PROBLEMA 1: Rotar Arreglo

Pseudocódigo de la solución (O(n) tiempo, O(1) espacio)

```
función rotarDerecha(A, k):
   n = longitud(A)
   k = k mod n

   invertir(A, 0, n-1)
   invertir(A, 0, k-1)
   invertir(A, k, n-1)

función invertir(A, inicio, fin):
   mientras inicio < fin:
      intercambiar A[inicio], A[fin]
      inicio = inicio + 1
      fin = fin - 1
```

Análisis de complejidad

- Complejidad temporal:
  O(n), porque cada inversión recorre el arreglo una vez.
- Complejidad espacial:
  O(1), solo se usan variables auxiliares.

Evolución de la solución (mejora)

- Solución inicial típica: mover un elemento k veces
  Complejidad: O(n · k) → ineficiente cuando k es grande.
- Mejora intermedia: usar un arreglo auxiliar
  Complejidad: O(n) tiempo, O(n) espacio.
- Solución final (reversiones):
  Mantiene O(n) tiempo y reduce el espacio extra a O(1).

PROBLEMA 2: Eliminar Duplicados

Pseudocódigo de la solución

```
función eliminarDuplicados(A):
   si longitud(A) == 0:
      retornar 0

   indice = 1

   para i desde 1 hasta longitud(A) - 1:
      si A[i] != A[i - 1]:
         A[indice] = A[i]
         indice = indice + 1

   retornar indice
```

Análisis de complejidad

- Complejidad temporal:
  O(n), se recorre el arreglo una sola vez.
- Complejidad espacial:
  O(1), se modifica el arreglo en el mismo espacio.

Evolución de la solución

- Solución inicial común: crear un nuevo arreglo sin duplicados
  Tiempo O(n), espacio O(n).
- Solución final: dos punteros
  Mantiene O(n) tiempo y reduce espacio a O(1).

PROBLEMA 3: Mover Ceros al Final

Pseudocódigo de la solución

```
función moverCeros(A):
   indice = 0

   para i desde 0 hasta longitud(A) - 1:
      si A[i] != 0:
         intercambiar A[indice], A[i]
         indice = indice + 1
```

Análisis de complejidad

- Complejidad temporal:
  O(n), cada elemento se procesa una vez.
- Complejidad espacial:
  O(1), solo se usan variables auxiliares.

Evolución de la solución

- Solución inicial típica: usar un arreglo auxiliar
  Tiempo O(n), espacio O(n).
- Solución optimizada: swaps in-place con dos índices
  O(n) tiempo y O(1) espacio.

PROBLEMA 4: Encontrar Elemento Mayoritario

Pseudocódigo de la solución (Boyer–Moore)

```
función elementoMayoritario(A):
   candidato = A[0]
   contador = 1

   para i desde 1 hasta longitud(A) - 1:
      si A[i] == candidato:
         contador = contador + 1
      sino:
         contador = contador - 1
         si contador == 0:
            candidato = A[i]
            contador = 1

   retornar candidato
```

Análisis de complejidad

- Complejidad temporal:
  O(n), una sola pasada por el arreglo.
- Complejidad espacial:
  O(1), solo variables constantes.

Evolución de la solución

- Solución inicial común: usar hashmap para contar frecuencias
  Tiempo O(n), espacio O(n).
- Solución final: algoritmo de votación de Boyer–Moore
  O(n) tiempo y O(1) espacio.