UNIVERSIDAD TECNOLÓGICA DEL NORTE DE GUANAJUATO

Licenciatura en Ingeniería en Tecnologías de la Información e Innovación Digital -Especialidad Desarrollo de Software

Multiplataforma

Estructura de Datos

Unidad 2 Estructura de Datos

Trabajo: Ordenamiento por Intercambio visuaalgo.net

Fernando Miguel Olvera Juárez

No. De Control 1224100597

Profesor Barrón

Dolores Hidalgo C.I.N. a martes 28 de octubre del 2025.

GTD0141

Preguntas y evidencias

- **¿Cuántos intercambios se realizaron?**

  En total fueron 7

- **¿Qué pasa si la lista ya está ordenada?**

  No pasaría nada ya que como ya esta en orden no es necesario acomodarla

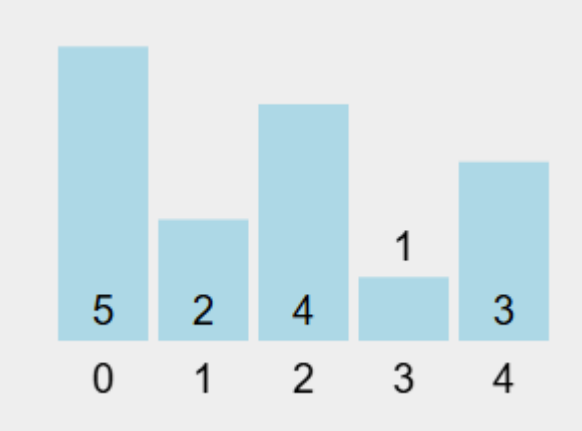
- **¿Qué eficiencia tiene este algoritmo frente a listas grandes?**

  El ordenamiento por intercambio es bien lento con listas grandes, porque compara todo con todo ($O(n^2)$), en cambio Arrays.sort() es mucho más rápido ($O(n \log n)$).

Desafíos

Desafío 1 Comprensión básica

Para este ejercicio utilicé un arreglo pequeño de 5 elementos: [5, 2, 4, 1, 3]. El objetivo fue simular el ordenamiento por intercambio (Bubble Sort) paso a paso y describir qué elementos se comparan y cuáles se intercambian.



Pase 1:

- Comparo 5 y 2 → se intercambian → [2, 5, 4, 1, 3]
- Comparo 5 y 4 → se intercambian → [2, 4, 5, 1, 3]
- Comparo 5 y 1 → se intercambian → [2, 4, 1, 5, 3]
- Comparo 5 y 3 → se intercambian → [2, 4, 1, 3, 5] (El 5 queda en su posición final.)

Pase 2:

- Comparo 2 y 4 → no se intercambian → [2, 4, 1, 3, 5]
- Comparo 4 y 1 → se intercambian → [2, 1, 4, 3, 5]
- Comparo 4 y 3 → se intercambian → [2, 1, 3, 4, 5] (El 4 queda en su posición final.)

Pase 3:

- Comparo 2 y 1 → se intercambian → [1, 2, 3, 4, 5]
- Comparo 2 y 3 → no se intercambian → [1, 2, 3, 4, 5]

Pase 4:

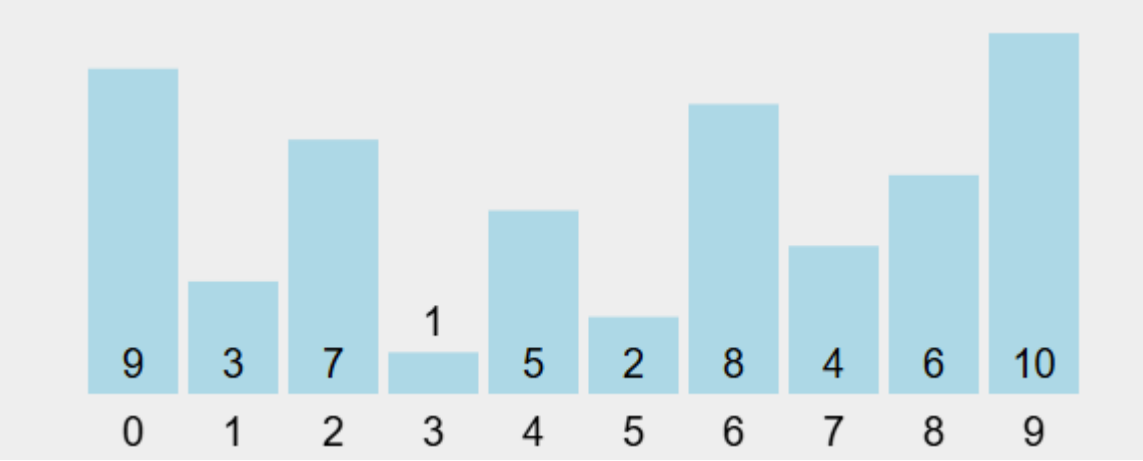
- Comparo 1 y 2 → no se intercambian → [1, 2, 3, 4, 5]

  Resultado final: [1, 2, 3, 4, 5]

  En total se realizaron **10 comparaciones** y **7 intercambios**. El algoritmo terminó con el arreglo completamente ordenado de menor a mayor.

Desafío 2 – Datos aleatorios

Para este ejercicio generé un arreglo con 10 números aleatorios y apliqué el método de ordenamiento por intercambio (Bubble Sort). A continuación muestro los estados del arreglo durante el proceso.



Estado inicial:
[9, 3, 7, 1, 5, 2, 8, 4, 6, 10]

Durante el ordenamiento, en cada pasada se comparan los elementos adyacentes y se intercambian cuando están en el orden incorrecto. Poco a poco los valores más grandes van quedando al final del arreglo.

Estado a la mitad del proceso (después de la 5ª pasada):
[3, 1, 2, 5, 4, 6, 7, 8, 9, 10]

Hasta este punto, los valores mayores (del 6 al 10) ya están en su posición correcta y el resto sigue ajustándose en las siguientes pasadas.

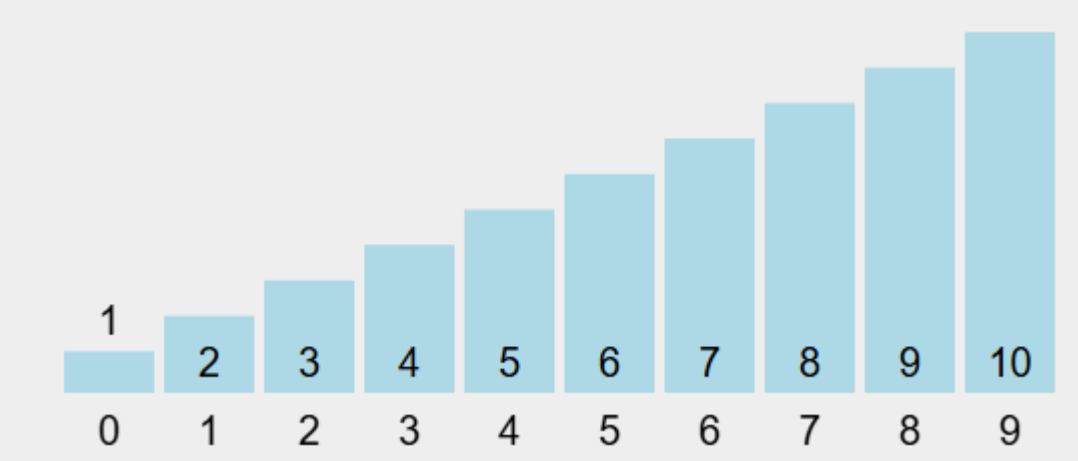
Resultado final:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Conclusión:
El algoritmo fue recorriendo el arreglo comparando e intercambiando elementos hasta dejarlo totalmente ordenado de menor a mayor. Se observa que el método realiza muchas comparaciones, pero cumple su objetivo correctamente.

Desafío 3 – Datos preordenados

Para este ejercicio utilicé un arreglo que ya está ordenado de menor a mayor:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

El objetivo fue simular el ordenamiento por intercambio (Bubble Sort) y analizar cuántas comparaciones realiza el algoritmo y si se llegan a hacer intercambios.



Simulación:
Durante el proceso, el algoritmo sigue recorriendo el arreglo comparando cada par de elementos adyacentes, pero como todos están en el orden correcto, **no se realiza ningún intercambio**.

Comparaciones realizadas:
En teoría, Bubble Sort compara los elementos aunque estén ordenados, haciendo un total de
n-1 + n-2 + ... + 1 = 45 comparaciones para 10 elementos.

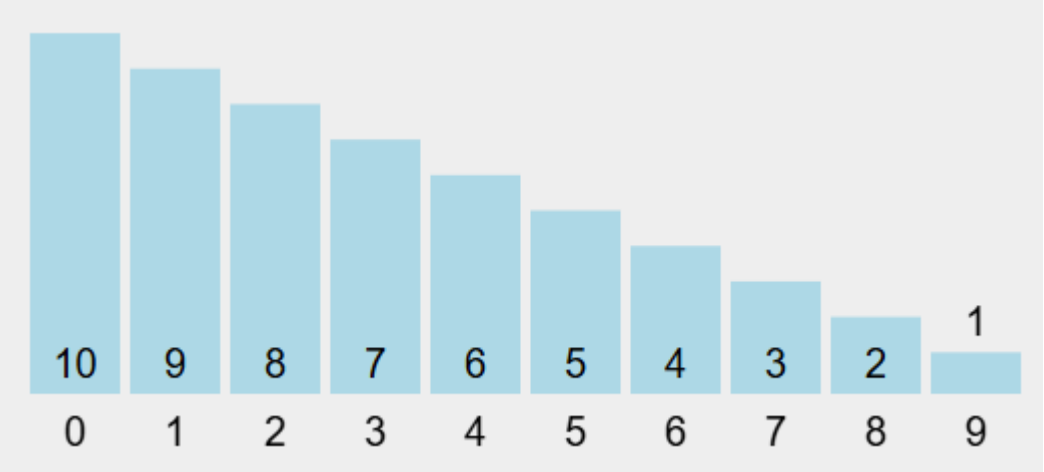
Sin embargo, algunas implementaciones más eficientes de Bubble Sort detectan que **no hubo intercambios en la primera pasada** y detienen el proceso automáticamente.
En ese caso, el algoritmo solo haría **9 comparaciones** (una por cada par) y terminaría, ya que reconoce que el arreglo ya está ordenado.

Resultado final:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
No se realizaron intercambios, y el arreglo se mantuvo igual.

Conclusión:
En un arreglo ya ordenado, el método por intercambio no modifica los datos. Aunque realiza comparaciones, puede optimizarse para detenerse al detectar que no hay cambios. Esto confirma que el **mejor caso** del algoritmo ocurre cuando los datos ya están ordenados, con una complejidad **O(n)**.

Desafío 4 – Datos inversamente ordenados

Para este ejercicio ingresé un arreglo completamente inverso, es decir, en orden descendente:
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]



El objetivo fue analizar cuántos pasos necesita el algoritmo de ordenamiento por intercambio (Bubble Sort) para dejarlo en orden ascendente.

Simulación:
En este caso, el algoritmo realiza el **máximo número posible de comparaciones e intercambios**, ya que todos los elementos están en la posición contraria.
En cada pasada, el valor más grande se va “burbujando” hasta el final del arreglo, pero el proceso se repite hasta que todos queden ordenados.

Número de pasos:

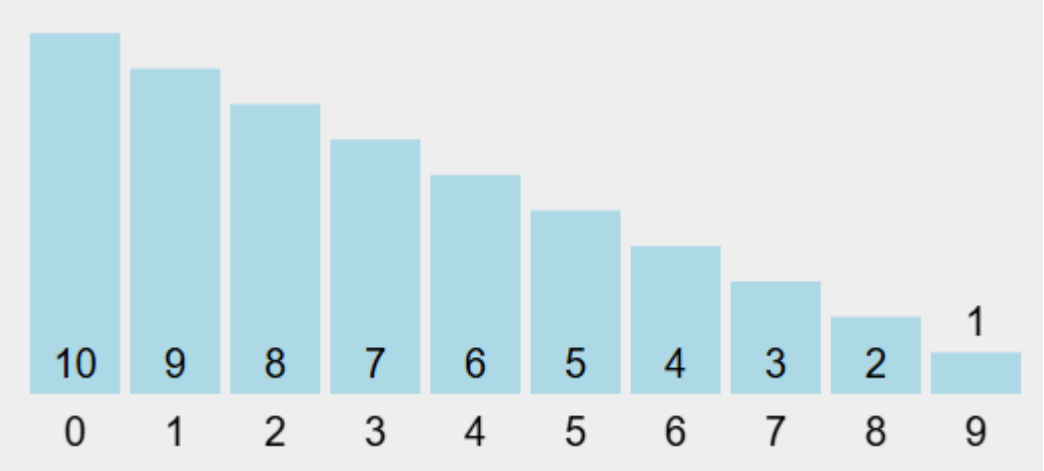
- Para un arreglo de 10 elementos, se hacen **9 pasadas completas**.
- En total se realizan **45 comparaciones** y **45 intercambios**.

Resultado final:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Conclusión:
Este es el **peor caso** para el algoritmo Bubble Sort.
Su complejidad temporal es **O(n²)**, ya que el número de comparaciones crece cuadráticamente con la cantidad de elementos. Entre más grande sea la lista, más lento se vuelve el proceso.

Desafío 5 – Comparativa con otro algoritmo

Para este desafío utilicé el mismo conjunto de datos del desafío anterior:
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]



Simulé dos algoritmos distintos: **Interchange Sort** y **Bubble Sort**, para comparar su comportamiento.

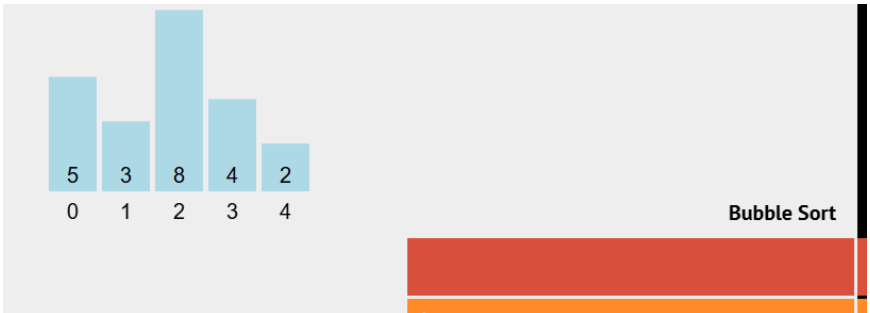
Resultados observados:

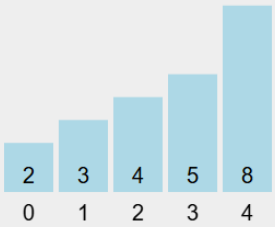
Algoritmo	Comparaciones aproximadas	Intercambios	Tiempo / Pasos	Complejidad
Bubble Sort	45	45	Más lento (repite pasadas hasta el final)	O(n²)
Interchange Sort	45	45	Un poco más rápido (no necesita pasadas extras)	O(n²)

Análisis:
Aunque ambos algoritmos tienen la misma complejidad **O(n²)**, **Interchange Sort** tiende a ser ligeramente más rápido porque compara todos los pares de elementos directamente sin hacer pasadas completas como Bubble Sort.
Sin embargo, en términos generales, ambos se consideran **ineficientes para listas grandes**, ya que el número de comparaciones crece mucho conforme aumenta el tamaño del arreglo.

Conclusión:
Los dos métodos logran ordenar correctamente el arreglo, pero **Interchange Sort puede ser un poco más eficiente** en tiempo. Aun así, para conjuntos de datos grandes conviene usar algoritmos más avanzados como **QuickSort o MergeSort**, que tienen una complejidad **O(n log n)**.

Evidencias





Bubble Sort

List is sorted!
Inversion Index = 7.

```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true; ++swapCounter
while swapped
```