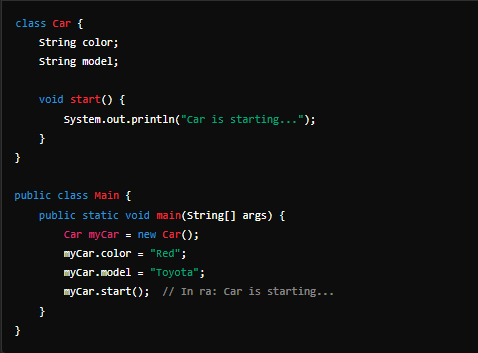
# **Object Oriented Programming**

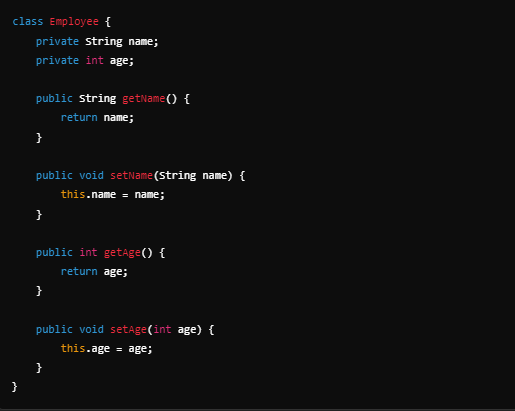
### 1. **Lớp (Class) và Đối tượng (Object)**

* **Lớp (Class)** là một bản thiết kế mô tả các đặc điểm và hành vi chung của đối tượng. Một lớp bao gồm các thuộc tính (biến) và phương thức (hàm).
* **Đối tượng (Object)** là một thể hiện của lớp. Khi lớp được định nghĩa, ta có thể tạo ra nhiều đối tượng từ lớp đó.
* Ví dụ:



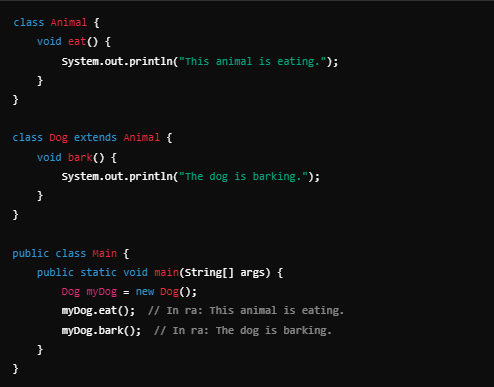
### 2. **Đóng gói (Encapsulation)**

* Đóng gói là việc nhóm các thuộc tính và phương thức có liên quan vào trong một lớp. Các thuộc tính này thường được ẩn (private) và chỉ có thể truy cập thông qua các phương thức getter và setter.
* Điều này giúp bảo vệ dữ liệu khỏi việc bị sửa đổi trực tiếp từ bên ngoài và tạo ra một giao diện rõ ràng để tương tác với lớp.
* Ví dụ:



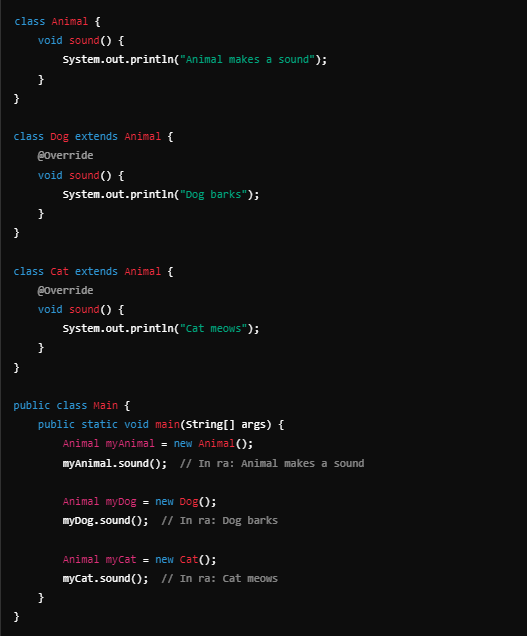
### 3. **Kế thừa (Inheritance)**

* Kế thừa cho phép một lớp con (subclass) kế thừa các đặc điểm và hành vi từ lớp cha (superclass). Điều này giúp tái sử dụng mã nguồn và dễ dàng mở rộng các chức năng.
* Lớp con có thể ghi đè (override) các phương thức của lớp cha để thay đổi hành vi.
* Ví dụ:



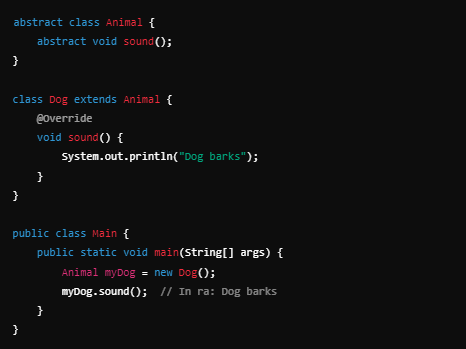
### 4. **Đa hình (Polymorphism)**

* Đa hình cho phép các đối tượng khác nhau có thể có hành vi khác nhau khi gọi cùng một phương thức. Đa hình có thể được thực hiện thông qua:
  + **Nạp chồng phương thức (Method Overloading):** Khi cùng tên phương thức nhưng khác tham số.
  + **Ghi đè phương thức (Method Overriding):** Khi lớp con ghi đè lại phương thức của lớp cha.
* Ví dụ:



### 5. **Trừu tượng (Abstraction)**

* Trừu tượng là việc ẩn chi tiết cài đặt và chỉ cung cấp các phương thức cần thiết cho người dùng. Điều này được thực hiện thông qua các lớp trừu tượng (abstract classes) hoặc giao diện (interfaces).
* Các lớp trừu tượng không thể tạo đối tượng trực tiếp, nhưng các lớp con phải cài đặt các phương thức trừu tượng.
* Ví dụ:



# **Cấu trúc dữ liệu và giải thuật**

## A.Cấu trúc dữ liệu

### 1.Mảng (Array)

Mảng là một cấu trúc dữ liệu cơ bản, trong đó các phần tử có cùng kiểu dữ liệu được lưu trữ liên tiếp trong bộ nhớ. Mảng có chỉ số (index) giúp truy xuất các phần tử nhanh chóng.

* **Ưu điểm:**
  + Truy cập ngẫu nhiên (random access) tới bất kỳ phần tử nào.
  + Quản lý bộ nhớ đơn giản và hiệu quả.
* **Nhược điểm:**
  + Kích thước mảng phải được xác định trước khi sử dụng (cứng nhắc).
  + Chèn hoặc xóa phần tử vào giữa mảng có thể tốn thời gian (O(n)).
* **Ứng dụng:** Dùng trong việc lưu trữ dữ liệu liên tục như danh sách các số, bảng dữ liệu.

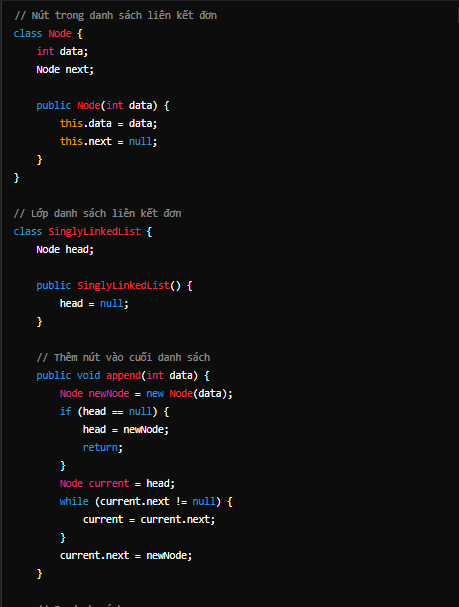
### 2.Singly Linked List

1. Danh sách liên kết đơn là một tập hợp các phần tử, được gọi là nút (node), được liên kết với nhau theo một thứ tự tuyến tính. Mỗi nút chứa hai thành phần:

* **Dữ liệu (Data)**: Lưu trữ giá trị của phần tử.
* **Con trỏ (Next)**: Lưu trữ địa chỉ của nút tiếp theo trong danh sách.

Nút cuối cùng trong danh sách có con trỏ Next trỏ đến NULL (hoặc null trong Java), biểu thị kết thúc danh sách.

* **Ưu điểm**:
  + Kích thước động: Có thể dễ dàng thêm hoặc xóa nút mà không cần phải xác định kích thước ban đầu.
  + Tiết kiệm bộ nhớ: Chỉ sử dụng bộ nhớ cần thiết cho các nút hiện có.
  + Chèn và xóa hiệu quả: Việc chèn và xóa nút ở giữa danh sách có độ phức tạp O(1) nếu đã biết vị trí.
* **Nhược điểm**:
  + Truy cập ngẫu nhiên chậm: Để truy cập một nút ở vị trí bất kỳ, cần phải duyệt từ đầu danh sách (độ phức tạp O(n)).
  + Tốn thêm bộ nhớ cho con trỏ: Mỗi nút cần thêm bộ nhớ để lưu trữ con trỏ.



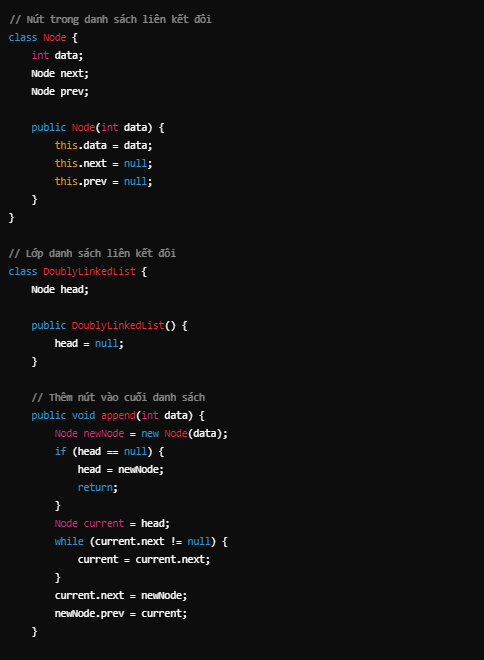
### 3.Doubly Linked List

Danh sách liên kết đôi tương tự như danh sách liên kết đơn, nhưng mỗi nút trong danh sách liên kết đôi chứa ba thành phần:

* **Dữ liệu (Data)**: Lưu trữ giá trị của phần tử.
* **Con trỏ Next**: Trỏ đến nút tiếp theo trong danh sách.
* **Con trỏ Prev (Previous)**: Trỏ đến nút trước nút hiện tại.

Nút đầu tiên trong danh sách có con trỏ Prev là NULL, và nút cuối cùng có con trỏ Next là NULL.

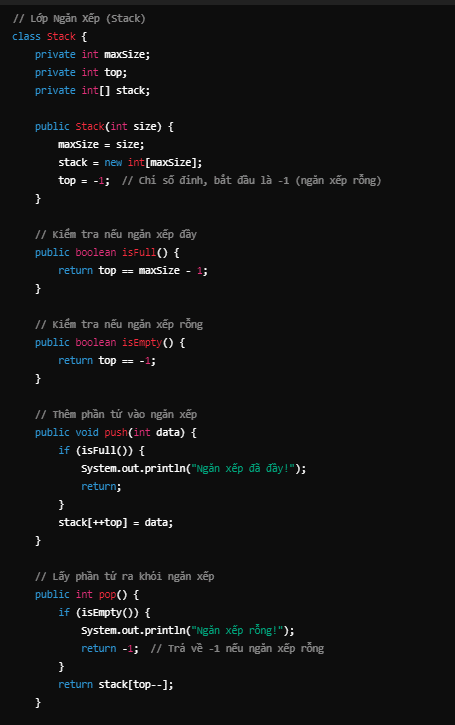
* **Ưu điểm**:
  + Duyệt theo cả hai chiều: Cho phép duyệt danh sách theo cả chiều xuôi và chiều ngược, giúp thực hiện một số thao tác hiệu quả hơn.
  + Xóa nút dễ dàng hơn: Không cần phải duyệt tìm nút trước nút cần xóa nếu đã có con trỏ đến nút đó.
* **Nhược điểm**:
  + Tốn thêm bộ nhớ: Mỗi nút cần thêm một con trỏ Prev, do đó tốn nhiều bộ nhớ hơn so với danh sách liên kết đơn.
  + Các thao tác phức tạp hơn: Việc thêm và xóa nút phức tạp hơn một chút do cần cập nhật cả hai con trỏ.



### 4.Stack

Ngăn xếp là một cấu trúc dữ liệu theo nguyên lý "LIFO" (Last In, First Out). Phần tử được thêm vào và loại bỏ từ cùng một đầu gọi là "đỉnh" của ngăn xếp.

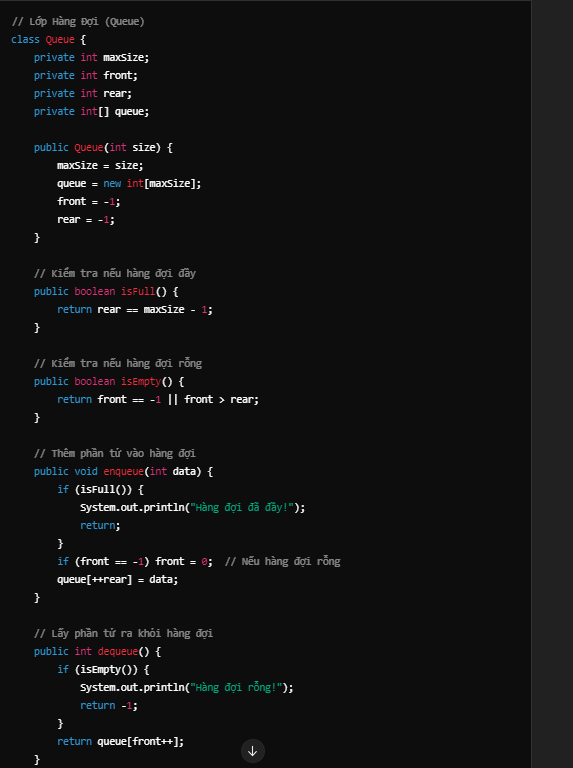
* **Ưu điểm:**
  + Dễ dàng thực hiện các thao tác đẩy (push) và lấy (pop) phần tử.
  + Tốt cho việc lưu trữ và truy xuất các phần tử theo thứ tự ngược lại.
* **Nhược điểm:**
  + Không thể truy xuất phần tử ngoài cùng (top) mà không phải loại bỏ phần tử khác.
* **Ứng dụng:**
  + Kiểm tra biểu thức toán học (trong các biểu thức dạng hậu tố, tiền tố).
  + Thực thi các cuộc gọi hàm trong ngôn ngữ lập trình.
  + Duyệt đồ thị (DFS - Depth First Search).



### 5.Queue

Hàng đợi là một cấu trúc dữ liệu theo nguyên lý "FIFO" (First In, First Out). Phần tử được thêm vào cuối hàng đợi và loại bỏ từ đầu hàng đợi.

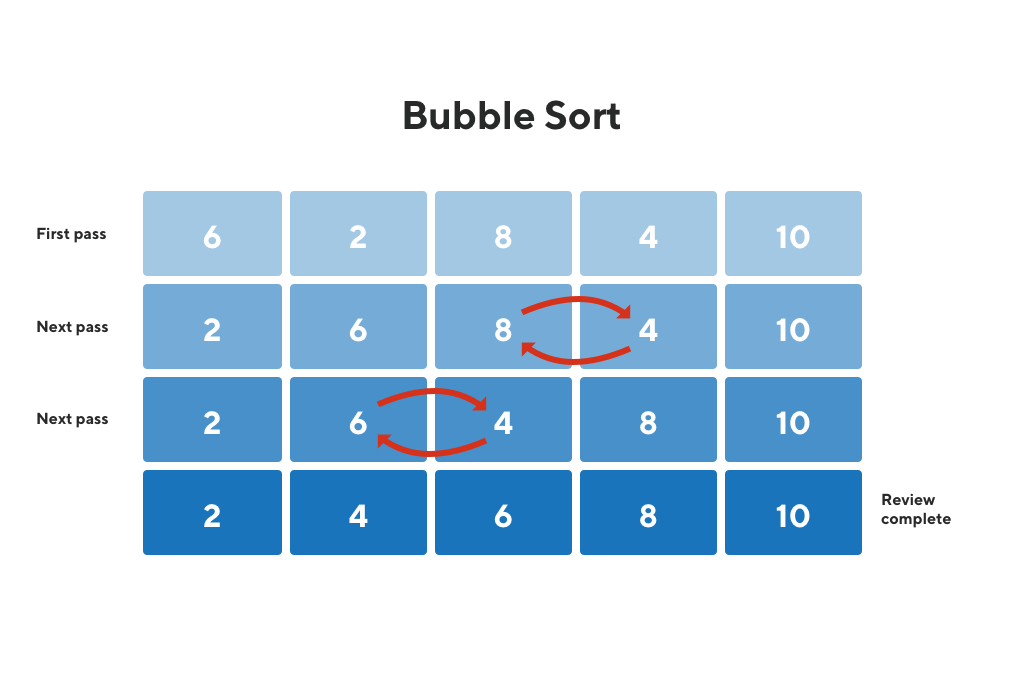
* **Ưu điểm:**
  + Tốt cho các tình huống yêu cầu xử lý theo thứ tự (như xử lý các tác vụ theo thứ tự).
* **Nhược điểm:**
  + Không thể truy xuất trực tiếp các phần tử ngoài cùng mà không loại bỏ phần tử khác.
* **Ứng dụng:**
  + Xử lý các tác vụ trong hệ thống, chẳng hạn như quản lý tác vụ trong hệ điều hành.
  + Cây quyết định trong mạng máy tính.



## B.Giải thuật

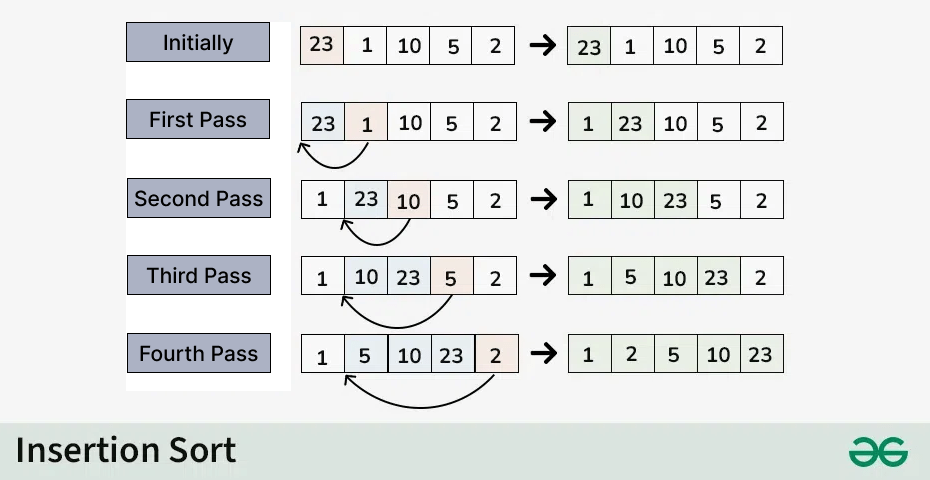
1. **Bubble Sort**  
   Bubble Sort (Sắp xếp nổi bọt) là một thuật toán sắp xếp đơn giản, hoạt động bằng cách liên tục so sánh các cặp phần tử liền kề và hoán đổi chúng nếu chúng không đúng thứ tự.

* **Ưu điểm**:
  + Dễ hiểu và dễ cài đặt.
  + Không tốn thêm nhiều bộ nhớ.
* **Nhược điểm**:
  + Hiệu suất kém đối với các mảng lớn. Không nên sử dụng cho các ứng dụng yêu cầu tốc độ cao.



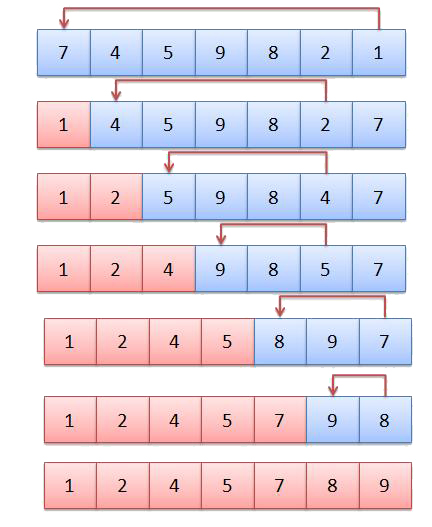
1. **Insertion Sort**  
   Insertion Sort (Sắp xếp chèn) là một thuật toán sắp xếp đơn giản, hoạt động bằng cách xây dựng một mảng con đã được sắp xếp từ từ bằng cách lấy từng phần tử từ mảng chưa sắp xếp và chèn nó vào đúng vị trí trong mảng con đã sắp xếp.

* **Ưu điểm**:
  + Đơn giản và dễ cài đặt.
  + Hiệu quả với các mảng nhỏ hoặc gần như đã được sắp xếp.
  + Sắp xếp tại chỗ (không cần thêm bộ nhớ).
  + Ổn định (thứ tự của các phần tử bằng nhau được giữ nguyên).
* **Nhược điểm**:
  + Kém hiệu quả với các mảng lớn.



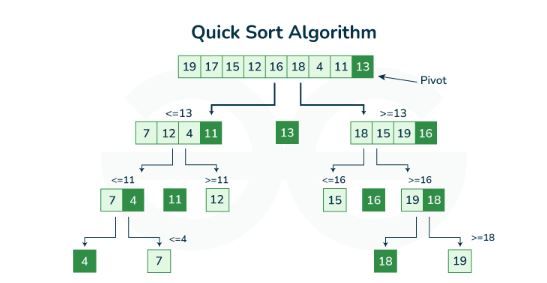
1. **Selection Sort**  
   Selection Sort (Sắp xếp chọn) là một thuật toán sắp xếp đơn giản, hoạt động bằng cách liên tục tìm phần tử nhỏ nhất (hoặc lớn nhất, tùy theo thứ tự sắp xếp) từ phần mảng chưa được sắp xếp và đặt nó vào vị trí đúng của nó trong mảng đã được sắp xếp.

* **Ưu điểm**:
  + Đơn giản và dễ cài đặt.
  + Hoạt động tốt với các mảng nhỏ.
  + Số lần hoán đổi ít hơn so với Bubble Sort.
* **Nhược điểm**:
  + Hiệu suất kém đối với các mảng lớn.



1. **Quick Sort**  
   Quick Sort hoạt động bằng cách chọn một phần tử làm chốt, chia mảng thành hai phần dựa trên chốt đó, và sau đó tiếp tục sắp xếp đệ quy hai phần này.

* **Ưu điểm**:
  + Nhanh: Thường là thuật toán sắp xếp nhanh nhất (trung bình O(n log n)).
  + Hiệu quả bộ nhớ: Sử dụng ít bộ nhớ phụ (trung bình O(log n)).
* **Nhược điểm**:
  + Trường hợp xấu nhất chậm: Có thể chậm đến O(n^2) nếu chọn chốt không tốt.
  + Không ổn định: Thứ tự các phần tử bằng nhau có thể bị thay đổi.
  + Khó tối ưu tuyệt đối: Việc chọn chốt ảnh hưởng lớn đến hiệu suất.



5.Merge sort

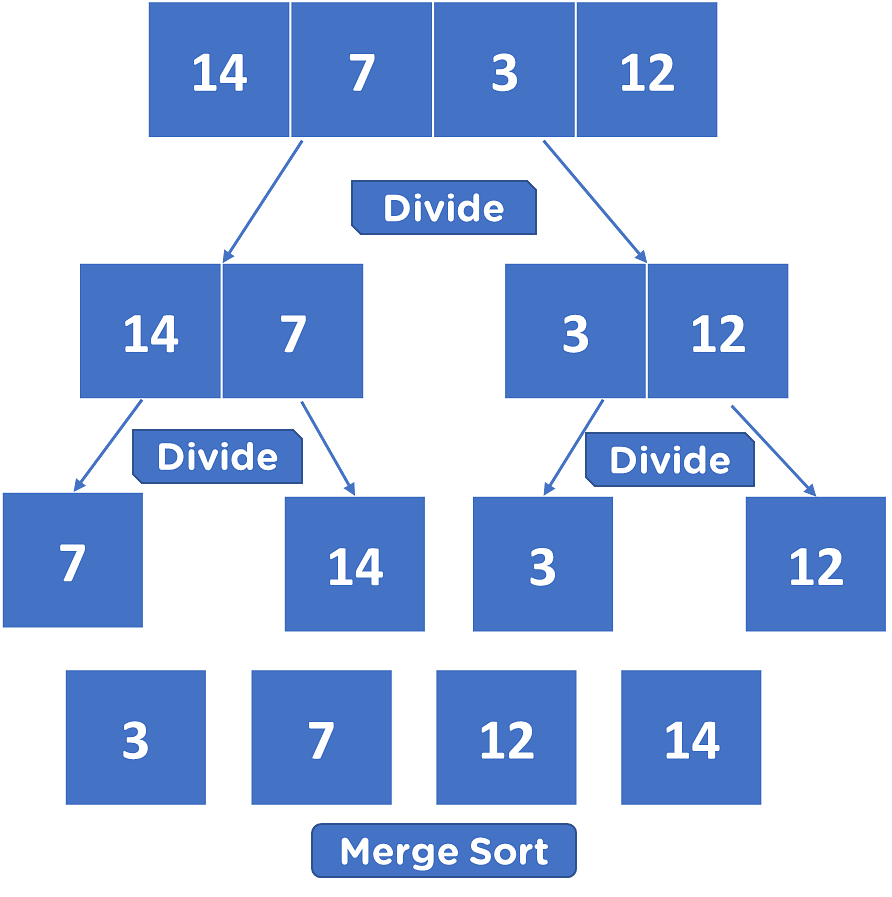
**Merge Sort** là một thuật toán sắp xếp dựa trên nguyên lý chia và trị (divide and conquer), chia dãy thành hai phần nhỏ hơn, sắp xếp mỗi phần, và sau đó hợp nhất (merge) chúng lại với nhau.

**Ưu điểm :**

1. Độ phức tạp thời gian **O(n log n)** ổn định, hiệu quả với dữ liệu lớn.
2. **Sắp xếp ổn định**.
3. Thích hợp với **dữ liệu ngoài bộ nhớ**.
4. Cấu trúc dễ dàng **phân tách và kết hợp**.

**Nhược điểm :**

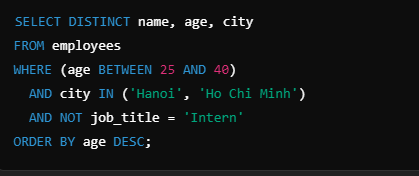
1. **Hiệu suất kém với dữ liệu nhỏ**.
2. **Phức tạp khi triển khai**.



# **SQL**

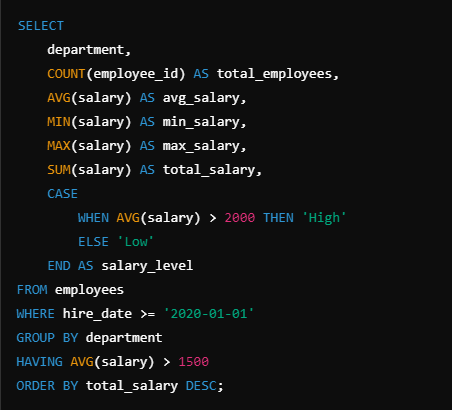
### Phần 1: Truy vấn và Lọc Dữ Liệu (Cơ bản)

* + **SQL Select**: Câu lệnh cơ bản để truy vấn dữ liệu.
  + **SQL Select Distinct**: Lọc các giá trị duy nhất.
  + **SQL Where**: Lọc dữ liệu theo điều kiện.
  + **SQL Order By**: Sắp xếp kết quả truy vấn.
  + **SQL And / Or / Not**: Kết hợp các điều kiện với các phép toán logic.
  + **SQL In / Between**: Tìm dữ liệu trong một dãy giá trị hoặc trong một khoảng.
* **Ví dụ**:



### **Phần 2: Các Hàm Tổng Hợp và Thống Kê**

* + **SQL Aggregate Functions**: Hàm Min, Max, Count, Sum, Avg.
  + **SQL Group By**: Nhóm dữ liệu theo một hoặc nhiều trường.
  + **SQL Having**: Lọc kết quả sau khi nhóm.
  + **SQL Case**: Thực hiện các phép toán điều kiện trong truy vấn.
  + **SQL Null Functions**: Làm việc với giá trị NULL trong SQL.
* **Ví dụ**:



### **Phần 3: Thao Tác Dữ Liệu (Cập nhật và Xóa)**

* + **SQL Insert Into**: Chèn dữ liệu vào bảng.
  + **SQL Update**: Cập nhật dữ liệu đã có trong bảng.
  + **SQL Delete**: Xóa dữ liệu từ bảng.
  + **SQL Insert Into Select**: Chèn dữ liệu từ kết quả của câu lệnh SELECT.

### **Phần 4: Kết Hợp Dữ Liệu**

* + **SQL Joins**: Inner Join, Left Join, Right Join, Full Join, Self Join.
  + **SQL Union**: Kết hợp kết quả từ nhiều câu lệnh SELECT.
  + **SQL Exists**: Kiểm tra sự tồn tại của các giá trị trong một truy vấn con.
  + **SQL Any, All**: So sánh các giá trị với tập hợp kết quả.

### **Phần 5: Quản Lý Cơ Sở Dữ Liệu**

* + **SQL Create DB / Drop DB**: Tạo và xóa cơ sở dữ liệu.
  + **SQL Create Table / Drop Table**: Tạo và xóa bảng.
  + **SQL Alter Table**: Sửa đổi cấu trúc bảng (thêm hoặc xóa cột).
  + **SQL Backup DB**: Sao lưu cơ sở dữ liệu.

### **Phần 6: Ràng Buộc và Chỉ Mục**

* + **SQL Constraints**: Not Null, Unique, Primary Key, Foreign Key, Check, Default.
  + **SQL Index**: Tạo chỉ mục để tối ưu hóa việc truy vấn.
  + **SQL Auto Increment**: Tự động tăng giá trị của trường.