**Guion para Presentación en Video (20 minutos)**  
**Título:** *Ordenamiento y Búsqueda en Estructuras de Datos: Teoría, Implementación y Resultados*  
**Expositores:** Nikolas Barroso e Iván Huarcaya

### ⏱️ Duración estimada: 20 minutos

### 🎬 Introducción (2:00 min) — *Nikolas*

Hola, somos estudiantes de la Tecnicatura en Programación. Mi nombre es Nikolas Barroso y me acompaña mi compañero Iván Huarcaya. En este video vamos a presentarles nuestro trabajo integrador de la materia Programación I. El tema que abordamos es uno de los fundamentos más importantes de la informática: los **algoritmos de búsqueda y ordenamiento**.

Tal como mostramos en la primera diapositiva, estos algoritmos optimizan operaciones clave como la recuperación de datos, la clasificación y el análisis de grandes volúmenes de información. La forma en que seleccionamos y aplicamos estos algoritmos impacta directamente en el rendimiento de nuestras aplicaciones y la experiencia del usuario.

A lo largo de esta presentación vamos a explicar su funcionamiento, sus ventajas y desventajas, y cómo los implementamos en Python. También vamos a mostrar los resultados que obtuvimos al probarlos sobre diferentes conjuntos de datos.

### 🎯 Objetivos y contexto del trabajo (2:00 min) — *Iván*

Este trabajo surge de la necesidad de comprender cómo mejorar la eficiencia de nuestras aplicaciones usando herramientas fundamentales: los algoritmos. Tal como se observa en la diapositiva de objetivos, nos propusimos:

1. Describir y analizar el funcionamiento de cinco algoritmos fundamentales: tres de ordenamiento y dos de búsqueda.
2. Implementarlos usando el lenguaje de programación Python.
3. Comparar su desempeño en diferentes contextos.
4. Formular recomendaciones prácticas según el tipo de problema.
5. Concientizar sobre la importancia de tomar decisiones algorítmicas bien fundamentadas.

Nuestro enfoque integró teoría, codificación y experimentación, buscando entender qué algoritmos funcionan mejor según el tipo y tamaño de los datos.

### 📚 Fundamentos de búsqueda (3:30 min) — *Nikolas*

Comencemos por el concepto de búsqueda. Como se explica en la diapositiva del marco teórico, la búsqueda es el proceso de localizar un elemento específico dentro de una estructura de datos. Puede realizarse en listas, arreglos o incluso en estructuras más complejas como árboles y tablas hash.

La eficiencia de la búsqueda varía en función del algoritmo. Analizamos dos tipos principales:

* **Búsqueda lineal**: como muestra la diapositiva, consiste en recorrer uno por uno todos los elementos hasta encontrar el que buscamos. Es fácil de implementar y no requiere que los datos estén ordenados. Sin embargo, su complejidad O(n) la hace poco recomendable para listas grandes.
* **Búsqueda binaria**: divide el conjunto ordenado en mitades sucesivas. Su complejidad es O(log n), lo que la hace muy eficiente. Pero requiere una precondición importante: los datos deben estar previamente ordenados.

Ambos métodos tienen sus aplicaciones, y conocer sus límites es clave para elegir correctamente.

### 🧮 Fundamentos de ordenamiento (3:30 min) — *Iván*

El ordenamiento, tal como muestra la siguiente diapositiva, es una operación que reorganiza los datos para optimizar procesos como la búsqueda y el análisis.

Existen múltiples algoritmos de ordenamiento, y cada uno responde mejor a distintos escenarios. Evaluamos cinco de los más utilizados:

* **Bubble Sort**: muy intuitivo pero ineficiente para grandes volúmenes, con una complejidad de O(n²). Compara elementos adyacentes y los intercambia si están desordenados.
* **Quick Sort**: selecciona un pivote y divide el conjunto en subgrupos. En promedio, es muy eficiente, con O(n log n). Sin embargo, puede alcanzar O(n²) si el pivote se elige mal.
* **Selection Sort**: selecciona el menor elemento en cada iteración. Es estable y predecible, pero su eficiencia es baja.
* **Insertion Sort**: construye la lista ordenada insertando cada nuevo elemento. Es ideal para listas pequeñas o parcialmente ordenadas.
* **Merge Sort**: divide y conquista. Es consistente, estable, y garantiza O(n log n), aunque requiere más memoria. Muy útil en ordenamiento externo o en contextos que requieren estabilidad.

Esta clasificación, presentada gráficamente en la diapositiva, nos permitió entender qué algoritmo conviene usar en cada caso.

### 💻 Implementaciones en Python (4:00 min) — *Nikolas e Iván*

.

Casos de Aplicación Práctica (4:00 min) — Iván

Iván: > Para ilustrar la importancia de estos algoritmos en escenarios del mundo real, les presentamos dos casos de estudio centrados en un sistema de gestión para un refugio de animales. Estos ejemplos muestran cómo la elección adecuada de algoritmos de búsqueda y ordenamiento impacta directamente en la eficiencia y funcionalidad de una aplicación.

Caso 1: Sistema de Gestión para Refugio de Animales

Objetivo: Administrar la información de todos los animales del refugio de manera eficiente.

Aquí, la correcta gestión de los datos de cada animal es crucial. Hemos identificado varios algoritmos clave:

Búsqueda Binaria: Se utilizaría para localizar animales de forma rápida por su ID de chip o número de registro. Este método es extremadamente eficiente, con una complejidad de O(log n), siempre que los datos estén previamente ordenados.

Ordenamiento Rápido (Quick Sort): Ideal para organizar listas de animales por criterios como la fecha de ingreso, edad o estado de salud. Quick Sort es conocido por su eficiencia promedio de O(n log n).

Búsqueda de Interpolación: Una mejora de la búsqueda binaria que estimaría la posición del elemento, acelerando la búsqueda en datos bien distribuidos. Sería útil para buscar animales por peso o edad, donde los datos suelen tener una distribución relativamente uniforme.

La estructura de datos para cada animal se representaría con una clase Animal en Python, que contendría atributos como id\_chip, nombre, especie, edad, peso, fecha\_ingreso y estado\_salud.

En casos de uso reales, la búsqueda rápida de un animal por chip se realizaría con búsqueda binaria, lo que es muy eficiente en listas grandes. Para listados de animales listos para adopción, se usaría ordenamiento por edad o tiempo en el refugio. La complejidad de los algoritmos varía según el tamaño del refugio: para refugios pequeños, cualquier algoritmo es eficiente; para medianos, Quick Sort es recomendado; y para refugios grandes, la búsqueda binaria se vuelve esencial para consultas rápidas.

Nikolas

Caso 2: Sistema de Adopciones y Seguimiento

Objetivo: Administrar de forma efectiva el proceso de adopción y el seguimiento posterior de los animales, asegurando el mejor hogar para cada uno.

Este sistema se enfoca en la interacción entre adoptantes y animales, y en garantizar la compatibilidad:

Búsqueda Lineal con Filtros Múltiples: ¿Para qué? Para encontrar adoptantes que cumplan varios requisitos para un animal específico, como rango de edad, ingresos, tipo de vivienda o experiencia con animales. ¿Cómo funciona? Recorre la lista de adoptantes y aplica múltiples condiciones de filtro a la vez. Es flexible y no requiere que los datos estén ordenados, ideal para un "matching" complejo.

Ordenamiento Rápido (Quick Sort): ¿Para qué? Para generar listas de compatibilidad entre animales y adoptantes, o para organizar listas de espera. ¿Cómo funciona? Es un algoritmo de ordenamiento muy eficiente que permite organizar rápidamente los datos (por ejemplo, de mayor a menor compatibilidad), presentando los candidatos más adecuados primero.

Búsqueda por Interpolación: ¿Para qué? Para buscar adoptantes por rangos numéricos como edad o ingresos, donde estos datos suelen estar distribuidos de manera relativamente uniforme. ¿Cómo funciona? Similar a la búsqueda binaria, pero estima la posición del elemento basándose en su valor, lo que la hace más rápida en datos con distribución uniforme.

La estructura de datos base para este sistema implicaría el uso de clases como Adoptante (con detalles como nombre, edad, ingresos y preferencias) y Adopcion (registrando el animal, el adoptante y las fechas de seguimiento), permitiendo una gestión detallada y un seguimiento completo.

Estos ejemplos demuestran cómo la selección estratégica de algoritmos es crucial para el rendimiento y la utilidad de cualquier sistema de software.

### 📊 Análisis de resultados (2:30 min) — *Nikolas*

Tal como se ve en la diapositiva de resultados, QuickSort tuvo el mejor rendimiento promedio. MergeSort fue el más estable, especialmente útil cuando se requiere ordenamiento constante sin importar la entrada.

BubbleSort, como era de esperarse, fue el más lento. En algunos casos tardó hasta 10 veces más que QuickSort.

En búsqueda:

* La binaria superó ampliamente a la lineal, pero solo cuando la lista estaba ordenada.
* La búsqueda lineal sigue siendo útil para listas pequeñas o no ordenadas.

Estas diferencias refuerzan la importancia de elegir el algoritmo según el tipo de datos y las condiciones del problema.

### ✅ Recomendaciones y buenas prácticas (2:00 min) — *Iván*

A partir de todo este trabajo, formulamos algunas recomendaciones prácticas, que también están reflejadas en la diapositiva correspondiente:

* Para listas grandes, usar QuickSort o MergeSort.
* En listas pequeñas o ya parcialmente ordenadas, Insertion Sort es excelente.
* Evitar Bubble Sort en producción.
* Usar búsqueda binaria solo en listas ordenadas.
* Considerar estructuras hash cuando se requiere acceso instantáneo.

Evaluar bien el tipo de entrada, la necesidad de estabilidad y los recursos disponibles es clave.

### 🧠 Reflexión final (1:30 min) — *Nikolas*

Este trabajo nos permitió comprobar que la eficiencia de un programa no solo depende del lenguaje o del hardware, sino de las decisiones algorítmicas que tomamos. Entendimos cómo conceptos como complejidad computacional, estabilidad y uso de memoria afectan el rendimiento.

Además, la práctica de implementación y análisis nos dio herramientas valiosas para enfrentar desafíos reales en el desarrollo de software.

**Duración total aproximada: 20 minutos**  
**Recomendaciones técnicas:** acompañar con las diapositivas proporcionadas, que incluyen:

* Objetivos y motivación del trabajo.
* Conceptos teóricos con gráficos.
* Comparación visual de algoritmos.
* Resultados y recomendaciones finales.

**Sugerencia opcional:** utilizar efectos de transición suave y animaciones breves para mostrar pasos de los algoritmos (como la división en QuickSort o la comparación en búsqueda binaria).