

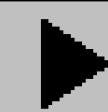
2^a VERIFICAÇÃO DE APRENDIZAGEM

FUNDAMENTOS DE PROBLEMAS COMPUTACIONAIS I

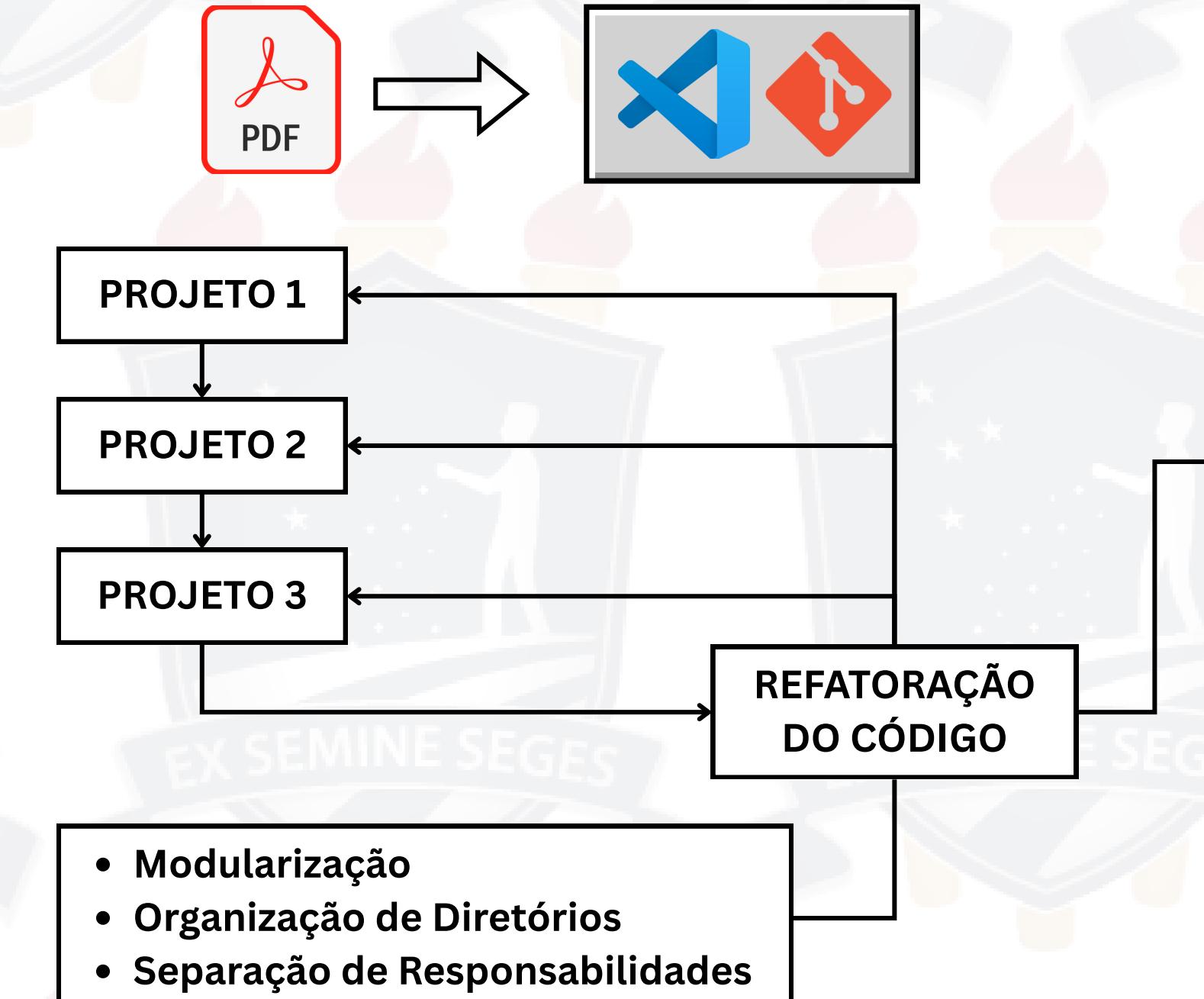
Docente: Prof. Dr. Lucas Cambuim

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

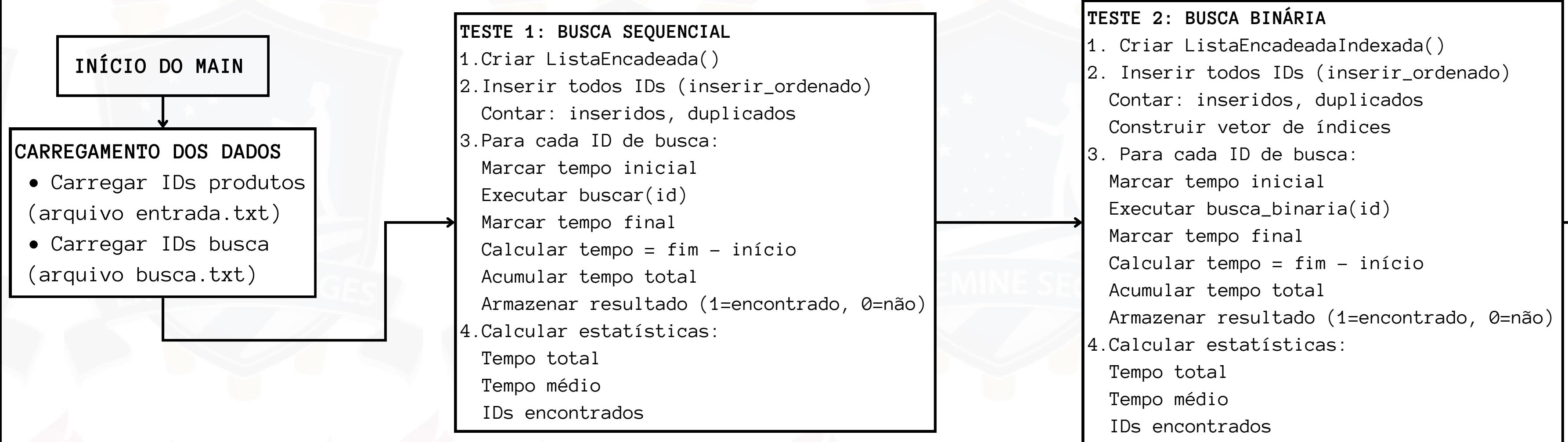
Projeto 2: Comparação de 3 Métodos (Seq. + Bin. + Hash)
Script principal (testes + gráficos)
Dados de entrada (IDs para inserir)
Dados de busca (IDs para procurar)
Resultados da tabela hash
Gráfico comparativo

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

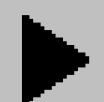
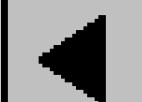
- Gerar resultado_sequencial.txt (1 ou 0 para cada busca)
- Gerar resultado_binaria.txt (1 ou 0 para cada busca)



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



```
estruturas_dados > lista_encadeada_com_indices.py
1 > ....
5
6 > class No: ...
11
12 < class ListaEncadeadaIndexada:
13     # Lista encadeada com vetor de indices
14     # Armazena tuplas no vetor
15 >     def __init__(self, auto_indexar=True):
16         self.ponta = None
17         self.vetor_indices = []
18
19     def inserir_ordenado(self, id_produto):
20         if self.ponta is None:
21             self.ponta = No(id_produto)
22             self.vetor_indices.append((id_produto, self.ponta))
23         else:
24             anterior = self.ponta
25             while anterior.proximo is not None:
26                 anterior = anterior.proximo
27             anterior.proximo = No(id_produto)
28             self.vetor_indices.append((id_produto, self.ponta))
29
30     def _reajustar_indices(self):
31         if self.ponta is not None:
32             self.vetor_indices = []
33             atual = self.ponta
34             while atual is not None:
35                 self.vetor_indices.append((atual.id_produto, atual))
36                 atual = atual.proximo
37
38     def construir_indices(self):
39         # Constrói o vetor de indices após todas as inserções
40         # Deve ser chamado uma única vez após inserir todos os elementos (quando auto_indexar=False)
41         self._reajustar_indices()
42
43     def busca_binaria(self, id_produto):
44         # Realiza busca binária no vetor de indices
45         esquerda = 0
46         direita = len(self.vetor_indices) - 1
47
48         while esquerda <= direita:
49             meio = (esquerda + direita) // 2
50             meio_id = self.vetor_indices[meio][0]
51
52             if meio_id == id_produto:
53                 return 1 # Encontrado
54             elif meio_id < id_produto:
55                 esquerda = meio + 1
56             else:
57                 direita = meio - 1
58
59         return -1 # Não encontrado
60
61     def exibir(self): # Exibe a lista encadeada
62         resultado = []
63         atual = self.ponta
64         while atual is not None:
65             resultado.append(atual.id_produto)
66             atual = atual.proximo
67         print(resultado)
68
69     def exibir_vetor_indices(self):
70         print(self.vetor_indices)
71
72     def obter.todos_ids(self):
73         return [indice[0] for indice in self.vetor_indices]
74
75     def obter.todos_no(self):
76         return [indice[1] for indice in self.vetor_indices]
77
78     def obter.todos_no(self):
79         return [indice[1] for indice in self.vetor_indices]
80
81     def obter.todos_no(self):
82         return [indice[1] for indice in self.vetor_indices]
83
84     def obter.todos_no(self):
85         return [indice[1] for indice in self.vetor_indices]
86
87     def obter.todos_no(self):
88         return [indice[1] for indice in self.vetor_indices]
89
90     def obter.todos_no(self):
91         return [indice[1] for indice in self.vetor_indices]
92
93     def obter.todos_no(self):
94         return [indice[1] for indice in self.vetor_indices]
95
96     def obter.todos_no(self):
97         return [indice[1] for indice in self.vetor_indices]
98
99     def obter.todos_no(self):
100        return [indice[1] for indice in self.vetor_indices]
101
102    def obter.todos_no(self):
103        return [indice[1] for indice in self.vetor_indices]
104
105    def obter.todos_no(self):
106        return [indice[1] for indice in self.vetor_indices]
107
108    def obter.todos_no(self):
109        return [indice[1] for indice in self.vetor_indices]
110
111    def obter.todos_no(self):
112        return [indice[1] for indice in self.vetor_indices]
113
114    def obter.todos_no(self):
115        return [indice[1] for indice in self.vetor_indices]
116
117    def obter.todos_no(self):
118        return [indice[1] for indice in self.vetor_indices]
119
120    def obter.todos_no(self):
121        return [indice[1] for indice in self.vetor_indices]
122
123    def obter.todos_no(self):
124        return [indice[1] for indice in self.vetor_indices]
125
126    def obter.todos_no(self):
127        return [indice[1] for indice in self.vetor_indices]
128
129    def obter.todos_no(self):
130        return [indice[1] for indice in self.vetor_indices]
131
132    def obter.todos_no(self):
133        return [indice[1] for indice in self.vetor_indices]
134
135    def obter.todos_no(self):
136        return [indice[1] for indice in self.vetor_indices]
137
138    def obter.todos_no(self):
139        return [indice[1] for indice in self.vetor_indices]
140
141    def obter.todos_no(self):
142        return [indice[1] for indice in self.vetor_indices]
143
144    def obter.todos_no(self):
145        return [indice[1] for indice in self.vetor_indices]
146
147    def obter.todos_no(self):
148        return [indice[1] for indice in self.vetor_indices]
149
150    def obter.todos_no(self):
151        return [indice[1] for indice in self.vetor_indices]
152
153    def obter.todos_no(self):
154        return [indice[1] for indice in self.vetor_indices]
155
156    def obter.todos_no(self):
157        return [indice[1] for indice in self.vetor_indices]
158
159    def obter.todos_no(self):
160        return [indice[1] for indice in self.vetor_indices]
161
162    def obter.todos_no(self):
163        return [indice[1] for indice in self.vetor_indices]
164
165    def obter.todos_no(self):
166        return [indice[1] for indice in self.vetor_indices]
167
168    def obter.todos_no(self):
169        return [indice[1] for indice in self.vetor_indices]
170
171    def obter.todos_no(self):
172        return [indice[1] for indice in self.vetor_indices]
173
174    def obter.todos_no(self):
175        return [indice[1] for indice in self.vetor_indices]
176
177    def obter.todos_no(self):
178        return [indice[1] for indice in self.vetor_indices]
179
180    def obter.todos_no(self):
181        return [indice[1] for indice in self.vetor_indices]
182
183    def obter.todos_no(self):
184        return [indice[1] for indice in self.vetor_indices]
185
186    def obter.todos_no(self):
187        return [indice[1] for indice in self.vetor_indices]
188
189    def obter.todos_no(self):
190        return [indice[1] for indice in self.vetor_indices]
191
192    def obter.todos_no(self):
193        return [indice[1] for indice in self.vetor_indices]
194
195    def obter.todos_no(self):
196        return [indice[1] for indice in self.vetor_indices]
197
198    def obter.todos_no(self):
199        return [indice[1] for indice in self.vetor_indices]
200
201    def obter.todos_no(self):
202        return [indice[1] for indice in self.vetor_indices]
203
204    def obter.todos_no(self):
205        return [indice[1] for indice in self.vetor_indices]
206
207    def obter.todos_no(self):
208        return [indice[1] for indice in self.vetor_indices]
209
210    def obter.todos_no(self):
211        return [indice[1] for indice in self.vetor_indices]
212
213    def obter.todos_no(self):
214        return [indice[1] for indice in self.vetor_indices]
215
216    def obter.todos_no(self):
217        return [indice[1] for indice in self.vetor_indices]
218
219    def obter.todos_no(self):
220        return [indice[1] for indice in self.vetor_indices]
221
222    def obter.todos_no(self):
223        return [indice[1] for indice in self.vetor_indices]
224
225    def obter.todos_no(self):
226        return [indice[1] for indice in self.vetor_indices]
227
228    def obter.todos_no(self):
229        return [indice[1] for indice in self.vetor_indices]
230
231    def obter.todos_no(self):
232        return [indice[1] for indice in self.vetor_indices]
233
234    def obter.todos_no(self):
235        return [indice[1] for indice in self.vetor_indices]
236
237    def obter.todos_no(self):
238        return [indice[1] for indice in self.vetor_indices]
239
240    def obter.todos_no(self):
241        return [indice[1] for indice in self.vetor_indices]
242
243    def obter.todos_no(self):
244        return [indice[1] for indice in self.vetor_indices]
245
246    def obter.todos_no(self):
247        return [indice[1] for indice in self.vetor_indices]
248
249    def obter.todos_no(self):
250        return [indice[1] for indice in self.vetor_indices]
251
252    def obter.todos_no(self):
253        return [indice[1] for indice in self.vetor_indices]
254
255    def obter.todos_no(self):
256        return [indice[1] for indice in self.vetor_indices]
257
258    def obter.todos_no(self):
259        return [indice[1] for indice in self.vetor_indices]
260
261    def obter.todos_no(self):
262        return [indice[1] for indice in self.vetor_indices]
263
264    def obter.todos_no(self):
265        return [indice[1] for indice in self.vetor_indices]
266
267    def obter.todos_no(self):
268        return [indice[1] for indice in self.vetor_indices]
269
270    def obter.todos_no(self):
271        return [indice[1] for indice in self.vetor_indices]
272
273    def obter.todos_no(self):
274        return [indice[1] for indice in self.vetor_indices]
275
276    def obter.todos_no(self):
277        return [indice[1] for indice in self.vetor_indices]
278
279    def obter.todos_no(self):
280        return [indice[1] for indice in self.vetor_indices]
281
282    def obter.todos_no(self):
283        return [indice[1] for indice in self.vetor_indices]
284
285    def obter.todos_no(self):
286        return [indice[1] for indice in self.vetor_indices]
287
288    def obter.todos_no(self):
289        return [indice[1] for indice in self.vetor_indices]
290
291    def obter.todos_no(self):
292        return [indice[1] for indice in self.vetor_indices]
293
294    def obter.todos_no(self):
295        return [indice[1] for indice in self.vetor_indices]
296
297    def obter.todos_no(self):
298        return [indice[1] for indice in self.vetor_indices]
299
299 >
```

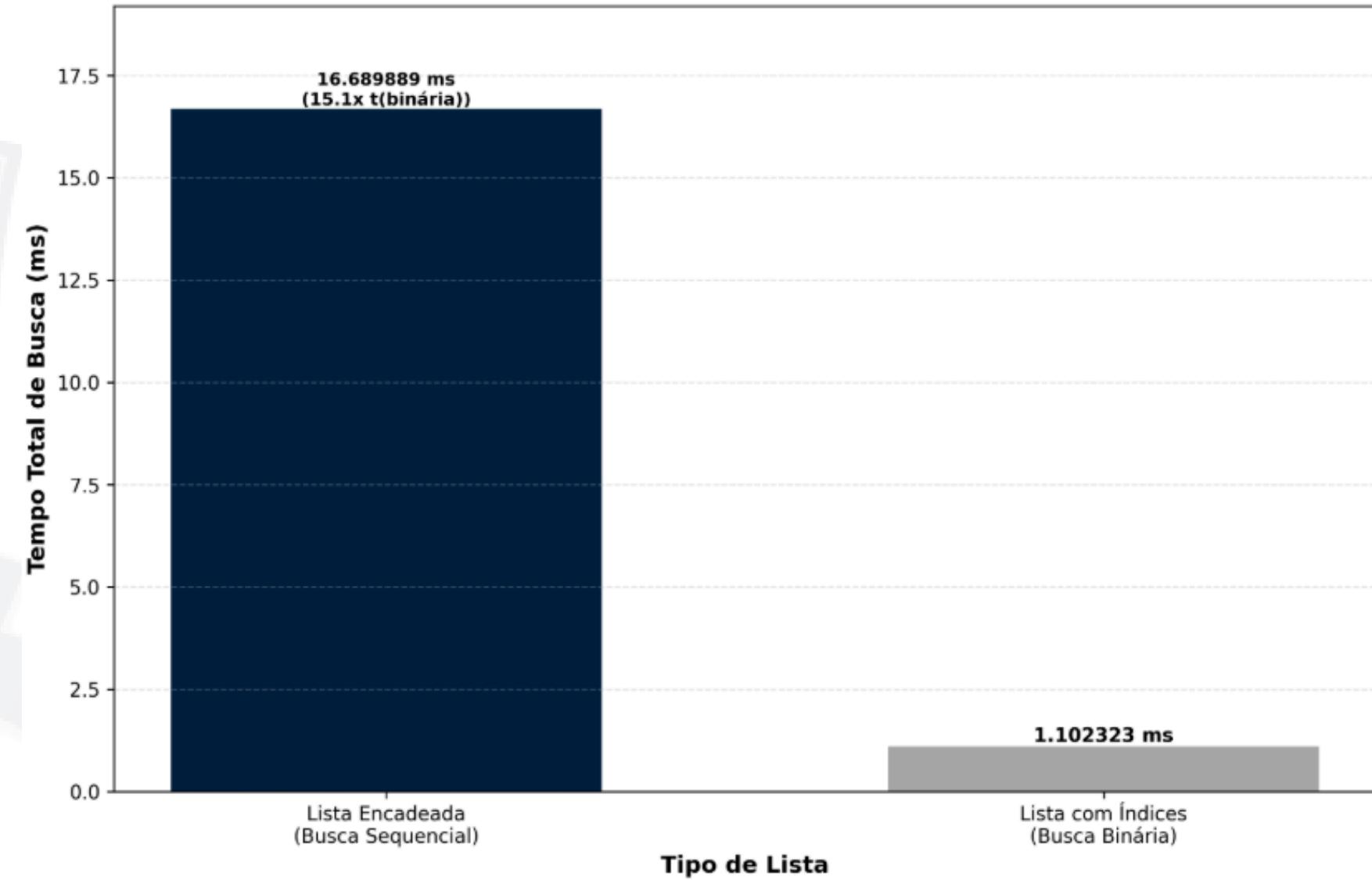




Comparação de desempenho

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

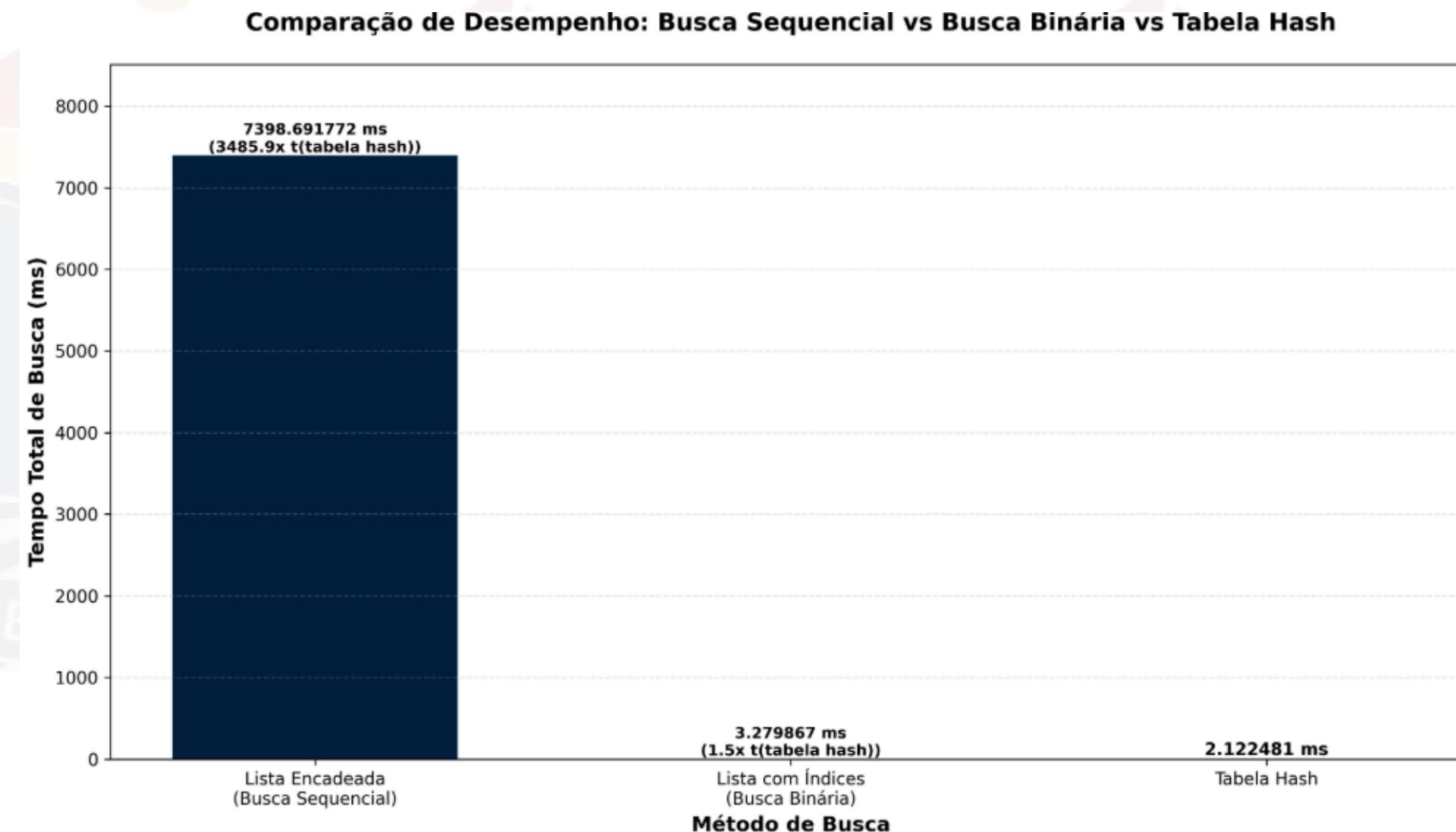
    return -1 # ID não encontrado
```





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

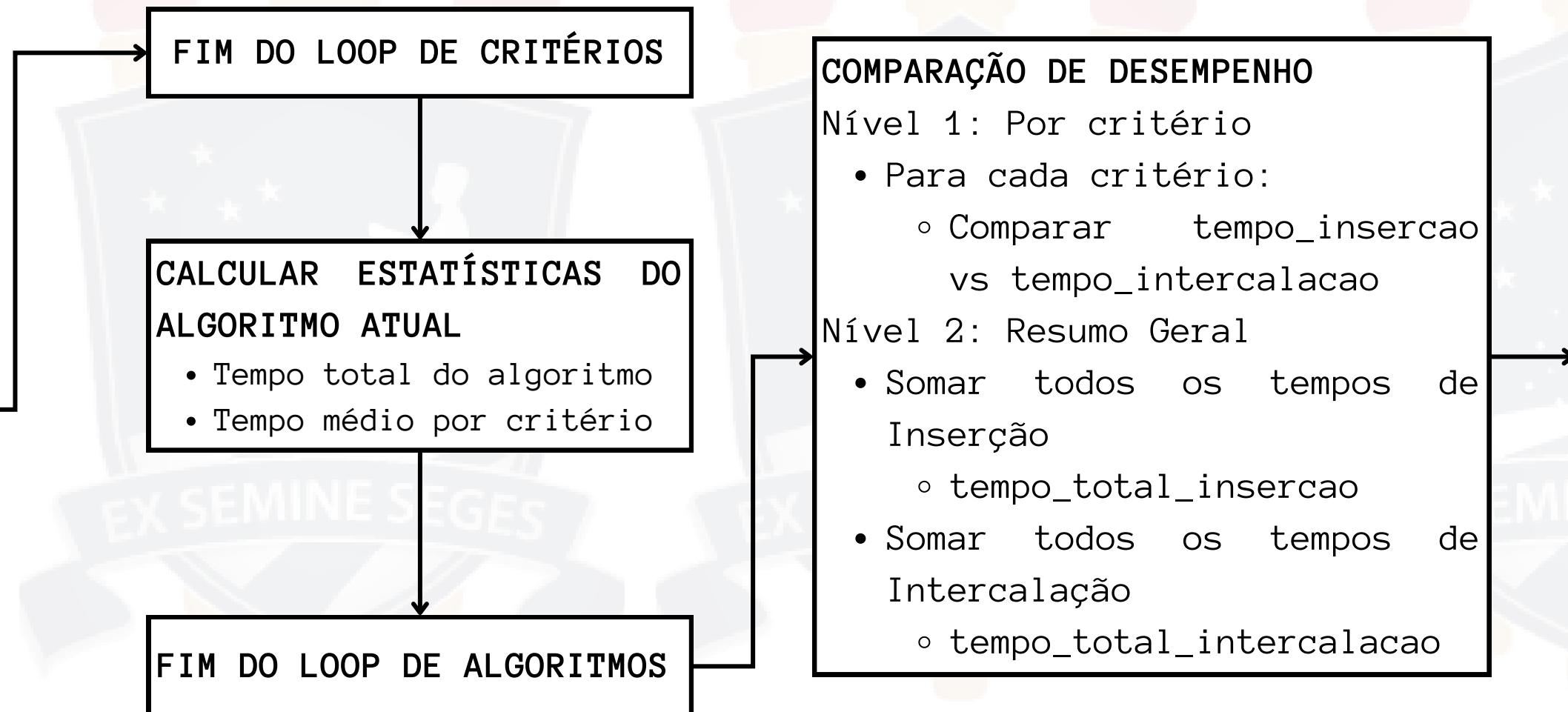
- Implementar e avaliar a eficiência e o comportamento de diferentes algoritmos de ordenação (Insertion Sort e Merge Sort) aplicados a um catálogo de produtos real, desenvolvendo a ordenação flexível dos dados por múltiplos critérios.



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

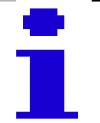
Tipo: Barras agrupadas
 Eixo X: Critérios
 Eixo Y: Tempo (ms)
 2 barras por critérios
 Adicionar valores e fator nas barras
 Salvar: comparacao_por_criterio.png

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

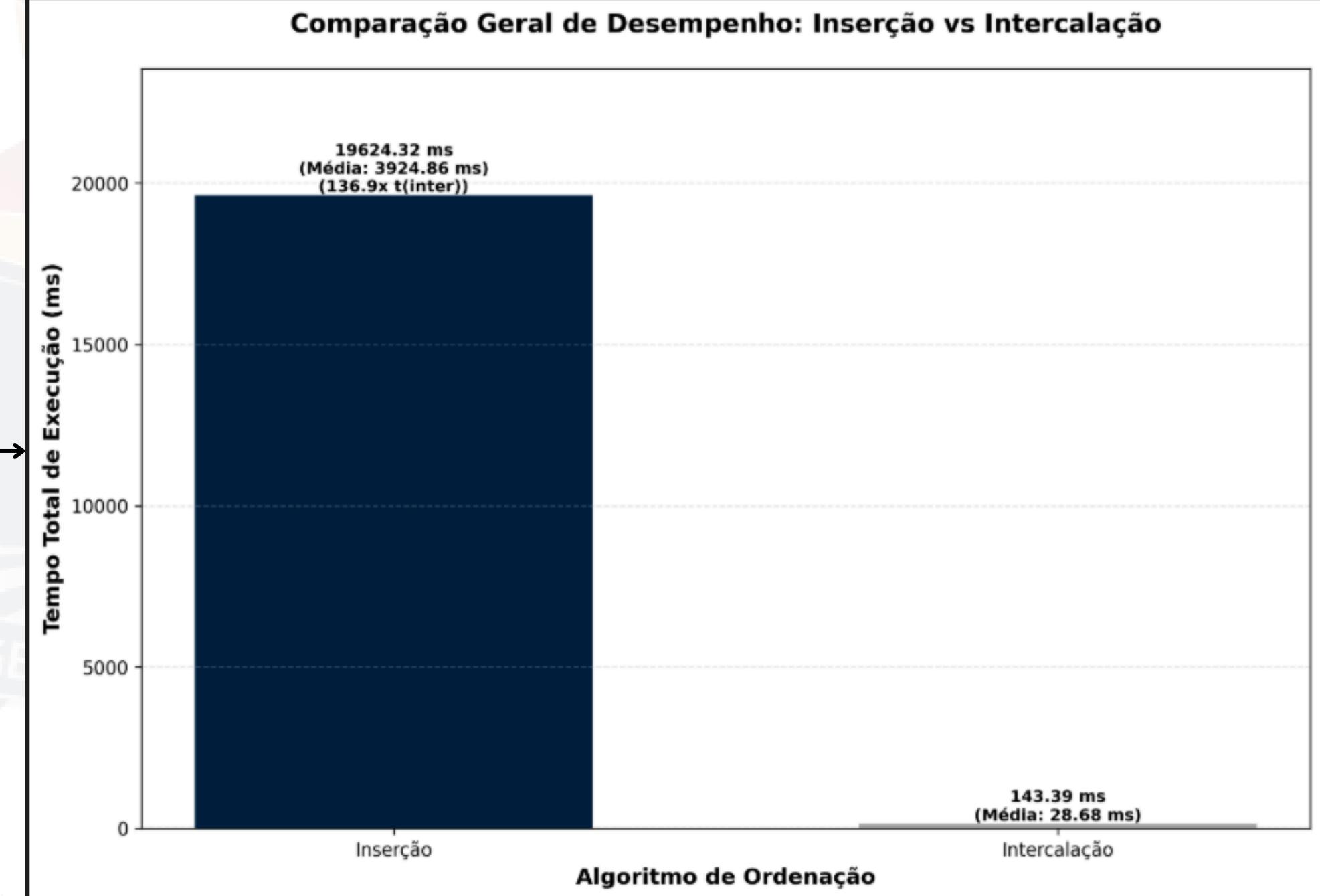
- Complexidade: $O(n^2)$
- Método: Iterativo

MERGE SORT

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

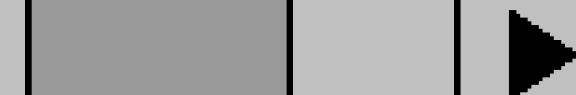
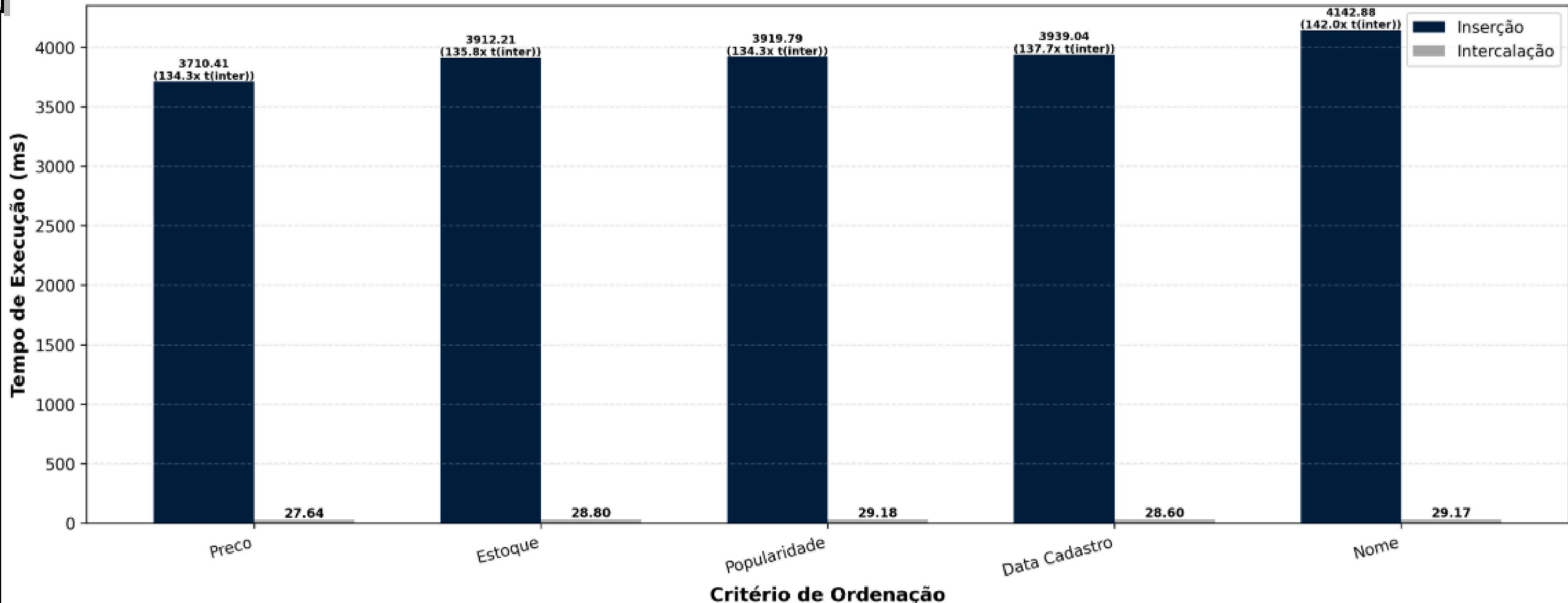
RESULTADO GERAL

- Intercalação foi 136x mais rápido



i

Comparação de Desempenho por Critério de Ordenação



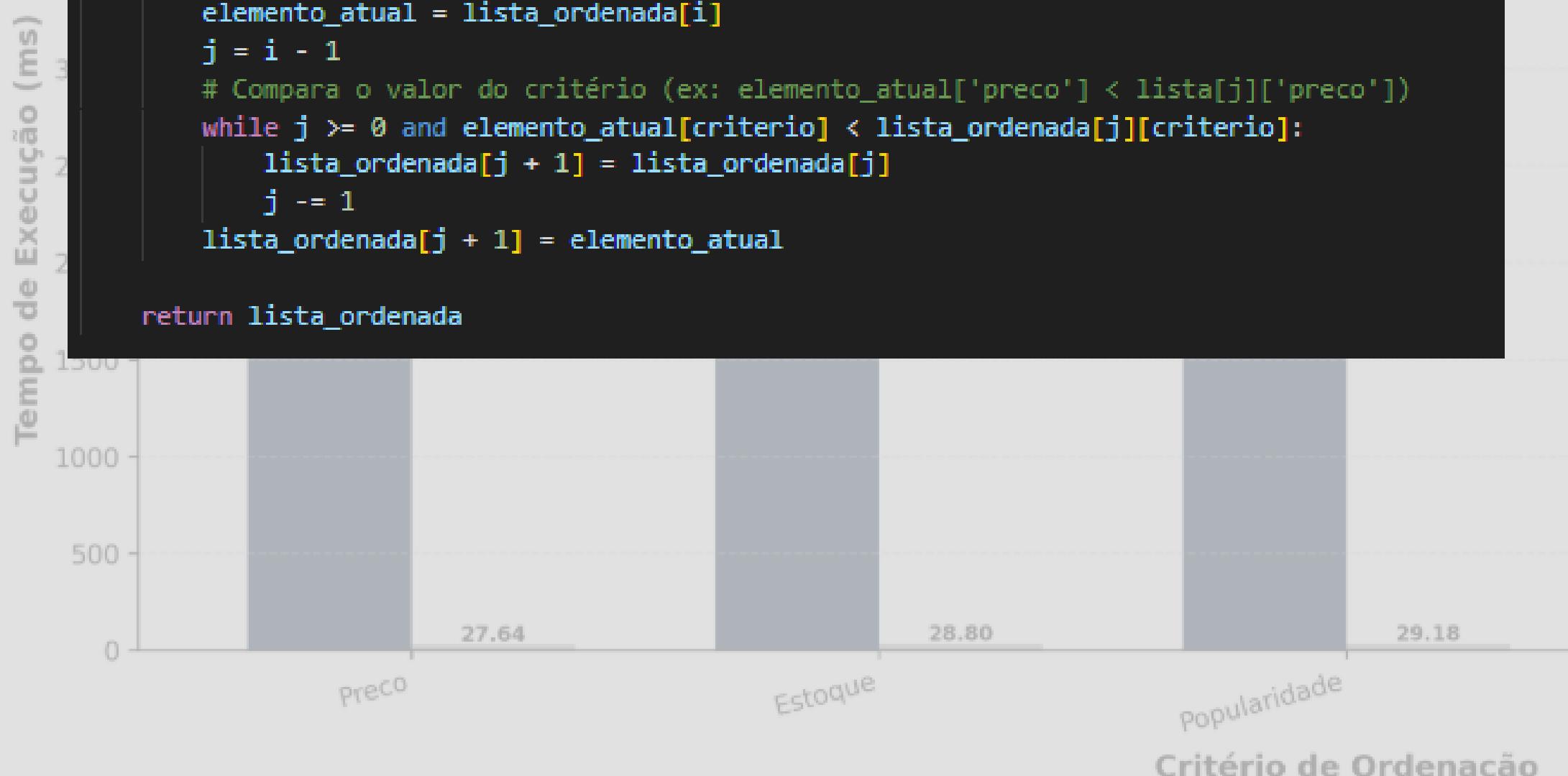


INSERÇÃO

```
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
```



INTERCALAÇÃO

```
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!

