**Trabajo Práctico Integrador**

**Comparación entre Búsqueda Binaria y Lineal**

**Alumnos**

Miguel Andrés Del Valle Rodriguez, Ignacio Nicolas Rodriguez Villordo

**Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.**

**Programación 1**

**Docente Titular**

Francisco Luis Quarñolo

5 de Junio de 2025

Contenido

[Introducción 3](#_Toc200312429)

[Marco teórico 3](#_Toc200312430)

[Búsqueda Binaria 3](#_Toc200312431)

[Ventaja 3](#_Toc200312432)

[Desventaja 4](#_Toc200312433)

[Búsqueda Lineal 4](#_Toc200312434)

[Ventajas 4](#_Toc200312435)

[Desventajas 4](#_Toc200312436)

[Caso práctico 4](#_Toc200312437)

[Reglas 4](#_Toc200312438)

[Ejemplo 4](#_Toc200312439)

[Implementación en Python 5](#_Toc200312440)

[Metodología 7](#_Toc200312441)

[Resultados Obtenidos 7](#_Toc200312442)

[Conclusión 9](#_Toc200312443)

[Bibliografía 10](#_Toc200312444)

# Introducción

En la programación, la búsqueda y el ordenamiento son dos operaciones fundamentales que, cuando se dominan correctamente, permiten manejar la información de manera eficiente. Además, una buena implementación de estos algoritmos marca la diferencia entre un programa optimizado y uno lento. Por eso, su conocimiento teórico y práctico es esencial para cualquier programador.

En este informe pondremos en práctica los conocimientos adquiridos durante el cursado de la materia Programación 1, implementando un programa que utiliza un algoritmo de búsqueda binaria, y luego lo compararemos con una implementación equivalente que utiliza búsqueda lineal.

# Marco teórico

Para poder comprender adecuadamente la solución implementada, es necesario conocer los algoritmos de búsqueda utilizados: la búsqueda binaria y la búsqueda lineal.

## Búsqueda Binaria

La búsqueda binaria es un algoritmo eficiente que permite encontrar un elemento en una lista ordenada, reduciendo el espacio de búsqueda a la mitad en cada paso.

Pasos del algoritmo:

1. Se selecciona el valor medio del rango actual.
2. Se compara ese valor con el objetivo.
3. Si es mayor o menor, se descarta la mitad correspondiente y se repite el proceso en la mitad restante.
4. Se repite hasta encontrar el valor o confirmar que no existe.

### Ventaja

* Complejidad temporal O(log n), lo que significa que es muy eficiente.

### Desventaja

* Solo puede aplicarse sobre listas ordenadas. En listas desordenadas, puede devolver resultados incorrectos.

## Búsqueda Lineal

La búsqueda lineal es un algoritmo sencillo que recorre los elementos de una lista uno por uno hasta encontrar el valor buscado o llegar al final.

### Ventajas

* No requiere que la lista esté ordenada.

Desventajas

* Su tiempo de ejecución es O(n), lo que significa que puede ser muy ineficiente para listas largas.

## Caso práctico

Esta situación problemática fue extraída del sitio LeetCode, que es una plataforma en línea para la preparación de entrevistas de codificación.

Una persona debe tomar una secuencia de trenes de longitud n, con un tiempo límite dado (en horas decimales), y se debe encontrar la velocidad mínima a la que deben viajar los trenes para que llegue a destino a tiempo.

Reglas

* La velocidad debe ser un número entero mayor o igual a 1 y menor o igual a 10.000.000.
* Cada tren (excepto el último) solo puede partir en una hora entera (por lo que puede haber que esperar).
* La llegada del último tren no se redondea.
* Si no es posible llegar a tiempo, se devuelve -1.

### Ejemplo

trenes = [1, 3, 2], hora = 2.7

Con velocidad 3:

* 1/3 = 0.33 → se redondea a 1 hora
* 3/3 = 1 hora → total acumulado: 2 horas
* 2/3 = 0.67 horas (no se redondea)

Tiempo total = 2.67 horas → llega a tiempo

## Implementación en Python

Texto

El contenido generado por IA puede ser incorrecto.

Primero importamos la librería math, que nos será de utilidad más adelante.

Definimos una función *calcularVelocidadBinaria*, que recibe un arreglo *trenes* con la cantidad de trenes que se tienen que recorrer y un valor *hora* que recibe las horas limites que se pueden utilizar para que la persona llegue a su trabajo.

Definimos izquierda y derecha, que son los límites iniciales de velocidad y una variable valor que será la velocidad final a la que viajaran los trenes, aunque se inicializa en -1 para que, en caso de no llegar, muestra ese valor como error. Luego, con un bucle while, tomamos el valor del medio y luego consultamos a la función *llega*, que se explicará luego, si efectivamente, con esa velocidad, llegará a destino antes de las horas limite, en caso de llegar, se procede a asignar la variable medio a valor, ya que va a ser una posible respuesta, y luego guardamos el valor “medio - 1” en la variable *derecha*, para utilizar la mitad menor y tratar de encontrar una velocidad más baja. En caso de que no se llegue, se asigna el valor “medio + 1” a *izquierda* utilizar la mitad mayor y tratar de encontrar una respuesta que se adapte a los parámetros ingresados.

Después, tenemos la función *calcularVelocidadLineal*, ésta es una implementación del mismo problema, pero de forma lineal. Con un bucle *while*, que iterará mientras que *velocidad* sea menor al límite, se pasará por los elementos uno por uno, hasta que se encuentre el buscado, devolviendo el valor o, en el peor caso, no encontrará el elemento y saldrá del bucle para devolver -1, lo que significa que no habrá encontrado el elemento.

La función *llega* recibe por parámetros el arreglo de trenes, la velocidad (el valor medio de la función *calcularVelocidadBinaria* o el elemento actual en *calcularVelocidadLineal*) y el límite de horas. El funcionamiento es simple: primero se declara la variable tiempo, que será la que se comparará al final con el parámetro hora para ver si efectivamente llega o no, luego, con un bucle *for*, se procede dividir la distancia de los trenes con la velocidad actual, aquí se utiliza la función *ceil* de la librería *math* para redondear todos los viajes que lleguen en una hora decimal, menos el último, hacía el entero próximo. Por último, se devolverá True si la hora es mayor al tiempo que toman los viajes, o False si es menor.

# Metodología

Para poder comprobar el correcto funcionamiento del programa y de nuestra implementación de búsqueda binaria, pedimos a la inteligencia artificial de Google, Gemini, que nos genere casos de prueba, que sigan el siguiente modelo:

* *calcularVelocidadBinaria([1,2,…,n], hora)* o *calcularVelocidadLineal([1,2,…,n], hora)*

Quedando los casos de prueba como se ven a continuación  
Texto

El contenido generado por IA puede ser incorrecto.

Por otro lado, agregamos las variables time\_start y time\_end al principio y final de cada ejecución, tanto binaria como lineal, para poder medir el tiempo que le toma a cada implementación.

# Resultados Obtenidos

Los resultados obtenidos y esperados fueron los siguientes:

|  |  |  |  |
| --- | --- | --- | --- |
| Función y parámetros | Resultado Esperado | Resultado Obtenido (binaria) | Resultado Obtenido (lineal) |
| calcularVelocidad([1,3,2], 6.0) | 1 | 1 | 1 |
| calcularVelocidad([1,3,2], 2.7) | 3 | 3 | 3 |
| calcularVelocidad([1,1,100000], 0.5) | -1 | -1 | -1 |
| calcularVelocidad([100], 1.0) | 100 | 100 | 100 |
| calcularVelocidad([5,5,5], 3.0) | 5 | 5 | 5 |
| calcularVelocidad([1,1,1], 3.0) | 1 | 1 | 1 |
| calcularVelocidad([10,20,30], 6.0) | 10 | 10 | 10 |
| calcularVelocidad([10,20,30], 5.5) | 12 | 12 | 12 |
| calcularVelocidad([1000000,1000000], 2.0) | 1000000 | 1000000 | 1000000 |
| calcularVelocidad([1,1,1,1,1,1,1,1,1,1], 10.0) | 1 | 1 | 1 |
| **Tiempo total:** | | 0.000610 s | 3.300266 s |

Estas pruebas demuestran, primeramente, que las pruebas fueron correctas y que nuestro código cumple con los requerimientos, por lo que resulta válido.

Por otro lado, tenemos la comparación entre el tiempo de ambas ejecuciones, en la que podemos ver que la búsqueda binaria es extremadamente más rápida que la lineal. Cuando la búsqueda lineal tarda más de 3 segundos en ejecutarse, la binaria no llega a demorar una milésima de segundo.

Si bien no se puede usar una búsqueda binaria en todos los casos, éste, en especial, aplica perfectamente para que sea utilizada, ya que los valores están ordenados (números del 1 al 10.000.000) y además es una lista inmensa, lo que la hace sumamente recomendable.

# Conclusión

Para cerrar este informe, podemos afirmar que la búsqueda binaria, siendo debidamente aplicada, es una herramienta muy fuerte en el hallazgo, o no, de datos. Toma una ínfima fracción de lo que demora la lineal.

# Bibliografía

* LeetCode. (s.f.). *Minimum Speed to Arrive on Time*. Recuperado el 2 de junio de 2025, de <https://leetcode.com/problems/minimum-speed-to-arrive-on-time/>
* Python Software Foundation. (s.f.). *Python 3 Documentation*. Recuperado el 02 de junio de 2025, de <https://docs.python.org/3/>
* Google. (s.f.). *Gemini*. Recuperado el 5 de junio de 2025, de [https://gemini.google.com/](https://gemini.google.com/app?hl=es-419)