**Fundamentos de Java**

* + Sintaxe Básica:
    - Tipos de dados (int, char, float, boolean), variáveis, operadores, controle de fluxo (if, switch, for, while, do-while).
  + Classes e Objetos:
    - Definição de classes, criação de objetos, métodos, construtores, encapsulamento.
  + Herança:
    - Superclasse, subclasse, extends keyword, método super().
  + Polimorfismo:
    - Sobrecarga de métodos, sobrescrita de métodos, classes abstratas, interfaces.
  + Encapsulamento:
    - Modificadores de acesso (private, protected, public), getters e setters.
  + Pacotes e Importação:
    - Organização de classes em pacotes, uso da palavra-chave import.

**Coleções**

* + **List**: Uma lista é uma coleção ordenada que pode conter elementos duplicados. A posição dos elementos é mantida.
    - **ArrayList**: Usa um array dinâmico internamente. Bom para acesso aleatório (get) rápido, mas operações de inserção e remoção são mais lentas (exceto no fim da lista).

**A computer screen with colorful text

Description automatically generated with medium confidence**

* + - * **Use ArrayList quando a aplicação requer acesso rápido a elementos por índice.**
    - **LinkedList**: Usa uma lista duplamente ligada internamente. Bom para inserções e remoções rápidas em ambas as extremidades da lista, mas o acesso aleatório é mais lento.

A black background with white text

Description automatically generated

* + - * Use **LinkedList** quando a aplicação requer muitas inserções ou remoções de elementos nas extremidades da lista
  + **Set**: Um conjunto é uma coleção que não permite elementos duplicados
    - **HashSet**: Usa uma tabela hash para armazenar elementos. Não garante nenhuma ordem dos elementos. Operações são geralmente em tempo constante.

A black background with white text

Description automatically generated

* + - * **Use HashSet para operações rápidas (inserção, remoção) e não necessita de ordem.**
    - **TreeSet:** Usa uma árvore de busca binária (árvore Red-Black). Mantém os elementos ordenados. Operações são em tempo logarítmico

**A black background with white text

Description automatically generated**

* + - * **Use TreeSet quando a ordem dos elementos for importante.**
    - **LinkedHashSet: Mantém uma tabela hash e uma lista duplamente ligada. Mantém a ordem de inserção.**

**A black background with white text

Description automatically generated**

* + - * **Use LinkedHashSet quando a ordem de inserção precisar ser preservada.**
  + **Map**: Um mapa é uma coleção que mapeia chaves para valores. Não permite chaves duplicadas.
    - **HashMap**: Usa uma tabela hash para armazenar pares chave-valor. Não garante nenhuma ordem das chaves.

A black background with white text

Description automatically generated

* + - * Use HashMap para operações rápidas e não necessita de ordem nas chaves.
    - **TreeMap**: Usa uma árvore de busca binária (árvore Red-Black). Mantém as chaves ordenadas

A black background with white text

Description automatically generated

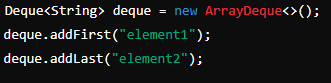
* + - * Use TreeMap para manter as chaves ordenadas.
    - **LinkedHashMap**: Mantém uma tabela hash e uma lista duplamente ligada. Mantém a ordem de inserção das chaves.



* + - * Use LinkedHashMap quando a ordem de inserção das chaves for importante.
  + **Queue:** Uma fila é uma coleção usada para armazenar múltiplos elementos antes de serem processados. Normalmente segue a ordem FIFO (First-In-First-Out).

****

* + - * **Use Queue quando precisar de uma estrutura FIFO.**
  + **Deque (Double-Ended Queue) é uma fila onde elementos podem ser inseridos e removidos de ambas as extremidades.**

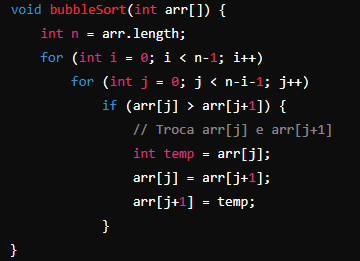
****

* + - * **Use Deque quando precisar adicionar e remover elementos de ambas as extremidades da fila.**

**Algoritmos de ordenação**

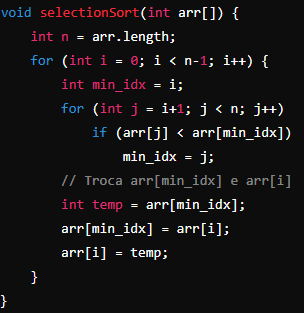
* **Bubble Sort**
* É um algoritmo de ordenação simples que compara cada par de elementos adjacentes e troca se estiverem na ordem errada.
* Ideal para listas pequenas ou quase ordenadas.
* Não é eficiente para listas grandes devido à sua complexidade de tempo O(n²).
* **Passo a Passo:**

1. Percorre a lista, comparando elementos adjacentes.
2. Se o primeiro elemento é maior que o segundo, troca-os.
3. Repetir até que a lista esteja ordenada.



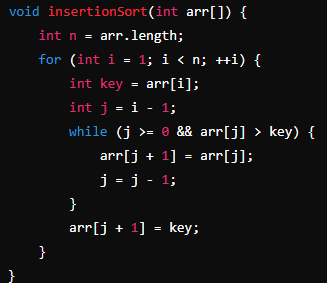
* **Selection Sort**
  + **Encontra o menor elemento e coloca-o no início da lista.**
  + **Repete o processo para o restante da lista.**
  + **É eficiente para listas pequenas, mas tem complexidade de tempo O(n²).**
  + **Passo a Passo:**

1. Percorre a lista para encontrar o menor elemento.
2. Troca-o com o primeiro elemento.
3. Repete o processo para o subarray restante.



* **Selection Sort**
  + **Constrói a lista ordenada um elemento de cada vez.**
  + **Ideal para listas pequenas ou quase ordenadas.**
  + **Complexidade de tempo O(n²), mas mais eficiente que Bubble Sort e Selection Sort para listas pequenas.**
  + **Passo a Passo:**

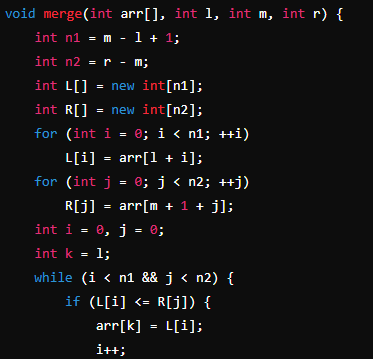
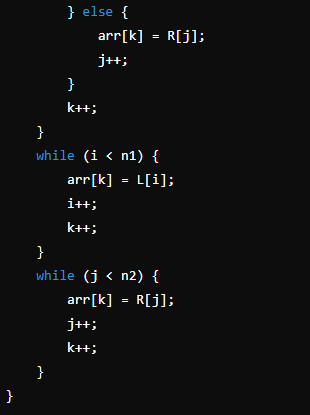
1. Considera o primeiro elemento como uma lista ordenada.
2. Insere o próximo elemento na posição correta da lista ordenada.
3. Repete o processo para todos os elementos.

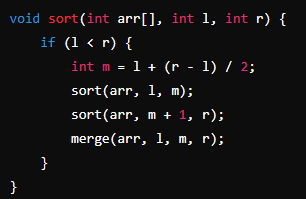


* **Merge Sort**
  + **Divide a lista em duas metades, ordena cada metade e depois combina-as.**
  + **Utiliza a abordagem "dividir e conquistar".**
  + **Complexidade de tempo O(n log n), eficiente para listas grandes.**
  + **Passo a Passo:**

1. Divide a lista ao meio até que cada sublista tenha um único elemento.
2. Combina sublistas adjacentes para formar listas ordenadas.
3. Repete até que toda a lista esteja ordenada.

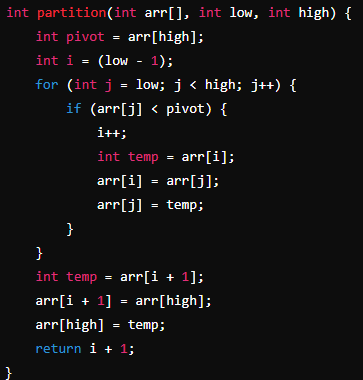
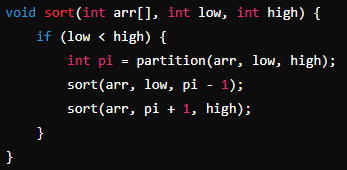
( Codigo no i++ continua no } else{ )



* **Quick Sort**
  + **Seleciona um "pivô" e particiona a lista em duas partes: menores que o pivô e maiores que o pivô.**
  + **Ordena recursivamente as duas partes.**
  + **Complexidade de tempo média O(n log n), mas pode ser O(n²) no pior caso se o pivô não for escolhido bem.**
  + **Passo a Passo:**

1. Seleciona um elemento como pivô.
2. Reorganiza a lista de modo que todos os elementos menores que o pivô fiquem à esquerda e todos os maiores, à direita.
3. Aplica recursivamente o mesmo processo às sublistas à esquerda e à direita do pivô

**Quando Usar Cada Algoritmo**

* **Bubble Sort: Use apenas para listas muito pequenas ou como uma introdução à ordenação.**
* **Selection Sort: Útil para listas pequenas e quando a troca de elementos tem um custo relativamente baixo.**
* **Insertion Sort: Ideal para listas pequenas ou quase ordenadas; é mais eficiente que Bubble Sort e Selection Sort nessas condições.**
* **Merge Sort: Use para listas grandes onde a estabilidade da ordenação é importante. Bom desempenho garantido de O(n log n).**
* **Quick Sort: Geralmente o mais eficiente para listas grandes na prática. Escolha do pivô é crucial para evitar o pior caso O(n²).**

**Fundamentos de SQL**

**SQL (Structured Query Language) é uma linguagem padrão para gerenciamento e manipulação de bases de dados relacionais.**

**SGBD: Sistema de Gerenciamento de Banco de Dados como MySQL, PostgreSQL, SQL Server, Oracle.**

**Comandos SQL Básicos**

* **DDL (Data Definition Language):**
  + **CREATE: Cria objetos no banco de dados (tabelas, índices, etc.).**
  + **ALTER: Modifica a estrutura de objetos existentes.**
  + **DROP: Remove objetos do banco de dados.**

**A computer screen shot of text

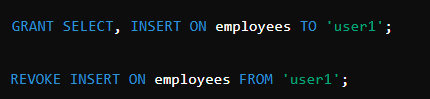
Description automatically generated**

* **DML (Data Manipulation Language):**
  + **SELECT: Consulta dados.**
  + **INSERT: Insere novos dados.**
  + **UPDATE: Atualiza dados existentes.**
  + **DELETE: Remove dados.**

**A computer code with white text

Description automatically generated**

* **DCL (Data Control Language):**
* **GRANT: Concede permissões a usuários.**
* **REVOKE: Revoga permissões de usuários.**

****

* **TCL (Transaction Control Language):**
  + **COMMIT:** Confirma uma transação, salvando todas as alterações feitas no banco de dados permanentemente**.**
  + **ROLLBACK:** Reverte uma transação, desfazendo todas as alterações feitas no banco de dados desde o último **COMMIT**.**.**
  + **SAVEPOINT:** Define um ponto de salvamento dentro de uma transação, permitindo que você possa reverter parte da transação**.**

A computer screen with text

Description automatically generated

**Consultas SQL Avançadas**

* **Funções de Agregação:**
  + **COUNT**: Conta o número de linhas em um conjunto de resultados
  + **SUM**: Retorna a soma total de um conjunto de valores
  + **AVG**: Calcula a média (média aritmética) de um conjunto de valores
  + **MAX**: Retorna o maior valor em um conjunto de valores
  + **MIN**: Retorna o menor valor em um conjunto de valores

A black background with white text

Description automatically generated

* **Cláusulas:**
  + **WHERE**: Filtra registos.
  + **GROUP** **BY**: Agrupa registos.
  + **HAVING**: Filtra grupos.
  + **ORDER** **BY**: Ordena registos.

A group of text on a black background

Description automatically generated

* **Joins:**
  + **INNER JOIN:** Retorna apenas as linhas que têm correspondência em ambas as tabelas**.**
    - **Uso**: Quando você precisa de registos que existem em ambas as tabelas.
  + **LEFT JOIN:** Retorna todas as linhas da tabela esquerda (a primeira tabela mencionada), e as linhas correspondentes da tabela direita. Se não houver correspondência, os resultados da tabela direita serão null
    - **Uso**: Quando você precisa de todos os registos da tabela esquerda, independentemente de terem correspondência na tabela direita.
  + **RIGHT JOIN:** Retorna todas as linhas da tabela direita (a segunda tabela mencionada), e as linhas correspondentes da tabela esquerda. Se não houver correspondência, os resultados da tabela esquerda serão NULL.
    - **Uso**: Quando você precisa de todos os registos da tabela direita, independentemente de terem correspondência na tabela esquerda.
  + **FULL OUTER JOIN:** retorna todas as linhas quando há uma correspondência em uma das tabelas. Isso inclui linhas que não têm correspondência na tabela esquerda ou na tabela direita.
    - **Uso**: Quando você precisa de todos os registos de ambas as tabelas, independentemente de terem correspondência.
  + **CROSS JOIN:** Retorna o produto cartesiano das duas tabelas, ou seja, combina cada linha da primeira tabela com cada linha da segunda tabela.
    - **Uso**: Raramente usado sozinho, mas pode ser útil em determinadas situações de análise de dados.
  + **SELF JOIN**: Um join de uma tabela consigo mesma. Útil para encontrar relações hierárquicas ou outras associações dentro de uma única tabela.
    - **Uso:** Quando você precisa relacionar registos dentro da mesma tabela**.**