Práctica 2

Objetivo

Aplicar el enfoque “Divide y Vencerás” a diferentes tipos de problemas para obtener soluciones recursivas a los mismos.

Divide y vencerás

Los algoritmos de Divide y Vencerás no abordan la resolución de la instancia del problema original directamente. En su lugar, dividen el problema en instancias de menor tamaño, las resuelven recursivamente aplicando el mismo enfoque Divide y Vencerás, y combinan sus soluciones para obtener la solución a la instancia original. De forma general, el coste en tiempo de cómputo de un algoritmo Divide y Vencerás tiene la siguiente forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

donde a es el número de veces que hacemos llamadas recursivas, b es el factor por el que se divide el tamaño de la entrada, y la expresión $f(n)$ representa el coste de trocear los datos de entrada y combinar las soluciones de las instancias más pequeñas del problema.

La efectividad de estos algoritmos se basa en encontrar alguna propiedad del problema a resolver que nos permita dividir el problema en instancias más pequeñas. Por ejemplo, en la búsqueda binaria utilizamos el hecho de que los datos están ordenados.

Por otro lado, a la hora de aplicar el enfoque Divide y Vencerás, encontrar la forma de dividir el problema de entrada y de combinar los resultados de los subproblemas no es fácil. Sin embargo, **en muchas ocasiones no hacen falta ideas felices sino algo de experiencia**, que nos permita intentar abordar el problema utilizando estrategias similares a las empleadas en otros problemas que ya hemos estudiado o trabajado previamente.

Ejercicio 1. Completa el método `int buscarKesimo(int [] v, int k)`, de la clase `KesimoElemento`. Dicho método recibe un array `v` de números enteros, posiblemente desordenados, y encuentra el elemento que ocuparía la *k-ésima* posición del array (empezamos contando por 0) si estuviera ordenado. Se permite modificar la lista de entrada si se considera necesario, pero la complejidad del método ha de ser $\theta(n)$ en el mejor caso.

Pista: Se aconseja diseñar un algoritmo que utilice el método `partir` del algoritmo `Quicksort`.

Ejercicio 2. Completa el método `int numeroInversiones(int [] v)` de la clase `ProblemaInversiones`, el cual recibe un array `v` de números enteros y cuenta el número de posiciones invertidas que haya. Decimos que 2 posiciones del array i y j (con $i < j$) están invertidas, si $v[i] > v[j]$, es decir, si los elementos de las posiciones i y j no están relativamente ordenados de manera creciente entre sí. Se permite modificar el array si se considera necesario, pero la complejidad del método ha de ser $\theta(nlogn)$.

Pista: Se aconseja utilizar la misma estrategia que `MergeSort`, pero cambiando algo del método `mezclar()` para que se adapte a la nueva tarea requerida.

Ejercicio 3. Dado un array ordenado `v` de números enteros consecutivos, sabemos que todos los elementos son únicos excepto uno, que está repetido un número indeterminado de veces. Por ejemplo, v = {2,3,4,4,4,5,6}.

Queremos encontrar dicho elemento. Completa el método `int encuentraElem(int [] v)` de la clase `ArraysConRepeticiones`, para que en tiempo $\theta(\log n)$ encuentre y devuelva dicho elemento repetido.

Pista: Se aconseja utilizar una estrategia similar a la utilizada por la búsqueda binaria para localizar el elemento.