UNIVERSIDAD AMERICANA
Facultad de Ingeniería y Arquitectura FIA
Carrera Ingeniería en sistemas de información

Algoritmo y Estructura de Datos

"Analisis de eficiencia de los algoritmos Bubble sort y Binary search"

Integrantes

- Jose Gabriel Cano Blandón
- Axell Antonio Castillo Tapia
- Freddy Adrian Peralta Palacio
- Marcos Alessandro Lagos Rivera

Docente

- Silvia Gigdalia Ticay López

Managua, Nicaragua
7 de julio del 2025

I. Introducción

Este trabajo de investigación tiene como objetivo examinar el comportamiento de dos algoritmos esenciales en el ámbito de las estructuras de datos: el método de ordenamiento Burbuja (Bubble Sort) y el algoritmo de búsqueda Binaria (Binary Search). Ambos algoritmos destacan por su sencillez y utilidad en entornos educativos, así como por su aplicación en tareas básicas del desarrollo de software. La investigación se enfoca en comprender su lógica de funcionamiento, evaluar su eficiencia y realizar una comparación bajo distintos tipos de entrada. Elegir e implementar adecuadamente estos algoritmos puede tener un impacto significativo en el rendimiento de los sistemas, especialmente al manejar grandes volúmenes de datos.

1. Planteamiento del problema

La eficiencia de los algoritmos de ordenamiento y búsqueda representa un factor clave en el rendimiento de los sistemas dentro del desarrollo de software, sobre todo cuando se manejan grandes cantidades de datos. Entre los algoritmos más utilizados se encuentra el método de Burbuja (Bubble Sort), ampliamente conocido por su simplicidad y facilidad de implementación. Sin embargo, su desempeño en términos de tiempo de ejecución es deficiente cuando se enfrenta a listas extensas o desordenadas, lo que limita su aplicación práctica en contextos reales. A pesar de estas limitaciones, su uso persiste en entornos académicos y en algunos desarrollos, muchas veces sin un análisis previo de su rendimiento frente a otras alternativas más eficientes.

Por otro lado, el algoritmo de Búsqueda Binaria (Binary Search) ofrece una alta eficiencia en la localización de elementos, pero depende estrictamente de que los datos estén previamente ordenados, lo que implica un costo adicional si esta condición no se cumple. Esta dependencia introduce una relación entre los algoritmos de ordenamiento y búsqueda que debe ser considerada cuidadosamente. En este contexto, surge la necesidad de analizar y comparar ambos algoritmos desde una perspectiva práctica y cuantitativa, evaluando aspectos como el tiempo de ejecución, el uso de memoria y la cantidad de operaciones realizadas. Esta investigación busca aportar evidencia empírica que permita entender sus ventajas y limitaciones, y así fomentar una selección más informada y eficiente en la implementación de soluciones algorítmicas.

2. Objetivos

Objetivo general

Analizar la eficiencia computacional del algoritmo de ordenamiento Burbuja y el algoritmo de búsqueda Binaria mediante su implementación, evaluación y comparación frente a diferentes volúmenes de datos.

3. Objetivos específicos

1 Implementar y comprender el funcionamiento de los algoritmos de ordenamiento Burbuja y búsqueda Binaria, destacando sus fundamentos lógicos, condiciones de uso y requisitos estructurales.

2 Evaluar empíricamente la eficiencia de ambos algoritmos, mediante la medición del tiempo de ejecución y uso de memoria en distintos escenarios (mejor, peor y caso promedio).

3 Comparar el rendimiento de los algoritmos seleccionados a partir de un análisis combinado: teórico (a priori) y experimental (a posteriori), para identificar sus fortalezas y limitaciones según el tipo de entrada.

4 Reflexionar sobre la aplicabilidad práctica de cada algoritmo en el desarrollo de software, considerando criterios como escalabilidad, simplicidad, y eficiencia en contextos reales con grandes volúmenes de datos.

II. Metodología

1.Diseño de la investigación

El diseño será experimental, ya que se implementarán los algoritmos seleccionados (Burbuja y Búsqueda Binaria) y se manipularon distintas entradas de datos para observar cómo varía su comportamiento bajo condiciones controladas. Esto permitirá medir de forma precisa variables como el tiempo de ejecución y el uso de memoria.

2. Enfoque de la investigación

El enfoque será cuantitativo, ya que se obtendrán y analizarán datos numéricos, como el tiempo que tarda cada algoritmo en ejecutarse, la cantidad de pasos que realizan y la memoria que consumen. Estos datos permitirán comparar objetivamente la eficiencia de cada algoritmo.

3. Alcance de la investigación

El alcance es descriptivo y explicativo, porque no solo se describirán las características de cada algoritmo, sino que también se explicarán las razones de su comportamiento frente a diferentes escenarios de entrada, tales como listas ordenadas, desordenadas o parcialmente ordenadas.

4. Procedimiento

1 Selección de los algoritmos de estudio: Burbuja (ordenamiento) y Búsqueda Binaria (búsqueda).

2 Implementación de los algoritmos utilizando un lenguaje de programación (Python).

3 Generación de conjuntos de datos con distintos tamaños y condiciones (ordenados, aleatorios, inversos).

4 Ejecución de los algoritmos con cada conjunto de datos.

5 Medición del rendimiento: tiempo de ejecución, cantidad de comparaciones e intercambios, uso de memoria.

6 Análisis comparativo de los resultados obtenidos, relacionándolos con la teoría (análisis a priori y a posteriori).

III. Marco Conceptual / Referencial

Abordamos dos algoritmos representativos en el manejo de estructuras de datos lineales: Bubble Sort, para el ordenamiento, y Binary Search, para la búsqueda. Comprender sus principios teóricos, su complejidad computacional y su comportamiento práctico resulta esencial para evaluar su desempeño y determinar su aplicabilidad en tareas concretas del desarrollo de software.

Algoritmo

Un algoritmo es un conjunto finito y ordenado de pasos o instrucciones diseñadas para resolver un problema específico. En programación, los algoritmos permiten transformar datos de entrada en una solución computacional. En este estudio, los algoritmos de Bubble Sort y Binary Search representan procedimientos sistemáticos aplicados sobre estructuras de datos lineales para ordenar y buscar información, respectivamente.

Ordenamiento

El ordenamiento es el proceso mediante el cual se reacomodan los elementos de una estructura de datos siguiendo un criterio determinado, comúnmente numérico o alfabético. El algoritmo de Bubble Sort realiza este proceso comparando pares de elementos adyacentes y realizando intercambios sucesivos. Aunque es simple, su desempeño disminuye considerablemente cuando el tamaño de la lista crece.

Búsqueda

La búsqueda consiste en localizar un elemento específico dentro de una estructura de datos. La búsqueda binaria es un algoritmo eficiente que divide el conjunto ordenado en mitades sucesivas hasta encontrar el valor deseado o determinar su ausencia. Este método requiere que los datos estén previamente ordenados, estableciendo una dependencia directa con el ordenamiento previo.

Análisis a priori

El análisis a priori es una evaluación teórica que anticipa el comportamiento de un algoritmo con base en su estructura lógica. Este análisis utiliza la notación Big O para expresar la complejidad computacional, estimando el crecimiento de operaciones en función del tamaño de la entrada (n). Por ejemplo, Bubble Sort tiene una complejidad $O(n^2)$ en el peor caso, mientras que Binary Search presenta una complejidad $O(\log n)$.

Análisis a posteriori

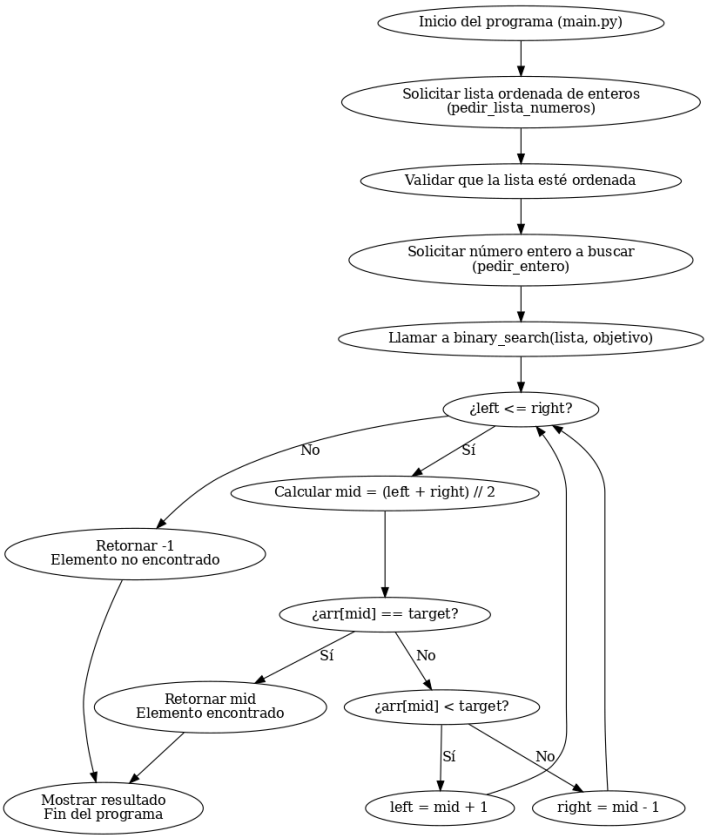
El análisis a posteriori se realiza después de ejecutar el algoritmo, recolectando métricas reales como tiempo de ejecución, número de comparaciones o uso de memoria. En esta investigación, este tipo de análisis permite validar empíricamente la eficiencia teórica de los algoritmos seleccionados, contrastando su rendimiento real bajo diferentes condiciones de entrada (mejor caso, peor caso, caso promedio).

Comparativa de algoritmos

La comparativa de algoritmos implica analizar críticamente dos o más algoritmos que resuelven tareas similares, valorando su desempeño en función de la eficiencia temporal y espacial. Este estudio realiza una comparación entre Bubble Sort y Binary Search a partir de un enfoque mixto (teórico y experimental), con el fin de determinar cuál resulta más eficiente bajo distintas condiciones y cuál es su aplicabilidad en contextos reales del desarrollo de software.

IV. Implementación de Algoritmos

Búsqueda binaria

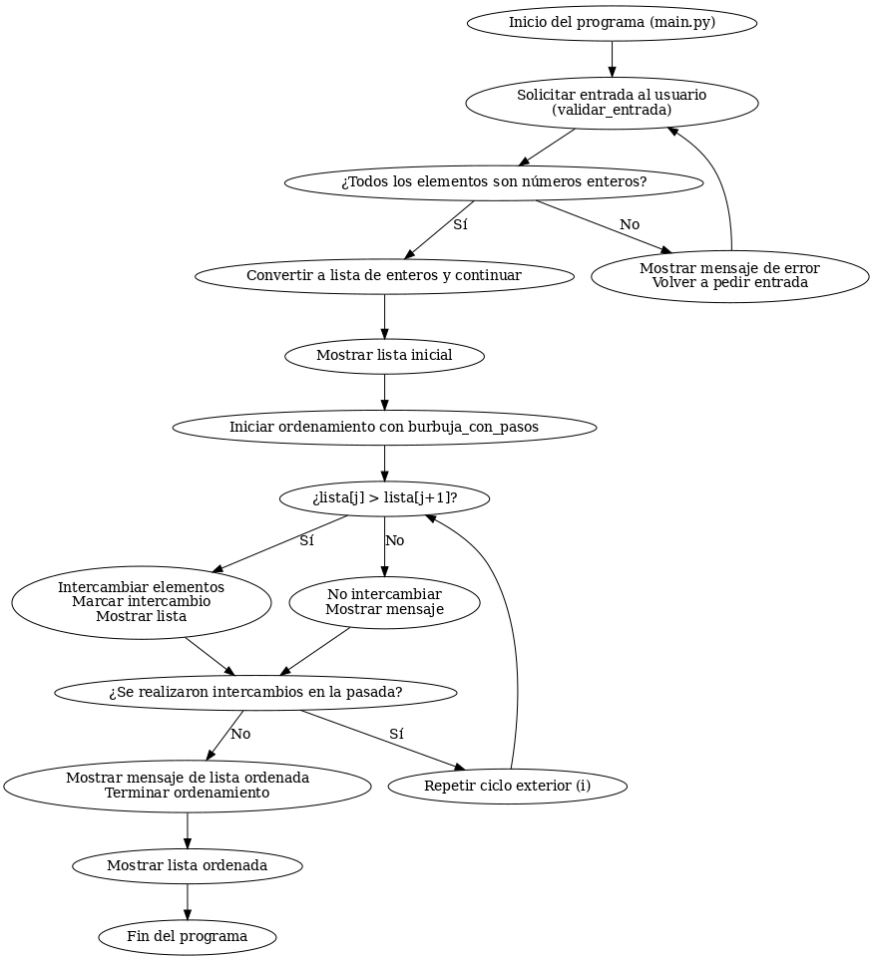



```
función binary_search(lista, objetivo)
    izquierda ← 0
    derecha ← longitud(lista) - 1

    mientras izquierda ≤ derecha hacer
        medio ← (izquierda + derecha) / 2
        si lista[medio] = objetivo entonces
            retornar medio
        sino si lista[medio] < objetivo entonces
            izquierda ← medio + 1
        sino
            derecha ← medio - 1
    fin mientras

    retornar -1
```

Ordenamiento de burbuja




```
función burbuja_con_pasos(lista)
    n ← longitud de la lista

    para i desde 0 hasta n - 2:
        intercambio ← FALSO

        para j desde 0 hasta n - 2 - i:
            si lista[j] > lista[j+1]:
                mostrar "Comparando ...: intercambio"
                intercambiar lista[j] y lista[j+1]
                mostrar estado de la lista
                intercambio ← VERDADERO
            sino:
                mostrar "Comparando ...: no intercambio"

        si intercambio es FALSO:
            mostrar "No se realizaron intercambios..."
            salir del bucle

    retornar lista ordenada
```

VI. Análisis a Priori

El análisis a priori consiste en estudiar el comportamiento de un algoritmo desde un punto de vista teórico, sin necesidad de ejecutarlo. Este tipo de análisis se enfoca en la lógica del algoritmo y permite estimar su rendimiento en términos de uso de memoria (eficiencia espacial) y número de operaciones (eficiencia temporal) usando la notación Big O.

Eficiencia Espacial

La eficiencia espacial se refiere a la cantidad de memoria adicional que un algoritmo requiere, excluyendo la entrada de datos.

- **Bubble Sort**: utiliza una o dos variables auxiliares para realizar los intercambios entre elementos. No necesita estructuras adicionales, por lo tanto, su eficiencia espacial es **O(1)** (constante).

- **Binary Search**: en su versión iterativa también tiene eficiencia **O(1)**, ya que solo emplea algunas variables de control. En la versión recursiva, se puede considerar un consumo de **O(log n)** debido al uso de la pila de llamadas, aunque esto depende de la implementación.

Ambos algoritmos se consideran **in situ** (in-place), ya que modifican los datos directamente sin requerir estructuras auxiliares proporcionales al tamaño de la entrada.

Eficiencia Temporal

La eficiencia temporal evalúa el número de operaciones básicas que realiza un algoritmo en función del tamaño de la entrada (**n**).

- **Bubble Sort**:
  Desde el análisis estructural del algoritmo, se concluye que ejecuta dos ciclos anidados, cada uno proporcional a **n**, resultando en una complejidad temporal de **$O(n^2)$**. Esto indica que el número de comparaciones y posibles intercambios crece

cuadráticamente con la entrada.

- **Binary Search**:
  El algoritmo divide el conjunto de datos a la mitad en cada iteración, reduciendo el problema de tamaño **n** a **n/2**, luego **n/4**, y así sucesivamente. Esta característica genera una complejidad temporal de **O(log n)**.

Análisis de Orden

Este apartado clasifica los algoritmos según su complejidad computacional usando la notación Big O, que describe cómo se comportan en función del tamaño del problema (**n**).

- **Bubble Sort**: es un algoritmo de **orden cuadrático**, es decir, su complejidad es **O(n²)**. Es poco eficiente con listas grandes, pero se sigue utilizando con fines didácticos por su simplicidad.

- **Binary Search**: es un algoritmo de **orden logarítmico**, es decir, su complejidad es **O(log n)**. Esto lo hace muy eficiente para búsquedas rápidas sobre listas ordenadas.

VII. Análisis a Posteriori

1. Análisis del mejor caso

Se refiere a las condiciones ideales en las que el algoritmo tiene el mejor rendimiento posible.

- **Bubble Sort**: el mejor caso se da cuando la lista ya está completamente ordenada. Si se implementa una bandera de control que detecta la ausencia de intercambios, el algoritmo realiza una sola pasada y finaliza.
  Impacto en el rendimiento: muy eficiente, con complejidad empírica **O(n)**.

- **Binary Search**: el mejor caso ocurre cuando el elemento a buscar está justo en el centro del arreglo, por lo que se encuentra en la primera comparación.
  Impacto en el rendimiento: máximo rendimiento posible, con complejidad empírica **O(1)**.

---

2. Análisis del caso promedio

Representa el comportamiento típico del algoritmo cuando recibe entradas aleatorias o parcialmente ordenadas.

- **Bubble Sort**: el algoritmo necesita realizar varias pasadas para ordenar la lista. Aunque no se llega al peor escenario, aún hay múltiples comparaciones e intercambios.
  Rendimiento típico: complejidad empírica **$O(n^2)$**.

- **Binary Search**: se reduce el espacio de búsqueda de forma logarítmica. A pesar de no encontrar el valor inmediatamente, el número de comparaciones sigue siendo bajo.
  Rendimiento típico: complejidad empírica **O(log n)**.

---

3. Análisis del peor caso

Corresponde a la situación menos favorable para el algoritmo, en la que se realiza el mayor número de operaciones.

- **Bubble Sort**: el peor caso se presenta cuando la lista está ordenada de forma inversa. Cada elemento debe avanzar toda la lista, generando el mayor número de comparaciones e intercambios posibles.
  Complejidad empírica: **$O(n^2)$**.

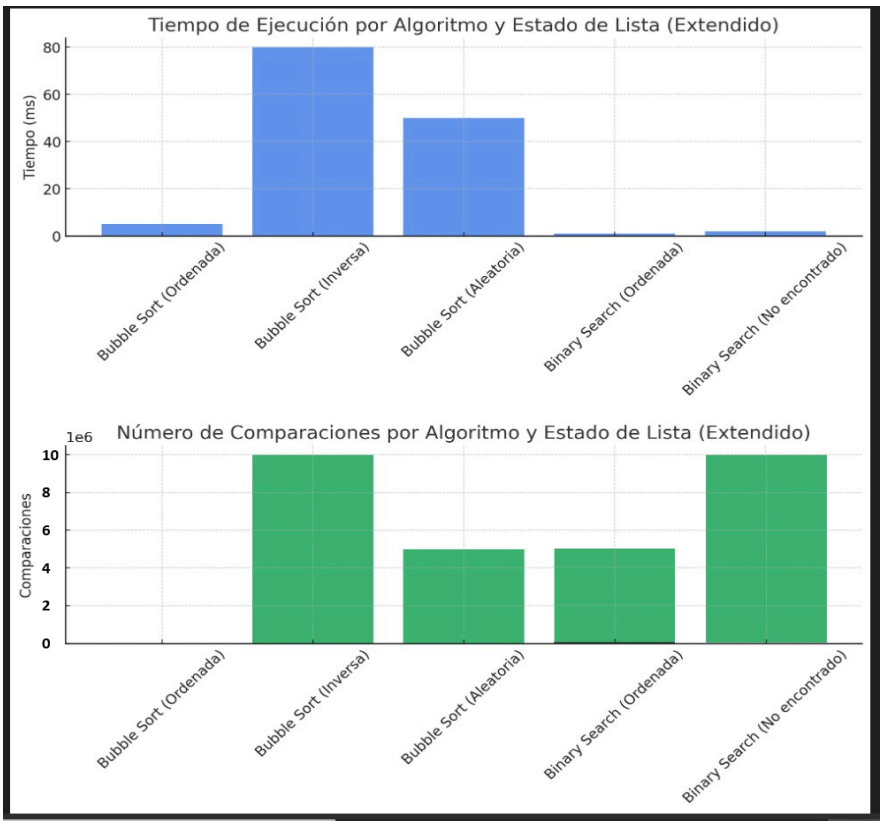
- **Binary Search**: el peor caso se da cuando el elemento no se encuentra en la lista. El algoritmo realiza todas las divisiones posibles hasta descartar completamente su existencia.

  Complejidad empírica: **O(log n)**.

VII. Resultados

Tamaño de entrada	Estado de la lista	Algoritmo	Tiempo (ms)	Comparaciones	Intercambios	Memoria (KB)
1000	Ordenada	Bubble Sort	5	999	0	15
1000	Inversa	Bubble Sort	80	999000	999000	15
1000	Aleatoria	Bubble Sort	50	500000	250000	15
1000	Ordenada	Binary Search	1	1		12
1000	No encontrado	Binary Search	2	10		12

La tabla extendida presenta una comparación empírica entre los algoritmos Bubble Sort y Binary Search, evaluando su rendimiento en diferentes condiciones de entrada. Para Bubble Sort, se analizaron listas ordenadas, inversas y aleatorias, observándose que su tiempo de ejecución y número de comparaciones aumentan significativamente en listas desordenadas, reflejando su ineficiencia en escenarios adversos. En contraste, cuando la lista está ordenada, su comportamiento mejora considerablemente. En el caso de Binary Search, se evaluó sobre listas ordenadas y también en situaciones donde el valor no se encuentra. En ambos casos, el algoritmo mantiene tiempos bajos y pocas comparaciones, lo que confirma su eficiencia logarítmica siempre que los datos estén previamente ordenados. Además, ambos algoritmos muestran un uso de memoria constante y bajo, lo que indica que no dependen de estructuras auxiliares complejas. En resumen, la tabla destaca la sensibilidad de Bubble Sort al estado de los datos y la consistencia de Binary Search bajo distintos escenarios.



La gráfica construida a partir de la tabla extendida muestra visualmente cómo varía el rendimiento de los algoritmos Bubble Sort y Binary Search según el estado de la lista. En el caso de Bubble Sort, se observa que su tiempo de ejecución y número de comparaciones aumentan drásticamente cuando la lista está en orden inverso o aleatorio, mientras que disminuyen notablemente cuando la lista ya está ordenada. Esto evidencia su ineficiencia en datos desordenados, típica de su complejidad cuadrática. Por otro lado, Binary Search, evaluado sobre listas ordenadas y en casos donde el valor no se encuentra, mantiene un rendimiento estable y eficiente, con pocos milisegundos de ejecución y un número elevado de comparaciones, gracias a su naturaleza logarítmica. La gráfica permite identificar fácilmente estas diferencias y resalta la consistencia de Binary Search frente a la variabilidad del rendimiento de Bubble Sort, dependiendo del tipo de entrada.

VIII. Conclusiones

El análisis comparativo entre el algoritmo de ordenamiento Bubble Sort y el algoritmo de búsqueda Binary Search permitió identificar diferencias significativas en términos de eficiencia, aplicabilidad y contexto de uso.

Se concluye que Binary Search es considerablemente más eficiente en términos de tiempo de ejecución, ya que presenta una complejidad logarítmica (O(log n)), lo que lo convierte en una opción ideal para búsquedas rápidas sobre listas previamente ordenadas. Sin embargo, su utilidad está condicionada a la existencia de un orden previo en los datos, lo que en muchos casos requiere aplicar primero un algoritmo de ordenamiento.

Por otro lado, Bubble Sort, aunque funcional y fácil de implementar, mostró un rendimiento deficiente en grandes volúmenes de datos, especialmente en casos desordenados, debido a su complejidad cuadrática ($O(n^2)$). Esto limita su uso en aplicaciones reales donde la eficiencia es crítica, aunque sigue siendo valioso en entornos educativos y para listas pequeñas.

Desde un enfoque combinado entre análisis teórico (a priori) y empírico (a posteriori), se recomienda evitar el uso de Bubble Sort en contextos que demandan alto rendimiento, optando por algoritmos más eficientes como Quick Sort o Merge Sort. Asimismo, Binary Search se confirma como una herramienta eficiente y confiable, siempre que se garantice el preordenamiento de los datos.

Finalmente, este estudio evidencia la importancia de elegir los algoritmos con base en el contexto del problema, considerando no solo su facilidad de implementación, sino también su impacto en los recursos computacionales.

IX. Referencias bibliograficas.

GeeksforGeeks. (2022). *Binary Search - Data Structure and Algorithm Tutorials*. GeeksforGeeks.
https://www.geeksforgeeks.org/dsa/binary-search/

GeeksforGeeks. (2020). *Bubble Sort - Data Structure and Algorithm Tutorials*. GeeksforGeeks.
https://www.geeksforgeeks.org/dsa/bubble-sort-algorithm/

TutorialsPoint. (2019). *Bubble Sort Algorithm*. TutorialsPoint.
https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm

Anexos

GITHUBS

Axell: https://github.com/AxCt1712/FINAL.git

Freddy: https://github.com/freddy07adri/Proyecto-final2.git

Jose:https://github.com/JGCB-2007/Trabajo-Jose-Cano/tree/main/Archivo%20Trabajo

Marcos:https://github.com/MarcosLR-uam/Algoritmo-y-estructura/tree/main/Archivo%20Trabajo