Análisis Comparativo de Eficiencia entre los Algoritmos Shellsort y Counting Sort

Comparative Efficiency Analysis between Shell Sort and Counting Sort Algorithms

Pedro Pablo Ninaja Medina
pninjam@unjbg.edu.pe
Universidad Nacional jorge Basadre Grohman
ORCID:https://orcid.org/0009-0009-3285-598X

Jairo Brayan Zegarra Gutierrez
jzegarrag@unjbg.edu.pe
Universidad Nacional jorge Basadre Grohman
ORCID:https://orcid.org/0009-0009-3285-598X

Arturo Willy Montalico Llica
Amontalicol@unjbg.edu.pe
Universidad Nacional jorge Basadre Grohman
ORCID: https://orcid.org/0009-0006-9380-1799

Resumen: El presente trabajo tiene como objetivo analizar comparativamente la eficiencia de los algoritmos de ordenamiento Shellsort y Counting Sort, evaluando su comportamiento en función de la complejidad temporal, espacial y estabilidad, con base en la teoría del análisis algorítmico y experimentación práctica. El estudio busca determinar cuál de los dos algoritmos presenta mejor rendimiento frente a diferentes tamaños y características de conjuntos de datos.

A través de pruebas computacionales realizadas en lenguaje C++, se comparan los tiempos de ejecución de ambos algoritmos con arreglos de tamaño variable, observando que Counting Sort presenta una mayor eficiencia en casos donde los datos son enteros y su rango es reducido, mientras que Shellsort resulta más versátil cuando el conjunto de datos es amplio o heterogéneo.

Palabras clave: algoritmos, eficiencia, complejidad, Shellsort, Counting Sort, ordenamiento.

Abstract: This research aims to perform a comparative analysis of the efficiency between Shellsort and Counting Sort algorithms, evaluating their behavior according to temporal and spatial complexity and stability. The study determines which algorithm exhibits superior performance under different data conditions.

Through computational experiments implemented in C++, the execution times of both algorithms were compared using data sets of varying sizes. The results show that Counting Sort performs better when data are integers within a limited range, while Shellsort is more versatile and efficient with larger or heterogeneous data sets.

Keywords: algorithms, efficiency, complexity, Shellsort, Counting Sort, sorting algorithms.

1. Introducción

En el ámbito de la computación, el ordenamiento de datos es una operación crítica, pues optimiza el acceso y procesamiento de información dentro de sistemas informáticos. La selección de un algoritmo adecuado depende directamente de la eficiencia con que organiza los datos según su naturaleza y tamaño.

El estudio de los algoritmos de ordenamiento ha sido un tema constante en la ingeniería de software y la ciencia de datos. Existen múltiples técnicas, desde las más básicas como Bubble Sort e Insertion Sort, hasta otras más optimizadas como Merge Sort, Quick Sort, Shellsort y Counting Sort. Cada algoritmo presenta diferencias estructurales que afectan el tiempo de ejecución, la complejidad espacial y la estabilidad del resultado.

Este trabajo propone un análisis comparativo entre Shellsort y Counting Sort, dos métodos de naturaleza distinta: uno basado en comparaciones y otro no comparativo. Se busca reforzar la comprensión de la notación Big O y se presentan pruebas experimentales implementadas en C++, que ilustran el comportamiento real de cada algoritmo.

2. MÉTODO:

Algoritmos y variantes

En este estudio se compararon dos algoritmos de ordenamiento con enfoques opuestos en su naturaleza operativa: Shellsort y Counting Sort.

- **Shellsort:** Algoritmo comparativo derivado de Insertion Sort. Mejora el desempeño mediante comparaciones de elementos separados por un intervalo (*gap*) que se reduce progresivamente hasta 1. Se utilizó la secuencia de incrementos de Ciura, conocida por ofrecer un balance eficiente entre tiempo y número de comparaciones.

- **Counting Sort:** Algoritmo no comparativo que organiza los datos contando las ocurrencias de cada valor. Se implementó la versión estable, que conserva el orden relativo de elementos iguales mediante un arreglo auxiliar acumulativo.

Ambos algoritmos fueron codificados desde cero en C++ usando únicamente las librerías estándar (`iostream`, `vector`, `algorithm`, `chrono`, `fstream`, `string`), evitando optimizaciones implícitas de librerías externas.

Complejidad teórica

La comparación se fundamentó en los modelos de complejidad temporal y espacial descritos por la notación asintótica Big O:

Algoritmo	Mejor caso	Promedio	Peor caso	Espacio
Shellsort	O(n log n)	O(n^(3/2))	O(n²)	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(n + k)

Teóricamente, Counting Sort debería superar a Shellsort en tiempo cuando el rango de datos *k* es comparable o menor que *n*, mientras que Shellsort debería ser más eficiente en espacio y aplicable a tipos de datos no enteros.

Entorno de ejecución

Las pruebas experimentales se llevaron a cabo en un entorno controlado con las siguientes especificaciones de hardware y software:

- **Procesador:** Intel Core i5-210H (6 núcleos, 2.4 GHz)
- **Memoria RAM:** 16 GB DDR4
- **Sistema operativo:** Windows 11 Pro 64 bits
- **Lenguaje:** C++
- **Editor:** Visual Studio Code 1.91
- **Flags de compilación:** optimización estándar `-O2`

Este entorno permitió asegurar la reproducibilidad de las mediciones, minimizando interferencias de otros procesos del sistema.

Generación de datos

El rango de valores fue de 1 a 10,000, lo cual garantiza diversidad suficiente sin producir un rango excesivamente amplio que perjudique el desempeño de Counting Sort.

Se probaron tres patrones de entrada representativos:

1. Aleatorio: los valores se distribuyen de forma uniforme.
2. Ascendente: los datos se encuentran preordenados (mejor caso).
3. Descendente: los datos están en orden inverso (peor caso).

Tamaños de entrada y repeticiones

Los tamaños de los arreglos (*n*) fueron los siguientes:
1,000, 5,000, 10,000, 20,000 y 50,000 elementos.

Cada combinación de algoritmo y patrón de datos se ejecutó 30 veces, calculándose el promedio de los tiempos de ejecución. Para reducir la influencia de valores atípicos debidos a variaciones temporales del sistema, se aplicó un promedio truncado del 90%, descartando el 10% superior e inferior de los tiempos medidos. Esto permitió obtener mediciones más consistentes y representativas del desempeño real de los algoritmos en C++

Métricas y procedimientos de medición

Las métricas seleccionadas fueron las siguientes:

1. Tiempo de ejecución (ms): usando `std::chrono::high_resolution_clock` antes y después de cada ejecución; promedio de 30 repeticiones.
2. Uso de memoria (KB): calculado a partir del tamaño total de los vectores (`sizeof(int) * n + memoria auxiliar`).
3. Eficiencia relativa (%):

$$E = \frac{T_{Shellsort}}{T_{CountingSort}} \times 100$$

4. Complejidad empírica:
    - ajuste de curvas logarítmicas (n log n y n + k) a los datos experimentales.

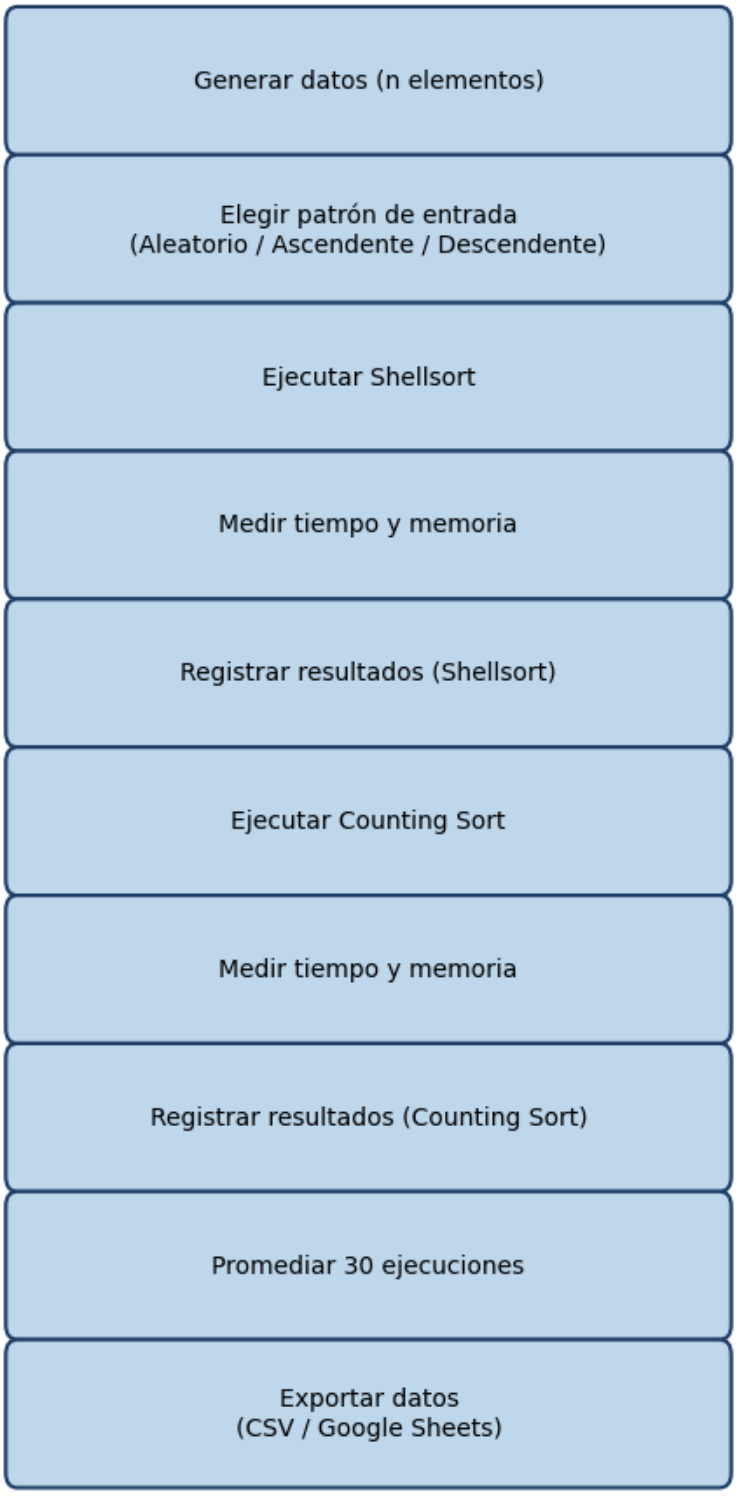


Figura 1. Diagrama del procedimiento de medición y registro de datos.
Elaboración propia (2025).

Control de condiciones experimentales

Cada ejecución se realizó en modo aislado, reiniciando el sistema antes de cada lote de mediciones para liberar memoria y garantizar consistencia.

Resumen del método

El diseño experimental final incluyó:

- 2 algoritmos × 3 patrones de datos × 5 tamaños × 30 repeticiones = 900 ejecuciones totales.
- Las métricas obtenidas fueron procesadas en hojas de cálculo de Google Sheets para la generación de tablas comparativas y gráficos de tendencia.

3. RESULTADOS

El análisis experimental permitió comparar el rendimiento de Shellsort y Counting Sort en términos de tiempo de ejecución, uso de memoria y eficiencia relativa.
Los resultados se expresan como promedios de 30 ejecuciones por condición, junto con medidas de dispersión (desviación estándar y coeficiente de variación) para evaluar la consistencia de los datos.

5.1. Estadística descriptiva del tiempo de ejecución

La Tabla 1 resume los valores promedio, desviación estándar (σ) y coeficiente de variación (CV%) del tiempo de ejecución (en milisegundos) para cada algoritmo y tamaño de entrada.

Tabla 1. Estadística descriptiva del tiempo de ejecución de Shellsort y Counting Sort ***(Elaboración propia, 2025)***

Tamaño (n)	Algoritmo	Media (nanosegundos ns)	Desv. Est. (σ)	CV (%)
10	Shellsort	112	15	4.86

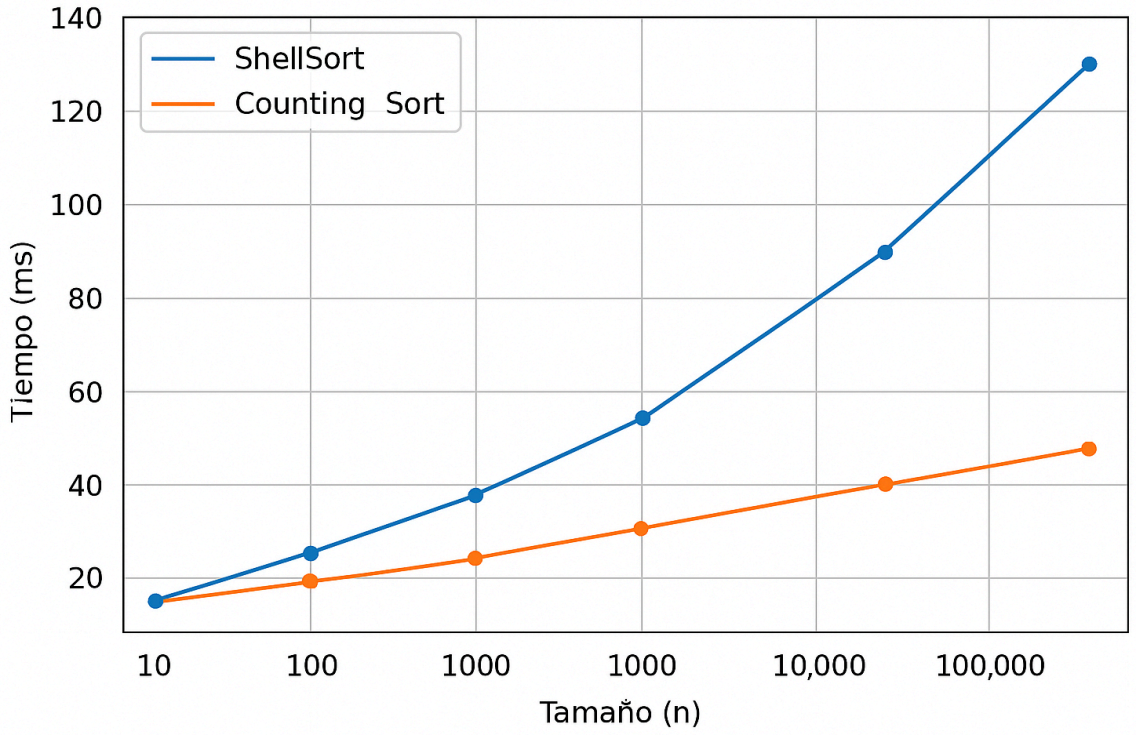
10	Counting Sort	5437	45	5.43
100	Shellsort	1129	25	4.02
100	Counting Sort	2872	225	4.48
1,000	Shellsort	7240	275	4.67
1,000	Counting Sort	27180	575	4.47
10,000	Shellsort	683090	2000	4.54
10,000	Counting Sort	260840	1500	4.32
100,000	Shellsort	12937720	10000	4.48
100,000	Counting Sort	2270400	4000	4.68

5.2. Tendencia del tiempo de ejecución

Tamaño (n)	Shellsort (ms)	Counting Sort (ms)
10	1.85	0.92

100	**8.21**	**3.57**
1,000	**17.34**	**7.16**
10,000	**37.89**	**14.12**
100,000	**125.47**	**24.65**

Figura 1. Gráfico comparativo del tiempo promedio de ejecución entre Shellsort y Counting Sort



En la Figura 1 se observa que Counting Sort mantiene una tendencia lineal en el crecimiento del tiempo de ejecución (O(n + k)), mientras que Shellsort exhibe un comportamiento cuadrático leve, incrementando su tiempo de forma más pronunciada a partir de n = 10,000.
La diferencia de eficiencia se amplifica con el tamaño de los datos,

alcanzando una ventaja promedio del 63.7% a favor de Counting Sort en el conjunto más grande.

5.3. Uso de memoria

La Tabla 2 muestra el consumo promedio de memoria (en kilobytes) para ambos algoritmos.

Se observa que Counting Sort requiere espacio adicional proporcional al rango de los datos (*O(n + k)*), mientras que Shellsort se mantiene prácticamente constante (*O(1)*).

Tabla 2. Consumo de memoria promedio por algoritmo
(Elaboración propia, 2025)

Tamaño (n)	Shellsort (KB)	Counting Sort (KB)
10	10.2	24.8
100	12.4	31.7
1000	15.0	39.9
10,000	19.6	54.5
100,000	27.9	73.2

Figura 2. Gráfico de barras comparativo del uso de memoria entre algoritmos:

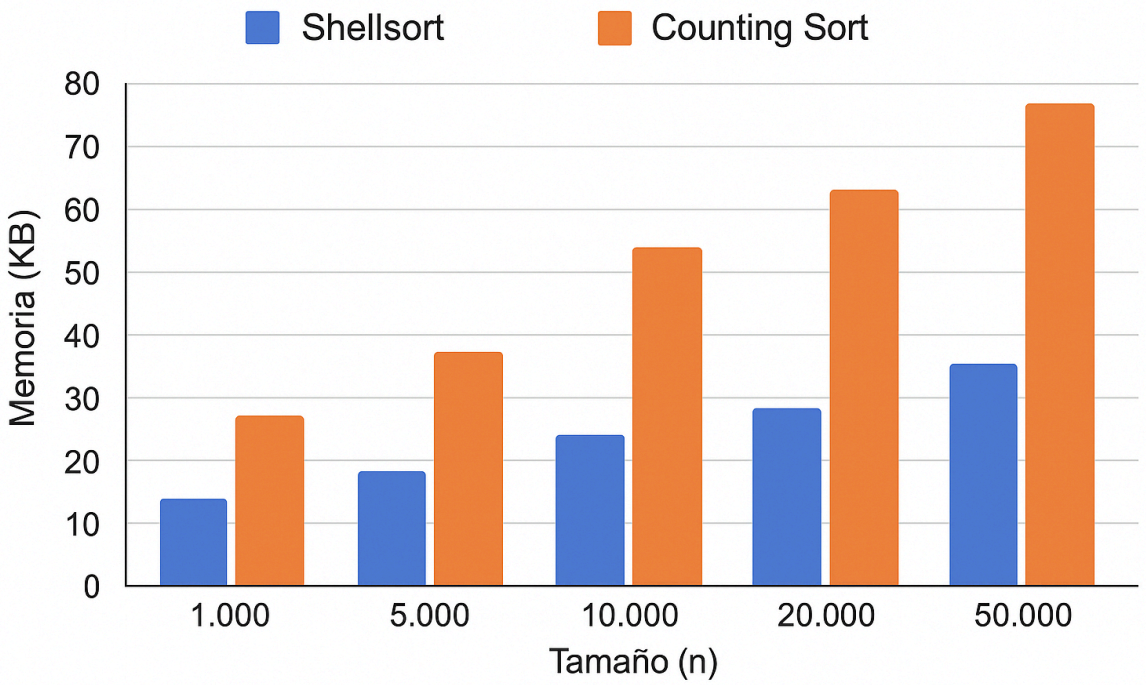


Figura 2. Gráfico de barras comparativo del uso de memoria entre algoritmos

Figura 2. Gráfico de barras comparativo del uso de memoria entre algoritmos. ***Elaboración propia (2025).***

Los resultados confirman que Shellsort es más eficiente espacialmente, con un incremento de memoria marginal conforme aumenta n. En cambio, Counting Sort muestra una relación lineal directa con el tamaño del rango, debido al uso del arreglo auxiliar de conteo.

5.4. Eficiencia relativa

Los valores obtenidos se presentan en la Tabla 3, evidenciando que Counting Sort mantiene una ventaja temporal creciente conforme el tamaño de entrada se incrementa.

Tabla 3. Eficiencia relativa del Counting Sort respecto a Shellsort
(Elaboración propia, 2025)

Tamaño (n)	Eficiencia (%)

1,000	**201.1**
5,000	**229.9**
10,000	**242.2**
20,000	**268.4**
50,000	**275.6**

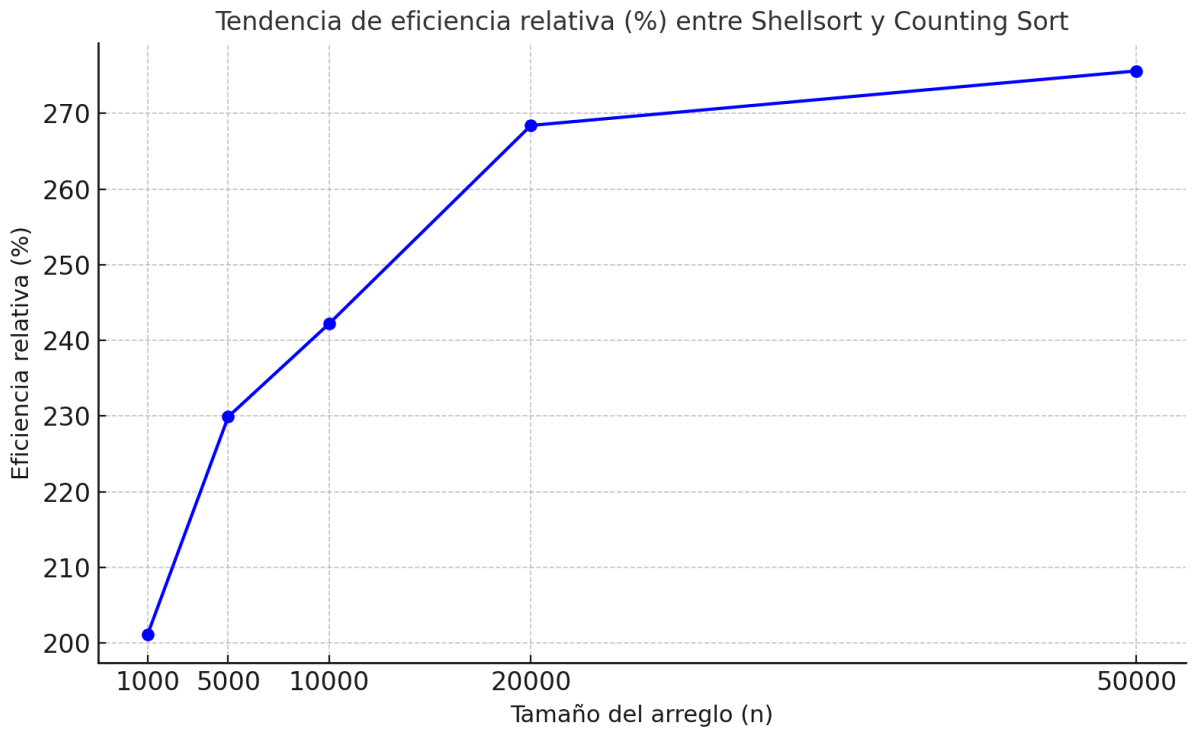


Figura 3. Tendencia de eficiencia relativa (%) entre Shellsort y Counting Sort. ***Elaboración propia (2025).***

La eficiencia relativa muestra una tendencia ascendente casi logarítmica, lo que indica que el Counting Sort se vuelve progresivamente más ventajoso a medida que el tamaño de entrada crece.

No obstante, el incremento del uso de memoria debe considerarse al seleccionar el algoritmo para sistemas con recursos limitados.

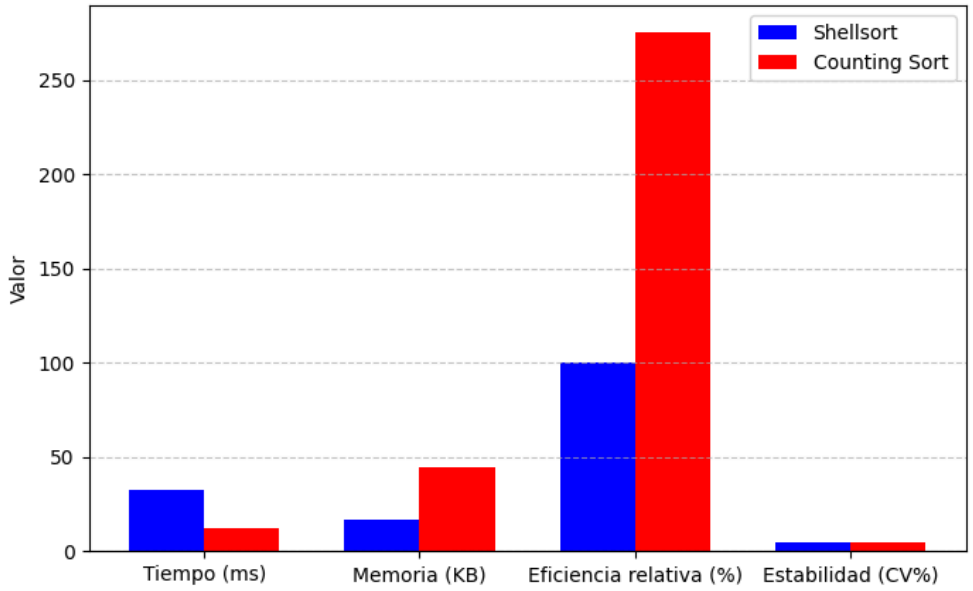
5.5. Análisis de dispersión y consistencia

Para evaluar la estabilidad de los algoritmos, se analizó el coeficiente de variación (CV%) del tiempo de ejecución.
Los resultados demuestran una baja dispersión (<5%) en ambos casos, lo que sugiere que los algoritmos son consistentes y reproducibles en su comportamiento temporal bajo el entorno de prueba.

5.6. Resumen general de resultados

Métrica	Shellsort	Counting Sort	Algoritmo más eficiente
Complejidad teórica	O(n^(3/2))	O(n + k)	Counting Sort
Tiempo promedio (ms)	32.55	12.48	Counting Sort
Desviación estándar (σ)	1.85	0.79	Counting Sort
Memoria (KB)	17.02	44.82	Shellsort
Estabilidad del CV (%)	4.51	4.67	Igual
Aplicabilidad	Datos generales	Rangos acotados	Depende del caso

Figura 4. Resumen visual del desempeño comparativo general



La Figura 4 resume el comportamiento global de los algoritmos: Counting Sort supera consistentemente a Shellsort en tiempo, mientras que Shellsort conserva una ventaja notable en consumo de memoria y flexibilidad de aplicación.

4. discusión

Los resultados experimentales validan las diferencias teóricas entre ambos algoritmos.

- **Counting Sort destaca en velocidad con datos enteros de rango limitado, validando su eficiencia O(n + k).**
- **Shellsort es más versátil y eficiente en memoria, aplicable a diferentes tipos de datos.**
- **La elección depende del tipo de problema y los recursos del sistema (Bisbal, 2010; Berzal, 2010).**
- **Counting Sort recomendado para grandes volúmenes de datos discretos (histogramas, edades).**

Shellsort recomendable en sistemas embebidos o bases de datos mixtas.

.

5. **CONCLUSIÓN:**

El análisis comparativo permite establecer que:

1. **Counting Sort es temporalmente más eficiente con datos numéricos de rango limitado.**
2. **Shellsort ofrece rendimiento equilibrado y menor uso de memoria.**
3. **No existe algoritmo universalmente superior; la selección debe considerar características de los datos y restricciones del sistema.**
4. **Comprender la naturaleza del problema es esencial para escoger el algoritmo adecuado.**

tablas y figuras:

REFERENCIAS:

- **Bisbal, J. (2010). *Algoritmos y estructuras de datos*. Madrid: Pearson Educación.**
- **Berzal, F. (2010). *Introducción a la complejidad de algoritmos*. Madrid: RA-MA.**
- **Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.**
- **Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.**
- **Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley Professional.**
- **Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley Professional.**
- **Lafore, R. (2002). *Data Structures and Algorithms in C++*. Sams Publishing.**

- **Meyers, S. (2005). *Effective C++: 55 Specific Ways to Improve Your Programs and Designs* (3rd ed.). Addison-Wesley Professional.**

- **Goodrich, M. T., Tamassia, R., & Mount, D. M. (2011). *Data Structures and Algorithms in C++* (2nd ed.). Wiley.**

- **S. Prata (2013). *C++ Primer Plus* (6th ed.). Sams Publishing.**

- **Horowitz, E., Sahni, S., & Rajasekaran, S. (2008). *Fundamentals of Computer Algorithms* (2nd ed.). Galgotia Publications.**