


Carpeta de Investigación: Análisis de Algoritmos en Python

 Carátula

Título del proyecto: Análisis de la eficiencia algorítmica en Python

Alumnos:

Marcos López (marcoslopez@gmail.com)

Sofía Díaz (sofiadiaz@gmail.com)

Materia: Programación I

Profesor: Ing. Laura Fernández

Fecha de Entrega: 5 de mayo de 2025

Introducción

El análisis de algoritmos permite medir y comparar la eficiencia de los programas en términos de tiempo de ejecución y uso de memoria. Comprender cómo se comporta un algoritmo frente al aumento de datos es esencial para desarrollar aplicaciones eficientes y escalables.

En esta investigación se estudian los conceptos de eficiencia temporal y espacial, se implementan ejemplos prácticos y se presentan las conclusiones.

Marco Teórico

¿Qué es el Análisis de Algoritmos? Es el estudio formal del rendimiento de los algoritmos, buscando optimizar:

- Tiempo de ejecución (Eficiencia Temporal)
- Uso de memoria (Eficiencia Espacial)

Conceptos Clave:

- Notación Big-O: Describe el peor caso de crecimiento del algoritmo (por ejemplo: $O(n)$, $O(\log n)$, $O(n^2)$).
- Tiempo de ejecución real: Se puede medir usando módulos como time en Python.
- Complejidad espacial: Se refiere a la cantidad de memoria adicional que utiliza el algoritmo.

Caso Práctico

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

Código principal (analisis_algoritmos.py):

```
import time
```

```
# Algoritmo 1: Suma iterativa
```

```
def suma_iterativa(n):
```

```
    total = 0
```

```
    for i in range(1, n+1):
```

```
        total += i
```

```
    return total
```

```
# Algoritmo 2: Fórmula matemática
```

```
def suma_formula(n):
```

```
    return n * (n + 1) // 2
```

```
def medir_tiempo(funcion, n):
```

```
    inicio = time.time()
```

```
    resultado = funcion(n)
```

```
    fin = time.time()
```

```
    return resultado, fin - inicio
```

```
if __name__ == "__main__":
```

```
    n = 10_000_000
```

```
    resultado1, tiempo1 = medir_tiempo(suma_iterativa, n)
```

```
resultado2, tiempo2 = medir_tiempo(suma_formula, n)
```

```
print(f"Suma Iterativa: Resultado={resultado1}, Tiempo={tiempo1:.8f} segundos")
```

```
print(f"Suma Fórmula: Resultado={resultado2}, Tiempo={tiempo2:.8f} segundos")
```

Resultado esperado (salida aproximada):

Suma Iterativa: Resultado=50000005000000, Tiempo=0.76234567 segundos

Suma Fórmula: Resultado=50000005000000, Tiempo=0.00000123 segundos

4. Metodología Utilizada

- Análisis teórico de la complejidad de cada algoritmo:
 - Suma iterativa: $O(n)$ en tiempo, $O(1)$ en espacio.
 - Suma por fórmula: $O(1)$ en tiempo y espacio.
- Implementación de los algoritmos en Python.
- Medición práctica usando la función `time.time()`.
- Comparación de resultados de tiempo real.
- Documentación del proceso en repositorio GitHub.

Resultados Obtenidos

- Ambos algoritmos devuelven el mismo resultado correcto.
- La fórmula matemática es mucho más rápida que la iterativa para valores grandes de n .
- El tiempo de ejecución para $n = 10.000.000$ en la suma iterativa fue cientos de veces mayor que usando la fórmula.

Conclusiones

El análisis de algoritmos permite elegir mejores soluciones en función de la cantidad de datos. Aunque dos algoritmos resuelvan el mismo problema, su eficiencia puede ser muy diferente. Python, aunque es un lenguaje interpretado, permite aplicar técnicas de análisis de algoritmos que son esenciales para cualquier tipo de desarrollo serio.

Recomendación: Siempre analizar la complejidad de los algoritmos antes de implementarlos en proyectos que manejarán grandes volúmenes de datos.

Bibliografía

- Grokking Algorithms, Aditya Bhargava
- Documentación oficial Python time: <https://docs.python.org/3/library/time.html>
- Big O Cheat Sheet: <https://www.bigocheatsheet.com/>

Anexos

- Captura de resultados de la ejecución: (insertar imagen de terminal mostrando tiempos de ejecución)

Repositorio en GitHub: <https://github.com/grupo-analisis-algoritmos/analisis-python>

Video explicativo

(enlace a YouTube o Drive mostrando el código y la medición de tiempos)



Notas de entrega

El repositorio debe tener:

- Código de los algoritmos.
- Archivo de README explicativo.

El video debe incluir:

- Introducción breve sobre análisis de algoritmos.
- Demostración práctica de los tiempos de ejecución.
- Reflexión grupal sobre el aprendizaje del análisis de eficiencia.