**Algoritmos y Representación de Datos**

**Escuela Colombiana de Ingeniería Julio Garavito**

**Análisis de los Algoritmos de Ordenamiento**

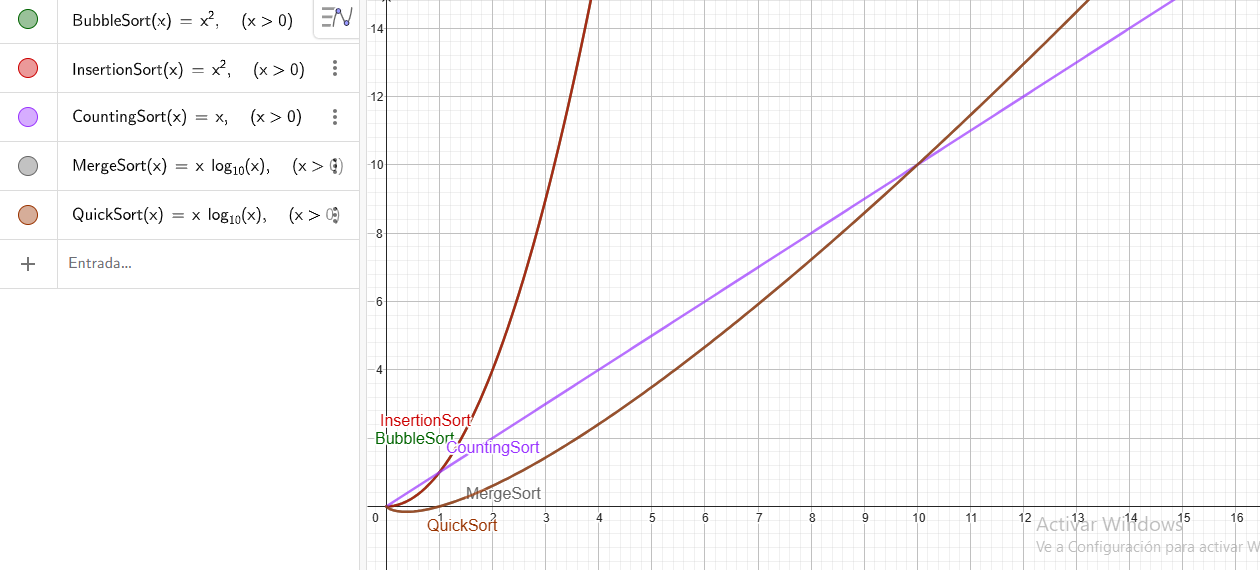
**Sebastian Cardona Parra**

**Docente: Rafael Alberto Niquefa Velasquez**

**2025-1**

# Introducción

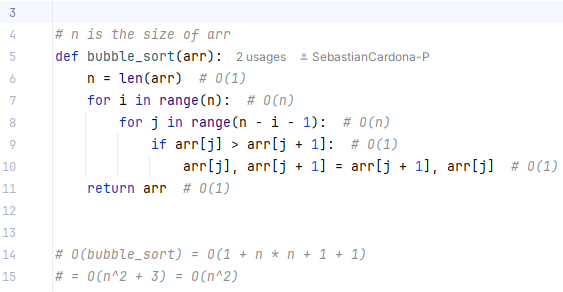
El ordenamiento de datos es una tarea fundamental en informática y se han desarrollado múltiples algoritmos para optimizar este proceso. En este informe, se analizó cinco algoritmos de ordenamiento: Bubble Sort, Insertion Sort, Merge Sort, Quick Sort y Counting Sort, comparando su eficiencia en términos de tiempo de ejecución y complejidad computacional.



# Algoritmos

## Bubble Sort

Bubble sort es un algoritmo de ordenamiento simple que funciona comparando elementos adyacentes de una lista y los intercambia en el orden correcto. Este proceso se repite hasta que toda la lista esté ordenada



### Complejidad Algorítmica

Peor caso (lista en orden inverso) tiene una complejidad de O(n2)

Mejor caso (lista ya ordenada) tiene una complejidad de O(n)

Caso promedio tiene una complejidad de O(n2)

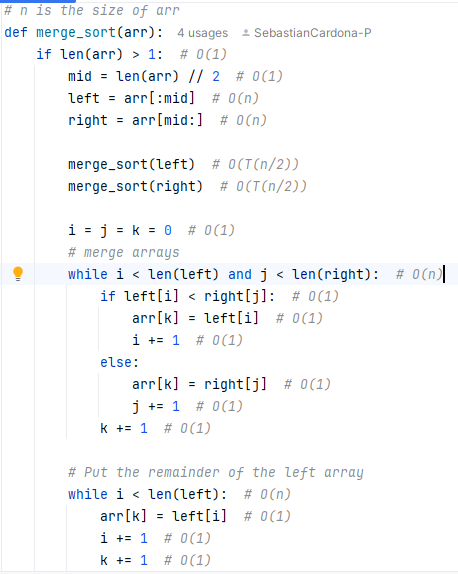
### Características

Este es un algoritmo que es fácil de entender e implementar, pero es ineficiente cuando se implementan en listas grandes

## Merge Sort

Merge Sort es un algoritmo recursivo de ordenamiento más eficiente siguiendo el paradigma de “divide y vencerás”, pues divide la lista en dos partes, se ordenan de manera independiente y luego se fusionan en orden

Texto

El contenido generado por IA puede ser incorrecto.

### Complejidad Algorítmica

Al ser un algoritmo recursivo con el paradigma de dividir y vencer, se usó el teorema maestro para el cálculo de su complejidad

Peor caso tiene una complejidad de O(n log n)

Mejor caso tiene una complejidad de O(n log n)

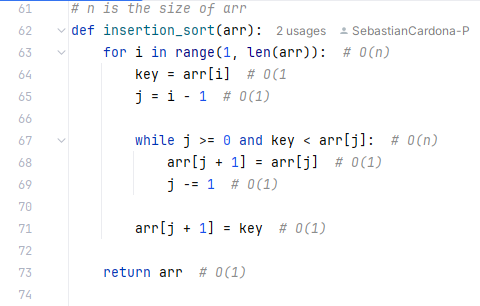
Caso promedio tiene una complejidad de O(n log n)

### Características

Es un algoritmo más eficiente que el anterior para grandes conjuntos de datos que funciona bien con estructuras de datos enlazadas, pero que requieren memoria extra para las sub listas

## Insertion Sort

Es un algoritmo eficiente para listas pequeñas, en el cual se basa en la suposición de que el primer elemento ya está ordenado, toma un elemento de la parte desordenada de la lista y la inserta de forma adecuada en la parte ordenada, luego repite este proceso



### Complejidad Algorítmica

Peor caso (lista en orden inverso) tiene una complejidad de O(n2)

Mejor caso (lista ya ordenada) tiene una complejidad de O(n)

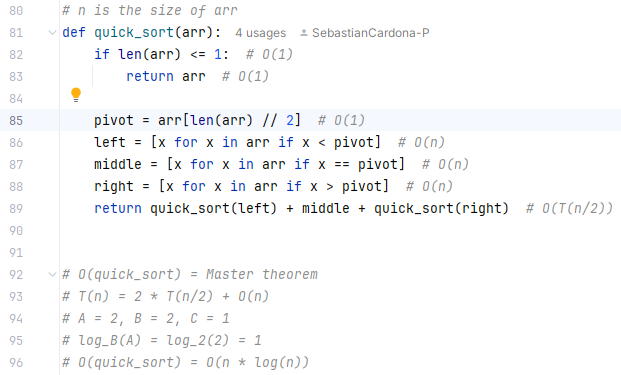
Caso promedio tiene una complejidad de O(n2)

### Características

Es un algoritmo sencillo y eficiente para listas pequeñas y casi ordenadas, pero es ineficiente para listas de mayor tamaño

## Quick Sort

Quick sort es uno de los algoritmos más eficientes de ordenamiento, al igual que merge sort sigue el paradigma de dividir y vencer, pero tiene una eficiencia en el uso de memoria. Básicamente se elije un pivote, los elementos menores se ponen atrás del pivote y los mayores adelante, luego recursivamente se aplica el quick sort atrás y adelante del pivote



### Complejidad Algorítmica

Al ser un algoritmo recursivo con el paradigma de dividir y vencer, se usó el teorema maestro para el cálculo de su complejidad

Peor caso (Siempre se escoge el peor pivote, mayor o menor elemento en la lista) tiene una complejidad de O(n2)

Mejor caso tiene una complejidad de O(n log n)

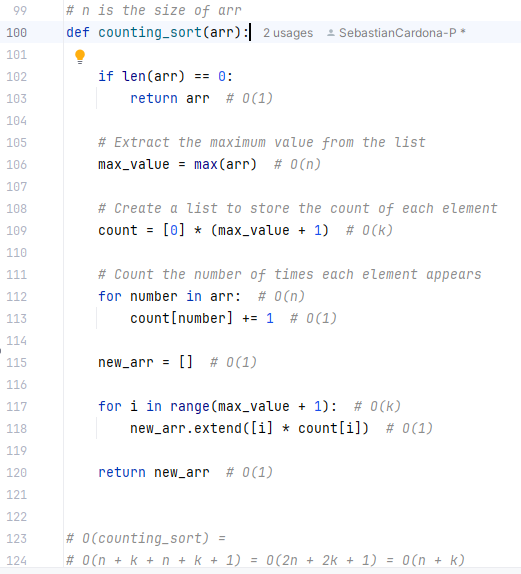
Caso promedio tiene una complejidad de O(n log n)

### Características

Es un algoritmo eficiente para grandes cantidades de datos que no necesita memoria extra significativa para funcionar, pero se debe tener cuidado al escoger el pivote para no demorar y aumentar más la complejidad

## Counting Sort

**Counting sort** es un algoritmo de ordenamiento eficiente cuando se trabaja con números dentro de un **rango conocido y no negativos**. Se basa en contar la frecuencia de cada elemento y usar esa información para reconstruir la lista ordenada. El problema de este algoritmo es que necesita crear una nueva lista con tamaño igual al máximo valor del arreglo dado, por lo que cuando el valor máximo es muy elevado, se usa demasiada memoria y se vuelve ineficiente.



### Complejidad Algorítmica

N es el tamaño del array y k es el valor del número máximo del arreglo

Peor caso tiene una complejidad de O(n + k)

Mejor caso tiene una complejidad de O(n + k)

Caso promedio tiene una complejidad de O(n + k)

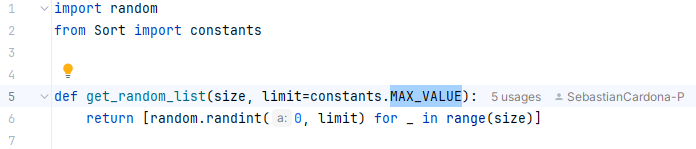
### Características

Más eficiente que quick sort si el rango de valores k es pequeño con comparación de n, útil para ordenar grandes volúmenes de datos en un rango k limitado, pero ineficiente si k tiene un rango muy alto

# Análisis y Gráficas

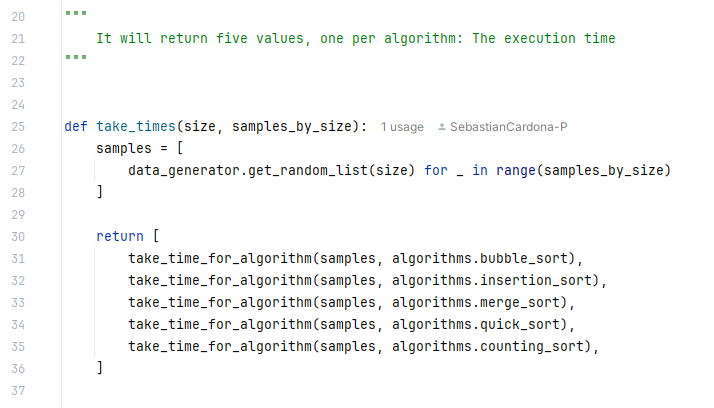
## Metodología del Análisis

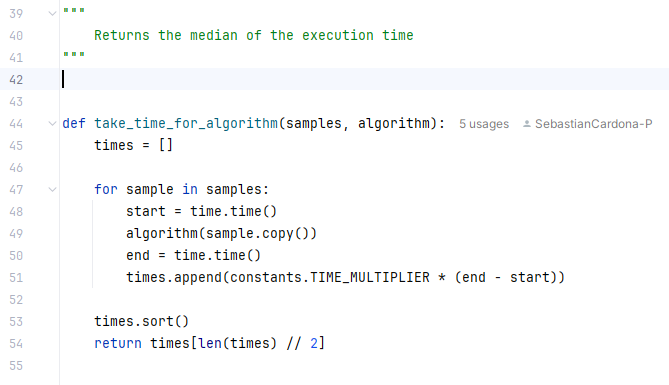
Se implementó una serie de módulos en Python para realizar las pruebas. Primeramente, se creo un modulo de generación de datos, el cual dado un tamaño y un límite de valor máximo, se crea una lista con valores aleatorios con el tamaño dado y con números de 0 al valor máximo que llamaremos MAX\_VALUE



También hay un modulo que toma el tiempo de ejecución de cada uno de los algoritmos para una serie datos aleatorios generados de diferentes tamaños

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto. 

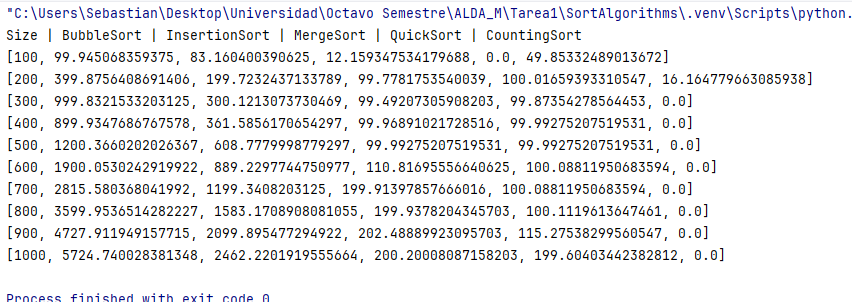


Para por último desde la función principal “app.py” parametrizar todo y tomas los datos

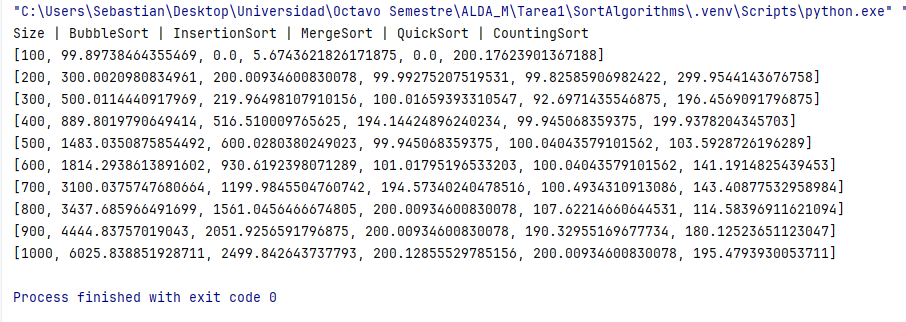


## Toma de datos:

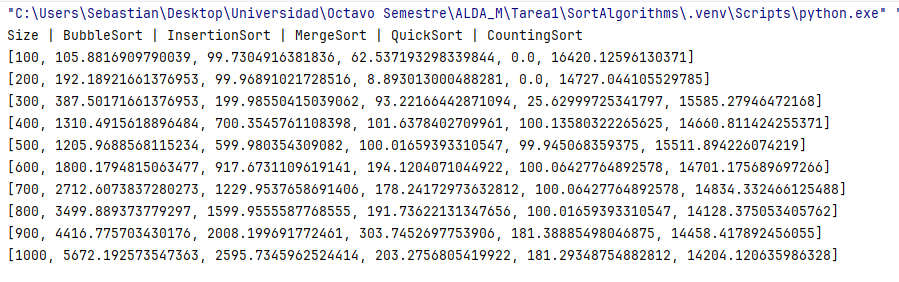
1. Para listas de tamaño mínimo 100 y tamaño máximo 1000, haciendo 10 ejemplos por tamaño y con MAX\_VALUE = 1000

****

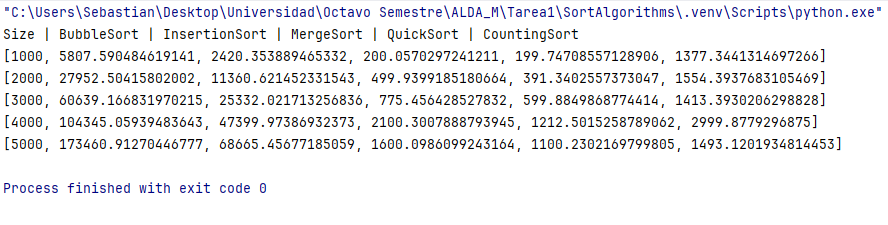
1. Para listas de tamaño mínimo 100 y tamaño máximo 1000, haciendo 10 ejemplos por tamaño y con MAX\_VALUE = 10000

****

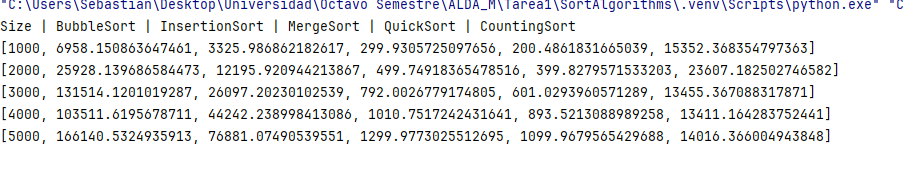
1. Para listas de tamaño mínimo 100 y tamaño máximo 1000, haciendo 10 ejemplos por tamaño y con MAX\_VALUE = 1000000

****

1. Para listas de tamaño mínimo 1000 y tamaño máximo 5000, haciendo 10 ejemplos por tamaño y con MAX\_VALUE = 100000

****

1. Para listas de tamaño mínimo 1000 y tamaño máximo 5000, haciendo 10 ejemplos por tamaño y con MAX\_VALUE = 1000000

****