**UNIDAD 1**

**ALGORISMOS 2**

**INTEGRANTES:**

**DANCY DAYANA GOMEZ GALINDEZ**

**SEBASTIAN CASAÑAS SALCEDO**

**ANA SOFIA ANGEL GALAN**

****

**Docente: JORGE GAMARRA**

**UNIVERSIDAD REMINGTON DE CALI**

**Facultad de Ingeniería**

**Ingeniería En Sistema**

**Cali, Agosto 2025**

**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** | **Comparaciones** |
| Ordenado | Inicio | 36 |
| Ordenado | Final | 36 |
| Ordenado | Medio | 16 |
| Ordenado | Aleatorio | 22 |
| Invertido | Inicio | 36 |
| Invertido | Final | 36 |
| Invertido | Medio | 16 |
| Invertido | Aleatorio | 22 |
| Aleatorio | Inicio | 19 |
| Aleatorio | Final | 16 |
| Aleatorio | Medio | 19 |
| Aleatorio | Aleatorio | 18 |

### Preguntas para análisis

1. ¿Qué combinación de tipo de arreglo + pivote resultó más ineficiente?
2. ¿Qué combinación fue la mas eficiente?
3. ¿Por qué la elección del pivote influye tanto en el rendimiento del algoritmo?
4. ¿Cuál estrategia de elección de pivote recomendarías para arreglos que no conoces previamente? ¿Por qué?
5. ¿Te parece que Quick Sort es siempre mejor que otros métodos? ¿En qué casos no lo sería?

**1: RP/:** Teniendo en cuenta las combinaciones de tipo de arreglo + pivote el que resulto más eficiente fue el ordenado y el invertido ya que su resultado fue de 36 cooperaciones,   
Pivote medio en arreglos ordenados/invertidos, y pivote final en un arreglo aleatorio.

**2: RP/:** teniendo en cuenta los resultados la mas eficiente fue la las que generaron solo 16 comparaciones como lo fue el ordenado e invertido.

**Pivote medio = 16**

**Invertido + pivote medio = 16**

**Aleatorio + pivote final = 16**

**3 RP/:** por qué es un elemento del arreglo que usamos para dividir el problema en dos partes. Por qué el pivote divide cerca de la mitad, trabajamos con problemas más pequeños de forma parecida y termina más rápido porque lo reduce de forma más equilibrada.

Si el pivote está cerca de la **mediana** se generan particiones balanceadas y la complejidad promedio es **O(n log n)**

1. **RP/:**QuickSelect: Pivote aleatorio para mantener la complejidad promedio O(n) y evitar peores casos en entradas desconocidas, Si se quiere mayor seguridad, usar mediana de tres de tres o cinco elementos para reducir aún más la probabilidad de desbalance.

Si hay orden parcial; el pivote aleatorio evita fácilmente patrones adversos (inputs diseñados).

**5 RP/:**Quick Sort no siempre es una buena opción, aunque es muy eficiente por su rendimiento promedio y uso en muchos sistemas y bibliotecas.

Sin embargo, no es la mejor opción cuando: Necesitas garantías de rendimiento en el peor caso, requieres estabilidad, Tienes limitaciones de memoria de pila o de memoria total. la implementación estándar de QuickSort no es estable — usa Merge Sort estable.