**Análisis y Diseño de Algoritmos**
Divide y Vencerás
(2º de Grado Ing. Inf., Ing. Sw., Ing. Comp.)
E.T.S.I. INFORMÁTICA

# Relación de Problemas de Divide y Vencerás

1. Dado un vector ordenado $V$ de $n$ enteros distintos, escribir un algoritmo que en tiempo $O(logn)$ encuentre un número $i$ tal que $1 \leq i \leq n$ y $V[i] = i$, siempre que exista.

2. Dados dos vectores de $n$ enteros cada uno y ordenados de forma creciente, escribir un algoritmo para hallar la mediana del vector formado por los $2n$ enteros, cuya complejidad sea $logn$.

3. Dado un vector $V$ de $n$ elementos (no necesariamente ordenables), se dice que un elemento $x$ es mayoritario en $V$ cuando el número de veces que $x$ aparece en $V$ es mayor que $n/2$. Escribir un algoritmo que en tiempo $O(nlogn)$ decida si un vector tiene un elemento mayoritario y lo devuelva si lo tiene.

4. Dado un vector $V$ de $n$ enteros. Escribir un algoritmo que en tiempo $O(nlogn)$ encuentre el subvector (formado por elementos consecutivos) cuya suma sea máxima.

5. Dado un vector $V$ de enteros todos distintos y un número entero $S$. Diseñar un algoritmo que en tiempo $O(nlogn)$ determine si existen o no dos elementos de $V$ tales que su suma sea $S$.

6. Sea un array $A[]$ que contiene valores numéricos naturales todos distintos. Decimos que dos posiciones $i, j$ (con $i < j$) forman una inversión si se cumple que $A[i] > A[j]$; es decir, si los elementos en las posiciones $i$ y $j$ no están relativamente ordenados de manera creciente entre sí. Deseamos obtener una función `int numInversiones (int[] A)` que devuelva el número total de inversiones del array. Por ejemplo, si $A = \{2, 4, 1, 3, 5\}$, entonces $numInversiones(A) = 3$ (correspondientes a las posiciones (1,3), (2,3) y (2,4)).

   Se pide:

   *a*) Diseñar un algoritmo para resolver el problema mediante un enfoque de fuerza bruta, e indicar su complejidad en términos del número de comparaciones entre elementos realizadas.

   *b*) Diseñar un algoritmo alternativo siguiendo un enfoque de divide y vencerás, de manera que sea más eficiente que el enfoque de fuerza bruta (Indicación: Si el alumno lo ve necesario para la resolución del problema, se podrán mezclar los elementos).

   *c*) Plantear y resolver mediante el Teorema Maestro una recurrencia para el coste computacional del algoritmo de divide y vencerás.

7. Sea $A$ un array unimodal consistente en una secuencia estrictamente creciente de números enteros seguida por una secuencia estrictamente decreciente. Se pretende construir una función `int pico (int[] A)` que, dado un array unimodal $A$, devuelva su pico, o sea, el valor del array a partir del cual la secuencia pasa de ser creciente a decreciente. (Nota: se asumirá que las secuencias creciente y decreciente no son vacías y que el array que se recibe es efectivamente unimodal y no únicamente una secuencia de números creciente o decreciente). Por ejemplo, si $A = \{1, 5, 7, 9, 6\}$, entonces $pico(A) = 9$. Se pide:

*a*) Diseñar un algoritmo para resolver el problema mediante un enfoque de fuerza bruta, e indicar su complejidad en términos del número de operaciones elementales ejecutadas (se puede simplificar realizando el cálculo con respecto al número de comparaciones entre elementos realizadas).

*b*) Diseñar un algoritmo alternativo siguiendo un enfoque de divide y vencerás, de manera que sea más eficiente que el enfoque de fuerza bruta.

*c*) Plantear y resolver (quizás mediante el Teorema Maestro) una recurrencia para el coste computacional del algoritmo de divide y vencerás.

8. Diseña un algoritmo Divide y Vencerás, de complejidad $O(nlogn)$ que, dado un array de $n$ compontes enteras (*int a[0..n-1]*), encuentre el subvector de $a$ creciente más largo (o uno de ellos, si hay más de uno). Un subvector creciente de $a$ es un subvector *a[i..j]* con $i \leq j$ que satisface que para todo $i \leq k_1, k_2 \leq j$, si $k_1 \leq k_2$ entonces $a[k_1] \leq a[k_2]$.

## Problemas complementarios

1. Confeccione una función basada en el enfoque de Divide y Vencerás que tome como parámetros dos números naturales $a$ y $n$ y devuelva $a^n$. Calcule su complejidad. ¿Es mejor que la del algoritmo de fuerza bruta? Si no lo es, ¿puede modificarse la función para que lo sea?

2. Dado un vector $V$ de $n$ elementos y un número natural $k$ diseñar un algoritmo que transponga los $k$ primeros elementos de $V$ con los elementos de las $n$–$k$ últimas posiciones, sin hacer uso de un vector auxiliar. Por ejemplo, si $V = \{a, b, c, d, e, f, g, h, i, j\}$ y $k = 3$, entonces el resultado deseado debe ser $V = \{d, e, f, g, h, i, j, a, b, c\}$.

3. Diseñar un algoritmo de búsqueda "ternaria", que primero compare con el elemento en posición $n/3$ de la lista, si éste es menor que el elemento $x$ a buscar entonces compara con el elemento en posición $2n/3$, y si no coincide con $x$ busca recursivamente en una sublista de tamaño $1/3$ de la original. Comparar este algoritmo con el de búsqueda binaria. Definir una expresión recurrente para el número de comparaciones de elementos que se realizan en el peor caso, y encuentre el orden de crecimiento mediante el Teorema Maestro. A continuación, resuelva de manera exacta la anterior recurrencia

4. En una habitación oscura se tienen dos cajones en uno de los cuales hay $n$ tornillos de varios tamaños y en el otro las correspondientes $n$ tuercas. Es necesario emparejar cada tornillo con su tuerca correspondiente pero no es posible comparar tornillos con tornillos ni tuercas con tuercas, la única comparación posible es la de tuercas con tornillos para decidir si es demasiado grande, demasiado pequeña o se ajusta al tornillo. Escribir un algoritmo que en tiempo $O(nlogn)$ empareje los tornillos con las tuercas.

5. Diseñar un algoritmo de búsqueda binaria que, en vez de dividir la lista de elementos en dos mitades del mismo tamaño, la divida en dos partes de tamaños $1/3$ y $2/3$. Comparar este algoritmo con el original.

6. Sea $A$ una matriz $2 \times 2$:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

Se desea calcular $A^n = \overbrace{A \times A \times \cdots A}^{n}$.

*a*) Diseñar un enfoque basado en fuerza bruta. ¿Cuál es su complejidad?

*b*) Construir un algoritmo Divide&Vencerás mejor que el algoritmo de fuerza bruta previo.

*c*) Expresar la complejidad del algoritmo D&V del apartado anterior mediante una ecuación de recurrencia, e indicar su orden de complejidad utilizando el Teorema Maestro.