

Herramienta de Profiling Utilizada:

Para medir los tiempos de ejecución de cada algoritmo, utilicé el profiler integrado de Java, apoyándome en la función `System.nanoTime()`. Además, complementé mis mediciones con VisualVM, una herramienta gratuita de monitoreo y análisis de aplicaciones Java, que me permitió confirmar el uso de CPU y detectar posibles cuellos de botella durante la ejecución de los algoritmos.

Metodología:

1. Medición de Tiempos:

Cada algoritmo se ejecutó con arreglos de distintos tamaños (10, 50, 100, 500, 1000, 2000 y 3000 elementos). Antes y después de cada ejecución, se capturaron los tiempos con `System.nanoTime()`, y se calculó la diferencia para obtener el tiempo transcurrido en nanosegundos (ns).

2. Ejecución en Datos Desordenados y Ordenados:

Además de las pruebas con datos aleatorios (desordenados), realicé una prueba adicional con un arreglo ya ordenado (en el caso de 3000 elementos) para observar el comportamiento de cada algoritmo ante datos preordenados.

Resultados Obtenidos:

- **Para arreglos desordenados:**

- **Tamaño: 10**

- Insertion: 8,792 ns
- Merge: 8,333 ns
- Quick: 5,416 ns
- Radix: 1,779,292 ns (*nota: Radix presentó problemas en algunos casos*)
- Bucket: 532,958 ns (*nota: Bucket también mostró problemas en algunos casos*)
- Heap: 8,333 ns

- **Tamaño: 50**

- Insertion: 58,292 ns
- Merge: 33,250 ns
- Quick: 32,125 ns
- Radix: 32,958 ns
- Bucket: 75,583 ns
- Heap: 52,833 ns

- **Tamaño: 100**
 - Insertion: 218,500 ns
 - Merge: 74,333 ns
 - Quick: 66,417 ns
 - Radix: 54,459 ns
 - Bucket: 166,084 ns
 - Heap: 67,833 ns
- **Tamaño: 500**
 - Insertion: 2,544,208 ns
 - Merge: 281,208 ns
 - Quick: 176,709 ns
 - Radix: 187,291 ns
 - Bucket: 599,958 ns
 - Heap: 106,167 ns
- **Tamaño: 1000**
 - Insertion: 2,199,792 ns
 - Merge: 181,750 ns
 - Quick: 140,667 ns
 - Radix: 252,625 ns
 - Bucket: 842,834 ns
 - Heap: 226,042 ns
- **Tamaño: 2000**
 - Insertion: 46,180,458 ns
 - Merge: 1,223,667 ns
 - Quick: 274,666 ns
 - Radix: 533,792 ns
 - Bucket: 1,104,375 ns

- Heap: 3,518,666 ns
- **Tamaño: 3000**
- Insertion: 6,023,125 ns
- Merge: 2,330,541 ns
- Quicksort: 441,000 ns
- Radix: 751,500 ns
- Bucket: 935,208 ns
- Heap: 3,596,959 ns
- **Para un arreglo ya ordenado (Tamaño: 3000):**
- Insertion: 125,916 ns
- Merge: 1,477,500 ns
- Quicksort: 79,346,583 ns (*se observó un comportamiento anómalo en Quicksort, probablemente debido a la selección del pivote en datos ordenados*)
- Radix: 745,625 ns
- Bucket: 874,541 ns
- Heap: 4,791,041 ns

Conclusiones:

- Los algoritmos con complejidad $O(n \log n)$ (Merge, Quicksort y Heap) muestran un rendimiento significativamente mejor que Insertion sort para tamaños mayores, tal como se esperaba teóricamente.
- La medición de Quicksort en datos ya ordenados reveló un comportamiento inusual (tiempo excesivamente alto), lo que indica que la estrategia de elección del pivote en esa implementación no es la óptima para casos ordenados, un aspecto a tener en cuenta.
- Los algoritmos Radix y Bucket tuvieron resultados inconsistentes en algunos casos, lo que sugiere que su implementación podría requerir ajustes para funcionar de manera robusta con distintos conjuntos de datos.

Estos resultados me permitieron confirmar que la complejidad teórica se refleja en la práctica, y también identificar áreas de mejora en las implementaciones de ciertos algoritmos. VisualVM me ayudó a corroborar que el uso de CPU y memoria se mantenía dentro de parámetros aceptables durante las pruebas, lo que respalda la validez de las mediciones obtenidas.

