Boletín 1

Jaime Ansorena Carrasco
2020401497

Profesor: José Fuentes Sepúlveda
Ayudante: Leonardo Enrique Lovera Emanuelli

7 de octubre de 2024

1. Resumen

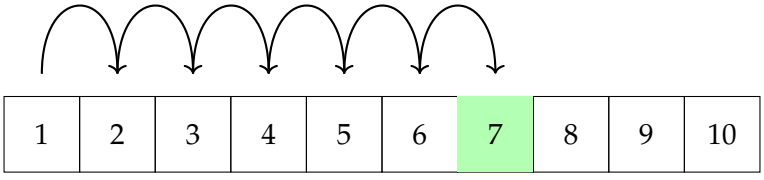
¿Cómo localizar datos almacenados con una identificación determinada?. El problema de búsqueda puede definirse en esta simple pregunta. En el presente boletín se analizaron tres algoritmos de búsqueda en secuencias: secuencial, binaria y galopante. El objetivo principal fue comparar su rendimiento teórico como experimental, considerando diferentes configuraciones en los tamaños de entrada y la posición del objetivo.

Los resultados experimentales muestran que la búsqueda secuencial es mucho más lenta que la binaria y la galopante, especialmente en grandes volúmenes de datos, donde puede ser hasta 1000 veces más lenta que la binaria. La búsqueda galopante es eficiente cuando el elemento se encuentra al inicio de los datos o en estructuras de tamaño desconocido, mientras que la búsqueda binaria es preferible para tiempos de respuesta consistentes en datos ordenados de gran tamaño.

2. Algoritmos

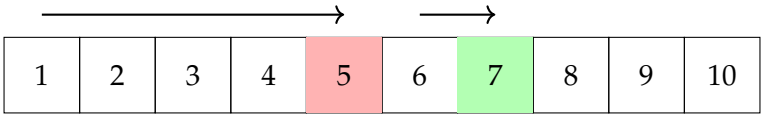
Búsqueda secuencial

"BEGIN AT THE BEGINNING, and go on till you find the right key; then stop." El procedimiento secuencial consiste en la manera más obvia de buscar: revisar cada elemento de la estructura de datos en secuencia hasta encontrar el elemento o llegar al final de la estructura de datos. Este algoritmo es el único que funciona tanto en estructuras que se encuentran ordenadas como no ordenadas, ya que no depende del orden de los elementos.



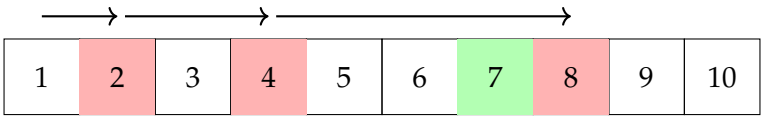
Búsqueda binaria

"Divide and conquer: start in the middle, and adjust your range depending on whether the key is larger or smaller." El algoritmo de búsqueda binaria se utiliza en listas ordenadas. Se selecciona repetidamente el elemento del medio de la lista para compararlo con el valor objetivo y reducir el rango de búsqueda a la mitad. Este proceso continúa hasta que se encuentra el elemento o se reduce el rango a cero.



Búsqueda galopante

"Start small and exponentially expand your search range, then refine it with binary search." La búsqueda galopante primero realiza saltos exponenciales hasta que encuentra un rango que contiene el valor buscado, y luego realiza una búsqueda binaria en ese rango. Este algoritmo es útil en listas ordenadas de tamaño desconocido o muy grandes.



3. Análisis Teórico

Búsqueda secuencial

El peor caso ocurre cuando el elemento a buscar no se encuentre en la estructura, o se encuentra al final de esta misma. Complejidad temporal $O(n)$ resulta de la necesidad del algoritmo de recorrer la estructura completa y realizar n comparaciones.
La complejidad espacial de este algoritmo de búsqueda es $O(1)$, es decir, usa una cantidad constante de espacio adicional independiente del tamaño de la entrada.

Búsqueda binaria

Para que la búsqueda binaria funcione correctamente, la estructura de datos debe estar previamente ordenada y permitir el acceso aleatorio a los elementos.
La complejidad temporal es $O(\log n)$ en el caso promedio y peor caso, ya que en cada iteración el espacio de búsqueda se reduce a la mitad. Luego de $\log n$ comparaciones, se encuentra el elemento buscado o se determina que no esta presente en la estructura.
Respecto a la complejidad espacial, utiliza $O(1)$ espacio adicional.

Búsqueda galopante

Este algoritmo introduce el concepto de análisis adaptativo, donde la complejidad depende de la posición del elemento buscado en la estructura de datos.

La complejidad temporal de la búsqueda galopante es $O(\log p)$, donde p es la posición del elemento buscado. El algoritmo realiza aproximadamente $\log p$ saltos exponenciales para sobrepasar el rango en el que se encuentra el elemento. Una vez identificado el rango, se realiza una búsqueda binaria dentro de dicho intervalo.

$$\lfloor \log p \rfloor + 1 + O(\log p) = O(\log p) < O(\log n)$$

En términos de complejidad espacial, la búsqueda galopante utiliza $O(1)$ espacio adicional, ya que solo requiere variables para los índices de búsqueda.

Resumen

Algoritmo	Complejidad Temporal	Complejidad Espacial
Búsqueda Secuencial	$O(n)$	$O(1)$
Búsqueda Binaria	$O(\log n)$	$O(1)$
Búsqueda Galopante	$O(\log p)$	$O(1)$

Implementaciones

Las implementaciones de los distintos algoritmos de búsqueda se obtuvieron del repositorio del curso Estructuras de Datos y Algoritmos Avanzados.

4. Resultados experimentales

Inicialización

Para realizar los experimentos, se utilizaron tres algoritmos de búsqueda: búsqueda secuencial, búsqueda binaria y búsqueda galopante. Se creó un `std::vector` poblado con enteros consecutivos de 32 bits desde 0 hasta $n - 1$, donde n varía según el experimento. Estos datos fueron generados internamente y no se utilizaron datasets externos.

Se realizaron pruebas sobre cada algoritmo ejecutándolo en 4000 repeticiones para cada tamaño de entrada. Se midió tanto el tiempo promedio por operación como la desviación estándar, con el fin de obtener resultados estadísticamente significativos y mitigar el efecto de fluctuaciones de rendimiento aleatorias.

Primer experimento

El primero `<exp1>` considera diferentes configuraciones en los tamaños de entrada:

```
for (int i = 0; i < n; i++)
        data[i] = i;
```

donde n es el largo del vector y varía según parámetros que serán definidos y discutidos posteriormente. El objetivo a buscar se define como `int target = n - 1` para todos los algoritmos.

Segundo experimento

El segundo experimento varía la posición del objetivo a buscar. Se define una variable `int64_t fixed_n = 20000` que representa un vector de largo fijo, y se define `int target = n - 1` como la posición a buscar.

Especificaciones

Los experimentos se realizaron en un equipo de esritorio con CPU i9-9900k que posee 8 núcleos a 3.6 GHz y 32 GB DDR4 a 3600 MHz. Dado que los algoritmos no involucran memoria secundaria, todas las operaciones se realizaron en memoria principal (RAM). El sistema operativo es Linux Mint 22 x86_64 con el kernel de linux 6.8.0-45

Compilación

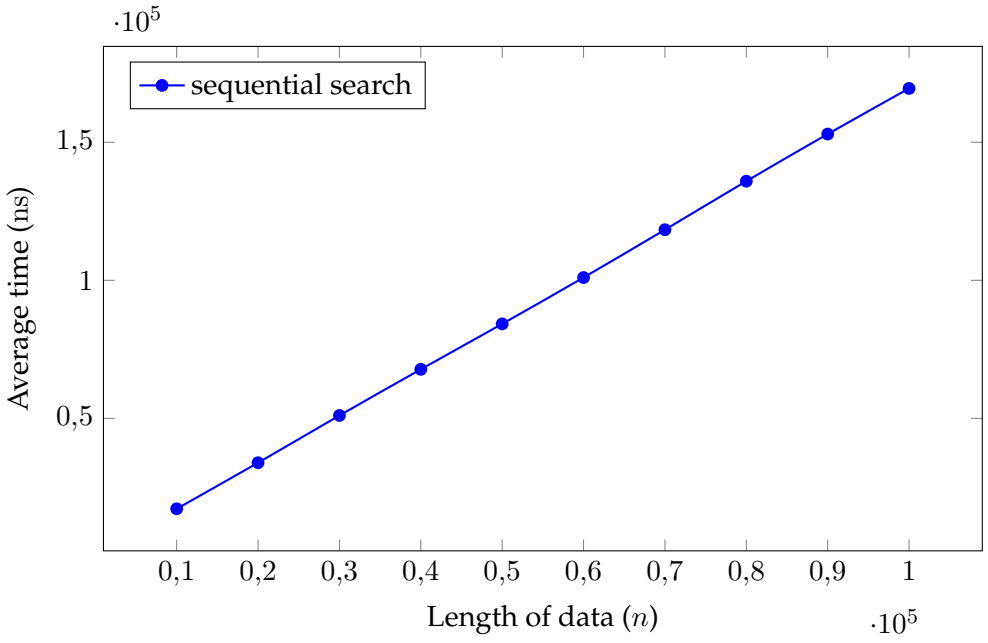
Para automatizar el análisis experimental se utilizó la herramienta `uhr`, disponible en el repositorio del curso. Esta se compiló usando el compilador GNU G++ como:

```
g++ -O0 -std=c++11 -Wall -Wpedantic -o <algorithm>_<exp> uhr_<exp>.cpp
```

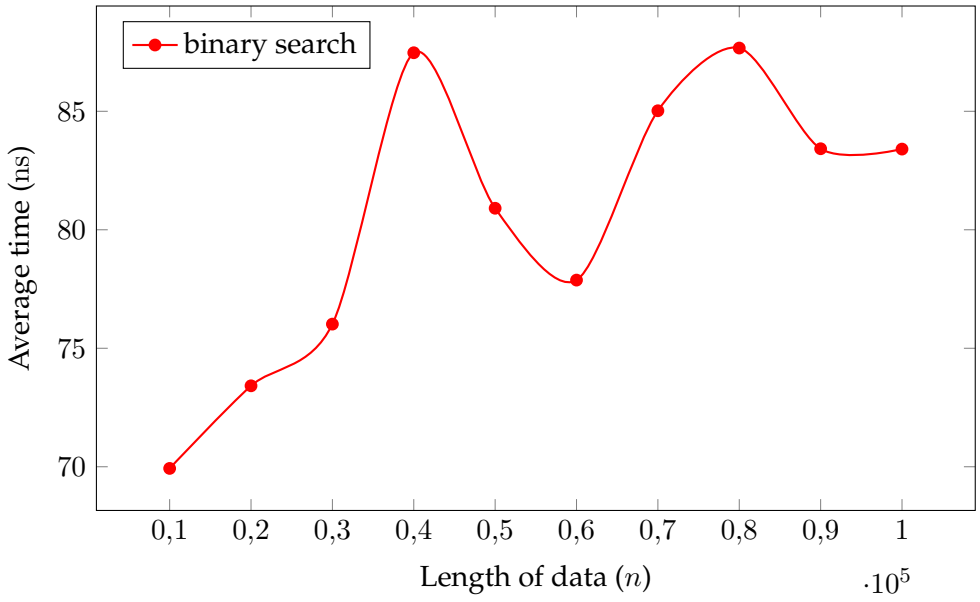
donde `<algorithm>` corresponde al algoritmo utilizado y `<exp>` al experimento. Cada experimento generó 3 ejecutables (uno por cada algoritmo) que luego fueron automatizados mediante un script de bash para probar distintos valores de LOWER, UPPER y STEP. El detalle se encuentra en el archivo `run_experiments.sh`.

Resultados experimento 1

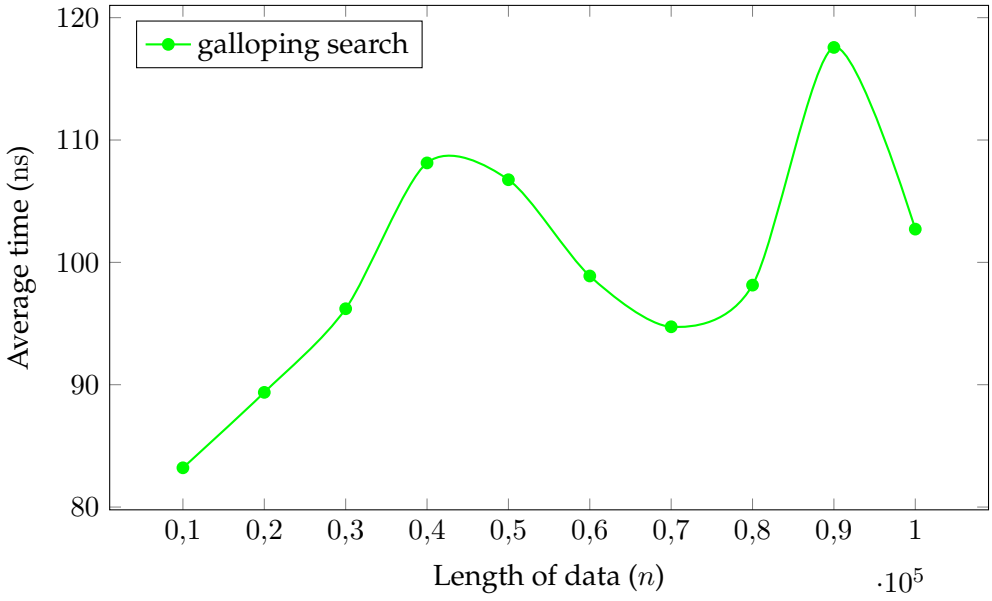
Para el primer experimento se utilizaron los valores de `LOWER=10000, UPPER=100000 y STEP=10000`



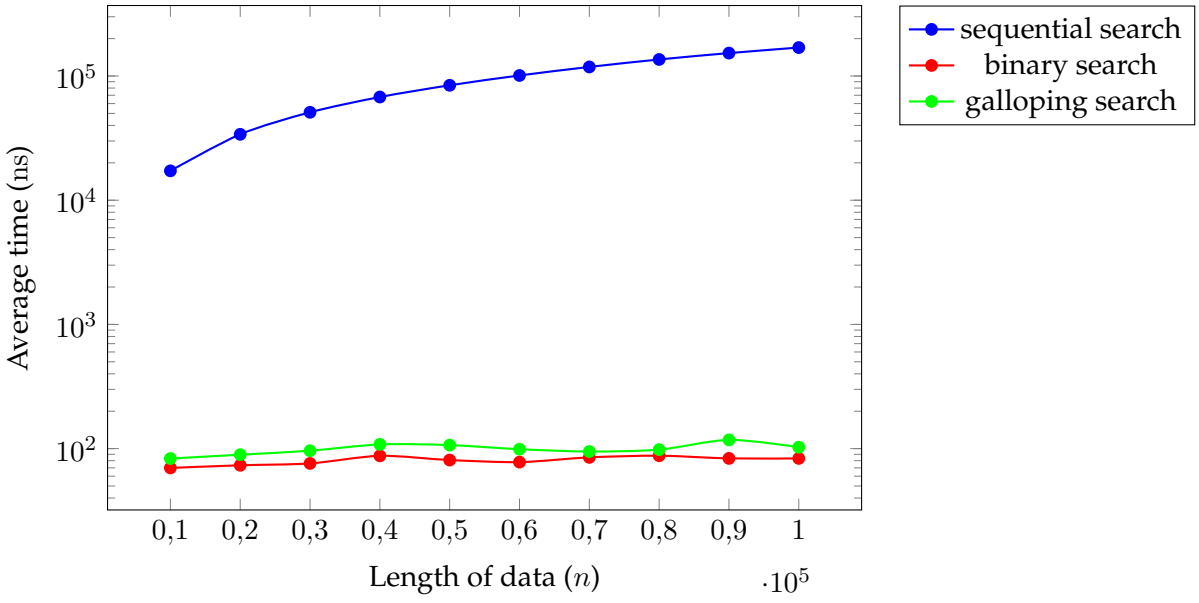
Se observa el resultado esperado, es decir, una relación lineal, donde el tiempo de ejecución aumenta proporcionalmente al tamaño del conjunto de datos. Esto refleja la complejidad temporal del algoritmo $O(n)$.



Para la búsqueda binaria se observan tiempos promedios mas bajos (en ordenes de magnitud) que la búsqueda secuencial, dada la naturaleza logarítmica de este algoritmo. Las fluctuaciones en el tiempo de ejecución podrían estar asociadas a gestión de caché o la distribución de los datos en la memoria. La forma interpolada de la curva denota la complejidad teórica $O(\log n)$.



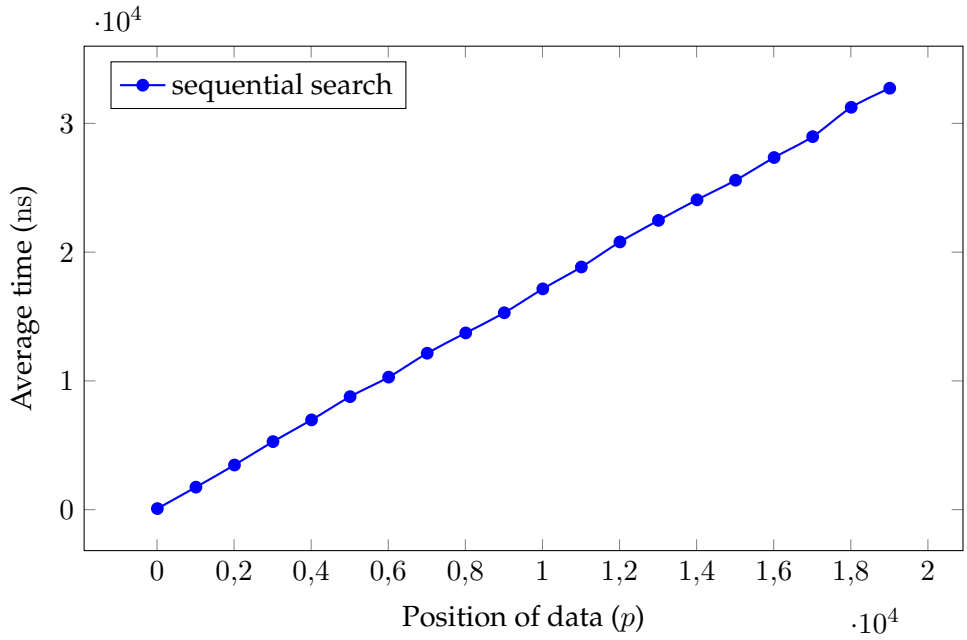
La búsqueda galopante sigue un compartimiento similar a la búsqueda a la binaria, donde los tiempos promedios aumentan logarítmicamente a medida que crece el tamaño del vector. Hay que considerar que en este caso el elemento a buscar se encuentra al final del arreglo, lo cual considera un análisis de peor de caso.



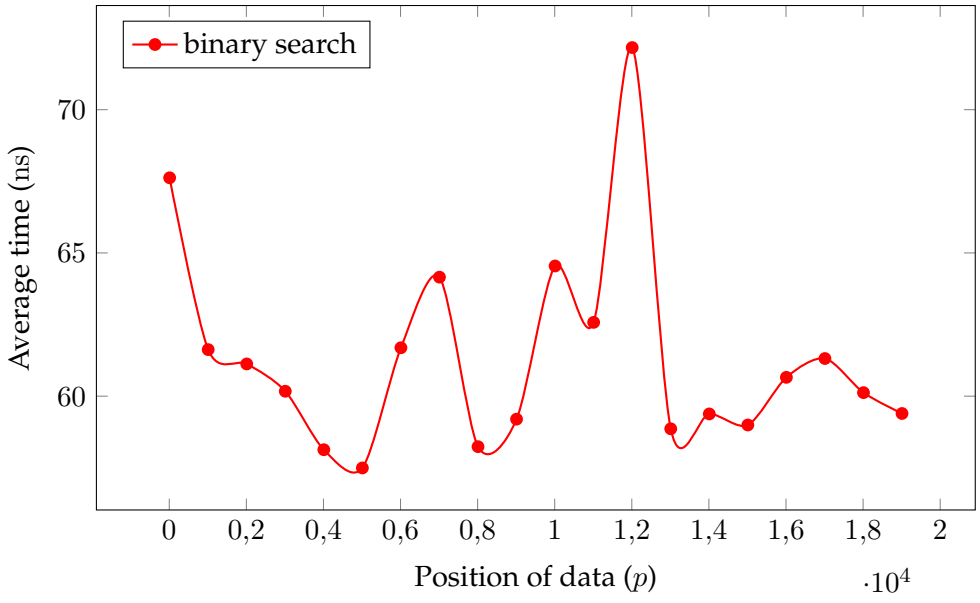
Los tres algoritmos pueden compararse usando una escala logarítmica en el eje y. La búsqueda secuencial crece linealmente al tamaño de los datos, mientras que la búsqueda binaria y galopante presentan un comportamiento similar.

5. Resultados experimento 2

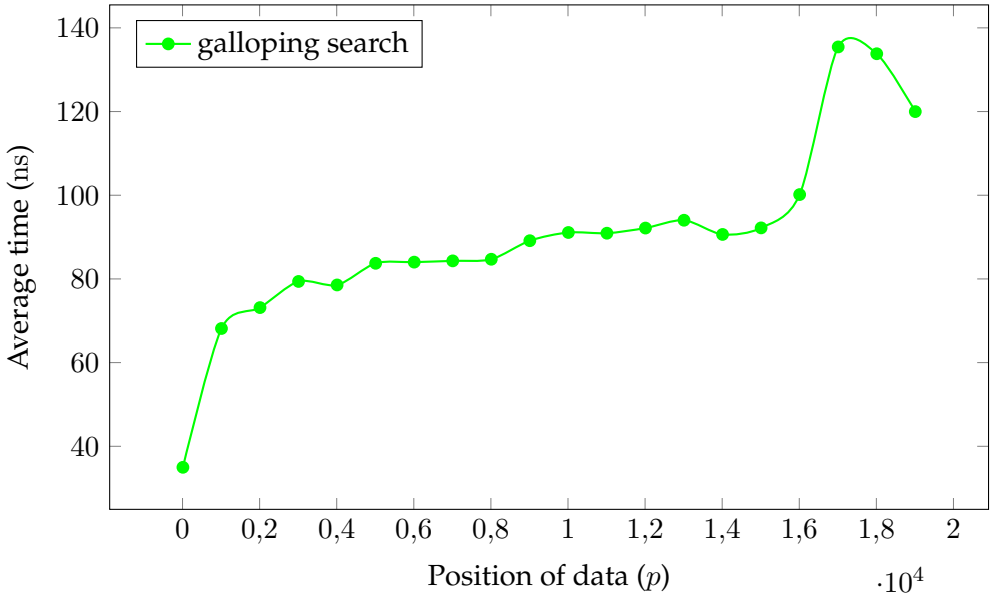
Para el segundo experimento se utilizaron los valores de `LOWER=1000`, `UPPER=20000` y `STEP=1000` y un tamaño fijo del vector `int64_t fixed_n = 20000`.



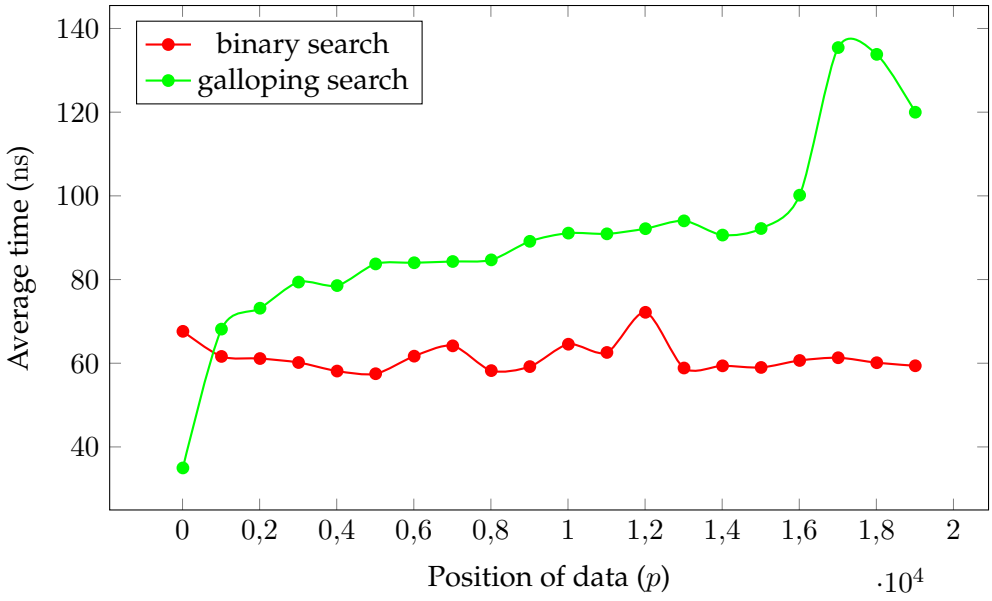
El tiempo promedio de búsqueda aumenta de manera lineal conforme se incrementa la posición del dato. Esto se debe a que la búsqueda secuencial recorre cada elemento desde el principio hasta encontrar el objetivo, lo que hace que el tiempo sea proporcional a la posición del dato.



Aunque el tiempo de búsqueda se mantiene relativamente bajo y estable, se observan fluctuaciones en el tiempo promedio. Esto está asociado con la naturaleza de la búsqueda binaria, que divide el conjunto de datos repetidamente. Este proceso permite que la búsqueda binaria tenga un mejor rendimiento en ciertos rangos, especialmente cuando el dato buscado se encuentra cerca de las divisiones de datos óptimas, donde el algoritmo puede reducir rápidamente el espacio de búsqueda. Sin embargo, en las otras divisiones la búsqueda binaria es menos eficiente, resultando en un aumento en el tiempo de búsqueda, como se observa en los picos en el gráfico.



Esta búsqueda comienza con tiempos promedios bajos, debido a la naturaleza exponencial que presenta, pero empeora a medida que va aumentando la posicion del elemento a buscar. Este comportamiento sugiere que, si bien la búsqueda galopante es eficiente al principio, su rendimiento puede deteriorarse para elementos que se encuentran en posiciones más lejanas, posiblemente debido a un mayor número de pasos de expansión necesarios en esos casos.



La búsqueda binaria mantiene una ventaja en términos de estabilidad, mientras que la búsqueda galloping es menos eficiente para posiciones de elementos más altas debido a la necesidad de expandir el rango de búsqueda varias veces, lo que resulta en más operaciones.

6. Conclusiones

La búsqueda secuencial es significativamente más lenta que las otras dos. Para tamaños grandes, la búsqueda secuencial presenta tiempos promedio de ejecución de hasta 3 ordenes de magnitud mayores que la búsqueda binaria y galopante. En el segundo experimento, la búsqueda secuencial tarda en promedio hasta 1000 veces más que la búsqueda binaria para los mismos tamaños de entrada, demostrando la ineficiencia cuando se trata de grandes volúmenes de datos. La búsqueda secuencial es la única opción disponible cuando se tiene una estructura que no esta ordenada, siendo el único caso donde presenta mayor utilidad.

La búsqueda binaria y galopante no presentaron mayores diferencias a medida que se varia el tamaño de la estructura, pero si presentan diferencias cuando se realiza una análisis adaptativo.

La búsqueda binaria es más rápida y consistente en promedio que la búsqueda galopante en rangos de datos medianos y altos. Sin embargo, la búsqueda galopante presenta mejor rendimiento cuando el elemento buscado está cerca del inicio del conjunto de datos. Si el elemento buscado se encuentra con alta frecuencia en posiciones iniciales del conjunto de datos, es preferible usar la búsqueda galopante, debido a su expansión exponencial. De igual manera, la búsqueda galopante presenta utilidad cuando no se conoce el tamaño de la estructura. Para aplicaciones que requieren tiempos de respuesta consistentes en grandes tamaños, es preferible la búsqueda binaria.

7. Referencias

- Knuth, Donald (1998). Sorting and Searching. The Art of Computer Programming. Vol. 3 (2nd ed.). Addison-Wesley Professional.
- Apuntes Clase 1, Problema inicial, Estructuras de Datos y Algoritmos Avanzados, Profesor José Fuentes, Universidad de Concepción, 2024.

8. Anexo

Secuencial exp1

n	t_mean	t_stdev
10000	17240.2	3420.24
20000	33928.1	2536.6
30000	51041.8	3734.12
40000	67757.6	3852.51
50000	84214.7	1968.98
60000	101012	2025.18
70000	118328	4704.19
80000	135904	7133.84
90000	152970	6327.27
100000	169496	6893.54

Binaria exp1

n	t_mean	t_stdev
10000	69.93	5.71999
20000	73.4158	5.71897
30000	76.0195	214.534
40000	87.4707	494.397
50000	80.9082	240.844
60000	77.8765	67.5735
70000	85.0228	326.705
80000	87.6688	352.795
90000	83.4197	234.699
100000	83.405	233.265

Galopante exp1

n	t_mean	t_stdev
10000	83.2087	7.05681
20000	89.3738	49.0406
30000	96.2097	41.6356
40000	108.128	947.964
50000	106.755	413.216
60000	98.8868	6.65199
70000	94.7352	5.55147
80000	98.1393	5.90081
90000	117.568	1023.77
100000	102.717	7.37949

Secuencial exp2

n	t_mean	t_stdev
10	85.4772	300.2
1010	1749.71	455.653
2010	3468.76	722.188
3010	5286.54	721
4010	6972.85	510.579
5010	8777.92	1886.45
6010	10294.1	1500.82
7010	12150.4	2517.44
8010	13727.8	1959.36
9010	15291.8	1232.3
10010	17148.4	2119.18
11010	18848.1	1980.47
12010	20794	3458.95
13010	22466.9	717.933
14010	24063.5	2965.74
15010	25585	1917.3
16010	27349	2490.29
17010	28969.4	2132.31
18010	31247.9	4484.37
19010	32730.9	448.736

Binaria exp2

n	t_mean	t_stdev
10	67.6225	384.213
1010	61.6247	3.92566
2010	61.1245	4.83486
3010	60.1713	35.7661
4010	58.1305	5.37287
5010	57.4943	4.67209
6010	61.6902	5.18749
7010	64.1543	227.764
8010	58.2368	5.58588
9010	59.197	7.55625
10010	64.544	253.694
11010	62.5763	5.53289
12010	72.1688	478.731
13010	58.8608	6.88289
14010	59.3795	5.86605
15010	58.9937	46.6236
16010	60.6568	2.95366
17010	61.3137	2.67481
18010	60.123	32.8825
19010	59.3952	3.93489

Galopante exp2

n	t_mean	t_stdev
10	34.9785	3.1983
1010	68.1545	7.43606
2010	73.1497	6.6848
3010	79.3965	7.3248
4010	78.5588	30.6184
5010	83.7352	7.22674
6010	84.0135	45.1262
7010	84.3135	7.7444
8010	84.7013	33.8254
9010	89.135	35.2369
10010	91.097	37.0707
11010	90.9255	27.7887
12010	92.156	9.02837
13010	94.025	9.44217
14010	90.6292	8.70943
15010	92.1963	10.9417
16010	100.161	29.8038
17010	135.441	26.681
18010	133.815	22.718
19010	119.987	88.9926