**1- Ejercicio de suma recursiva**

Escribe una función llamada **suma\_recursiva** que reciba un número entero positivo **n** y devuelva la suma de los primeros **n** números naturales de forma recursiva. La función debe cumplir con la siguiente especificación

**2- Ejercicio de combinación de listas ordenadas**

Escribe una función llamada **combinar\_listas\_ordenadas** que reciba dos listas ordenadas de números enteros **lista1** y **lista2**, y devuelva una lista ordenada que contenga todos los elementos de ambas listas sin duplicados. La función debe cumplir con la siguiente especificación

**3- Ejercicio de recuento de elementos únicos**

Escribe un programa que genere una lista de 30 números enteros aleatorios en el rango de 0 a 49, y luego muestre cada número único en la lista junto con la cantidad de veces que aparece. El programa debe cumplir con la siguiente especificación

**4- Desafío de gestión de circuito de carreras en Java**

El desafío consiste en desarrollar un sistema de gestión para un circuito de carreras que pueda manejar tanto vehículos de tipo coche como de tipo moto. El sistema debe permitir agregar participantes al circuito, inicializar la carrera y realizar acciones específicas para cada tipo de vehículo.

**Clases:**

1. **Vehiculo:**
   * Atributos:
     + nombre (String): El nombre del piloto.
     + dorsal (int): El número identificador del vehículo.
     + velocidadActual (int): La velocidad actual del vehículo (inicializada en 0).
     + velocidadMaxima (int): La velocidad máxima del vehículo.
     + tipoCombustible (String): El tipo de combustible del vehículo (electrico, gasolina, gasoil, etc.).
     + numeroRuedas (int): El número de ruedas del vehículo.
   * Métodos:
     + Constructor: **Vehiculo(String nombre, int dorsal, int velocidadMaxima, String tipoCombustible, int numeroRuedas)**
     + **sonidoMotor()**: Imprime el sonido del motor del vehículo según su tipo de combustible.
     + **acelerar(int kmh)**: Acelera el vehículo en la cantidad de kilómetros por hora especificada.
     + **mostrarInfo()**: Retorna una cadena de texto con la información del vehículo.
2. **Coche (Hereda de Vehiculo):**
   * Atributos adicionales:
     + ventanilla (String): Estado de la ventanilla del coche (subida o bajada).
   * Métodos adicionales:
     + **bajarVentanilla()**: Baja la ventanilla del coche.
     + **mostrarInfo()**: Sobrescribe el método de la clase padre para incluir información específica del coche.
3. **Moto (Hereda de Vehiculo):**
   * Atributos adicionales:
     + abrochado (String): Estado del casco del piloto (abrochado o desabrochado).
   * Métodos adicionales:
     + **abrocharCasco()**: Abrocha el casco del piloto.
     + **mostrarInfo()**: Sobrescribe el método de la clase padre para incluir información específica de la moto.
4. **Circuito:**
   * Atributos:
     + nombre (String): Nombre del circuito.
     + listaParticipantes (ArrayList<Vehiculo>): Lista de participantes en el circuito.
   * Métodos:
     + Constructor: **Circuito(String nombre)**
     + **addParticipante(Vehiculo participante)**: Agrega un participante al circuito.
     + **mostrarInfo()**: Imprime la información del circuito y la lista de participantes.
     + **inicializaCarrera()**: Inicializa la carrera mostrando la información de cada participante.

**Uso del sistema:**

1. Crear instancias de Coche y Moto con sus respectivos atributos.
2. Crear una instancia de Circuito con un nombre específico.
3. Agregar participantes al circuito utilizando el método **addParticipante**.
4. Inicializar la carrera con el método **inicializaCarrera**.
5. Realizar acciones específicas para cada tipo de vehículo, como acelerar, bajar ventanillas o abrochar el casco.
6. Mostrar información detallada de cada participante con el método **mostrarInfo**.