Trabajo Integrador

Programación 1

Análisis de Algoritmos

Alumnos :

Martin Monzon Rubano, martinmonzonfacultad@gmail.com

Christian Emmanuel Olivero, christian.97.olivero@gmail.com

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación

Profesor Coordinador: Alberto Cortez

Profesor a cargo comisión:

Cinthia Rigoni

Julieta Trapé

Nicolas Quirós Ricci

Sebastián Bruselario

Ariel Enferrel

9 de Junio de 2025

Tabla de contenido

Introducción

El análisis de algoritmos es una herramienta clave para entender cómo se comportan nuestras soluciones ante distintos volúmenes de datos. Más allá de que un algoritmo funcione correctamente, es fundamental poder evaluar su eficiencia y saber cuánto tiempo o recursos puede llegar a consumir. En este trabajo, trabajaremos con los fundamentos del análisis de algoritmos, aprendiendo a expresar funciones de tiempo, a identificar su crecimiento y a usar la notación Big-O para clasificar distintas soluciones.

Además, comparamos algoritmos que resuelven un mismo problema para poder elegir cuál es la opción más eficiente según el contexto. Esta capacidad de análisis no solo mejora el rendimiento de los programas, sino que también nos ayuda a pensar de forma más crítica y ordenada a la hora de diseñar soluciones.

Marco Teórico

Algoritmo: Un algoritmo es un conjunto de pasos claros y ordenados que sirven para resolver un problema específico. Para que sea útil, tiene que funcionar bien, ser eficiente y lo suficientemente robusto como para manejar distintos tipos de situaciones. Los algoritmos son la base de toda la programación, ya que detrás de cada programa hay una serie de instrucciones que siguen una lógica para llegar a un resultado.

Análisis de algoritmos: Nos permite elegir el algoritmo más adecuado para cada situación, comparando la eficiencia de distintas soluciones para un mismo problema

Análisis empírico: Mide el tiempo de ejecución de un algoritmo en la práctica, registrando el tiempo que toma completar cada ejecución. Para esto se ejecutan pruebas con diferentes tamaños de entrada para ver en qué tiempo soluciona según el tamaño.

Análisis Teórico: El análisis teórico nos permite evaluar qué tan eficiente es un algoritmo sin tener que programarlo y probarlo. En lugar de eso, analizamos su estructura a partir del pseudocódigo y calculamos una función, llamada T(n), que nos dice aproximadamente cuántas operaciones realiza el algoritmo según el tamaño de la entrada.

Notación Big O: La notación Big-O se usa para describir cómo crece una función cuando el tamaño de la entrada se hace muy grande. Básicamente, nos ayuda a entender qué tan rápido crece el tiempo de ejecución de un algoritmo. Lo bueno de esta notación es que ignora las constantes y los detalles chicos, y se enfoca solo en lo que más influye cuando los datos son muchos, lo que la hace ideal para comparar algoritmos entre sí.

Caso Práctico

Hemos elegido el siguiente enunciado de un programa entregado en unidades anteriores para hacer un " Análisis de Algoritmo".

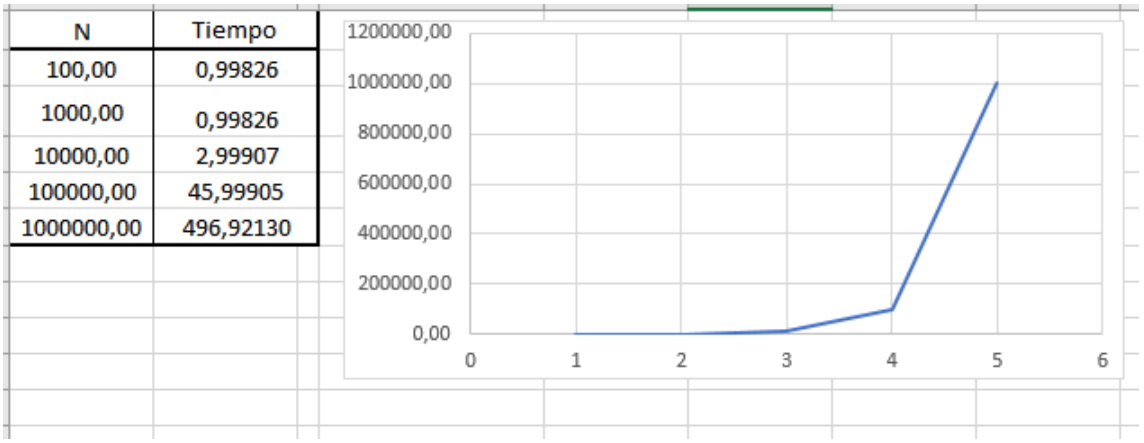
"Elabora un programa que permita al usuario ingresar 100 números enteros y luego calcule la media de esos valores"

En este caso hemos creado una función que calcule la media para tener un registro de salida más limpio y ordenado, importamos y usamos la función time para registrar el tiempo de ejecución y para tener los valores de salida sólo llamamos la funcion "sacar_Media (n) " para distintos valores de n que en este caso los hemos probado para : 100, 1000, 10000, 100000, 1000000, obteniendo los resultados respectivos

```python
import random
import time
def sacar_Media(n):

    inicio = time.time()                                    #Tiempo Final

    numeros = list(range(n))
    suma = 0

    for i in range(n):
        suma += numeros[i]

    media = suma / n
    print(f"La media de los numeros es {media}")

    fin = time.time()                                       #Tiempo final

    tiempo_total = (fin - inicio) * 1000
    print(f"El tiempo de ejecución fue {tiempo_total:.6f} milisegundos")
    print("-------------------")

    return media

sacar_Media(100)
sacar_Media(1000)
sacar_Media(10000)
sacar_Media(100000)
sacar_Media(1000000)
```

```
La media de los numeros es 49.5
El tiempo de ejecución fue 0.994682 milisegundos
------------------
La media de los numeros es 499.5
El tiempo de ejecución fue 1.002550 milisegundos
------------------
La media de los numeros es 4999.5
El tiempo de ejecución fue 5.023003 milisegundos
------------------
La media de los numeros es 49999.5
El tiempo de ejecución fue 225.983143 milisegundos
------------------
La media de los numeros es 499999.5
El tiempo de ejecución fue 1547.567368 milisegundos
------------------
```

Para el análisis del algoritmo solo nos vamos a basar en el tiempo utilizado para el desarrollo del mismo que está expresado en milisegundos.
Con estos datos armamos una planilla de excel y generamos un gráfico donde encontramos el comportamiento lineal del algoritmo de manera experimental o empírica, esto quiere decir que a medida que aumentamos la entrada “N” que es la cantidad de números para determinar la media, mayor es el tiempo en calcular dicho valor.



N	Tiempo
100,00	0,99826
1000,00	0,99826
10000,00	2,99907
100000,00	45,99905
1000000,00	496,92130

Análisis teórico

A Continuación detallamos en cada línea de comando la cantidad de operaciones necesitadas para construir la función T(n) para que luego determinamos por medio de la notación Big-O el comportamiento del problema

```python
import random
import time
def sacar_Media(n):
    numeros = list(range(n))    # 3 operacion
    suma = 0                    # 1 operacion
    for i in range(n):          # 2*n operacion
        suma += numeros[i]
    media = suma / n            # 2 operaciones
    print(f"La media de los numeros es {media}")  # 1 operacion
    return media                                  # 1 operacion

# Total: 3 + 1 + 2*n + 2 + 1 + 1
#         = 2*n + 8 ->  O(n)
#         -> Comportamiento lienal
```

Entonces por medio del análisis teórico y haciendo uso de la notación Big-O determinamos que el algoritmo se comporta de manera lineal

Metodología Utilizada

1. Selección del algoritmo a analizar.
2. Modificación del algoritmo para lograr una mejor visualización de los conceptos a utilizar.
3. Hacer el análisis empírico.
4. Trasladar los resultados registrados de tiempo en python a un archivo excel.
5. Generar la tabla de datos y crear el gráfico.
6. Determinar y calcular en el código la cantidad de procesos para hacer un análisis teórico.
7. Comparar resultados

Librerías utilizadas:
-random:es un módulo de la biblioteca estándar de Python que proporciona funciones para generar números aleatorios.
-time:módulo de la biblioteca estándar, y sirve para trabajar con el tiempo (esperas, medir duración, obtener el tiempo actual, etc.).

Resultados Obtenidos

- Se realizaron con éxito ambas pruebas (teórico y empírico) y se pudo llegar a la misma conclusión sobre el comportamiento del algoritmo.
- Se aprendió e incorporó los conceptos sobre análisis de rendimiento de algoritmos para un análisis más completo e integral sobre los mismos.

Conclusión

El algoritmo desarrollado para calcular la media de n números enteros muestra un comportamiento lineal tanto en el análisis teórico como en la ejecución práctica.

Desde el punto de vista teórico, determinamos que el número total de operaciones es proporcional a 2n + 8, lo que se simplifica como $T(n) \in O(n)$. Esto significa que el tiempo de ejecución crece linealmente a medida que aumenta la cantidad de números procesados.

En el análisis empírico, al ejecutar la función con distintos valores de n (100, 1000, 10000, etc.), observamos que el tiempo de ejecución también crece de forma aproximadamente lineal. Por ejemplo, al aumentar el tamaño de entrada por 10 veces, el tiempo de ejecución también aumenta aproximadamente 10 veces (con ligeras variaciones naturales del sistema).

Por lo tanto, podemos concluir que el algoritmo tiene una eficiencia adecuada para manejar grandes volúmenes de datos y su complejidad es lineal, es decir, O(n). Esto lo hace escalable y predecible en términos de rendimiento conforme crece el tamaño de la entrada.

Bibliografía:

Ariel Enferrer. Análisis Teórico y Notación Big-O. Pdf

Ariel Enferrer. Análisis de Algoritmos empíricos. Pdf

Ariel Enferrer. Estructuras de control en análisis teórico. Youtube. URL: https://www.youtube.com/watch?v=AJOiiMo8IVE

Ariel Enferrer. Introducción al Big-O. Youtube. URL: https://www.youtube.com/watch?v=j8PqpS21Gpo

Ariel Enferrer. Análisis empírico sumN. Youtube. URL: https://www.youtube.com/watch?v=ptsijUkV0iI

Anexos:

Captura de pantalla de python con el código

Captura de pantalla de excel con el análisis

Repositorio Github: https://github.com/CHRISTIAN2320/Trabajo_Integrador_Programacion1/blob/main/README.md

Link video presentación del proyecto en YouTube:

https://www.youtube.com/watch?v=LJMDfPrIv0w