void reverseArray(int[] array) {

for (int i = 0; i < array.length / 2; i++) {

int reversePosition = array.length - 1 - 1;

int tmpVal = array[i];

array[i] = array[reversePosition];

array[reversePosition] = tmpVal;

### **Temporal**

* **Complejidad temporal lineal O(n):**
* El bucle for itera aproximadamente la mitad del tamaño del arreglo (n/2).
* Dentro del bucle, se realizan un número constante de operaciones (asignaciones, cálculos de índices).
* Por lo tanto, el tiempo de ejecución crece linealmente con respecto al tamaño del arreglo de entrada.

### **Explicación Detallada**

* **Bucle for:** Itera desde el índice 0 hasta aproximadamente la mitad del arreglo. Esto es suficiente para invertir todos los elementos, ya que al intercambiar un par de elementos, ambos quedan en su posición final.
* **Cálculo de reversePosition:** Calcula el índice del elemento a intercambiar con el elemento actual. Restando 1 a array.length se obtiene el último índice, y restando nuevamente i se obtiene el índice simétrico.
* **Intercambio de elementos:** Utiliza una variable temporal tmpVal para intercambiar los valores de los elementos en las posiciones i y reversePosition.

### **Cómo Desarrollar Códigos Más Efectivos**

* **Algoritmos eficientes:** Investigar y utilizar algoritmos con menor complejidad temporal para problemas comunes.
* **Estructuras de datos adecuadas:** Elegir las estructuras de datos que mejor se adapten al problema, optimizando las operaciones necesarias.
* **Optimización de código:** Evitar operaciones innecesarias, utilizar bucles eficientes y aprovechar las características del lenguaje de programación.
* **Análisis de algoritmos:** Estudiar la teoría de la complejidad algorítmica para comprender cómo evaluar la eficiencia de diferentes soluciones.
* **Pruebas y benchmarking:** Medir el rendimiento de diferentes implementaciones para identificar cuellos de botella y realizar mejoras.

**En resumen,** el código para invertir un arreglo proporcionado tiene una complejidad espacial constante y una complejidad temporal lineal. Esto significa que su consumo de memoria es bajo y su tiempo de ejecución crece proporcionalmente al tamaño del arreglo, lo cual es un buen desempeño para esta tarea.