void printUnorderedPairs(int[] array) {

for (int i = 0; i < array.length - 1; i++) {

for (int j = i + 1; j < array.length; j++) {

System.out.println(i + " " + j);

}

Este código, al ejecutarse, imprime todos los pares no ordenados de índices de un arreglo dado. Es decir, para cada elemento del arreglo, se combina con todos los elementos que le siguen, generando así todos los posibles pares sin importar el orden de los elementos dentro del par.

**Ejemplo:**Si el arreglo es [1, 2, 3], el código imprimirá:

* 0 1
* 0 2
* 1 2

### **Complejidad Temporal (Big O)**

La complejidad temporal de este código es **O(n^2)**.

**¿Por qué O(n^2)?**

* **Dos bucles anidados:** La presencia de dos bucles for anidados es la clave para entender la complejidad cuadrática.
* **Iteraciones internas:**
* El bucle externo itera n-1 veces, donde n es el tamaño del arreglo.
* Para cada iteración del bucle externo, el bucle interno itera un número decreciente de veces, pero en promedio, realiza aproximadamente n/2 iteraciones.
* **Multiplicación de iteraciones:** Al multiplicar las iteraciones de ambos bucles, obtenemos un orden de crecimiento cuadrático.

### **Explicación Detallada**

* **Bucle externo:** Itera sobre cada elemento del arreglo, excepto el último.
* **Bucle interno:** Para cada elemento del bucle externo, itera sobre todos los elementos posteriores en el arreglo.
* **Impresión de pares:** En cada iteración del bucle interno, se imprime un par de índices.

**Visualización:**

Imagina una matriz donde las filas representan las iteraciones del bucle externo y las columnas representan las iteraciones del bucle interno. Cada celda de la matriz representa un par de índices impreso. A medida que el tamaño del arreglo aumenta, la matriz crece cuadráticamente, lo que refleja la complejidad temporal O(n^2).

### **Implicaciones de la Complejidad Cuadrática**

* **Rendimiento:** A medida que el tamaño del arreglo aumenta, el tiempo de ejecución del algoritmo crece de manera proporcional al cuadrado del tamaño del arreglo. Esto significa que el algoritmo puede volverse muy lento para arreglos grandes.
* **Escalabilidad:** El algoritmo no escala bien para grandes conjuntos de datos.

### **¿Cómo mejorar la eficiencia?**

Si el objetivo es generar todos los pares no ordenados de índices, este algoritmo es una implementación directa y fácil de entender. Sin embargo, si el rendimiento es una preocupación y se necesitan generar pares de manera más eficiente, se podrían explorar otras estructuras de datos o algoritmos más sofisticados.

**En resumen,** la complejidad temporal O(n^2) del código printUnorderedPairs indica que su tiempo de ejecución crece cuadráticamente con respecto al tamaño del arreglo de entrada. Es importante tener en cuenta esta complejidad al evaluar el rendimiento del algoritmo en diferentes escenarios.