UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN | FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS
ASIGNATURA: PROGRAMACIÓN AVANZADA | CICLO: SEGUNDO | SEMESTRE: 2025-II

Comparación de algoritmos de ordenación

COUNTINGSORT - SHAKERSORT

Russbell Juan Pablo Arratia Paz

Alexander Efrain Contreras Rodriguez

Danitza Carmen Capía Quiñonez

Objetivos

IMPLEMENTAR

REGISTRAR

ANALIZAR

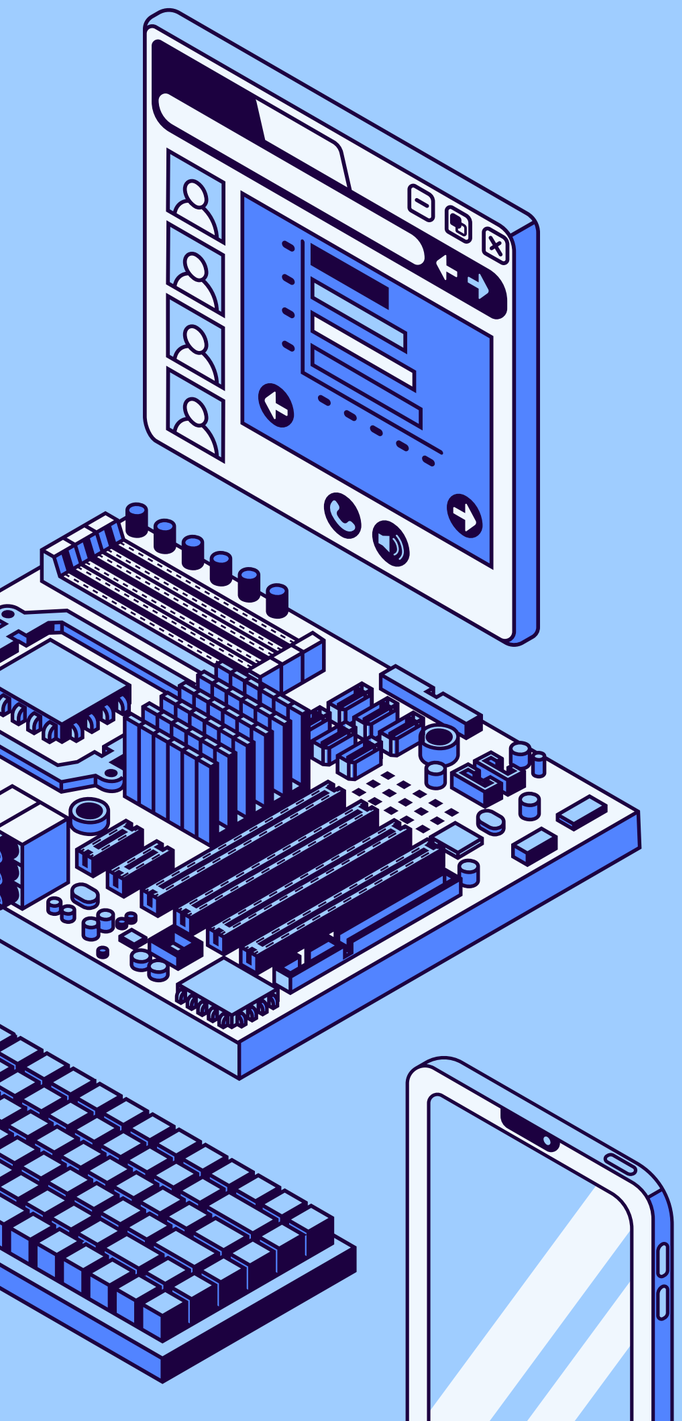
VISUALIZAR

Metodologia

1. Comparación entre algoritmos de ordenacmiento en espacios controlados.

2. 50 Realizaciones por tamaño.

3. Evaluación según complejidad.



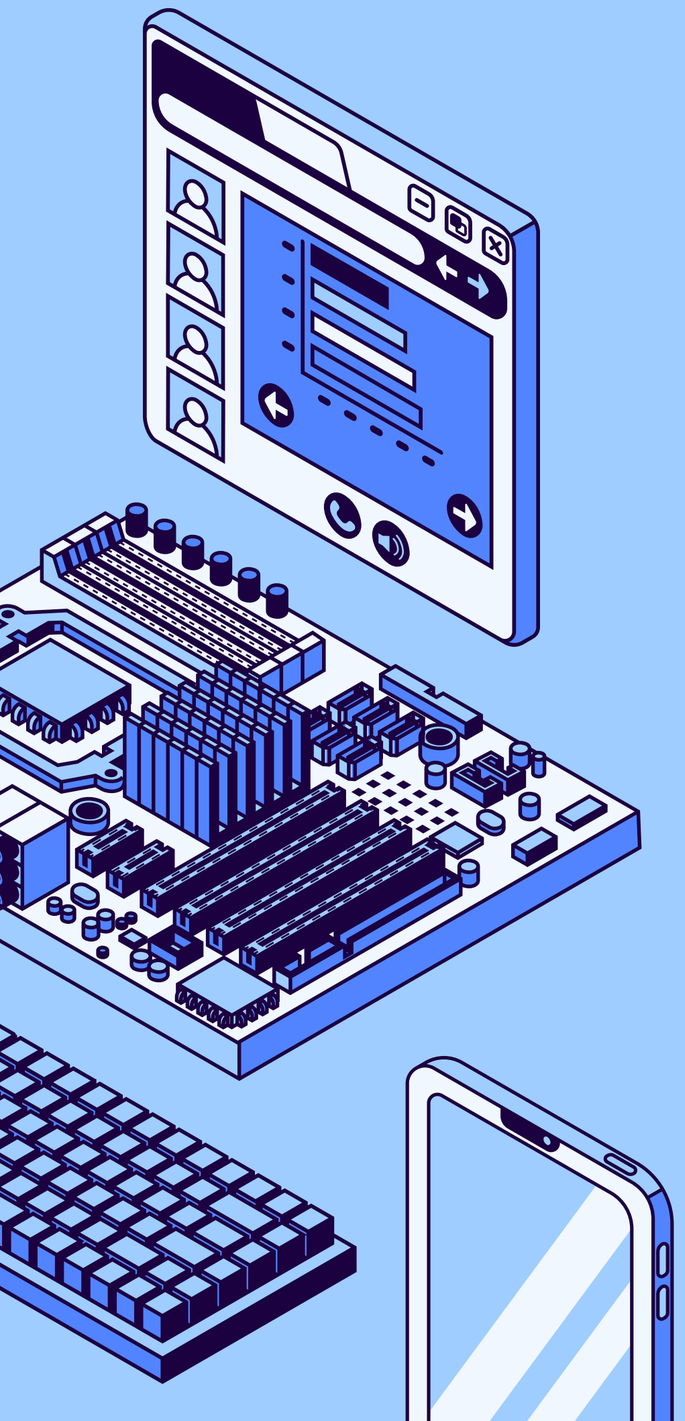
Counting Sort

Su modo de ordenación no es por comparación, sino que cuenta cuántas veces aparece un valor en el arreglo, infomación que usa para para colocarlos en orden

Mejor, promedio y peor caso: O(n + k)

Complejidad de espacio: O(k)

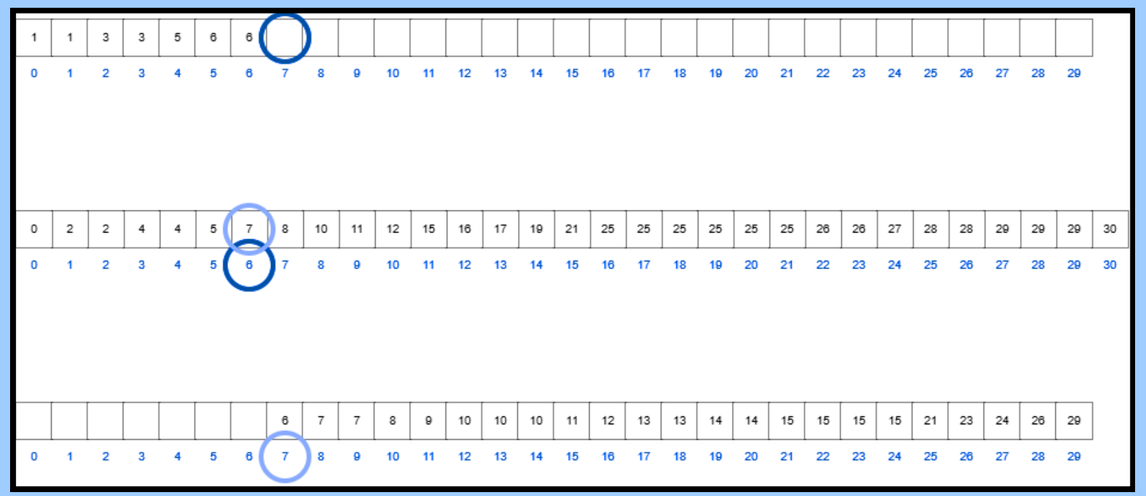
Counting Sort



26	1	24	6	23	11	13	12	5	13	29	7	14	9	10	15	10	1	3	6	8	15	3	10	21	7	14	15	6	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

0	2	2	4	4	5	8	10	11	12	15	16	17	19	21	4	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

0	0	2	2	4	4	5	8	10	11	12	15	16	17	19	21	25	25	25	25	25	25	26	26	27	28	28	29	29	29	30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30



Variante del ordenamiento por selección que busca el mínimo y el máximo para posicionarlos a los extremos en cada pasada, hasta terminar en el medio.

Mejor caso: O(n)

Peor caso: $O(n^2)$

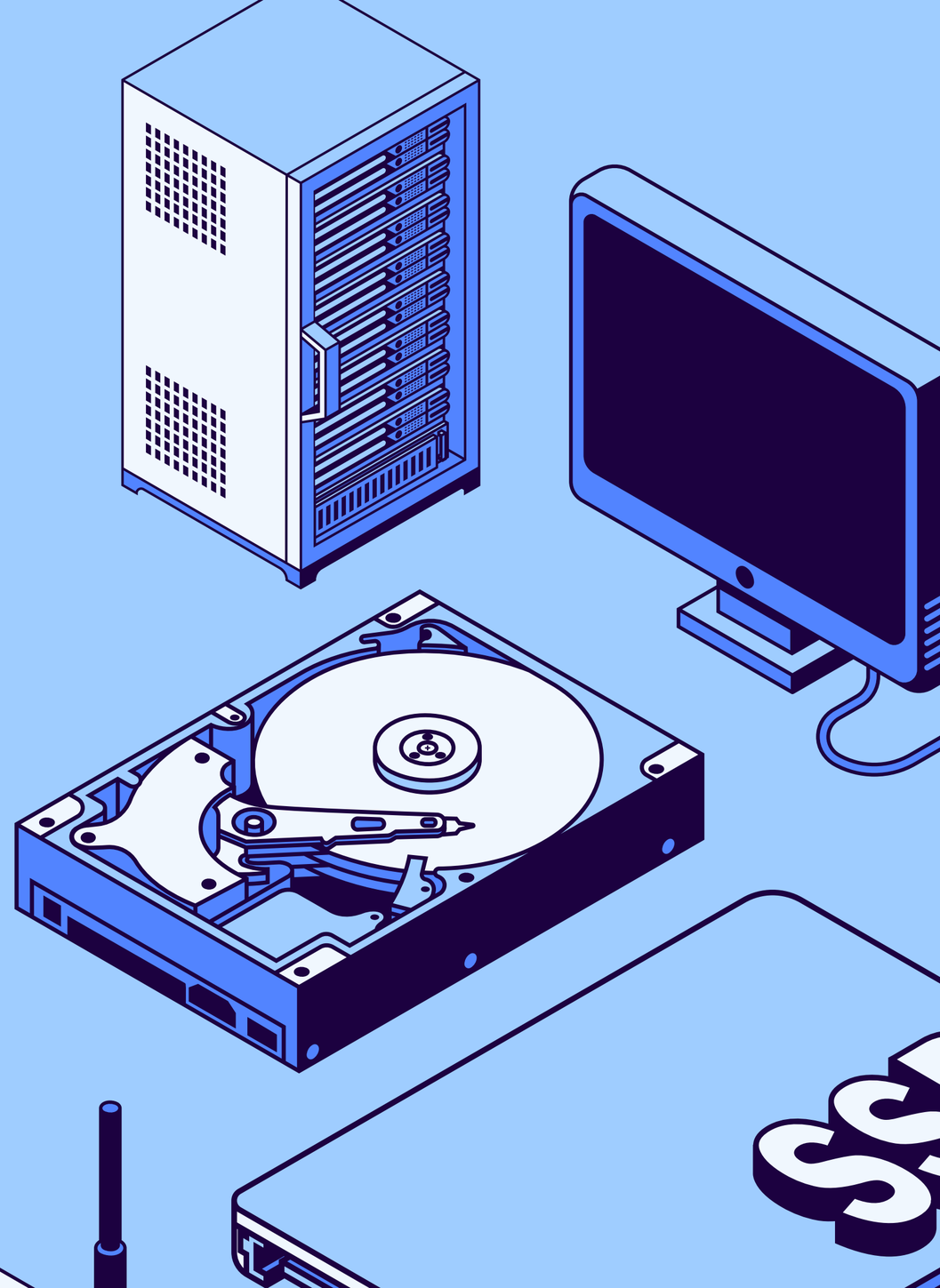
Complejidad promedio: $O(n^2)$

Complejidad de espacio: O(1)

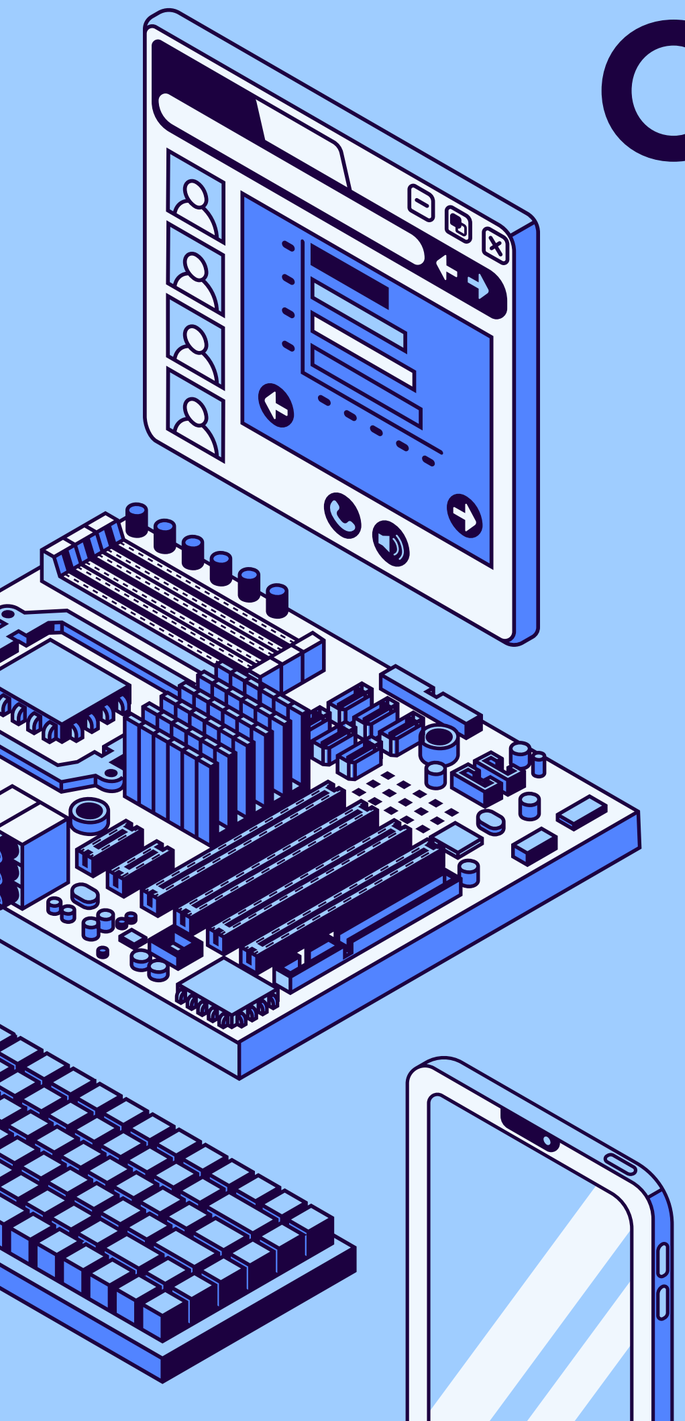
Intercambio Directo Bidireccional

Resultados

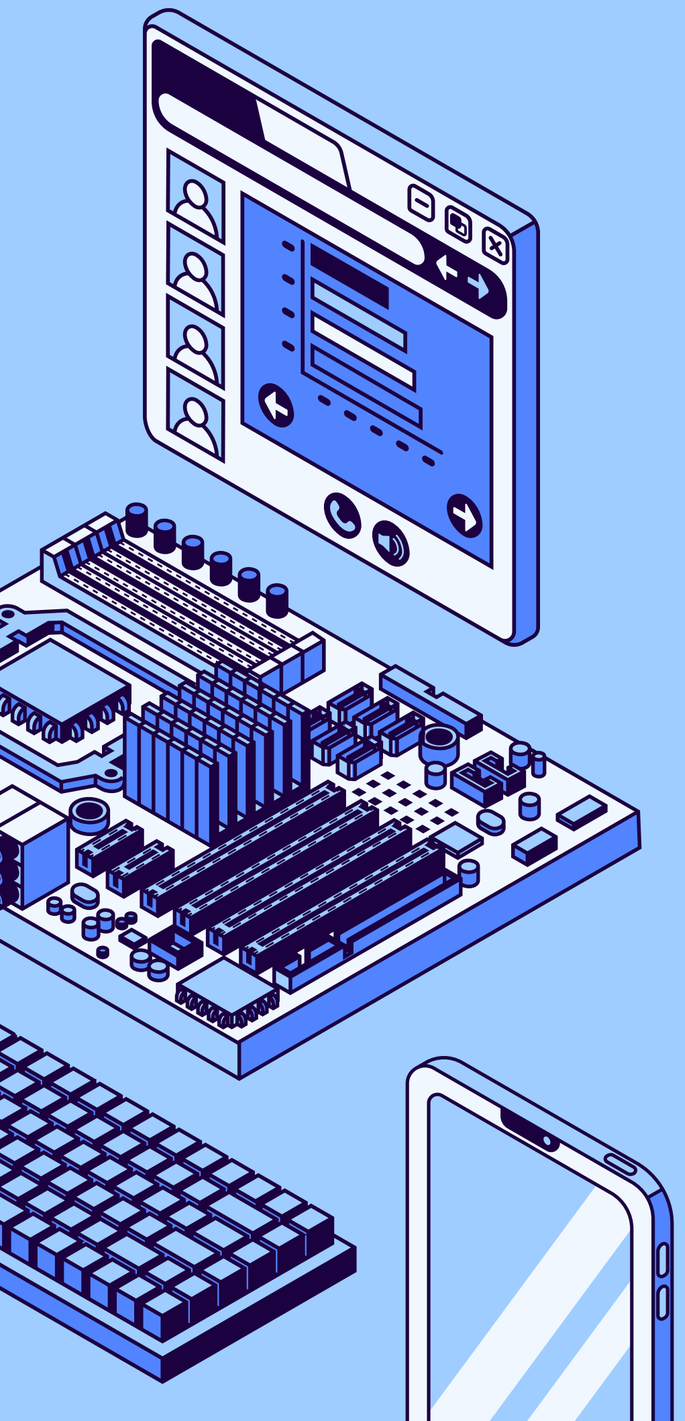
Luego de la ejecusión de las pruebas con ambos algoritmos, se obtuvieron los resultados a continuación, seguidos de un análisis comparativo.



Counting Sort



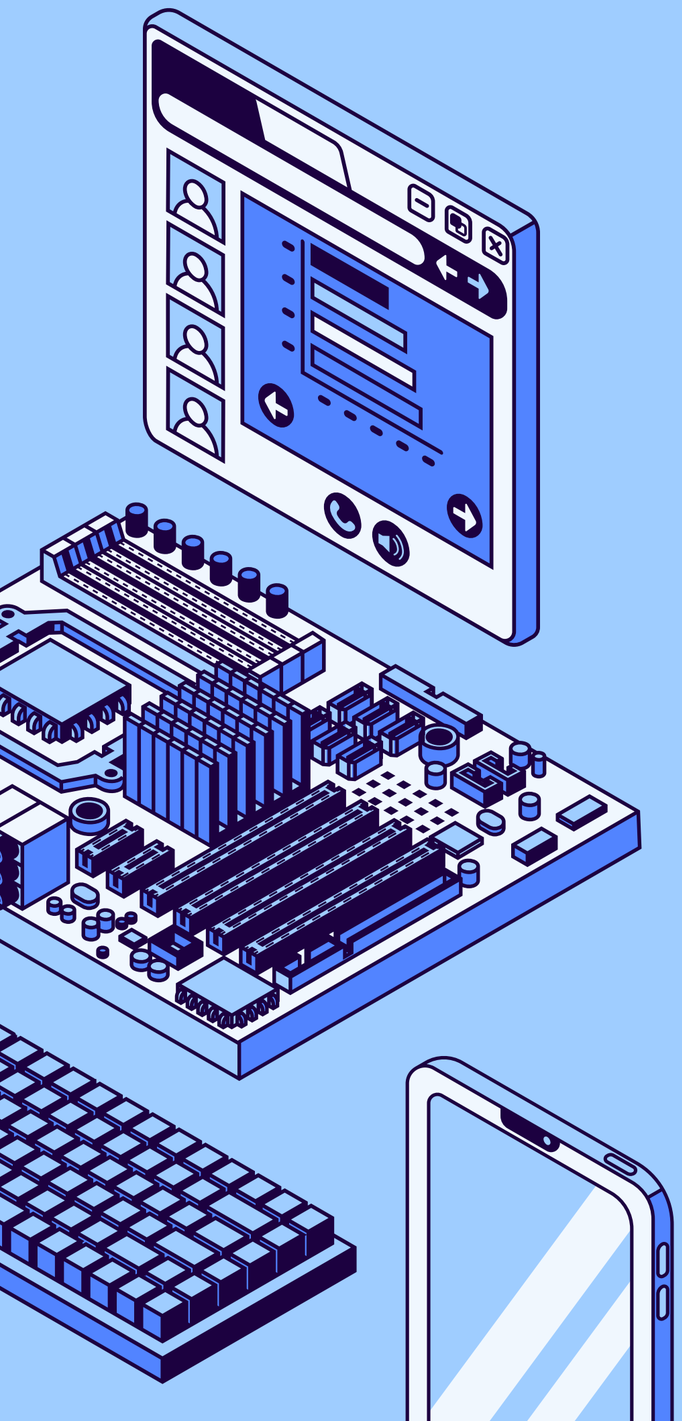
Característica de datos	Tiempo en microsegundos										Tiempo
n=1000, rango=1000	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Promedio**
Arreglo aleatorio	0	0	0	0	0	0	0	0	0	0	**0**
Arreglo ascendente	0	0	0	0	0	0	0	0	0	0	**0**
Arreglo descendente	0	0	0	0	0	0	0	0	0	0	**0**
Arreglo casi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



Counting Sort

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



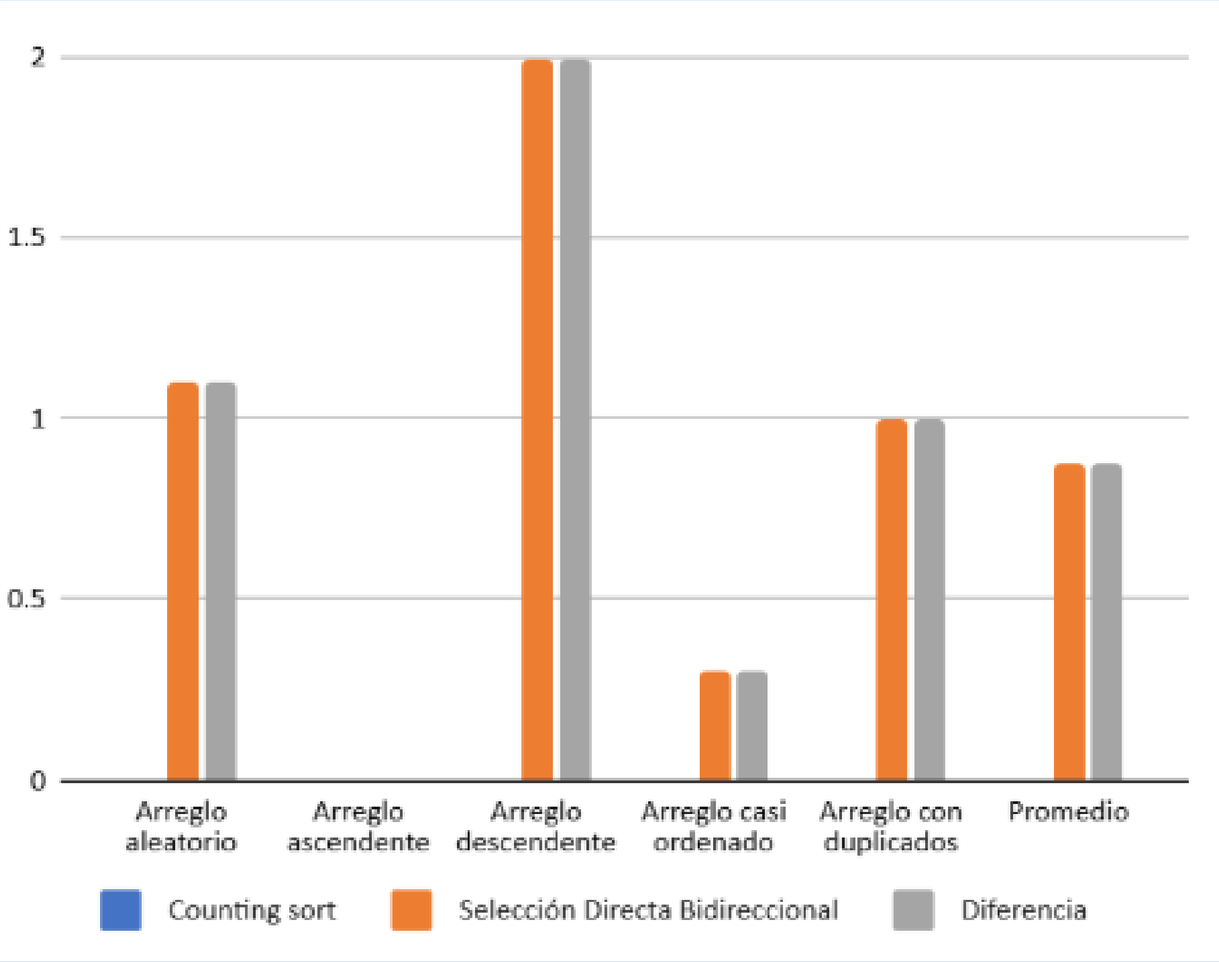
Counting Sort

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

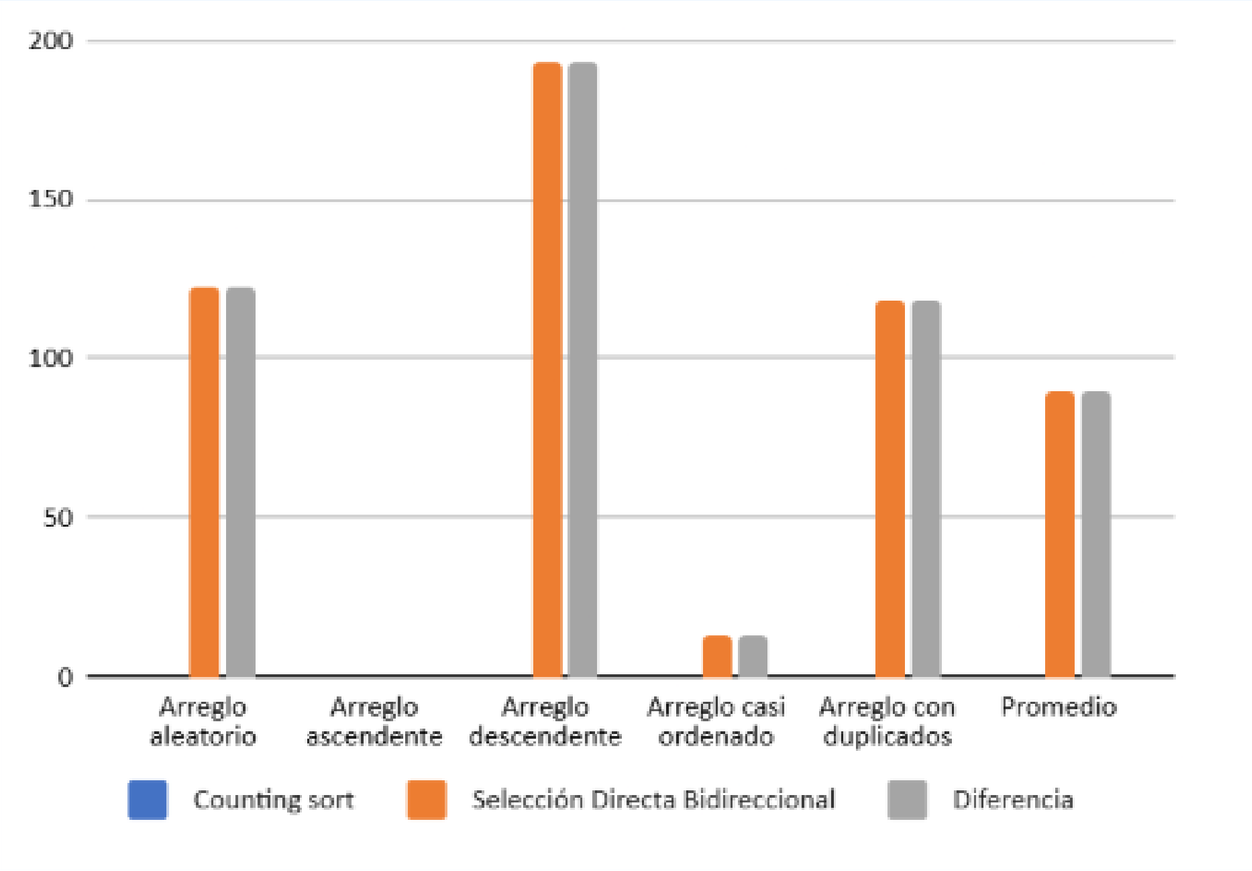
Intercambio Directo Bidireccional

Análisis



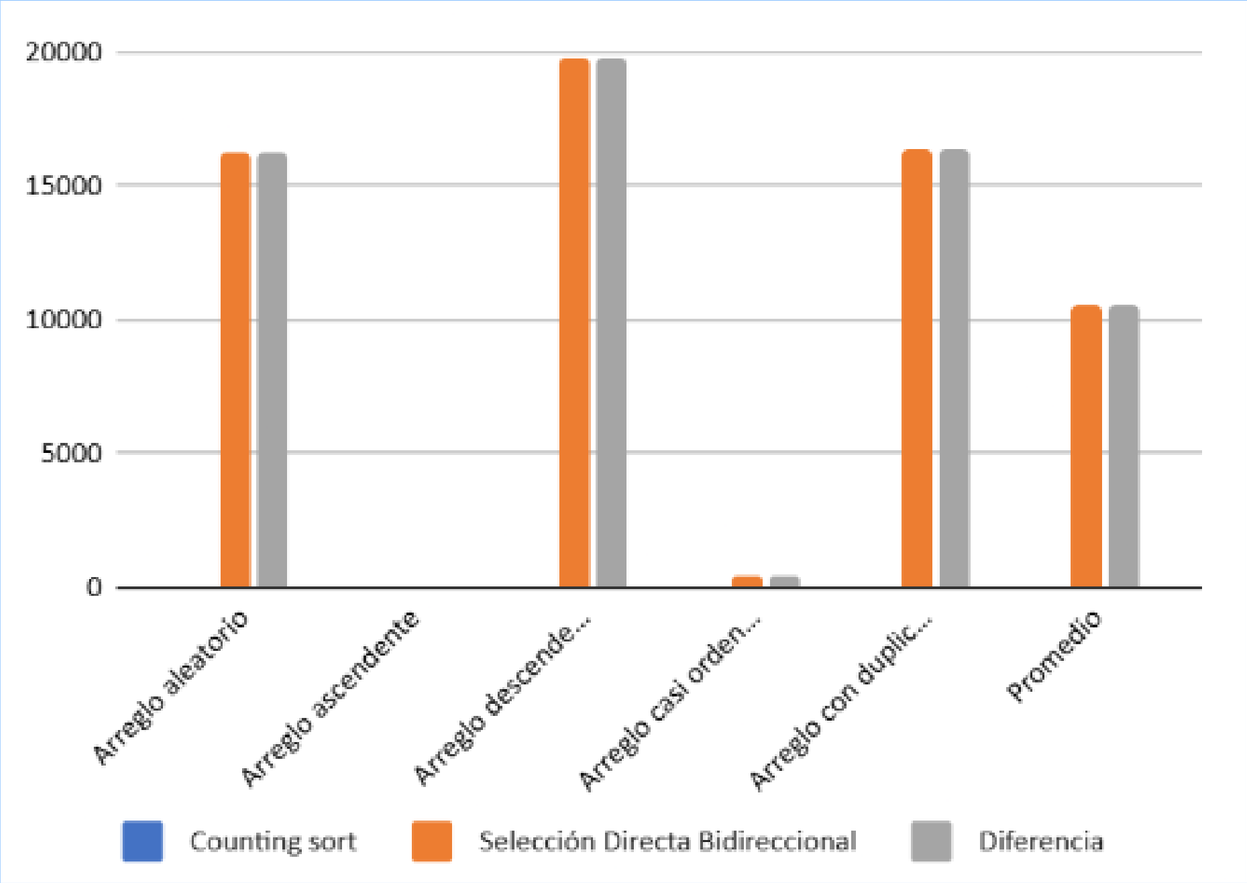
n=1000, rango=1000	Counting sort	Selección Directa Bidireccional	Diferencia
Arreglo aleatorio	0	1.0969	1.0969
Arreglo ascendente	0	0	0
Arreglo descendente	0	1.9954	1.9954
Arreglo casi ordenado	0	0.2993	0.2993
Arreglo con duplicados	0	0.9975	0.9975
Promedio	**0**	**0.87782**	**0.87782**

Análisis



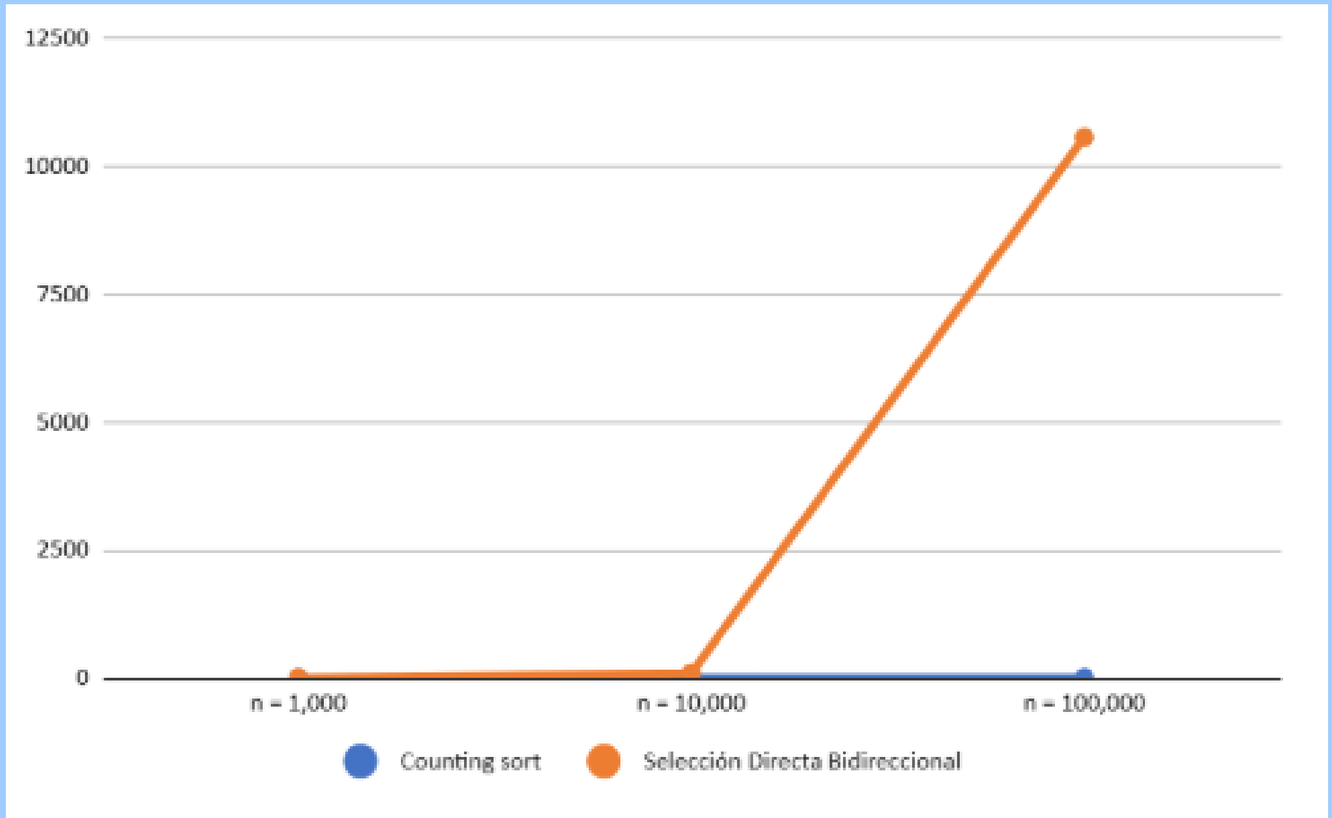
n=10000, rango=10000	Counting sort	Selección Directa Bidireccional	Diferencia
Arreglo aleatorio	0.0996	122.2617	122.1621
Arreglo ascendente	0.0997	0	0.0997
Arreglo descendente	0.0997	193.325	193.2253
Arreglo casi ordenado	0.0996	13.0171	12.9175
Arreglo con duplicados	0.1992	118.8217	118.6225
Promedio	**0.11956**	**89.4851**	**89.36554**

Análisis



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**

Análisis



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

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.

Gracias