Quick Sort

Definición

Es un algoritmo de ordenamiento que escoge un pivote como punto de comparación entre los elementos de un arreglo. Los pasos de este algoritmo son:

1. Elegir un pivote
2. Partición de los valores en menores y mayores respecto al pivote
3. Recursividad, para aplicar en cada partición el algoritmo

Partición

Existen tres esquemas de partición:

a. Partición de Naive: El esquema "naive" es el más sencillo pero menos eficiente porque utiliza arreglos auxiliares para dividir los elementos.
b. Partición de Lomuto: Este esquema utiliza un único índice para dividir el arreglo en dos partes.
c. Partición de Hoare: El esquema de Hoare usa dos índices que se mueven desde los extremos hacia el centro.

Método	Ventajas	Desventajas	Uso Ideal
Lomuto	Fácil de implementar; menos memoria	Más intercambios; sensible al pivote	Arreglos pequeños
Hoare	Menos intercambios	Más complejo de implementar	Grandes arreglos desbalanceados
Naive	Simple y fácil de entender	Alto uso de memoria; ineficiente	Entender la lógica de QuickSort

Elección de Pivote

La elección del pivote es un paso crítico en el algoritmo QuickSort, ya que influye directamente en la eficiencia y el equilibrio de las particiones del arreglo. Un pivote bien elegido minimiza la cantidad de pasos necesarios para ordenar, mientras que una mala elección puede provocar divisiones desbalanceadas y un desempeño subóptimo. Las principales estrategias para seleccionarlo son las siguientes:

Método	Costo Computacional	Probabilidad de Desbalance	Complejidad Promedio	Usado en la Práctica
Primer elemento	Bajo	Alta	Media	Común en casos simples
Último elemento	Bajo	Alta	Media	Moderado
Aleatorio	Bajo	Baja	Buena	Común en grandes datos
Elemento Central	Bajo	Media	Media	Moderado
Mediana de Tres	Medio	Muy Baja	Buena	Frecuentemente usado
Mediana Exacta	Alto	Ninguna	Excelente	Raro

Casos Clave

- **Caso Promedio (Θ (n log n)):** Al elegir un pivote equilibrado, cada partición divide el arreglo aproximadamente a la mitad, logrando un rendimiento óptimo.
- **Peor Caso (Θ (n^2)):** Ocurre si el pivote siempre es el mayor o menor elemento, lo que da lugar a particiones desbalanceadas. Ejemplo: una lista ya ordenada.
- **Mejor Caso (Θ (n log n)):** El pivote siempre divide el arreglo en dos mitades iguales.

Ejemplo:

```php
<?php
    function partition(&$arr, $low, $high) {
        $pivot = $arr[$high];
        $i = $low - 1;

        for ($j = $low; $j < $high; $j++) {
            if ($arr[$j] < $pivot) {
                $i++;
                [$arr[$i], $arr[$j]] = [$arr[$j], $arr[$i]];
            }
        }

        [$arr[$i + 1], $arr[$high]] = [$arr[$high], $arr[$i + 1]];
        return $i + 1;
    }

    function quickSort(&$arr, $low, $high) {
        if ($low < $high) {
            $pi = partition($arr, $low, $high);

            quickSort($arr, $low, $pi - 1);
            quickSort($arr, $pi + 1, $high);
        }
    }
```

Binary Search

Es un algoritmo de búsqueda en arreglos previamente ordenados, consiste en dividir repetitivamente a la mitad los intervalos basándose en el "mid" como inicio del rastreo:

1. Encontrar el index del mid
2. Dividir en dos mitades respecto al mid
3. Elegir una mitad, si es menor la izquierda, si es mayor la derecha
4. Repetir el proceso hasta encontrar el valor

Estrategias de implementación

a. Iteratividad: Se utiliza un bucle para ajustar los límites de búsqueda hasta encontrar el elemento o agotar las posibilidades
b. Recursividad: Llama múltilples veces a la misma función con diferentes límites de búsqueda, hasta que una encuentre el valor o acabe la recursión.

Característica	Iterativa	Recursiva
Eficiencia en memoria	Más eficiente	Menos eficiente (usa pila)
Código más directo	A veces más largo con bucles	Más compacto con recursión
Riesgo de desbordamiento de pila	No	Sí (en arrays muy grandes)
Legibilidad	Más familiar para principiantes	Elegante, pero requiere experiencia

Casos Claves:

- **Mejor caso**: $\Theta(1)$ cuando el elemento se encuentra en el primer intento
- **Peor caso**: $\Theta(\log_2(n))$, debido a la reducción exponencial del espacio de búsqueda en cada iteración

Limitaciones:

- Requiere datos ordenados, lo cual puede añadir un costo previo de $\Theta(n \log_n)$ si es necesario ordenarlos.
- No es adecuada para estructuras no contiguas (como listas enlazadas).

Ejemplo:

```php
<?php
function binarySearch($arr, $x){
    $low =0;
    $high = count($arr) - 1;
    $mid = 0;

    while($low <= $high){
        $mid = floor(($low + $high) / 2);

        if($arr[$mid] == $x){
            return $mid;
        }

        if($arr[$mid] < $x){
            $low = $mid + 1;
        }else{
            $high = $mid - 1;
        }
    }

    return -1;
}
```