

COMPARACIÓN ENTRE LOS ALGORITMOS DE ORDENAMIENTO INTERCAMBIO DIRECTO BIDIRECCIONAL Y COUNTING SORT EN TÉRMINOS DE EFICIENCIA

COMPARISON BETWEEN BIDIRECTIONAL DIRECT EXCHANGE AND COUNTING SORT SORTING ALGORITHMS IN TERMS OF EFFICIENCY

Russbell Juan Pablo Arratia Paz

Escuela de Ingeniería en Informática y Sistemas
Universidad Nacional Jorge Basadre Grohmann
Tacna, Perú

<https://orcid.org/0009-0006-5163-7863>

rjparratiap@unjbg.edu.pe

Alexander Efrain Contreras Rodriguez

Escuela de Ingeniería en Informática y Sistemas
Universidad Nacional Jorge Basadre Grohmann
Tacna, Perú

<https://orcid.org/0009-0000-3931-6684>

acontrerasr@unjbg.edu.pe

Danitza Carmen Capía Quiñonez

Escuela de Ingeniería en Informática y Sistemas
Universidad Nacional Jorge Basadre Grohmann
Tacna, Perú

<https://orcid.org/0009-0006-1299-2520>

dccapiaq@unjbg.edu.pe

Resumen

Se realizará una comparación entre los algoritmos de Intercambio Directo Bidireccional y Counting Sort, con el propósito de evaluar su eficiencia en la ordenación de datos numéricos. Para ello, se aplicarán pruebas experimentales que permitan analizar su rendimiento bajo distintas condiciones, considerando criterios como tiempo de ejecución, complejidad teórica y uso de memoria. Los resultados obtenidos permitirán determinar en qué contextos cada algoritmo presenta un mejor desempeño y cuál resulta más adecuado según las características de los datos y las necesidades de implementación.

Palabras claves: algoritmos, eficiencia, complejidad, eficiencia, notación O.

Abstract

A comparison will be made between the Bidirectional Direct Exchange and the Counting Sort algorithms in order to evaluate their efficiency in sorting numerical data. Experimental tests will be conducted to analyze their performance under different conditions, considering criteria such as execution time, theoretical complexity, and memory usage. The obtained results will allow determining in which contexts each algorithm shows better performance and which one is more suitable depending on the characteristics of the data and the implementation requirements.

Keywords: algorithms, efficiency, complexity, performance, Big O notation.

1. Introducción

Un algoritmo de ordenamiento es una clásica aplicación en computación, tanto aplicada como estudiada. Tiene de entrada elementos en secuencia donde se tiene una clave asignada a cada uno, la cual se permite comparar y ordenar (Arráiz et al. 1996) Cuando hablamos de algoritmos de ordenamiento existen una gran variedad, con diferentes eficiencias y estructuras en código, por lo que resulta sensato buscar algoritmos que se adecuen a las necesidades requeridas, entonces entra el coste computacional para analizar rendimientos.

Se entiende por eficiencia de un algoritmo a los recursos de cómputo requerido para su ejecución, este concepto de eficiencia permite comparar resolviendo el mismo problema utilizando distintos algoritmos (Duch, 2007)

El objetivo del presente artículo científico es comparar los algoritmos de ordenamiento Counting Sort y Selección Directa Bidimensional y medir su rendimiento en distintos escenarios propuestos tanto en cantidad de datos como el rango y orden inicial, usando al tiempo como indicador de referencia.

Counting Sort

Counting sort es un algoritmo de ordenamiento no comparativo que ordena elementos contando la frecuencia de cada uno. Para ordenar, crea un arreglo auxiliar para almacenar el recuento de cada elemento, luego utiliza este recuento para determinar la posición final de cada elemento en el arreglo de salida.

Bajpai y Kots, 2014, nos dicen que el algoritmo de ordenación counting sort es eficiente y asume que todos los elementos a ordenar son de tipo entero en el rango de 1 a k , donde k es otro entero.

La idea básica del ordenamiento es determinar, para cada elemento, el número de elementos. Con esto se puede colocar los elementos directamente en su posición correcta.

Intercambio Directo Bidireccional

El algoritmo de intercambio bidireccional, también llamado burbuja bidireccional o shaking sort, es una mejora al algoritmo de burbuja que intercambia elementos uno al lado del otro este algo va en ambos sentidos, primero mueve las cosas grandes hacia abajo de izquierda a derecha, luego empuja las pequeñas hacia arriba de derecha a izquierda y se detiene cuando un pase completo en ambos sentidos no intercambia nada, lo que significa que la lista está toda ordenada

Incerpi y Sedgewick, 1987, dicen que el shaker sort incrementa las secuencias para cada paso que realice, además que hace movimientos innecesarios solo movimiento máximos y mínimos.

2. Metodología

Para la comparación se eligieron los algoritmos Selección Directa Bidimensional y Counting Sort, ambos pertenecientes a la categoría de ordenamientos, pero con enfoques distintos: el primero está basado en comparaciones, y el segundo en conteo de frecuencias. Para garantizar la consistencia de los resultados obtenidos, las pruebas experimentales se realizaron bajo un entorno controlado. Todas las ejecuciones se efectuaron en el mismo equipo y utilizando la misma configuración de hardware y software para evitar las variaciones externas. A continuación se facilitarán algunos componentes:

- CPU: Intel Ryzen 5 5600G
- Memoria Ram: 16 GB
- Frecuencia base: 3.90 GHz
- Núcleos: 6 (se usará solo 1 para las pruebas)
- Hilos: 12 procesadores lógicos
- Sistema operativo: Windows 10 Pro 64 bits
- Compilador: Dev c++ Version 6.3

Para mayor detalle véase las figuras (1,2,3 y 4).

Se considera la complejidad teórica de ambos algoritmos y consumo de memoria: Counting sort, siendo $O(n + k)$ de carácter lineal pero su rendimiento se ve afectado al tener un rango de valores mayor al número de elementos ($k > n$, ocupando una memoria proporcional a $O(n + k)$). Intercambio bidireccional, siendo $O(n^2)$ en el peor y caso promedio, y teniendo carácter lineal en el mejor de los casos, se caracteriza por su bajo consumo de memoria $O(1)$ ya que realiza los cambios sobre el arreglo original. Se usarán datos aleatorios pero probándose el mismo dato en cada repetición por algoritmo, siendo un total de tres tamaños (1000, 10000, 100000) realizándose 50 repeticiones en cada uno pero siendo 10 repeticiones por cada patrón de entrada. Se calcularon los tiempos en microsegundos (ms) haciendo uso de la función `std::chrono::steady_clock`. de la librería *chrono*. Se usará un contador dentro del código para contar el número de intercambios y comparaciones.

3. Resultados

- **Counting Sort**

Nótese que counting no realiza comparaciones ni intercambios, además, el peor de los casos de counting sort es cuando k es mayor que n , por lo tanto los tiempos no se disparan.

- **Muestra de 1,000**

El algoritmo counting tiene carácter lineal en el mejor, peor y casos promedios. Pero teniendo un efecto casi inmediato mientras “ n ” es menor. Por lo cual el tiempo promedio con arreglos de 1000 datos con las condiciones de aleatorios, ascendentes, descendentes, casi ordenados con repeticiones es 0 ms. Como se ve en la *Tabla 1*.

- **Muestra de 10,000**

Counting sort es lineal en un tiempo temporal, por ello con arreglos de tamaño 10000 aún no se notan grandes cantidad de tiempo. Por lo cual el tiempo promedio con arreglos de 1000 datos con las condiciones de aleatorios, ascendentes, descendentes, casi ordenados con repeticiones es 0.11956 ms. Como se ve en la *Tabla 2*.

- **Muestra de 100,000**

Al medir un tamaño de arreglo similar al rango de valores no se notan grandes cantidades de tiempo, sin embargo al ser n igual a 100000 el tiempo si aumento siendo algo notorio. Por lo cual el tiempo promedio con arreglos de 100000 datos con las condiciones de aleatorios, ascendentes, descendentes, casi ordenados con repeticiones es 0.87 ms. Como se ve en la *Tabla 3*.

- **Muestra con $K > N$**

Se probó un arreglo aleatorio con $K=200$ N y $n=10\ 000\ 000$ para notar el aumento de tiempo. siendo el tiempo 6.94E equivalente a 69 segundos, notando también el aumento considerable de memoria, aumentando hasta en 7GB más de consumo. Como se ve en la *Tabla 4*.

- **Intercambio Directo Bidireccional**

Observaciones:

- **Tiempos de ejecución**

El tiempo aumenta de forma no lineal con n : al multiplicar n por 10, el tiempo se multiplica aproximadamente por 100–150.

Esto confirma el comportamiento cuadrático ($O(n^2)$).

En arreglos ascendentes, el tiempo es casi cero porque el algoritmo detecta que el arreglo ya está ordenado y termina rápidamente.

En arreglos descendentes, el tiempo es máximo, ya que se realizan la mayor cantidad de pasadas posibles.

En casi ordenados, el tiempo es bajo, mostrando que el algoritmo puede reducir iteraciones cuando el desorden es leve.

Los arreglos con duplicados tienen tiempos similares a los aleatorios, porque los intercambios siguen siendo numerosos.

Véase las tablas 5, 8 y 11

- **Intercambios**

Los intercambios también crecen cuadráticamente, ya que en el peor caso el algoritmo compara e intercambia cada elemento múltiples veces.

En arreglos ascendentes, no hay intercambios (ya están ordenados). En descendentes, el número de intercambios es máximo, igual al de comparaciones.

En casi ordenados, los intercambios bajan mucho, mostrando que el algoritmo evita movimientos innecesarios si el arreglo ya está casi en orden.

En duplicados, el comportamiento se asemeja al aleatorio, pero ligeramente menor por repeticiones de valores.

Véase las tablas 6, 9 y 12

- **Comparaciones**

El número de comparaciones crece aproximadamente con n^2 , confirmando la complejidad teórica.

De $n = 1000$ / $n=10000$ / $n=100000$, las comparaciones se multiplican por ≈ 100 .

En arreglos ascendentes, las comparaciones son lineales ($O(n)$) porque el algoritmo reconoce que ya no necesita más pasadas.

En arreglos descendentes, el valor es cercano al máximo posible ($(n(n-1)/2)$), lo que representa el peor caso.

En casi ordenados, hay una reducción significativa de comparaciones, mostrando eficiencia parcial cuando el desorden es bajo.

Véase las tablas 7, 10 y 13

4. Discusión

- Costo computacional

-Teórico

Teóricamente, Counting Sort tiene una complejidad temporal $O(n + k)$, donde k es el rango de los valores del arreglo. Esto significa que su tiempo de ejecución crece de manera casi lineal con respecto al tamaño de los datos, siempre que el rango no sea excesivamente grande. Además, su complejidad espacial también es $O(n + k)$, ya que necesita arreglos adicionales para contar y almacenar los resultados. En cambio, la Selección Directa Bidireccional (o Shaker Sort) posee una complejidad temporal $O(n^2)$, lo que implica que el número de comparaciones e intercambios crece cuadráticamente al aumentar n , siendo mucho menos eficiente para tamaños grandes.

-Empírico

Los resultados experimentales confirman este comportamiento. Para $n = 1,000$, Counting Sort prácticamente no presenta tiempo significativo de ejecución, mientras que la Selección Directa Bidireccional alcanza un promedio de 0.87782 microsegundos. Cuando el tamaño del arreglo aumenta a $n = 10,000$, el tiempo de Counting Sort se mantiene muy bajo (0.11956 μs), mientras que el Bidireccional incrementa a 89.4851 μs . Finalmente, con $n = 100,000$, Counting Sort solo requiere 0.73778 μs en promedio, frente a los 10,551.0165 μs del método bidireccional. Esto demuestra que el crecimiento del costo temporal en Counting Sort es casi lineal, mientras que en la Selección Directa Bidireccional es exponencial.

- Comparación de resultados

Para $n=1000$:

Para tamaños pequeños, ambos algoritmos terminan muy rápido.

Counting Sort es inmediato ($\approx 0 \mu s$), mientras que Shaker Sort ya muestra valores perceptibles.

Diferencia leve, pero Counting Sort es más constante en cualquier tipo de arreglo.

Bidireccional mejora en arreglos ordenados ($0 \mu s$), lo que indica detección temprana de orden.

Para $n=10000$:

El tiempo de Counting Sort se mantiene casi constante y extremadamente bajo ($\approx 0.1 \mu s$).

En cambio, Shaker Sort se vuelve mucho más lento (hasta $193 \mu s$ en descendente).

La diferencia promedio es $\approx 89 \mu s$, lo que refleja un aumento de ~ 1000 veces más tiempo.

Nuevamente, en ascendente se mantiene rápido, confirmando que Shaker puede optimizar casos ya ordenados.

Para $n=100000$:

La diferencia ahora es abismal: el Bidireccional tarda más de 10,000 veces más que Counting Sort.

Mientras Counting Sort sigue en tiempo casi constante ($< 1 \mu s$), el Bidireccional escala hasta decenas de miles de microsegundos.

El crecimiento cuadrático del Bidireccional se hace evidente, frente al crecimiento lineal de Counting Sort.

Análisis general:

Counting Sort crece suavemente con n : el aumento es proporcional (lineal).

Intercambio Bidireccional crece exponencialmente: cuando n se multiplica $\times 10$, el tiempo se multiplica $\times 100$ o más.

La diferencia promedio pasa de menos de $1 \mu s$ ($n=1000$) a más de $10 \text{ mil } \mu s$ ($n=100000$).

Esto confirma que Counting Sort es muchísimo más eficiente para volúmenes grandes, especialmente cuando el rango es acotado.

5. Conclusiones

En síntesis, el algoritmo Counting Sort al ser temporalmente lineal, tiene mayor eficacia respecto al algoritmo ShakerSort, ya que este último es de costo computacional cuadrática. Aunque en cuanto a memoria el ShakerSort obtiene ventaja al realizar el ordenamiento en el mismo arreglo.

En conclusión, el costo computacional de Counting Sort es considerablemente más bajo que el de la Selección Directa Bidireccional. Counting Sort mantiene un comportamiento estable y eficiente incluso con grandes volúmenes de datos, mientras que la Selección Directa Bidireccional presenta un aumento drástico del tiempo de ejecución conforme crece n . Por lo tanto, desde el punto de vista del costo computacional, Counting Sort es mucho más eficiente y escalable, aunque requiere un poco más de memoria auxiliar.

6. Tablas y Figuras

Figura 1
Información detallada sobre el sistema, hardware y sistema operativo

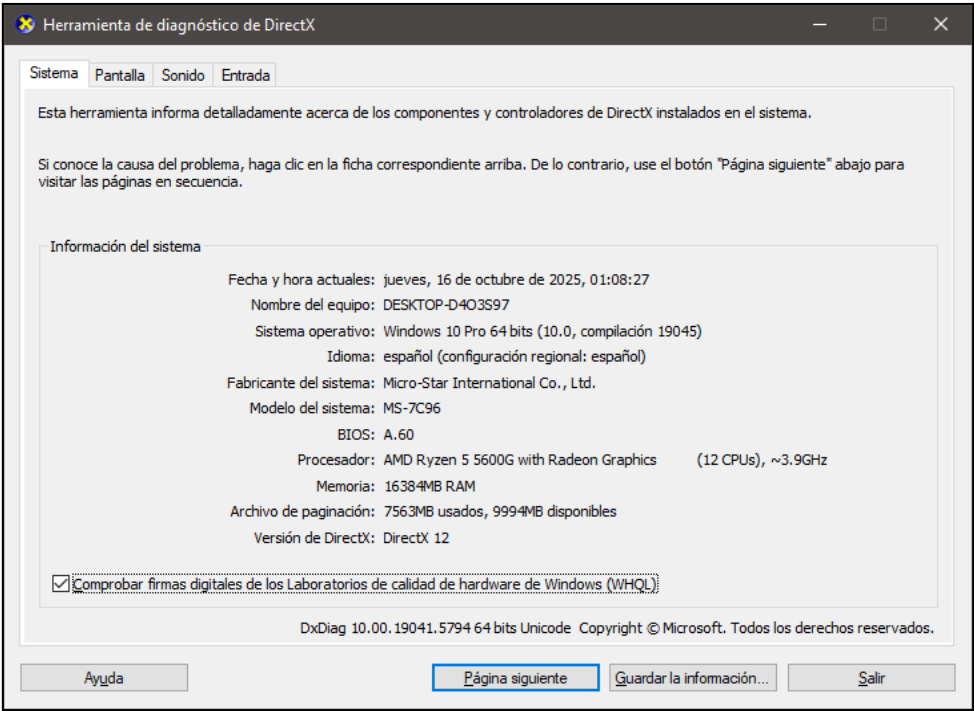


Figura 2
Información detallada sobre la unidad de procesamiento gráfico (GPU)

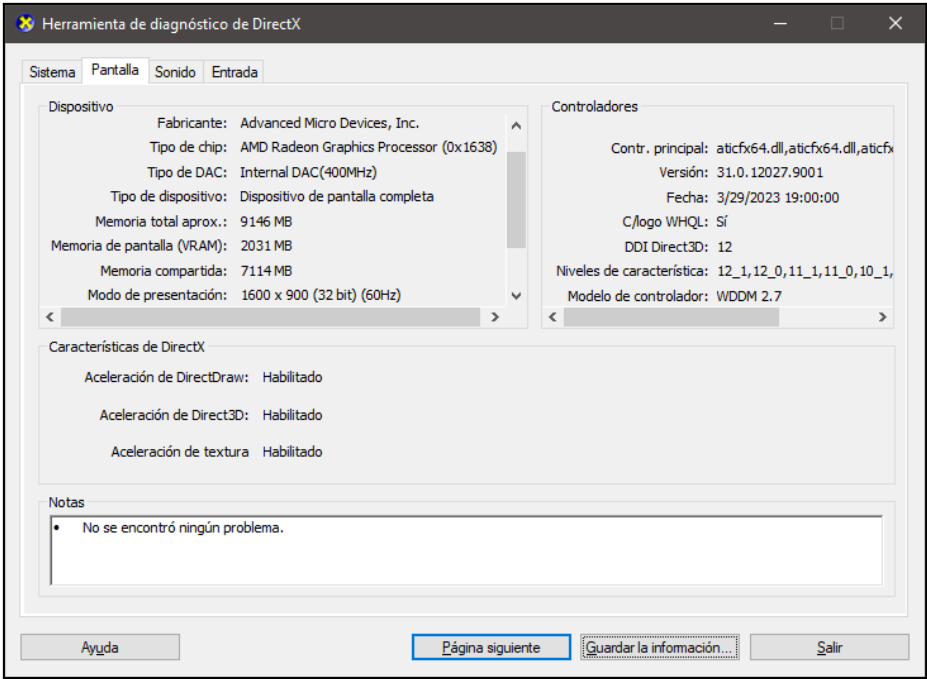


Figura 3

Información del rendimiento Unidad Central de Procesamiento (CPU) limitado a 1 núcleo

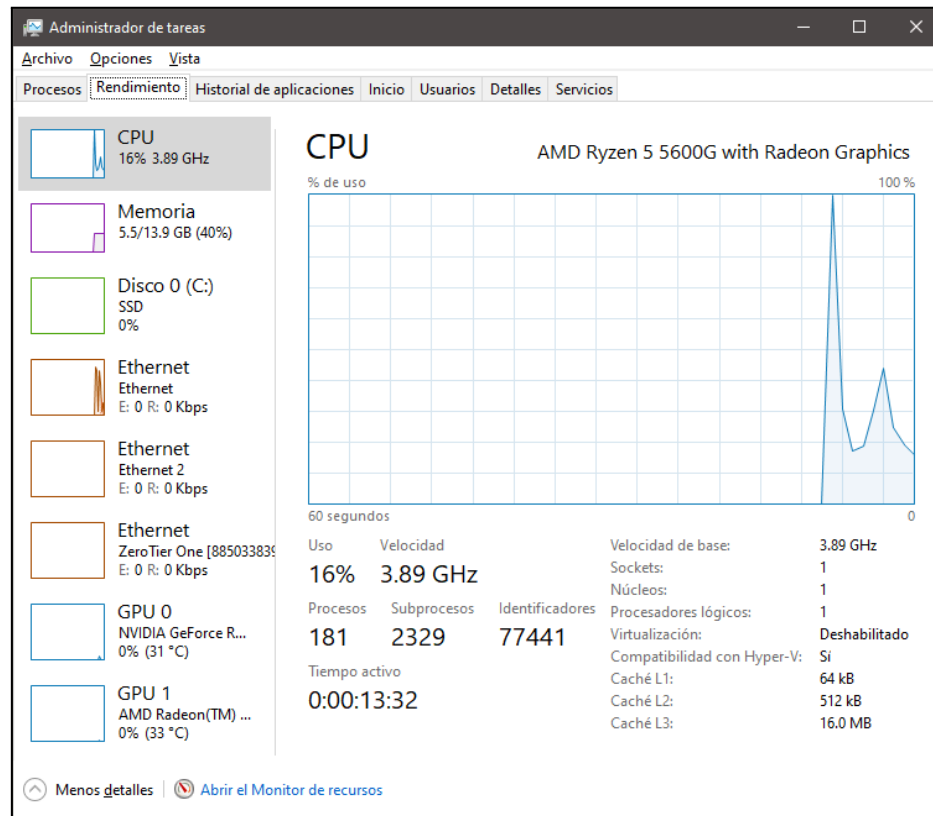


Figura 5

Información del compilador DEV C++ Versión 6.3



Tabla 1

Resultados de eficiencia del algoritmo Counting Sort cuando $n=1000$ y el rango=1000

Característica de datos	Tiempo en microsegundos										Tiempo
n=1000, rango=1000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	0	0	0	0	0	0	0	0	0	0	0
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	0
Arreglo descendente	0	0	0	0	0	0	0	0	0	0	0
Arreglo casi ordenado	0	0	0	0	0	0	0	0	0	0	0
Arreglo con duplicados	0	0	0	0	0	0	0	0	0	0	0
											0

Tabla 2

Resultados de eficiencia del algoritmo Counting Sort cuando $n=10000$ y el rango=10000

Característica de datos	Tiempo en microsegundos										Tiempo
n=10000, rango=10000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	0.000997	0	0.000997	0	0	0	0.000994	0.000998	0.001029	0	0.0005015
Arreglo ascendente	0.000998	0	0	0.000996	0	0	0	0	0.000997	0	0.0002991
Arreglo descendente	0.000999	0	0	0	0	0	0	0	0	0	0.0000999
Arreglo casi ordenado	0.000997	0.000965	0	0.000997	0	0	0.000964	0	0	0	0.0003923
Arreglo con duplicados	0	0.000997	0.000997	0	0	0	0	0.000998	0.001001	0.000998	0.0004991
											0.00035838

Tabla 3

Resultados de eficiencia del algoritmo Counting Sort cuando $n=100000$ y el rango=100000

Característica de datos	Tiempo en microsegundos										Tiempo
n=100000, rango=10000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	0.002992	0.002959	0.001995	0.001995	0.002993	0.002993	0.001992	0.002958	0.002992	0.002991	0.002686
Arreglo ascendente	0.002026	0.001962	0.002027	0.001964	0.001995	0.000997	0.001994	0.001994	0.001994	0.001995	0.0018948
Arreglo descendente	0.001997	0.001993	0.000997	0.001997	0.002038	0.001955	0.000996	0.001996	0.001996	0.000997	0.0016962
Arreglo casi ordenado	0.001995	0.001994	0.001994	0.002994	0.001996	0.001995	0.000996	0.001993	0.001993	0.001995	0.0019945
Arreglo con duplicados	0.001994	0.00399	0.002993	0.002992	0.002991	0.013963	0.006981	0.002993	0.003989	0.000998	0.0043884
											0.00253198

Tabla 4

Resultados de eficiencia del algoritmo Counting Sort cuando $2k > n$

Característica de dato	Tiempo en microsegundos					Promedio
$n=10000000$, rango= $200n$	1	2	3	4	5	
Arreglo $k > n$	6.9396	5.75858	5.79799	5.90908	5.79804	6.040658

Tabla 5

Resultados de eficiencia en tiempo del algoritmo Shaking Sort cuando $n=1000$ y el rango= 1000

Característica de datos	Tiempo en microsegundos										Tiempo Promedio
$n=1000$, rango= 1000	1	2	3	4	5	6	7	8	9	10	
Arreglo aleatorio	0.002994	0.00399	0.004019	0.002962	0.002991	0.002993	0.002994	0.002999	0.002992	0.00302	0.0031945
Arreglo ascendente	0	0	0	0	0.000997	0	0	0	0	0	0.0000997
Arreglo descendente	0.004986	0.004987	0.005985	0.004021	0.004988	0.003956	0.005984	0.004024	0.004984	0.005953	0.0049868
Arreglo casi ordenado	0	0.000998	0	0	0.000997	0	0.000997	0	0	0	0.0002992
Arreglo con duplicados	0.003959	0.003989	0.004021	0.002964	0.003034	0.003948	0.002992	0.003022	0.002993	0.002997	0.0033915
											0.00239434

Tabla 6

Resultados de comparaciones del algoritmo Shaking Sort cuando $n=1000$ y el rango= 1000

Característica de datos	Comparaciones										Tiempo
$n=10000$, rango= 10000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	366630	398475	376740	367659	372744	395760	380184	386925	382130	372240	379948.7
Arreglo ascendente	999	999	999	999	999	999	999	999	999	999	999
Arreglo descendente	499500	499500	499500	499500	499500	499500	499500	499500	499500	499500	499500
Arreglo casi ordenado	51569	49674	51569	53460	45872	57230	53460	60984	53460	51569	52884.7
Arreglo con duplicados	389285	382130	375249	372240	368172	381645	377729	392084	375747	385019	379930
											262652.48

Tabla 7

Resultados de eficiencia en intercambios del algoritmo Shaking Sort cuando $n=1000$ y el rango= 1000

Característica de datos	Intercambios										Tiempo
n=10000, rango=10000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	240277	263063	253527	248745	245158	260792	243857	257054	249554	247024	250905.1
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	0
Arreglo descendente	499500	499500	499500	499500	499500	499500	499500	499500	499500	499500	499500
Arreglo casi ordenado	31476	29526	30846	30540	32070	34324	33142	31970	37304	32022	32322
Arreglo con duplicados	254325	251495	248171	243108	242900	247083	248035	253012	242938	249925	248099.2
											206165.26

Tabla 8

Resultados de eficiencia en tiempo del algoritmo Shaking Sort cuando n=10000 y el rango=10000

Característica de datos	Tiempo en microsegundos										Tiempo
n=10000, rango=10000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	0.342085	0.348034	0.373632	0.345077	0.350975	0.34912	0.352257	0.348791	0.342053	0.347071	0.3499095
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	0
Arreglo descendente	0.697932	0.567514	0.522571	0.511665	0.529618	0.516959	0.522836	0.505197	0.500505	0.482213	0.535701
Arreglo casi ordenado	0.036899	0.036934	0.034954	0.03885	0.041889	0.034913	0.036933	0.036901	0.039403	0.037861	0.0375537
Arreglo con duplicados	0.385505	0.386997	0.375963	0.349619	0.372542	0.354053	0.368016	0.352348	0.365786	0.354053	0.3664882
											0.25793048

Tabla 9

Resultados de eficiencia en comparaciones del algoritmo Shaking Sort cuando n=10000 y el rango=10000

Característica de datos	Comparaciones										Comparaciones
n=10000, rango=10000	1	2	3	4	5	6	7	8	9	10	Promedio
Arreglo aleatorio	37316870	37572380	38080479	37741275	37507497	37562409	37397310	37696680	37392290	37864725	37613191.5
Arreglo ascendente	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
Arreglo descendente	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000
Arreglo casi ordenado	4789214	4789214	4751172	4560722	5206620	4675040	4675040	4789214	4789214	4998159	4802360.9
Arreglo con duplicados	37190670	37607247	37542455	37205847	37532472	37362149	38361924	37417380	38158455	37437434	37581603.3
											26000430.94

Tabla 10

Resultados de eficiencia en intercambios del algoritmo Shaking Sort cuando $n=10000$ y el rango=10000

Característica de datos	Intercambios										Intercambios Promedio
$n=10000$, rango=10000	1	2	3	4	5	6	7	8	9	10	
Arreglo aleatorio	24891714	24976231	25162961	25238639	25068335	24976780	24812254	25111558	24825986	25359575	25042403.3
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	0
Arreglo descendente	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000	49995000
Arreglo casi ordenado	3084448	3153738	3109360	2980887	3353468	3113288	3064142	3147424	3162430	3105408	3127459.3
Arreglo con duplicados	24914476	25066977	25033484	24748945	24777997	24868376	25161482	24986545	25192548	24962824	24971365.4
											20627245.6

Tabla 11

Resultados de eficiencia en tiempo del algoritmo Shaking Sort cuando $n=100000$ y el rango=100000

Característica de datos	Tiempo en microsegundos										Tiempo Promedio
$n=100000$, rango=10000	1	2	3	4	5	6	7	8	9	10	
Arreglo aleatorio	0.0163079	0.0155652	0.0169236	0.0164133	0.0148055	0.0136744	0.0164863	0.015984	0.0157083	0.0143169	0.01561854
Arreglo ascendente	0.0000000948135	0.0000000932433	0.0000000106488	0.00000000980323	0.0000000106359	0.00000000980809	0.0000000107579	0.00000000986246	0.000000010101247	0.000000010102964	0.00000010074316
Arreglo descendente	0.0201738	0.017935	0.0187145	0.0208695	0.0210124	0.0185993	0.0191723	0.0212627	0.02078	0.0193951	0.01979146
Arreglo casi ordenado	0.000424275	0.000427336	0.000380779	0.000393977	0.000372012	0.000431695	0.000373649	0.000348039	0.000415232	0.000412275	0.0003979269
Arreglo con duplicados	0.0155181	0.0172884	0.0148217	0.0150983	0.0165574	0.0163194	0.0178859	0.0170246	0.0159074	0.0160078	0.0162429
											0.010410185528632

Tabla 12

Resultados de eficiencia en comparaciones del algoritmo Shaking Sort cuando $n=100000$ y el rango=100000

Característica de datos	Comparaciones										Comparaciones Promedio
$n=100000$, rango=10000	1	2	3	4	5	6	7	8	9	10	
Arreglo aleatorio	3288810478	3232759590	3239255381	3453284242	3132551774	3351444860	3452009077	3296173271	3154801971	3356939270	3295802991.4
Arreglo ascendente	100459	97187	96513	98966	93403	100994	97727	97906	98606	101279	98304
Arreglo descendente	5183966291	5027225200	5340816136	4962966131	4833314175	4965392485	4680134663	5371465714	4889081192	5519765566	5077412755.3
Arreglo casi ordenado	90074292	91379183	97785939	85601344	84873796	99229542	97185686	92708199	90299922	95239411	92437731.4
Arreglo con duplicados	3453098774	3104869945	3380741700	3272967257	3225421315	2935484086	3365276117	3271884233	3156965290	3160223983	3232693270

2339689010.42

Tabla 13

Resultados de eficiencia en intercambios del algoritmo Shaking Sort cuando $n=100000$ y el rango= 100000

Característica de datos	Intercambios										Intercambios Promedio
n=100000, rango=10000	1	2	3	4	5	6	7	8	9	10	
Arreglo aleatorio	2547704443	2379274885	2409704720	2531623985	2310222624	2653914706	2438355859	2373651538	2431211081	2537767552	2461343139.3
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	0
Arreglo descendente	4861708230	4891054540	5052301618	4881436662	5540674042	4862318664	5197990714	4980718935	4547566926	4543817768	4935958809.9
Arreglo casi ordenado	91369671	90502548	89130260	90429433	89533102	86851422	92575997	91788890	85317367	87231767	89473045.7
Arreglo con duplicados	2522358248	2488305371	2573024330	2326735583	2367736415	2344406491	2559455263	2405617737	2493188399	2510851526	2459167936.3
											1989188586.24

Tabla 14

Comparación de eficiencia en tiempo del algoritmo Shaking Sort cuando $n=1000$ y el rango= 1000

n=1000, rango=1000	Counting sort	Selección Directa Bidireccional	Diferencia
Arreglo aleatorio	0	0.0031945	0.0031945
Arreglo ascendente	0	0.0000997	0.0000997
Arreglo descendente	0	0.0049868	0.0049868
Arreglo casi ordenado	0	0.0002992	0.0002992
Arreglo con duplicados	0	0.0033915	0.0033915
Promedio	0	0.00239434	0.00239434

Tabla 15

Comparación de eficiencia en tiempo del algoritmo Shaking Sort cuando $n=10000$ y el rango= 10000

n=10000, rango=10000	Counting sort	Selección Directa Bidireccional	Diferencia
Arreglo aleatorio	0.0005015	0.3499095	0.349408
Arreglo ascendente	0.0002991	0	0.0002991
Arreglo descendente	0.0000999	0.535701	0.5356011
Arreglo casi ordenado	0.0003923	0.0375537	0.0371614
Arreglo con duplicados	0.0004991	0.3664882	0.3659891
Promedio	0.00035838	0.25793048	0.2575721

Tabla 16

Comparación de eficiencia en tiempo del algoritmo Shaking Sort cuando $n=100000$ y el rango= 100000

n=100000, rango=100000	Counting sort	Selección Directa Bidireccional	Diferencia
Arreglo aleatorio	0.002686	15.61854	15.615854
Arreglo ascendente	0.0018948	0.00010074316	0.00179405684
Arreglo descendente	0.0016962	19.79146	19.7897638
Arreglo casi ordenado	0.0019945	0.3979269	0.3959324
Arreglo con duplicados	0.0043884	16.2429	16.2385116
Promedio	0.00253198	10.410185528632	10.407653548632

Tabla 17

Resumen de comparación de eficiencia en tiempo del algoritmo Shaking Sort en los diferentes tamaños de n y rango

Tamaño de n = rango	Counting sort	Selección Directa Bidireccional
$n = 1,000$	0	0.00239434
$n = 10,000$	0.00035838	0.25793048
$n = 100,000$	0.00035838	10.410185528632

Gráfico 1

Comparación de eficiencia en microsegundos de Counting Sort - Shaker Sort cuando $n = 1000$ y el rango = 1000

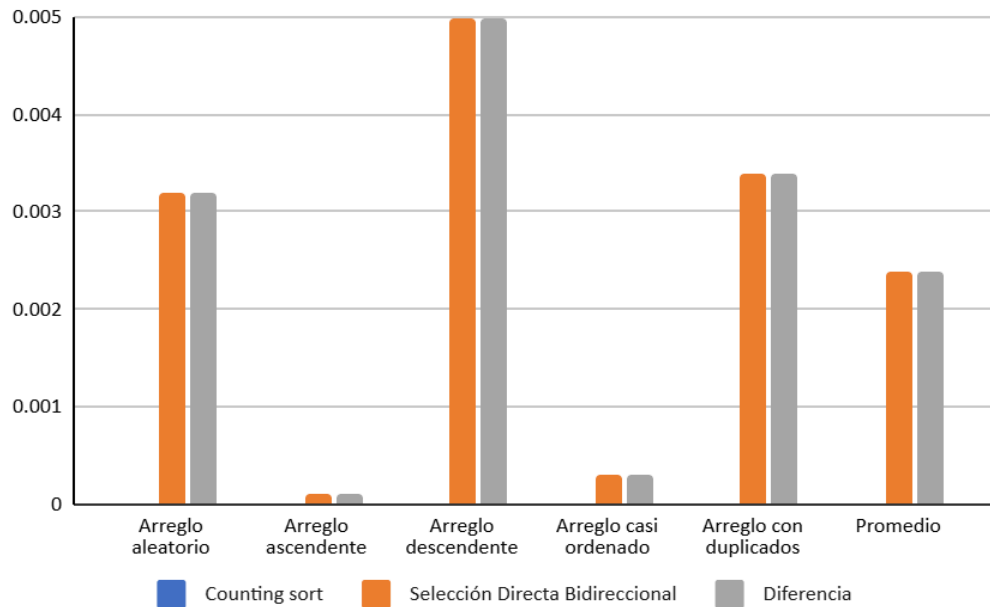


Gráfico 2

Comparación de eficiencia en microsegundos de Counting Sort - Shaker Sort cuando $n = 10000$ y el rango = 10000

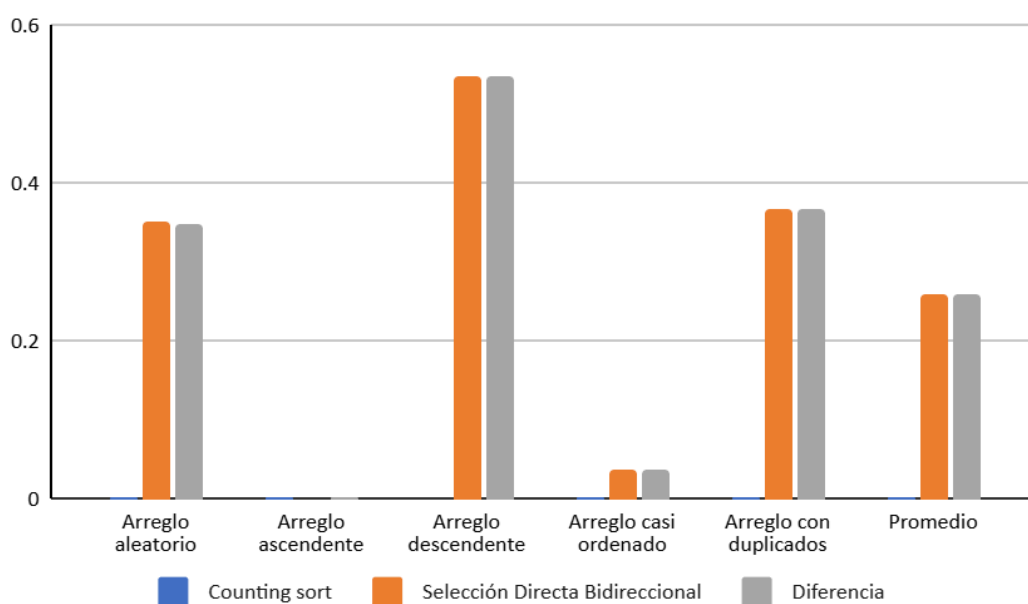


Gráfico 3

Comparación de eficiencia en microsegundos de Counting Sort - Shaker Sort cuando $n = 100000$ y el rango = 100000

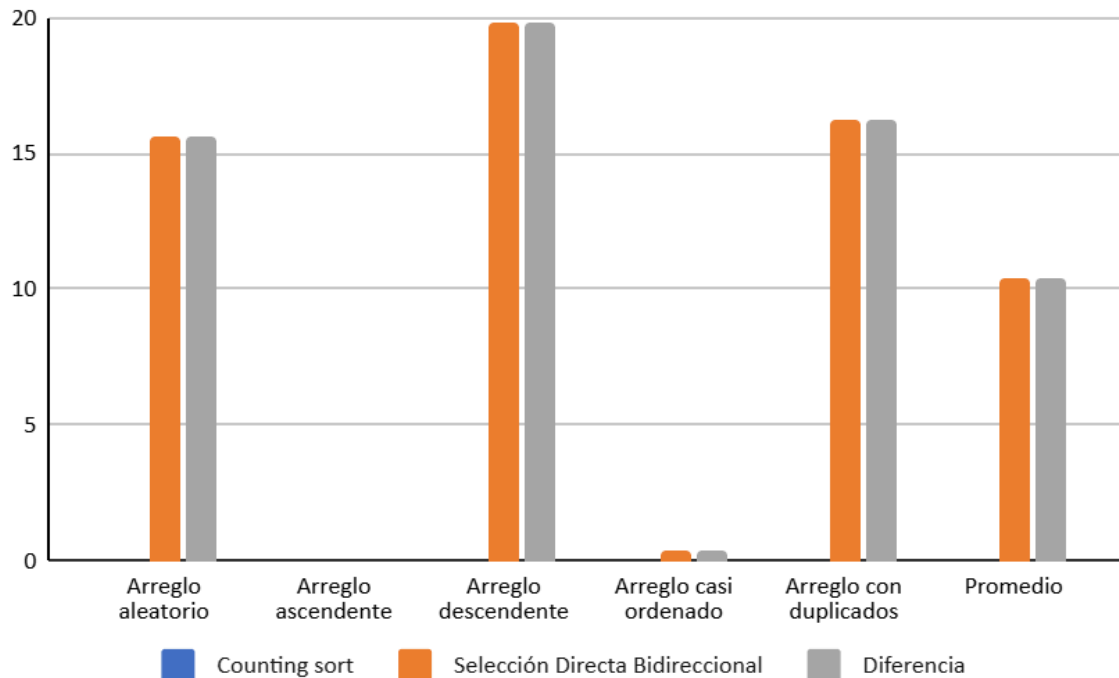
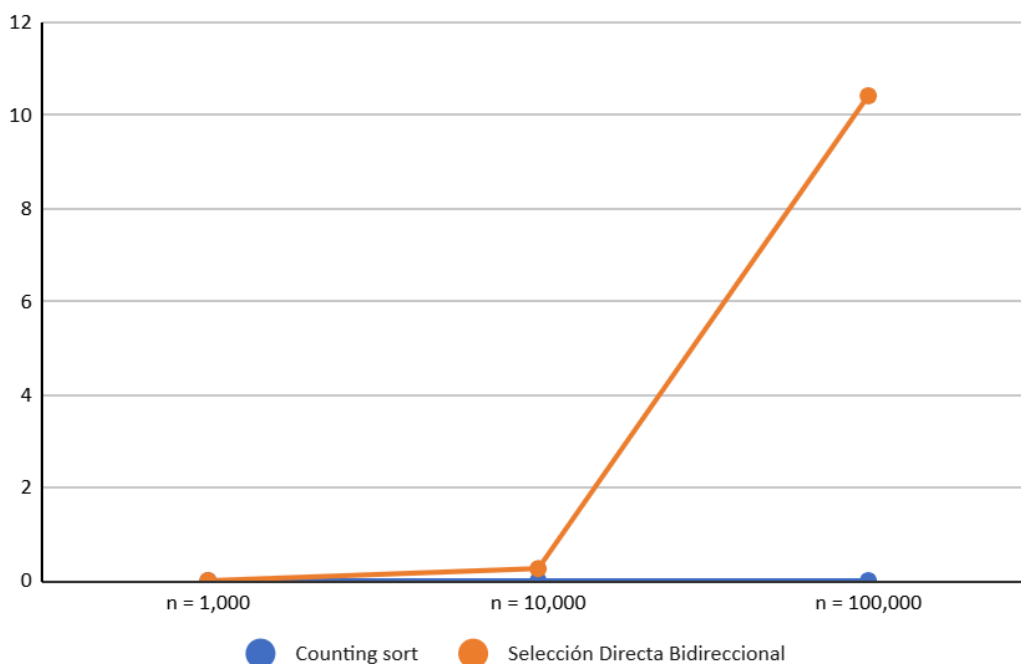


Gráfico 4

Resumen de comparación de eficiencia en microsegundos de Counting Sort - Shaker Sort en los 3 casos de n y rango para los datos



7. Agradecimientos

Agradecemos primero a Dios por habernos acompañado y guiado a lo largo de nuestra vida, y brindarnos experiencias llenas de aprendizaje y felicidad.

En segundo lugar a nuestros docentes por guiarnos, darnos conocimiento, y apoyo durante nuestro proceso de formación.

Finalmente a nuestras familias, por su constante apoyo, ánimo y motivación en los momentos más difíciles.

8. Referencias

- Arráiz et, al. (2004). El Impacto del Cache en Dos Algoritmos de Ordenamiento. Recuperado de:
https://clei.org/proceedings_data/CLEI1996/CLEI1996%20-%20Tomo%201/Por%20capitulo_OCR/CLEI1996_371-382_OCR.pdf
- Duch, A. (2007). Análisis de Algoritmos. *Barcelona, Universidad Politécnica de Barcelona*. Recuperado de:
https://d1wqtxts1xzle7.cloudfront.net/58103052/analisis-libre.pdf?1546463232=&response-content-disposition=inline%3B+filename%3DAnalisis_de_Algoritmos.pdf&Expires=1760602822&Signature=TOwN~vA72vdQ95y6EAHRLh1XYFaDT~GvpoOR-F2BBGWmwUM4YuuYdUvy1JnelV29sZOn8IS0G5J7kXqqTnpLTOMCVZOIOgkeI8VTTTOVbdlWDS6Eelh-5fWexGxZie3m0KNCP2bt8wMWd4S6LEI1tzY6jw95mcMRygtsvjdTU8dvQ2mblyLrB50xN5MjL33KKuxe~CwcQXPmmZRBXeboHfYe-Y9gKphWNa1366Ss3~FfnK6AYGJWKNd4c~2iGbdII B4i8SZCEstS1aLecmJx99XdrZTB7NCIfnAdM071A6JnCkqER7HzQYourGoG3iptlLxYfAWls0tUGo~8FjwH2A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
- Bajpai, K., & Kots, A. (2014). Implementing and analyzing an efficient version of counting sort (e-counting sort). *International Journal of Computer Applications*, 98(9), 1-2. Recuperado de:
https://d1wqtxts1xzle7.cloudfront.net/76736447/pxc3897427-libre.pdf?1639809014=&response-content-disposition=inline%3B+filename%3DImplementing_and_Analyzing_an_Efficient.pdf&Expires=1760987148&Signature=M~VZbpeyNU2rJgCx4P2ry3pEk10cicD3umZjySFRnZIfDHkYj31Oi8HVLkr4iKHel2jUsV7OqEM5IceA11la4Or4xXFh4PT0bQ0GrykySIv~ht5sYnH48R0gqmDom6o2Y6LVy1YsFEgsY4gSK1rNSZudOEyuDHgexWj~Fi~jFBKP4vc1kkmnJCFYtvfXKHnDEI3fllLSWRKOoYNSsNlaY4Z4r6pi56qhmLyBzura1bbSG3rQkilRoZQv-D7E4-ekADkp88MDPp4aEW8e4kipY7msMa-AR-M9tN-riOgfOTr9HyRlf6PRMcpqWeDKsdeAomC51rtGV-ugSxK405eLg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
- Incerpi, J., & Sedgewick, R. (1987). Practical variations of shellsort. *Information Processing Letters*, 26(1), 37-43. Recuperado de:
<https://sedgewick.io/wp-content/themes/sedgewick/papers/1987Practicar.pdf>