Investigación Algoritmo y Estructuras de Datos

Mario Grillo #23020227
Ruben Montenegro
Fernando Reyes

1. Introducción

Planteamiento del problema

> En el desarrollo de software y soluciones computacionales, la eficiencia de los algoritmos juega un papel crucial. Por ello, es fundamental comparar diferentes enfoques para resolver problemas comunes como la búsqueda de elementos en listas y el ordenamiento de datos.
>
> La búsqueda binaria es un algoritmo eficiente para localizar un elemento en una lista ordenada. Divide repetidamente el rango de búsqueda a la mitad hasta encontrar el objetivo o determinar su ausencia

Objetivos

- Analizar y comparar el funcionamiento de la búsqueda binaria y el algoritmo Bubble Sort.
- Implementar ambos algoritmos en un lenguaje de programación (Python o C).
- Evaluar su eficiencia mediante análisis teóricos y experimentales.

---

2. Metodología

- Se realizó una revisión bibliográfica sobre los algoritmos seleccionados.
- Se implementaron ambos algoritmos en Python.
- Se realizaron pruebas con distintos tamaños de datos y se midió el tiempo de ejecución.
- Se elaboraron gráficos comparativos para ilustrar los resultados.

3. Análisis Teórico

• Complejidad temporal:

- Mejor caso: O(1) — El elemento se encuentra en el centro en la primera comparación.
- Caso promedio: O(log n).
- Peor caso: O(log n) — El elemento no está presente o se encuentra en un extremo.

• Complejidad espacial: O(1) — Uso constante de memoria adicional.

• Requisito: La lista debe estar ordenada previamente.

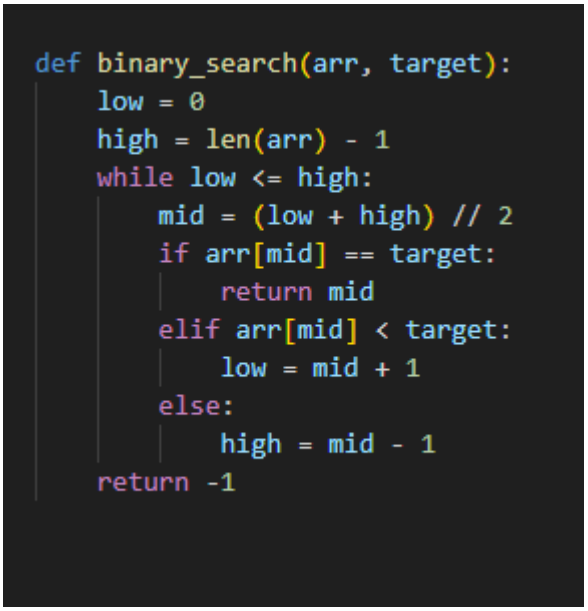
• Tipo de algoritmo: No es estable ni adaptable en cuanto a ordenamiento, ya que es un algoritmo de búsqueda, no de ordenamiento.

---

4. Implementación del algoritmo

Búsqueda Binaria

- Requiere una lista previamente ordenada.
- Divide el arreglo a la mitad en cada iteración hasta encontrar el valor o determinar que no existe.



```python
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

Bubble Sort

- Recorre la lista múltiples veces.
- Intercambia elementos adyacentes si están en el orden incorrecto.

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

4. Medición Empírica

Para validar el análisis teórico, se implementó la búsqueda binaria en Python. Se midió el tiempo promedio por búsqueda en listas de diferentes tamaños, realizando 20000 búsquedas aleatorias por experimento y promediando sobre 5 ejecuciones.

Tamaño de entrada (n)	Tiempo prom. por búsqueda (s)
1.000	2.03397438e-06
10.000	3.16738209e-06
50.000	3.60419804e-06
100.000	3.98348666e-06
200.000	4.32382857e-06

Análisis de eficiencia

Búsqueda Binaria

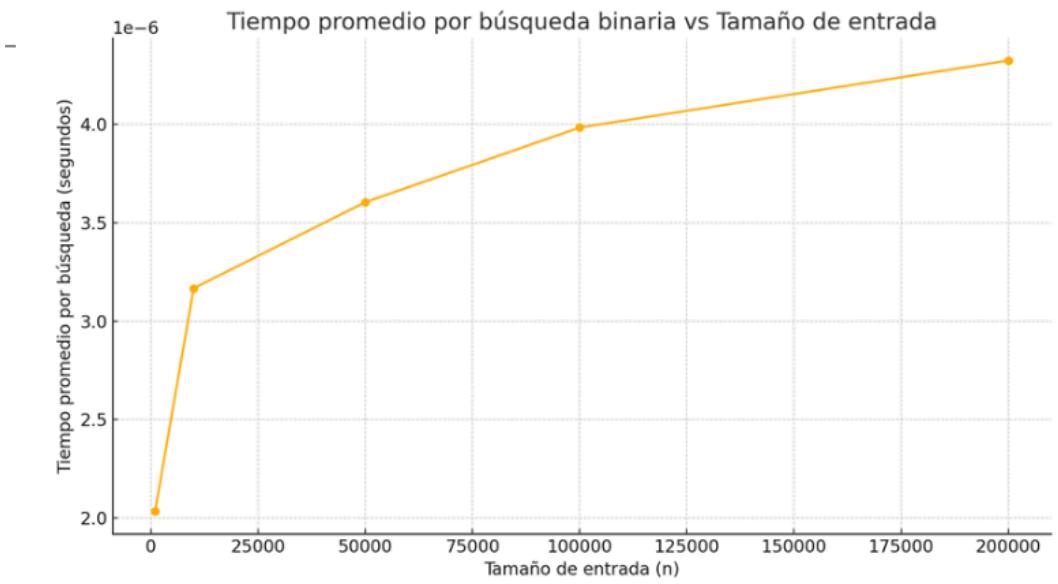
- A priori:
    - Mejor caso: O(1)
    - Peor caso: O(log n)
- A posteriori:
    - Tiempo medido con listas de 1000, 10,000 y 100,000 elementos ordenados.

Bubble Sort

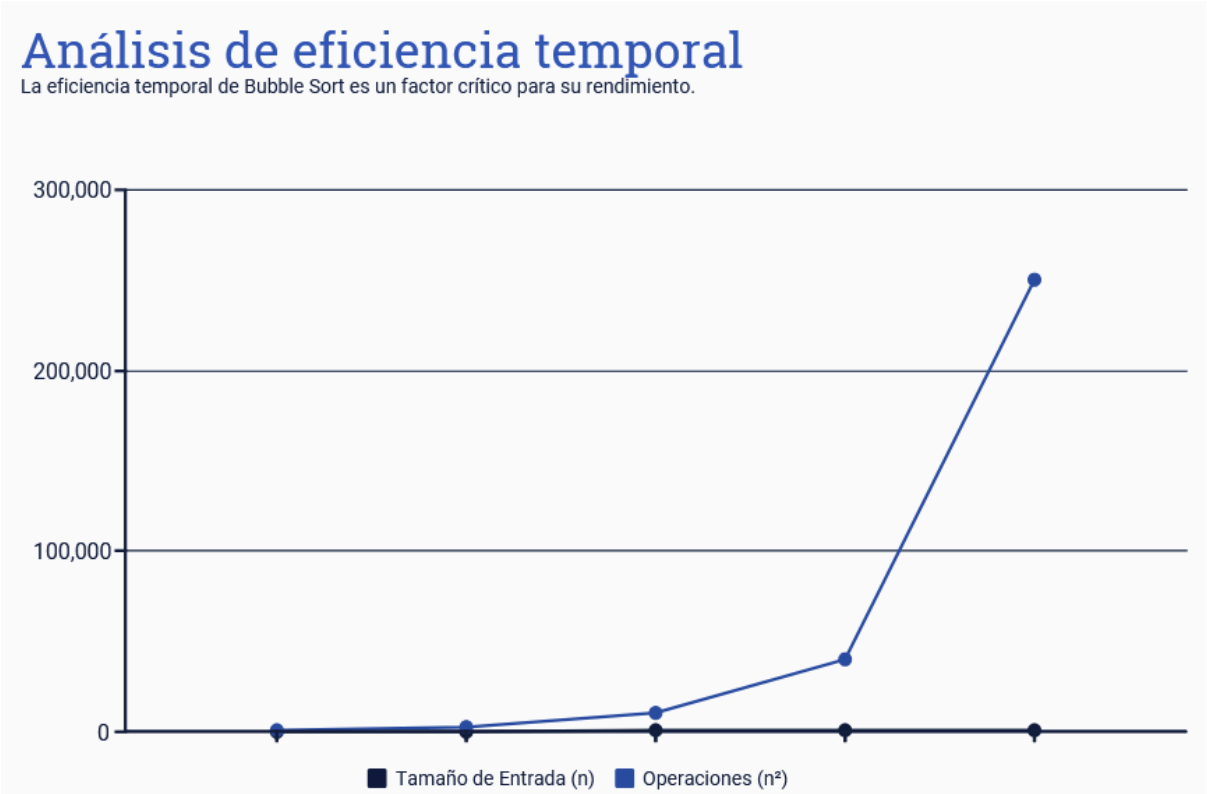
- A priori:
    - Mejor caso: O(n) *(si ya está ordenado)*
    - Peor caso: O(n²)
- A posteriori:
    - Tiempo medido con listas pequeñas y medianas (por su ineficiencia en grandes volúmenes).

Ejecución del código

La gráfica evidencia el crecimiento casi logarítmico del tiempo de búsqueda a medida que aumenta el tamaño de la lista, confirmando la complejidad O(log n).



En el caso de bubble sort:



Bubble Sort es muy eficiente en el uso de memoria. No requiere espacio adicional proporcional al tamaño de la entrada. Solo utiliza una cantidad constante de memoria auxiliar para los intercambios. Aunque el rendimiento cae drásticamente con listas grandes.

5. Resultados obtenidos

- **Búsqueda Binaria** demostró alta eficiencia en listas grandes ordenadas.
    - Tiempo de búsqueda prácticamente constante respecto al tamaño de la lista.
- **Bubble Sort** mostró ineficiencia con listas mayores a 500 elementos.
    - El tiempo de ejecución creció exponencialmente.
- Los datos experimentales respaldan los análisis teóricos.

---

6. Conclusiones

- La **búsqueda binaria** es extremadamente eficiente, pero depende de que la lista esté ordenada.

- El **Bubble Sort**, aunque fácil de implementar, no es recomendable para listas grandes.

- Esta investigación resalta la importancia de seleccionar el algoritmo adecuado según el contexto del problema.

- A nivel educativo, ambos algoritmos son útiles para comprender principios de complejidad algorítmica y análisis computacional.

La búsqueda binaria demuestra ser altamente eficiente para colecciones grandes, con un crecimiento logarítmico de tiempo. Su principal limitación es la necesidad de que los datos estén previamente ordenados. Para entornos donde las búsquedas son frecuentes y actualizar el ordenamiento es costoso, suele combinarse con algoritmos de mantenimiento de estructuras ordenadas (por ejemplo, árboles balanceados).