Ultra-Quicksort
Johanna Beltran y Diego Triviño
2012

Índice

1. Introducción

'**Ultra-Quicksort**' es un problema de programación el cual encontramos en el juez virtual UVA con el número **10810**.
Este documento busca mostrar una de las tantas soluciones desde el enfoque matemático teniendo en cuenta que el objetivo es realizar la implementación de la solución del problema en cualquier lenguaje de programación con la ayuda de este documento.
Este problema puede ser resuelto utilizando la estrategia de 'divide y vencerás'.
Esta estrategia es una técnica de diseño de algoritmos la cual consiste en dividir de forma recurrente un problema en subproblemas más sencillos hasta que se encuentre un caso base.

Esta técnica consta fundamentalmente de los siguientes pasos:

1. Descomponer el problema a resolver en un cierto número de subproblemas más pequeños.
2. Resolver independientemente cada subproblema.
3. Combinar los resultados obtenidos para construir la solución del problema original.

2. Definición del problema

Este problema consiste en determinar el número de operaciones que Ultra-QuickSort necesita llevar a cabo para ordenar una secuencia de entrada.

Se deben tener en cuenta las siguientes restricciones para la solución del problema:

1. Para cada caso de prueba se debe ingresar un número n y una secuencia de n números enteros.
2. Para n se debe tener en cuenta que $1 \leq n < 500000$ y que cada número que conforma la secuencia dada debe ser $0 \leq n_i \leq 999999999$.

2.1. Entrada

La entrada contiene varios casos de prueba. Cada caso de prueba comienza con una línea que contiene un único entero $n < 500{,}000$. Cada una de las siguientes n líneas contiene un único número entero $0 \leq a_i \leq 999{,}999{,}999$, el $i - ésimo$ elemento de entrada de secuencia. La entrada es terminada por una secuencia de longitud $n = 0$. Esta secuencia no debe ser procesada.

EJEMPLO

5

9

1

0

5

4

3

1

2

3

0

2.2. Salida

Para cada secuencia de entrada, el programa imprime una sola línea que contiene un número entero *op* , el número mínimo de operaciones de intercambio necesarias para reordenar la secuencia de entrada dada.

EJEMPLO ANTERIOR

6

0

3. Modelamiento matemático

Dado un número entero positivo n, se recibe una secuencia de n números enteros:

$$E_1,\ E_2,\ E_3,\ ...,\ E_n$$

4. Planteamiento de la solución

Para este problema utilizaremos el algoritmo de ordenamiento **Quicksort** que permite, en promedio, ordenar n elementos en el menor tiempo posible; el algoritmo trabaja de la siguiente forma:

[1]

1. Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.

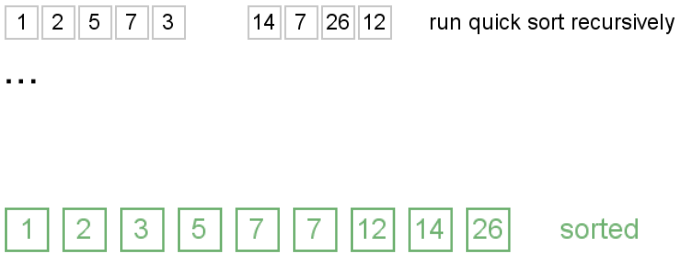
[2]

2. 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.
[3]

[1] http://www.algolist.net/Algorithms/Sorting/Quicksort
[2] http://www.algolist.net/Algorithms/Sorting/Quicksort
[3] http://www.algolist.net/Algorithms/Sorting/Quicksort

3. 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.

4. 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.



[4]

5. 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 $O\left(n\ \log\ n\right)$.

6. En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $O\left(n^2\right)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas.

Para este problema debemos contar el número de veces que el algoritmo tiene que implementar el metodo de ordenamiento solo si es necesario, es decir, si la lista ingresada ya esta ordenada no debe ingresar al metodo de ordenamiento.

[4] http://www.algolist.net/Algorithms/Sorting/Quicksort

5. Conclusiones

1. Este algoritmo de ordenación es un ejemplo claro de que el método divide y vencerás es efectivo cuando tienes cantidades grandes de datos por trabajar y necesitas ahorrar tiempo y recursos.

2. En el método de ordenación quicksort es mejor tener el pivote en el medio del vector puesto que es mas fácil de implementar y su código es mas sencillo.