Etapa 9 — Busca e Ordenação (Searching and Sorting)

Trilha: Lógica de Programação

Introdução

- **Busca (Search)** → encontrar um valor dentro de um conjunto de dados.
- **Ordenação (Sorting)** → organizar os dados em uma sequência lógica (crescente ou decrescente).
- São operações **fundamentais** em algoritmos e estruturas de dados.
- “Buscar e ordenar são as bases para entender o funcionamento de bancos de dados, listas e algoritmos avançados.”

Tipos de Busca

1. **Busca Sequencial (Linear Search)**
   - Verifica cada elemento da lista até encontrar o alvo.
   - Simples, mas pouco eficiente para listas grandes.
2. **Busca Binária (Binary Search)**
   - Requer lista **ordenada**.
   - Divide o conjunto ao meio a cada passo, reduzindo o número de comparações.

Busca Sequencial (Linear Search)

- **Pseudocódigo:**

```
funcao busca_linear(vetor, valor)
    para i de 1 até tamanho(vetor) faça
        se vetor[i] = valor entao
            retorna i
        fimse
    fimpara
    retorna -1 // não encontrado
fimfuncao
```

- **Vantagem:** simples de implementar.
- **Desvantagem:** ineficiente em listas grandes (O(n)).

Busca Binária (Binary Search)

- **Pseudocódigo:**

```
funcao busca_binaria(vetor, valor)
        inicio ← 1
        fim ← tamanho(vetor)
        enquanto inicio <= fim faça
                meio ← (inicio + fim) / 2
                se vetor[meio] = valor entao
                        retorna meio
                senao se vetor[meio] < valor entao
                        inicio ← meio + 1
                senao
                        fim ← meio - 1
                fimse
        fimenquanto
        retorna -1
fimfuncao
```

- **Complexidade:** O(log n)
- **Mais rápida**, porém exige lista **ordenada**.

Tipos de Ordenação

- **Bubble Sort** — compara pares de elementos vizinhos e troca-os se estiverem fora de ordem.
- **Selection Sort** — encontra o menor elemento e o move para o início.
- **Insertion Sort** — insere elementos ordenadamente em uma lista parcialmente ordenada.
- **Merge Sort / Quick Sort** — algoritmos mais eficientes, baseados em **divisão e conquista (divide and conquer).**

Bubble Sort

- **Pseudocódigo:**

```
procedimento bubble_sort(vetor)
    para i de 1 até tamanho(vetor) - 1 faça
        para j de 1 até tamanho(vetor) - i faça
            se vetor[j] > vetor[j + 1] entao
                troque(vetor[j], vetor[j + 1])
            fimse
        fimpara
    fimpara
Fimprocedimento
```

- **Fácil de implementar**, mas **ineficiente** para grandes volumes ($O(n^2)$).

Selection Sort

- **Pseudocódigo:**

```
procedimento selection_sort(vetor)
        para i de 1 até tamanho(vetor) - 1 faça
                min ← i
                para j de i + 1 até tamanho(vetor) faça
                se vetor[j] < vetor[min] entao
                                min ← j
                fimse
                fimpara
                troque(vetor[i], vetor[min])
        fimpara
fimprocedimento
```

- **Vantagem:** faz menos trocas que o Bubble Sort.
- **Desvantagem:** ainda é $O(n^2)$.

Insertion Sort

- **Pseudocódigo:**

```
procedimento insertion_sort(vetor)
    para i de 2 até tamanho(vetor) faça
        chave ← vetor[i]
        j ← i - 1
        enquanto j > 0 e vetor[j] > chave faça
            vetor[j + 1] ← vetor[j]
            j ← j - 1
        fimenquanto
        vetor[j + 1] ← chave
    fimpara
fimprocedimento
```

- **Bom para listas pequenas** ou quase ordenadas.

Merge Sort (Divisão e Conquista)

- **Ideia:**
  - Divide o vetor em metades até ter partes pequenas.
  - Ordena e junta novamente (*merge*).
- Complexidade: **O(n log n)**

Muito eficiente para grandes volumes.

Quick Sort

- **Ideia:**
  - Escolhe um **pivô (pivot)**.
  - Separa valores menores e maiores que o pivô.
  - Chama recursivamente para as duas partes.
- **Complexidade média:** $O(n \log n)$
- **Pior caso:** $O(n^2)$ (quando os pivôs são mal escolhidos)

Comparativo

Algoritmo	**Complexidade**	**Estável**	**Estrutura**
Bubble Sort	$O(n^2)$	Sim	Simples
Selection Sort	$O(n^2)$	Não	Simples
Insertion Sort	$O(n^2)$	Sim	Simples
Merge Sort	$O(n \log n)$	Sim	Recursivo
Quick Sort	$O(n \log n)$	Não	Recursivo

Aplicações Práticas

- **Busca binária** em listas de nomes, IDs ou produtos.
- **Ordenação** em cadastros e relatórios.
- Base para algoritmos de **pesquisa de dados e bancos de dados**.
- Usados em **jogos**, **IA** e **ciência de dados**.

Boas Práticas

- Ordene os dados antes de aplicar **busca binária**.
- Escolha o algoritmo de ordenação conforme o **tamanho dos dados**.
- Prefira **Merge** ou **Quick Sort** para grandes volumes.
- Teste e compare tempos de execução.

Conclusão

- **Busca e ordenação** são operações centrais em lógica e estruturas de dados.
- Conhecer suas diferenças e aplicações melhora a **eficiência** dos algoritmos.
- A escolha correta do método depende do **tamanho e organização** dos dados.
- **Resumo:**

“Buscar é encontrar; ordenar é preparar o caminho para encontrar mais rápido.”