

**Universidad Tecnológica Nacional**  
**Tecnicatura Universitaria en Programación a Distancia**  
**Programación 1**  
**Comisión 5 – Eric Suárez - Gonzalo Vega**  
**Trabajo Práctico Integrador**



Título del trabajo: **“Algoritmos de Búsqueda y Ordenamiento”**

- Alumnos: Eric Suárez Dubs, [ericgodzilladubs@gmail.com](mailto:ericgodzilladubs@gmail.com)

Gonzalo Vega, [gonzaarg03@gmail.com](mailto:gonzaarg03@gmail.com)

- Materia: **Programación 1**
- Profesor/a: **Walter Pintos**
- Fecha de Entrega: **09/06/2025**

**Índice**

1. Introducción	3
2. Marco Teórico	4
3. Caso Práctico	5
4. Metodología Utilizada	8
5. Resultados Obtenidos	9
6. Conclusiones	10
7. Bibliografía	11
8. Anexos	12

**Universidad Tecnológica Nacional**  
**Tecnicatura Universitaria en Programación a Distancia**  
**Programación 1**  
**Comisión 5 – Eric Suárez - Gonzalo Vega**  
**Trabajo Práctico Integrador**

## **1. Introducción**

### ¿Por qué se eligió este tema?

Se eligió la búsqueda binaria como tema de desarrollo para este trabajo con el fin de mejorar nuestro entendimiento de los algoritmos. A través de prueba y error, uno es capaz de encontrar nuevas soluciones a viejos problemas así como también conocer y familiarizarse con todas las herramientas necesarias o requeridas para llevar a cabo un desarrollo y un producto final profesional.

Esto entrega un desafío que, aunque sencillo o difícil para algunos, es capaz de brindar una mejor forma de entender el cómo, dónde y cuándo se deben implementar todas estas herramientas para asegurar un proyecto digno de nivel universitario

### ¿Qué importancia tiene en la programación?

La búsqueda y el ordenamiento son pilares fundamentales en la programación, ya que representan operaciones esenciales para el tratamiento, análisis y organización de datos. Casi cualquier aplicación, sistema o software que procese información —ya sea en forma de listas, registros, matrices o bases de datos— hace uso de algún tipo de algoritmo de búsqueda u ordenamiento en algún momento de su ejecución. Estos algoritmos no solo permiten acceder a la información de manera rápida y precisa, sino que también facilitan tareas más complejas como la clasificación, el filtrado, la comparación y la toma de decisiones automatizadas.

El dominio de estas técnicas permite abordar una gran variedad de problemas con mayor eficacia, ya que implementar correctamente un algoritmo adecuado para una situación específica puede traducirse en un uso más eficiente de los recursos del sistema, una reducción significativa en el tiempo de respuesta, y una experiencia de usuario más fluida y satisfactoria.

De hecho, en aplicaciones que manejan grandes volúmenes de datos —como sistemas de búsqueda, motores de recomendación, o plataformas financieras— el buen uso de estos algoritmos puede marcar la diferencia entre una solución funcional y una completamente ineficiente.

Además, los algoritmos de búsqueda y ordenamiento sirven como base para una comprensión más profunda de las estructuras de datos, como arreglos, listas enlazadas, árboles o tablas hash. Comprender cómo interactúan estos elementos permite diseñar soluciones más robustas, flexibles y adaptables. Por otro lado, al enfrentarse a problemas de programación, estos algoritmos entrenan el pensamiento lógico y estructurado, fomentando un enfoque paso a paso que es indispensable para escribir código limpio, mantenible y optimizado.

La importancia de estos conceptos no se limita a su uso directo, sino que también influye en la capacidad del programador para comprender algoritmos más avanzados o híbridos, y para participar en la mejora continua del rendimiento de sus aplicaciones. En muchos procesos de selección técnica y entrevistas laborales, los algoritmos de búsqueda y ordenamiento son evaluados precisamente porque demuestran no solo conocimientos técnicos, sino también habilidades de razonamiento, análisis de eficiencia y capacidad de resolución de problemas complejos.

### ¿Qué objetivos se propone alcanzar con el desarrollo del trabajo?

El desarrollo de este trabajo tiene como objetivo principal profundizar en la comprensión teórica y práctica de los algoritmos de búsqueda y ordenamiento, dos herramientas fundamentales en el mundo de la programación y las ciencias de la computación. Entender cómo funcionan estos algoritmos a nivel estructural y lógico, sino que también ser capaces de aplicarlos correctamente en una amplia variedad de contextos, evaluando su eficiencia, complejidad y pertinencia según el problema a resolver. Esto incluye el análisis de tiempos de ejecución, la cantidad de operaciones realizadas por cada algoritmo y su comportamiento en distintos escenarios, como listas pequeñas, grandes o parcialmente ordenadas.

A través del estudio sistemático de estos algoritmos, se desarrollarán habilidades analíticas y de resolución de problemas, esenciales para cualquier profesional de la informática. Del mismo modo, se pretende ganar una comprensión más sólida sobre cómo elegir el algoritmo más adecuado en función de criterios como la eficiencia, la claridad del código, la facilidad de implementación y los requerimientos del sistema. Este criterio de selección es fundamental para lograr soluciones más limpias, rápidas y escalables en entornos de programación reales.

Por último, este trabajo también busca fomentar la reflexión crítica sobre la relación entre teoría y práctica, invitando al estudiante a observar cómo conceptos abstractos se traducen en soluciones concretas que afectan directamente el rendimiento de un programa. De este modo, se espera no solo adquirir conocimientos técnicos, sino también una visión integral que permita aplicar estas herramientas en proyectos futuros, ya sea en el ámbito académico o en el profesional, con un enfoque riguroso, eficiente y fundamentado.

## **2. Marco Teórico**

La búsqueda binaria es un algoritmo eficiente utilizado para localizar un elemento específico dentro de una estructura de datos ordenada, generalmente una lista o un arreglo. Es uno de los métodos de búsqueda más importantes en la informática debido a su bajo costo computacional en comparación con métodos más simples, como la búsqueda lineal. Este algoritmo se fundamenta en el paradigma de divide y vencerás, lo que le permite reducir significativamente el número de comparaciones necesarias para encontrar un valor.

La búsqueda binaria parte del supuesto de que la colección de datos está ordenada. El algoritmo comienza comparando el valor buscado con el elemento ubicado en el centro de la estructura. Si el

valor coincide, la búsqueda finaliza. Si el valor buscado es menor que el elemento central, la búsqueda continúa en la mitad inferior del conjunto; si es mayor, en la mitad superior. Este proceso se repite recursiva o iterativamente, dividiendo a la mitad el conjunto en cada paso, hasta encontrar el elemento deseado o hasta que el subconjunto a evaluar quede vacío.

Este enfoque permite que la búsqueda binaria tenga una complejidad temporal de  $O(\log n)$  en el peor de los casos, lo que representa una mejora significativa frente a la búsqueda lineal, cuya complejidad es  $O(n)$ . Gracias a esta eficiencia, la búsqueda binaria se utiliza con frecuencia en algoritmos más complejos, motores de búsqueda, sistemas de gestión de bases de datos y bibliotecas estándar de programación.

Una de las principales ventajas de la búsqueda binaria es su eficiencia, sobre todo cuando se trata de conjuntos de datos grandes. Al dividir el problema a la mitad en cada iteración, el número de comparaciones necesarias se reduce exponencialmente. Además, es un algoritmo sencillo de implementar y fácil de adaptar a funciones recursivas e iterativas, lo que lo convierte en una herramienta versátil en la programación.

Otra ventaja importante es su utilidad como base para resolver problemas más complejos. Muchos algoritmos de búsqueda avanzada, como las búsquedas en árboles binarios de búsqueda (BST) o en estructuras como los árboles AVL y B-trees, se fundamentan en la lógica de la búsqueda binaria.

Estudiar la búsqueda binaria permite al estudiante comprender conceptos clave como la división del problema, la recursividad, la eficiencia algorítmica y el análisis de complejidad. Su simplicidad estructural pero alto rendimiento la convierten en un ejemplo didáctico ideal para introducir la lógica algorítmica y la comparación entre distintas estrategias de resolución.

Desde una perspectiva académica y profesional, dominar la búsqueda binaria contribuye al desarrollo de habilidades esenciales en programación, tales como el análisis de estructuras de datos, la toma de decisiones basada en condiciones lógicas, y la optimización de procesos computacionales.

### 3. Caso Práctico

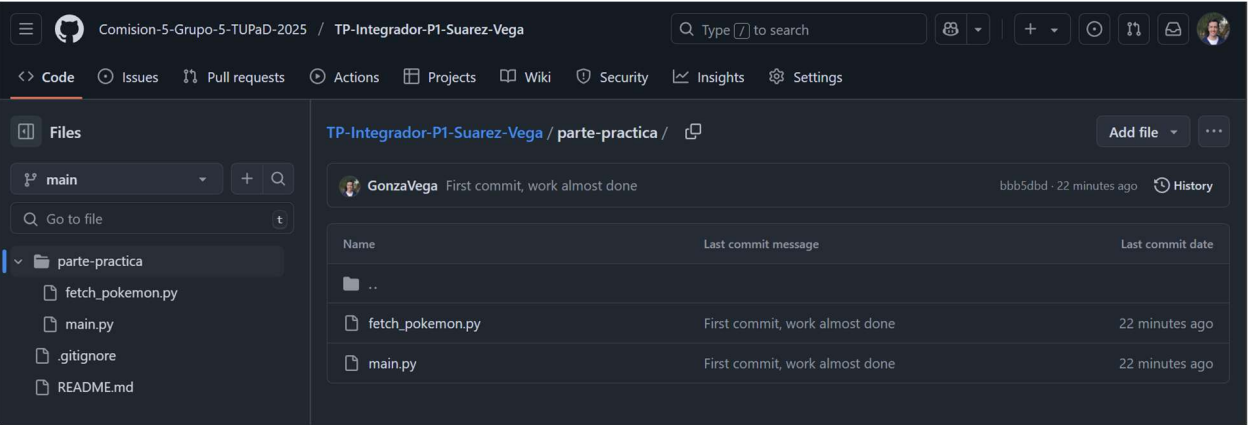
#### Descripción del problema:

El caso práctico desarrollado consiste en implementar un sistema de búsqueda binaria aplicado a una lista de nombres de Pokémon obtenida desde la API pública PokeAPI. El objetivo es permitir que el usuario ingrese el nombre de un Pokémon y que el sistema determine, mediante búsqueda binaria, si se encuentra en la Pokédex. Además, se muestra el tiempo que tardó la búsqueda y se

recupera información complementaria como los tipos del Pokémon y algunos de sus movimientos principales.

Código:

Estructura de carpetas:



Contenido de archivos:

main.py

```

1  import time
2  import fetch_pokemon
3
4  pokemones = (fetch_pokemon.obtener_pokemones()) #Con la función auxiliar se obtiene una lista ordenada.
5  parametro = input("Ingrese el nombre de un Pokémon para buscarlo, por ejemplo 'Charmander': ")
6  parametro = parametro.lower()
```

```

8  def busqueda_binaria(pokemon):
9      comienzo_ejecucion = time.time()#Tomamos el tiempo al inicio de ejecución.
10     inicio = 0
11     final = len(pokemones) - 1
12
13     while inicio <= final:
14         medio = (inicio + final) // 2
15         if pokemon == pokemones[medio]:
16             final_ejecucion = time.time()#Tomamos el tiempo al final de ejecución.
17             print(f"El tiempo de ejecución de la búsqueda fue de {final_ejecucion - comienzo_ejecucion}")#Calculamos el tiempo empleado por el algoritmo.
18             print(f"Se encontró el elemento a buscar, estaba en la posición {medio}")#Informamos en que posición se encuentra el medio de la búsqueda.
19             return pokemones[medio]
20         elif pokemon > pokemones[medio]:
21             print(f"Iteración de búsqueda, el medio se encuentra en la posición {medio}")#Informamos en que posición se encuentra el medio de la búsqueda.
22             inicio = medio + 1
23         elif pokemon < pokemones[medio]:
24             print(f"Iteración de búsqueda, el medio se encuentra en la posición {medio}")#Informamos en que posición se encuentra el medio de la búsqueda.
25             final = medio - 1
26         else:
27             return None
28     final_ejecucion = time.time()#Tomamos el tiempo al final de ejecución.
29     print(f"El tiempo de ejecución de la búsqueda fue de {final_ejecucion - comienzo_ejecucion}")#Calculamos el tiempo empleado por el algoritmo.
```

```

31 def maneja_busqueda_pokemon():
32     resultado = busqueda_binaria(parametro)#Con el parametro ingresado por el usuario, llamamos a la búsqueda binaria.
33
34     if resultado is None:#Si no se encuentra el Pokémon, notificamos al usuario.
35         return f"El Pokémon '{parametro.capitalize()}' no esta en la Pokedex, o el profesor todavía no lo descubre."
36     else:
37         pokemon_encontrado = fetch_pokemon.encuentra_pokemon(resultado)#Si se encuentra el pokémon, encuentra_pokemon() llama al endpoint individual.
38         nombre = pokemon_encontrado['name'].capitalize()
39         tipos = []
40         for tipo in pokemon_encontrado['types']:#A veces Los pokemones son de mas de un tipo, construimos un array de tipos.
41             tipos.append(tipo['type']['name'])
42         poderes = []
43         for tipo in pokemon_encontrado['moves'][:5]:#A veces Los pokemones tienen muchos poderes, construimos un array con Los 5 primeros poderes que tenga.
44             poderes.append(tipo['move']['name'])
45         tipos = ", ".join(tipos)
46         tipos = f"es de tipo: {tipos}" if len(tipos) == 1 else f"es de tipos: {tipos}"
47         poderes = ", ".join(poderes)
48         poderes = f"tiene este poder: {poderes}" if len(poderes) == 1 else f"tiene estos poderes: {poderes}"
49
50         return f"¡Encontramos tu Pokémon! Se llama {nombre}, {tipos} y {poderes}"
51
52
53 print(maneja_busqueda_pokemon())
```

fetch\_pokemon.py

```
fetch_pokemon.py M X
TP-Integrador-P1-Suarez-Vega > parte-practica > fetch_pokemon.py > encuentra_pokemon
1  from requests import get, RequestException
2
3  def obtener_pokemones():#Llamamos el endpoint del listado de pokemones, estableciendo el limite de items en 1000.
4      url = "https://pokeapi.co/api/v2/pokemon?limit=1000"
5
6      try:
7          respuesta = get(url, timeout=10)
8          respuesta.raise_for_status()#Buscamos que respuesta arroje un error si existe.
9          datos = respuesta.json()
10         nombres = [p["name"] for p in datos["results"]]
11
12         return sorted(nombres)#Ordenamos la lista devuelta para poder usar busqueda binaria.
13     except RequestException as error:#Si respuesta tiene error, lo informamos aquí.
14         print(f"Error al consultar la PokeAPI: {error}")
15         return []
16
17 def encuentra_pokemon(pokemon):#Buscamos un pokemon especifico.
18     url = f"https://pokeapi.co/api/v2/pokemon/{pokemon}"
19
20     try:
21         respuesta = get(url, timeout=10)
22         respuesta.raise_for_status()#Buscamos que respuesta arroje un error si existe.
23         return respuesta.json()
24     except RequestException as error:
25         print(f"Error al recuperar los datos de {pokemon.capitalize()}: {error}")#Si respuesta tiene error, lo informamos aquí.
26         return []
```

#### Decisiones de diseño:

- Se utilizó la PokeAPI por ser una fuente de datos abierta, estructurada y adecuada para trabajar con listas ordenadas.
- Se eligió ordenar alfabéticamente por nombre para aplicar búsqueda binaria, ya que este algoritmo requiere un conjunto ordenado por la clave de búsqueda.
- El algoritmo fue implementado de forma iterativa por simplicidad y claridad didáctica.
- Se limitaron los movimientos mostrados a los primeros cinco para evitar salidas excesivamente largas.

#### Validación del funcionamiento:

Se realizaron pruebas ingresando nombres válidos e inválidos (ej. bulbasaur, Machop, Perromon) para verificar la respuesta del sistema en ambos casos. Se midió el tiempo de ejecución con `time.time()` y se observó un comportamiento correcto en todas las búsquedas.

```
Windows PowerShell
PS A:\Documentos\UTN Programación\Primer Año\Primer Semestre\Programacion I\TP Integrador\Codigo\TP-In
tegrador-P1-Suarez-Vega\parte-practica> python main.py
Ingrese el nombre de un Pokémon para buscarlo, por ejemplo 'Charmander': bulbasaur
Iteración de búsqueda, el medio se encuentra en la posición 499
Iteración de búsqueda, el medio se encuentra en la posición 249
Iteración de búsqueda, el medio se encuentra en la posición 124
Iteración de búsqueda, el medio se encuentra en la posición 61
Iteración de búsqueda, el medio se encuentra en la posición 92
Iteración de búsqueda, el medio se encuentra en la posición 108
Iteración de búsqueda, el medio se encuentra en la posición 100
Iteración de búsqueda, el medio se encuentra en la posición 96
Iteración de búsqueda, el medio se encuentra en la posición 94
El tiempo de ejecución de la búsqueda fue de 0.001001119613647461
Se encontró el elemento a buscar, estaba en la posición 93
¡Encontramos tu Pokémon! Se llama Bulbasaur, es de tipos: grass, poison y tiene estos poderes: razor-w
ind, swords-dance, cut, bind, vine-whip
PS A:\Documentos\UTN Programación\Primer Año\Primer Semestre\Programacion I\TP Integrador\Codigo\TP-In
tegrador-P1-Suarez-Vega\parte-practica> python main.py
Ingrese el nombre de un Pokémon para buscarlo, por ejemplo 'Charmander': Machop
Iteración de búsqueda, el medio se encuentra en la posición 499
Iteración de búsqueda, el medio se encuentra en la posición 749
Iteración de búsqueda, el medio se encuentra en la posición 624
Iteración de búsqueda, el medio se encuentra en la posición 561
Iteración de búsqueda, el medio se encuentra en la posición 530
Iteración de búsqueda, el medio se encuentra en la posición 514
Iteración de búsqueda, el medio se encuentra en la posición 506
Iteración de búsqueda, el medio se encuentra en la posición 502
Iteración de búsqueda, el medio se encuentra en la posición 504
El tiempo de ejecución de la búsqueda fue de 0.0
Se encontró el elemento a buscar, estaba en la posición 503
¡Encontramos tu Pokémon! Se llama Machop, es de tipos: fighting y tiene estos poderes: karate-chop, me
ga-punch, fire-punch, ice-punch, thunder-punch
PS A:\Documentos\UTN Programación\Primer Año\Primer Semestre\Programacion I\TP Integrador\Codigo\TP-In
tegrador-P1-Suarez-Vega\parte-practica> python main.py
Ingrese el nombre de un Pokémon para buscarlo, por ejemplo 'Charmander': Perromon
Iteración de búsqueda, el medio se encuentra en la posición 499
Iteración de búsqueda, el medio se encuentra en la posición 749
Iteración de búsqueda, el medio se encuentra en la posición 624
Iteración de búsqueda, el medio se encuentra en la posición 686
Iteración de búsqueda, el medio se encuentra en la posición 655
Iteración de búsqueda, el medio se encuentra en la posición 639
Iteración de búsqueda, el medio se encuentra en la posición 631
Iteración de búsqueda, el medio se encuentra en la posición 635
Iteración de búsqueda, el medio se encuentra en la posición 633
Iteración de búsqueda, el medio se encuentra en la posición 632
El tiempo de ejecución de la búsqueda fue de 0.0009918212890625
El Pokémon 'Perromon' no está en la Pokedex, o el profesor todavía no lo descubre.
PS A:\Documentos\UTN Programación\Primer Año\Primer Semestre\Programacion I\TP Integrador\Codigo\TP-In
tegrador-P1-Suarez-Vega\parte-practica> |
```

4. Metodología Utilizada

El desarrollo del trabajo siguió una secuencia ordenada de pasos que incluyó tanto investigación teórica como experimentación práctica:

- Investigación previa: Se revisaron recursos como la documentación de la PokeAPI, artículos sobre búsqueda binaria y el módulo requests en Python. También se tomó como referencia el código base visto en clases.
- Diseño del algoritmo: Se decidió implementar la búsqueda binaria de forma iterativa, sobre una lista de nombres ordenados.
- Construcción del código: Se dividió el proyecto en dos archivos: uno para obtener los datos (fetch\_pokemon.py) y otro para ejecutar la búsqueda (main.py), promoviendo una separación clara de responsabilidades.
- Herramientas utilizadas:
  - Lenguaje: Python 3.12



- IDE: Visual Studio Code
  - Librerías: requests, time
  - Plataforma: sistema operativo Windows 11
  - Command Line: Windows PowerShell
- Trabajo colaborativo: El trabajo fue realizado en grupo. La parte teórica fue desarrollada por Eric Suárez Dubs, y el desarrollo del caso práctico, código y pruebas fue responsabilidad de Gonzalo Vega, asegurando así una distribución equilibrada de tareas.

## 5. Resultados Obtenidos

El sistema de búsqueda binaria desarrollado funcionó correctamente y permitió resolver el problema propuesto. Entre los logros más destacados se encuentran:

- Se logró implementar correctamente un algoritmo de búsqueda binaria funcional sobre datos reales obtenidos por API.
- El sistema identifica correctamente si un Pokémon existe o no dentro de la Pokédex y recupera su información relevante.
- Se midió el tiempo de búsqueda con precisión utilizando `time.time()` y se confirmó la eficiencia del algoritmo incluso con listas de 1000 elementos.
- Se logró una estructura de código clara y modular, que permite entender fácilmente cómo interactúan los distintos componentes.

### Casos de prueba

- Búsqueda de "bulbasaur" → encontrado correctamente en pocas iteraciones.
- Búsqueda de "Machop" → encontrado correctamente.
- Búsqueda de "Perromon" → no encontrado, con mensaje adecuado.

### Errores corregidos

- Se corrigió el manejo de mayúsculas/minúsculas al normalizar la entrada del usuario con `.lower()`.
- Se agregó control de errores en las llamadas a la API mediante `try-except` y `raise_for_status()`.

## Evaluación de rendimiento

- Se observó que la búsqueda binaria localiza elementos en menos de 10 iteraciones incluso con 1000 Pokémon.
- El tiempo de ejecución reportado fue menor a 0.01 segundos en la mayoría de los casos.

## 6. Conclusiones

La búsqueda binaria es un algoritmo fundamental dentro del estudio de las ciencias de la computación, debido a su simplicidad conceptual y alta eficiencia en la localización de datos dentro de colecciones ordenadas. A diferencia de métodos más básicos, como la búsqueda lineal, la búsqueda binaria reduce considerablemente el número de operaciones necesarias al dividir sistemáticamente el problema en cada iteración, lo que la convierte en una herramienta ideal para trabajar con grandes volúmenes de información.

La implementación de este algoritmo, aplicada a un caso práctico con datos reales obtenidos desde una API, permitió reforzar el vínculo entre teoría y práctica, así como adquirir herramientas de programación más robustas. También fortaleció habilidades como el análisis de eficiencia, el diseño modular y el uso adecuado de bibliotecas externas.

Comprender y aplicar este algoritmo no solo mejora el rendimiento de los programas, sino que también fomenta el pensamiento lógico y estructurado, esencial en cualquier entorno de desarrollo. Por estas razones, la búsqueda binaria continúa siendo un pilar clave en la formación de programadores y en el diseño de sistemas computacionales eficientes.

**Universidad Tecnológica Nacional**  
**Tecnicatura Universitaria en Programación a Distancia**  
**Programación 1**  
**Comisión 5 – Eric Suárez - Gonzalo Vega**  
**Trabajo Práctico Integrador**

**7. Bibliografía**

- GeeksforGeeks. (s.f.). Binary Search. <https://www.geeksforgeeks.org/binary-search/>.
- Stack Overflow. (s.f.). Binary search in Python. Stack Exchange Inc. <https://stackoverflow.com>.
- GitHub. (s.f.). GitHub Discussions. <https://github.com/discussions>.
- PokeAPI. (n.d.). *PokeAPI: The RESTful Pokémon API*. <https://pokeapi.co/>.
- OpenAI. (2024). *ChatGPT, modelo GPT-4* [asistencia técnica y conceptual].
- Coding Tree. (2020). *¿Qué es la BÚSQUEDA BINARIA y por qué es tan GENIAL?* [Video]. YouTube. <https://www.youtube.com/watch?v=L7BPRzPzZtY>.
- Academia Coder. (2019). *BÚSQUEDA BINARIA con Python 3 - Clase #56 del curso de Python (Parte 1)* [Video]. YouTube. [https://www.youtube.com/watch?v=6-e\\_kVZkECI](https://www.youtube.com/watch?v=6-e_kVZkECI).
- Academia Coder. (2019). *BÚSQUEDA BINARIA con Python 3 - Clase #57 del curso de Python (Parte 2)* [Video]. YouTube. <https://www.youtube.com/watch?v=7rV0khDePlw>.
- Tecnicatura. (2025). *BÚSQUEDA* [Video]. YouTube. <https://www.youtube.com/watch?v=gJlQTq80llg>.
- Tecnicatura. (2025). *ORDENAMIENTO* [Video]. YouTube. <https://www.youtube.com/watch?v=xntUhrhtLaw>.

## 8. Anexos.

Repositorio Github: <https://github.com/Comision-5-Grupo-5-TUPaD-2025/TP-Integrador-P1-Suarez-Vega>.

Video Youtube: <https://youtu.be/Lar0d1uv2a4>

Diapositivas utilizadas:



## ¿Por qué elegimos la búsqueda binaria?

### Diapositiva 1 – Introducción al tema

Hola, soy Eric Suarez Dubs de la comisión 5, y hoy les presento nuestro trabajo sobre búsqueda binaria.

Elegimos este tema porque queríamos profundizar nuestro conocimiento en algoritmos, que son una parte fundamental en la programación.

Creemos que a través de la práctica, la prueba y el error, uno puede entender mejor cómo resolver problemas reales con herramientas profesionales.

Este tipo de desafío, aunque pueda parecer sencillo, nos permite aprender cuándo y cómo aplicar distintas soluciones de forma eficiente, y desarrollar habilidades que nos van a servir tanto en el ámbito académico como profesional.



**Universidad Tecnológica Nacional**

**Tecnicatura Universitaria en Programación a Distancia**

**Programación 1**

**Comisión 5 – Eric Suárez - Gonzalo Vega**

**Trabajo Práctico Integrador**

## Importancia en la programación

¿Por qué es importante la búsqueda en programación?

Porque casi cualquier software o sistema que maneje información necesita buscar o ordenar datos. Desde aplicaciones móviles hasta bases de datos o motores de búsqueda.

Estos algoritmos permiten acceder a la información de manera rápida, y también facilitan tareas como clasificar, comparar o tomar decisiones de forma automática.

Entender cómo funcionan estas técnicas mejora el rendimiento del sistema, hace el código más limpio, y nos permite usar los recursos del equipo de forma más eficiente.

Por eso, conocer y dominar este tipo de algoritmos es clave para cualquier programador.

## Objetivos del trabajo

El objetivo principal de este trabajo fue entender cómo funcionan los algoritmos de búsqueda y ordenamiento, tanto desde el punto de vista teórico como práctico.

Queremos ser capaces de aplicar estos conocimientos en diferentes contextos, sabiendo cuándo usar cada algoritmo según su eficiencia y su complejidad.

Analizamos cómo se comportan con listas pequeñas, grandes, o parcialmente ordenadas, y medimos cuántas operaciones realiza cada uno.

Además, este análisis nos ayuda a desarrollar habilidades de pensamiento lógico, resolución de problemas y toma de decisiones técnicas.

En resumen, buscamos ir más allá del código y entender qué hay detrás de una buena solución.

---

## ¿Qué es la búsqueda binaria?

La búsqueda binaria es un algoritmo que sirve para encontrar un elemento dentro de una lista ordenada.

A diferencia de una búsqueda simple, que revisa cada elemento uno por uno, la búsqueda binaria divide el problema a la mitad en cada paso.

Compara el valor que buscamos con el del centro de la lista. Si es menor, busca en la mitad inferior; si es mayor, en la mitad superior.

Esto se repite hasta encontrar el valor o hasta que no quede nada por revisar.

Este algoritmo tiene una complejidad de  $O(\log n)$ , lo que significa que funciona muy bien incluso con grandes volúmenes de datos.

Es simple de entender y fácil de implementar, pero al mismo tiempo muy poderoso.



---

## ¿Cómo funciona la búsqueda binaria?

Para que la búsqueda binaria funcione correctamente, la lista debe estar ordenada.


El algoritmo empieza comparando el número que buscamos con el elemento del medio. Si son iguales, lo encontró. Si no, descarta la mitad que no sirve, y repite el proceso con la otra mitad.

Por ejemplo, si estamos buscando el número 42 en una lista de cien elementos, primero se fija en el elemento número 50.

Si 42 es menor, busca en la mitad de abajo. Si es mayor, en la mitad de arriba.

De esta forma, en cada paso se reduce el espacio de búsqueda a la mitad, lo que ahorra muchísimo tiempo.


Este proceso se puede hacer con un bucle o de forma recursiva, y su lógica también se usa en estructuras más complejas como árboles binarios.



---

## Caso Práctico - Búsqueda Binaria

TUPaD - Programación 1 - Comisión 5  
Alumnos: Eric Suárez Dubs, Gonzalo Vega.



Universidad Tecnológica Nacional

Tecnicatura Universitaria en Programación a Distancia

Programación 1

Comisión 5 – Eric Suárez - Gonzalo Vega

Trabajo Práctico Integrador

## Conclusión.

- La **búsqueda binaria** es un algoritmo esencial en ciencias de la computación por su eficiencia y simplicidad lógica.
- Permite **localizar datos en colecciones ordenadas** reduciendo exponencialmente el número de comparaciones necesarias.
- Este proyecto permitió **aplicar el algoritmo en un contexto real**, integrando APIs externas, diseño modular y análisis de rendimiento.
- Se fortalecieron habilidades clave: pensamiento lógico, programación estructurada, uso de librerías, y manejo de errores.
- Comprender este tipo de algoritmos es clave para afrontar desafíos más complejos en el desarrollo de software y sistemas.