1. ¿Cómo funciona el backtracking en este problema?

El algoritmo usa **backtracking** para colocar **N reinas** en un tablero de tamaño **N×N** sin que se ataquen entre sí. Funciona de la siguiente manera:

- Intenta colocar una reina en la primera fila en una columna válida.
- Si logra colocarla, intenta colocar otra reina en la siguiente fila, repitiendo el proceso.
- Si en una fila no puede colocar ninguna reina sin violar las reglas, retrocede (**backtrack**) a la fila anterior y prueba otra opción.
- Continúa hasta colocar todas las reinas o determinar que no hay solución.

2. ¿Qué pasa cuando el algoritmo encuentra una solución? ¿Qué ocurre cuando no puede colocar más reinas?

- Si encuentra una solución, significa que todas las **N reinas** fueron colocadas en el tablero de manera segura. La función `solve` retorna `true` y el programa imprime el tablero.
- Si no hay solución, `solve` retorna `false` y el programa termina sin mostrar una configuración válida.

3. ¿Qué sucede en el código cuando el algoritmo "retrocede"? ¿Cómo se visualiza en Python Tutor?

- Cuando el algoritmo **retrocede**, significa que se colocó una reina en una posición incorrecta, lo que impide colocar las siguientes reinas.
- En ese caso, **deshace la última colocación** (`board[row][col] = 0`) y prueba con otra columna en la misma fila.
- En **Python Tutor**, este proceso se puede visualizar viendo cómo la matriz `board` cambia a medida que las reinas se colocan y eliminan en diferentes posiciones.

4. ¿Qué modificaciones harías para aumentar N a 8? ¿Cómo crees que cambiaría el tiempo de ejecución?

Para cambiar `N` a 8, simplemente modificamos la constante:

```
private static final int N = 8;
```

Esto haría que el algoritmo resolviera el problema para un **tablero de 8×8**.

Impacto en el tiempo de ejecución:

- Reducir `N` de 30 a 8 haría que la ejecución sea **mucho más rápida**, ya que el número de combinaciones posibles se reduce drásticamente.
- Con `N = 30`, el algoritmo puede tardar demasiado, ya que el número de configuraciones a probar es **exponencial** en `N`. Para valores grandes de `N`, una **mejora sería usar heurísticas** como **algoritmos genéticos o búsqueda con poda**.

5. ¿Por qué el método `isSafe` es crucial en este algoritmo?

El método `isSafe` es fundamental porque:

- Verifica si una reina puede ser colocada en una posición sin ser atacada.
- Se asegura de que no haya otra reina en la **misma columna** o en **diagonales superiores**.
- **Optimiza el proceso**, ya que evita colocar reinas en posiciones conflictivas, reduciendo la cantidad de combinaciones a explorar.

Si `isSafe` no estuviera, el algoritmo intentaría todas las combinaciones posibles sin filtrar posiciones inválidas, lo que haría que la ejecución sea **extremadamente ineficiente**.