🔢 Números Perfectos, Abundantes y Deficientes

✨ Números Perfectos

- Los **números perfectos** son conocidos desde la **Antigüedad**.
- Los primeros registros aparecen en los **matemáticos griegos**, especialmente en **Euclides (alrededor del 300 a.C.)**.
- Euclides mostró que ciertos números de la forma `2^(p-1) * (2^p - 1)` son perfectos, donde `2^p - 1` es un número primo (ahora llamados **primos de Mersenne**).
- Ejemplos clásicos: **6, 28, 496, 8128**.
- Se pensaba que tenían **propiedades místicas y religiosas**, ya que representaban **armonía y perfección**.

💡 Curiosidades

> - ¿Existen infinitos números perfectos? Aún es un **problema abierto**.
> - Hasta 2024, se conocen **52 números perfectos**.
> - Se desconoce si existe algún **número perfecto impar**.
> - La suma de divisores y su cantidad se puede calcular usando propiedades matemáticas de los números y su factorización.

Números Abundantes y Deficientes

- Los **números abundantes** y **deficientes** también fueron estudiados por los **pitagóricos**, quienes clasificaban los números según la suma de sus divisores.
- **Número abundante:** la suma de sus divisores propios es mayor que el número.
  - Ejemplo: 12 → 1 + 2 + 3 + 4 + 6 = 16 > 12 ☑
- **Número deficiente:** la suma de sus divisores propios es menor que el número.
  - Ejemplo: 8 → 1 + 2 + 4 = 7 < 8 ☑
- Esta clasificación ayudó a los matemáticos antiguos a explorar **patrones numéricos** y relaciones entre números, y sentó las bases de la **teoría de números** moderna.

> 📝 Tip: Estos conceptos son útiles en **programación competitiva** y **problemas de teoría de números**, especialmente al calcular divisores y analizar patrones numéricos.

📊 Ejemplos de Números

Número	Divisores Propios	Suma	Tipo
6	1, 2, 3	6	Perfecto ✨
28	1, 2, 4, 7, 14	28	Perfecto ✨
12	1, 2, 3, 4, 6	16	Abundante ⚡
18	1, 2, 3, 6, 9	21	Abundante ⚡
8	1, 2, 4	7	Deficiente 🔘

Número	Divisores Propios	Suma	Tipo
15	1, 3, 5	9	Deficiente 🔘

💻 Algoritmo por Fuerza Bruta: Números Perfectos, Abundantes y Deficientes O(n)

Idea del algoritmo por fuerza bruta

1. Un número se clasifica según la **suma de sus divisores propios**:
   - **Perfecto** ✨: suma = número
   - **Abundante** ⚡: suma > número
   - **Deficiente** 🔘: suma < número
2. Para calcular la suma de divisores propios de un número `n`, se recorre **cada entero desde 1 hasta n-1**.
3. Si el entero divide exactamente a `n` (`n % i == 0`), se suma a la suma total.
4. Finalmente, se compara la suma obtenida con `n` para determinar su tipo.

Implementación por fuerza bruta

```
# Función que calcula la suma de divisores propios
def suma_divisores(n):
    suma = 0
    for i in range(1, n):
        if n % i == 0:
            suma += i
    return suma

# Función que determina el tipo de número
def tipo_numero(n):
    s = suma_divisores(n)
    if s == n:
        return "Perfecto"
    elif s > n:
        return "Abundante"
    else:
        return "Deficiente"

# Ejemplos de uso
numeros = [6, 28, 12, 18, 8, 15]

for num in numeros:
    print(f"{num}: {tipo_numero(num)}")
```

📊 Ejemplos de salida

Número	Tipo
6	Perfecto

Número	Tipo
28	Perfecto
12	Abundante
18	Abundante
8	Deficiente
15	Deficiente

> **Notas:** Su complejidad es `O(n)`, lo que significa que para números muy grandes puede ser lento.

💻 Algoritmo optimizado: Números Perfectos, Abundantes y Deficientes `O(√n)`

💻 Idea del Algoritmo Optimizado

1. Observación clave: los divisores de un número `n` siempre aparecen en **pares** `(i, n//i)`.
2. Basta con iterar hasta `√n`.
3. Cada divisor encontrado `i` suma a la suma total **junto con su par** `n//i`.
4. Excluimos el número `n` de la suma (solo divisores propios).
5. Finalmente, comparamos la suma con `n` para clasificarlo.

> ☑ Esto reduce la complejidad de O(n) a **O(√n)**, mucho más eficiente para números grandes.

Implementación algoritmo optimizado

```
# Suma de divisores propios usando optimización
def suma_divisores_optimizada(n):
    if n == 1:
        return 0  # 1 es deficiente
    suma = 1  # 1 siempre es divisor propio
    limite = int(n**(1/2))
    for i in range(2, limite + 1):
        if n % i == 0:
            suma += i
            if i != n // i: # sumamos el otro divisor de la forma n//i si es
diferente de i
                suma += n // i
    return suma

# Determinar tipo de número
def tipo_numero_optimizado(n):
    s = suma_divisores_optimizada(n)
    if s == n:
        return "Perfecto"
    elif s > n:
        return "Abundante"
    else:
        return "Deficiente"
```

```
# Ejemplos
numeros = [6, 28, 12, 18, 8, 15]

for num in numeros:
    print(f"{num}: {tipo_numero_optimizado(num)}")
```

> **Notas:**
> Ideal para **programación competitiva** cuando los números son grandes.
> Mantiene la lógica simple, pero mucho más eficiente que recorrer hasta `n-1`.

Problemas: Clase 4 – Números Perfectos, Abundantes y Deficientes