

# Algoritmo de ordenamiento rápido (Quicksort)

Es un algoritmo de ordenación creado por el científico británico en computación Charles Antony Richard Hoare

El algoritmo trabaja de la siguiente forma:

- Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

- En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es  $\Omega(n \cdot \log n)$ .
- En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de  $O(n^2)$ . El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente.
- En el caso promedio, el orden es  $O(n \cdot \log n)$ .

## Pseudocódigo

QUICKSORT( $A, p, r$ )

```
1  if  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      QUICKSORT( $A, p, q - 1$ )
4      QUICKSORT( $A, q + 1, r$ )
```

### Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray  $A[p..r]$  in place.

PARTITION( $A, p, r$ )

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
```