Análisis de la eficiencia algorítmica en Python

Alumnos:

Lorenzo Pattini – lorenzopattini78@gmail.com
Uriel Palma – palmaurieldev@gmail.com

Materia: Programación I

Profesor: Sebastián Bruselario

Tutora: Virginia Cimino

Fecha de Entrega: 09-06-2025

Introducción

En programación, la eficiencia de un algoritmo se refiere a cómo utiliza los recursos computacionales (tiempo y memoria) para resolver un problema. El análisis de algoritmos nos permite comparar diferentes soluciones y elegir la más óptima según el contexto. Comprender cómo se comportan frente al aumento de datos es esencial para desarrollar aplicaciones eficientes y escalables.

En esta investigación se estudian los conceptos de Eficiencia Temporal (Tiempo) y Espacial (Memoria), se implementan ejemplos prácticos y se presentan las conclusiones.

Marco Teórico

¿Qué es el Análisis de Algoritmos? Es una disciplina de la informática que estudia el rendimiento de los algoritmos, Se enfoca principalmente en dos factores:

- Tiempo de ejecución (Eficiencia en el tiempo iterativo)
- Uso de memoria (Eficiencia de la memoria requerida para los cálculos matemáticos)

Conceptos Clave:

- Complejidad temporal: Se expresa usualmente mediante la notación Big-O, que describe cómo crece el tiempo de ejecución de un algoritmo en función del tamaño de la entrada n. los ejemplos más comunes son:

➢ O(1): Tiempo constante

➢ O(log n): Tiempo logarítmico

➢ O(n): Tiempo lineal

➢ O(n2):Tiempo cuadrático

- Complejidad espacial: Se refiere a la cantidad de memoria adicional que necesita el algoritmo. Ambas funciones evaluadas son de complejidad O(1) en espacio, ya que solo utilizan variables auxiliares sin importar el tamaño de la lista.

Caso Práctico

Se analiza y compara dos algoritmos para calcular la suma de los primeros n números naturales:

Código principal:

```
import time
import random

numeros_0 = [random.randint(1, 1000000) for i in range(1000)]
numeros_1 = [random.randint(1, 1000000) for i in range(10000)]
numeros_2 = [random.randint(1, 1000000) for i in range(100000)]
numeros_3 = [random.randint(1, 1000000) for i in range(1000000)]
numeros_4 = [random.randint(1, 1000000) for i in range(10000000)]
numeros_total = [numeros_0, numeros_1, numeros_2, numeros_3,
numeros_4]

def min_manual(lista):
    if not lista:
        return None
    minimo = lista[0]
    for numero in lista:
        if numero < minimo:
            minimo = numero
    return minimo

def min_auto(lista):
    if not lista:
        return None
    return min(lista)

print("Resultados de la búsqueda del minimo automático")
```

```
for i in range(5):
    inicio = time.time()
    minimo_auto = min_auto(numeros_total[i])
    tiempo = time.time() - inicio
    print(f"{10**(i+3)} {tiempo:.10f}")

print("Resultados de la búsqueda del minimo manual")


for i in range(5):
    inicio = time.time()
    minimo_manual = min_manual(numeros_total[i])
    tiempo = time.time() - inicio
    print(f"{10**(i+3)} {tiempo:.10f}")
```

Resultado Aproximado:

Búsqueda del mínimo automático:

Resultados:

1000 0.0000000000
10000 0.0000000000
100000 0.0010347366
1000000 0.0059986115
10000000 0.0670058727

Búsqueda del mínimo manual:

Resultados:

1000 0.0000000000
10000 0.0000000000
100000 0.0010268688
1000000 0.0139691830
10000000 0.1250259876

Metodología Utilizada

- Análisis teórico de la complejidad de cada algoritmo:
    - Búsqueda manual del número mínimo: O(n) en tiempo, O(1) en espacio.
    - Búsqueda automática con la función min( ): O(n) en tiempo, O(1) en espacio.
- Implementación de los algoritmos en Python.
- Generación de listas de prueba de distintos tamaños (1 000, 10 000, 100 000, 1 000 000, 10 000 000 elementos.)
- Medición práctica usando la función time.time().
- Comparación de resultados de tiempo real entre búsqueda manual y automática.
- Documentación del proceso en repositorio GitHub.

Resultados Obtenidos

- Ambos algoritmos devuelven correctamente el mínimo y reportan el tiempo de ejecución.
- La búsqueda automática con el uso de min() es considerablemente más rápida que la implementación manual.
- El tiempo de ejecución para min_manual crece con el tamaño de la lista y en todo momento fue mayor que el tiempo de min_auto, la brecha se hace notar más a medida que los elementos aumentan.

Conclusiones

El análisis de algoritmos es esencial para optimizar programas, especialmente en aplicaciones con grandes volúmenes de datos. Se ve más reflejado en las grandes empresas, donde encontrar la eficiencia en los algoritmos es un criterio de aceptación de suma importancia.

En el desarrollo de software, la eficiencia de los algoritmos no es un lujo, sino una necesidad crítica. A medida que las aplicaciones manejan volúmenes masivos de datos (big data, inteligencia artificial, sistemas distribuidos), elegir un algoritmo ineficiente puede resultar en:

- Tiempos de ejecución prohibitivos (horas en lugar de segundos).
- Consumo excesivo de memoria (colapsando servidores o dispositivos).
- Mala experiencia de usuario (lentitud en aplicaciones web/móviles).

En los resultados obtenidos vemos reflejado que ambas funciones resuelven correctamente el problema planteado. Sin embargo, los tiempos de ejecución muestran la superioridad en velocidad de la versión automática, sobre todo cuando aumenta el tamaño de los datos, esto se debe principalmente a la optimización interna del lenguaje utilizado.

Como reflexión, este trabajo nos dio la oportunidad de entender que el análisis de datos

implica también evaluar cómo se comportan las soluciones más allá de la teoría, especialmente en entornos donde se manipulan grandes volúmenes de datos.

Anexos

- Captura de resultados de la ejecución de ambos algoritmos y su tiempo de iteración en la terminal:

```
Resultados de la búsqueda del minimo automático
1000 0.0000000000
10000 0.0000000000
100000 0.0010015965
1000000 0.0062534809
10000000 0.0590045452
////////////////////////////////////////////////
Resultados de la búsqueda del minimo manual
1000 0.0009980202
10000 0.0000000000
100000 0.0010004044
1000000 0.0110981464
10000000 0.1110434532
PS C:\Users\Usuario>
```

Gráfico representando el mínimo automatico

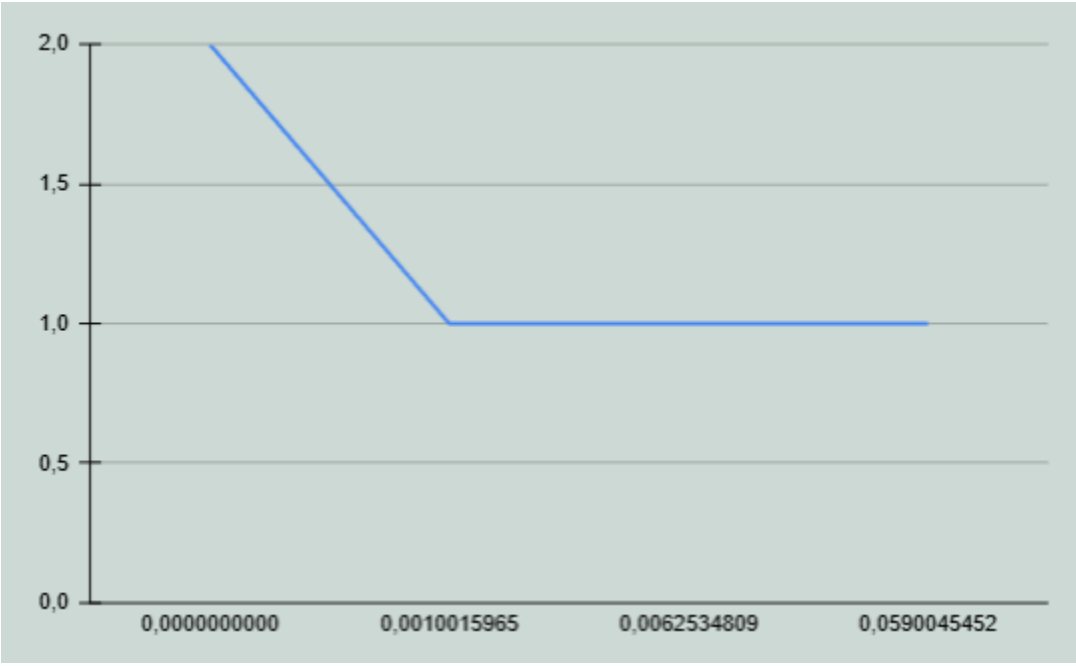
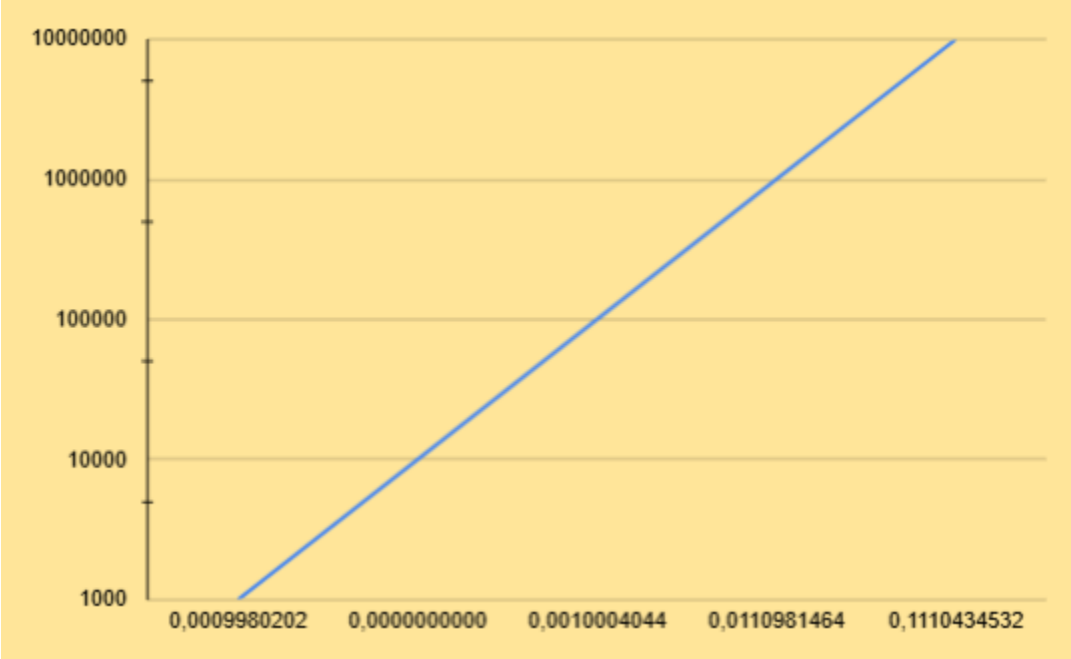


Gráfico representando el mínimo manual



- Repositorio en GitHub: https://github.com/PalmaUriel/Prog1-Integrador-Uriel-Lorenzo

Bibliografía

- Documentación y material extraído de la Unidad: Análisis de algoritmos, TUPAD de la UTN.
- https://www.bigocheatsheet.com

Video explicativo

URL: https://youtu.be/ZYtwa9ePWD0?si=qHgxUmaZ5HW0A6mr