ANALISIS DE ALGORITMOS

ALUMNOS:

- Elizondo, Iñaki - *elizondo.inaki25@gmail.com*
- Ducoli, Maximiliano - *maxiducoli@gmail.com*

MATERIA:

- **Programación I**

PROFESOR:

- **Ariel Enferrel**

TUTOR:

- **Franco González**

FECHA DE ENTREGA:

- **09/06/2025**

INTRODUCCIÓN:

Este tema se eligió debido a que se considera que es un tema absolutamente importante dentro del mundo de la programación, es vital el hecho de que el programa no solo resuelva las operaciones correctamente, sino que también las haga de una forma optimizada, es decir siendo rápido, conciso y con el menor costo de recursos posible.

Se pretende poder comprender que cantidad de recursos consumen distintas metodologías de trabajo, y poder dominar así el arte de la correcta optimización de las distintas estructuras de trabajo.

MARCO TEÓRICO:

¿Qué es el Análisis de Algoritmos?

El análisis de algoritmos estudia cómo de rápido o lento funciona un programa y cuánta memoria usa. El objetivo es mejorar:

- **Tiempo de ejecución (eficiencia temporal)**
- **Uso de memoria (eficiencia espacial)**

Conceptos Clave

1. **Notación Big-O :**
   Muestra cómo crece el tiempo de ejecución de un algoritmo en función del tamaño de los datos de entrada, en el peor caso.
   Ejemplos:
   - **O(1): Constante**
   - **O(log n): Logarítmico**
   - **O(n): Lineal**
   - **O(n^2): Cuadrático**
2. **Tiempo de Ejecución Real :**
   Se puede medir con herramientas como el módulo time en Python.
3. **Complejidad Espacial :**
   Indica cuánta memoria adicional usa un algoritmo durante su ejecución.

En este marco teórico se presenta el fundamento conceptual detrás de dos algoritmos utilizados para determinar si un número es primo. Un número primo es aquel que solo tiene dos divisores: 1 y él mismo. Este concepto, aunque sencillo desde el punto de vista matemático, requiere de una implementación eficiente en programación para evitar consumir recursos innecesarios, especialmente cuando se trabaja con números grandes.

Se analizaron dos funciones distintas:

- **es_primo(x) : Implementa una solución eficiente basada en la propiedad matemática que indica que basta con probar divisores hasta la raíz cuadrada del número. Esto reduce significativamente la cantidad de operaciones necesarias.**
- **Solo revisa divisores hasta la raíz cuadrada de x.**
- **Complejidad Temporal: O(√n)**
- **Complejidad Espacial: O(1)**

- **es_primo_malo(x) : Es una versión básica que prueba todos los divisores desde 2 hasta x - 1, lo cual resulta ineficiente a medida que crece el valor de entrada.**
- **Revisa todos los números desde 2 hasta x - 1.**
- **Complejidad Temporal: O(n)**
- **Complejidad Espacial: O(1)**

Para comparar ambas funciones, se utilizó una función adicional llamada medir_tiempo(), que ejecuta cada algoritmo múltiples veces y devuelve el tiempo promedio de ejecución. Esta medición permitió evaluar empíricamente la diferencia de rendimiento entre ambos métodos.

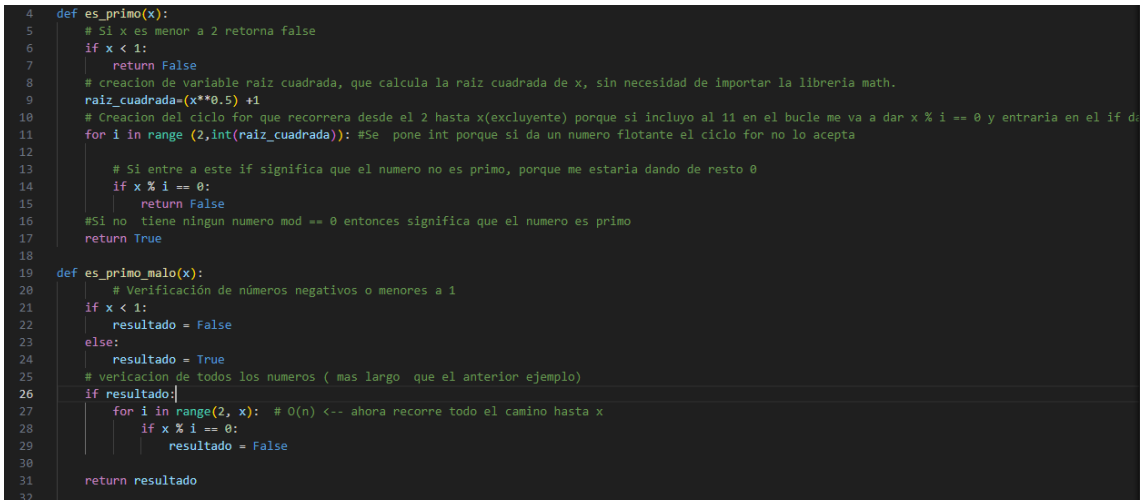
El objetivo del análisis fue mostrar cómo el diseño del algoritmo afecta el desempeño del programa, destacando la importancia de escribir código eficiente desde las primeras etapas del aprendizaje en programación. Este caso práctico representa una introducción al análisis de complejidad computacional y a la optimización de algoritmos desde una perspectiva práctica y accesible para estudiantes principiantes.

CASO PRÁCTICO:

Descripción del Problema

Se requiere implementar y comparar dos algoritmos para determinar si un número es primo, evaluando su eficiencia en términos de tiempo de ejecución.

Código Fuente Comentado



```python
def es_primo(x):
    # Si x es menor a 2 retorna false
    if x < 1:
        return False
    # creacion de variable raiz cuadrada, que calcula la raiz cuadrada de x, sin necesidad de importar la libreria math.
    raiz_cuadrada=(x**0.5) +1
    # Creacion del ciclo for que recorrera desde el 2 hasta x(excluyente) porque si incluyo al 11 en el bucle me va a dar x % i == 0 y entraria en el if d
    for i in range (2,int(raiz_cuadrada)): #Se  pone int porque si da un numero flotante el ciclo for no lo acepta

        # Si entre a este if significa que el numero no es primo, porque me estaria dando de resto 0
        if x % i == 0:
            return False
    #Si no  tiene ningun numero mod == 0 entonces significa que el numero es primo
    return True

def es_primo_malo(x):
        # Verificación de números negativos o menores a 1
    if x < 1:
        resultado = False
    else:
        resultado = True
    # vericacion de todos los numeros ( mas largo  que el anterior ejemplo)
    if resultado:
        for i in range(2, x):  # O(n) <-- ahora recorre todo el camino hasta x
            if x % i == 0:
                resultado = False

    return resultado
```

Explicación de decisiones de diseño:

- Se usó **time** para medir tiempos reales de ejecución.
- Se repitió cada prueba 100 veces para obtener un promedio confiable.
- Se imprimieron resultados formateados para facilitar la lectura y comparación.

Validación del funcionamiento

Se realizaron pruebas con números primos y no primos, verificando manualmente los resultados. En todas las ejecuciones, ambas funciones devolvieron el resultado esperado.

METODOLOGÍA UTILIZADA:

Para desarrollar este trabajo se siguieron los siguientes pasos:

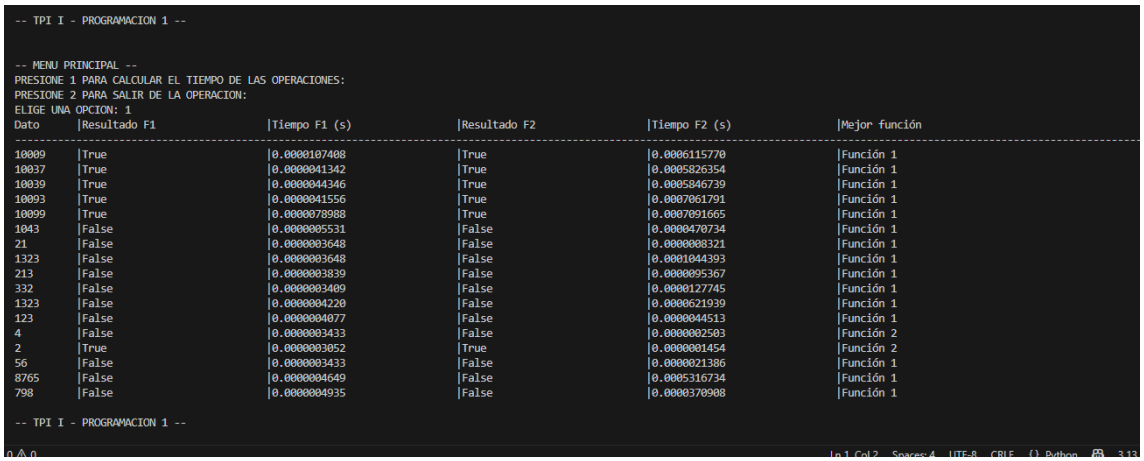
1. Investigación previa: Consulta de conceptos sobre números primos y algoritmos relacionados en documentación oficial de Python, páginas web y apuntes de clase.
2. Diseño del código: Implementación de las funciones con base en conocimientos adquiridos durante el primer cuatrimestre.

3. Pruebas y validación: Ejecución de múltiples casos de prueba para asegurar el correcto funcionamiento de las funciones.
4. Medición de tiempos: Uso de la librería **time** para evaluar el rendimiento de cada algoritmo.
5. Redacción del informe: Documentación completa del trabajo según la plantilla establecida.

RESULTADOS OBTENIDOS:

Se logró implementar correctamente ambas funciones, validando su funcionamiento con diversos ejemplos. La medición de tiempos mostró claramente que la función **es_primo**, que utiliza la raíz cuadrada, es mucho más eficiente que **es_primo_malo**.

Ejemplo de resultados obtenidos



```
-- TPI I - PROGRAMACION 1 --


-- MENU PRINCIPAL --
PRESIONE 1 PARA CALCULAR EL TIEMPO DE LAS OPERACIONES:
PRESIONE 2 PARA SALIR DE LA OPERACION:
ELIGE UNA OPCION: 1
Dato     |Resultado F1           |Tiempo F1 (s)          |Resultado F2           |Tiempo F2 (s)          |Mejor función
-------------------------------------------------------------------------------------------------------------------------
10009    |True                   |0.0000107408           |True                   |0.0006115770           |Función 1
10037    |True                   |0.0000041342           |True                   |0.0005826354           |Función 1
10039    |True                   |0.0000044346           |True                   |0.0005846739           |Función 1
10093    |True                   |0.0000041556           |True                   |0.0007061791           |Función 1
10099    |True                   |0.0000078988           |True                   |0.0007091665           |Función 1
1043     |False                  |0.0000005531           |False                  |0.0000470734           |Función 1
21       |False                  |0.0000003648           |False                  |0.0000008321           |Función 1
1323     |False                  |0.0000003648           |False                  |0.0001044393           |Función 1
213      |False                  |0.0000003839           |False                  |0.0000095367           |Función 1
332      |False                  |0.0000003409           |False                  |0.0000127745           |Función 1
1323     |False                  |0.0000004220           |False                  |0.0000621939           |Función 1
123      |False                  |0.0000004077           |False                  |0.0000044513           |Función 1
4        |False                  |0.0000003433           |False                  |0.0000002503           |Función 2
2        |True                   |0.0000003052           |True                   |0.0000001454           |Función 2
56       |False                  |0.0000003433           |False                  |0.0000021386           |Función 1
8765     |False                  |0.0000004649           |False                  |0.0005316734           |Función 1
798      |False                  |0.0000004935           |False                  |0.0000370908           |Función 1

-- TPI I - PROGRAMACION 1 --
```

Estos resultados reflejan que a medida que aumenta el tamaño del número, la diferencia de rendimiento entre ambos algoritmos se vuelve más notable.

CONCLUSIONES:

Este trabajo nos permitió aprender varias cosas importantes:

- La importancia de elegir buenos algoritmos: Pequeños cambios en la lógica pueden tener un gran impacto en el desempeño.
- Cómo medir el tiempo de ejecución de funciones: Es clave para entender qué partes del código consumen más recursos.
- Validación y prueba de código: Nos ayudó a asegurarnos de que nuestra solución funciona correctamente.

Aunque fue un desafío, pudimos superar las dificultades mediante investigación, ensayo y error, y consultas al profesor. Creemos que este tipo de trabajos fortalece nuestras bases como futuros programadores.

Posibles mejoras futuras:

- Realizar gráficos comparativos de rendimiento.
- Probar con números más grandes para ver el comportamiento real.

BIBLIOGRAFÍA:

- **Python Software Foundation. (2024). Python 3 Documentation .** https://docs.python.org/3/
- **© 2025 Smartick.** https://www.smartick.es/blog/matematicas/numeros-enteros/numeros-primos-y-numeros-compuestos/
- **Apuntes de la plataforma de Programación I**
- **© 2025 El Libro De Python.** https://ellibrodepython.com/
- **© 2025 El Libro De Python.** https://ellibrodepython.com/numeros-primos-python

ANEXOS:

- **Anexo A:** Captura de pantalla del código fuente y del resultado del programa (Ver en páginas superiores).
- Anexo B: Archivo fuente completo (.py) en :

  Github @maxiducoli: https://github.com/maxiducoli/TPI-II---Programaci-n-I

  Github @Picaporte25: **https://github.com/Picaporte25/TP_INTEGRADOR_PROGRAMACION_I**

- **Anexo C: Video explicativo:** https://drive.google.com/drive/folders/14IPDDnO4JJSBR5UqQ9UBdwASYjY6nC6g