

ALGORITMOS CYCLE SORT VS INTERCAMBIO DIRECTO CON SEÑAL

INTEGRANTES:

- Franklin Jhonatan Mamani Apaza
- Josue Fernando Mamani Lima
- Josue Santiago Maquera Pilco

2023-119030

2025-119055

2025- 119005



INTRODUCCIÓN

El presente estudio analiza y compara los algoritmos Cycle Sort y Intercambio Directo con Señal, dos métodos clásicos de ordenamiento.

Ambos buscan organizar datos con diferentes enfoques en eficiencia y complejidad.

Cycle Sort se destaca por su mínima cantidad de escrituras, mientras que el Intercambio Directo con Señal optimiza la detección de listas ya ordenadas.

La comparación entre ambos permite evaluar su rendimiento frente a distintos tamaños de entrada.

De esta forma, se busca determinar cuál resulta más eficiente en contextos de programación y optimización computacional.

OBJETIVOS

dar a conocer el metodo de intercambio directo con señal, como es su funcionamiento y que tan eficiente es

dar a conocer el metodo de cycle sort, como es su funcionamiento y que tan eficiente es

realizar una comparacion entre los dos algoritmos y determinar su complejidad asi como mostrar cual seria mas conveniente en diferentes situaciones

CYCLE SORT

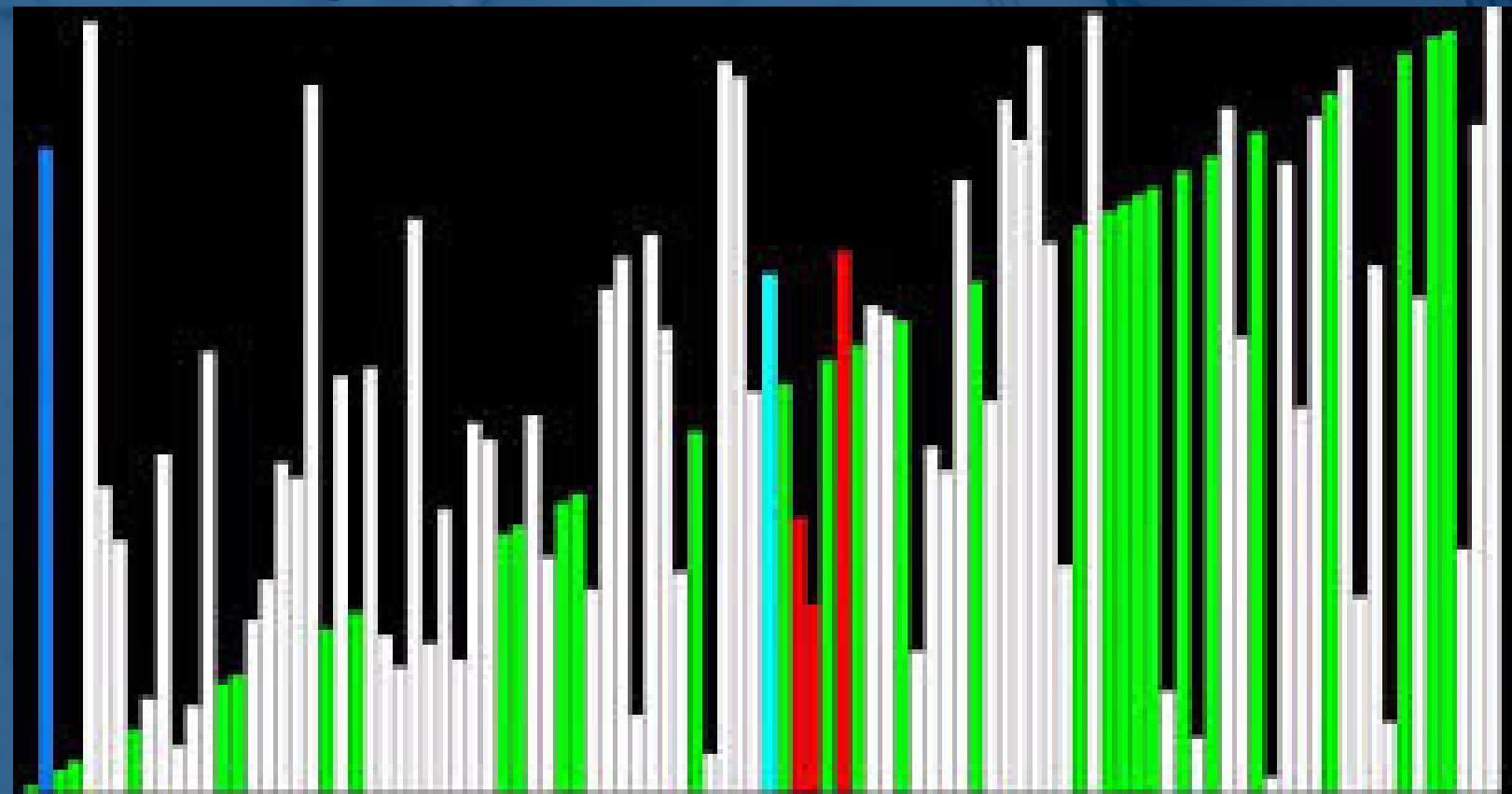
El Cycle Sort es un algoritmo de ordenamiento que busca minimizar el número de escrituras en memoria.

Su principio se basa en identificar ciclos dentro del arreglo y colocar cada elemento en su posición correcta.

A diferencia de otros métodos, realiza muy pocos intercambios, lo que lo hace útil cuando las escrituras son costosas.

Tiene una complejidad $O(n^2)$ en promedio, aunque destaca por su eficiencia en operaciones de escritura.

Se emplea en contextos donde la durabilidad del almacenamiento es prioritaria, como en memorias flash o EEPROM.



INTERCAMBIO DIRECTO CON SEÑAL

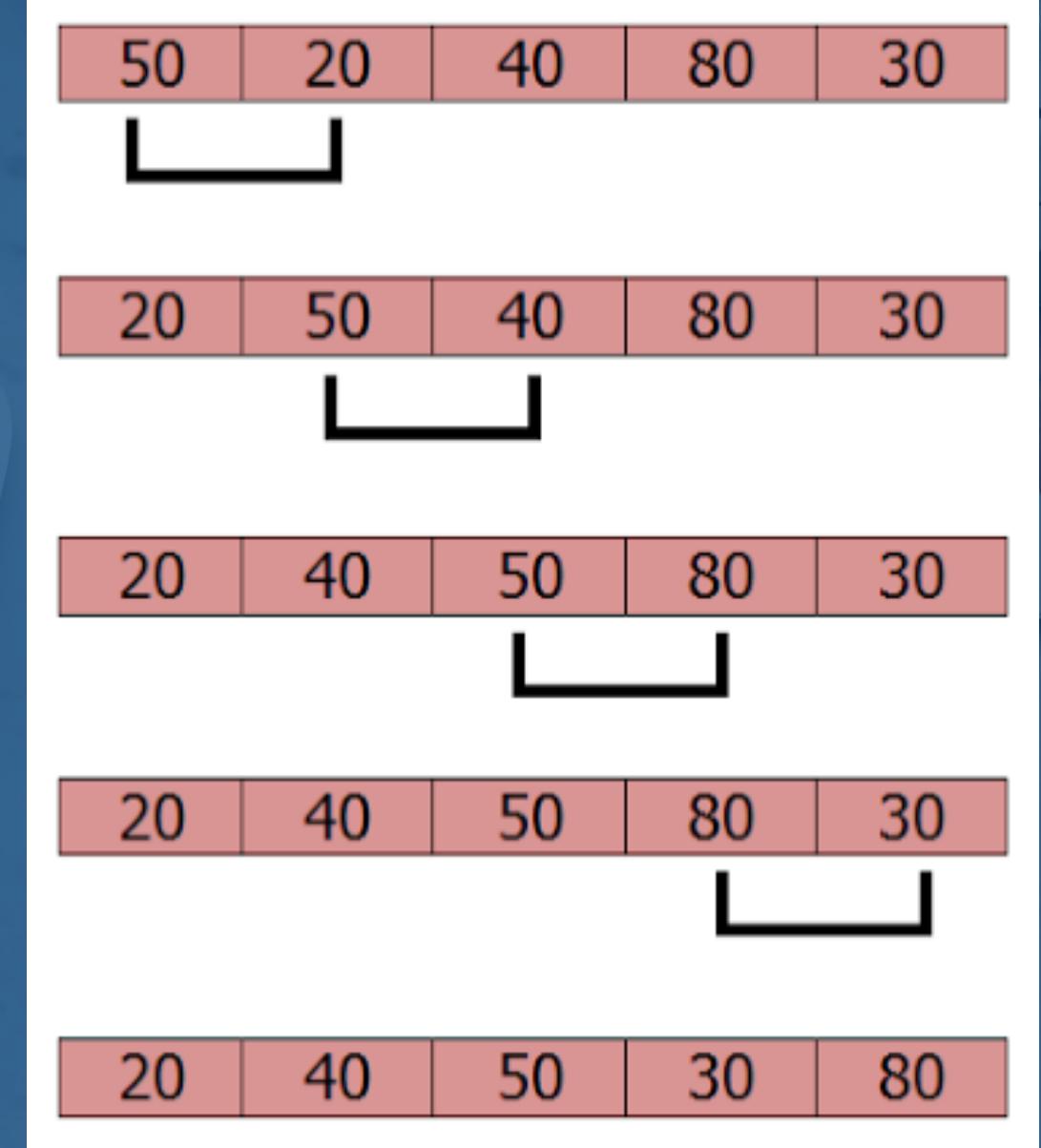
El Intercambio Directo con Señal, también conocido como Bubble Sort optimizado, mejora el rendimiento del método clásico.

Utiliza una señal o bandera que detecta si se realizaron intercambios en una pasada.

Si no hay cambios, el algoritmo finaliza antes, evitando comparaciones innecesarias.

Mantiene una complejidad $O(n^2)$, pero su eficiencia aumenta en listas parcialmente ordenadas.

Es un método sencillo, de fácil implementación y útil en entornos educativos o de bajo requerimiento computacional.



INTERCAMBIO DIRECTO CON SEÑAL

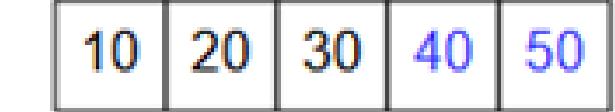
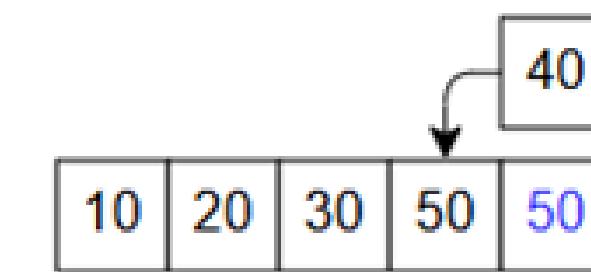
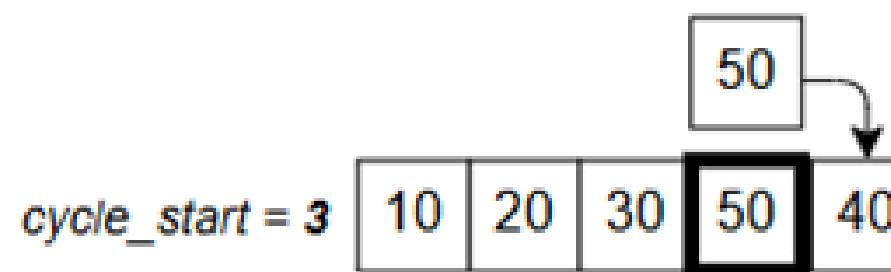
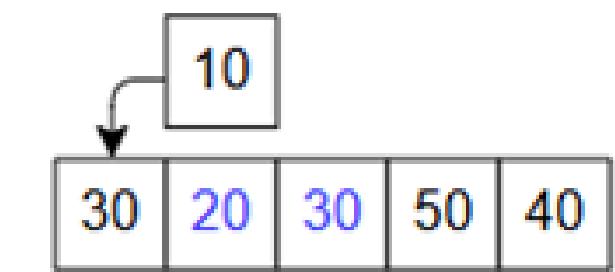
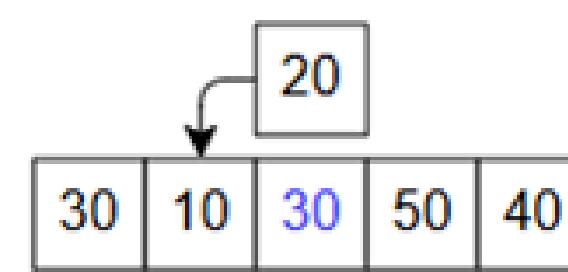
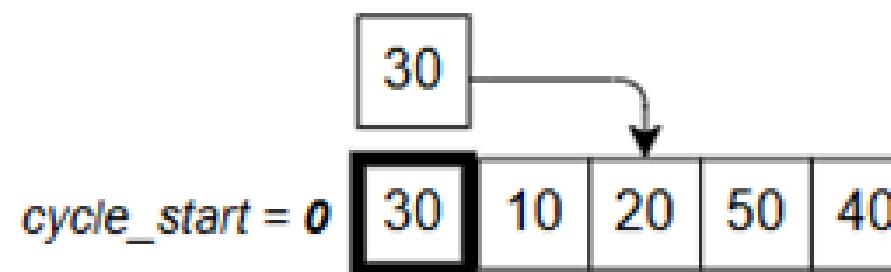
CODIGO

```
while (i <= n - 1 && cen == 1) {  
    cen = 0;  
    for (int j = 0; j < n - i; j++) {  
        if (A[j] > A[j + 1]) {  
            int aux = A[j];  
            A[j] = A[j + 1];  
            A[j + 1] = aux;  
            cen = 1;  
        }  
    }  
    i = i + 1;  
}
```

CYCLE SORT

cycle sort

como funciona cycle sort?



CYCLE SORT

algoritmo de cycle sort

```
void cycle_sort(int A[], int n) {
    // writes: contador del número de intercambios que se hacen al arreglo
    int writes = 0;
    // Recorremos desde el primer índice hasta el penúltimo,
    for (int inicio = 0; inicio <= n - 2; inicio++) {
        // "item" es el valor que queremos colocar correctamente en este ciclo
        int item = A[inicio];
        // "pos" empezará como la posición inicial del ciclo
        int pos = inicio;
        // Determinar la posición correcta de "item" contando
        for (int i = inicio + 1; i < n; i++) {
            if (A[i] < item) {
                pos++;
            }
        }
        // Si pos sigue siendo igual a inicio, eso significa que "item" ya está en su lugar
        if (pos == inicio) {
            continue;
        }
        // Si ya hay elementos iguales a "item" en la ubicación de destino, avanza
        while (item == A[pos]) {
            pos += 1;
        }
        // Si pos es distinto de inicio, intercambiamos "item" con A[pos].
        if (pos != inicio) {
            swap(item, A[pos]);
            writes++;
        }
    }
}
```

```
// Ahora seguimos rotando los elementos del ciclo hasta que lleguemos al inicio
while (pos != inicio) {
    // Reiniciar pos para encontrar el siguiente lugar correcto para "item"
    pos = inicio;
    // Nuevamente contar cuántos elementos menores que "item" hay
    // después de "inicio", para hallar la posición final correcta.
    for (int i = inicio + 1; i < n; i++) {
        if (A[i] < item) {
            pos += 1;
        }
    }
    // Si hay elementos iguales delante, saltar posición hasta una menor que "item"
    while (item == A[pos]) {
        pos += 1;
    }
    // Intercambiar si no está ya en su lugar
    if (item != A[pos]) {
        swap(item, A[pos]);
        writes++;
    }
}
// Al final, imprime cuántos intercambios se hicieron en total
cout << writes << endl;
```

RESULTADOS

A continuacion se mostrara los resultados de los tiempos obtenidos de los dos algoritmos:

intercambio directo con señal

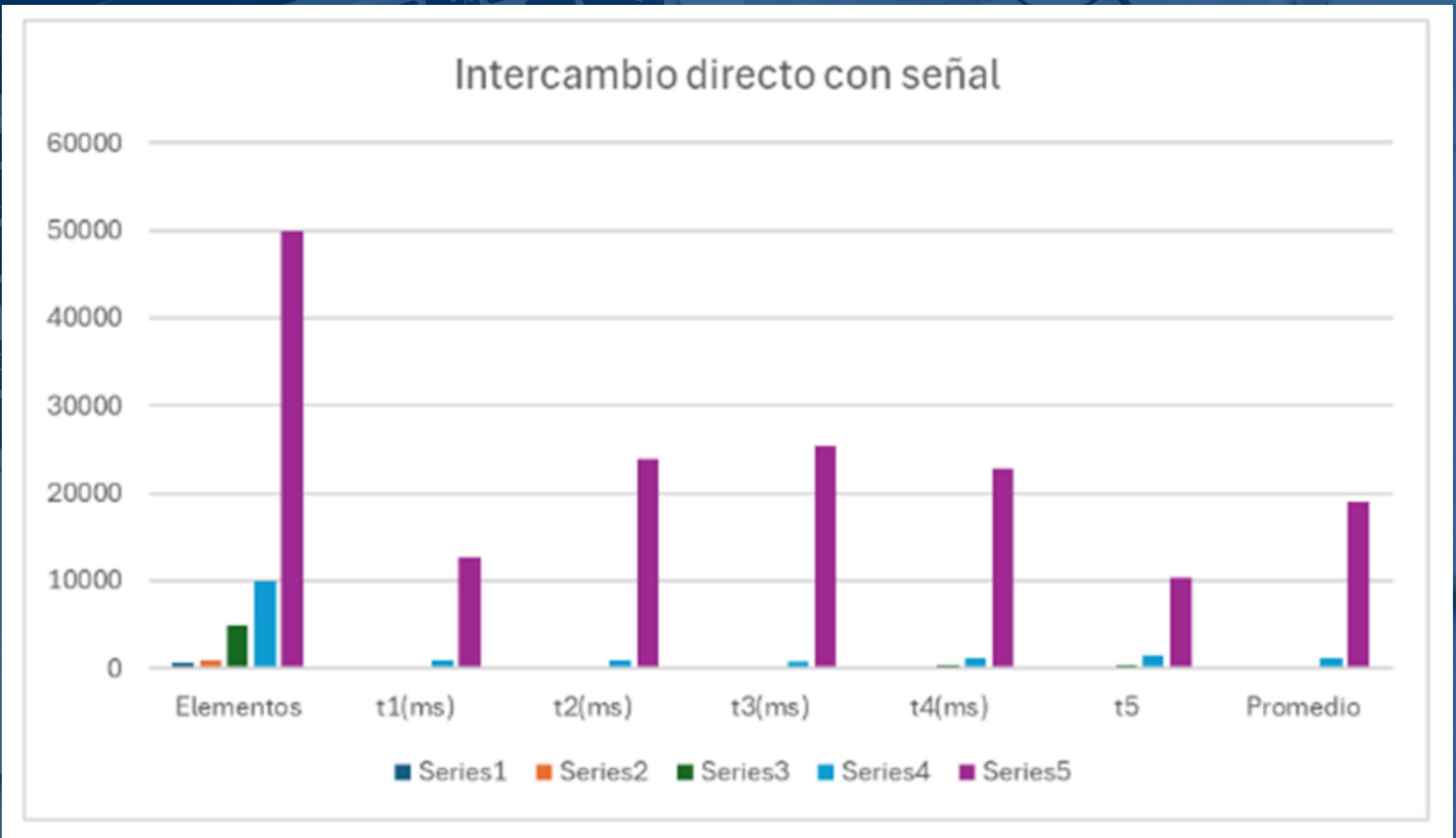
Elementos	t ₁ (ms)	t ₂ (ms)	t ₃ (ms)	t ₄ (ms)	t ₅ (ms)	Promedio
500	0	2	1	2	2	1.4
1000	10	11	9	14	10	10.8
5000	300	231	190	379	376	295.2
10000	947	974	746	1128	1515	1062
50000	12668	23882	25288	22773	10373	18996.8

Elementos	t ₁ (ms)	t ₂ (ms)	t ₃ (ms)	t ₄ (ms)	t ₅ (ms)	Promedio
500	9	10	4	2	6	6.2
1000	19	22	24	22	13	20
5000	504	285	341	516	654	460
10000	2000	925	2140	1256	2197	1703.6
50000	40289	36325	74614	37209	74804	52648.2



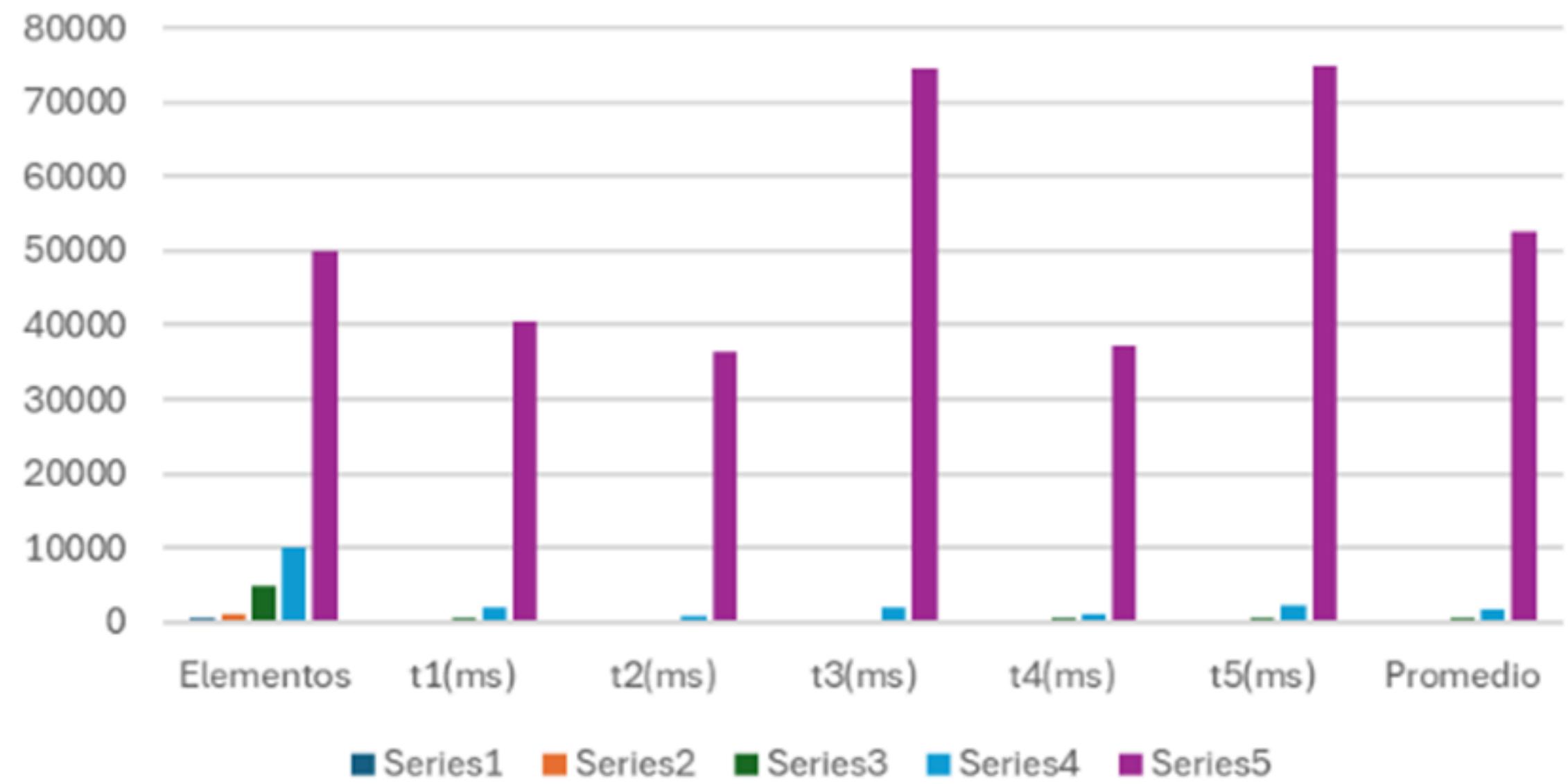
cycle sort

COMPARACIONES



COMPARACIONES

Cycle sort



COMPLEJIDAD

Algoritmo	Mejor caso	Caso promedio	Peor caso
Cycle Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Intercambio Directo con Señal	$O(n)$	$O(n^2)$	$O(n^2)$

CONCLUSIÓN

La comparación entre Cycle Sort y Intercambio Directo con Señal demuestra diferencias notables en su eficiencia.

Cycle Sort destaca por reducir al mínimo las escrituras, siendo ideal para dispositivos con memoria limitada.

En cambio, el Intercambio Directo con Señal optimiza el proceso cuando los datos están parcialmente ordenados.

Ambos algoritmos mantienen una complejidad similar, pero difieren en su aplicación práctica.

El estudio evidencia que la elección del método depende del contexto y de los recursos disponibles.

¡Eso es todo, amigos!



GRACIAS