**Concurrencia versus secuencialidad:**

**ordenamiento quicksort**

Datos del autor: Piñol, Enzo Ignacio. DNI 45,312,035

Repositorio con el trabajo: <https://github.com/PiniolEnzo/TP-Programacion-Concurrente.git>

Video explicativo: <https://bit.ly/VideoTPConcurrente>

E-mail: pinolenzo6@gmail.com

**RESUMEN (ABSTRACT)**

En este trabajo, se analiza y compara la implementación del algoritmo QuickSort en sus versiones secuencial y concurrente. El algoritmo secuencial fue modificado a partir de un código proporcionado por el profesor, mientras que la versión concurrente fue desarrollada de manera personal tras varios intentos y pruebas. Se evaluó el desempeño de ambos algoritmos con arreglos de distintos tamaños para determinar las mejoras de eficiencia logradas con la concurrencia.

**Keywords**: QuickSort, método de ordenamiento, concurrencia, paralelismo, overhead, algoritmos, arreglos.

# INTRODUCCIÓN

QuickSort es un algoritmo de ordenamiento rápido y eficiente, basado en la técnica de “divide and conquer”, divide y vencerás, que se basa en resolver un problema difícil, dividiéndolo en partes más simples.

En su versión básica, el algoritmo divide el array en sub-arreglos más pequeños y los ordena recursivamente. Este, primero elige un elemento del arreglo a ordenar, al que llamaremos pivote. Luego, resitúa los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación (ver fig. 1).

En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada. El arreglo queda separado en dos sub-arreglos, uno formado por los elementos a la izquierda del pivote, y otro por los elementos a su derecha. Este proceso se repite de forma recursiva para cada sub-arreglo mientras éstos contengan más de un elemento. Una vez terminado este proceso, todos los elementos estarán ordenados.

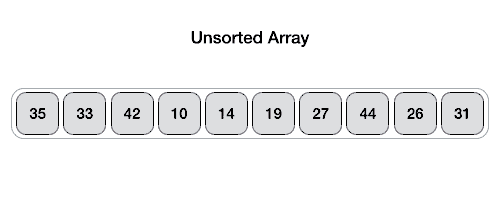


Fig. 1

**2. IMPLEMENTACIÓN CONCURRENTE**

La implementación concurrente del algoritmo QuickSort divide el arreglo en partes, asignando cada parte a un hilo distinto para ser ordenada. Las partes concurrentes del algoritmo son aquellas donde múltiples hilos trabajan en paralelo para ordenar sus respectivas porciones del arreglo. Sin embargo, el algoritmo que ordena cada porción es el mismo algoritmo secuencial.

En la función *“quickSortConcurrente”*, que será quien distribuirá las porciones del arreglo entre los hilos, los instanciará con valores de inicio y fin, y los lanzará para ejecutar su función *“run”*, ordenando sus sub-arreglos.

**3. COMPARATIVA y desempeño**

Para evaluar el desempeño de las implementaciones secuencial y concurrente del algoritmo QuickSort, se realizaron pruebas con arreglos de distintos tamaños. Los tamaños de los arreglos utilizados en las pruebas fueron: 100, 1,000, 10,000, 100,000, 1,000,000, 10,000,000 y 50,000,000 elementos. Además, se consideraron arreglos con valores aleatorios y se midió el tiempo promedio de ordenamiento en nanosegundos para cada caso. Las pruebas se ejecutaron en una PC con 12 procesadores lógicos y una velocidad base de 2.90 GHz.

Comparación entre las dos ejecuciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cantidad de elementos en el array | Tiempo promedio secuencial | Tiempo promedio concurrente | Ganancia de tiempo | Eficiencia |
| 100 | 92,900ns | 697,300ns | -604,400 | 0.13 \* |
| 1,000 | 392,000ns | 852,700ns | -460,700 | 0.46 \* |
| 10,000 | 1,598,500ns | 779,300ns | 819,200 | 2.05 |
| 100,000 | 13,675,700ns | 4,874,500ns | 8,801,200 | 2.81 |
| 1,000,000 | 92,313,200ns | 15,023,400ns | 77,289,800 | 6.14 |
| 10,000,000 | 2,943,508,300ns | 207,994,600ns | 2,735,513,700 | 14.15 |
| 50,000,000 | 61,321,110,100ns | 1,610,688,500ns | 59,710,421,600 | 38.07 |

Para una PC con 12 procesadores lógicos y una velocidad base de 2.90 GHz.

\*en este punto la concurrencia es deficiente.

**4. CONCLUSIÓN**

Para arreglos pequeños (100 y 1,000 elementos), la implementación secuencial superó a la concurrente. Esto se debe al overhead asociado con la creación y gestión de múltiples hilos, que no se justifica para tareas pequeñas.

Para arreglos medianos y grandes (10,000 elementos en adelante), la implementación concurrente mostró mejoras significativas en el tiempo de ordenamiento. A medida que el tamaño del array aumentó, la ganancia de tiempo y la eficiencia de la implementación concurrente también aumentaron, demostrando la escalabilidad del enfoque concurrente.

La eficiencia se calcula como la razón entre el tiempo secuencial y el tiempo concurrente. Una eficiencia mayor que 1 indica que el algoritmo concurrente es más rápido.