BUBBLESORT VS BUCKETSORT

Juan Esteban Alzate Ospina

Universidad EAFIT

Curso: Datos y Algoritmos 2

Docente: Alexander De Jesús Narváez Berrio

Domingo, 1 de septiembre 2024

A continuación compararemos la efectividad/tiempo de ejecución de los algoritmos BubbleSort y BucketSort, estos tendrán la tarea de organizar un archivo .txt de aproximadamente 240.000 palabras en aumentos de 10.000 en 10.000 para así poder ver su desempeño.

Adjunto tabla de resultados:

Número de datos	Tiempo BubbleSort (ns)	Tiempo BucketSort (ns)	Tiempo en segundos (BubbleSort)	Tiempo en segundos (BucketSort)
10000	500,193,600	7,386,599	0.5002	0.0074
20000	2,295,726,499	6,946,500	2.2957	0.0069
30000	4,994,404,601	6,562,200	4.9944	0.0066
40000	9,410,400,900	5,839,400	9.4104	0.0058
50000	14,274,785,000	4,803,200	14.2748	0.0048
60000	20,666,045,200	5,329,701	20.666	0.0053
70000	28,981,122,100	5,617,000	28.9811	0.0056
80000	37,931,117,900	10,632,401	37.9311	0.0106
90000	47,503,466,300	7,600,900	47.5035	0.0076
100000	59,645,660,099	8,550,400	59.6457	0.0086
110000	74,116,953,001	9,686,100	74.117	0.0097
120000	87,448,256,200	10,368,800	87.4483	0.0104
130000	102,663,206,400	13,404,500	102.6632	0.0134
140000	119,999,638,700	13,357,000	120	0.0134
150000	614,507,914,800	15,478,200	614.5079	0.0155
160000	157,752,066,700	20,036,100	157.7521	0.02
170000	176,624,738,600	21,640,400	176.6247	0.0216
180000	199,376,595,300	23,858,700	199.3766	0.0239
190000	664,669,108,600	24,498,900	664.6691	0.0245
200000	246,939,872,800	25,336,400	246.9399	0.0253
210000	274,308,955,400	24,731,200	274.309	0.0247
220000	301,649,682,000	28,123,200	301.6497	0.0281
230000	331,974,228,700	25,015,900	331.9742	0.025
240000	378,266,348,100	33,232,900	378.2663	0.0332
247047	426,625,984,900	37,804,800	426.626	0.0378

Estos algoritmos fueron ejecutados en mi maquina personal un HP-PavilionGaming 16 en la IDE Intellij con las siguientes especificaciones:

Procesador (CPU):

AMD Ryzen 5 4600H:

6 núcleos y 12 hilos.

Frecuencia base de 3.0 GHz, hasta 4.0 GHz en modo boost.

Caché L3 de 8 MB.

Tarjeta Gráfica (GPU):

NVIDIA GeForce GTX 1650Ti (4GB GDDR6)

Memoria RAM:

16GB DDR4-3200hz

Almacenamiento:

SSD PCIe NVMe de 512GB

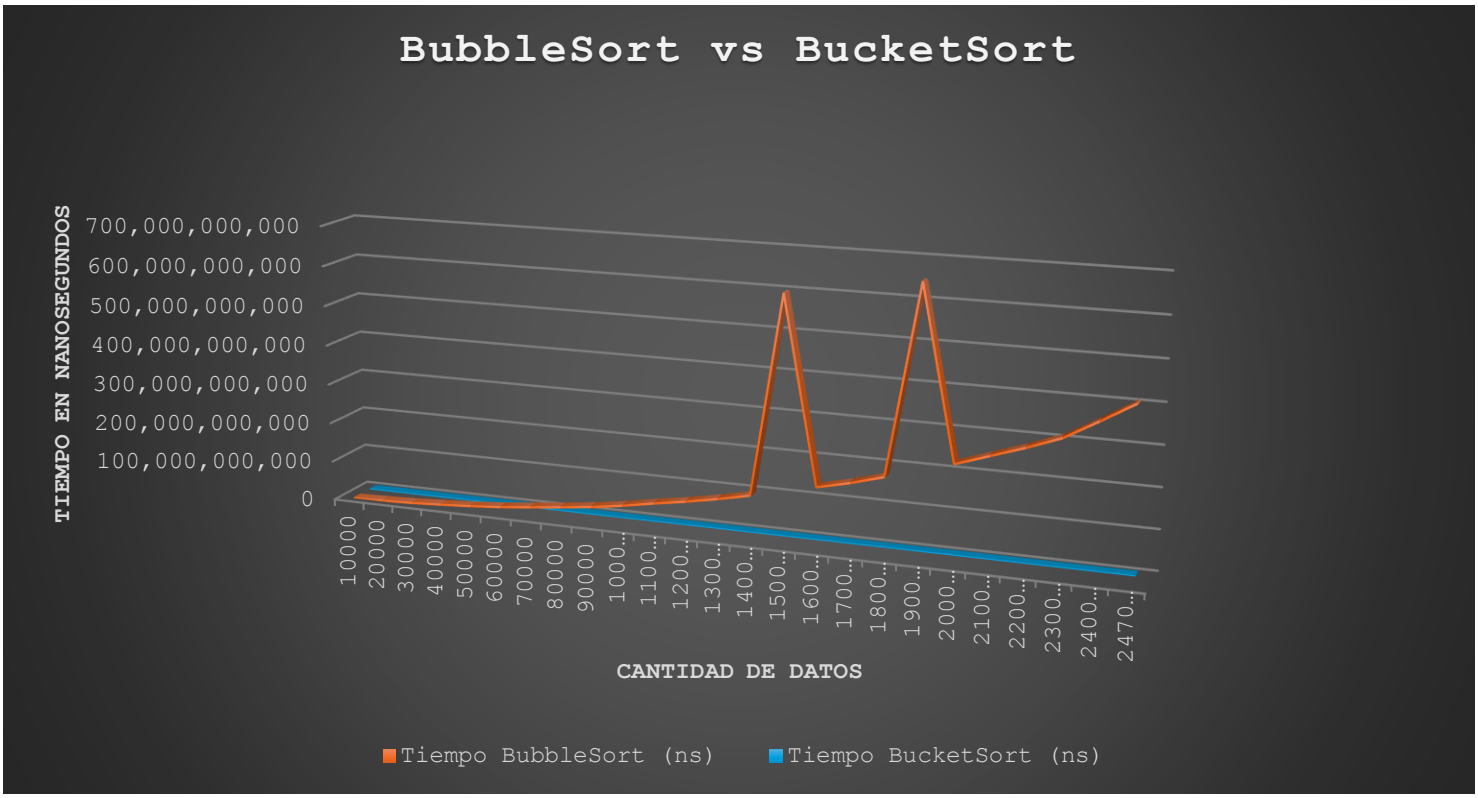
Batería:

Batería de iones de litio de 3 celdas, 52.5 Wh.

Sistema Operativo:

Windows 11.

Ahora veremos los resultados de la prueba de forma grafica para evidenciara de mejor manera la gran diferencia entre estos 2 algoritmos.



Esta diferencia se debe principalmente a su complejidad algorítmica:

BubbleSort (Complejidad Temporal O(n2)**)**:

Es un algoritmo de ordenamiento simple que compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que la lista esté ordenada. Debido a que realiza una gran cantidad de comparaciones e intercambios, su rendimiento se degrada considerablemente cuando se trabaja con grandes volúmenes de datos.

Impacto en la Gráfica: Como se observa en la gráfica, a medida que aumenta la cantidad de datos, el tiempo de ejecución de BubbleSort crece de manera cuadrática. Esto resulta en tiempos de ejecución extremadamente largos cuando se trabaja con conjuntos de datos grandes.

BucketSort (Complejidad Temporal O(n+k) **)**:

Es un algoritmo de ordenamiento que distribuye los elementos de la lista en un número finito de cubetas o "buckets". Luego, cada cubeta se ordena individualmente, ya sea utilizando un algoritmo de ordenamiento diferente o recursivamente usando BucketSort. Finalmente, se concatenan los elementos de todas las cubetas para obtener la lista ordenada. BucketSort es más eficiente cuando los datos están uniformemente distribuidos y se utiliza un número adecuado de cubetas.

Impacto en la Gráfica: BucketSort muestra tiempos de ejecución significativamente más bajos en comparación con BubbleSort, especialmente a medida que el tamaño de los datos aumenta. Esto se debe a su complejidad lineal en el mejor de los casos. Sin embargo, es posible que veas algunas fluctuaciones en la gráfica, posiblemente debido a la variabilidad en cómo se distribuyen los datos en los buckets y la eficiencia del algoritmo de ordenamiento secundario usado dentro de cada cubeta.

Conclusiones

La razón principal por la cual BubbleSort es mucho más lento que BucketSort es porque hace muchas más operaciones a medida que aumenta el tamaño de la lista. El número de comparaciones aumenta rápidamente con n debido a su complejidad cuadrática, lo que aumenta el tiempo de ejecución para listas grandes.

Por el contrario, BucketSort maneja mucho mejor el aumento del tamaño de los datos, especialmente si los datos están distribuidos uniformemente y hay un número adecuado de cubetas. Como se puede ver en la gráfica, los tiempos de BubbleSort aumentan significativamente, mientras que los tiempos de BucketSort permanecen relativamente bajos y constantes.

Como resultado, la diferencia significativa principalmente y como hemos venido mencionando, debido a la gran cantidad de operaciones que debe hacer una respecto a la otra por numero de datos y su naturaleza de complejidad respectivamente.