TRABAJO DE INVESTIGACIÓN

Análisis de Algoritmos en Python

Título del Proyecto:

Análisis del rendimiento de los algoritmos en Python

Presentado por:

- Francisco Meier (franmeier16@gmail.com)
- Valentin Mattoni (valentinmattoni@gmail.com)

Materia:

Programación I

Profesor: Bruselario, Sebastián.

Tutora: Cimino, Virginia.

Fecha de Entrega:

20 de Junio de 2025

Índice

Introducción

El análisis de algoritmos nos permite conocer el tiempo de ejecución y el espacio en memoria que requiere un algoritmo, lo cual es esencial para evaluar su rendimiento al compararlo con otras soluciones, considerando tanto su velocidad como su consumo de recursos. Entender cómo se comporta con diferentes cantidades de datos nos permite optimizar su eficiencia y asegurar su capacidad de escalabilidad.

En el presente trabajo de investigación abordaremos los conceptos de eficiencia temporal y espacial, y su implementación en casos prácticos y conclusión.

Marco Teórico

¿Qué es un algoritmo?

Es una serie de pasos lógicos que nos permiten resolver un problema

¿Qué es el análisis de algoritmos?

El análisis de algoritmos es una parte fundamental de la informática que supone evaluar el rendimiento de algoritmos y programas. La eficiencia se mide en términos de:

1. Tiempo de ejecución (eficiencia temporal)
2. Uso de memoria (eficiencia espacial)

Conceptos importantes:

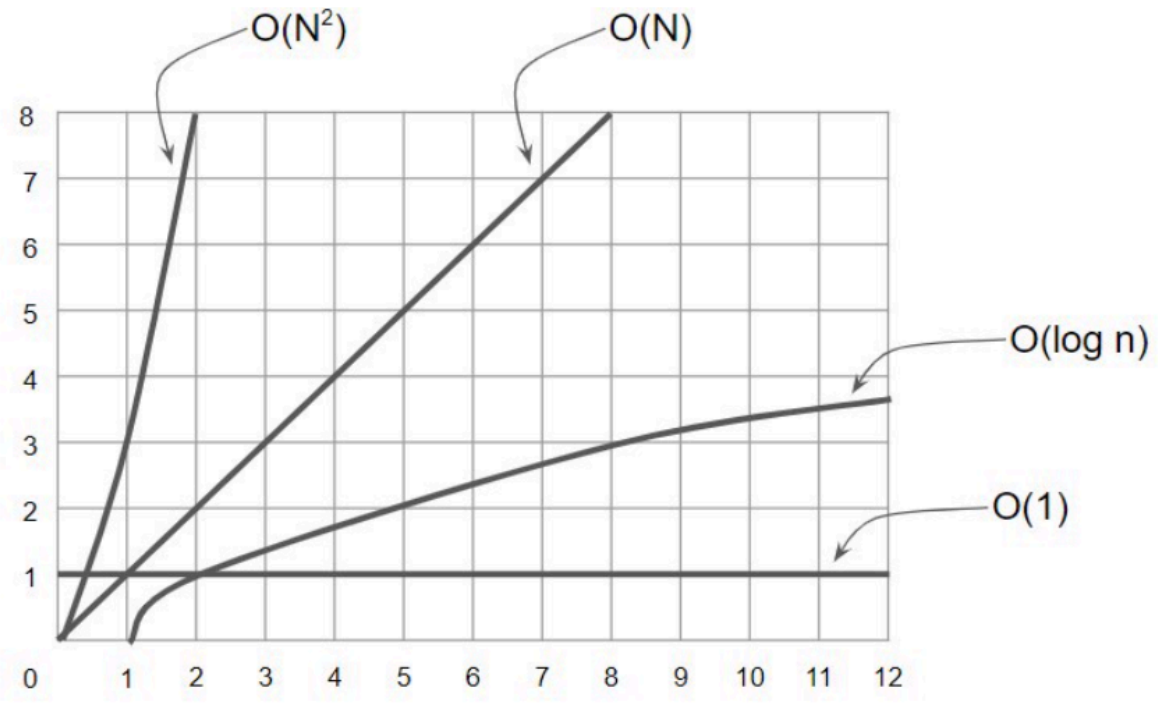
Recursividad: La recursividad es cuando una función se llama a sí misma directa o indirectamente para resolver un problema.

Notación Big O: nos muestra la eficiencia del peor de los casos de un algoritmo. Para determinar el rendimiento del algoritmo, se tiene en cuenta la eficiencia temporal y espacial.

Eficiencia Temporal: la eficiencia temporal especifica cuánto tiempo llevará ejecutar un algoritmo en función de su tamaño de entrada.

Eficiencia Espacial: la eficiencia espacial de un algoritmo especifica la cantidad total de espacio o memoria necesaria para ejecutar un algoritmo en función del tamaño de la entrada.

Gráfico notación big O



Este gráfico muestra cómo responde un algoritmo al aumentar los datos: si la curva sube suavemente, el algoritmo mantiene buen rendimiento con mucha información; si se dispara abruptamente, significa que se vuelve ineficiente con grandes volúmenes. Por ejemplo, buscar en un diccionario organizado sigue siendo rápido aunque tenga miles de palabras, mientras que revisar una lista desordenada se hace cada vez más lento al agregar más elementos. La forma de la curva revela si el método es escalable o si colapsa ante grandes cantidades de datos, permitiendo elegir la mejor opción según las necesidades del problema.

En resumen:

- O(1): El más rápido (siempre igual).
- O(log n): Casi igual de rápido.
- O(N): Tarda más si hay más datos.
- O(N²): Se vuelve muy lento con muchos datos.

Cuanto más alta la línea, más lento el algoritmo. O(1) es el mejor, O(N²) el menos eficiente.

Caso Práctico

```
import time


def fibonacci_recursivo(n):
    if n <= 1:
        return n
    return fibonacci_recursivo(n-1) + fibonacci_recursivo(n-2)


def fibonacci_iterativo(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a


def medir_tiempo(funcion, n):
    inicio = time.time()
    resultado = funcion(n)
    fin = time.time()
    return resultado, fin - inicio


if __name__ == "__main__":
    n = 42
    resultado1, tiempo1 = medir_tiempo(fibonacci_iterativo, n)
    resultado2, tiempo2 = medir_tiempo(fibonacci_recursivo, n)
    print(f"Fibonacci Iterativo: Resultado={resultado1}, Tiempo={tiempo1:.8f} segundos")
    print(f"Fibonacci Recursivo: Resultado={resultado2}, Tiempo={tiempo2:.8f}
segundos")
```

4.Metodología Utilizada

- Realizacion del Analisis teorico de la complejidad de los dos algoritmos:
    - Fibonacci iterativo: O(n) en tiempo y O(1) en espacio.
    - Fibonacci recursivo: $O(2^n)$ en tiempo y O(n) en 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.
- El algoritmo iterativo de fibonacci es notablemente más rápido que el recursivo para valores grandes de n. La fórmula recursiva tarda cientos o miles de veces más que la iterativa debido a su complejidad exponencial($O(2^n)$) frente a la lineal(O(n)) del método iterativo.
- El tiempo de ejecución para n = 42 en fibonacci recursivo fue cientos o incluso miles de veces mayor que con el método iterativo

Conclusión

El análisis de algoritmos nos permite evaluar el rendimiento de los algoritmos considerando dos factores importantes: el tiempo de ejecución y el consumo de memoria. Esta evaluación resulta sustancial cuando existen múltiples algoritmos capaces de resolver un mismo problema, ya que nos brinda criterios para seleccionar la opción más eficiente y adecuada a nuestros requerimientos.

Fuentes

1. GeeksforGeeks. (s.f.). *Analysis of Algorithms*. Recuperado el [fecha de acceso], de https://www.geeksforgeeks.org/analysis-of-algorithms/
2. GeeksforGeeks. (s.f.). *What is Algorithm and Why Analysis of it is Important?* Recuperado el [fecha de acceso], de https://www.geeksforgeeks.org/what-is-algorithm-and-why-analysis-of-it-is-important/
3. freeCodeCamp. (s.f.). *Hoja de trucos Big O*. Recuperado el [fecha de acceso], de https://www.freecodecamp.org/espanol/news/hoja-de-trucos-big-o/#:~:text=%C2%BFQu%C3%A9%20es%20la%20complejidad%20del,del%20tiempo%20ara%20cualquier%20algoritmo