# **Análisis del código Java**

## **Estructura general**

El código define una clase CodeSamples que contiene múltiples métodos implementando diferentes algoritmos y operaciones. También incluye una clase interna Employee.

## **Métodos y sus complejidades**

1. oNSum(), oNSum2(), oNSum3(): O(n)
2. oN2um(): O(n^2)
3. max(): O(1)
4. linearSearch(): O(n)
5. printPairs(): O(n^2)
6. nFactorialExample(): O(n!)
7. reverseArray(): O(n)
8. printUnorderedPairs(): O(n^2)
9. product(): O(b)
10. copyArray(): O(n^2) debido a appendToNew()
11. appendToNew(): O(n)
12. printPairsTwoArrays(): O(n\*m)
13. intPowerOfTwo(): O(log n)
14. fibonacci(): O(2^n)
15. fibonacciMemo(): O(n)
16. fibonacciBottomUp(): O(n)
17. Operaciones de bits (getBit, setBit, clearBit, etc.): O(1)

## **Análisis de calidad del código**

### **Puntos positivos:**

1. Nombres de métodos descriptivos
2. Uso de tipos primitivos cuando es apropiado
3. Implementación de algoritmos clásicos (búsqueda lineal, Fibonacci)
4. Variedad de enfoques para problemas similares (ej. diferentes implementaciones de Fibonacci)

### **Áreas de mejora:**

1. Falta de comentarios explicativos
2. Algunos métodos podrían beneficiarse de parámetros en lugar de arrays hardcodeados
3. Algunos nombres de variables podrían ser más descriptivos (ej. val, e)
4. Falta de manejo de excepciones y validación de entrada

## **Complejidad algorítmica**

El código muestra una variedad de complejidades, desde O(1) hasta O(n!). La mayoría de los métodos tienen una complejidad aceptable para sus propósitos, pero algunos podrían optimizarse:

1. copyArray() podría mejorarse a O(n) utilizando un array de tamaño fijo
2. fibonacci() podría usar memoización o enfoque bottom-up para mejorar de O(2^n) a O(n)

## **Programación limpia**

El código cumple con algunos principios de programación limpia, como nombres de métodos descriptivos y métodos cortos con una sola responsabilidad. Sin embargo, podría mejorar en:

1. Añadir comentarios explicativos
2. Mejorar nombres de algunas variables
3. Utilizar parámetros en lugar de arrays hardcodeados
4. Implementar manejo de excepciones

## **Esquema detallado**

1. Clase Employee

* Atributos: name
* Métodos: constructor, getName()

1. Métodos de suma y producto

* oNSum(), oNSum2(), oNSum3()

1. Operaciones de búsqueda y comparación

* max(), linearSearch()

1. Métodos de impresión

* printPairs(), printUnorderedPairs(), printPairsTwoArrays()

1. Operaciones con arrays

* reverseArray(), copyArray(), appendToNew()

1. Algoritmos matemáticos

* product(), intPowerOfTwo(), fibonacci() (tres versiones)

1. Operaciones de bits

* getBit(), setBit(), clearBit(), clearBitsFromMostSignificantToIndex(), cleatBitsFromIndexToLessSignificant()