Comparación de algoritmos de ordenación: Merge sort y Shell sort

Comparison of Sorting Algorithms: Merge Sort and Shell Sort

Samuel Alexander Gomez Salcedo
sgomezs@unjbg.edu.pe
Universidad Nacional Jorge Basadre Grohmann
Tacna, Perú
Lucía de los Ángeles Angulo Yugra
languloy@unjbg.edu.pe
Universidad Nacional Jorge Basadre Grohmann
Tacna, Perú
Anghelo del Piero Calderon Morales
adcalderonm@unjbg.edu.pe
Universidad Nacional Jorge Basadre Grohmann
Tacna, Perú

Resumen

El impacto que las computadoras y la informática han tenido en todos los aspectos de la sociedad ha incrementado el interés por la capacidad de desarrollar, analizar e implementar algoritmos. En este contexto, el presente artículo tiene como objetivo realizar un análisis estadístico comparativo del tiempo de ordenación entre los algoritmos Merge Sort, con una complejidad temporal de O(n log n), y Shell Sort, cuya complejidad varía entre O(n log² n) y O(n²), ambos considerados métodos algorítmicos de ordenamiento. Para evaluar su desempeño, se realizaron 30 pruebas por condición con arreglos de 1,000, 10,000 y 100,000 datos, aplicando ambos algoritmos. Los resultados permiten concluir que, estadísticamente, existen diferencias significativas entre los tiempos de ordenación de los algoritmos mencionados, siendo Merge Sort el que presenta mejor desempeño en promedio.

Palabras clave: Merge sort, Shell sort, Algoritmos de ordenamiento, metodos algoritmicos

Abstract

This study presents a comparative analysis of the performance of the Merge Sort and Shell Sort algorithms, considering their theoretical time complexity and empirical behavior. Merge Sort exhibits a complexity of O(n log n), whereas Shell Sort ranges between O(n log² n) and O(n²) depending on the sequence of increments used. To evaluate their efficiency, 30 experimental trials were conducted for each condition using datasets of 1,000, 10,000, and 100,000 elements, measuring the sorting time in each case. The results indicate statistically significant differences between both algorithms, showing that Merge Sort achieves a better average performance than Shell Sort, particularly with larger data volumes. These findings confirm the relationship between theoretical complexity and practical behavior of sorting algorithms, providing quantitative evidence of their relative efficiency.

Keywords: Merge Sort, Shell Sort, Sorting Algorithms, Algorithmic Methods

1 Introducción:

El hábito de ordenar se ha vuelto parte de nuestras vidas; el que uno sea más o menos hábil realizando pequeñas tareas cotidianas está directamente relacionado con que las cosas implicadas estén más o menos ordenadas. La razón explica que es más sencillo buscar algo cuando los objetos están ordenados (Instituto de Investigación Tecnológica, s.f.). En el mundo de la computación, la ordenación es una operación fundamental para la eficiencia del procesamiento de datos. Su tarea permite la organización rápida de información. El estudio de métodos de ordenación de este artículo se basó en dos algoritmos conocidos: Merge Sort, que ofrece buen rendimiento y estabilidad, ideal para listas extensas, y Shell Sort, que brinda un buen equilibrio entre simplicidad y eficiencia para listas de tamaño medio.

Anteriormente se estudió el algoritmo Shell Sort y su rendimiento con diferente cantidad de datos; el método Merge Sort no se estudió lo suficiente. Sin embargo, en este artículo se investigaron ambos métodos para realizar una comparación significativa de sus diferencias y las ventajas y desventajas de cada uno.

El propósito de este artículo es comparar el rendimiento de los algoritmos de ordenación Merge Sort y Shell Sort, evaluando sus fortalezas y debilidades en distintos contextos de uso. La comparación se realizó utilizando métricas clave como:

- **Tiempo de ejecución**: para medir la velocidad con la que cada algoritmo ordena conjuntos de datos de diferentes tamaños.
- **Uso de memoria:** para analizar la eficiencia en el consumo de recursos, especialmente en sistemas con limitaciones de almacenamiento.
- **Estabilidad del algoritmo:** para determinar si preservan el orden relativo de elementos iguales.
- **Complejidad computacional:** para entender su comportamiento teórico en el mejor, peor y caso promedio.

2 Metodología:

Para el desarrollo del trabajo se basó en un estudio cuantitativo y comparativo para comprobar el tiempo de ejecución de los algoritmos de ordenación se utilizó la siguiente metodología:

- Investigación documental
- Experimentación con los algoritmos de ordenamiento

- Análisis de los tiempos de ejecución del ordenamiento de elementos de los dos métodos de ordenación Merge Sort y Shell Sort
- Interpretación de resultados
- Conclusión

2.1 Desarrollo:

- Implementación y entorno: Los algoritmos de ordenación seleccionados fueron codificados en el lenguaje C + + utilizando el programa DEV C + + para su comparación.
  Uno de los algoritmos es Shell sort la cual tiene una complejidad de O(nlogn) que sería el mejor caso en cuando el arreglo está casi ordenado, O(n^(3/2)) en caso de datos aleatorios y O(n^2) la cual depende de una secuencia de incrementos usada.
  El otro es Merge sort la cual tiene una complejidad de O(nlogn) la que tiene es la misma complejidad en el mejor, promedio y el peor de los casos, lo que nos dice que merge sort es estable y no empeora.
- Datos a registrar: Se registró los tiempos de ejecución con precisión utilizando una función de medición, la cual podía medir entre milisegundo, microsegundo y nanosegundo. Y esos resultados fueron a almacenar dentro de una hoja de cálculo en excel para un análisis estadístico.
- Datos: Se generaron diferentes textos con diferente cantidad de elementos entre 1000, 10000, 100000 elementos en un arreglo y tre tipos de escenario de entrada para analizar los métodos dentro de diferentes escenarios: Aleatorios, ordenados ascendentemente y ordenados decrecientemente
- Entorno de ejecución: Los algoritmos fueron ejecutados en una computadora con las siguiente características:
  Intel(R) Core(TM) i5-6200U
  CPU a 2.30GHz
  8 GB Memoria RAM
  Sistema operativo de 64 bits
  Procesador x64

3 Resultados:

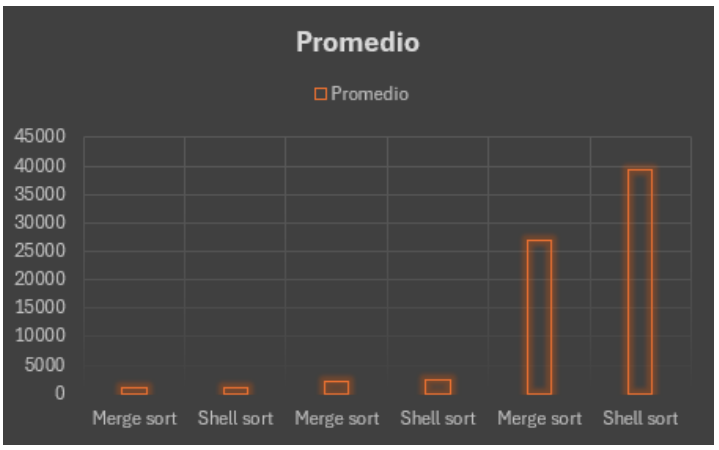
Los resultados experimentales mostraron diferencias significativas en el tiempo de ejecución de los algoritmos Merge Sort y Shell Sort bajo distintos patrones de datos y tamaños de entrada.

En el caso del patrón aleatorio, el algoritmo Merge Sort presentó tiempos promedio de 1024.3 µs, 2035.3 µs y 27012.13 µs para n = 1000, 10000 y 100000 respectivamente. Por su parte, Shell Sort mostró valores mayores de 2530.5 µs, 21355.1 µs y 39212.53 µs, evidenciando una menor eficiencia para grandes volúmenes de datos, que se muestra en la *Tabla 1*.

Tabla 1: Datos estadísticos de Merge y Shell sort en una patrón aleatorio.

Algoritmo	Patron	n	Promedio	Mediana	Desviacion Estandar	Min	Max
Merge sort	Aleatorio	1000	1024.3	995.5	13.96	997	1035
Shell sort	Aleatorio	1000	1024.83	1029.5	72.51	966	1165
Merge sort	Aleatorio	10000	2035.3	1995	199.18	1993	3088
Shell sort	Aleatorio	10000	2530.5	2135.5	618.86	1952	3992
Merge sort	Aleatorio	100000	27012.13	25909	4065.29	23884	40889
Shell sort	Aleatorio	100000	39212.53	36866.5	6691.25	33949	63211

Figura 1: Gráfico de tiempo de 1000, 10000 y 100000 datos aleatorios.

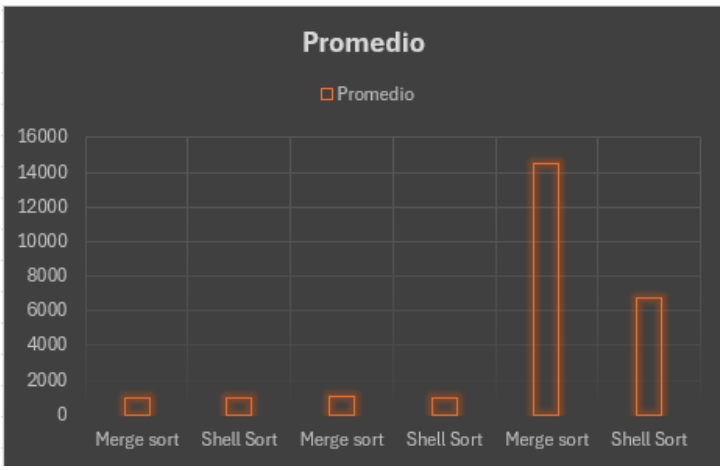


Para el patrón ascendente, ambos algoritmos mostraron un rendimiento más estable. Merge Sort mantuvo tiempos promedio entre 996.4 µs y 14498.3 µs, mientras que Shell Sort presentó resultados entre 994 µs y 6758.13 µs. Esto sugiere que Shellsort se ve favorecido cuando los datos ya están parcialmente ordenados, reduciendo la cantidad de comparaciones e intercambios, que se ve en la *Tabla 2*.

Tabla 2: Datos estadísticos de Merge y Shell sort en un patrón ascendente.

Algoritmo	Patron	n	Promedio	Mediana	Desvicion Estandar	Min	Max
Merge sort	Ascendente	1000	996.4	997	1.51	994	998
Shell Sort	Ascendente	1000	994	994	0	994	994
Merge sort	Ascendente	10000	1079.1	997.5	260.34	958	2034
Shell Sort	Ascendente	10000	1004.05	997	20.64	959	1040
Merge sort	Ascendente	100000	14498.3	13985.5	1736.57	12924	19084
Shell Sort	Ascendente	100000	6758.13	6941.5	963.63	5981	10973

Figura 2: Gráfico de tiempo de 1000, 10000 y 100000 datos ascendentes.

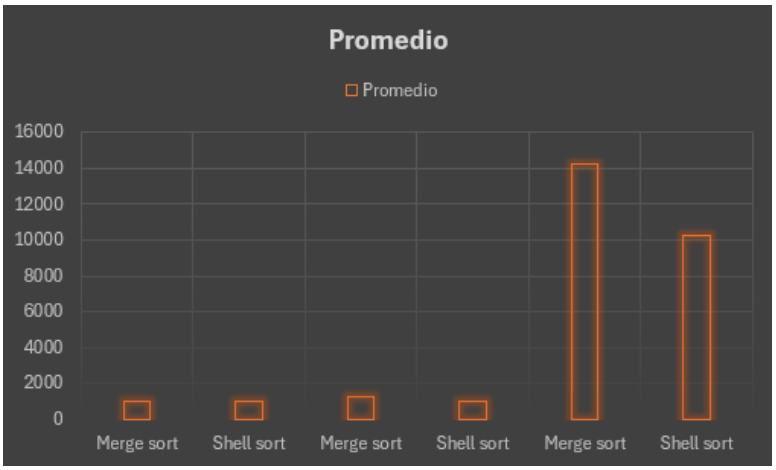


En el patrón descendente, los resultados indican que Merge Sort conserva una tendencia de crecimiento más controlada con el tamaño de n (1036 µs, 1268.4 µs y 14222.25 µs), mientras que Shell Sort presenta tiempos menores para n pequeñas pero más irregulares a gran escala (988.5 µs, 1001.59 µs y 10252 µs). En general, Merge Sort demostró mayor estabilidad y consistencia ante cambios en el patrón de entrada, que se muestra en la *Tabla 3*.

Tabla 3: Datos estadísticos de Merge y Shell sort en un patrón descendente.

Algoritmo	Patron	n	Promedio	Mediana	Desviacion Estandar	Min	Max
Merge sort	Descendente	1000	1036	1036	0	1036	1036
Shell sort	Descendente	1000	988.5	988.5	10.6	981	996
Merge sort	Descendente	10000	1268.4	998	453.24	992	2039
Shell sort	Descendente	10000	1010.59	997	43.72	991	1197
Merge sort	Descendente	100000	14222.25	13979.5	505.52	13912	16071
Shell sort	Descendente	100000	10252	9975	456.79	9642	11152

Figura 3: Gráfico de tiempo de 1000, 10000 y 100000 datos descendentes.



En cuanto a la desviación estándar, se observó que Mergesort mantiene menor variabilidad en los tiempos cuando n es pequeño, pero Shell Sort presenta fluctuaciones más marcadas al aumentar el tamaño de los datos, reflejando su mayor dependencia de la distribución inicial de los elementos.

4 Discusión:

Los resultados concuerdan con el análisis teórico de la literatura (Cormen et al., *Introduction to Algorithms*, 2009; Sedgewick & Wayne, *Algorithms*, 2011), donde se establece que Merge Sort posee una complejidad temporal promedio y peor caso de

O(n log n), mientras que Shell Sort varía entre O(n log² n) y O(n²) dependiendo de la secuencia de incrementos empleada.

El mejor comportamiento de Merge Sort en conjuntos grandes y aleatorios se debe a su estructura divide y vencerás, que garantiza un número estable de comparaciones y movimientos de datos independientemente de su orden inicial. En contraste, Shell Sort es más eficiente para listas parcialmente ordenadas o de tamaño moderado, pero su rendimiento decrece con listas extensas y desordenadas debido a la cantidad variable de comparaciones en cada pasada.

Asimismo, la estabilidad observada en Merge Sort lo hace más adecuado para aplicaciones donde se requiere preservar el orden relativo de los elementos y donde el tamaño de los datos es considerable. Por otro lado, Shell Sort, al no requerir memoria adicional significativa, puede ser preferible en sistemas con recursos limitados o cuando el tamaño de entrada no es muy grande.

En síntesis, los resultados experimentales respaldan la teoría algorítmica clásica: Merge Sort es más eficiente y consistente para grandes volúmenes de datos, mientras que Shell Sort ofrece ventajas en casos más pequeños o parcialmente ordenados.

5 Conclusiones:

Merge Sort es una excelente opción cuando se requiere estabilidad en la ordenación, ya que conserva el orden relativo de los elementos iguales. Además, su rendimiento consistente de O(n·log n) lo hace predecible y confiable, ideal para aplicaciones donde la consistencia del rendimiento es importante.

Shell Sort resultó ser una opción eficiente para conjuntos de datos pequeños o casi ordenados, ya que su estrategia de comparación entre elementos distantes permitió reducir significativamente la cantidad de intercambios necesarios. Esta característica le otorgó una ventaja en escenarios donde otros algoritmos más complejos no ofrecían mejoras sustanciales en el rendimiento.

6 Referencias:

Instituto de Investigación Tecnológica. (s.f.). *Algoritmos de ordenación (I)*. https://www.iit.comillas.edu/documentacion/revista/IIT-01-156R/Algoritmos_de_ordenaci%c3%b3n_%28I%29.pdf

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.