CÓDIGO 1

```
func procesar(lista):
   total = 0
   for i desde 1 hasta n:
      j = i
      while j > 0:
         total = total + lista[j]
         j = j / 2
   return total
```

MI ANÁLISIS:

El ciclo for se ejecuta n veces.
El ciclo while divide j entre 2, por lo que su complejidad es O(log n).

Al estar anidados, la complejidad total es:

O(n log n)

Este análisis corresponde al peor caso, ya que el ciclo interno siempre se ejecuta el mismo número de veces.

CÓDIGO 2

```
func buscarDuplicados(lista):
   for i desde 1 hasta n:
      for j desde 1 hasta n:
         if lista[i] == lista[j]:
            return true
   return false
```

MI ANÁLISIS:

El algoritmo tiene dos ciclos anidados, por lo que su complejidad es $O(n^2)$.
Sin embargo, como existe un return dentro del ciclo, el algoritmo puede terminar antes.

Por lo tanto, la complejidad final es O(n) en el peor caso.

CÓDIGO 3

```
func dividirYContar(n):
   if n <= 1:
      return 1
```

```
  return dividirYContar(n / 2) + 1
```

MI ANÁLISIS:

En cada llamada recursiva el valor de n se divide entre 2.
Esto significa que el número de llamadas crece proporcionalmente a n.

Por lo tanto, la complejidad del algoritmo es O(n).

CÓDIGO 4

```
func procesarMatriz(matriz):
  for i desde 1 hasta n:
    for j desde 1 hasta n:
      print(matriz[i][j])

  for k desde 1 hasta n:
    print(k)
```

MI ANÁLISIS:

El primer bloque tiene dos ciclos anidados, lo que da $O(n^2)$.
El segundo bloque tiene un solo ciclo, O(n).

Al combinar ambas partes, la complejidad total es:

$O(n^3)$

Auditoría (resuelto)

Codigo 1

Errores detectados:

- Se asumió que el ciclo while ejecuta siempre log n veces.
- Se ignoró que j inicia en i, no en n.

Corrección:

- El ciclo interno es O(log i), no O(log n).
- La complejidad total sigue siendo O(n log n), pero el razonamiento es incorrecto.

Verificación:
Se detectaron todos los errores.

Codigo 2

Errores detectados:

- Se confundió el mejor caso con el peor caso.
- Se asumió que un retorno temprano reduce la complejidad del peor caso.

Corrección:

- En el peor caso no hay duplicados.
- Ambos ciclos se ejecutan completamente.

Complejidad correcta: $O(n^2)$

Verificación:
Se detectaron todos los errores.

Codigo 3

Errores detectados:

- Se interpretó incorrectamente el crecimiento de la recursión.
- Se confundió crecimiento lineal con logarítmico.

Corrección:

- Cada llamada reduce el problema a la mitad.
- El número de llamadas es log n.

Complejidad correcta: O(log n)

Verificación:
Se detectaron todos los errores.

Codigo 4

Errores detectados:

- Se sumaron complejidades como si se multiplicaran.
- Se concluyó incorrectamente $O(n^3)$.

Corrección:

- Los bloques secuenciales se suman, no se multiplican.

- $O(n^2) + O(n)$ se reduce a $O(n^2)$.

Verificación:
Se detectaron todos los errores.

Resultados metacognitivos

Porcentaje de errores detectados:
100%

Tipos de errores más difíciles de detectar:

- Diferenciar mejor caso y peor caso.
- Detectar errores de justificación cuando el resultado final parece correcto.
- Manejo incorrecto de recursión.
- Confusión entre suma y multiplicación de complejidades.