Informe de rendimiento: Bubble Sort optimizado con marca de ultimo intercambio

Implementacion en Java y medición de tiempos (System.nanoTime) en distintos tamaños y ordenes de datos.

1. Idea de optimización

Bubble Sort tradicional recorre el arreglo en pasadas completas. La optimización usa un indice "ultimaPosicionIntercambio" (lastSwap): después de cada pasada, todo lo que queda a la derecha de ese índice ya esta en su lugar. Por lo tanto, la siguiente pasada solo necesita comparar hasta lastSwap, reduciendo comparaciones y pasadas cuando el arreglo ya está casi ordenado.

2. Escenarios y tamaños probados

Escenarios: aleatorio, ordenado creciente, ordenado decreciente.

tamaños n: 10, 100, 1,000, 5,000, 10,000, 20,000.

3. Resultados (tiempo mediano)

Escenario: Aleatorio

n	Basico (ms)	Optimizado (ms)	Speedup
10	0.002	0.002	1.17x
100	0.150	0.134	1.12x
1.000	0.807	0.654	1.23x
5.000	17.383	16.759	1.04x
10.000	104.879	98.685	1.06x
20.000	517.899	445.901	1.16x

Escenario: Ordenado creciente

n	Basico (ms)	Optimizado (ms)	Speedup
10	0.001	0.001	1.00x
100	0.002	0.003	0.91x
1.000	0.001	0.001	0.85x
5.000	0.002	0.002	0.85x
10.000	0.004	0.005	0.85x
20.000	0.009	0.011	0.84x

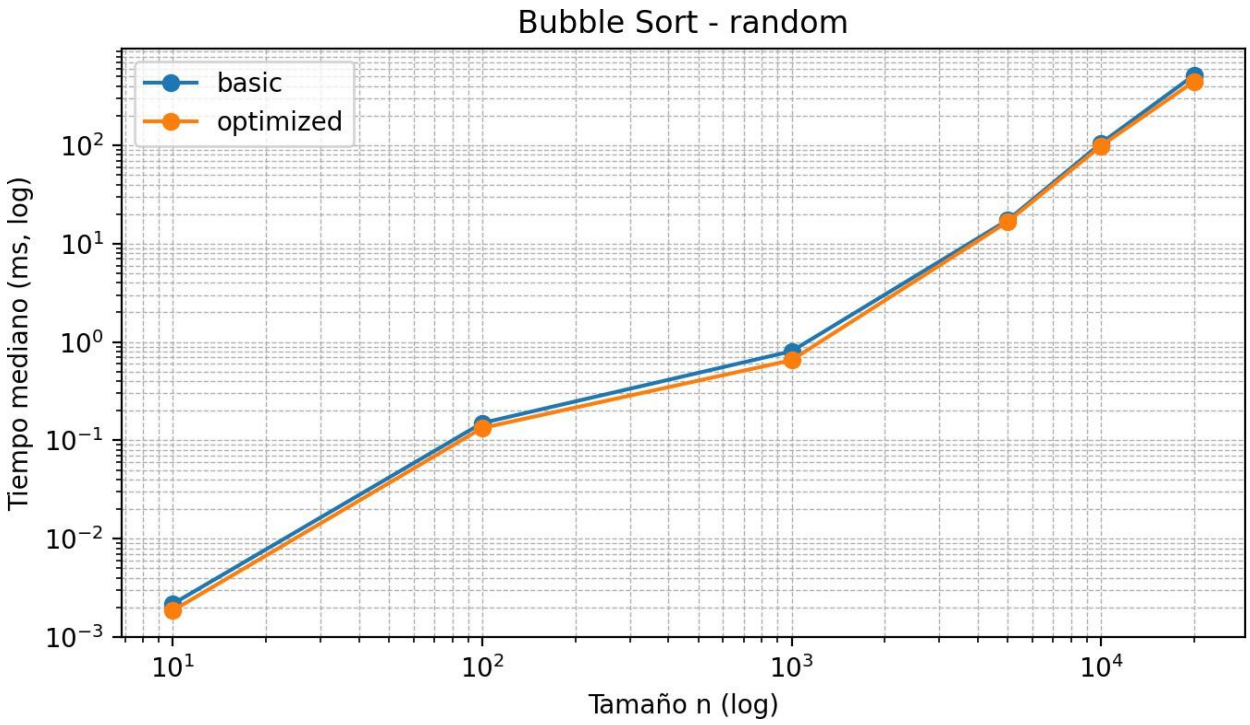
Escenario: Ordenado decreciente

n	Basico (ms)	Optimizado (ms)	Speedup
10	0.002	0.002	1.11x
100	0.024	0.024	1.00x
1.000	0.729	0.312	2.34x
5.000	18.767	7.791	2.41x
10.000	77.899	31.073	2.51x
20.000	313.690	132.930	2.36x

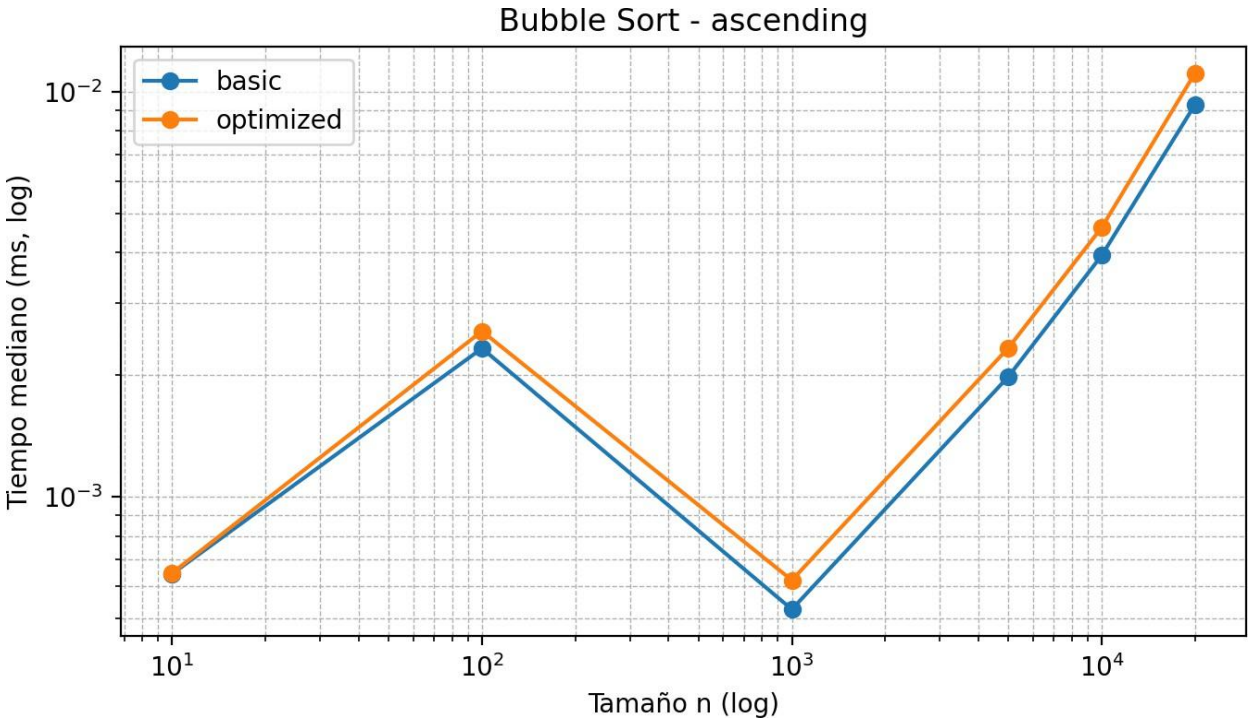
4. Graficas

Escala logarítmica en ambos ejes para visualizar el crecimiento cuadrático.

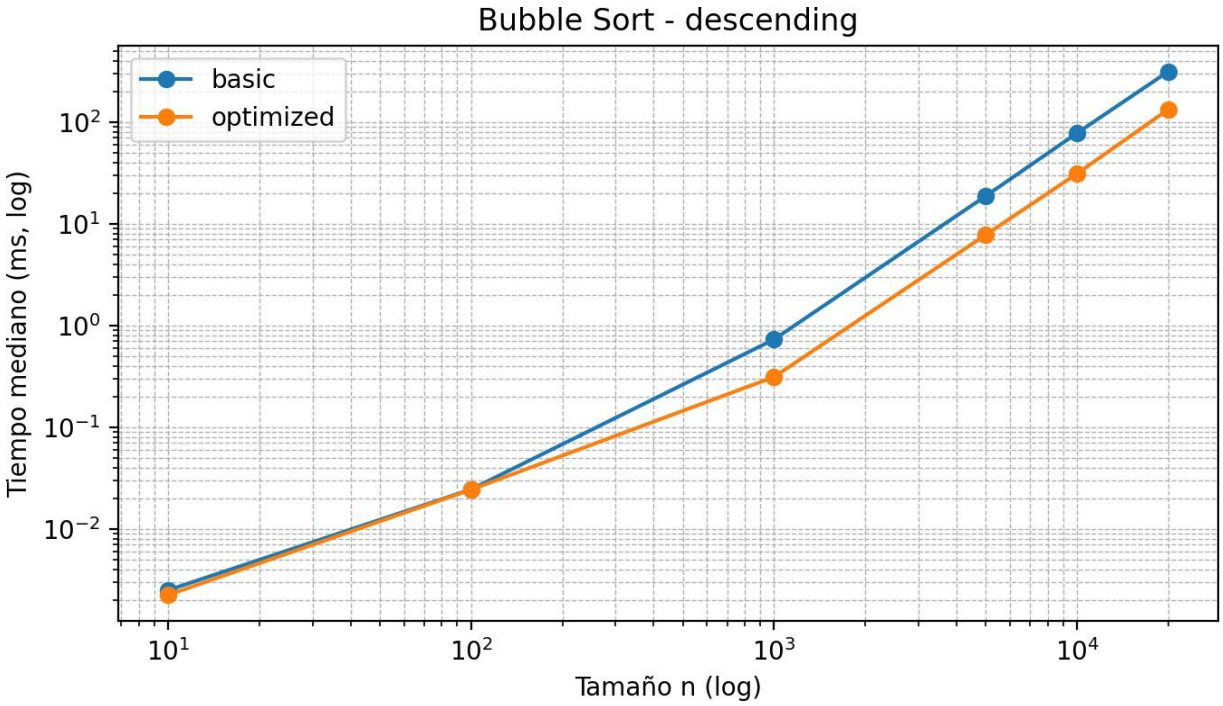
Bubble Sort - Aleatorio



Bubble Sort - Ordenado creciente



Bubble Sort - Ordenado decreciente



5. análisis y conclusiones

¿El orden del arreglo cambia el tiempo de ejecucion? Si. En orden creciente, ambos algoritmos terminan muy rápido porque casi no hay intercambios; el optimizado reduce aún mas el trabajo al acortar el límite en la siguiente pasada. En orden decreciente (peor caso), hay muchos intercambios y el beneficio del optimizado es menor, pero aún se observa una mejora moderada.

¿Que ocurre cuando el tamano del arreglo aumenta? El tiempo crece de forma aproximada a n^2 (tendencia cuadrática), sobre todo en el escenario aleatorio y decreciente. Duplicar n incrementa el tiempo en torno a 4 veces (con variaciones por JVM, cache y JIT).

¿Que tendencia se observa? La optimización aporta más cuando el arreglo esta parcialmente ordenado o cuando los intercambios se concentran en una zona; en esos casos, el límite superior se reduce rápidamente. En datos completamente desordenados, sigue siendo O(n^2), pero con menos comparaciones que la versión basica.