Resumen: Ordenación Rápida (Quicksort)

La ordenación rápida, conocida como *Quicksort*, es un algoritmo eficiente de ordenación basado en la técnica de *divide y vencerás*, descrito en la sección 7.4.2 del libro *Fundamentos de Algoritmia* de G. Brassard y P. Bratley (página 265). Desarrollado por C.A.R. Hoare, es ampliamente utilizado por su simplicidad y alto rendimiento en casos promedio, siendo una herramienta clave en informática.

Quicksort selecciona un elemento del arreglo, el *pivote*, y reorganiza los elementos para que los menores o iguales queden a su izquierda y los mayores a su derecha mediante un proceso llamado *partición*. Esto divide el arreglo en dos subarreglos que se ordenan recursivamente. La elección del pivote (primer elemento, último, central o aleatorio) afecta el rendimiento. La partición usa dos índices que recorren el arreglo desde los extremos, intercambiando elementos hasta posicionar el pivote en su lugar final.

La complejidad promedio de *Quicksort* es $O(n \log n)$, ideal para arreglos grandes, pero en el peor caso (por ejemplo, arreglo ordenado con pivote extremo), puede alcanzar $O(n^2)$. Estrategias como elegir el pivote aleatoriamente o usar la mediana de un subconjunto mitigan este problema. *Quicksort* es un algoritmo *in-place*, requiriendo poca memoria adicional, pero no es estable, pudiendo alterar el orden relativo de elementos iguales.

En la práctica, su versatilidad y eficiencia lo hacen ideal para aplicaciones como bases de datos. Optimizar la elección del pivote o combinarlo con algoritmos como la ordenación por inserción para subarreglos pequeños mejora su rendimiento.

Ejercicios Resueltos

1. **Simulación Manual**: Dado el arreglo $[8, 3, 5, 1, 9, 2, 7, 4, 6]$, realiza *Quicksort* con el último elemento como pivote.
   Solución: Tomamos el pivote 6. Partición: comparamos desde los extremos. Intercambiamos $8 > 6$ con $4 < 6$: $[4, 3, 5, 1, 2, 6, 7, 8, 9]$. El pivote queda en la posición 5. Subarreglos: $[4, 3, 5, 1, 2]$ y $[7, 8, 9]$. Para $[4, 3, 5, 1, 2]$, pivote 2: $[1, 2, 5, 3, 4]$. Subarreglos: $[1]$ (ordenado) y $[5, 3, 4]$. Pivote 4: $[3, 4, 5]$. Continúa hasta obtener $[1, 2, 3, 4, 5, 6, 7, 8, 9]$. Para $[7, 8, 9]$, pivote 9, ya ordenado. Resultado final: $[1, 2, 3, 4, 5, 6, 7, 8, 9]$.

2. **Análisis de Pivotes**: Para el arreglo $[1, 2, 3, 4, 5]$, explica el efecto de elegir el primer elemento como pivote.
   Solución: Si elegimos 1 como pivote, todos los elementos son mayores, resultando en subarreglos $[]$ y $[2, 3, 4, 5]$. En la siguiente iteración, pivote 2, subarreglos $[]$ y $[3, 4, 5]$, y así sucesivamente. Esto genera $n - 1$ particiones, cada una reduciendo el tamaño en 1, dando una complejidad de $O(n^2)$. Para evitarlo, usar un pivote aleatorio o la mediana de tres elementos distribuye mejor los subarreglos, acercándose a $O(n \log n)$.

3. **Implementación**: Escribe *Quicksort* en pseudocódigo con pivote como la mediana de tres.

Solución:

```
Quicksort(A, inicio, fin)
    si inicio < fin entonces
        pivote = MedianaDeTres(A, inicio, fin)
        indicePivote = Particion(A, inicio, fin, pivote)
        Quicksort(A, inicio, indicePivote - 1)
        Quicksort(A, indicePivote + 1, fin)

MedianaDeTres(A, inicio, fin)
    medio = (inicio + fin) / 2
    a = A[inicio], b = A[medio], c = A[fin]
    devolver mediana de (a, b, c)

Particion(A, inicio, fin, pivote)
    intercambiar A[fin] con pivote
    i = inicio - 1
    para j = inicio hasta fin - 1
        si A[j] <= pivote entonces
            i = i + 1
            intercambiar A[i] con A[j]
    intercambiar A[i + 1] con A[fin]
    devolver i + 1
```

La mediana de tres reduce la probabilidad de elegir un pivote extremo, mejorando el rendimiento en arreglos parcialmente ordenados.

4. **Comparación de Casos**: Compara *Quicksort* en un arreglo aleatorio $[7, 2, 9, 4, 1, 8, 3, 6, 5, 10$ y uno casi ordenado $[1, 2, 3, 4, 5, 6, 7, 8, 10, 9]$ con pivote como primer elemento.
   Solución: Para el arreglo aleatorio, pivote 7: partición da $[2, 4, 1, 3, 5, 6, 7, 8, 9, 10]$, con 6 comparaciones. Subarreglos $[2, 4, 1, 3, 5, 6]$ y $[8, 9, 10]$ requieren menos particiones debido a la distribución equilibrada, aproximándose a $O(n \log n)$. Para el arreglo casi ordenado, pivote 1: subarreglos $[]$ y $[2, 3, 4, 5, 6, 7, 8, 10, 9]$, con 9 comparaciones. Esto se repite, generando $O(n^2)$ por particiones desbalanceadas. El arreglo aleatorio requiere menos comparaciones debido a particiones más equilibradas.

5. **Optimización**: Explica la combinación de *Quicksort* con ordenación por inserción para subarreglos pequeños.
   Solución: Para subarreglos de menos de 10 elementos, *Quicksort* tiene sobrecarga recursiva alta. La ordenación por inserción, con complejidad $O(n^2)$, es más eficiente para arreglos pequeños debido a su simplicidad y bajo costo constante. Implementar *Quicksort* hasta que el subarreglo tenga, por ejemplo, 10 elementos, y luego usar inserción, reduce el número de llamadas recursivas. Esto mejora el rendimiento en la práctica, especialmente en arreglos con muchos subarreglos pequeños, combinando la eficiencia de *Quicksort* para particiones grandes con la simplicidad de inserción para las pequeñas.

Conclusión

Quicksort es un algoritmo versátil y eficiente, ideal para aplicaciones prácticas, aunque su rendimiento depende de la elección del pivote. Los ejercicios resueltos ilustran su funcionamiento, desde simulaciones prácticas hasta optimizaciones, destacando su fortaleza en casos promedio y la necesidad de estrategias para evitar casos patológicos.