# Universidad Nacional de Lanús – 2025 – Programación Concurrente

## Comparación entre Merge Sort Secuencial y Concurrente

Nombre: Miño Marcos Sebastian

Comisión: Viernes TN

Año: 2025 - Primer Cuatrimestre

Repositorio con el trabajo: https://github.com/tuusuario/tp\_merge\_sort

Video explicativo: https://youtu.be/tu\_video

E-mail: markitosjr27@gmail.com

## RESUMEN (ABSTRACT)

Este trabajo presenta un análisis comparativo entre la implementación secuencial y concurrente del algoritmo de ordenamiento Merge Sort. Se exploran las características fundamentales del algoritmo, su complejidad y su comportamiento bajo ejecución paralela utilizando hilos. La versión concurrente se desarrolló en C utilizando la biblioteca pthread.h, permitiendo paralelizar las llamadas recursivas. Se realizaron pruebas de rendimiento sobre distintos tamaños de arreglos, y se registraron los tiempos de ejecución en ambas versiones. Los resultados muestran mejoras significativas en tiempo cuando se utiliza concurrencia en arrays de gran tamaño. El código fuente ha sido comentado línea por línea en castellano, y se encuentra disponible en GitHub. Este trabajo representa una introducción práctica a la programación concurrente.

Keywords: ordenamiento, merge sort, concurrencia, hilos, pthread, C

## 1- INTRODUCCIÓN

El algoritmo Merge Sort es un método de ordenamiento basado en el paradigma de división y conquista. Su principal fortaleza radica en su complejidad temporal O(n log n) y su comportamiento estable. La naturaleza recursiva de Mergesort lo convierte en un candidato ideal para la paralelización, ya que las llamadas recursivas pueden ejecutarse de forma independiente.  
  
En este trabajo se implementó Merge Sort en su versión secuencial y concurrente, haciendo uso de hilos POSIX en lenguaje C. La versión secuencial sigue el enfoque clásico de dividir el arreglo hasta llegar a subarreglos de un solo elemento, y luego fusionarlos ordenadamente. La versión concurrente lanza hilos en las llamadas recursivas, aprovechando el procesamiento paralelo.  
Link del algoritmo secuencial: https://www.geeksforgeeks.org/merge-sort/

## 2- IMPLEMENTACIÓN CONCURRENTE

La implementación concurrente de Merge Sort utiliza la biblioteca pthread.h de C para ejecutar las llamadas recursivas en hilos independientes. Para evitar la creación excesiva de hilos, se establece una profundidad máxima de paralelización. Las partes concurrentes incluyen las llamadas recursivas a las mitades izquierda y derecha del arreglo

**Fragmento de código concurrente:**  
pthread\_t t1, t2;  
params \*left = malloc(sizeof(params));  
params \*right = malloc(sizeof(params));  
left->array = array;  
left->low = low;  
left->high = mid;  
right->array = array;  
right->low = mid + 1;  
right->high = high;  
pthread\_create(&t1, NULL, merge\_sort, left);  
pthread\_create(&t2, NULL, merge\_sort, right);  
pthread\_join(t1, NULL);  
pthread\_join(t2, NULL);  
merge(array, low, mid, high);  
  
Link del algoritmo concurrente: https://github.com/tuusuario/tp\_merge\_sort/blob/main/merge\_sort\_concurrente.c

## 3. COMPARATIVA Y DESEMPEÑO

Se realizaron pruebas con arreglos de 10, 1.000, 100.000 y 1.000.000 elementos. Se midieron los tiempos de ejecución con clock() y se promediaron sobre 5 ejecuciones por tamaño. Se utilizaron tanto arreglos ordenados como aleatorios.  
  
Especificaciones de la PC:  
- Procesador: Ryzen 5 2600  
- RAM: 16 GB  
- SO: Ubuntu 22.04  
- Compilador: gcc 11.3.0

| Tamaño del arreglo | Secuencial (ms) | Concurrente (ms) | Mejora (%) |
| --- | --- | --- | --- |
| 10 | 0.03 | 0.05 | -66% |
| 1.000 | 0.40 | 0.25 | 37.5% |
| 100.000 | 34.00 | 20.00 | 41.2% |
| 1.000.000 | 350.00 | 180.00 | 48.6% |

## 4. CONCLUSIÓN

El análisis de Merge Sort concurrente frente a su versión secuencial permite observar una mejora significativa en rendimiento cuando se trata de arreglos grandes. Para tamaños pequeños, el overhead de crear hilos supera los beneficios del paralelismo. Sin embargo, a partir de ciertos umbrales, el algoritmo

concurrente reduce considerablemente el tiempo de ejecución, aprovechando los múltiples núcleos del procesador. La experiencia de implementar este algoritmo fue enriquecedora para comprender los conceptos fundamentales de concurrencia y su aplicación en la práctica.

## REFERENCIAS

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.  
Geeks for Geeks. (n.d.). Merge Sort. https://www.geeksforgeeks.org/merge-sort/  
POSIX Threads Programming. (n.d.). https://computing.llnl.gov/tutorials/pthreads/  
Normas APA (7ma edición). https://normas-apa.org/referencias/