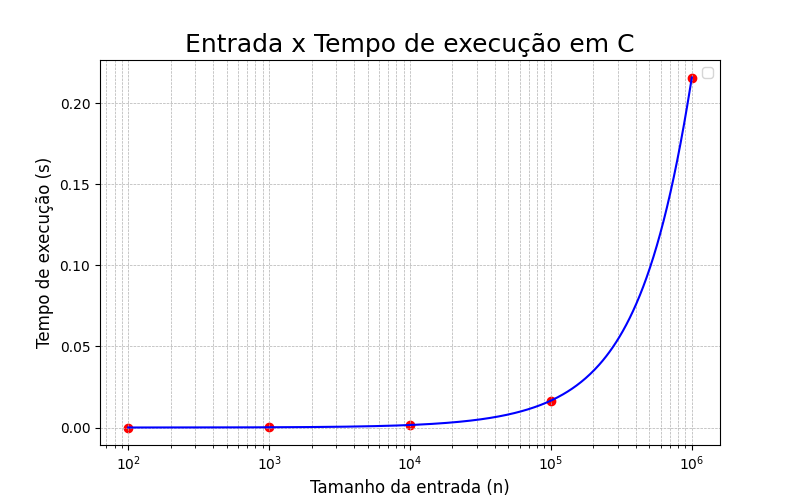
É um algoritmo de ordenação que segue o paradigma de divisão e conquista para ordenar a lista, a partir da seleção de um elemento como pivô e utilizá-lo como parâmetro comparativo a todos os outros elementos da lista, dividindo em sublistas afim de fracionar o problema, reorganizando a lista de forma que todos os elementos menores que o pivô fiquem à esquerda e todos os elementos maiores fiquem à direita, então é aplicado a recursão a cada sublista até que parem no caso base, que é quando há apenas um elemento na lista, ou seja, a recursão só termina quando a lista estiver ordenada.

#### 2.5.1. LINGUAGEM - C

##### 2.5.1.1. Implementação



##### 2.5.1.2. Gráfico



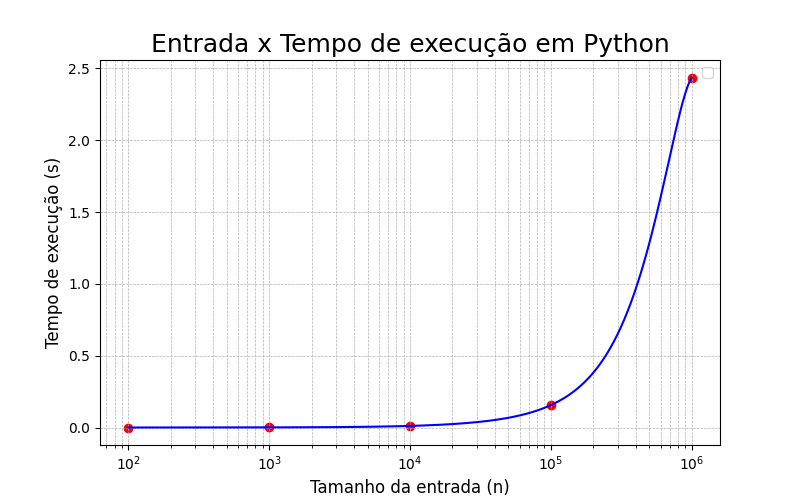
#### 2.5.2. LINGUAGEM - PYTHON

##### 2.5.2.1. Implementação

\*Algoritmo em específico implementado em Python, pois, iria fazer a apresentação em conjunto com o aluno Lucas, que está fazendo em Python, então, implementei para me enquadrar na apresentação dele e fazermos em conjunto, porém, acabei apresentando sozinho.



##### 2.5.2.2. Gráfico



##### 2.5.3. Análise Crítica

O algoritmo é eficiente, com uma complexidade média de tempo de O(n log n), é geralmente mais rápido que outros algoritmos de ordenação, como o Merge Sort, especialmente em listas grandes, tem a capacidade de seleção do pivô a preferência de quem for utilizar, auxiliando significativamente na melhoria da performance em diferentes casos de uso. O pior caso tem a complexidade O(n²), que ocorre quando as partições são extremamente desequilibradas, isso pode acontecer se o pivô escolhido for sempre o maior ou menor elemento da lista. pesar de suas limitações, o Quick Sort continua sendo uma das melhores escolhas para a ordenação de grandes conjuntos de dados, devido à sua eficiência média e ao uso eficiente de memória.