15-2-2026

Semana 3

Ejercicios- Dividir y conquistar

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

UNIVERSIDAD

Laila Zareth Romano Guerrero

DDYA-7

Ejercicios

1. **Dado un arreglo de N enteros, cuyos valores van en decremento y luego incremento, encontrar el menor número en el arreglo.**
   $\boldsymbol{N} = [\mathbf{2}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}]$
   $\boldsymbol{N} = [\mathbf{8}, \mathbf{5}, \mathbf{4}, \mathbf{3}, \mathbf{4}, \mathbf{10}]$
   Estrategia: Aplicar búsqueda binaria
   Lógica: Ejemplo de ejecución con el segundo arreglo
   - Nuestro mid en este caso apuntaría al numero 4
   - Entonces, nos preguntamos ¿4 es menor que 5? No, porque en realidad 4 es mayor que 3.
   - Como sabemos que 4 es mayor que 3, el siguiente a este numero es menor, el algoritmo se mueve a la derecha y así encontrara el numero 3.

<table>
<tr><th>Código</th><th>Cost</th><th>Times</th></tr>
<tr><td rowspan="11"><pre>def binary_search(A, k):
    low = 0
    high = len(A)
    while low < high:
        mid = low + (high - low) // 2
        if A[mid] == k:
            return mid
        else:
            if A[mid] > k:
                high = mid
            else:
                low = mid + 1</pre></td><td>C_1</td><td>1</td></tr>
<tr><td>C_2</td><td>1</td></tr>
<tr><td>C_3</td><td>$\log(n+1)$</td></tr>
<tr><td>C_4</td><td>$\log n$</td></tr>
<tr><td>C_5</td><td>$\log n$</td></tr>
<tr><td>C_6</td><td>$1\ (A[mid] == k)$</td></tr>
<tr><td>C_7</td><td></td></tr>
<tr><td>C_8</td><td>$\log n$</td></tr>
<tr><td>C_9</td><td>$\log n\ (A[mid] > k)$</td></tr>
<tr><td>C_{10}</td><td></td></tr>
<tr><td>C_{11}</td><td>$\log n$</td></tr>
</table>

Parámetros del teorema maestro

$$T(n) = aT(\frac{n}{b}) + f(n)$$

Donde en este caso:

$a = 1$ Ya que en cada iteración el algoritmo solo explora un subproblema, ya sea la mitad de la derecha o la izquierda, mas no ambos.

$b = 2$ Al querer buscaren cada caso, la búsqueda se divide a la mitad en cada paso al calcular **mid**.

$f(n) = O(1)$ Todas las operaciones dentro de nuestro bucle **while** tienen un costo constante.

$$f(n) = O(n^{\log_b a}) \rightarrow T(n) = O(n^{\log_b a} \log n)$$

$$T(n) = O(1 \times \log n) = O(\log n)$$

En conclusión, la complejidad para esta búsqueda binaria es logarítmica.

2. **Dado un arreglo N-1 enteros ordenados, cuyos valores están en el rango 1 a N, encontrar el entero faltante dado que uno de ellos no esta presente en el arreglo.**

$N = [1, 2, 3, 4, 6, 7, 8]$

$N = [1, 2, 3, 4, 5, 6, 8, 9]$

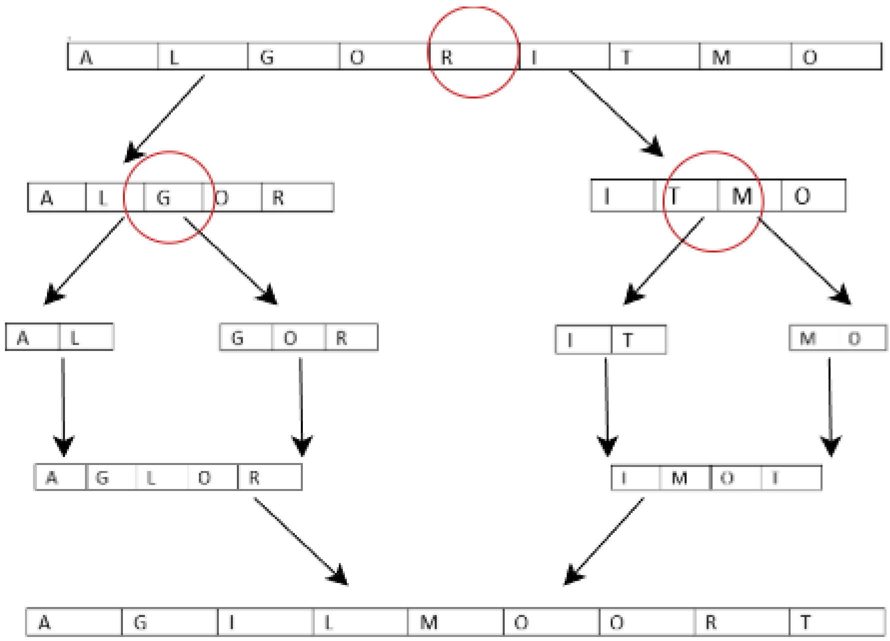
Al igual que en el anterior punto, considero que se debe usar búsqueda binaria ya que el enunciado especifica que los números están ordenados y la búsqueda binaria aprovecha cuando los arreglos están ordenados. También el problema permite que descartemos la mitad de los elementos que se encuentran en el arreglo, ya que al comparar el valor de la posición media con el índice podemos saber si el numero faltante se encuentra a la izquierda o la derecha.

3. Diseñar un algoritmo D&G para calcular el exponente de un número a^n

De manera clásica utilizaría como solución multiplicar a por si mismo cuantas n veces sea necesario, pero al pensarlo de forma D&G podemos aprovechar propiedades de las potencias, específicamente, cuando sea ingresado un número n impar tendríamos $a^n = a \times (n^{n/2})^2$ y si se ingresa un número par $a^n = (n^{n/2})^2$ .Teniendo en cuenta esto:

```
def potencia(base, exponente):
    #caso base 1, cualquier n a la 0 es igual a 1
    if exponente == 0:
        return 1
    #caso base 2, cualquier n elevado a la 1 es el mismo n
    if exponente == 1:
        return base
#llamo a la funcion, pero dividiendo a la mitad el exponente
    divide = potencia(base, exponente//2)
#conquisto
    if exponente % 2 == 0:
        return divide * divide
    else: #cuando el exponente es impar
        return base * divide * divide
```

4. Aplicar mergesort para ordenar "ALGORITMO" en orden alfabético.



5. **Dado un número X, encontrar el número N tal que la suma de los bits de cada numero desde 1 hasta N sea al menos X.**

X = 5
N = 4

sum_bits(1) = 1
sum_bits(2) = 1
sum_bits(3) = 2
sum_bits(4) = 1