



INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria de Ingeniería Campus  
Zacatecas



# Análisis y diseño de algoritmos

Grupo: 3CM1

Docente:

M. en C. Erika Sánchez-Femat

Alumna:

Melanie Aileen Roman Espitia

---

## Práctica 01: Análisis de casos

Fecha de Entrega: viernes 15 de septiembre de 2023 Zacatecas, México

## Contenido

INTRODUCCIÓN .....	3
DESARROLLO .....	4
ORDENAMIENTO BURBUJA .....	4
ORDENAMIENTO BURBUJA MEJORADA .....	6
CONCLUSIONES .....	8
REFERENCIAS .....	9

## INTRODUCCIÓN

---

La presente práctica tiene como objetivo proporcionar una comprensión sólida y completa de los conceptos fundamentales relacionados con la complejidad computacional de los algoritmos. Para lograr esto, es esencial comprender tres conceptos clave: el mejor caso, el peor caso y el caso promedio. Estos conceptos permiten evaluar exhaustivamente el rendimiento de un algoritmo en diversas circunstancias.

1. **Mejor Caso:** El "mejor caso" se refiere a la situación en la que un algoritmo funciona de manera más eficiente. En otras palabras, es el escenario ideal en el que el algoritmo requiere la menor cantidad de recursos (como tiempo o espacio) para completar su tarea. En el contexto del análisis de algoritmos, el mejor caso a menudo representa una entrada que está perfectamente adaptada al algoritmo, lo que minimiza la cantidad de trabajo necesario.
2. **Peor Caso:** El "peor caso" es el escenario en el que un algoritmo funciona de la manera menos eficiente posible. Representa la situación en la que el algoritmo requiere la mayor cantidad de recursos para completar su tarea. El análisis del peor caso es fundamental para garantizar que un algoritmo no se vuelva inaceptablemente lento bajo ninguna circunstancia.
3. **Caso Promedio:** El "caso promedio" representa el rendimiento esperado de un algoritmo cuando se le proporciona una gama de entradas posibles, distribuidas de acuerdo con una distribución de probabilidad específica. El caso promedio es importante porque refleja cómo se comportará el algoritmo en situaciones del mundo real, donde las entradas pueden variar ampliamente en su naturaleza.

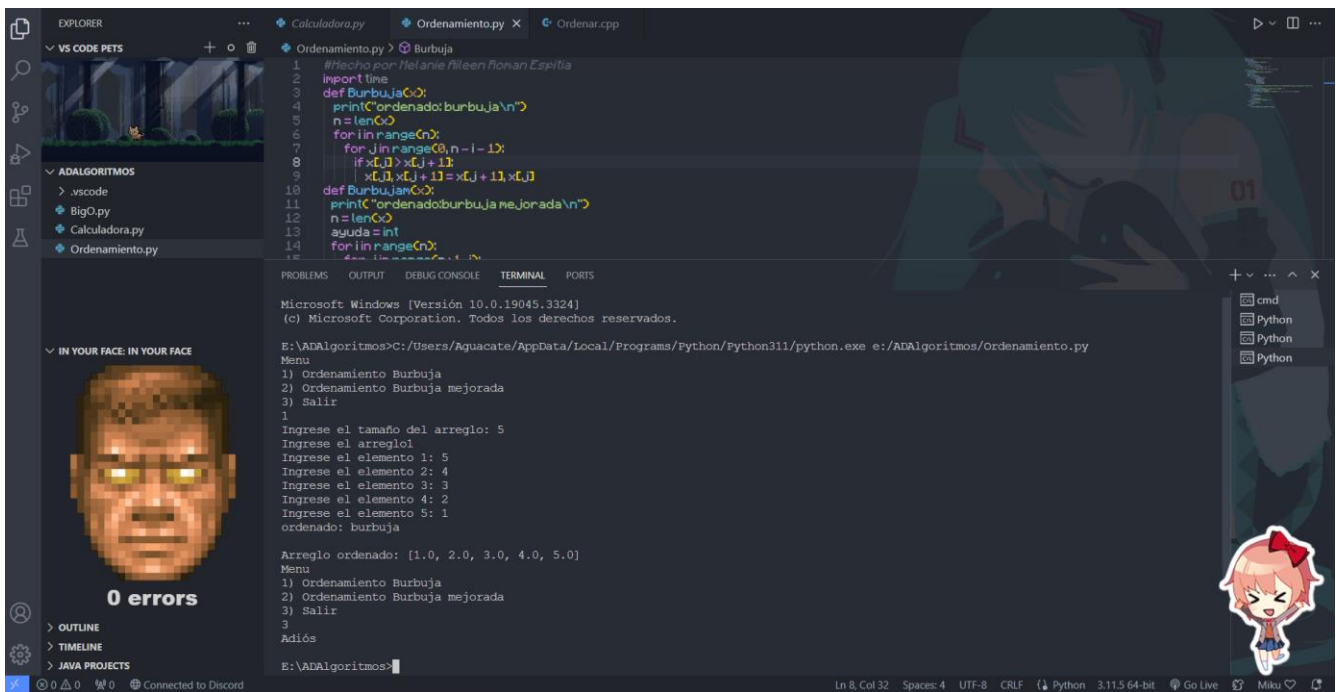
En este reporte de práctica, se analizarán dos métodos de ordenamiento: el ordenamiento burbuja y el ordenamiento burbuja mejorada. Estos algoritmos se implementarán en el lenguaje de programación Python, y se realizará un análisis exhaustivo de su rendimiento en los tres casos mencionados anteriormente. Además, se utilizará la notación BigO para describir la complejidad de cada algoritmo, lo que nos permitirá comprender cómo escalan en términos de tiempo de ejecución a medida que el tamaño de la entrada aumenta. Este análisis proporcionará una visión más profunda de la eficiencia y la idoneidad de estos algoritmos para diferentes situaciones y conjuntos de datos.

# DESARROLLO

## ORDENAMIENTO BURBUJA

### 1. Peor Caso (Worst Case):

- Notación Big O:  $O(n^2)$
- Explicación: El peor caso ocurre cuando el arreglo está en orden inverso, es decir, los elementos más grandes están al principio y los más pequeños al final. En esta situación, el algoritmo de burbuja debe realizar el máximo número de comparaciones y swaps posible. Cada elemento debe "burbujear" desde la posición inicial hasta su posición correcta en el extremo derecho del arreglo. Esto implica  $n - 1$  comparaciones y swaps para el primer elemento, luego  $n - 2$  para el segundo elemento, y así sucesivamente, hasta llegar a 1 comparación y swap para el último elemento. La suma de estos valores da como resultado una complejidad cuadrática  $O(n^2)$ .



The screenshot shows a VS Code editor with a Python file named 'Ordenamiento.py'. The code implements a bubble sort algorithm. The terminal output shows the program running and displaying the sorted array [1.0, 2.0, 3.0, 4.0, 5.0].

```
1 #Hecho por: Helaine Nileen Roman Espitia
2 import time
3 def burbuja(C):
4     print("Ordenado: burbuja\n")
5     n = len(C)
6     for i in range(n):
7         for j in range(n - i - 1):
8             if C[j] > C[j + 1]:
9                 C[j], C[j + 1] = C[j + 1], C[j]
10    def burbuja_m(C):
11        print("Ordenado: burbuja mejorada\n")
12        n = len(C)
13        ayuda = int
14        for i in range(n):
```

Terminal Output:

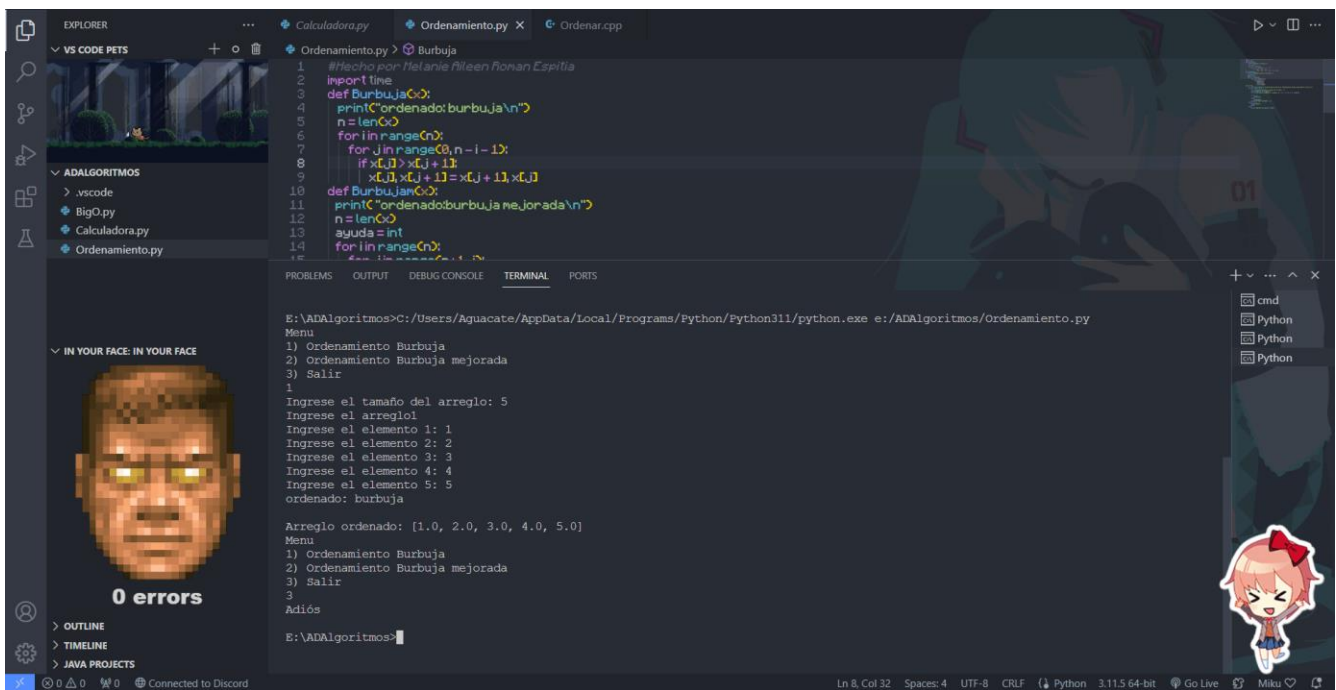
```
Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

E:\ADAlgoritmos>C:\Users\Aguacate\AppData\Local\Programs\Python\Python311\python.exe e:\ADAlgoritmos\Ordenamiento.py
Menu
1) Ordenamiento Burbuja
2) Ordenamiento Burbuja mejorada
3) Salir
Ingrese el tamaño del arreglo: 5
Ingrese el arreglo:
Ingrese el elemento 1: 5
Ingrese el elemento 2: 4
Ingrese el elemento 3: 3
Ingrese el elemento 4: 2
Ingrese el elemento 5: 1
ordenado: burbuja

Arreglo ordenado: [1.0, 2.0, 3.0, 4.0, 5.0]
Menu
1) Ordenamiento Burbuja
2) Ordenamiento Burbuja mejorada
3) Salir
Adiós
E:\ADAlgoritmos>
```

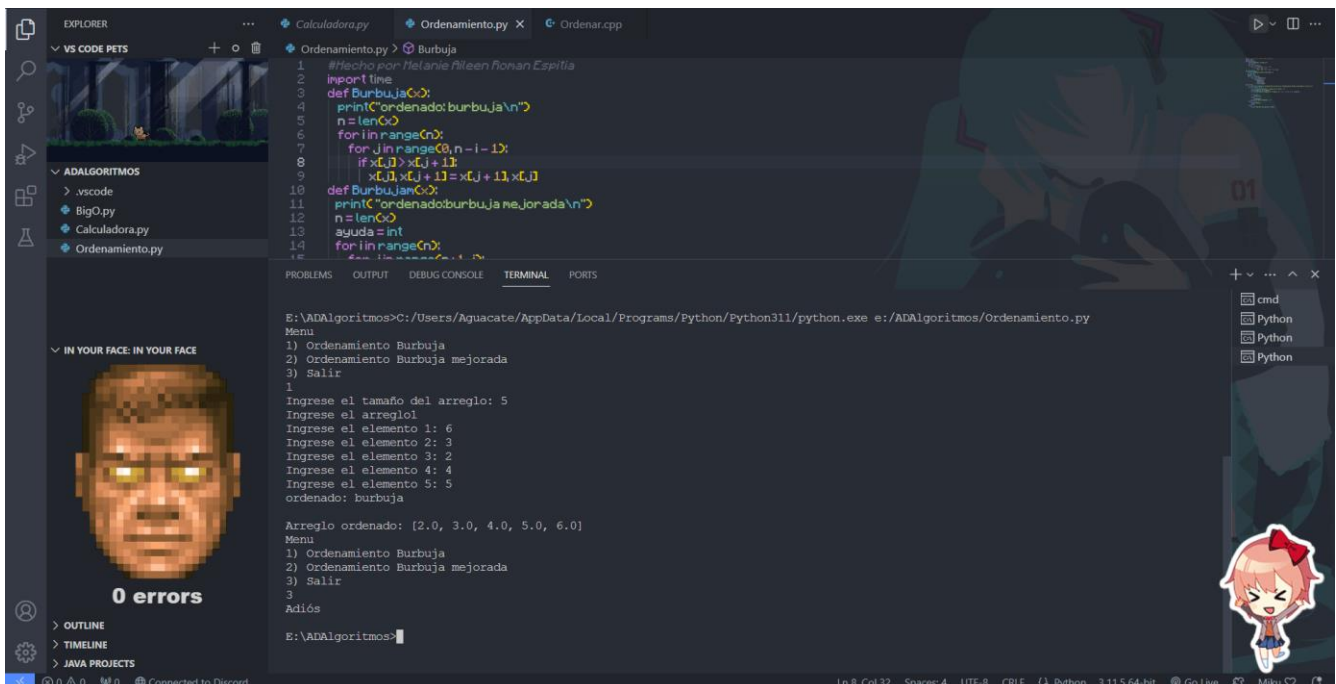
### 2. Mejor Caso (Best Case):

- Notación Big O:  $O(n)$
- Explicación: El mejor caso ocurre cuando el arreglo ya está ordenado en orden ascendente. En este caso, el algoritmo de burbuja solo necesita realizar una pasada por el arreglo para darse cuenta de que no se necesitan intercambios, ya que los elementos ya están en orden. Esto significa que el número de comparaciones y swaps es proporcional al tamaño del arreglo, lo que resulta en una complejidad lineal  $O(n)$ .



### 3. Caso Promedio (Average Case):

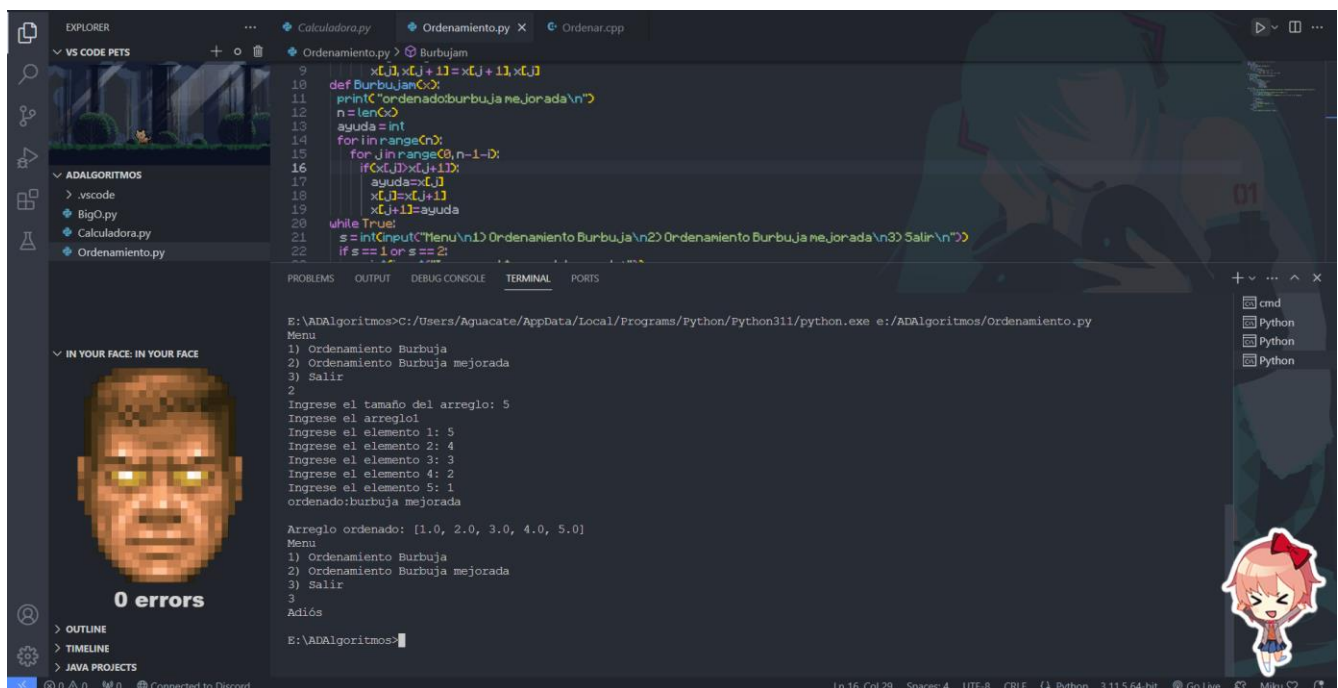
- Notación Big O:  $O(n^2)$
- Explicación: El caso promedio es más cercano al peor caso que al mejor caso. Esto se debe a que, en la práctica, es poco probable que los arreglos estén perfectamente ordenados o en orden inverso. En la mayoría de los casos, el algoritmo de burbuja realizará un número significativo de comparaciones y swaps adicionales en comparación con el mejor caso. Por lo tanto, se considera que el caso promedio tiene una complejidad cuadrática  $O(n^2)$ .



## ORDENAMIENTO BURBUJA MEJORADA

### 1. Peor Caso (Worst Case):

- Notación Big O:  $O(n^2)$
- Explicación: El peor caso ocurre cuando el arreglo está en orden inverso, similar al peor caso de la burbuja estándar. En esta situación, el algoritmo de burbuja mejorada debe realizar el máximo número de comparaciones y swaps posible. Cada elemento debe "burbujear" desde la posición inicial hasta su posición correcta en el extremo derecho del arreglo. Esto implica  $n - 1$  comparaciones y swaps para el primer elemento, luego  $n - 2$  para el segundo elemento, y así sucesivamente, hasta llegar a 1 comparación y swap para el último elemento. La suma de estos valores da como resultado una complejidad cuadrática  $O(n^2)$ , igual que en el caso de la burbuja estándar.



The screenshot shows a VS Code editor with a Python file named `Ordenamiento.py`. The code implements the Improved Bubble Sort algorithm. It includes a `def BurbujaMejorada()` function that takes an array and sorts it in ascending order. The function uses a `while` loop to repeatedly traverse the array, comparing adjacent elements and swapping them if they are in the wrong order. The `while` loop continues until no swaps are made in a full pass. The main part of the script prompts the user to enter the size of the array, the elements, and then displays the sorted array.

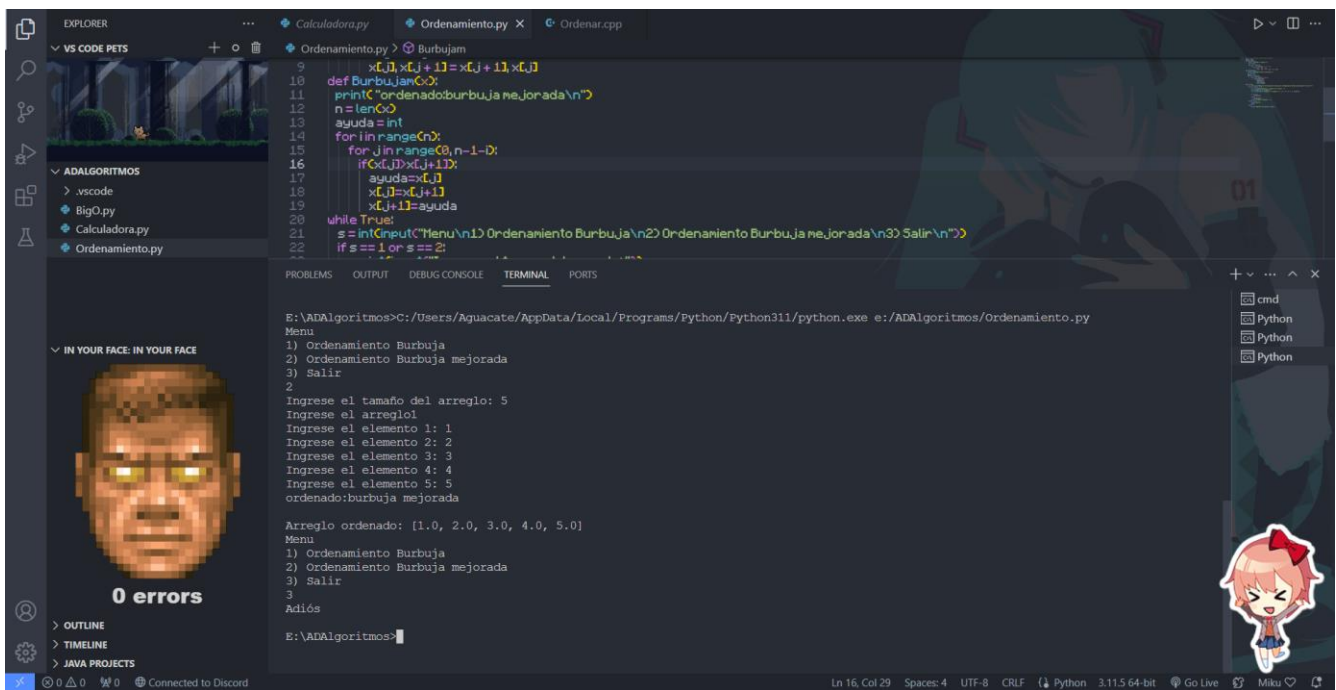
```
9 def BurbujaMejorada():
10     x = [0] * 10
11     print("Ordenado: burbuja mejorada\n")
12     n = len(x)
13     ayuda = int(input("Menu\n1) Ordenamiento Burbuja\n2) Ordenamiento Burbuja mejorada\n3) Salir\n"))
14     for i in range(n):
15         for j in range(0, n-1-i):
16             if x[j] > x[j+1]:
17                 ayuda = x[j]
18                 x[j] = x[j+1]
19                 x[j+1] = ayuda
20     while True:
21         s = int(input("Menu\n1) Ordenamiento Burbuja\n2) Ordenamiento Burbuja mejorada\n3) Salir\n"))
22         if s == 1 or s == 2:
```

The terminal output shows the execution of the program. It prompts the user to enter the size of the array (5), the elements (5, 4, 3, 2, 1), and then displays the sorted array: `Arreglo ordenado: [1.0, 2.0, 3.0, 4.0, 5.0]`. The terminal also shows the menu options: `Menu\n1) Ordenamiento Burbuja\n2) Ordenamiento Burbuja mejorada\n3) Salir\n`.

### 2. Mejor Caso (Best Case):

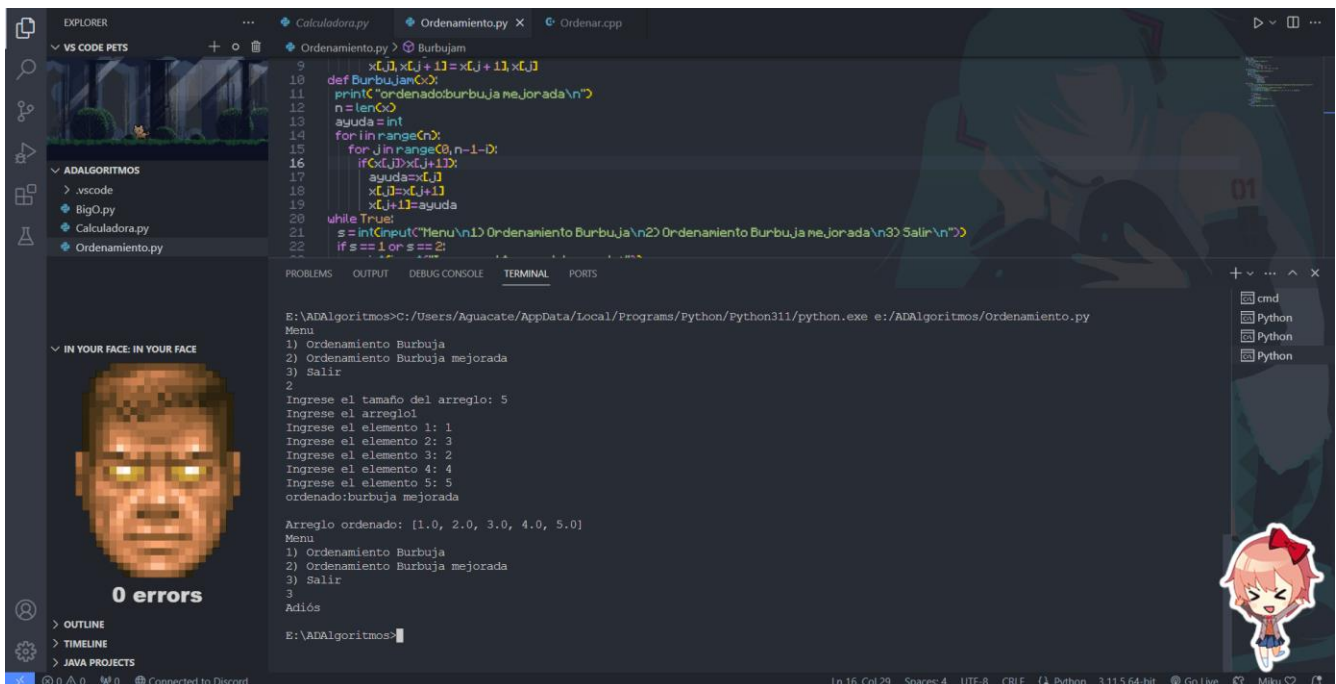
- Notación Big O:  $O(n)$
- Explicación: El mejor caso ocurre cuando el arreglo ya está ordenado en orden ascendente, similar al mejor caso de la burbuja estándar. En este caso, el algoritmo de burbuja mejorada solo necesita realizar una pasada por el arreglo para darse cuenta de que no se necesitan intercambios, ya que los elementos ya están en orden. Esto significa que el número de comparaciones y swaps es proporcional al tamaño del arreglo, lo que resulta en una complejidad lineal  $O(n)$ .





### 3. Caso Promedio (Average Case):

- Notación Big O:  $O(n^2)$
- Explicación: Al igual que en el caso de la burbuja estándar, el caso promedio de la burbuja mejorada se asemeja más al peor caso que al mejor caso. En la mayoría de los casos, el algoritmo de burbuja mejorada realizará un número significativo de comparaciones y swaps adicionales en comparación con el mejor caso. Por lo tanto, se considera que el caso promedio tiene una complejidad cuadrática  $O(n^2)$ .



## CONCLUSIONES

---

Concluyendo esta práctica de análisis de algoritmos de ordenamiento, se han alcanzado varios objetivos clave que contribuyen significativamente a nuestra comprensión de la complejidad computacional y la evaluación de algoritmos. Aquí están algunas conclusiones destacadas:

1. **Comprendiendo los Casos Principales:** A lo largo de esta práctica, hemos explorado y comprendido en profundidad los tres casos principales para analizar algoritmos: el mejor caso, el peor caso y el caso promedio. Hemos aprendido a identificar y evaluar el rendimiento de los algoritmos en estas diferentes situaciones, lo que nos permite anticipar cómo se comportarán en una variedad de escenarios del mundo real.
2. **Notación BigO:** Hemos adquirido una comprensión sólida de la notación BigO y su importancia en la descripción de la complejidad de los algoritmos. La notación BigO nos permite expresar de manera concisa cómo crece el tiempo de ejecución de un algoritmo en relación con el tamaño de la entrada. Esto es esencial para comparar y seleccionar algoritmos adecuados para tareas específicas.
3. **Ordenamiento Burbuja y Burbuja Mejorada:** Mediante la implementación y el análisis de los algoritmos de ordenamiento burbuja y burbuja mejorada, hemos visto en la práctica cómo estos conceptos se aplican en el mundo real. Hemos observado que el rendimiento de estos algoritmos varía significativamente según el tipo de entrada y hemos aprendido que, aunque el ordenamiento burbuja mejorada puede ser más eficiente en algunos casos, ambos algoritmos tienen una complejidad cuadrática en el peor caso.
4. **Importancia de la Eficiencia:** Esta práctica también destaca la importancia de la eficiencia de los algoritmos en el diseño de software. Entender cómo evaluar y seleccionar algoritmos basados en su complejidad y su rendimiento en diferentes situaciones es esencial para crear aplicaciones eficientes y que respondan de manera rápida a las necesidades del usuario.

En resumen, esta práctica nos ha proporcionado una base sólida en el análisis de algoritmos y en el uso de la notación BigO. Hemos ganado experiencia práctica al implementar y analizar algoritmos de ordenamiento y hemos aprendido a considerar diferentes escenarios de rendimiento al evaluar algoritmos. Estos conocimientos son esenciales en el mundo de la informática y nos permitirán tomar decisiones informadas en futuros proyectos de desarrollo de software.



## REFERENCIAS

---

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms" (3rd ed.). MIT Press.
2. Sedgewick, R., & Wayne, K. (2011). "Algorithms" (4th ed.). Addison-Wesley Professional.
3. Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2008). "Algorithms". McGraw-Hill Science/Engineering/Math.
4. Kleinberg, J., & Tardos, É. (2005). "Algorithm Design". Pearson/Addison-Wesley.
5. GeeksforGeeks (<https://www.geeksforgeeks.org/>)