Datos Generales

- **Título del trabajo**: Análisis de Algoritmos
- **Alumnos**: Verónica Analia Calcerano veronica_calcerano@hotmail.com / Gianina Azcurra (gianinazcurra@hotmail.com)
- **Materia**: Programación I
- **Profesor/a**: Rigoni Cinthia
- **Fecha de Entrega**: 08/06/2025

Índice

1. Introducción
2. Marco Teórico
3. Caso Práctico
4. Metodología Utilizada
5. Resultados Obtenidos
6. Conclusiones
7. Bibliografía
8. Anexos

1. Introducción

El presente trabajo tiene como finalidad analizar y comparar el comportamiento de dos algoritmos que calculan la media aritmética de una lista de números enteros. Se eligió este tema por su utilidad didáctica en el estudio de estructuras de programación fundamentales como bucles, listas, funciones y medición de tiempos de ejecución, contenidos abordados a lo largo de la materia Programación I. A su vez, permite introducir conceptos claves del análisis algorítmico, como la **complejidad computacional** y la notación **Big O**, esenciales para evaluar la eficiencia de un algoritmo a medida que crece la cantidad de datos de entrada.

La importancia del tema radica en que el cálculo de una media es una operación básica pero frecuente en todo tipo de aplicaciones reales, desde estadísticas simples hasta análisis de grandes volúmenes de datos. Evaluar la eficiencia con que se realiza esa operación permite entender cómo diferentes enfoques de programación pueden impactar en el rendimiento general de un sistema.

El objetivo principal de este trabajo es implementar y comparar dos formas de calcular la media de una lista de números: una usando una estructura iterativa que acumula los valores uno a uno, y otra utilizando la función incorporada sum() de Python. Se busca observar cómo se comporta cada algoritmo frente a listas de distinto tamaño y analizar las diferencias de tiempo utilizando herramientas básicas de medición. A través de este caso práctico, se pretende reforzar la comprensión de las estructuras secuenciales y repetitivas, el uso de funciones, y su relación con el rendimiento y la escalabilidad de los programas.

2. Marco Teórico

El análisis de algoritmos es una disciplina fundamental dentro de la programación, ya que permite evaluar la eficiencia de una solución ante diferentes volúmenes de datos. En este trabajo se comparan dos formas de calcular la **media aritmética**, utilizando estructuras de control básicas en Python.

Estructuras secuenciales, repetitivas y condicionales

Durante el desarrollo del trabajo se utilizan las estructuras vistas en la materia:

- **Secuenciales**, cuando las instrucciones se ejecutan una tras otra.
- **Repetitivas**, como el bucle for, que permite iterar sobre una secuencia de elementos.
- **Funciones**, que permiten encapsular bloques de código y reutilizarlos, favoreciendo la modularidad.
- **Listas**, estructuras de datos que almacenan múltiples valores y sobre las cuales se aplican operaciones como sum(), recorridos y cálculos.

Cálculo de la media

La **media aritmética** de un conjunto de n valores numéricos se define como:

$$media = suma\ de\ todos\ los\ elementos\ n \backslash cantidad\ de\ elementos$$

En Python, esto puede implementarse de dos formas:

- Utilizando una estructura iterativa (for) para sumar elemento por elemento.
- Usando la función incorporada sum(lista), que realiza la operación internamente.

Complejidad algorítmica y notación Big O

Un aspecto fundamental en el análisis de algoritmos es la **complejidad temporal**, es decir, cuánto tiempo requiere un algoritmo en función del tamaño de los datos de entrada. Esto se representa con la **notación Big O**. En este caso:

- Ambos algoritmos (el iterativo y el que utiliza sum) presentan una **complejidad lineal**, O(n), ya que recorren todos los elementos de la lista una vez.

Según *Downey (2012),* en su libro *"Think Python"*, cuando dos algoritmos tienen la misma complejidad asintótica, se comportan de forma similar a medida que la entrada crece. No obstante, pueden existir diferencias prácticas debido a la implementación.

3. Caso Práctico

Descripción del problema

El objetivo del caso práctico es comparar dos algoritmos que calculan la **media aritmética** de una lista de números enteros generados aleatoriamente. Se busca observar las diferencias en los tiempos de ejecución al aumentar la cantidad de datos de entrada, evaluando así la

eficiencia de cada enfoque. Esta comparación permite aplicar conceptos como estructuras iterativas, funciones, listas, y análisis algorítmico (complejidad temporal y notación Big O).

Decisiones de diseño

Se implementaron dos algoritmos para comparar rendimiento:

- Se eligió el bucle for para el primer algoritmo porque permite analizar explícitamente la estructura repetitiva.
- El segundo enfoque utiliza la función sum() de Python, más eficiente porque está optimizada en C a bajo nivel.
- Se utilizaron funciones (def) para modularizar el código y facilitar su lectura y reutilización.
- Se eligió la estructura if __name__ == "__main__": para establecer el punto de entrada del programa.
- Se aplicó el módulo random para generar datos y el módulo time para medir el rendimiento, herramientas estándar en Python.

Validación del funcionamiento

El programa fue ejecutado con listas de distintos tamaños, desde 10 hasta 100 millones de elementos. Se comprobó que ambos algoritmos arrojaban el mismo valor promedio, lo cual valida la **corrección lógica**. Además, se observó que el uso de sum() resultaba **más rápido**, especialmente con volúmenes grandes de datos, confirmando lo analizado en el marco teórico.

4. Metodología Utilizada

El desarrollo del presente trabajo se realizó siguiendo una metodología estructurada que combinó investigación teórica y experimentación práctica. A continuación, se detallan las etapas del proceso:

Investigación previa

En una primera instancia, se realizó una revisión de los contenidos trabajados en la asignatura **Programación I**, con especial énfasis en:

- **Estructuras de control secuenciales, condicionales y repetitivas**.
- **Listas y funciones en Python**.

- Conceptos teóricos como la **Notación Big O**, la **Sumatoria de Gauss** y la **eficiencia algorítmica**.

Se consultaron las siguientes fuentes:

- Apuntes y videos provistos por la cátedra.
- El libro *"Pensar en Python"* de Allen B. Downey.
- Documentación oficial de Python (docs.python.org) para comprender el comportamiento de funciones como sum() y los módulos time y random.

Diseño y pruebas

Luego de comprender los conceptos teóricos, se procedió al diseño de dos algoritmos simples para calcular la **media aritmética** de una lista de números:

- El primer algoritmo utiliza una **estructura repetitiva for** para recorrer la lista y realizar una suma acumulativa.
- El segundo algoritmo emplea la función sum() de Python, que realiza la misma operación, pero de forma optimizada.

Se realizaron dos funciones diferentes para cada algoritmo, y se creó una función adicional para **medir el tiempo de ejecución** de cada uno, utilizando el módulo time. Para probarlos, se generaron listas con distintos volúmenes de datos, desde 10^1 hasta 10^8, mediante el módulo random.

Herramientas y recursos

Durante el desarrollo se utilizaron las siguientes herramientas:

- **Lenguaje de programación**: Python 3.x.
- **IDE**: Visual Studio Code.
- **Módulos estándar**: time para medición de tiempos y random para generación de datos aleatorios.

- **Control de versiones**: Se utilizó almacenamiento local y GitHub para versiones posteriores del proyecto.

El trabajo se realizó en grupo. Ambas integrantes participaron de manera conjunta en todas las etapas del desarrollo: el diseño del código, la investigación y redacción del marco teórico, el análisis de resultados y la preparación de la presentación final. Para facilitar la organización y el seguimiento del trabajo, se utilizó una carpeta compartida donde se centralizaron los avances, correcciones y materiales complementarios.

5. Resultados Obtenidos

Durante el desarrollo del trabajo práctico se realizaron múltiples pruebas para evaluar el rendimiento de dos algoritmos encargados de calcular la media aritmética de una lista de números enteros. La evaluación se centró principalmente en comparar el tiempo de ejecución entre una versión iterativa (media_iteracion) y otra basada en la función nativa sum() de Python (media_sum).

Uno de los principales desafíos fue lograr que los resultados de tiempo fueran medibles y comparables. Al trabajar con listas pequeñas (por ejemplo, de tamaño 10^1 a 10^4), el tiempo de

n media_iteracion (ms)

10 0.000000

100 0.000000

1000 2.488136

10000 5.996943

100000 28.293133

1000000 671.829700

10000000 2306.119442

ejecución era tan bajo en ambos algoritmos que en ocasiones media_sum arrojaba un tiempo de 0.000000 milisegundos. Esto ocurrió porque sum() está optimizada a nivel de lenguaje y ejecuta con una eficiencia que supera la capacidad de medición con time.time() para intervalos tan breves.

Para solucionar esto, se aumentó el tamaño de la lista a 10 **elementos** (un millón), lo que permitió observar diferencias significativas entre ambos algoritmos:

- **media_iteracion:** 188.89 milisegundos
- **media_sum:** 22.01 milisegundos

Esto demuestra que, aunque ambos algoritmos tienen la misma complejidad teórica (O(n)), la implementación de sum() es mucho más eficiente en la práctica, debido a que está escrita en C y forma parte de la biblioteca estándar de Python.

Validación

El programa fue validado con distintos tamaños de entrada, y el comportamiento esperado se cumplió en todos los casos. También se corrigió un error en versiones anteriores donde no se ejecutaba correctamente la segunda parte de las pruebas por un error de indentación dentro del main. Además, se resolvió una confusión inicial sobre la inclusión de print() dentro de los bucles, lo que ralentizaba el programa innecesariamente.

Errores detectados y solucionados

Durante el desarrollo se presentaron varios inconvenientes técnicos que fueron corregidos:

- Se detectó que una versión previa del algoritmo no sumaba correctamente los valores porque dentro del bucle for se usaba total = num en lugar de total += num. Este error fue corregido para que la acumulación fuera funcional.
- En la primera versión del programa, no se estaba generando correctamente una nueva lista de números para el segundo conjunto de pruebas (el de media_sum()), por lo que los resultados no aparecían o eran incorrectos. Este problema fue solucionado reorganizando el bloque principal y asegurando que la generación de datos se realice antes de cada medición.
- También se corrigió un error de indentación en la función main() que provocaba que algunos bloques no se ejecutaran correctamente.

- Se verificó que la función print() para los resultados estuviera dentro del bucle de prueba, ya que de lo contrario no se mostraban todas las mediciones esperadas.

6. Conclusiones

El trabajo permitió aplicar de manera concreta conceptos clave como estructuras repetitivas, funciones y análisis de eficiencia algorítmica. Se comprendió cómo medir el rendimiento de distintos enfoques para resolver un mismo problema y se reforzó el valor práctico de herramientas como la notación Big O.

El uso de funciones optimizadas, como sum(), demostró ser más eficiente que soluciones personalizadas, a pesar de compartir la misma complejidad teórica. Esta experiencia puede trasladarse a futuros proyectos donde la optimización sea necesaria.

Entre las dificultades superadas estuvieron errores de indentación, tiempos de ejecución difíciles de medir, y la correcta organización del código, todas abordadas mediante pruebas, ajustes y revisión colaborativa.

---

7. Bibliografía

- Downey, A. (2016). *Pensar en Python: Cómo pensar como un informático* (2ª ed.). O'Reilly Media. https://greenteapress.com/wp/think-python-2e/

- Universidad Tecnológica Nacional. (s.f.). *Clases de Programación I – Tecnicatura Universitaria en Programación*. YouTube. https://www.youtube.com/@TecnicaturaTUP

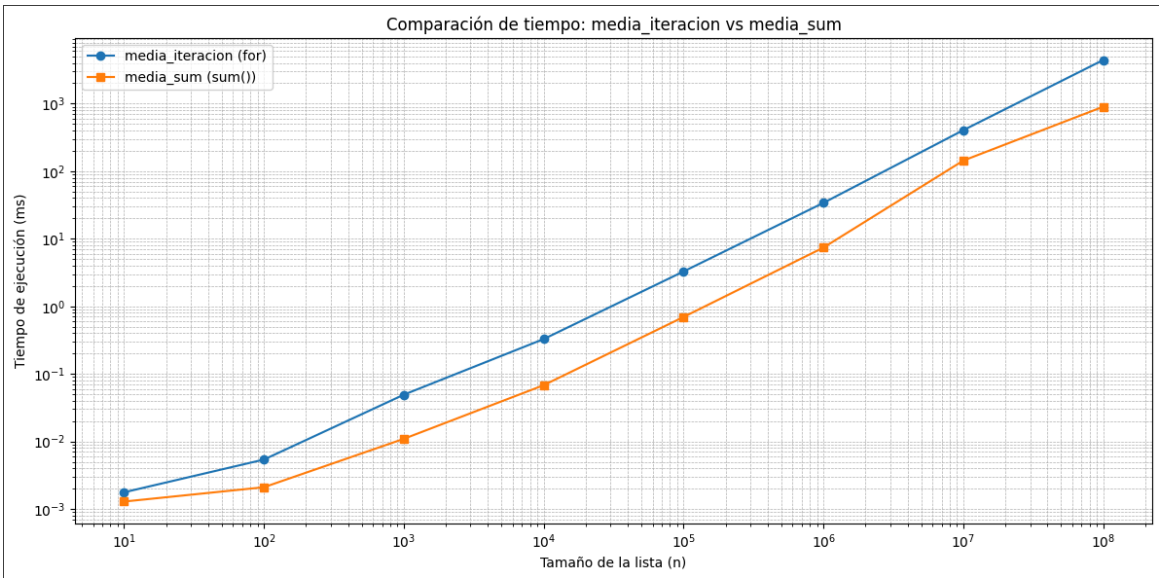
- Python Software Foundation. (s.f.). *The Python Language Reference*. https://docs.python.org/3/

- **Anexos**
  A continuación se anexan capturas del programa funcionando.
  - Enlace al video explicativo.
  - Código completo
  - Enlace al repositorio en GitHub
  - Cuadro comparativo.

C: > Users > USUARIO > Desktop > Programacion1_Practica > Tp_Integrador.py > tiempo

```python
import time
import random

#Algoritmo 1, suma los números 1 por 1, de manera iterativa. Realiza la sumatoria y el cálculo de la media

def media_iteracion(numeros):
    total=0
    for num in numeros:
        total+=num                  #Suma acumulativa
    return total/len(numeros)       # Se divide por la cantidad de elementos

#Algoritmo 2, utiliza la función sum para realizar la sumatoria y luego calcula la media.

def media_sum(numeros):             # sum() recorre la lista internamente
    return sum(numeros) / len(numeros)


#Función para medir el tiempo: mide el tiempo en milisegundos

def tiempo(funcion,numeros):      #Tiempo inicial
    start=time.time()
    resultado=funcion(numeros)    #Se ejecuta la función pasada como argumento
    end=time.time()
    return resultado, (end-start) * 1000 # Se devuelve el resultado y el tiempo en ms

#Programa Principal

def main():

#Pruebas

    print("n\t media_iteracion (ms)")
    for i in range(1, 9):            # Se prueban listas de tamaño 10^1 hasta 10^8
        cantidad = 10**i
        numeros = [random.randint(1, 100) for _ in range(cantidad)]     #Se genera una lista aleatoria
        _, duracion = tiempo(media_iteracion, numeros)
        print(f"{cantidad}\t{duracion:.6f}")


    print("\nn\t media_sum (ms)")
    for i in range(1, 9):            #Se respetan los mismos tamaños para el segundo algoritmo
        cantidad = 10**i
        numeros = [random.randint(1, 100) for _ in range(cantidad)] #Se genera una lista aleatoria
        _, duracion = tiempo(media_sum, numeros)
        print(f"{cantidad}\t{duracion:.6f}")

if __name__ == "__main__":
    main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
n        media_iteracion (ms)
10      0.000000
100     0.000000
100     0.000000
1000    0.999928
10000   5.996227
100000  23.988962
1000000 188.891888
100000  23.988962
1000000 188.891888
10000000        2231.720448
100000000       23095.793247
```

```
n        media_sum (ms)
10      0.000000
100     0.000000
1000    0.000000
10000   1.000881
100000  2.000093
1000000 22.017479
10000000        225.871086
```