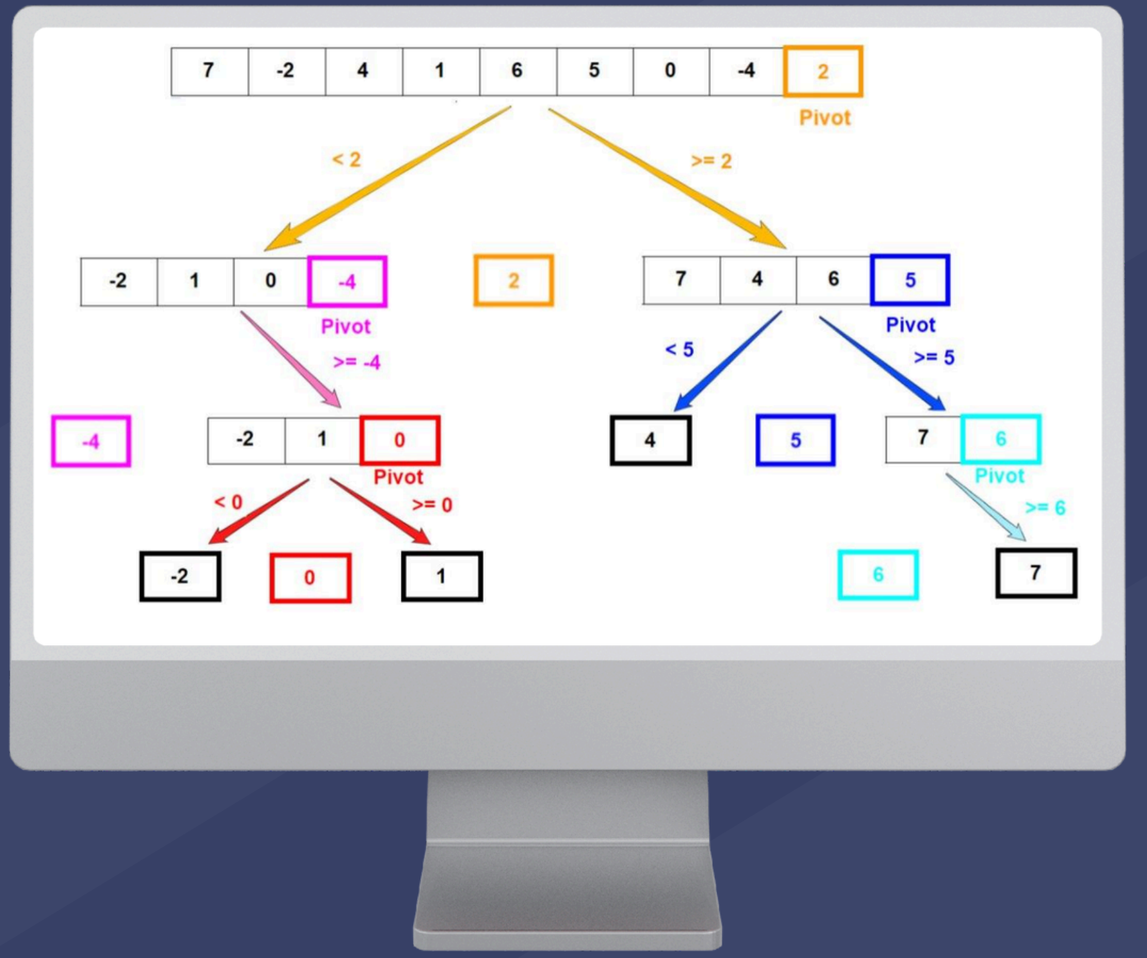
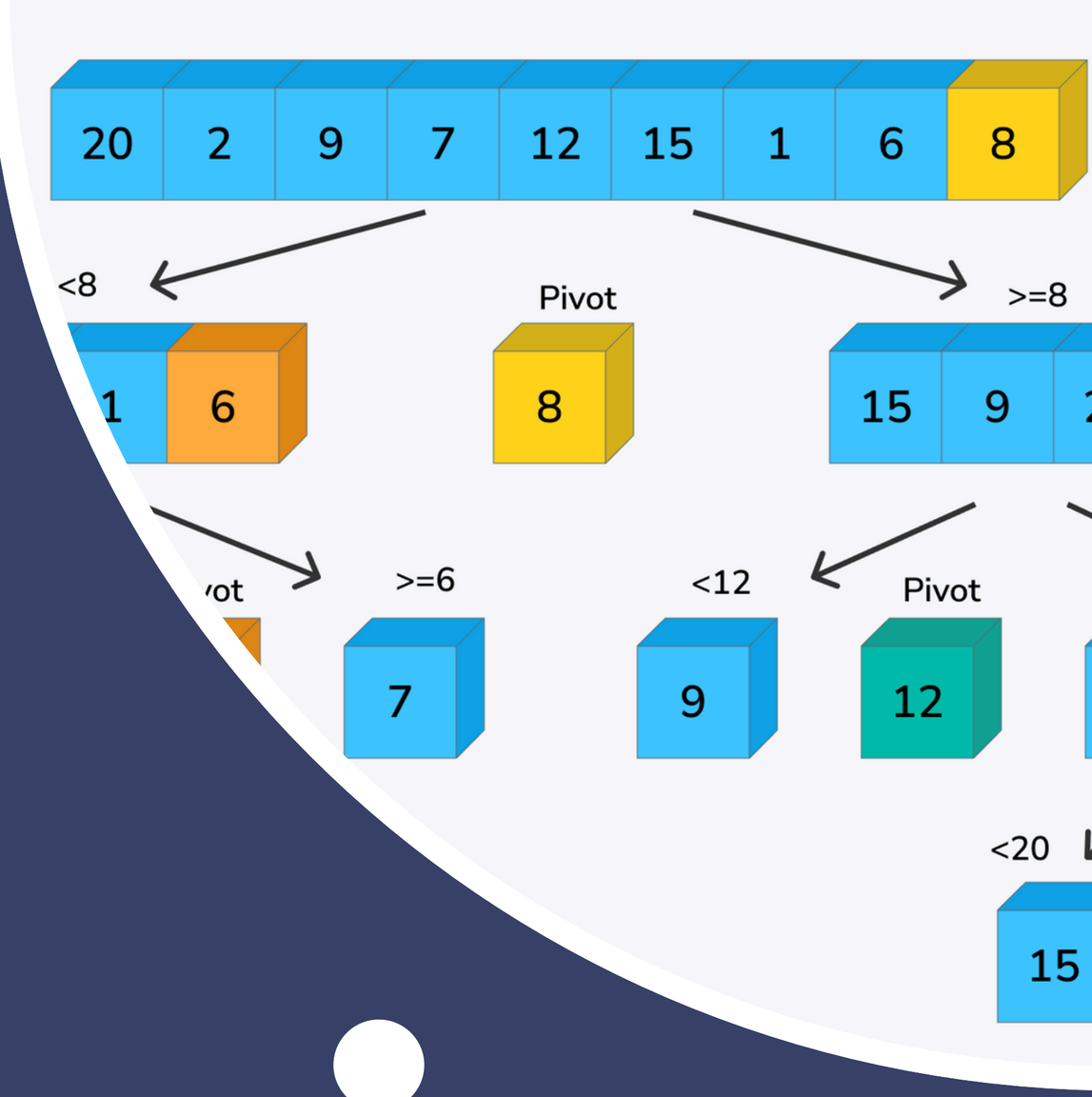
QUICK SORT

ORDENAMIENTO POR INTERCAMBIO



Introducción

- Quick Sort es un algoritmo de ordenamiento que pertenece a la categoría de los algoritmos de “divide y vencerás”
- Fue desarrollado por Tony Hoare en 1959 y se considera uno de los métodos más eficientes para ordenar listas o arreglos grandes.
- La idea principal del algoritmo es dividir la lista en partes más pequeñas, ordenar cada una de ellas de forma recursiva y luego combinarlas para obtener el resultado final.



Funcionamiento del algoritmo

Lista inicial: [5, 2, 8, 1, 14, 17, 4, 3, 19]

Paso 1. Elegir un pivote

Elegimos el último número como pivote: 19.
Comparamos todos los demás números con 19.
Todos son menores, así que 19 se queda al final de la lista.
Estado de la lista:

[5, 2, 8, 1, 14, 17, 4, 3, 19]

Paso 2. Repetir con la parte izquierda del pivote

Sublista izquierda: [5, 2, 8, 1, 14, 17, 4, 3]

- Elegimos 3 como pivote (último número de esta sublista).
- Movemos todos los números menores o iguales a 3 a la izquierda, y los mayores a la derecha.

[2, 1, 3, 5, 14, 17, 4, 8, 19]

*recuerda que el 3 es nuestro nuevo pivote

Paso 3. Repetir con las sublistas

- Sublista izquierda de 3: [2, 1]
  - Pivote = 1 → después de ordenar: [1, 2]
- Sublista derecha de 3: [5, 14, 17, 4, 8]
  - Pivote = 8 → separar menores y mayores: [5, 4, 8, 14, 17]
- Ordenamos [5, 4] → [4, 5]
  - [14, 17] ya está ordenado

Paso 4. Se juntamos todo y tenemos lista ordenada

- [1, 2, 3, 4, 5, 8, 14, 17, 19]

--Ejemplo visual--
https://visualgo.net/en/sorting?create=5%2C2%2C8%2C1%2C14%2C17%2C4%2C3%2C19&mode=Quick&run=true

Pivote: median of medians

array A = [3, -1, 5, 6, 19, 4, 2, 11, 0, 14, 9, 23, -4, -2, 8].

Matriz inicial: A= [3, -1, 5, 6, 19, 4, 2, 11, 0, 14, 9, 23, -4, -2, 8]

Paso 1. Dividir en frupos (usando indices)

Se usan rangos de índices en lugar de nuevas matrices:
Grupo 1: A[0...4] -> [3, -1, 5, 6, 19]
Grupo 2: A[5...9] -> [4, 2, 11, 0, 14]
Grupo 3: A[10...14] -> [9, 23, -4, -2, 8]

Paso 2. Ordenar cada grupo en su lugar

Ordenar los elementos en sus respectivas partes de la matriz original:
[3, -1, 5, 6, 19] se convierte en [-1, 3, 5, 6, 19]
[4, 2, 11, 0, 14] se convierte en [0, 2, 4, 11, 14]
[9, 23, -4, -2, 8] se convierte en [-4, -2, 8, 9, 23]

Matriz tras la ordenación in situ de los fragmentos:
A = [-1, 3, 5, 6, 19, 0, 2, 4, 11, 14, -4, -2, 8, 9, 23]

Paso 3. Extraer las medianas

Crear una nueva matriz, «medians», para almacenar las medianas de cada grupo:

- Mediana del grupo 1 (índice 2): A[2] = 5
- Mediana del grupo 2 (índice 7): A[7] = 4
- Mediana del grupo 3 (índice 12): A[12] = 8

Nueva matriz de medianas: medians = [5, 4, 8]

→ El algoritmo de mediana de medianas encontraría entonces recursivamente la mediana de esta nueva matriz de medianas, que sería 5. ←

Ventajas y Desventajas

Ventajas:

- Alta eficiencia: Más rápidos en la práctica.
- Uso eficiente de memoria: No necesita memoria adicional.
- Buen rendimiento con datos aleatorios: Si los datos no están ordenados de forma especial, suele ser más rápido que otros algoritmos.
- Fácil de implementar con recursión: Su estructura de “divide y vencerás” hace que el código sea compacto y lógico al dividir la lista en partes más pequeñas.

Desventajas:

- Peor caso desfavorable: Si la lista ya está parcialmente ordenada o si se elige un mal pivote, su rendimiento puede caer.
- No es estable: No mantiene el orden relativo de los elementos iguales, lo que puede ser un problema cuando se necesita estabilidad.
- Difícil de predecir en algunos casos: Su tiempo exacto de ejecución depende de cómo se elijan los pivotes y de la distribución de los datos.
- Uso de recursividad: La recursividad puede causar desbordamiento de pila.

Caracteristicas Tecnicas de Quick Sort

Estabilidad

Si hay dos elementos con el mismo valor, su posición relativa puede cambiar después de ordenar.

In - place

Usa solo unas pocas variables temporales para intercambiar los elementos dentro del mismo arreglo, sin necesidad de crear arreglos adicionales .

Adaptabilidad

Su rendimiento no se adapta si los datos ya están casi ordenados; de hecho, si la lista está muy ordenada, puede elegir pivotes poco favorables y empeorar el rendimiento.

Eficiencia del algoritmo

Complejidad Temporal

- Mejor caso: O(nlog(n))
- Caso promedio: O(nlog(n))

Razon: division del arreglo se acerca a la mitad.

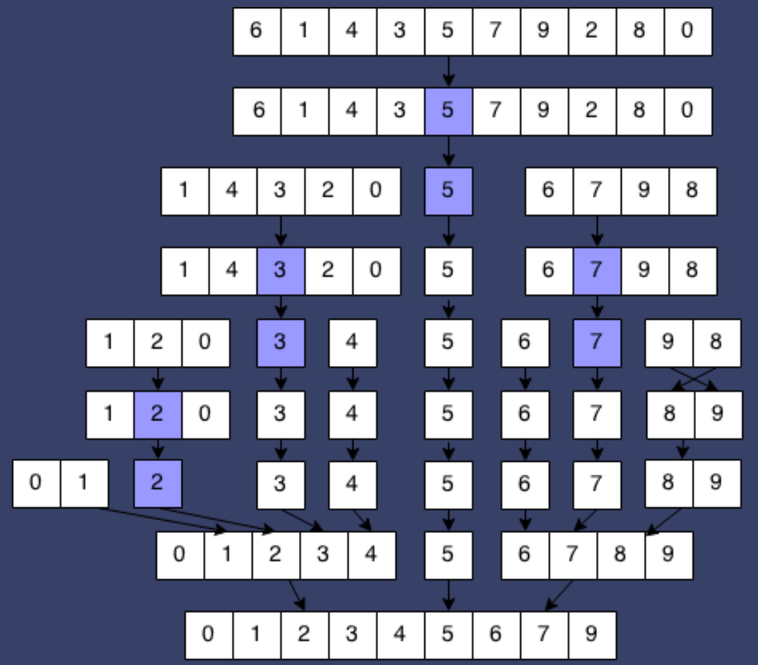
- Peor caso: $O(n^2)$

Razon: sub-arreglos con Gran diferencia de tamaños

Complejidad Espacial

- Regularmente: O(log(n))

Razon: profundidad de pila.



Casos de uso y Aplicaciones practicas

Caso de uso:

Bases de Datos: indexar y organizar datos.

Algoritmos de busqueda: paso intermedio o final.

Graficos y juegos: vértices, puntos o elementos.

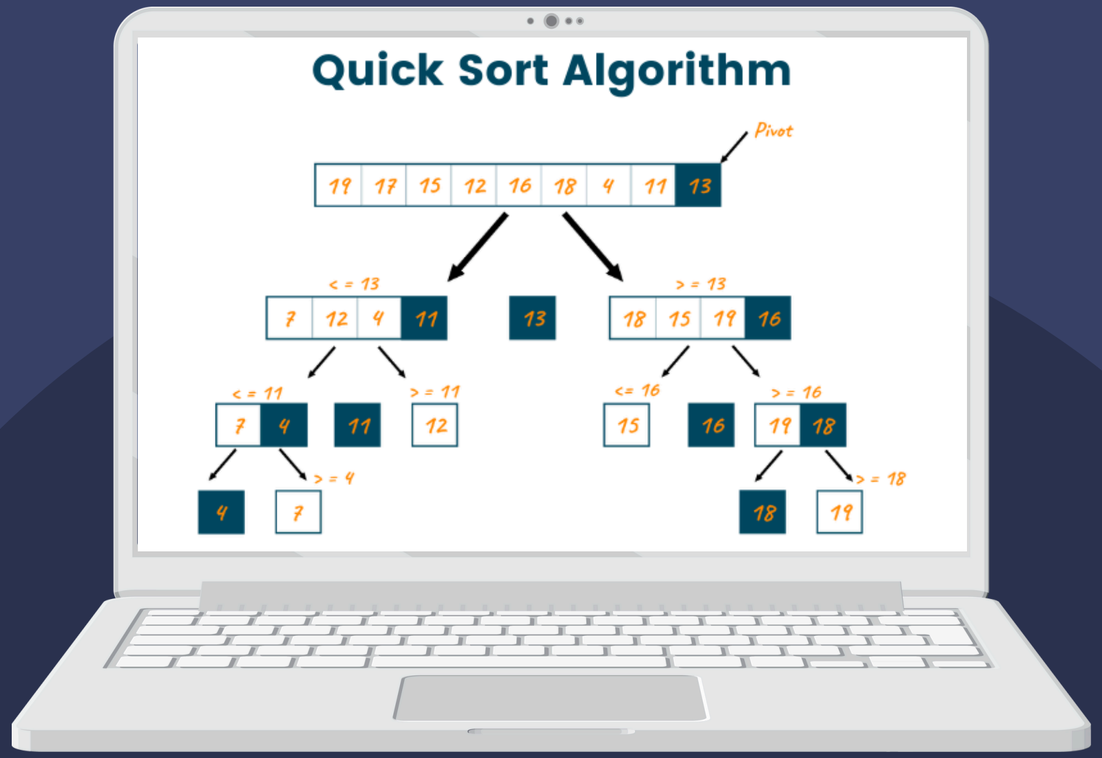
Por que se utiliza:

Eficiencia: rapidez en comparación.

Uso de memoria: algoritmo "in-place".

Velocidad en la practica: pero caso es poco común, caso promedio es eficiente.

Implementacion directa: Comprensión y adaptación.



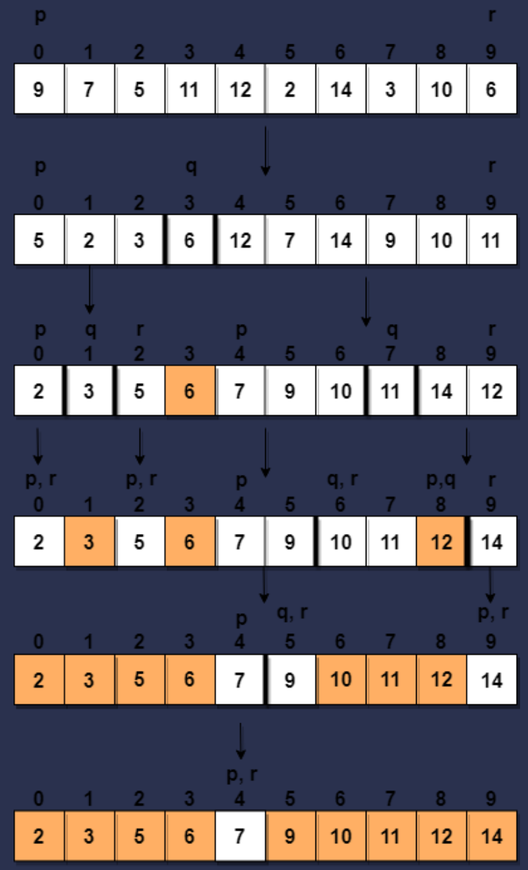
Conclusiones

Conclusión 01

Quick Sort es un algoritmo muy rápido y eficiente que organiza los datos dividiéndolos en partes más pequeñas. Aunque no siempre es estable y su peor caso puede ser lento, en la práctica es uno de los métodos más usados por su rapidez y bajo uso de memoria.

Conclusión 02

Quick sort se siente como el viejo no tan confiable. Buena velocidad, pero caso es raro, y es eficiente y simple de implementar.



MUCHAS GRACIAS