



**INSTITUTO POLITÉCNICO NACIONAL**  
**Escuela Superior de Cómputo**



**Análisis y diseño de algoritmos**

## **Ejercicio 9**

**Gil Juárez Hector David**

**3CV1**

**Mayo 19, 2024**



## Desarrollo

El problema del "magic index" consiste en encontrar un índice en un array ordenado de enteros distintos tal que el valor del array en ese índice sea igual al propio índice. Es decir, dado un array  $A$  de longitud  $n$ , estamos buscando un índice  $i$  (donde  $0 \leq i < n$ ) tal que  $A[i] = i$ .

### Ejemplo del Problema:

Imagina que tienes el siguiente array ordenado de enteros distintos:

$A = [-1, 0, 2, 4, 5]$

En este caso:

$A[0] = -1$

$A[1] = 0$

$A[2] = 2$

$A[3] = 4$

$A[4] = 5$

El índice mágico es 2 porque  $A[2] = 2$ .

### Propiedades del Problema:

Array Ordenado: El array está ordenado de forma creciente.

Elementos Distintos: Todos los elementos del array son distintos.

### Objetivo:

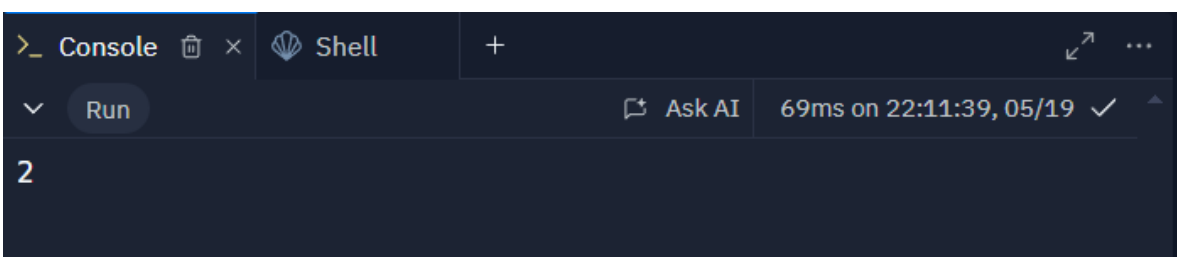
El objetivo es diseñar un método para encontrar dicho índice mágico de manera eficiente.

### Algoritmos para Resolver el Problema

- Búsqueda Lineal: Revisar cada elemento del array uno por uno hasta encontrar un índice mágico, si es que existe. Esto tiene una complejidad de tiempo  $O(n)$ .
- Búsqueda Binaria: Utilizar el hecho de que el array está ordenado para realizar una búsqueda binaria, lo que reduce la complejidad a  $O(\log n)$ .

### Solución sencilla:

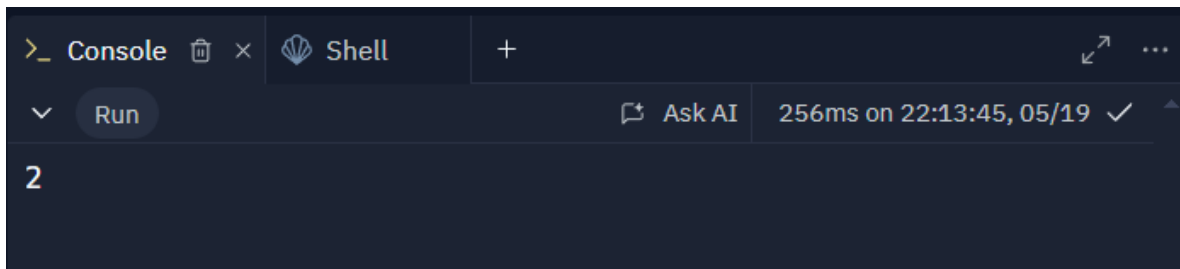
```
1 def find_magic_index_simple(arr):
2     for i in range(len(arr)):
3         if arr[i] == i:
4             return i
5     return -1
6 # Ejemplo de uso:
7 arr = [-1, 0, 2, 4, 5]
8 print(find_magic_index_simple(arr))
```



The screenshot shows a code editor with a 'Console' tab. The code from the previous block has been executed, and the output '2' is displayed in the console. The editor interface includes a 'Run' button, an 'Ask AI' button, and a timestamp '69ms on 22:11:39, 05/19'.

**Solución eficiente:**

```
1 def find_magic_index_efficient(arr):
2     def binary_search(arr, start, end):
3         if start > end:
4             return -1
5         mid = (start + end) // 2
6         if arr[mid] == mid:
7             return mid
8         elif arr[mid] > mid:
9             return binary_search(arr, start, mid - 1)
10        else:
11            return binary_search(arr, mid + 1, end)
12    return binary_search(arr, 0, len(arr) - 1)
13 # Ejemplo de uso:
14 arr = [-1, 0, 2, 4, 5]
15 print(find_magic_index_efficient(arr))
16
```

A screenshot of a code editor's interface. At the top, there are tabs for 'Console' and 'Shell'. Below the 'Console' tab, there is a 'Run' button and an 'Ask AI' button. To the right of the 'Ask AI' button, it says '256ms on 22:13:45, 05/19'. The main area of the console shows the number '2'.

1. Caso Base: Si start es mayor que end, significa que no se ha encontrado un índice mágico en el subarray y se retorna -1.
2. Caso Medio: Se calcula el índice medio mid del subarray.
3. Verificación: Si  $arr[mid]$  es igual a mid, hemos encontrado el índice mágico y se retorna mid.
4. División del Array: Si  $arr[mid]$  es mayor que mid, el índice mágico debe estar en el subarray izquierdo, así que se llama recursivamente a `binary_search` en la mitad izquierda del array. Si  $arr[mid]$  es menor que mid, el índice mágico debe estar en el subarray derecho, así que se llama recursivamente a `binary_search` en la mitad derecha del array.

Ambas soluciones cumplen con encontrar un índice mágico en el array, si es que existe, pero la solución eficiente utiliza el poder de la búsqueda binaria para mejorar significativamente el rendimiento en comparación con la búsqueda lineal.