# Algoritmo de ordenamiento rápido

Estudiante: Fernando Matamoros Acuña

Estructuras de Datos 1

Profesor: Romario Salas Cerdas

Universidad CENFOTEC

Diciembre 2025

# Funcionamiento del algoritmo

El algoritmo Quick Sort (ordenamiento rápido) es uno de los algoritmos de ordenamiento más eficientes y ampliamente utilizados en la práctica. Fue desarrollado por Tony Hoare en 1960 y se basa en el paradigma de "divide y vencerás" (divide and conquer).

El funcionamiento del algoritmo se puede resumir en los siguientes pasos:

- Selección del pivote: Se elige un elemento del arreglo como "pivote". Existen varias estrategias para seleccionar el pivote, siendo las más comunes: elegir el primer elemento, el último elemento, el elemento del medio, o un elemento aleatorio. En la implementación desarrollada se utiliza el último elemento del arreglo como pivote (estrategia de partición de Lomuto).
- Partición: Se reorganiza el arreglo de manera que todos los elementos menores o iguales al pivote queden a su izquierda, y todos los elementos mayores al pivote queden a su derecha. El pivote queda en su posición final correcta.
- Recursión: Se aplica recursivamente el mismo proceso a los sub-arreglos formados a la izquierda y derecha del pivote. Cada llamada recursiva ordena un sub-arreglo más pequeño.
- Caso base: Cuando un sub-arreglo tiene 0 o 1 elementos, se considera que ya está ordenado y se detiene la recursión.

La eficiencia del algoritmo radica en que, en promedio, divide el problema en dos partes aproximadamente iguales, lo que resulta en una complejidad temporal de O(n log n) en el caso promedio.

# Escenarios o casos de uso en los que se destaca el uso del algoritmo

Quick Sort es especialmente útil en los siguientes escenarios:

- Ordenamiento de grandes volúmenes de datos: Debido a su eficiencia en el caso promedio, Quick Sort es ideal para ordenar grandes conjuntos de datos donde se requiere un buen rendimiento.
- Aplicaciones de software de sistema: Muchos sistemas operativos y compiladores utilizan variantes de Quick Sort para ordenar estructuras internas, como la organización de procesos, la gestión de memoria, o la ordenación de símbolos en tablas.
- Bases de datos: Los sistemas de gestión de bases de datos frecuentemente emplean Quick Sort o algoritmos derivados para ordenar resultados de consultas y optimizar operaciones de búsqueda.
- Aplicaciones con datos aleatorios: Cuando los datos no están previamente ordenados y tienen una distribución aleatoria, Quick Sort muestra excelente rendimiento.
- Ordenamiento in-place: Quick Sort puede ordenar el arreglo sin necesidad de memoria adicional significativa (solo requiere espacio para la pila de recursión), lo que lo hace eficiente en términos de memoria.
- Librerías estándar: Muchas implementaciones de librerías estándar de lenguajes de programación (como C++ std::sort) utilizan variantes híbridas que combinan Quick Sort con otros algoritmos para garantizar rendimiento óptimo.

# Ventajas y desventajas del algoritmo

## Ventajas

- Eficiencia en el caso promedio: Quick Sort tiene una complejidad temporal de O(n log n) en el caso promedio, lo que lo hace uno de los algoritmos más rápidos para ordenamiento general.
- Ordenamiento in-place: A diferencia de algoritmos como Merge Sort, Quick Sort ordena el arreglo en su lugar original, requiriendo solo espacio adicional O(log n) para la pila de recursión.
- Buena localidad de referencia: El algoritmo accede a los elementos del arreglo de manera secuencial, lo que aprovecha bien la caché del procesador y mejora el rendimiento en la práctica.
- Versatilidad: Puede ser adaptado para diferentes tipos de datos y estrategias de selección de pivote según las necesidades específicas.
- Implementación relativamente simple: Aunque requiere cuidado en la implementación de la función de partición, el algoritmo en sí es conceptualmente sencillo.

## Desventajas

- Complejidad en el peor caso: En el peor escenario (cuando el pivote siempre es el elemento más pequeño o más grande), Quick Sort tiene una complejidad de $O(n^2)$, similar a algoritmos menos eficientes como Bubble Sort o Selection Sort.
- Dependencia de la selección del pivote: El rendimiento está directamente relacionado con cómo se selecciona el pivote. Una mala elección puede degradar significativamente el rendimiento.
- No es estable: Quick Sort no mantiene el orden relativo de elementos iguales. Si se requiere estabilidad, algoritmos como Merge Sort son más apropiados.
- Recursión: Aunque generalmente eficiente, el uso de recursión puede causar problemas de desbordamiento de pila en arreglos muy grandes, aunque esto puede mitigarse con implementaciones iterativas.

## Comparación con otros algoritmos clásicos

- Versus Bubble Sort: Quick Sort es significativamente más eficiente. Bubble Sort tiene complejidad $O(n^2)$ en todos los casos, mientras que Quick Sort tiene $O(n \log n)$ en promedio. Bubble Sort es más simple pero prácticamente inutilizable para arreglos grandes.
- Versus Merge Sort: Ambos tienen $O(n \log n)$ en promedio, pero Quick Sort generalmente es más rápido en la práctica debido a mejor localidad de caché. Sin embargo, Merge Sort tiene $O(n \log n)$ garantizado en todos los casos y es estable, mientras que Quick Sort puede degradarse a $O(n^2)$.
- Versus Heap Sort: Heap Sort tiene $O(n \log n)$ garantizado en todos los casos, pero generalmente es más lento que Quick Sort en la práctica debido a peor localidad de referencia. Quick Sort es preferido cuando se busca máximo rendimiento en el caso promedio.
- Versus Insertion Sort: Insertion Sort es $O(n^2)$ en el caso promedio, pero es muy eficiente para arreglos pequeños o casi ordenados. Por esta razón, muchas implementaciones híbridas combinan Quick Sort con Insertion Sort para arreglos pequeños.

# Explicación de la complejidad temporal del algoritmo

La complejidad temporal de Quick Sort varía según el caso:

- CASO PROMEDIO: O(n log n). En el caso promedio, cuando el pivote divide el arreglo en dos partes aproximadamente iguales, la altura del árbol de recursión es log n, y en cada nivel se procesan n elementos. Esto resulta en O(n log n) operaciones.
- CASO MEJOR: O(n log n). Similar al caso promedio, cuando siempre se obtienen particiones balanceadas.
- CASO PEOR: O($n^2$). Ocurre cuando el pivote siempre es el elemento más pequeño o más grande, resultando en particiones desbalanceadas donde un sub-arreglo tiene n-1 elementos y el otro tiene 0. En este caso, la altura del árbol es n, resultando en O($n^2$) operaciones.
- COMPLEJIDAD ESPACIAL: O(log n) en el caso promedio, debido a la pila de recursión. En el peor caso puede ser O(n).

La complejidad promedio de O(n log n) hace que Quick Sort sea uno de los algoritmos más eficientes para ordenamiento general, superando a algoritmos cuadráticos como Bubble Sort, Selection Sort e Insertion Sort, y siendo comparable o superior a otros algoritmos O(n log n) como Merge Sort y Heap Sort en términos de rendimiento práctico.

# Conclusión

Quick Sort es un algoritmo de ordenamiento altamente eficiente que destaca por su excelente rendimiento en el caso promedio. Aunque tiene la desventaja de degradarse a $O(n^2)$ en el peor caso, en la práctica es uno de los algoritmos más rápidos disponibles, especialmente cuando se combina con estrategias inteligentes de selección de pivote. Su implementación in-place y su buena localidad de referencia lo hacen ideal para aplicaciones de software de sistema donde se requiere ordenar grandes volúmenes de datos de manera eficiente.