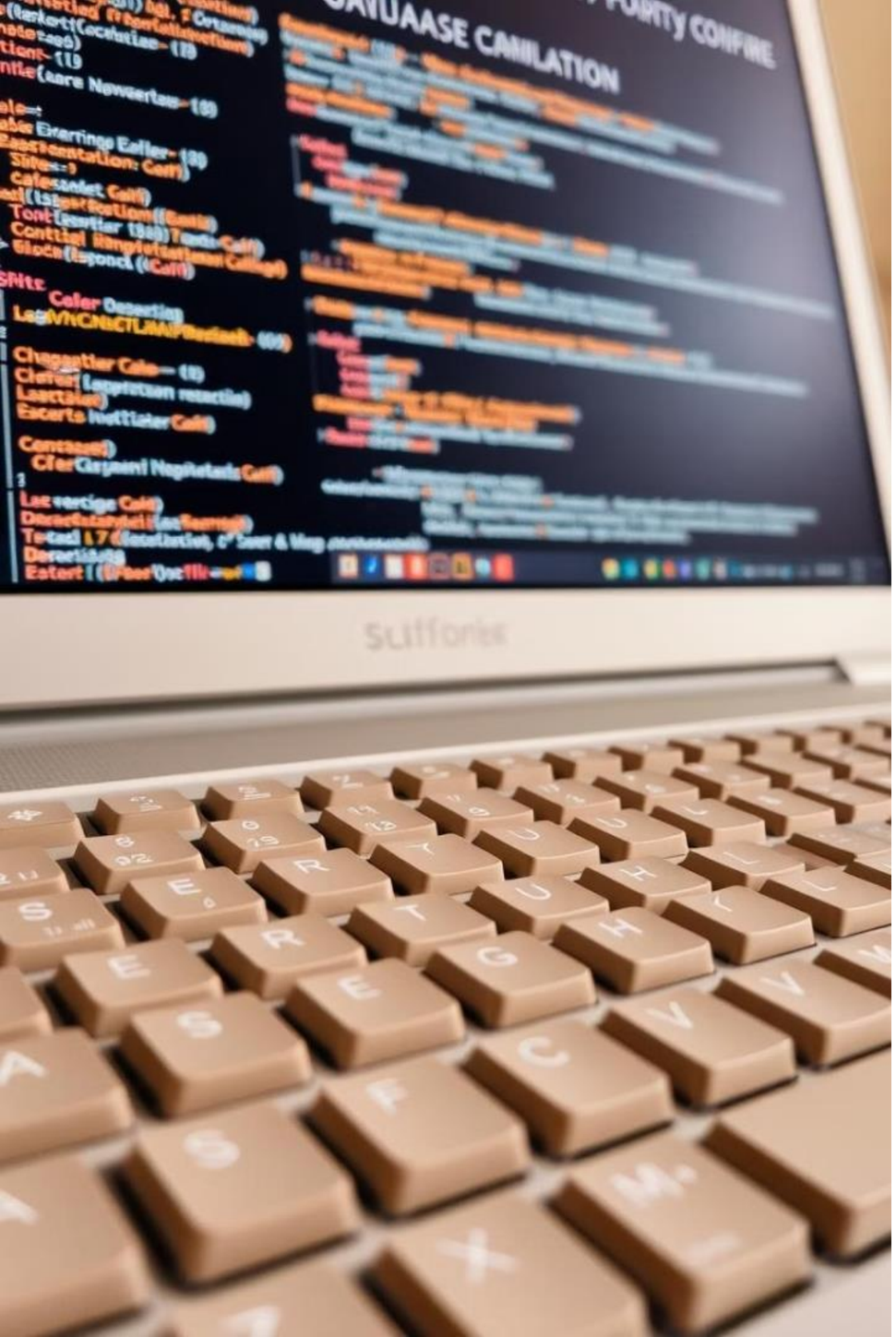


Análisis Teórico de Algoritmos

Esta presentación explorará los fundamentos del análisis teórico de algoritmos. Aprenderemos a analizar el rendimiento de los algoritmos mediante el conteo de operaciones, lo que nos permitirá estimar su eficiencia y comparar su comportamiento con el crecimiento del tamaño de la entrada.

por Ariel Enferrel

Introducción al Análisis Teórico

Objetivo

El objetivo principal del análisis teórico es obtener una función matemática T(n) que describa el número de operaciones que un algoritmo realiza para una entrada de tamaño n. Esta función nos da una visión general de la eficiencia del algoritmo.

Ventajas

El análisis teórico ofrece ventajas significativas, ya que nos permite:

- Comparar algoritmos para un mismo problema.
- Identificar la complejidad computacional del algoritmo.
- Predecir el rendimiento del algoritmo en diferentes tamaños de entrada.

Cálculo de T(n)

Principio Fundamental

Cada operación básica dentro de un algoritmo se cuenta como una unidad de tiempo (por ejemplo, 1 nanosegundo). Las operaciones básicas incluyen:

- Asignación de valor a una variable.
- Acceso a un elemento en un array.
- Evaluación de una expresión aritmética o lógica.
- Devolución de un valor en una función.

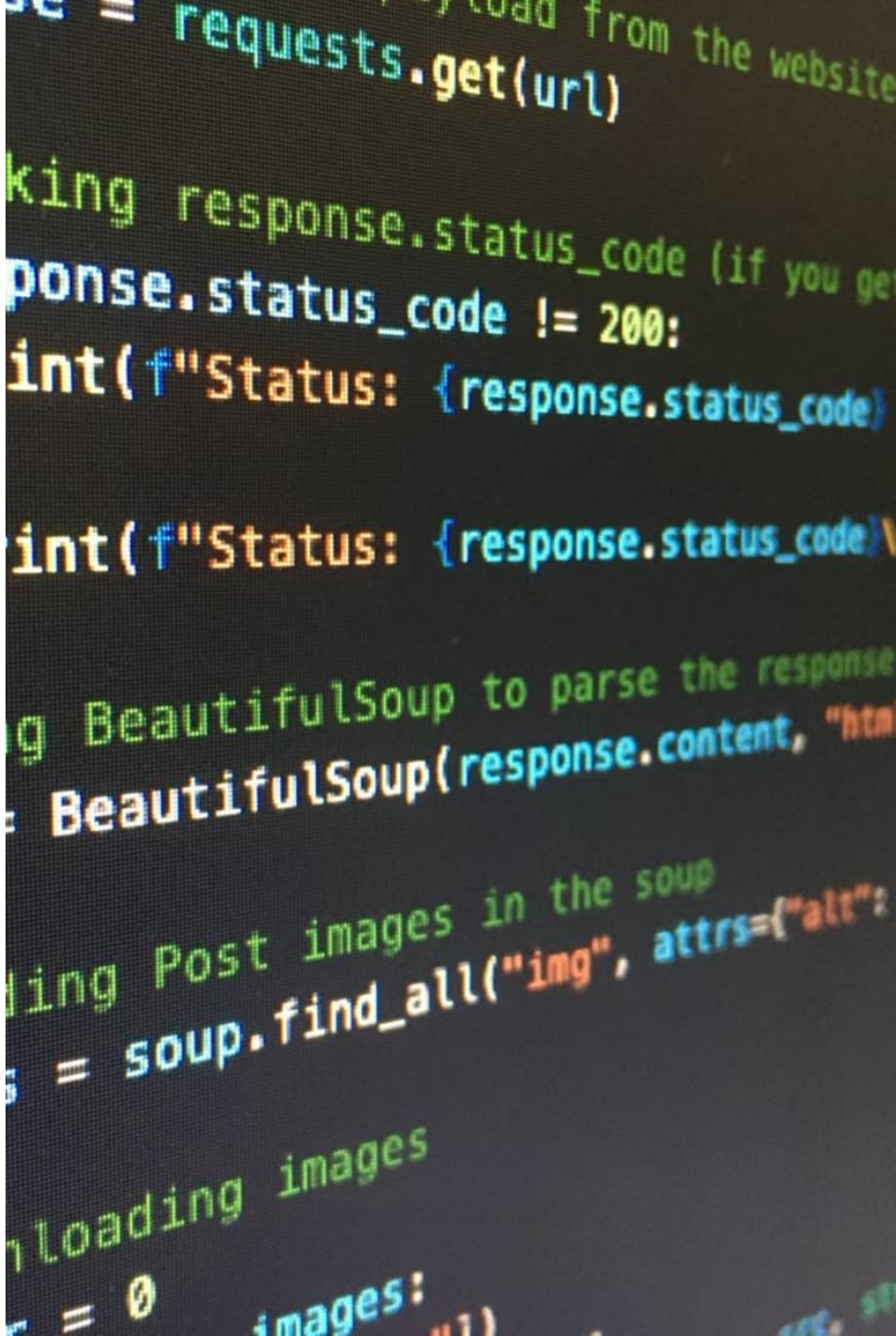
Ejemplos Prácticos

Para comprender mejor el conteo de operaciones, veamos algunos ejemplos concretos:

- "x=2": Una operación de asignación, por lo tanto, T(n)=1.
- "x=y+3": Dos operaciones (suma y asignación), por lo tanto, T(n)=2.
- "return a[0]+1": Tres operaciones (acceso a array, suma y devolución), por lo tanto, T(n)=3.

Cálculo de T(n): Ejemplos Detallados

Código	# Operaciones Primitivas
x = 2	1
x = y + 3	2 (1 suma + 1 asignación)
return a[0] + 1	3 (1 acceso + 1 suma + 1 return)
return node is None or node.elem > 5	5 (1 node is None + 1 obtener node.elem + 1 node.elem >5 + 1 or + 1 return)
print("Hola")	1



Estructuras de Control: Secuencia

Secuencia

Cuando un algoritmo se compone de varios bloques que se ejecutan en secuencia (uno después del otro), su función T(n) es la suma de las funciones T(n) de cada bloque individual.

```
T(n)=T(B1)+T(B2)+...+T(Bn)
```

Ejemplo Ilustrativo

Imaginemos un algoritmo que consta de tres bloques (B1, B2 y B3). Cada bloque tiene una función temporal específica, como T(B1)=2, T(B2)=5 y T(B3)=10. Entonces, la función T(n) del algoritmo completo será:

```
T(n) = 2 + 5 + 10 = 17
```

Estructura de Control: Secuencia - Ejemplo

```
def swap(a,b) # operations
    temp=a 1
    a=b 1
    b=temp 1
```

Este ejemplo muestra una función simple "swap" que intercambia los valores de dos variables. Analicemos las operaciones paso a paso:

- 1: Asignación "temp=a".
- 2: Asignación "a=b".
- 3: Asignación "b=temp".

La función T(n) para esta función es T(n) = 1 + 1 + 1 = 3, ya que se ejecutan 3 operaciones básicas.

Estructuras de Control: Bucles

Bucles (while, for)

Los bucles repiten la ejecución de un bloque de código un número determinado de veces (iteraciones). La función T(n) para un bucle se calcula como el producto de la función T(n) del bloque interno y el número de iteraciones.

```
while condition:
  B
for x in … :
  B
```

```
T loop (n) = T(B) * número de iteraciones
```

Ejemplo con Bucle For

Si un bucle "for" se ejecuta n veces y el bloque interno tiene una función T(n)=2, la función T(n) del bucle será T(n)=2*n.

Estructuras de Control: Condicionales

Condicionales (if-else)

Las estructuras condicionales (if-else) permiten la ejecución de diferentes bloques de código en función de una condición. La función T(n) se define como el máximo de las funciones T(n) de los bloques que se ejecutan.

```
if condition1:
  B1
elif condition2:
  B2 ...
else:
  Bk
```

```
Tif-else (n)= max(TB1 (n), TB2 (n), ..., TBk (n))
```

Interpretación

Esta formulación significa que la función T(n) del bloque condicional se calcula como el máximo de las funciones T(n) de todos los bloques posibles que se ejecuten, ya que solo uno se ejecutará en cada caso.

Estructuras de Control: Condicionales - Ejemplo

```
if opc == "inc":
    n = n + 1    #B1, TB1 (n)= 2
elif opc == "dec":
    n = n - 1    #B2, TB2 (n)= 2
elif opc == "mostrar":
    for i in range(1, n+1): #B3, TB3 (n)= n*1
        print(i)
else:
    print("Error: opc!!") #B4, TB4 (n)= 1
```

Este fragmento de código muestra un ejemplo de la estructura if-else. En este caso, el bloque con la mayor función temporal es B3, que tiene una función T(n)=n. La función T(n) total del algoritmo se calcula como:

```
T(n) = 1 + 1 + 1 + n = n + 3
```

Estructuras de Control: Bucles Anidados

Bucles Anidados

En los bucles anidados, un bucle se encuentra dentro de otro. La función T(n) se calcula multiplicando el número de iteraciones de cada bucle por la función T(n) del bloque interno.

```
for i in range(n):
    for j in range(n):
        print(i*j) #T(n) = 2
```

```
T(n) = n * n * 2 = 2n2
```

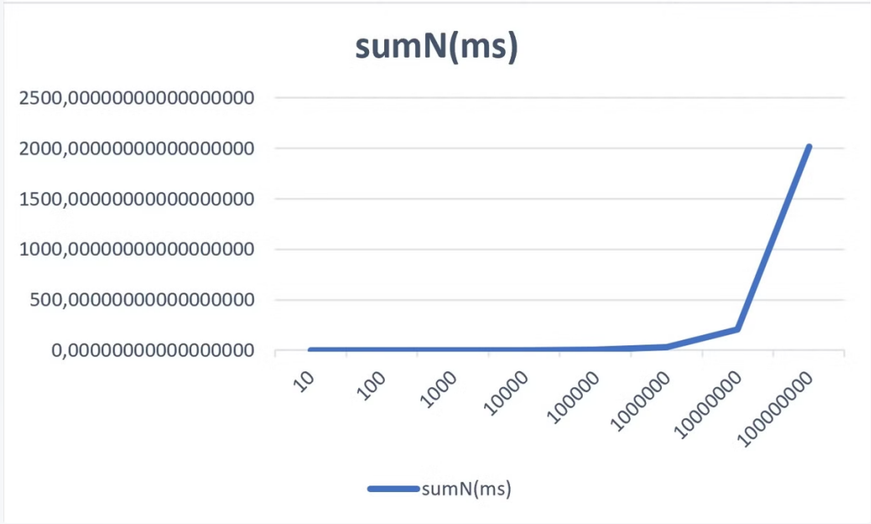
Interpretación

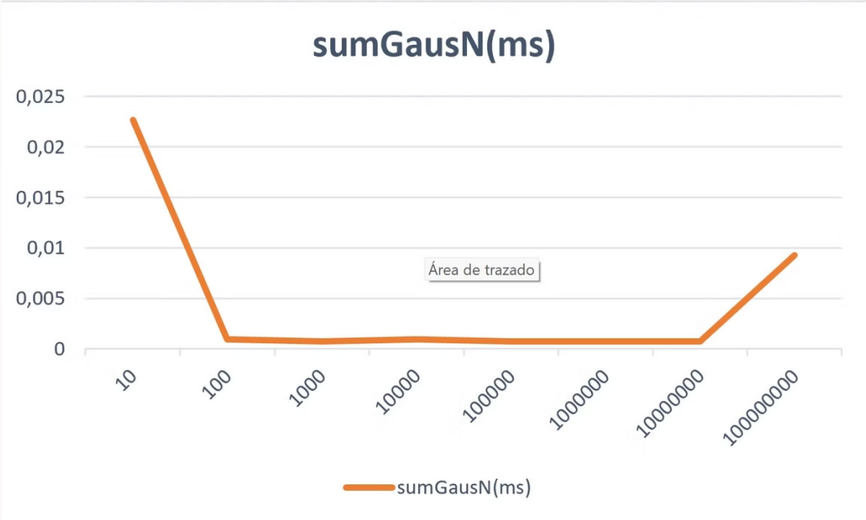
En el ejemplo anterior, el bucle externo se ejecuta n veces, y el bucle interno también se ejecuta n veces para cada iteración del bucle externo. El bloque interno, que realiza 2 operaciones básicas, se ejecuta n*n veces. Por lo tanto, la función T(n) del algoritmo es 2n2.

Ejemplo: Suma n primeros numeros

```
def sumar_numeros(n):
    resultado = 0        #T(n)=1
    for i in range(1, n+1):       #T(n)=2*n
        resultado += i        #T(n)=2
    return resultado      #T(n)=1
```

- T(n) = 1+1+2*n=2n + 2





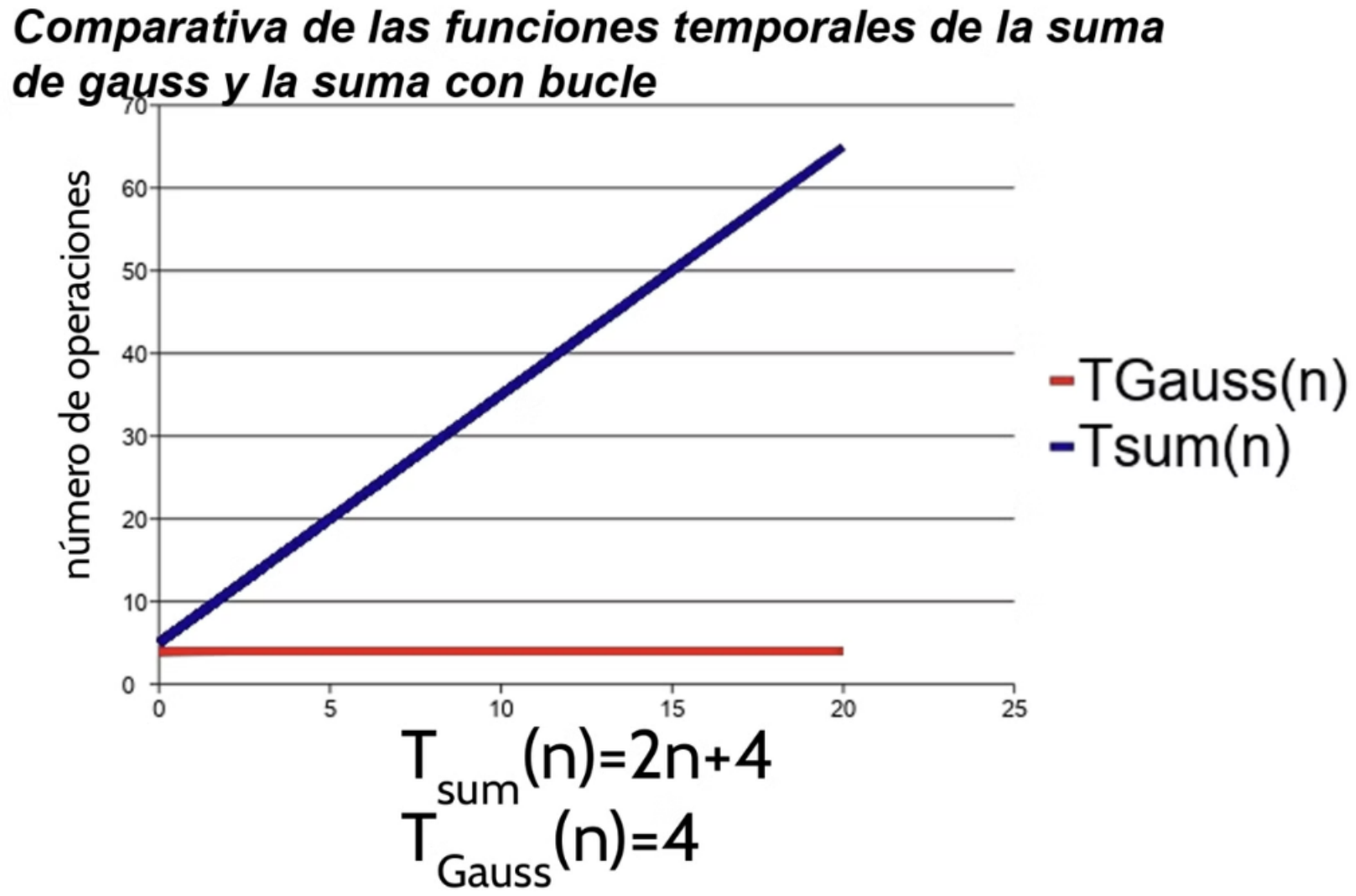
Ejemplo: Suma Gausiana

```
def sumar_numeros(n):

    return n*(n+1)/2        #T(n)=1+1+1+1
```

T(n) = 1+3=4

Comparativa entre ambos algortimos



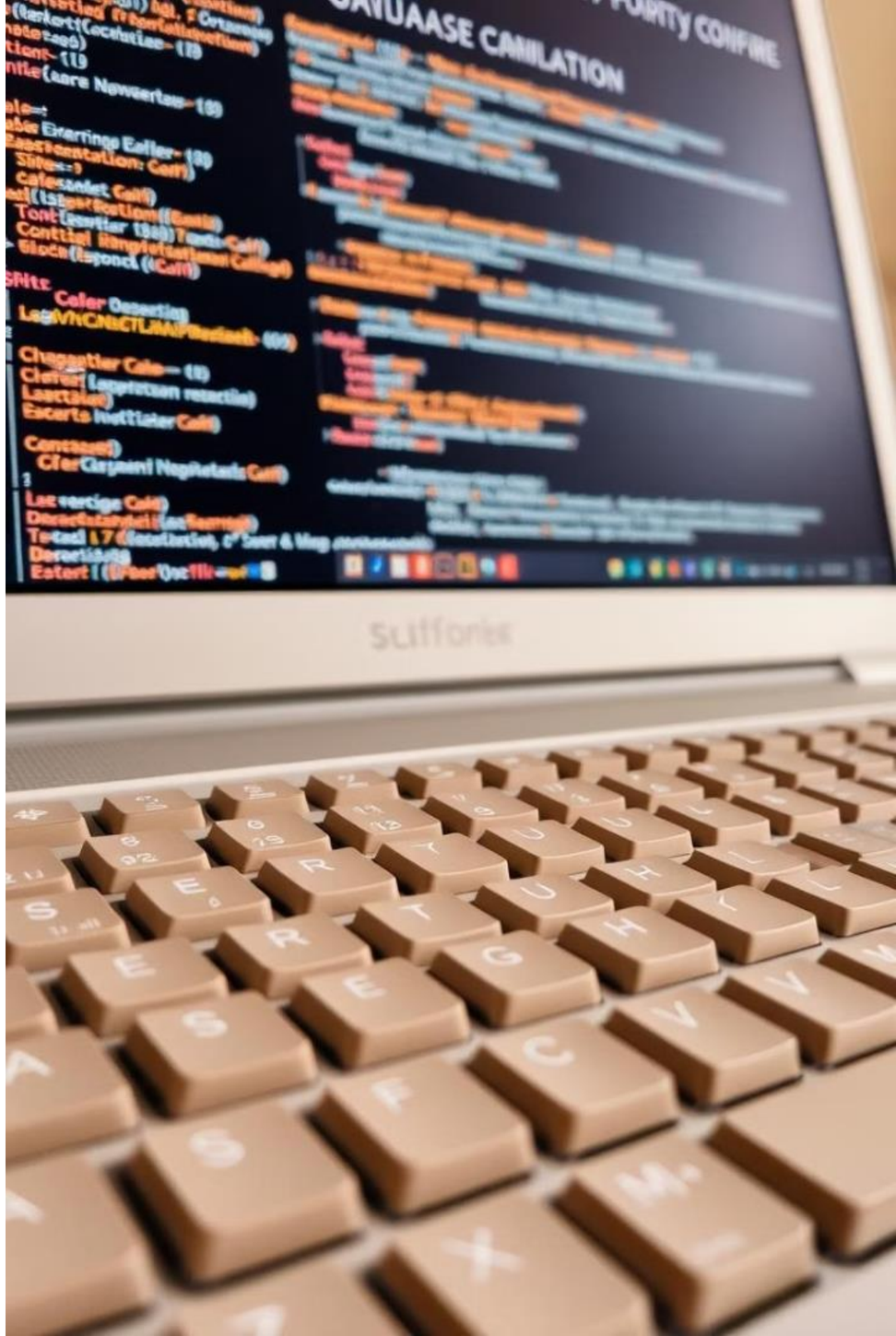
La T(n) de la suma de N con bucle es lineal con n.

La T(n) de Gaus no depende de n. Es constante.

Conclusión

•El análisis teórico permite calcular T(n) sin ejecución.

•Nos ahorra tiempo de implementación ya que no necesitamos hacer la prueba empírica.



Analizamos algoritmos

```
def operaciones_basicas(a, b):
    suma = a + b      #2 operaciones
    resta = a - b     #2 operaciones
    multiplicacion = a * b #2 operaciones

    if b != 0: #1 operaciones
        division = a / b #2 operaciones
    else:
        division = None #1 operaciones

    return (suma, resta, multiplicacion, division) #1
operaciones
```

La T(n)=2+2+2+1+2+1=10

Analizamos algoritmos

```
# Programa para sumar números ingresados por el usuario hasta que ingrese 0
total = 0 #1operacion

print("Ingresa números enteros para sumarlos. Ingresa 0 para finalizar.") #1operacion

while True:
    numero = int(input("Ingresa un número: ")) #1operación n veces
    if numero == 0:  # 1 operación n
        break        #1 operacion
    total += numero  # 2operación ejecuta n veces

print(f"El total acumulado es: {total}") #1 operacion
```

La $T(n)=1+1+n*4+1+1=4n+4$

El mejor caso seria que el usuario ingrese “0” en el primer intento
el peor caso seria que no se ingrese “0” en el primer caso.

Analizamos algoritmos

```
# Programa para calcular la media de 100 números ingresados por el usuario

cantidad_numeros = 100 #1operacion
suma = 0 #1operacion
print(f"Ingresa {cantidad_numeros} números enteros:") #1operacion

for i in range(cantidad_numeros): #n veces n=100
    numero = int(input(f"Ingresa el número {i + 1}: "))
#1operacion
    suma += numero   #2operacion
media = suma / cantidad_números #2operacion

print(f"\nLa media de los números ingresados es: {media}") #1
```

La T(n)=1+1+1+n*3+2+1=3n+6

Analizamos algoritmos

```python
filas = int(input("Ingresa el número de filas: ")) #1operacion
columnas = int(input("Ingresa el número de columnas: ")) #1operacion

matriz = []  # Lista para almacenar la matriz #1operacion
suma_total = 0  # Variable acumuladora #1operacion

for i in range(filas):   #f veces
    fila = []  # Lista para la fila actual #1operacion
    for j in range(columnas): #c veces  4*c
        numero = int(input(f"Ingrese el número para la posición ({i},{j}): ")) #1operacion
        fila.append(numero) #1operacion
        suma_total += numero  # Acumulamos la suma #2operacion
    matriz.append(fila)  # Agregamos la fila a la matriz #1operacion

print("Matriz ingresada:") #1operacion
for fila in matriz: #f veces
    print(fila)  # Mostramos la matriz #1operacion

print(f"La suma total de los elementos es: {suma_total}") #1operacion
```

La T(n)=1+1+1+1+f*(2+4*c)+1+1*f+1=6+4*f*c+3f
Donde c*f<n**2 T(n)=4n**2+3*n+6

No hay mejor caso, se debe recorrer complete el array.