UT9_TA1

**EJERCICIO 1:

✅ 1. Método de ordenación por Inserción Directa (seudocódigo)

El **algoritmo de Inserción Directa (Insertion Sort)** funciona tomando elementos uno por uno y colocándolos en la posición correcta dentro del subconjunto ordenado a la izquierda.

🔷 Seudocódigo:

```
Para i desde 1 hasta n-1 hacer:
    clave ← arreglo[i]
    j ← i - 1
    Mientras j ≥ 0 y arreglo[j] > clave hacer:
        arreglo[j + 1] ← arreglo[j]     // movimiento
        j ← j - 1                       // comparación
    arreglo[j + 1] ← clave              // movimiento final
```

📊 Análisis del orden de ejecución:

Caso	Comparaciones	Movimientos	Orden Temporal
Mejor caso (ordenado)	n - 1	0	**O(n)**
Peor caso (invertido)	n(n - 1)/2	n(n - 1)/2	**$O(n^2)$**
Caso promedio	≈ n^2 / 4	≈ n^2 / 4	**$O(n^2)$**

✅ 2. Aplicación paso a paso al conjunto de datos

Conjunto original:

```
256 - 458 - 655 - 298 - 043 - 648 - 778 - 621 - 655 - 019 - 124 - 847
```

Empezamos desde el índice 1.

📋 Tabla de evolución

Paso	Clave	Arreglo actual	C	M
1	458	256 - 458	1	0
2	655	256 - 458 - 655	2	0
3	298	256 - 298 - 458 - 655	3	3
4	043	043 - 256 - 298 - 458 - 655	4	4
5	648	043 - 256 - 298 - 458 - 648 - 655	4	1
6	778	043 - 256 - 298 - 458 - 648 - 655 - 778	5	0
7	621	043 - 256 - 298 - 458 - 621 - 648 - 655 - 778	6	3
8	655	043 - 256 - 298 - 458 - 621 - 648 - 655 - 655 - 778	7	2
9	019	019 - 043 - 256 - ...	8	9
10	124	019 - 043 - 124 - 256 ...	9	3
11	847	019 - 043 - 124 - 256 - ... - 847	10	0

🔢 Total:

- Comparaciones (C): **1 + 1 + 2 + 3 + 1 + 0 + 3 + 2 + 8 + 3 + 0 = 24**
- Movimientos (M): **0 + 0 + 2 + 3 + 1 + 0 + 3 + 2 + 8 + 3 + 0 = 22**

✅ Resultado Final:

```
019 - 043 - 124 - 256 - 298 - 458 - 621 - 648 - 655 - 655 - 778 - 847
```

EJERCICIO 2: Shellsort paso a paso con la secuencia de incrementos (5, 3, 1) aplicada al conjunto:

```
256 - 458 - 655 - 298 - 043 - 648 - 778 - 621 - 655 - 019 - 124 - 847
```

✅ 1. Shellsort paso a paso (con incrementos 5, 3, 1)

La idea de **Shellsort** es aplicar **insertion sort en subarreglos separados por el salto ("gap")**.

🔷 Secuencia de incrementos: (5, 3, 1)

🔶 Paso con gap = 5

Agrupamos así:
Los subarreglos son (índices separados por 5):

- Grupo 1: 0, 5 → 256, 648
- Grupo 2: 1, 6 → 458, 778
- Grupo 3: 2, 7 → 655, 621
- Grupo 4: 3, 8 → 298, 655
- Grupo 5: 4, 9 → 043, 019
- Grupo 6: 10 → 124
- Grupo 7: 11 → 847

Aplicamos Insertion Sort a cada uno:

Grupo	Ordenado	Comparaciones (C)	Movimientos (M)
256, 648	256, 648	1	0
458, 778	458, 778	1	0
655, 621	621, 655	1	1
298, 655	298, 655	1	0
043, 019	019, 043	1	1
124	124	0	0
847	847	0	0

C: 5, M: 2

🔽 Arreglo tras gap=5:

```
256 - 458 - 621 - 298 - 019 - 648 - 778 - 655 - 655 - 043 - 124 - 847
```

🔶 Paso con gap = 3

Subarreglos:

- (0,3,6,9): 256, 298, 778, 043 → **043, 256, 298, 778**
- (1,4,7,10): 458, 019, 655, 124 → **019, 124, 458, 655**
- (2,5,8,11): 621, 648, 655, 847 → ya está ordenado

Aplicamos Insertion Sort:

Grupo	Ordenado	C	M
256, 298, 778, 043 → 043, 256, 298, 778	3	3	
458, 019, 655, 124 → 019, 124, 458, 655	3	3	
621, 648, 655, 847 → igual	3	0	

C: 9, M: 6

🔽 Arreglo tras gap=3:

```
043 - 019 - 621 - 256 - 124 - 648 - 298 - 458 - 655 - 778 - 655 - 847
```

🔶 Paso final con gap = 1 (Insertion Sort completo)

Aplicamos insertion sort al arreglo actual:

```
043 - 019 - 621 - 256 - 124 - 648 - 298 - 458 - 655 - 778 - 655 - 847
```

Se ordena paso a paso:

Iteración	Clave	Comparaciones	Movimientos
1 (019)	1	1	1
2 (621)	2	0	0
3 (256)	3	1	1
4 (124)	4	2	2
5 (648)	5	0	0
6 (298)	6	2	2
7 (458)	7	2	2
8 (655)	8	0	0

Iteración	Clave	Comparaciones	Movimientos
9 (778)	9	0	0
10 (655)	10	1	1
11 (847)	11	0	0

C: 10, M: 9

⬅️ END Arreglo final:

```
019 - 043 - 124 - 256 - 298 - 458 - 621 - 648 - 655 - 655 - 778 - 847
```

✅ Totales

Gap	Comparaciones (C)	Movimientos (M)
5	5	2
3	9	6
1	10	9
TOTAL	**24**	**17**

✅ 2. Análisis del orden de ejecución de Shellsort

Shellsort depende fuertemente de la secuencia de incrementos:

Caso	Complejidad
Peor caso (original de Shell)	$O(n^2)$
Mejores secuencias (Sedgewick, Hibbard, etc.)	O(n^3/2), $O(n \log^2 n)$, hasta $O(n \log n)$

> Con secuencia (5, 3, 1), el orden sigue siendo cercano a **$O(n^2)$** en el peor caso, pero es más eficiente que insertion sort en la práctica.

EJERCICIO 3 - Método de la BURBUJA (Bubble Sort) paso a paso.

✅ 1. Ordenar el vector paso a paso con Bubble Sort

Vector inicial:

```
256 - 458 - 655 - 298 - 043 - 648 - 778 - 621 - 655 - 019 - 124 - 847
```

🔷 Lógica del algoritmo:

```
Para i desde 0 hasta n-2:
    Para j desde 0 hasta n-2-i:
        Si arreglo[j] > arreglo[j+1]:
            Intercambiar arreglo[j] y arreglo[j+1]
```

Cada **pasada** lleva el valor más grande al final del subconjunto.

📋 Iteraciones paso a paso con Comparaciones (C) y Movimientos (M)

Vamos a registrar comparaciones (C) y movimientos (M) por **pasada completa**.

🔶 Pasada 1:

Comparamos pares y movemos si están desordenados. Se hacen 11 comparaciones (n−1 = 12−1).

Resultado:

```
256 - 458 - 298 - 043 - 648 - 655 - 621 - 655 - 019 - 124 - 778 - 847
```

C: 11, M: 8

🔶 Pasada 2:

```
256 - 298 - 043 - 458 - 648 - 621 - 655 - 019 - 124 - 655 - 778 - 847
```

C: 10, M: 7

🔶 Pasada 3:

```
256 - 043 - 298 - 458 - 621 - 648 - 019 - 124 - 655 - 655 - 778 - 847
```

C: 9, M: 6

🔶 Pasada 4:

```
043 - 256 - 298 - 458 - 621 - 019 - 124 - 648 - 655 - 655 - 778 - 847
```

C: 8, M: 5

🔶 Pasada 5:

```
043 - 256 - 298 - 458 - 019 - 124 - 621 - 648 - 655 - 655 - 778 - 847
```

C: 7, M: 4

🔶 Pasada 6:

```
043 - 256 - 298 - 019 - 124 - 458 - 621 - 648 - 655 - 655 - 778 - 847
```

C: 6, M: 3

🔶 Pasada 7:

```
043 - 256 - 019 - 124 - 298 - 458 - 621 - 648 - 655 - 655 - 778 - 847
```

C: 5, M: 3

🔶 Pasada 8:

```
043 - 019 - 124 - 256 - 298 - 458 - 621 - 648 - 655 - 655 - 778 - 847
```

C: 4, M: 2

🔶 Pasada 9:

```
019 - 043 - 124 - 256 - 298 - 458 - 621 - 648 - 655 - 655 - 778 - 847
```

C: 3, M: 2

🔶 Pasada 10:

No se hace ningún movimiento.

C: 2, M: 0

🔶 Pasada 11:

Ya está ordenado, sin movimientos.

C: 1, M: 0

✅ Resumen

Pasada	Comparaciones (C)	Movimientos (M)
1	11	8

Pasada	Comparaciones (C)	Movimientos (M)
2	10	7
3	9	6
4	8	5
5	7	4
6	6	3
7	5	3
8	4	2
9	3	2
10	2	0
11	1	0
TOTAL	**66**	**40**

✅ 2. Análisis del orden de ejecución de Bubble Sort

Caso	Comparaciones	Movimientos	Complejidad
Mejor caso	$n - 1$	0	**$O(n)$**
Peor caso	$n(n - 1)/2$	$n(n - 1)/2$	**$O(n^2)$**
Promedio	$\approx n^2 / 2$	$\approx n^2 / 2$	**$O(n^2)$**

✅ 3. ¿Cómo reducir el tiempo de ejecución?

Puedes optimizar con una **bandera (flag)** para detectar si ya está ordenado:

```
Para i desde 0 hasta n-2:
    bandera ← falso
    Para j desde 0 hasta n-2-i:
        Si arreglo[j] > arreglo[j+1]:
            Intercambiar
            bandera ← verdadero
    Si bandera == falso:
        Terminar (ya está ordenado)
```

🟢 Esta **mejora** hace que el algoritmo termine **antes si no hubo intercambios**, reduciendo mucho el tiempo en casos casi ordenados (como en las últimas pasadas del ejemplo).
