# Boletín 1

Diego Ignacio Pérez Torres

## Resumen

Se analizaron tres algoritmos fundamentales de búsqueda:

- Búsqueda Secuencial (Linear Search)
- Búsqueda Binaria (Binary Search)
- Búsqueda Galloping (Galloping Search)

Estructura de datos utilizada: Arrays ordenados de enteros con tamaños que van desde 2^10 (1,024) hasta 2^20 (1,048,576) elementos.

Motivación: Comparar el rendimiento práctico de diferentes estrategias de búsqueda en arrays ordenados, validando las predicciones teóricas de complejidad temporal.

## Análisis teórico

**Secuencial**: consiste en recorrer un array elemento por elemento, desde el inicio (índice 0) hasta el final. En cada paso, compara el elemento actual con el valor buscado.

Si encuentra una coincidencia, el algoritmo termina y devuelve la posición del elemento.

Si llega al final del array sin encontrar el valor, el algoritmo concluye que el elemento no está presente.

Complejidades:

Peor Caso: O(n). Ocurre cuando el elemento está en la última posición o no se encuentra en el array, lo que obliga a recorrer los n elementos.

Caso Promedio: O(n). En promedio, se espera que el elemento se encuentre a mitad del array, requiriendo n/2 comparaciones, lo que sigue siendo de orden lineal.

Mejor Caso: O(1). El elemento se encuentra en la primera posición.

Espacial: O(1). No requiere memoria adicional, más allá de una variable para el índice

**Binario** (se debe tener como base que el array este ordenado y el tamaño del array): comienza con dos punteros uno al inicio y otro al final del array. Se calcula el punto medio y se compara con el valor buscado. Si coincide, se termina; si es menor, se busca en la mitad izquierda; si es mayor, en la derecha. El proceso se repite hasta encontrar el valor o hasta que los punteros se solapen, lo que indica que el elemento no está presente.

Complejidad Temporal: O(log n). En cada paso, el espacio de búsqueda se reduce a la mitad. Para un array de n elementos, se necesitan como máximo lg(n) pasos para encontrar el elemento o determinar su ausencia.

Complejidad Espacial: O(1). Solo requiere memoria para los punteros de izquierda, derecha y medio.

**Galopante** (se debe tener como base que el array este ordenado): comienza en el índice 1 y duplica el salto en cada paso. Se detiene al encontrar una posición donde el valor supera al buscado, acotando el rango posible. Luego, se aplica una búsqueda binaria dentro de ese sub-array para localizar el elemento.

Complejidad Temporal:

Peor Caso: O(log n). Si el elemento está al final, la fase de galope recorre hasta n, y la búsqueda binaria opera sobre un rango grande, resultando en una complejidad similar a la búsqueda binaria estándar.

Caso Promedio/Mejor Caso: O(log i), donde i es la posición del elemento. Si el elemento está en una posición temprana i, la fase de galope toma log(i) pasos para encontrar el rango, y la búsqueda binaria subsecuente también opera en un rango de tamaño i, manteniendo la complejidad logarítmica respecto a la posición, no al tamaño total del array.

Complejidad Espacial: O(1). Solo requiere memoria para las variables de la fase de galope y los punteros de la búsqueda binaria.

## Implementaciones

Solamente se usaron los códigos de search del curso:
https://github.com/jfuentess/edaa.git

## Resultados experimentales

1. **Generación de Datasets**
    - Herramienta: Programa personalizado dataG.cpp
    - Tamaños de arrays: Potencias de 2 desde 2^10 (1,024) hasta 2^20 (1,048,576) elementos
    - Distribución de elementos: Arrays ordenados ascendentemente con valores aleatorios
    - Elementos de búsqueda: 10 elementos por dataset:
    - 7 elementos existentes: Distribuidos aleatoriamente en deciles para cubrir todo el rango
    - 3 elementos no existentes: Valores que garantizadamente no están en el array
    - Archivos generados: 11 datasets (data_size_1024.txt hasta data_size_1048576.txt)
2. **Metodología de Medición**
    - Biblioteca de tiempo: std::chrono::high_resolution_clock de C++
    - Iteraciones por medición:
    - Búsqueda Binaria: 100,000 iteraciones
    - Búsqueda Secuencial: 10,000 iteraciones
    - Búsqueda Galloping: 50,000 iteraciones
    - Justificación: Diferentes números de iteraciones para obtener mediciones precisas dado que los algoritmos tienen órdenes de magnitud diferentes en tiempo de ejecución
    - A pesar de las diferentes iteraciones igualmente se sacó el promedio, pero no la desviación estándar ya que no era viable en galloping y binaria. Por lo tanto, tampoco se sacó en secuencial para seguir el mismo formato.
    - Se decidió hacer esto para que el tiempo no fuese 0 todo el rato.

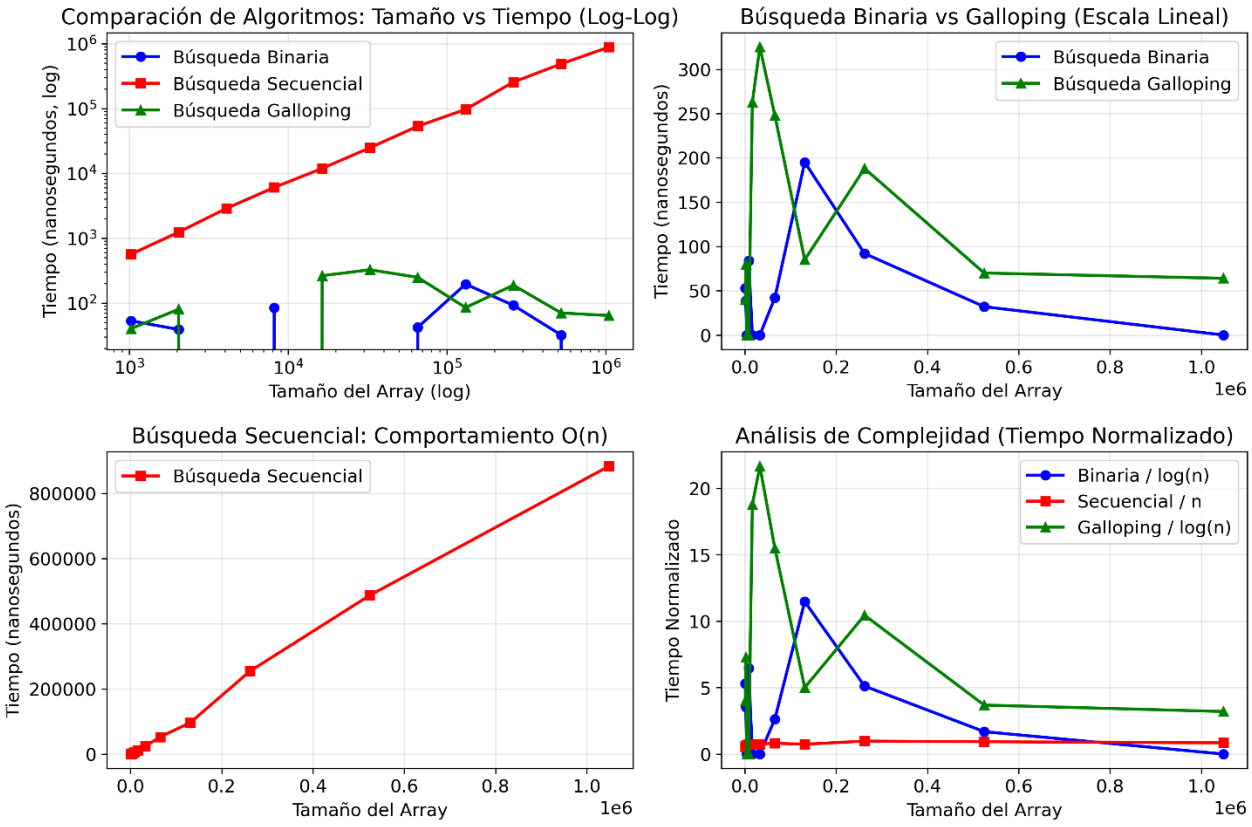
```cpp
const int iterations = 50000; // Moderate iterations for galloping search

auto start = std::chrono::high_resolution_clock::now();
for (int i = 0; i < iterations; i++) {
    result = galloping_search(data.data(), size, target_value);
}
auto end = std::chrono::high_resolution_clock::now();

// Calculate average time per operation in nanoseconds
auto total_duration = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);
long long average_time = total_duration.count() / iterations;
```

3. **Características de la Máquina de Experimentos**

- **Hardware**

    - Modelo: ASUS VivoBook X515UA_M515UA (Portátil)
    - Procesador: AMD Ryzen 7 5700U with Radeon Graphics
    - Velocidad base: 1797 MHz
    - 8 núcleos físicos, 16 procesadores lógicos
    - Arquitectura: x64 con Hyper-threading
    - Memoria RAM: 23,945 MB (~24 GB)
- Almacenamiento:
    - 2 SSD: 1024 GB + 512 GB
    - Sin discos HDD
    - Espacio disponible: ~1.1 TB total
- Software
    - Sistema Operativo: Windows 11 (64 bits)
    - Compilador: MinGW g++ con optimizaciones estándar
    - Sistema de archivos: NTFS

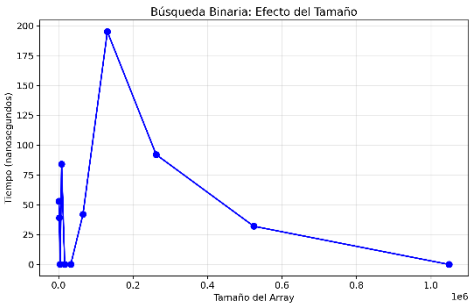
## Descripción de resultados

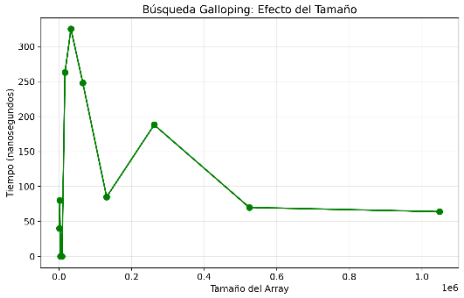


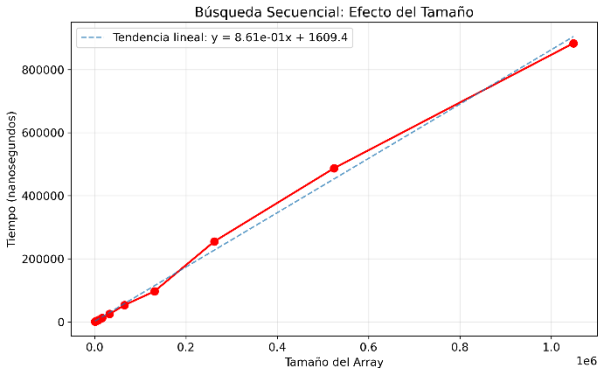

Se puede observar lo que ya se deduce desde el análisis teórico en relación a las diferencias de tiempo.

Sin embargo, aquí estamos en el experimento 1, el cual es de tiempo-size y manteniendo la posición fija, en este caso se eligió “posición fija en el medio - posición 3”, ya que, al crearse el dataset se subdividió en deciles y se marcó una posición al azar dentro de ese decil. En este caso estamos en la posición 3, la cual por el azar puede que haya estado en posiciones mas ventajosas a pesar del size, resultando en esa bajada de tiempo.

Gráficos individuales:

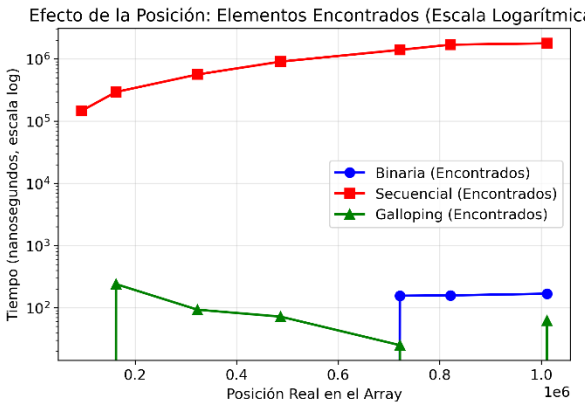


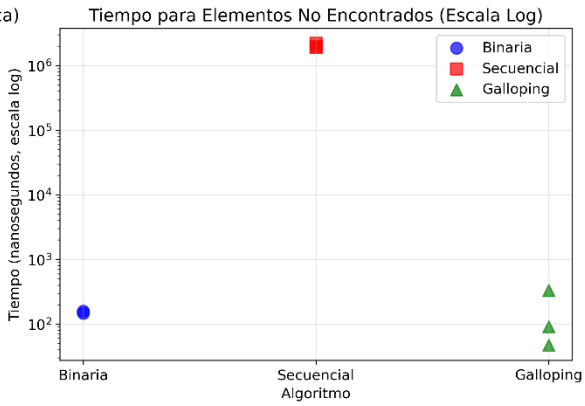



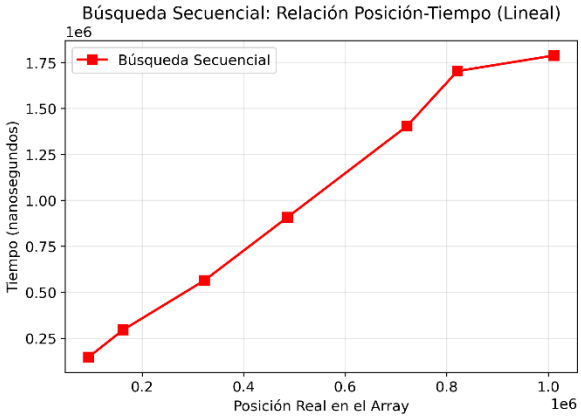




**Experimento 2:**

Solo se uso: data_size_1048576




Estadísticas Comparativas (Min-Max)

| Algoritmo | Tiempo Min (ns) | Tiempo Max (ns) |
|---|---|---|
| Binaria | 0 | 168 |
| Secuencial | 146380 | 2208093 |
| Galloping | 0 | 332 |

Secuencial

| PosicionR... | Tiempo |
|---|---|
| 94097 | 146380 |
| 162289 | 295198 |
| 322791 | 564826 |
| 486197 | 908101 |
| 721423 | 14034... |
| 820941 | 17032... |
| 1011229 | 17880... |
| -1 | 22080... |
| -1 | 19081... |
| -1 | 19580... |

Galopante

| PosicionR... | Tiempo |
|---|---|
| 94097 | 0 |
| 162289 | 240 |
| 322791 | 93 |
| 486197 | 72 |
| 721423 | 25 |
| 820941 | 0 |
| 1011229 | 62 |
| -1 | 47 |
| -1 | 332 |
| -1 | 91 |

Binaria

| PosicionR... | Tiempo |
|---|---|
| 94097 | 0 |
| 162289 | 0 |
| 322791 | 0 |
| 486197 | 0 |
| 721423 | 156 |
| 820941 | 158 |
| 1011229 | 168 |
| -1 | 0 |
| -1 | 158 |
| -1 | 147 |

La búsqueda galopante resulta más impredecible que la binaria, pero cuando el número buscado no existe, suele ofrecer resultados considerablemente mejores. En cambio, la búsqueda secuencial puede llegar a valores muy altos, como 1.788.077 o 2.208.093, mientras que las otras técnicas generan números mucho menores.

## Conclusiones

**Rendimiento relativo:**

Búsqueda binaria: 0-195 ns (prácticamente constante)

Búsqueda galloping: 0-325 ns (ligeramente variable)

Búsqueda secuencial: 563-883,312 ns (crecimiento lineal)

**Factores de mejora:**

Binaria es 3,000-15,000x más rápida que secuencial

Galloping es 1,000-15,000x más rápida que secuencial

Galloping es solo 1.2-2x más lenta que binaria

**Decisiones por Escenario**

Arrays grandes (>10K elementos): Usar búsqueda binaria, El límite de la función logarítmica es notable

Arrays pequeños (<1K elementos): Búsqueda secuencial aceptable, Simplicidad vs. Rendimiento

Galloping: Los resultados experimentales no demuestran ventajas de rendimiento sobre búsqueda binaria y presenta mayor variabilidad temporal (0-325 ns sin patrón predecible), lo que la hace menos confiable. Su uso se justifica únicamente en escenarios específicos donde se dispone de información adicional sobre la ubicación probable del elemento buscado, limitando su aplicabilidad práctica frente a la consistencia y predictibilidad de búsqueda binaria.

## Referencias

Implementaciones: https://github.com/jfuentess/edaa.git

Referencia: https://www.geeksforgeeks.org/dsa/exponential-search/