Investigación Ordenamiento por Mezcla Directa

Bonilla Caiza David Alejandro

Andrade Chicaiza Julio Aaron

Quiroz Herrera María Laura

Tiupul Guamán Danny Alexander

Universidad de las Fuerzas Armadas ESPE

28436: Estructuras de Datos

Ing. Washington Loza H. Mgs.

28 de noviembre del 2025

Índice

Tabla de Ilustraciones

Introducción

La presenta investigación hablara sobre el ordenamiento por mezcla directa también conocido como Merge Sort, uno de los algoritmos de ordenamiento mas usados actualmente para ciencia de datos ya que es eficiente, estable y se comporta de forma predecible incluso con una gran cantidad de datos. Su funcionamiento aplica el principio de Divide y Vencerás, donde se fragmentan estos datos en pequeñas porciones, se las ordena y se las vuelve a juntar, haciendo que tenga un rendimiento consistente a O(n log n), y ofreciendo una arquitectura lógica clara y fácil de entender. Abordaremos el concepto general, como funciona internamente, sus ventajas generales, entre otras cosas que ayudaran a gran medida entender cómo funciona este algoritmo.

Desarrollo

Concepto General del Algoritmo

El Ordenamiento de mezcla directa o "Merge Sort" es un algoritmo de ordenamiento que usa comparaciones, se caracteriza principalmente porque es estable, en pocas palabras eso significa que, si existen elementos iguales en la lista original, su orden se mantiene tras ordenar. (kartik, 2025)

En cuanto a su complejidad tenemos que el tiempo en el peor caso, promedio y "normal" es $O(n \log n)$, con n siendo el número de elementos. (Anonimo, 2025)

Cómo Funciona: Idea Base del "Divide y Vencerás"

En forma de resumen, su funcionamiento es simple, divide una matriz de datos en 2 mitades, las ordena recursivamente y al final las fusiona para obtener la matriz ordenada. (kartik, 2025)

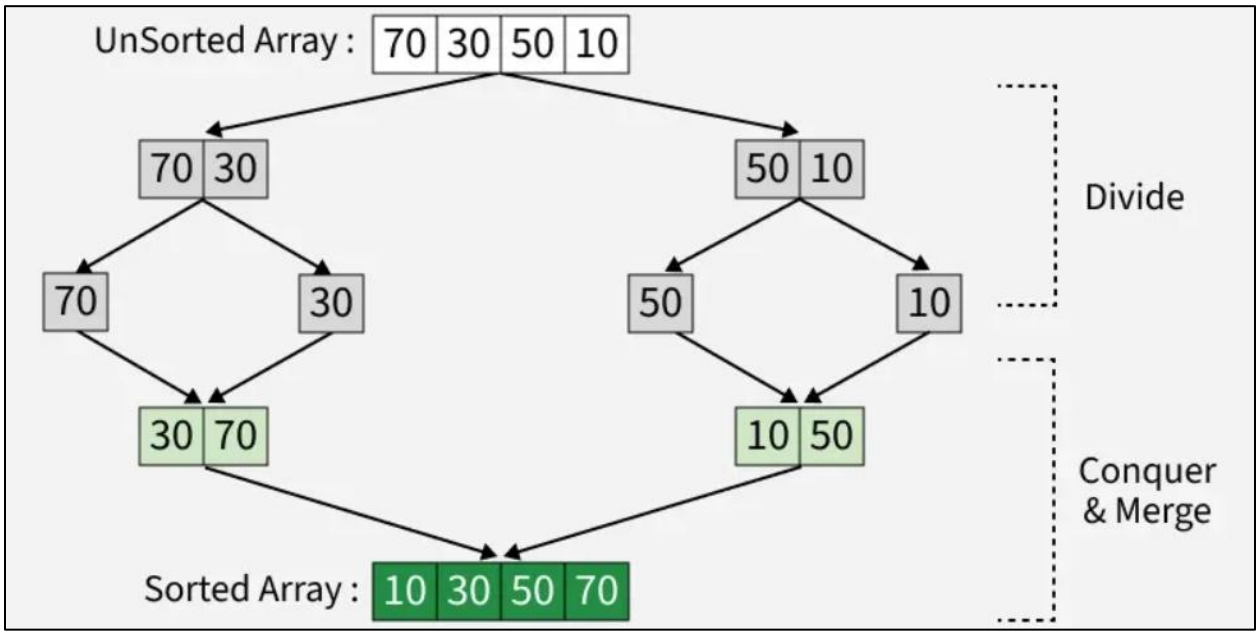

Ilustración 1. Ordenamiento de un Arreglo de Números Usando Merge Sort

La estrategia de dividir y conquistar se basa en los siguientes 3 puntos:

1. **Dividir:** Divide el conjunto de datos en dos mitades.
2. **Conquistar:** Cada mitad se ordena usando el mismo algoritmo.
3. **Combinar:** Una vez que las sublistas se ordenan, se fusionan en una sola.

Este concepto es simple, pero muy potente cuando se trata de ordenar una cantidad extensa de datos, ya que es más fácil ordenarlos por grupos grandes ya ordenados, que ordenar un conjunto grande desordenado.

Proceso Paso a Paso

Hasta ahora la explicación de cómo funciona ha sido muy sencilla, ahora profundizaremos en cómo funciona internamente este algoritmo:

Verificar caso base

Se verifica que la lista de elementos no sea únicamente de 1 o 0 elementos.

División del arreglo

Si la lista tiene más de un elemento, calcular un punto medio; dividir la lista en dos sublistas ("izquierda" y "derecha"), de tamaño aproximadamente igual.

Llamada recursiva

Se aplica la llamada recursiva de ordenamiento en ambas partes, izquierda y derecha.

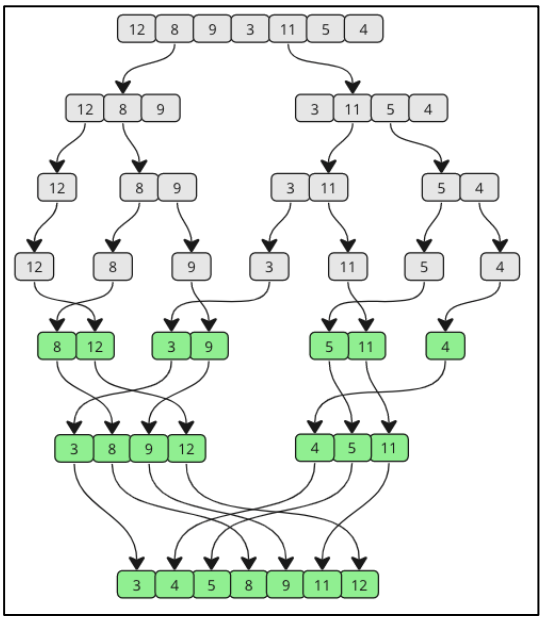

Ilustración 2. Paso a paso ilustrado de Merge Sort

Mezcla ordenada

Usamos dos punteros (uno por cada sublista), comparamos sus elementos actuales, tomamos el menor (suponiendo orden ascendente), lo copiamos al arreglo resultante, avanzamos el puntero correspondiente. Se repite este proceso hasta que la sublista se agote y luego copiamos los elementos restantes de la otra sublista.

Resultado Final

Después de que en niveles sucesivos de recursión se hayan realizado divisiones + merges, la lista original queda ordenada.

Implementación

La implementación de Merge Sort en C++ consiste en dividir y mezclar mediante pasos recursivos. De acuerdo con Goodrich, Tamassia y Goldwasser (2011), el algoritmo se estructura en funciones separadas: una que divide el arreglo y otra que se encarga de fusionar los elementos. El empleo de arreglos auxiliares durante la mezcla ayuda a mantener a un algoritmo estable y garantizar un ordenamiento lógico adecuado.

Segun Cormen, Leiserson, Rivest y Stein (2009) señalan que se emplean arreglos temporales con la finalidad de almacenar los datos durante la fase de mezcla, lo que asegura el orden relativamente igual en los elementos, lo que ayuda con la estabilidad del algoritmo.

Pseudocódigo estándar

```
procedure MezclaDirecta(F: archivo de N registros)
    crear lista de archivos temporales: runs = []
    while NO fin-de-archivo(F) do
        leer hasta M registros desde F → buffer[]
        ordenar buffer[] internamente (por ejemplo con MergeSort o QuickSort)
        crear un nuevo archivo temporal T
        escribir buffer[] ordenado en T
        agregar T a runs
    while (runs.size > 1) do
        tomar los dos primeros archivos de runs: A, B
        crear un nuevo archivo C (salida de la mezcla)
        abrir punteros de lectura para A y B
        leer primer registro de A → a, y de B → b
        while (A no vacío) y (B no vacío) do
            if a.dato ≤ b.dato then
                escribir a en C
                leer siguiente de A → a
            else
                escribir b en C
                leer siguiente de B → b
mientras (A no vacío) do escribir todo A en C
        mientras (B no vacío) do escribir todo B en C
        cerrar A, B
        borrar (o archivar) A y B
        agregar C al final de runs
renombrar runs[0] como F_ordenado
    return F_ordenado
```

Análisis de Complejidad

El algoritmo se basa en la misma idea del merge sort, pero aplicado a gran escala. Primero divide los datos en varios archivos o bloques más pequeños que sí pueden ordenarse en memoria. Cada bloque se ordena por separado y luego todos se van mezclando nuevamente hasta obtener un único archivo final ordenado.

De esta manera no depende de la Ram para almacenar todos los datos, utiliza el disco para los archivos temporales y en el peor de los casos que es la lectura y escritura el algoritmo está diseñado para minimizar las interacciones.

Tiempo (mejor, promedio, peor caso)

En cuanto al rendimiento, su complejidad temporal es O(n log n). Cada registro se lee y se escribe una vez durante la creación de los bloques iniciales, y después se vuelve a procesar durante las etapas de mezcla. Tanto en el mejor como en el peor caso, la complejidad se mantiene en n log n, por lo que el comportamiento del algoritmo es estable en cualquier situación.

Espacio

Merge Sort no es un algoritmo in-place, requiere memoria adicional. Para un arreglo de n elementos el algoritmo necesita O(n) memoria auxiliar para las sablistas durante la mezcla, porque al combinar, crea estructuras temporales y luego las reinserta en el arreglo original.

Ventajas y Desventajas

Ventajas

- **Estabilidad:** No altera el orden de los elementos iguales, por lo que resulta util para ordenar registros con claves repetidas.
- **Complejidad garantizada:** A diferencia de otros algoritmos como Quick Sort no depende del orden inicial de los datos. Su rendimiento consiste en: Mejor de los casos = Promedio = Peor de los casos = O(n log(n)).
- **Excelente para listas enlazadas:** En listas la división se hace con punteros, la mezcla se hace sin copiar enormes bloques de memoria.

- **Óptimo para datos muy grandes en almacenamiento externo:** Se puede usar para ordenamiento externo (archivos que no caben en RAM), dividiendo en bloques que sí caben.

Desventajas

- **Alto uso de memoria:** Necesita **O(n)** memoria adicional, lo cual es costoso en entornos con RAM limitada.
- **No es in-place:** No modifica en la misma estructura, por lo que es menos eficiente en pequeños dispositivos o sistemas integrados.
- **Costoso para arreglos pequeños:** Para arreglos pequeños, Insertion Sort suele ser más veloz por menor sobrecarga.

Aplicaciones Prácticas y Cuándo Usarlo

Cuando necesitas un algoritmo estable

Ejemplos reales:

- Ordenar registros con campos múltiples (por apellido y luego por nombre).
- Ordenar facturas por fecha donde puede haber duplicados.

Cuando manejas listas enlazadas

Merge Sort evita mover grandes bloques de memoria y solo ajusta punteros lo cual resulta muy eficiente.

Cuando trabajas con archivos muy grandes

En ordenamiento externo, se divide el archivo en fragmentos que se ordenan individualmente y luego se mezclan:

- bases de datos,
- archivos de logs enormes,
- procesamiento de Big Data.

Cuando requieres rendimiento garantizado

- Si necesitas evitar caídas a $O(n^2)$, como en QuickSort.

Conclusiones

Se concluye que el ordenamiento por mezcla directa es un algoritmo eficiente y estable, ideal cuando se necesita un rendimiento constante sin importar el estado inicial de los datos.

También se concluye que su enfoque de divide y vencerás simplifica el problema y permite ordenar listas grandes de manera rápida y estructurada.

Recomendaciones

Se aconseja usar este algoritmo cuando se trabaje con grandes volúmenes de datos o cuando la estabilidad del orden sea un requisito importante.

No es recomendable usarlo en sistemas con memoria limitada, ya que su proceso de mezcla requiere espacio adicional durante la ejecución.

Referencias

Anonimo. (16 de Nov de 2025). *Merge Sort*. Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Merge_sort?utm_source=chatgpt.com

kartik. (03 de Oct de 2025). *Merge Sort*. Obtenido de Geeksforgeeks: https://www.geeksforgeeks.org/dsa/merge-sort/

Efficient Merge Sort | Sorting Made Easy. (s. f.). En *AlgoWalker*. https://www.algowalker.com/merge-sort.html AlgoWalker

Merge Sort – Data Structure and Algorithms Tutorials. (s. f.). En *DSLearning*. https://yashchaturvediir.github.io/DSLearning/Merge.html