



# *Análisis de Algoritmos de Ordenamiento*

Trabajo Integrador de Programación I

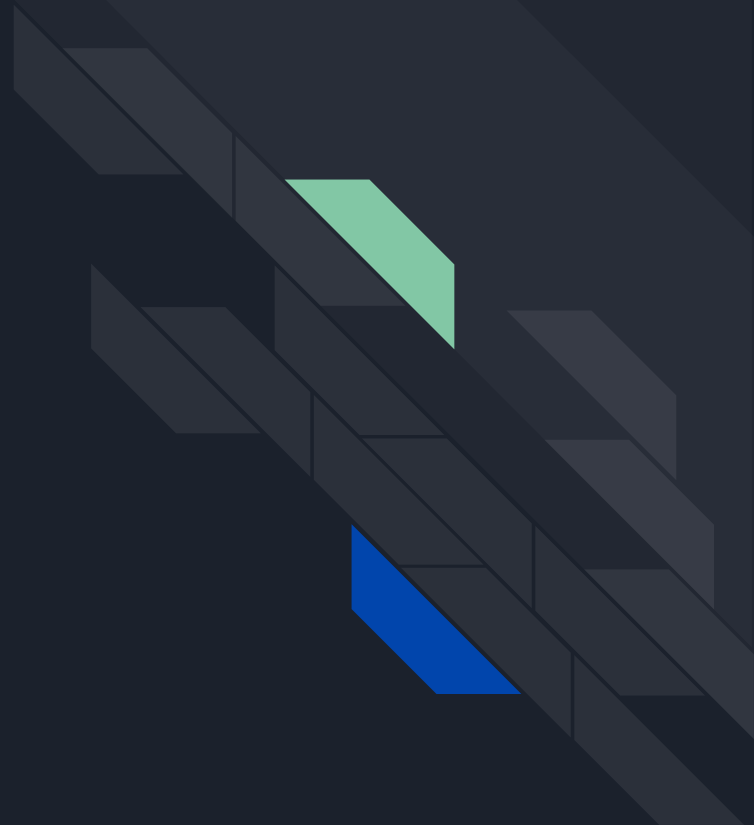
Por Ezequiel Alejandro Ventura y Abel Tomás Romero



# Introducción

## Objetivo del proyecto:

- **Comparar** la eficiencia de algoritmos de ordenamiento.
- **Evaluar** su rendimiento en listas con distintos niveles de orden.
- **Analizar** su comportamiento práctico y teórico.



# Marco Teórico

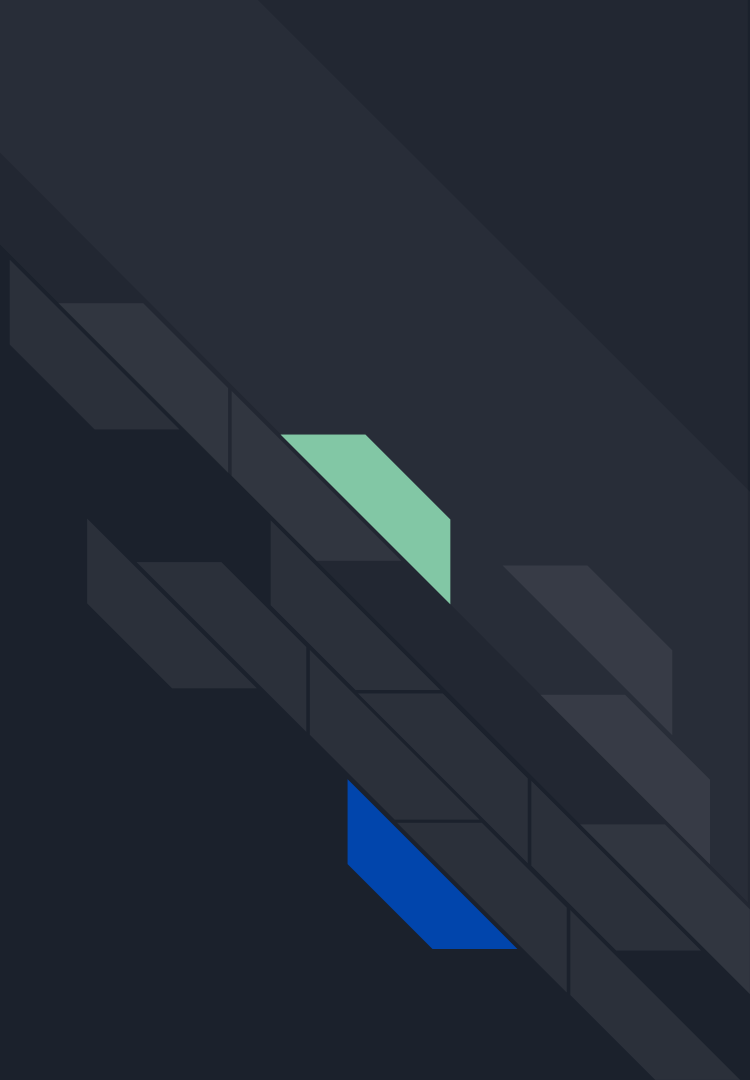
## ¿Qué es un Algoritmo de ordenamiento?

- Proceso para **reorganizar datos** en un orden específico.
- Fundamental para **búsquedas, acceso y organización** de información.
- Existen múltiples algoritmos con **enfoques y eficiencias** distintas.





## *¿Porque son importantes?*

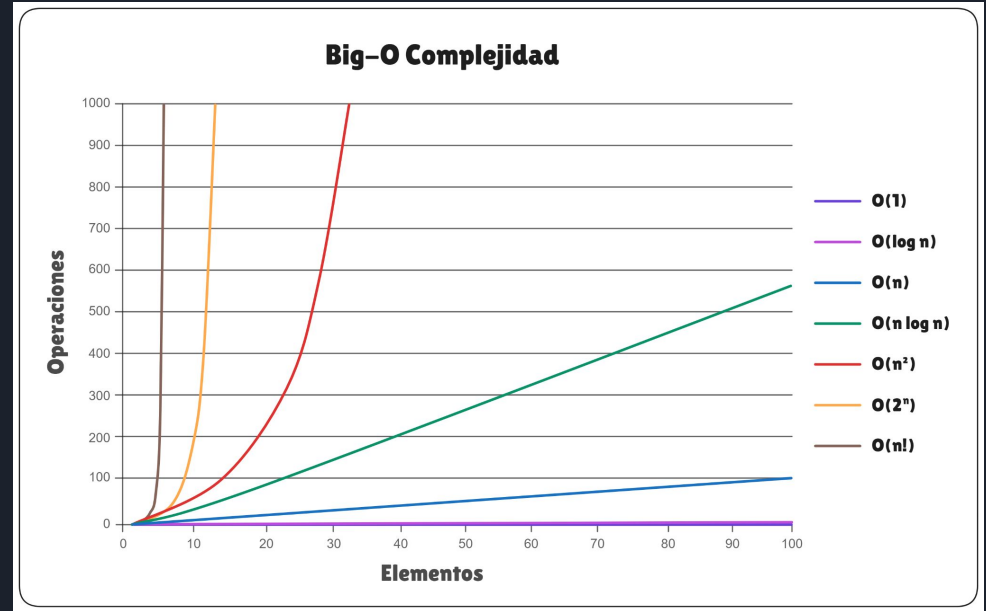
- Reducen la complejidad de problemas.
  - Se usan en búsquedas, bases de datos y estructuras de datos.
  - Base para muchos algoritmos modernos.
- 

# *Tipos de listas de entrada*

- **Aleatoria:** Los elementos están desordenados sin un patrón específico.
- **Ordenada:** Los elementos están en el orden correcto.
- **Ordenada inversamente:** Los elementos están en orden inverso.



- $O(1)$ : Constante
- $O(n)$ : Lineal
- $O(\log n)$ : Logaritmica
- $O(n \log n)$ : Log-lineal
- $O(n^2)$ : Cuadrática
- $O(2^n)$ : Exponencial
- $O(n!)$ : Factorial





# *Selection Sort*

Recorre la lista y selecciona el elemento más pequeño para colocarlo en su posición correcta, repitiendo el proceso para el resto de la lista.

**Complejidad temporal:**  $O(n^2)$

Peor caso	Caso medio	Mejor caso
$O(n^2)$	$O(n^2)$	$O(n)$



# *Insertion Sort*

Toma los elementos uno por uno y los inserta en la posición adecuada de una lista ordenada parcialmente.

**Complejidad temporal:**  $O(n^2)$

Peor caso	Caso medio	Mejor caso
$O(n^2)$	$O(n^2)$	$O(n)$





# *Bubble Sort*

Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Repite el proceso hasta que la lista esté ordenada.

**Complejidad temporal:**  $O(n^2)$

Peor caso	Caso medio	Mejor caso
$O(n^2)$	$O(n^2)$	$O(n)$



# Quick Sort

Utiliza un elemento “pivote” para dividir la lista en sublistas menores y mayores, y aplica recursivamente el mismo proceso en cada sublista.

**Complejidad temporal:**  $O(n \log n)$

Peor caso	Caso medio	Mejor caso
$O(n^2)$	$O(n \log n)$	$O(n \log n)$



# Merge Sort

Divide la lista a ordenar en sublistas más pequeñas, las ordena individualmente, y luego las fusiona de nuevo en una lista ordenada.

**Complejidad temporal:**  $O(n \log n)$

Peor caso	Caso medio	Mejor caso
$O(n \log n)$	$O(n \log n)$	$O(n \log n)$



# Tim Sort

Divide los datos en pequeños segmentos y ordenándolos mediante ordenación por inserción. Estos segmentos, se fusionan mediante ordenación por fusión.

**Complejidad temporal:**  $O(n \log n)$

Peor caso	Caso medio	Mejor caso
$O(n \log n)$	$O(n \log n)$	$O(n)$

The background is a dark navy blue. In the top-left corner, there are two overlapping geometric shapes: a blue parallelogram and a light green parallelogram. In the top-right corner, there is a grey, 3D-rendered circuit board pattern. In the bottom-left corner, there is a circular inset showing a detailed, high-magnification view of a printed circuit board (PCB) with various electronic components and solder points. The title 'Caso Práctico' is centered on the right side of the image in a light green, italicized, sans-serif font.

# *Caso Práctico*



# *Resultados Obtenidos*

- **Tim Sort** fue el algoritmo más rápido en todos los escenarios, gracias a su capacidad adaptativa.
- **Merge Sort** y **Quick Sort** también mostraron buen **rendimiento** y **estabilidad**, especialmente con listas aleatorias y ordenadas.
- Los algoritmos **cuadráticos** (Selection, Insertion y Bubble Sort) fueron **mucho más lentos**, especialmente con listas grandes e inversamente ordenadas.
- **Insertion Sort** tuvo buen desempeño solo con listas ya ordenadas.
- Se observaron **diferencias claras según el tipo y tamaño de lista**, lo que confirmó los análisis teóricos con datos reales.

# Conclusiones y cierre

- ✓ Aprendimos a **aplicar conceptos teóricos en un proyecto real**, entendiendo mejor el comportamiento y la eficiencia de los algoritmos.
- ✓ El trabajo ayudó a mejorar **habilidades en programación estructurada**, modularización de código y diseño de un sistema interactivo.
- ✓ En equipo resolvimos dificultades como la **organización del código**, **validaciones** y **diseño del menú**, mediante la comunicación y pruebas.
- ✓ Fue una experiencia útil para futuros proyectos donde el **rendimiento** y la **elección de algoritmos** sean clave
- ✓ Observamos cómo, a lo largo del tiempo, los algoritmos fueron mejorando su **eficiencia** en base al **manejo de datos** cada vez más **complejos**, como los que tenemos hoy en día.

*¡Gracias por ver el video!*