# Adivinar un número - Algoritmo recursivo

Moisés Quintero* Santiago Hernández†

February 21, 2024

**Abstract**

Este documento presenta el diseño e implementación de un algoritmo basado en la estrategia dividir-y-vencer para el juego de "adivinar un número". Se detalla la implementación del algoritmo y se presenta un documento de diseño que respalda la lógica subyacente.

Algoritmo, escritura formal, dividir y vencer, aleatorio, rangos.

## 1 Dividir y Vencer

La estrategia de "Dividir y Vencer" es un enfoque fundamental en el diseño de algoritmos que busca resolver problemas complejos dividiéndolos en subproblemas más pequeños, resolviendo cada subproblema de manera independiente y combinando sus soluciones para obtener la solución final del problema original. Este paradigma se caracteriza por tres pasos fundamentales: dividir, conquistar y combinar.

- La primera fase implica dividir el problema original en subproblemas más pequeños e independientes.
- La segunda fase consiste en resolver cada uno de los subproblemas de manera independiente. Se aplica la misma estrategia de "Dividir y Vencer" a cada subproblema, ya que, al ser más pequeños, son más manejables y su solución es más directa.
- La fase final implica combinar las soluciones de los subproblemas para obtener la solución completa del problema original. Este paso es esencial para garantizar que la solución global esté correctamente integrada a partir de las soluciones parciales.

La eficacia de la estrategia de "Dividir y Vencer" se evalúa mediante el análisis de la complejidad temporal y espacial del algoritmo resultante. La complejidad

*moisesd.quintero@javeriana.edu.co
†santiagoa_hernandez@javeriana.edu.co

temporal se expresa en función del tamaño del problema original $n$ y la complejidad temporal de cada subproblema. Si denotamos $T(n)$ como la complejidad temporal total, la relación recurrente típica se expresa como:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

donde $a$ es el número de subproblemas, $b$ es el factor de división, y $f(n)$ es el tiempo necesario para dividir, conquistar y combinar.

# 2 Análisis de Algoritmo

## 2.1 Entradas y Salidas

El algoritmo de adivinanza de números mediante recursividad toma dos parámetros de entrada: $b$, el límite inferior del rango (inicializado en 0); $e$, el límite superior del rango definido por el usuario; y $n$, que es la cantidad de números a adivinar por turno. La salida del algoritmo es el número adivinado. El código principal interactúa con el usuario para obtener los límites superiores e inferiores del rango ($e$) y ($b$). Esto lo hace con la pregunta: *"es Mayor(1),Menor(2) o Igual(otro) a tu numero?"*.

## 2.2 Funcionamiento

Este algoritmo emplea una estrategia de "Dividir y Vencer" basada en la recursividad. En cada llamada recursiva, le pregunta al "pensador" si el número pensado es mayor, menor o igual a *mid*. En el caso de que sea mayor, determina un nuevo límite inferior, y en el caso de que sea menor, determina un nuevo límite superior.

## 2.3 Complejidad

La complejidad temporal de este algoritmo recursivo puede ser analizada utilizando el Teorema Maestro. La relación recurrente que describe la complejidad es:

$$T(n) = T\left(\frac{n}{usu}\right) + O(n)$$

donde $n = e - b$. Aquí, $a = 1$ (el número de subproblemas); $f(n) = O(n)$ representa el tiempo necesario para dividir y comparar; y $usu =$ depende de la cantidad de números a adivinar por turno (parámetro ingresado por el usuario).

Aplicando el Teorema Maestro, identificamos un comportamiento generalizado. El algoritmo, siendo $d$ el grado de $O(n)$, siempre seguirá este patrón:

$$a < usu^d$$

Por lo tanto, la complejidad temporal del algoritmo de adivinanza de números mediante recursividad es $O(n)$

## 2.4 Notas de Implementación

- El algoritmo requiere constante interacción del usuario.
- Los valores de número máximo y cantidad de números a adivinar por turno, que son ingresados por el usuario, deben ser mayor a 0.
- Todos los valores de entradas serán números naturales (mayores a 0).
- La cantidad de partes a dividir el rango (La generacion del numero n) deber ser menor o igual a (e-b)

## 2.5 Pseudocódigo

**Algorithm 1** Adivinar Número - "Dividir y Vencer"

```
procedure ADIVINADOR(b, e, n)
    arreglo ← []
    if b = e then
        return b
    end if
    for i ← 1 to n do
        arreglo[i] ← b + ((e − b))//n) ∗ (i + 1)
    end for
    print arreglo
    for i ← 1 to n do
        print "El numero", arreglo[i], "es Mayor(1), Menor(2) o Igual(otro)
a tu numero?"
        respuesta ← input()
        if respuesta = 1 then
            if arreglo[i] < e then
                e ← arreglo[i]
            end if
        else if respuesta = 2 then
            if arreglo[i] > b then
                b ← arreglo[i]
            end if
        else
            return arreglo[i]
        end if
    end for
    return ADIVINADOR(b, e, n)
end procedure
procedure ADIVINAR_NUMERO
    while True do
        print "Hasta qué número quieres que se elija?"
        e ← input()
        print "Cuantos números quieres que adivine cada vez?"
        n ← input()
        if e ≥ 0 and n ≥ 0 then
            if n > e then
                print "No puedo dividir en más partes que el numero de ele-
mentos"
            end if
            print "Piensa un número del 0 al", e
            print ADIVINADOR(0, e, n), "es tu número!"
            break
        else
            print "No puedes y no puedo pensar en números negativos"
        end if
    end while
end procedure
```