

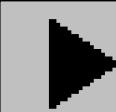
2^a VERIFICAÇÃO DE APRENDIZAGEM

FUNDAMENTOS DE PROBLEMAS COMPUTACIONAIS I

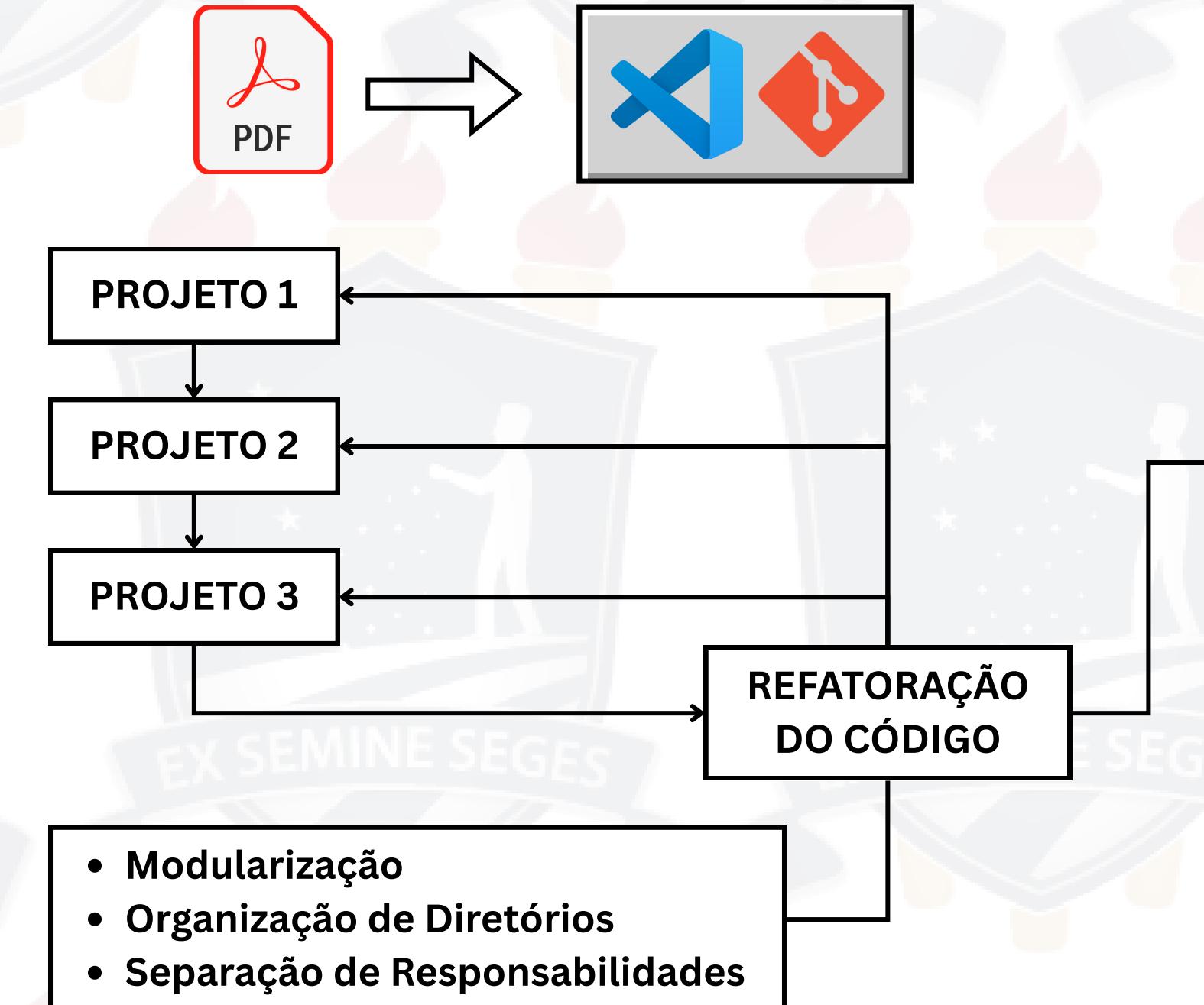
Docente: Prof. Dr. Lucas Cambuim

Discentes: Beatriz Kaori e Pablo Guilherme

Start



PROCESSO DE ELABORAÇÃO



```

fpc1-2VA/
    2VA.pdf
    estruturas_dados/
        __init__.py
        lista_encadeada.py
        lista_encadeada_com_indices.py
        tabela_hash.py
    projeto_1/
        projeto_1.py
        projeto_1_lista_IDS_entrada.txt
        projeto_1_lista_IDS_busca.txt
        projeto_1_resultado_busca_sequencial.txt
        projeto_1_resultado_busca_binaria.txt
        projeto_1_comparacao_desempenho.png
    projeto_2/
        projeto_2.py
        projeto_2_lista_IDS_entrada.txt
        projeto_2_lista_IDS_busca.txt
        projeto_2_resultado_tabela_hash.txt
        projeto_2_comparacao_desempenho.png
    projeto_3/
        projeto_3.py
        projeto_3_lista_produtos_entrada.txt
        projeto_3_requisicoes_listagem.txt
        projeto_3_resultado_insercao_*.txt
        projeto_3_resultado_intercalacao_*.txt
        projeto_3_comparacao_por_criterio.png
        projeto_3_comparacao_geral.png
    
```

Repositório Principal
Documento de especificação
Pacote de Estruturas de Dados Reutilizáveis
Inicialização do pacote Python
Classe ListaEncadeada (busca sequencial)
Classe ListaEncadeadaIndexada (busca binária)
Classe TabelaHash (busca por hash)

Projeto 1: Comparação de Buscas (Seq. vs Binária)
Script principal (testes + gráficos)
Dados de entrada (IDs para inserir)
Dados de busca (IDs para procurar)
Resultados da busca sequencial
Resultados da busca binária
Gráfico comparativo

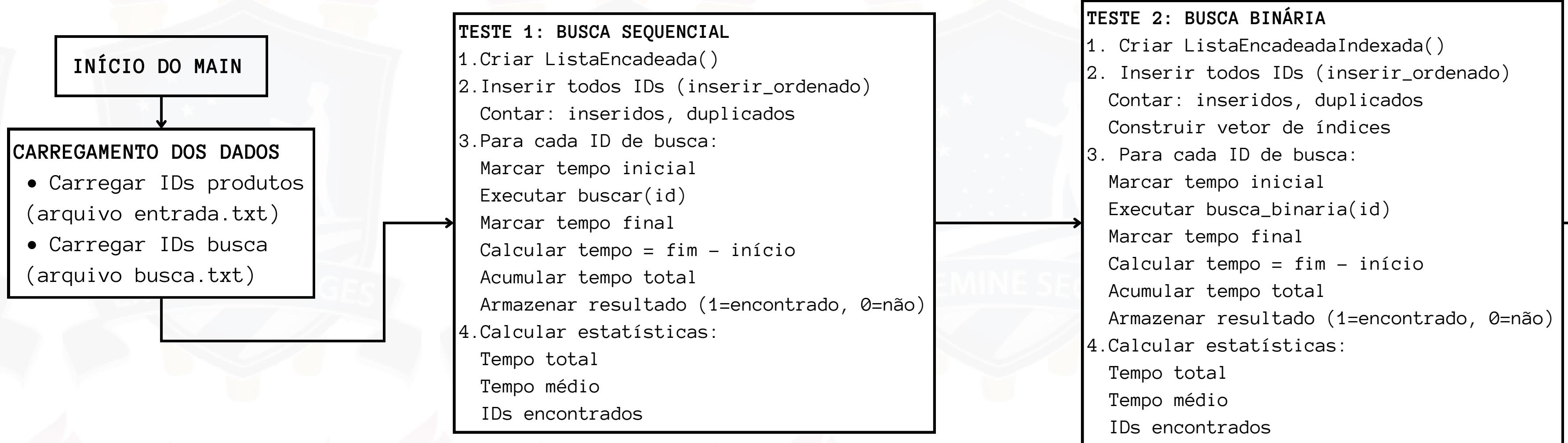
Projeto 2: Comparação de 3 Métodos (Seq. + Bin. + Hash)
Script principal (testes + gráficos)
Dados de entrada (IDs para inserir)
Dados de busca (IDs para procurar)
Resultados da tabela hash
Gráfico comparativo

Projeto 3: Comparação de Algoritmos de Ordenação
Script principal (ordenação + gráficos)
Dados de entrada (produtos)
Critérios de ordenação
Resultados Insertion Sort (5 critérios)
Resultados Merge Sort (5 critérios)
Gráfico: Comparaçao por criterio
Gráfico: Tempo total + média

PROJETO 1

Objetivo

- Implementar, evoluir e comparar um sistema de busca de uma estrutura linear tradicional para uma estrutura indexada que utiliza busca binária



PROJETO 1

Objetivo

- Implementar, evoluir e comparar um sistema de busca de uma estrutura linear tradicional para uma estrutura indexada que utiliza busca binária

COMPARAÇÃO DE DESEMPENHO

- Comparar tempo sequencial vs tempo binária
- Calcular fator de aceleração (quantas vezes mais rápida)

GERAR GRÁFICO COMPARATIVO

- Criar gráfico de barras
- Eixo X: Tipos de busca
- Eixo Y: Tempo (ms)
- Adicionar valores nas barras
- Mostrar fator comparativo
- Salvar: comparacao_desempenho.png

SALVAR ARQUIVOS DE RESULTADO

- Gerar resultado_sequencial.txt (1 ou 0 para cada busca)
- Gerar resultado_binaria.txt (1 ou 0 para cada busca)



Lista Encadeada



```
class No:  
    # Construtor do nó  
>    def __init__(self, id_produto): ...  
  
class ListaEncadeada:  
    # Construtor da lista  
>    def __init__(self): ...  
  
>    def inserir_ordenado(self, id_produto): ...  
  
>    def buscar(self, id_produto): ...
```



IMPLEMENTAÇÃO DE LISTA DE PRODUTOS COMO LIGADORES

S I - 2^a V.A.



```
def inserir_ordenado(self, id_produto): # Insere um ID de produto na lista de forma ordenada
    # e retorna 1 se inserido com sucesso e -1 se já existe
    novo_no = No(id_produto)
    # Caso da lista vazia
    if self.ponta is None:
        self.ponta = novo_no
        self.tamanho += 1
        return 1
    # Caso de inserção no início
    if id_produto < self.ponta.id_produto:
        novo_no.proximo = self.ponta
        self.ponta = novo_no
        self.tamanho += 1
        return 1
    # ID já existe no início
    if id_produto == self.ponta.id_produto:
        return -1
    # Procurar posição correta
    atual = self.ponta
    while atual.proximo is not None and atual.proximo.id_produto < id_produto:
        atual = atual.proximo
    # Verificar se o ID já existe
    if atual.proximo is not None and atual.proximo.id_produto == id_produto:
        return -1
    # Inserir na posição correta
    novo_no.proximo = atual.proximo
    atual.proximo = novo_no
    self.tamanho += 1
    return 1
```

```
def buscar(self, id_produto):
    # Busca um ID de produto na lista e retorna 1 se encontrado e -1 se não encontrado
    atual = self.ponta
    while atual is not None:
        if atual.id_produto == id_produto:
            return 1
        atual = atual.proximo
    return -1
```

na lista de forma ordenada ...

o início da lista (usado quando a ordem não importa) ...

retorna 1 se encontrado e -1 se não encontrado ...

e produtos ...

Lista Encadeada com índices

i

```
estruturas_dados > lista_encadeada_com_indices.py > ...
 1 > """
 5
 6 > class No:...
11
12 < class ListaEncadeadaIndexada:
13     # Lista encadeada com vetor de índices para busca binária
14     # Armazena tuplas no vetor_indices (id, referência_para_nó)
15 >     def __init__(self, auto_indexar=True): # Construtor da Lista com Índices. Se True, atualiza índices a cada inserção...
20
21 >     def inserir_ordenado(self, id_produto): # Insere um ID de produto na lista mantendo a ordem crescente, ...
62
63 >     def _reajustar_indices(self): # Reconstrói o vetor de índices (usado quando auto_indexar=True) ...
69
70 >     def construir_indices(self): # Constrói o vetor de índices após todas as inserções, chamado 1 vez quando auto_indexar=False...
76
77 >     def busca_binaria(self, id_produto): # Realiza busca binária no vetor de índices...
```



Lista Encadeada com índices



```
def construir_indices(self):
    # Constrói o vetor de índices após todas as inserções
    # Deve ser chamado uma única vez após inserir todos os elementos (quando auto_indexar=False)
    self.vetor_indices = []
    atual = self.ponta
    while atual is not None:
        self.vetor_indices.append((atual.id_produto, atual))
        atual = atual.proximo

def busca_binaria(self, id_produto):
    # Realiza busca binária no vetor de índices
    esquerda = 0
    direita = len(self.vetor_indices) - 1

    while esquerda <= direita:
        meio = (esquerda + direita) // 2
        meio_id = self.vetor_indices[meio][0]

        if meio_id == id_produto:
            return 1 # Encontrado
        elif meio_id < id_produto:
            esquerda = meio + 1
        else:
            direita = meio - 1

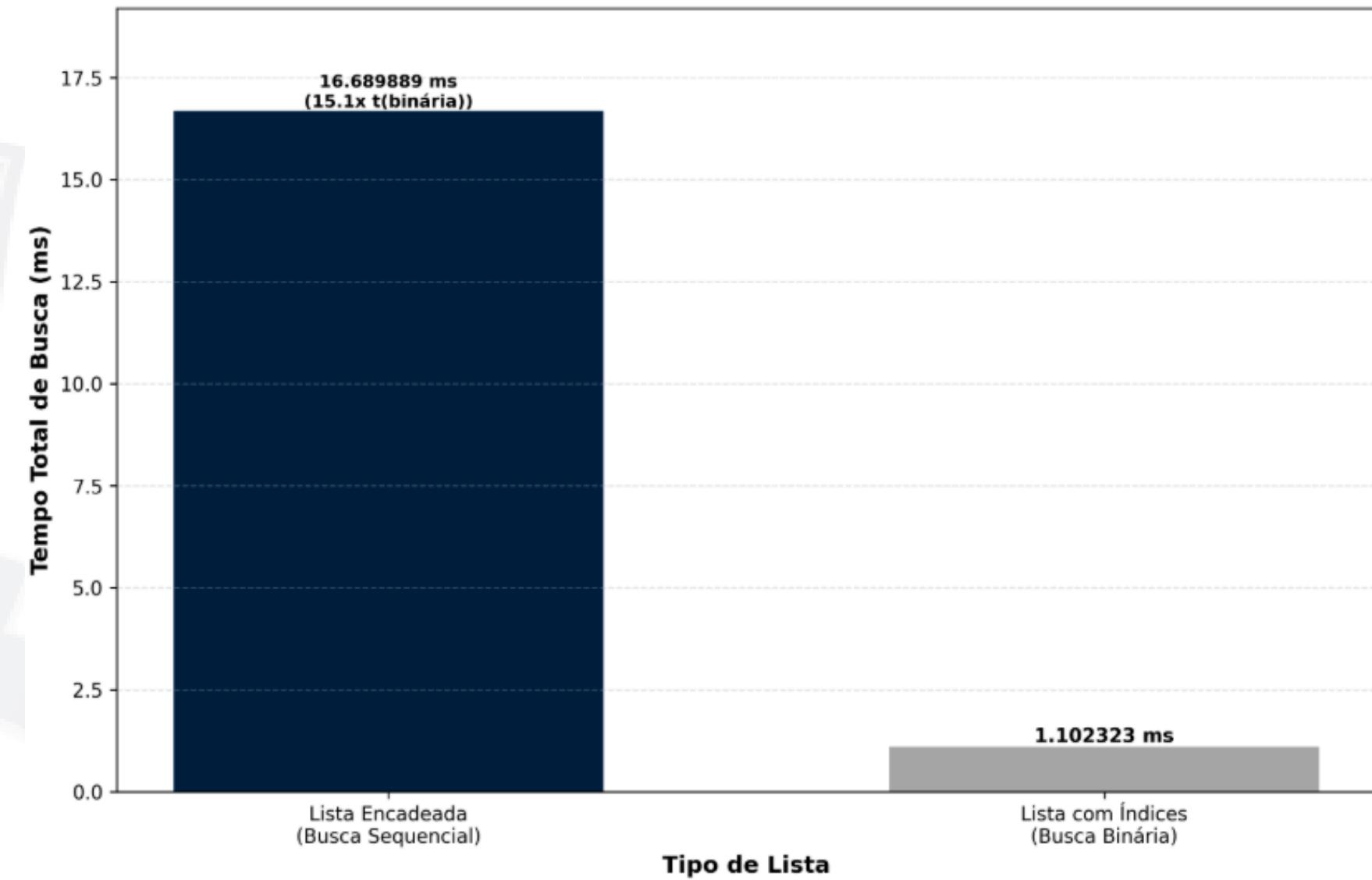
    return -1 # Não encontrado
```



Comparação de desempenho

| Característica | Sequencial $O(n)$ | Binária $O(\log n)$ |
|------------------|-------------------|---------------------|
| Pior Caso | Percorre tudo | Divide pela metade |
| Comparações (1M) | 1.000.000 | 20 |
| Crescimento | Linear | Logarítmico |
| Quando usar | Listas pequenas | Listas grandes |
| Requer ordem? | Não | Sim |
| Vantagem | Simples | Muito rápida |

Comparação de Desempenho: Busca Sequencial vs Busca Binária



PROJETO 2

Objetivo

- Implementar a estrutura de Tabela Hash para solucionar problemas de escalabilidade do catálogo de produtos e comparar com as abordagens do Projeto 1



Tabela Hash



```
class TabelaHash:
    # Tabela hash com encadeamento separado
    def __init__(self, tamanho = 7000):
        self.tamanho = tamanho
        self.tabela = [None] * tamanho

    def _hash(self, id_produto):
        # Função hash simples usando o operador módulo
        return id_produto % self.tamanho

    def inserir(self, id_produto):
        # Insere um ID de produto na tabela hash
        indice = self._hash(id_produto)
        novo_no = No(id_produto)

        # Caso da posição vazia
        if self.tabela[indice] is None:
            self.tabela[indice] = novo_no
            return 1

        # Inserção no início da lista encadeada
        atual = self.tabela[indice]
        if atual.id_produto == id_produto:
            return -1 # ID já existe

        while atual.proximo is not None:
            if atual.proximo.id_produto == id_produto:
                return -1 # ID já existe
            atual = atual.proximo

        atual.proximo = novo_no
        return 1
```

```
def buscar(self, id_produto):
    # Busca um ID de produto na tabela hash
    indice = self._hash(id_produto)
    atual = self.tabela[indice]

    while atual is not None:
        if atual.id_produto == id_produto:
            return 1 # ID encontrado
        atual = atual.proximo

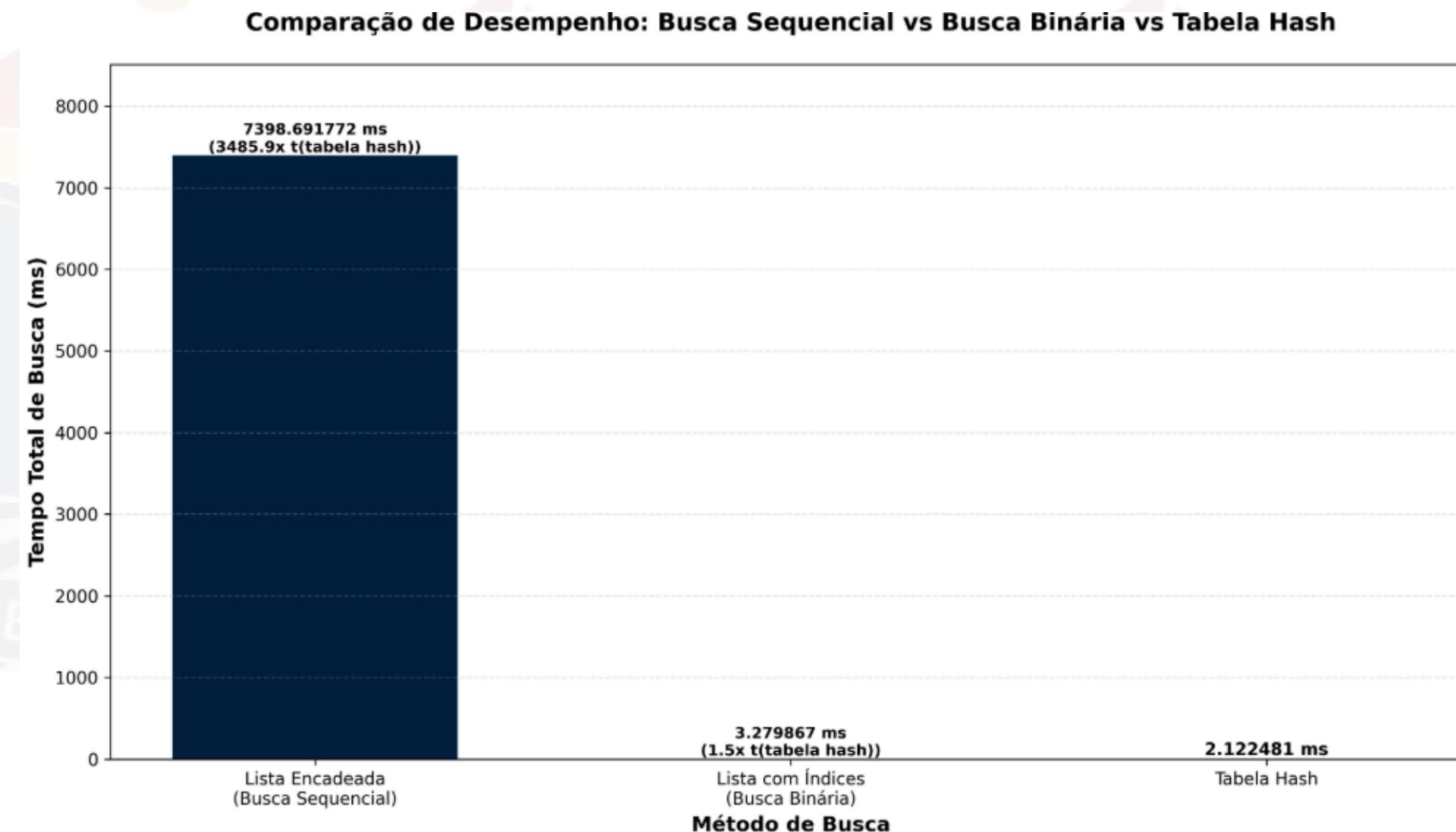
    return -1 # ID não encontrado
```





Comparação de desempenho

| Característica | Sequencial $O(n)$ | Binária $O(\log n)$ | Hash |
|------------------|-------------------|---------------------|---------------------|
| Pior Caso | Percorre tudo | Divide pela metade | Todas colisões |
| Comparações (1M) | 1.000.000 | 20 | 1-15 (~7 médio) |
| Crescimento | Linear | Logarítmico | Constante |
| Quando usar | Listas pequenas | Listas grandes | Muitas buscas |
| Requer ordem? | Não | Sim | Não |
| Vantagem | Simples | Muito rápida | Extremamente rápida |



PROJETO 3

Objetivo

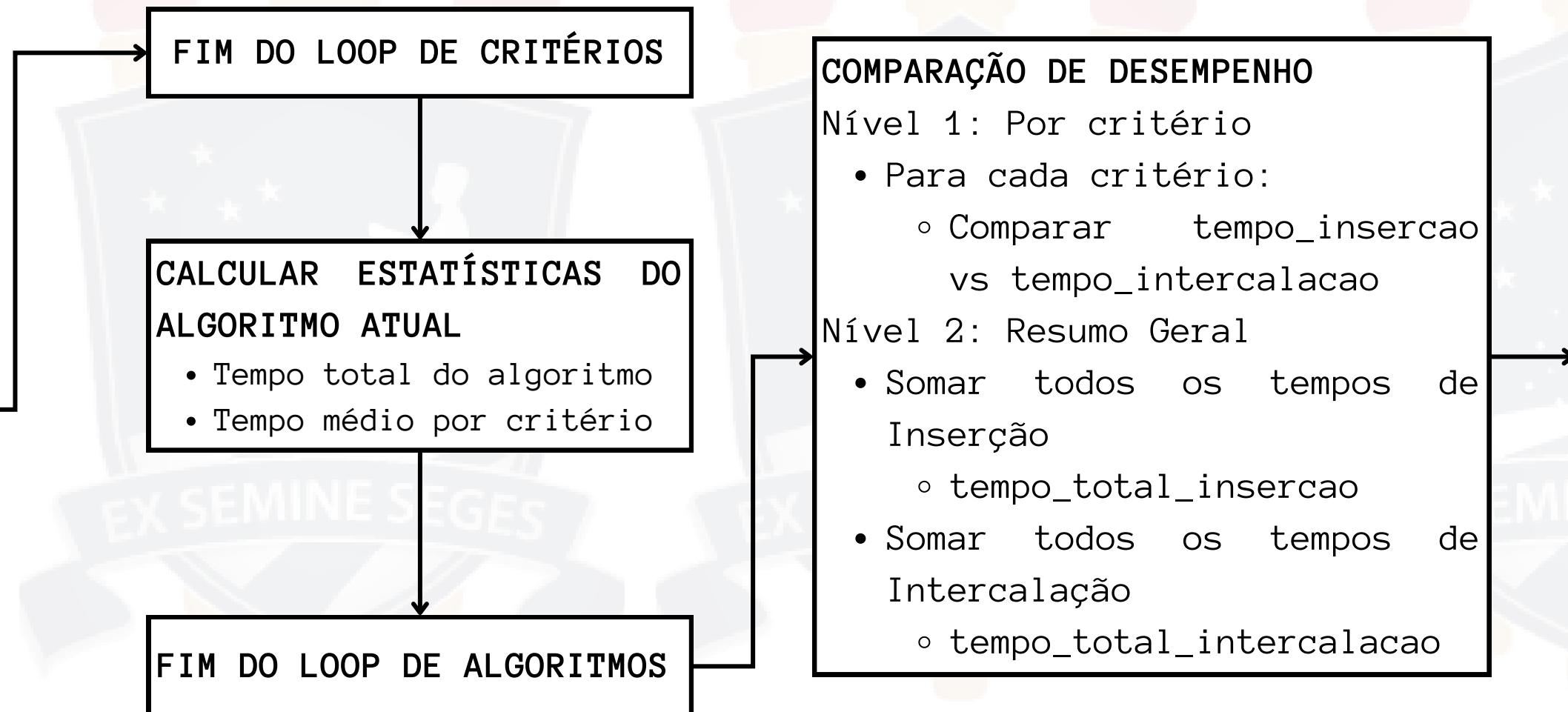
- Implementar e avaliar a eficiência e o comportamento de diferentes algoritmos de ordenação (Insertion Sort e Merge Sort) aplicados a um catálogo de produtos real, desenvolvendo a ordenação flexível dos dados por múltiplos critérios.



PROJETO 3

Objetivo

- Implementar e avaliar a eficiência e o comportamento de diferentes algoritmos de ordenação (Insertion Sort e Merge Sort) aplicados a um catálogo de produtos real, desenvolvendo a ordenação flexível dos dados por múltiplos critérios.



GERAR GRÁFICOS COMPARATIVOS

Gráfico 1: Comparaçao por Critérios

Tipo: Barras agrupadas
 Eixo X: Critérios
 Eixo Y: Tempo (ms)
 2 barras por critérios
 Adicionar valores e fator nas barras
 Salvar: comparacao_por_criterio.png

Nível 2: Resumo Geral

Tipo: Barras simples
 Eixo X: Algoritmos
 Eixo Y: Tempo Total (ms)
 2 barras: inserção e intercalação
 Adicionar informações nos rótulos:

- Tempo total
- Tempo médio
- Fator comparativo

 Salvar: comparacao_geral.png



MÉTRICAS COLETADAS:

- Tempo por critério para cada algoritmo
- Tempo total por algoritmo
- Tempo médio por critério de cada algoritmo
- Algoritmo mais rápido por critério
- Algoritmo mais rápido geral
- Fator de aceleração (X vezes mais rápido)

INSERTION SORT

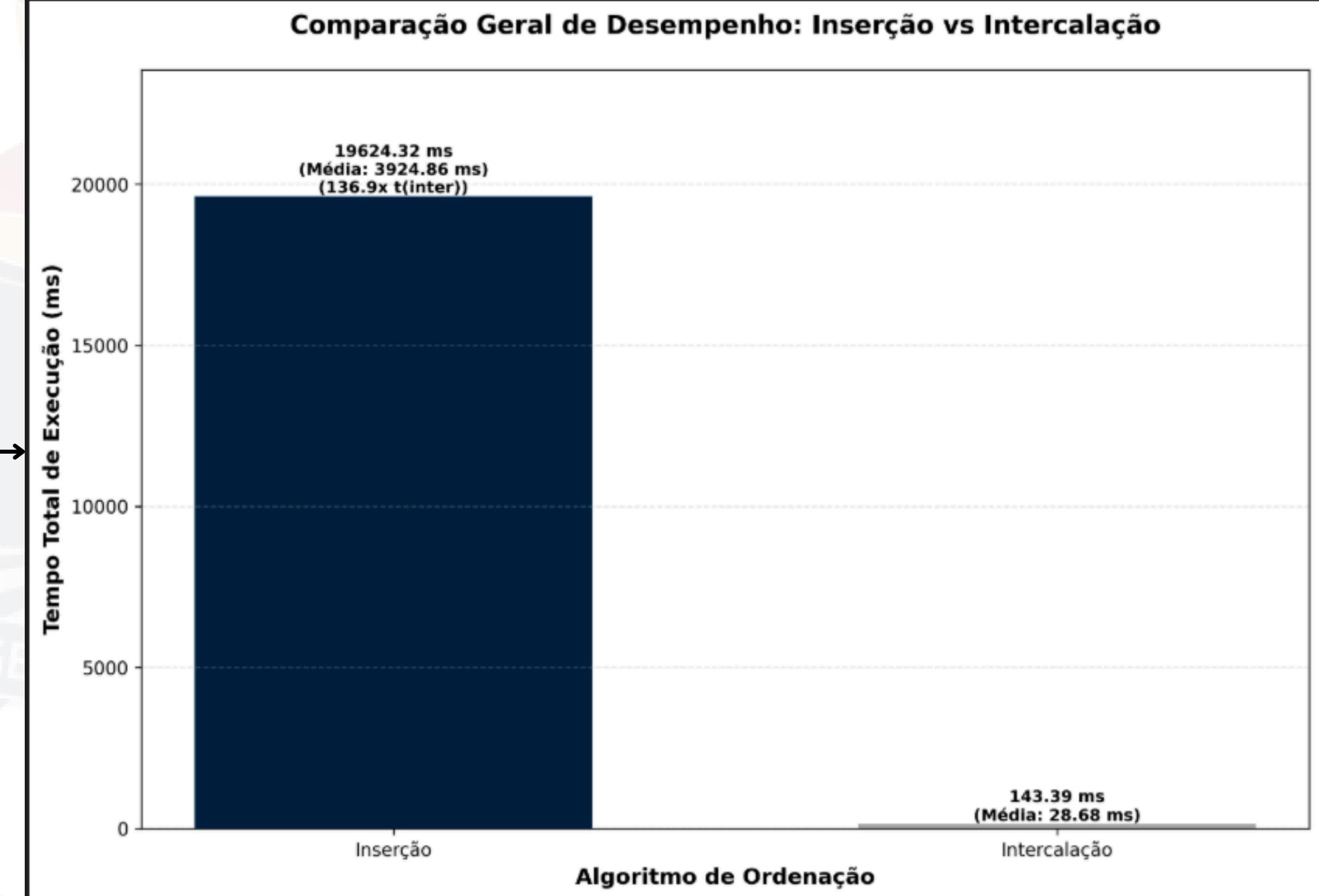
- Complexidade: $O(n^2)$
- Método: Iterativo

MERGE SORT

- Complexidade: $O(n \log n)$
- Método: Recursivo (Dividir e Conquistar)

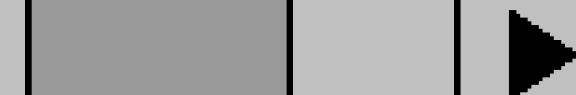
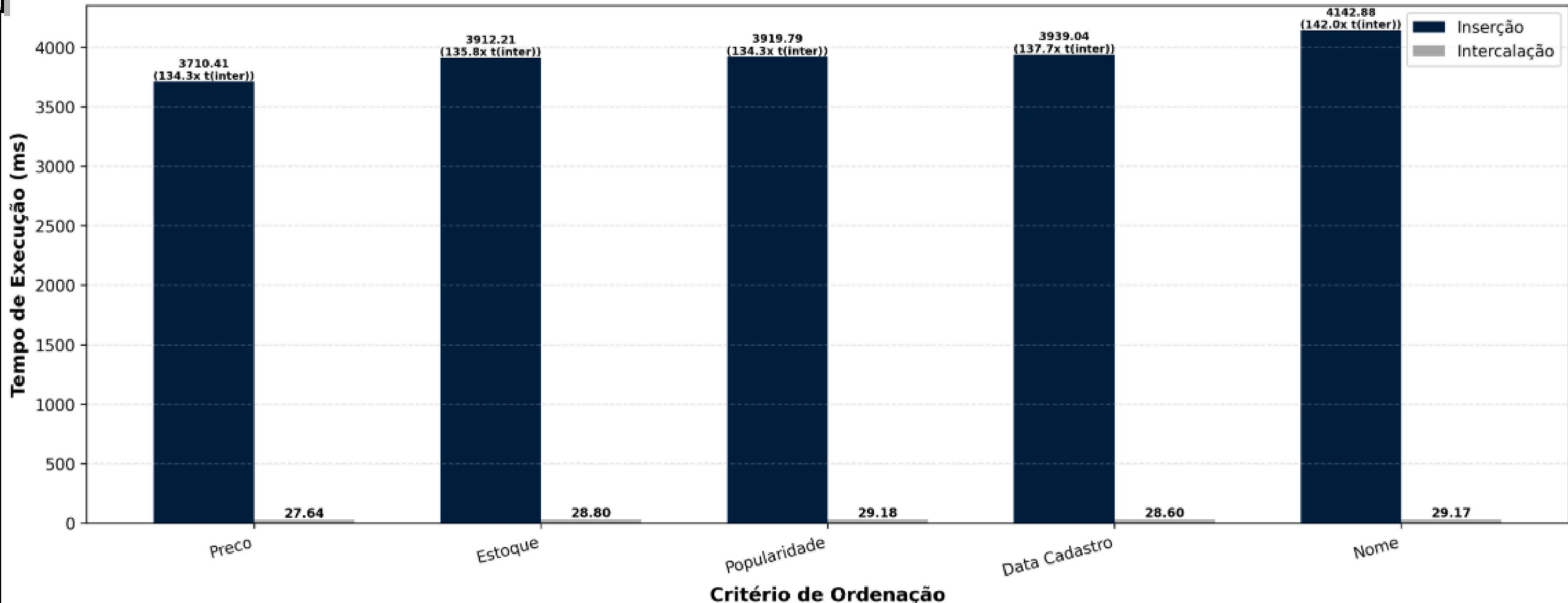
RESULTADO GERAL

- Intercalação foi 136x mais rápido



i

Comparação de Desempenho por Critério de Ordenação



INTERCALAÇÃO

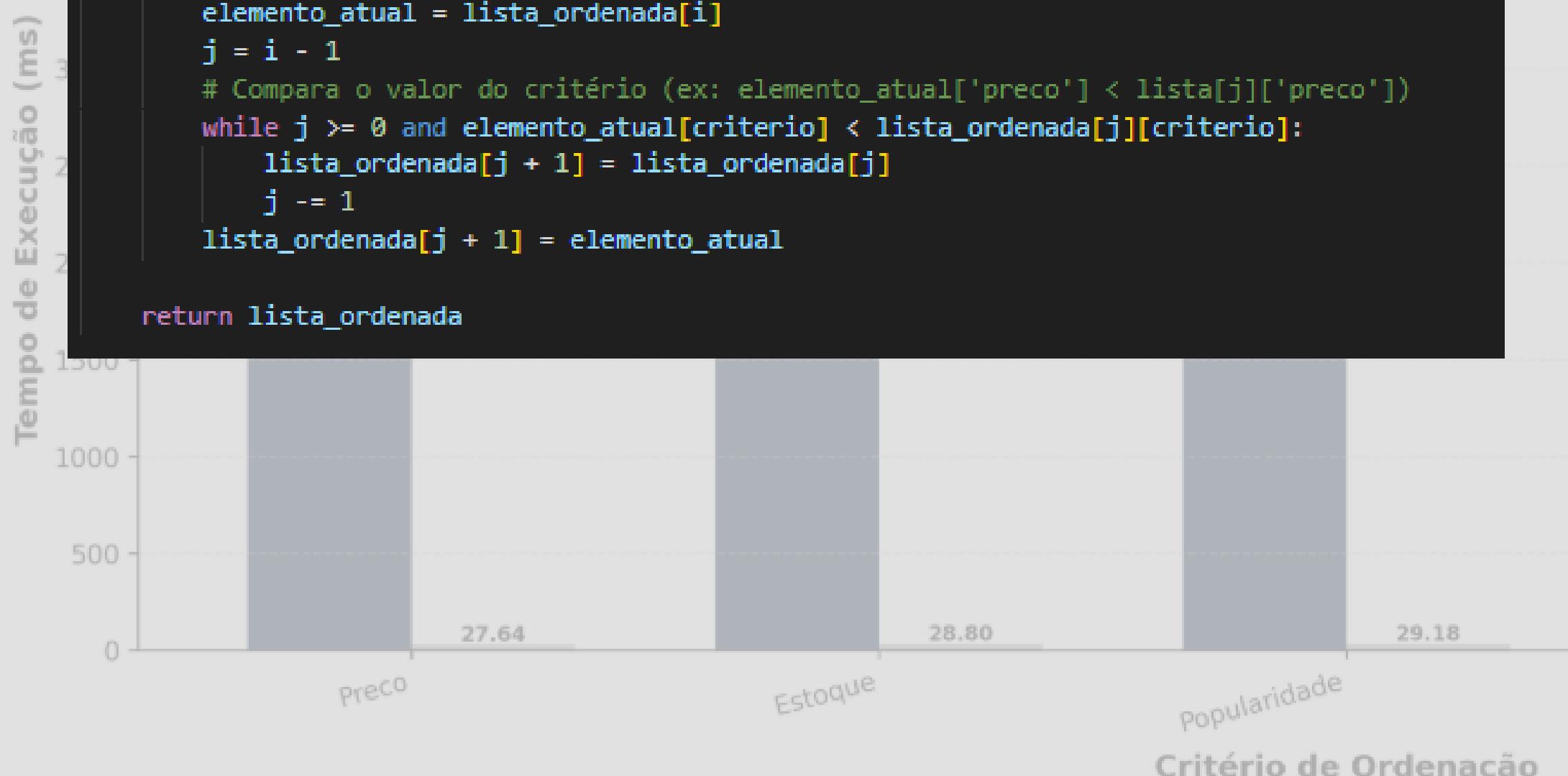


INSERÇÃO

```
def insertion_sort(lista, criterio): # Cria uma cópia da lista para não modificar a original
    lista_ordenada = lista.copy()
    n = len(lista_ordenada)

    for i in range(1, n):
        elemento_atual = lista_ordenada[i]
        j = i - 1
        # Compara o valor do critério (ex: elemento_atual['preco'] < lista[j]['preco'])
        while j >= 0 and elemento_atual[criterio] < lista_ordenada[j][criterio]:
            lista_ordenada[j + 1] = lista_ordenada[j]
            j -= 1
        lista_ordenada[j + 1] = elemento_atual

    return lista_ordenada
```



```
def merge_sort(lista, criterio):
    # Caso base da recursão
    if len(lista) <= 1:
        return lista

    # Divisão
    meio = len(lista) // 2
    esquerda = merge_sort(lista[:meio], criterio)
    direita = merge_sort(lista[meio:], criterio)

    # Intercalação (Conquista)
    return merge(esquerda, direita, criterio)

def merge(esquerda, direita, criterio):
    lista_ordenada = []
    i = j = 0

    while i < len(esquerda) and j < len(direita):
        # Comparação baseada no critério dinâmico
        if esquerda[i][criterio] <= direita[j][criterio]:
            lista_ordenada.append(esquerda[i])
            i += 1
        else:
            lista_ordenada.append(direita[j])
            j += 1

    # Adicionar os elementos restantes
    lista_ordenada.extend(esquerda[i:])
    lista_ordenada.extend(direita[j:])

    return lista_ordenada
```



Fim

OBRIGADO!



COMPARAÇÃO DE DESEMPENHO

```
Lista Encadeada Simples (Busca Sequencial): 6.1044489390 s  
Lista com Índices (Busca Binária): 0.0030614271 s  
Tabela Hash: 0.0013503839 s
```

Método mais rápido: Tabela Hash

Comparações:

```
Tabela Hash foi 4520.53x mais rápido que Busca Sequencial  
Tabela Hash foi 2.27x mais rápido que Busca Binária
```

> Salvando arquivo de resultado...

```
Arquivo gerado: projeto_2_resultado_tabela_hash.txt
```

> Gerando gráfico de desempenho...

```
Gráfico gerado: projeto_2/projeto_2_comparacao_desempenho
```

EXECUÇÃO CONCLUÍDA COM SUCESSO!

```
(.venv) kaori@beatrizkaori-Aspire-A15-51M:~/fpcl-2VA-4$
```

COMPARAÇÃO DE DESEMPENHO

```
Lista Encadeada Simples (Busca Sequencial): 6.1406486980 s  
Lista com Índices (Busca Binária): 0.0028428160 s  
Tabela Hash: 0.0004848401 s
```

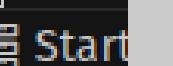
Método mais rápido: Tabela Hash

Comparações:

```
Tabela Hash foi 12665.31x mais rápido que Busca Sequencial  
Tabela Hash foi 5.86x mais rápido que Busca Binária
```

> Salvando arquivo de resultado...

```
Arquivo gerado: projeto_2_resultado_tabela_hash.txt
```



Start

COMPARAÇÃO DE DESEMPENHO

Lista Encadeada Simples (Busca Sequencial): 6.1406486980 s

Lista com Índices (Busca Binária): 0.0028428160 s

Tabela Hash: 0.0004848401 s

Método mais rápido: Tabela Hash

Comparações:

Tabela Hash foi 12665.31x mais rápido que Busca Sequencial

Tabela Hash foi 5.86x mais rápido que Busca Binária

> Salvando arquivo de resultado...

Arquivo gerado: projeto_2_resultado_tabela_hash.txt

Start JSON Server

Pablo Neves (1 da