COMPARATIVA DE ALGORITMOS

Merge Sort frente a Intercambio Bidireccional

Autores

Este estudio es realizado por **Tapia Paco, Maricielo Luz; Mamani Genix, Amidwar Jhoel; y Valdivia Vela, Wilfredo Junior** en la Universidad Nacional Jorge Basadre Grohmann, Tacna - Perú, enfatizando la importancia de los algoritmos de ordenamiento en la computación moderna.

Importancia de los algoritmos de ordenamiento

Algoritmos en la computación

Los **algoritmos de ordenamiento** son fundamentales en la computación moderna, facilitando el manejo eficiente de grandes volúmenes de datos.

Eficiencia en el procesamiento

La **eficiencia en el procesamiento** de datos es un desafío clave, afectando el rendimiento de diversas aplicaciones y sistemas informáticos en la actualidad.

Objetivos del estudio

Este estudio se centra en **comparar el rendimiento** de Merge Sort y Cocktail Shaker Sort, evaluando su eficacia en diferentes patrones de datos.

Descripciones del Algoritmo

El principio "Divide y Vencerás" en Merge Sort

Estrategia de división

Merge Sort utiliza la estrategia de "Divide y Vencerás", separando la lista en sublistas más pequeñas hasta que sean manejables.

Estabilidad y eficiencia

Este algoritmo es estable y eficiente, lo que significa que puede manejar grandes volúmenes de datos sin perder el orden original.

Diagrama de flujo

Un flujo de diagrama ilustra la división recursiva y la fusión efectiva de las sublistas, mostrando cómo se reconstruye la lista ordenada.

Cocktail Shaker Sort: An Efficient Approach

Bidirectional Variation Explained

The Cocktail Shaker Sort is a bidirectional sorting algorithm that improves upon the Bubble Sort by performing simultaneous forward and backward passes through a list.

Suitable for Small Lists

This algorithm is particularly effective for small or nearly sorted datasets, where it can outperform more complex algorithms due to reduced overhead.

Visualizing the Process

A flow diagram illustrates the forward and backward passes of the Cocktail Shaker Sort, highlighting its efficiency in repositioning elements quickly.

Metodología

Implementación de Algoritmos en C++

Detalles de Codificación

La implementación se realizó utilizando C++ en Visual Studio Code, permitiendo un ambiente controlado y eficiente para ejecutar pruebas de rendimiento.

Tamaños de Conjuntos de Datos

Se probaron conjuntos de datos con tamaños de 1,000; 10,000; y 100,000 elementos, proporcionando una amplia gama para evaluar el rendimiento de ambos algoritmos.

Tipos de Datos Evaluados

Se analizaron diferentes tipos de datos, incluyendo aleatorios, ordenados ascendentemente y ordenados descendentemente, para observar la variabilidad del rendimiento de cada algoritmo.

Medición de Tiempos de Ejecución en Nanosegundos

Importancia de la Precisión

La **precisión** en la medición del tiempo es crucial para evaluar el rendimiento de los algoritmos de ordenamiento.

Métodos de Registro

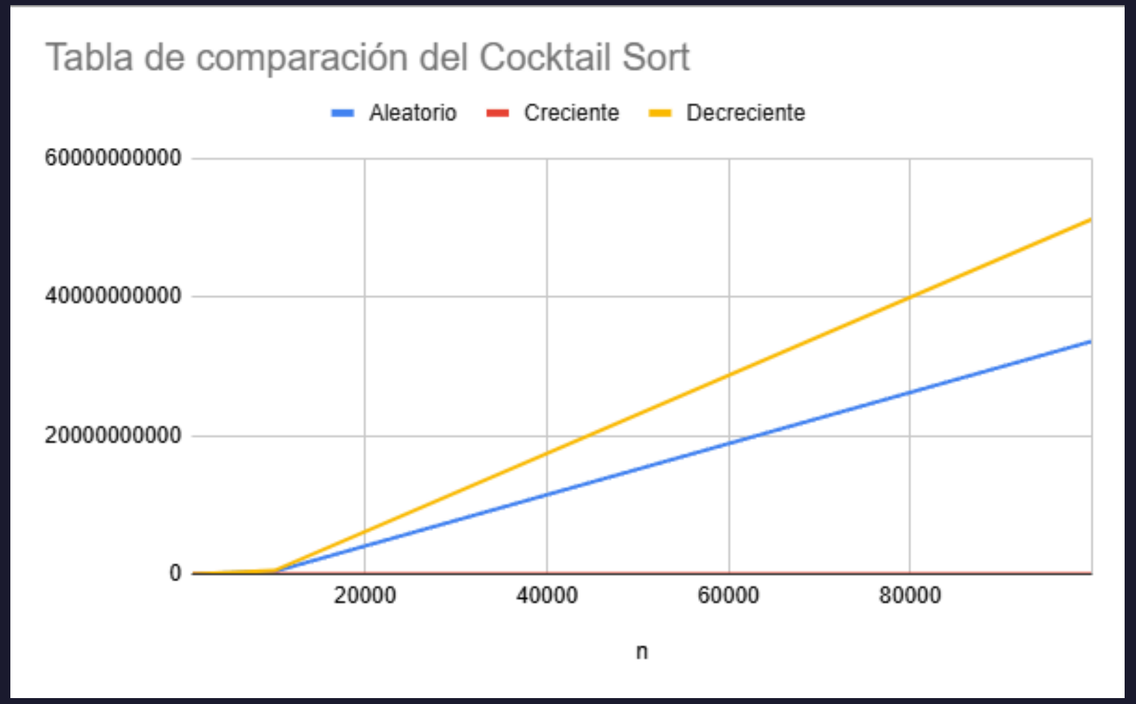
Se utilizan técnicas avanzadas para registrar el tiempo de ejecución en nanosegundos, garantizando alta resolución y confiabilidad en los datos recolectados.

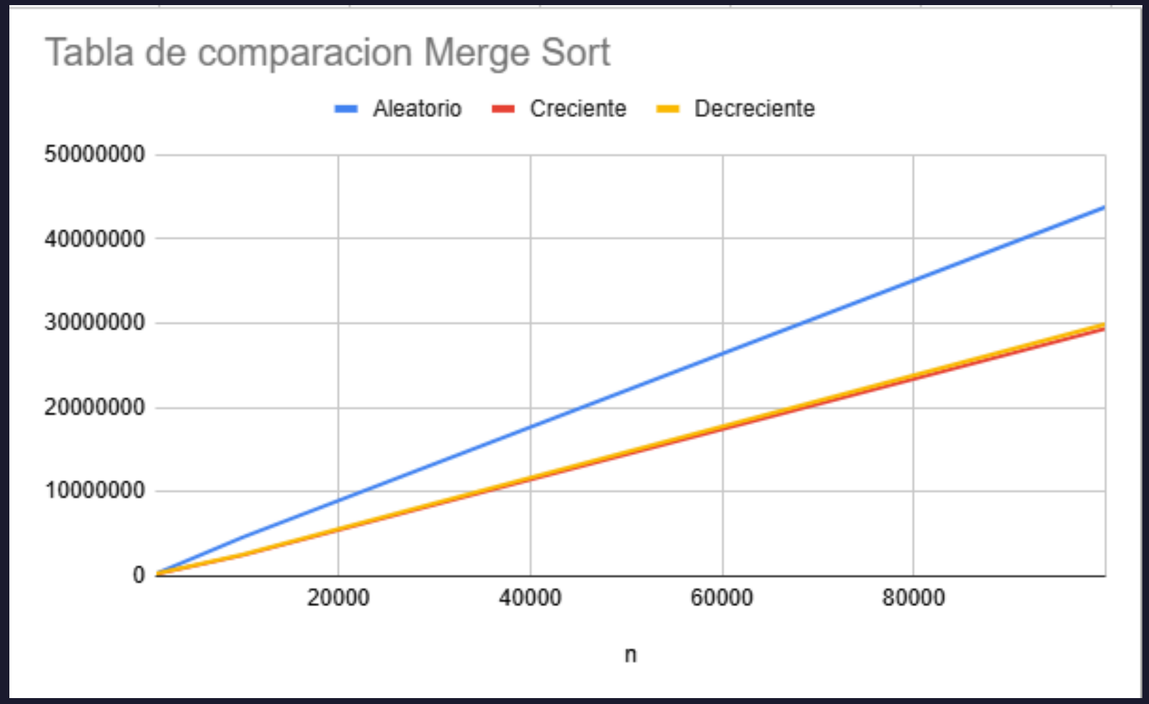
Comparación de Resultados

Al comparar diferentes algoritmos, la precisión de la medición permite analizar diferencias mínimas en su rendimiento y eficiencia de manera efectiva.

Resultados y Impacto

Comparación del Rendimiento de Algoritmos





Comparación del Rendimiento de Algoritmos

Eficiencia de Merge Sort

Merge Sort mostró un rendimiento **consistente y superior** en todas las pruebas realizadas, manejando bien grandes conjuntos de datos.

Variabilidad del Cocktail Shaker Sort

Cocktail Shaker Sort demostró una **eficiencia variable** según el patrón de datos, siendo menos efectivo en listas desordenadas.

Implicaciones Prácticas

La elección del algoritmo afecta **directamente la eficiencia** del sistema, siendo crucial considerar el tamaño y patrón de datos al seleccionar.

Comparación de Tiempos de Ejecución

Cocktail Shaker Sort

Este algoritmo muestra un rango de tiempos de ejecución **amplio**; desde 2,980 ns en listas ordenadas ascendentemente hasta 5,895,390 ns en listas ordenadas descendentemente.

Merge Sort

Merge Sort presenta tiempos de ejecución **más estables** entre 262,000 ns y 309,870 ns, mostrando eficiencia constante en todos los patrones de datos evaluados.

Eficiencia en Datos Grandes

Para listas de 100,000 elementos, Merge Sort es **significativamente más eficiente**, resaltando la importancia de elegir el algoritmo adecuado según las características de los datos.

Conclusiones sobre los Algoritmos de Ordenamiento

Recomendación para grandes datasets

Merge Sort es recomendado para conjuntos de datos grandes y variados debido a su eficiencia y estabilidad en el rendimiento.

Limitaciones de Intercambio Bidireccional

Cocktail Shaker Sort es solo adecuado para listas pequeñas o casi ordenadas, donde su rendimiento es más competitivo.

Impacto de la elección del algoritmo

La **elección del algoritmo** afecta directamente la eficiencia del sistema, lo que resalta la importancia de un análisis adecuado.

GRACIAS