**Taller de Laboratorio: Análisis del Pivote en Quick Sort**

### Objetivo del taller

* Comprender cómo influye la elección del pivote en el rendimiento del algoritmo Quick Sort.
* Analizar comparativamente el comportamiento del algoritmo con distintos tipos de arreglo (ordenado, desordenado, invertido).
* Generar conclusiones sobre qué tipo de elección de pivote es más eficiente según el caso.

### Parte 1: Código base para probar

Utiliza este código como plantilla, y modifica el pivote según se indique.

let pasos = 0;

function quickSort(arr, elegirPivote) {

if (arr.length <= 1) return arr;

const pivotIndex = elegirPivote(arr); // función para elegir el pivote

const pivot = arr[pivotIndex];

const left = [];

const right = [];

for (let i = 0; i < arr.length; i++) {

if (i === pivotIndex) continue;

pasos++;

if (arr[i] < pivot) {

left.push(arr[i]);

} else {

right.push(arr[i]);

}

}

return [...quickSort(left, elegirPivote), pivot, ...quickSort(right, elegirPivote)];

}

**Parte 2: Funciones para probar distintos pivotes**

**function pivoteInicio(arr) {**

**return 0;**

**}**

**function pivoteFinal(arr) {**

**return arr.length - 1;**

**}**

**function pivoteMedio(arr) {**

**return Math.floor(arr.length / 2);**

**}**

**function pivoteAleatorio(arr) {**

**return Math.floor(Math.random() \* arr.length);**

**}**

**Parte 3: Arreglos de prueba**

const ordenado = [1, 2, 3, 4, 5, 6, 7, 8, 9];

const invertido = [9, 8, 7, 6, 5, 4, 3, 2, 1];

const aleatorio = [4, 9, 1, 7, 2, 5, 8, 3, 6];

**Completa la siguiente tabla:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de arreglo** | **Tipo de pivote** | **Resultado ordenado** | **Comparaciones** |
| Ordenado | Inicio |  |  |
| Ordenado | Final |  |  |
| Ordenado | Medio |  |  |
| Ordenado | Aleatorio |  |  |
| Invertido | Inicio |  |  |
| Invertido | Final |  |  |
| Invertido | Medio |  |  |
| Invertido | Aleatorio |  |  |
| Aleatorio | Inicio |  |  |
| Aleatorio | Final |  |  |
| Aleatorio | Medio |  |  |
| Aleatorio | Aleatorio |  |  |

### Preguntas para análisis

1. ¿Qué combinación de tipo de arreglo + pivote resultó más ineficiente?

R/

Tipo de arreglo Tipo de pivote Interacciones

Ordenado Inicio 36

Ordenado Final 36

Invertido Inicio 36

Invertido Final 36

1. ¿Qué combinación fue la más eficiente?

R/:Tipo de arreglo Tipo de pivote Interacciones

Ordenado Medio 16

Invertido Medio 16

Aleatorio Final 16

1. ¿Por qué la elección del pivote influye tanto en el rendimiento del algoritmo?

R/:La elección del pivote influye profundamente en el rendimiento del algoritmo Quicksort, ya que determina cómo se divide (o particiona) el arreglo en cada paso recursivo.

1. ¿Cuál estrategia de elección de pivote recomendarías para arreglos que no conoces previamente? ¿Por qué?

R/:

la estrategia que manejaría seria cumpliendo estas 3 condiciones para reducir tiempo en la ejecución

Tipo de arreglo Tipo de pivote Interacciones

Ordenado Medio 16

Invertido Medio 16

Aleatorio Final 16

1. ¿Te parece que Quick Sort es siempre mejor que otros métodos? ¿En qué casos no lo sería?

R/:

Generalmente más rápido en la práctica que otros métodos como Merge Sort o Heap Sort, debido a su buena localización de caché

Cuando NO es mejor usar Quick Sort:

Cuando los datos ya están (casi) ordenados

En su implementación básica (sin mejoras como el "pivote aleatorio" o "mediana de tres"), Quick Sort puede volverse muy ineficiente, llegando a su peor caso O(n²).

En estos casos, algoritmos como Merge Sort o incluso Insertion Sort pueden ser más eficientes.

Cuando se requiere estabilidad

Quick Sort no es estable, lo que significa que puede cambiar el orden relativo de elementos iguales.

Si necesitas un algoritmo estable (por ejemplo, para ordenar objetos por múltiples criterios), deberías usar Merge Sort o Bubble Sort (aunque este último es lento).

Cuando se necesita garantizar el peor caso

Si es crucial evitar el peor caso de rendimiento (como en sistemas de tiempo real), Merge Sort (O(n log n) siempre) o Heap Sort pueden ser más seguros.

Con listas pequeñas

Para listas muy pequeñas (por ejemplo, menos de 10 elementos), Insertion Sort puede ser más rápido debido a su bajo overhead.