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.

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, ademas, el peor de los casos de counting sort es cuando k es mayor que n, por lo tanto los tiempos no se dispararan.

- 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 aun 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.*

- **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.

- **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.

- **Comparaciones**

  El número de comparaciones crece aproximadamente con n^2, confirmando la complejidad teórica.
  Ejemplo: de n=1000 a n=10000 a 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.

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 (≈0 µs), 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 µs), lo que indica detección temprana de orden.

Para n=10000:

El tiempo de Counting Sort se mantiene casi constante y extremadamente bajo (≈0.1 µs).

En cambio, Shaker Sort se vuelve mucho más lento (hasta 193 µs en descendente).

La diferencia promedio es ≈89 µs, 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 lo que Counting Sort.

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

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

Analisis general:

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

Intercambio Bidireccional crece exponencialmente: cuando nnn se multiplica ×10, el tiempo se multiplica ×100 o más.

La diferencia promedio pasa de menos de 1 µs (n=1000) a más de 10 mil µs (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.

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

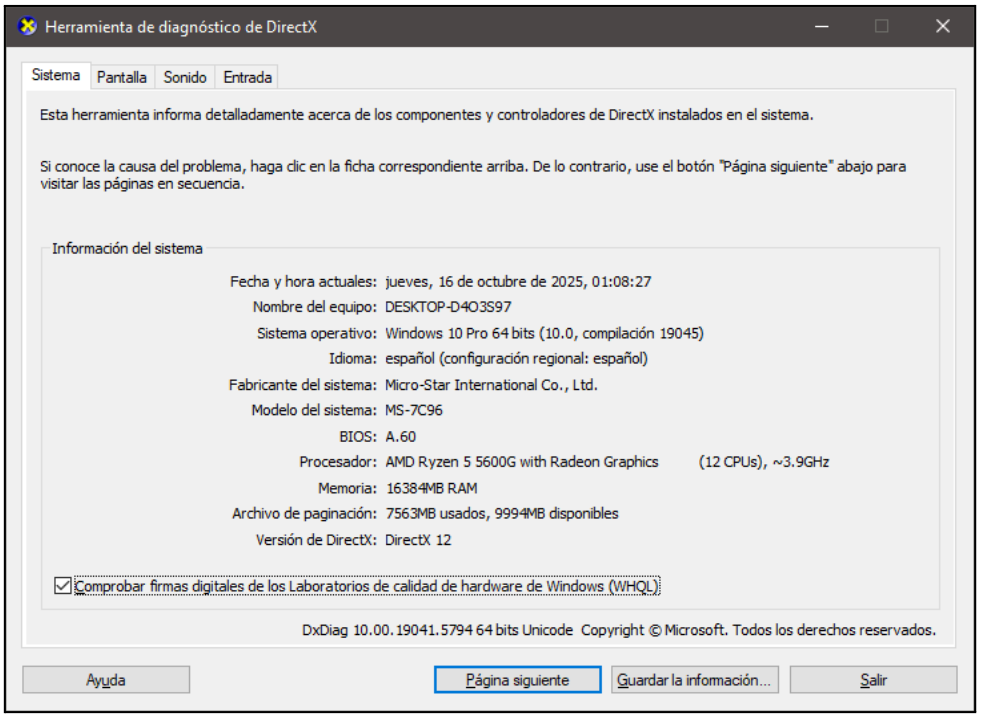


Figura 2

Información detallada sobre la unidad de procesamiento gráfico (GPU)

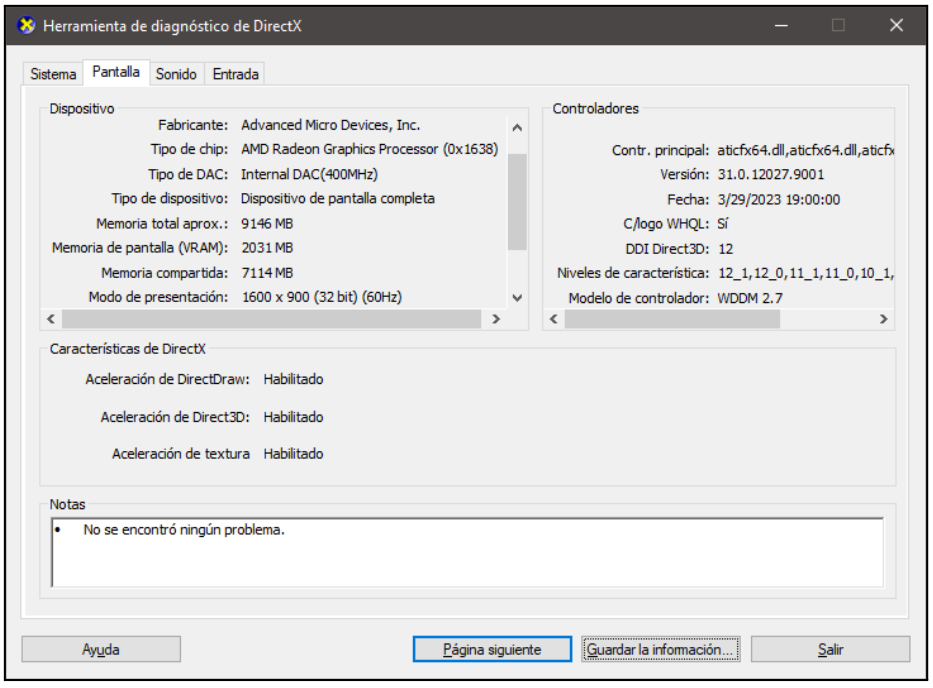


Figura 3

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

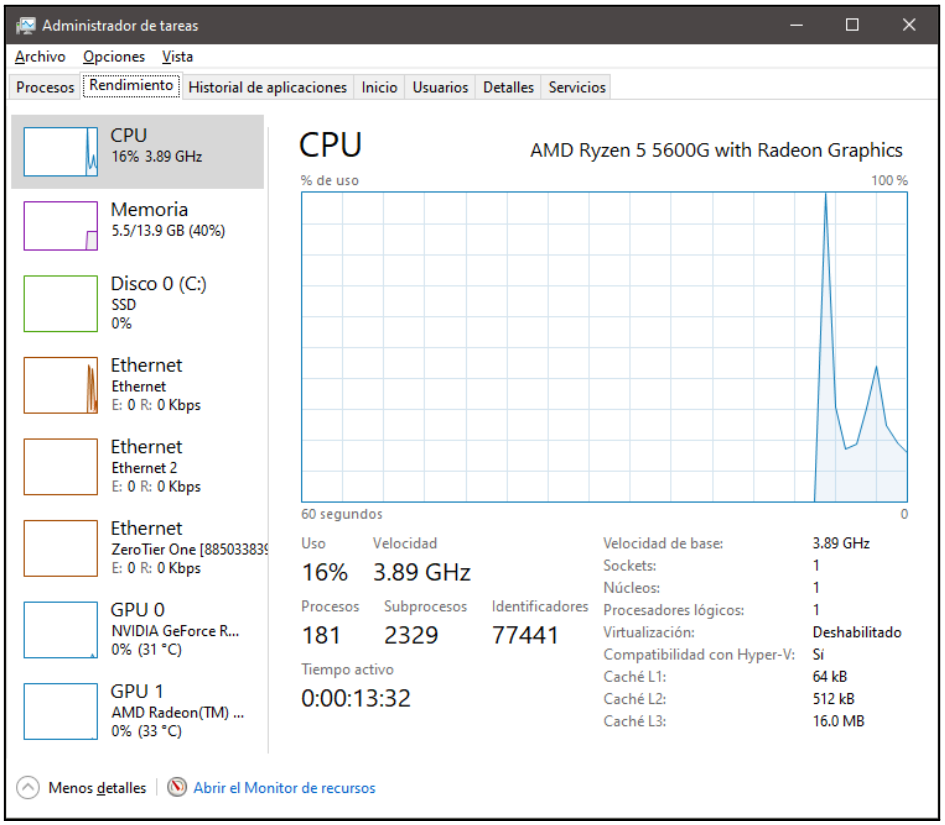


Figura 5

Información del compilador DEV C++ Versión 6.3

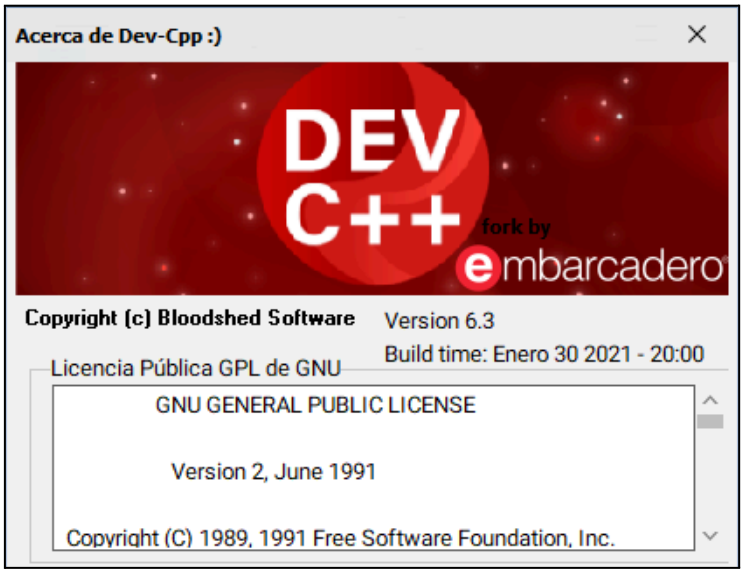


Tabla 1

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	0	0	0	0	0	0	0.996	0	0	**0.0996**
Arreglo ascendente	0	0	0	0	0	0.997	0	0	0	0	**0.0997**
Arreglo descendente	0	0	0	0	0.997	0	0	0	0	0	**0.0997**
Arreglo casi ordenado	0	0.996	0	0	0	0	0	0	0	0	**0.0996**
Arreglo con duplicados	0	0	0	0	0.996	0	0	0	0.996	0	**0.1992**
											0.11956

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	0.998	0.998	0.997	0	0	0.996	0.998	0.997	0.997	0.998	**0.7979**
Arreglo ascendente	0.997	0.997	0.997	0.997	0.997	0	0.997	0.997	0.997	0.997	**0.8973**
Arreglo descendente	0.997	0.996	0.996	0.997	0.998	0.996	0.998	0	0.997	0.997	**0.8972**
Arreglo casi ordenado	0	0.997	0.997	0.996	0	0.997	0	0.997	0.998	0	**0.5982**
Arreglo con duplicados	0	0.997	0	0	0	0	0.996	0.998	0.996	0.996	**0.4983**
											0.73778

Tabla 4

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	0.997	1.995	0.998	0.997	0.997	0.996	0.997	0.997	0.997	0.998	**1.0969**
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	**0**
Arreglo descendente	1.994	1.995	1.994	1.995	2.002	1.994	1.994	1.994	1.996	1.996	**1.9954**
Arreglo casi ordenado	0	0.998	0.998	0	0	0	0	0.997	0	0	**0.2993**
Arreglo con duplicados	0.998	0.998	0.997	0.998	0.998	0.997	0.998	0.997	0.997	0.997	**0.9975**
											0.87782

Tabla 5

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

Característica de datos	Comparaciones										Comparaciones Promedio
n=1000, rango=1000	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Arreglo aleatorio	334237	333139	344761	336587	326490	338249	338183	333004	329866	336394	**335091**
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	32422	32120	39323	33018	35296	33967	32995	31589	33385	35246	**33936.1**
Arreglo con duplicados	324961	329229	337692	335710	333954	335103	343241	329841	328885	329581	**332819.7**
											240469.16

Tabla 6

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

Característica de datos	*Intercambios*										*Intercambios Promedio*
n=1000, rango=1000	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	
Arreglo aleatorio	*248227*	*249526*	*257723*	*254145*	*245695*	*253613*	*250161*	*250598*	*249430*	*250270*	***250938.8***
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	*28894*	*29258*	*35984*	*29894*	*32524*	*31248*	*29716*	*28865*	*29540*	*32316*	***30823.9***
Arreglo con duplicados	*240739*	*242823*	*250053*	*251216*	*244556*	*248399*	*256239*	*245466*	*240072*	*245488*	***246505.1***
											205553.56

Tabla 7

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	121.674	125.501	122.367	124.26	121.297	122.211	121.246	120.711	121.676	121.674	**122.2617**
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	**0**
Arreglo descendente	193.503	194.485	193.5	193.178	193.571	192.37	194.02	193.537	192.486	192.6	**193.325**
Arreglo casi ordenado	12.964	12.965	13.478	12.967	12.966	12.965	12.967	13.966	11.967	12.966	**13.0171**
Arreglo con duplicados	118.715	119.705	119.706	118.273	119.335	118.666	120.708	116.706	118.714	117.689	**118.8217**
											89.4851

Tabla 8

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

Característica de datos	*Comparaciones*										*Comparaciones Promedio*
n=10000, rango=10000	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	
Arreglo aleatorio	*33227815*	*33694740*	*33481849*	*33464246*	*33131663*	*33517185*	*33368018*	*33203978*	*33215081*	*33442025*	***33374660***
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	*3342184*	*3318759*	*3258181*	*3381437*	*3319524*	*3159917*	*3284382*	*3451226*	*3241770*	*3281998*	***3303937.8***
Arreglo con duplicados	*33008955*	*33161397*	*33447244*	*32975720*	*33006601*	*33190891*	*33464685*	*32658756*	*33365663*	*33159491*	***33143940.3***
											23965507.42

Tabla 9

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	*24955452*	*25307285*	*25139135*	*25209058*	*24876890*	*25061789*	*24937929*	*24900256*	*25004865*	*25134996*	***25052765.5***
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	*3233710*	*3222192*	*3149330*	*3274086*	*3206564*	*3049070*	*3183320*	*3337764*	*3130450*	*3171128*	***3195761.4***
Arreglo con duplicados	*24370203*	*24565471*	*24768930*	*24408090*	*24406380*	*24470061*	*24730948*	*24163023*	*24720058*	*24527590*	***24513075.4***
											20551320.46

Tabla 10

Resultados de eficiencia en tiempo del algoritmo Shaking 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	16134.9	16289.2	16179.8	16217.3	16100.6	16092.5	16272.3	16379.9	16325.3	16365.4	**16235.72**
Arreglo ascendente	0	0	0	0	0	0	0.997	0	0	0	**0.0997**
Arreglo descendente	19457.4	19767.3	19849.7	19889.3	19805.4	19761.2	19665	19693.1	19630.2	19799.7	**19731.83**
Arreglo casi ordenado	403.386	410.171	381.516	381.859	399.585	386.045	376.038	384.639	388.91	392.779	**390.4928**
Arreglo con duplicados	16403.5	16563.6	16476.9	16365.8	16572.8	16377.2	16288.2	16278.7	16449.7	16193	**16396.94**
											10551.0165

Tabla 11

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	3320392667	3335684504	3328923079	3341477241	3325298834	3334259086	3330218507	3341652737	3338510630	3333964971	**3333038226**
Arreglo ascendente	99999	99999	99999	99999	99999	99999	99999	99999	99999	99999	**99999**
Arreglo descendente	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	**4999950000**
Arreglo casi ordenado	94806409	94219496	93633657	93798885	95074710	93666773	92477740	94008696	93984769	95551727	**94122286.2**
Arreglo con duplicados	3304293398	3317815112	3313140224	3312355437	3313922906	3305946646	3323178018	3316792441	3298001670	3308946280	**3311439213**
											2347729945

Tabla 12

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	2490609760	2497793616	2496818361	2505056319	2490515827	2498341152	2496134292	2506121431	2504988956	2496958038	**2498333775**
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	**0**
Arreglo descendente	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	4999950000	**4999950000**
Arreglo casi ordenado	88315567	87881261	87196260	87384084	88612166	87268520	86142056	87500946	87517226	89032390	**87685047.6**
Arreglo con duplicados	2448123970	2455322650	2451955697	2451160399	2452597105	2447075137	2459659610	2453434377	2439343581	2448647325	**2450731985**
											2007340162

Tabla 13

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	1.0969	1.0969
Arreglo ascendente	0	0	0
Arreglo descendente	0	1.9954	1.9954
Arreglo casi ordenado	0	0.2993	0.2993
Arreglo con duplicados	0	0.9975	0.9975
Promedio	**0**	**0.87782**	**0.87782**

Tabla 14

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	0.0996	122.2617	122.1621
Arreglo ascendente	0.0997	0	0.0997
Arreglo descendente	0.0997	193.325	193.2253
Arreglo casi ordenado	0.0996	13.0171	12.9175
Arreglo con duplicados	0.1992	118.8217	118.6225
Promedio	**0.11956**	**89.4851**	**89.36554**

Tabla 15

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	0.7979	16235.72	16234.9221
Arreglo ascendente	0.8973	0.0997	0.7976
Arreglo descendente	0.8972	19731.83	19730.9328
Arreglo casi ordenado	0.5982	390.4928	389.8946
Arreglo con duplicados	0.4983	16396.94	16396.4417
Promedio	**0.73778**	**10551.0165**	**10550.27872**

Tabla 16

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	0.87782
n = 10,000	0.11956	89.4851
n = 100,000	0.73778	10551.0165

Gráfico 1

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

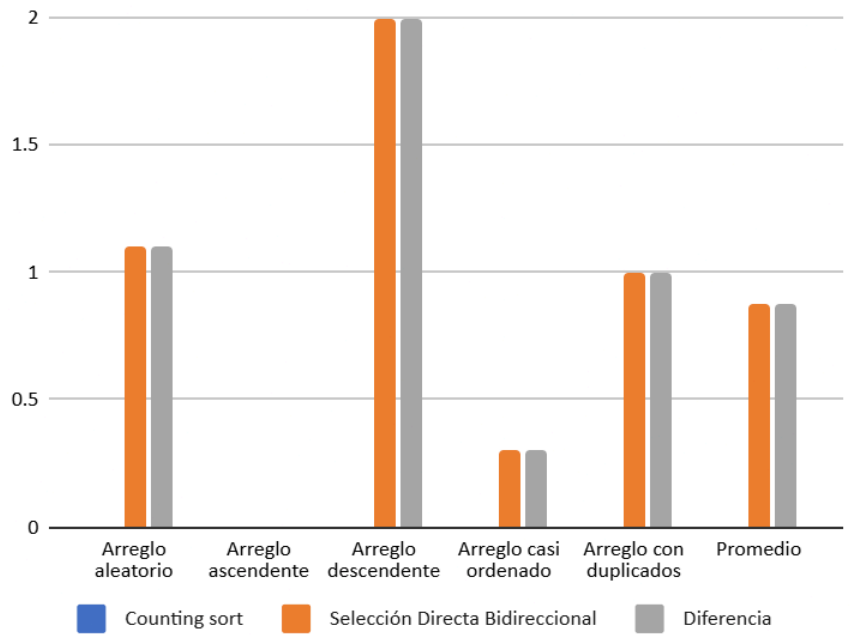


Gráfico 2

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

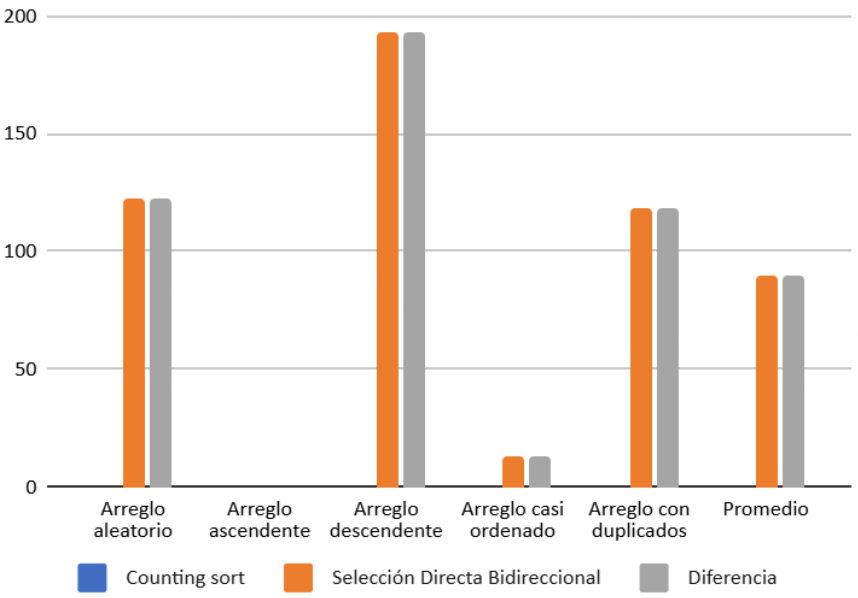


Gráfico 3

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

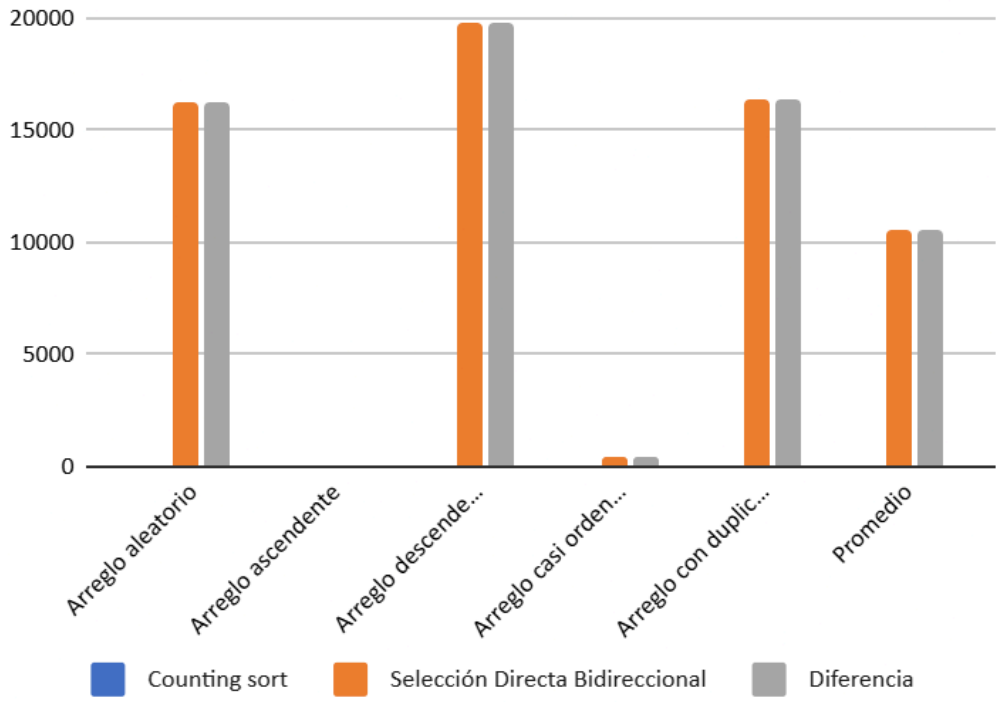
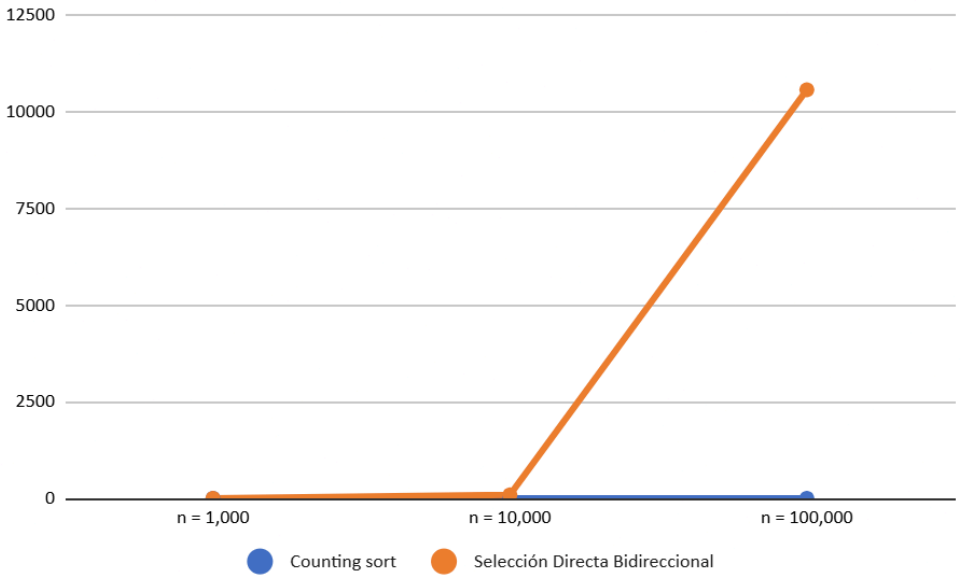


Gráfico 4

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=TOrWn~vA72vdQ95y6EAHRLh1XYFaDT~GvpoOR-F2BBGWmwUM4YuuYdUvy1JnelV29sZOn8IS0G5J7kXqqTnpLTOMCVZOlQgkeI8VTTOVbdlWDS6EeIh-5fWexGxZie3m0KNCP2bt8wMWd4S6LEI1tzY6jw95mcMRygtsvjdTU8dvQ2mbIyLrB50xN5MjL33KKuxe~CwcQXPmmZRBXeboHfYe-Y9gKphWNa1366Ss3~FfnK6AYGJWKNd4c~2iGbdlIB4i8SZCEstS1aLecmjx99XdrZTB7NCIfnAdM071A6JnCkqER7HzQYourGoG3iptILxYfAWls0tUGo~8FjwH2A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA**