Introducción al Análisis de Algoritmos y Análisis Empírico

1. Introducción al Análisis de Algoritmos

Un **algoritmo** es un conjunto de pasos o instrucciones bien definidas diseñadas para resolver un problema específico. En programación, los algoritmos son la base de cualquier solución, y su eficiencia es crucial para el rendimiento de las aplicaciones.

Características de un buen algoritmo:

- **Correcto:** Debe resolver el problema de manera precisa.
- **Robusto:** Debe manejar situaciones inesperadas sin fallar.
- **Eficiente:** Debe utilizar los recursos de manera óptima, especialmente en términos de tiempo de ejecución y uso de memoria.

¿Por qué analizar algoritmos?

- Un mismo problema puede tener múltiples soluciones (algoritmos).
- Es esencial comparar y seleccionar el algoritmo más eficiente.
- La elección de un algoritmo eficiente puede marcar la diferencia en aplicaciones con grandes volúmenes de datos o alta demanda de procesamiento.

Conceptos clave:

- **Complejidad temporal:** Cuánto tarda un algoritmo en resolver un problema.
- **Complejidad espacial:** Cuánta memoria utiliza un algoritmo.

2. Análisis Empírico

El **análisis empírico** es un enfoque práctico para medir la eficiencia de un algoritmo mediante la observación directa de su tiempo de ejecución. Este método implica implementar el algoritmo y medir cuánto tiempo tarda en resolver un problema para diferentes tamaños de entrada.

Pasos para realizar un análisis empírico:

1. **Implementación del algoritmo:** Escribir el código del algoritmo en un lenguaje de programación (en este caso, Python).
2. **Instrumentación:** Incluir instrucciones para medir el tiempo de ejecución, utilizando funciones como `time.time()` en Python.

3. **Ejecución con diferentes tamaños de entrada:** Ejecutar el programa con entradas de diferentes tamaños y obtener los tiempos de ejecución para cada tamaño.
4. **Visualización de resultados:** Importar los resultados a una hoja de cálculo y crear un gráfico de dispersión (X-Y) para analizar el comportamiento del tiempo de ejecución en función del tamaño de la entrada.

3. Ejemplo Práctico: Suma de los Primeros n Números

Problema: Sumar los primeros **n** números.

Implementación en Python:

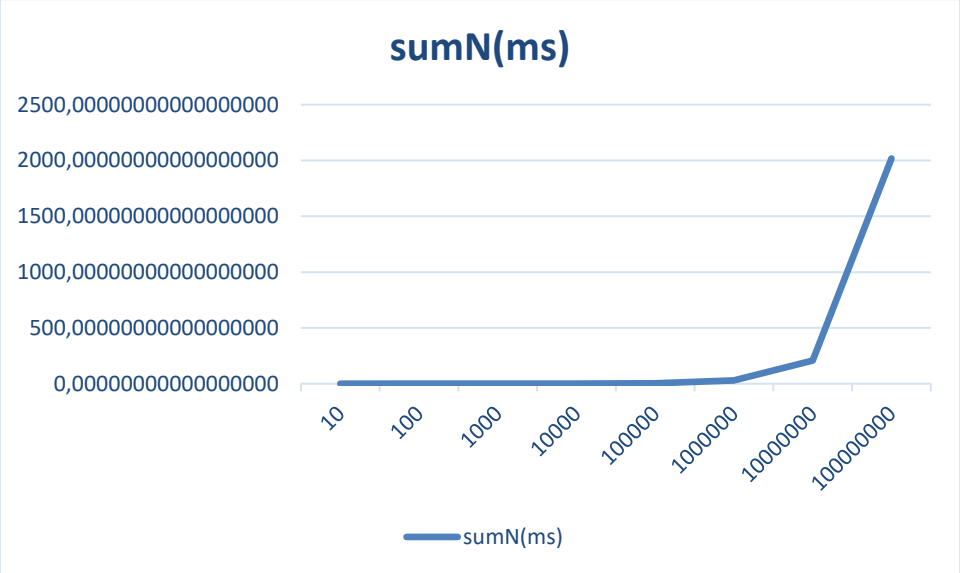
```
 def sumN(n):

    start=time.time()
    res=0
    for i in range(1,n+1):
        res+=i
    end= time.time()
    return res,(end-start)*1000

print("n\t sumN(ms)")
for i in range(1,9):
    _,tiempo_total=sumN(10**i)
    print(f"{10**i}\t {tiempo_total} ")
```

Resultados:

n	sumN(ms)
10	0,00405311584472656
100	0,00977516174316406
1000	0,06818771362304680
10000	0,47564506530761700
100000	4,92787361145019000
1000000	30,53379058837890000
10000000	206,39204978942800000
100000000	2017,73285865783000000

Gráfico de dispersión:

- Eje X: Tamaño de la entrada (n).
- Eje Y: Tiempo de ejecución (segundos).



Conclusión:

- El tiempo de ejecución aumenta linealmente con el tamaño de la entrada.
- El análisis empírico nos permite visualizar el comportamiento del algoritmo.

¿Existen otros algoritmos que resuelvan este problema?

Suma de Gauss:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

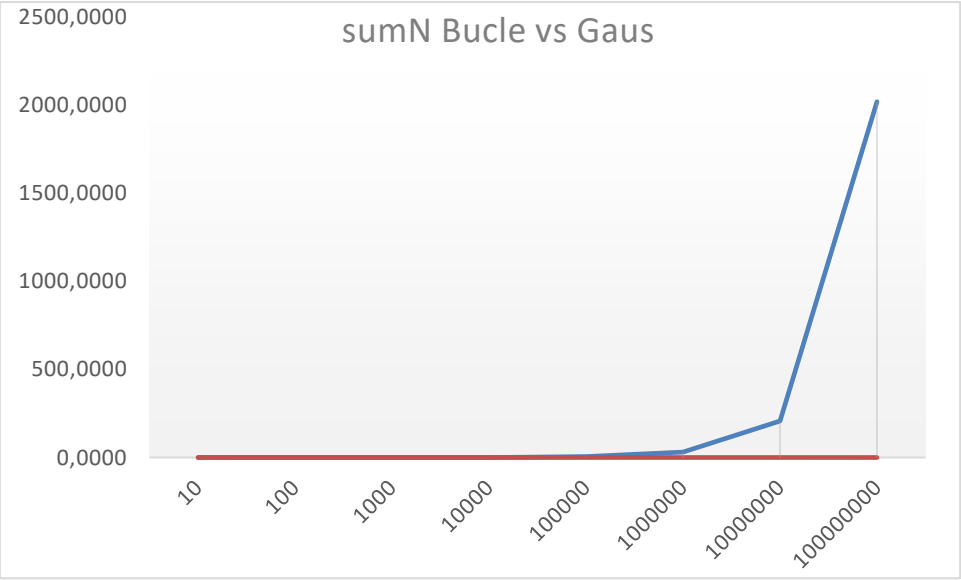
Realicemos su análisis empírico, siguiendo los cuatro pasos anteriores:

```
def sumGausN(n):
```

```
    start=time.time()
    res=n*(n+1)/2
    end=time.time()
    return res,(end-start)*1000

print("n\t sumGausN(ms)")
for i in range(1,9):
    _,tiempo_total=sumGausN(10**i)
    print(f"{10**i}\t {tiempo_total} ")
```

Resultados:

n	sumGausN(ms)
10	0,022649765
100	0,000953674
1000	0,000715256
10000	0,000953674
100000	0,000715256
1000000	0,000715256
10000000	0,000715256
100000000	0,009298325



La gráfica muestra claramente que la solución de Gauss es más eficiente que el primer algoritmo.

4. Ventajas y Desventajas del Análisis Empírico

Ventajas:

- Permite obtener gráficas que muestran el tiempo de ejecución de un algoritmo para diferentes tamaños de entrada.
- Facilita la comparación visual de los tiempos de ejecución de diferentes algoritmos para un mismo problema.

Desventajas:

- Requiere la implementación del algoritmo, lo que consume tiempo y recursos.
- La comparación requiere que los algoritmos se ejecuten en el mismo entorno de hardware y software.
- Los resultados pueden no ser representativos para todas las posibles entradas, ya que solo se evalúan un número finito de ellas.