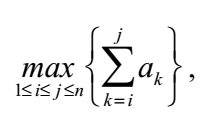
**MEDIANA DE DOS VECTORES**Sean X e Y dos vectores de tamaño n, ordenados de forma no decreciente.  
Necesitamos implementar un algoritmo para calcular la mediana de los 2n  
elementos que contienen X e Y. Recordemos que la mediana de un vector de k  
elementos es aquel elemento que ocupa la posición (k+1)÷2 una vez el vector está  
ordenado de forma creciente. Dicho de otra forma, la mediada es aquel elemento  
que, una vez ordenado el vector, deja la mitad de los elementos a cada uno de sus  
lados. Como en nuestro caso k = 2n (y por tanto par) buscamos el elemento en  
posición n de la unión ordenada de X e Y.  
Solución :  
Para resolver el problema utilizaremos una idea basada en el método de búsqueda  
binaria. Comenzaremos estudiando el caso base, que ocurre cuando tenemos dos  
vectores de un elemento cada uno (n = 1). En este caso la mediana será el mínimo  
de ambos números, pues obedeciendo a la definición sería el elemento que ocupa la  
primera posición (n = 1) si ordenásemos ambos vectores.  
Respecto al caso general, existe una forma de dividir el problema en  
subproblemas más pequeños. Sea Z el vector resultante de mezclar ordenadamente  
los vectores X e Y, y sea mZ la mediana de Z. Apoyándonos en el hecho de que X e  
Y se encuentran ordenados, es fácil calcular sus medianas (son los elementos que  
ocupan las posiciones centrales de ambos vectores), y que llamaremos mX y mY.

Ahora bien, si mX = mY entonces la mediana mZ va a coincidir también con ellas,  
pues al mezclar ambos vectores las medianas se situarán en el centro del vector.  
Si tenemos que mX < mY, podemos afirmar que la mediana mZ va a ser mayor  
que mX pero menor que mY, y por tanto mZ va a encontrarse en algún lugar de la  
segunda mitad del vector X o en algún lugar de la primera mitad de Y (por estar  
ambos vectores ordenados).  
Análogamente, si mX > mY la mediana mZ va a ser mayor que mY pero menor  
que mX, y por tanto mZ va a encontrarse en algún lugar de la primera mitad del  
vector X o en algún lugar de la segunda mitad de Y (por estar ambos vectores  
ordenados). Esta idea nos lleva a la siguiente versión de nuestro algoritmo.

**EL ELEMENTO MAYORITARIO**Sea a[1..n] un vector de enteros. Un elemento x se denomina elemento mayoritario  
de a si x aparece en el vector más de n/2 veces, es decir, Card{i | a[i]=x} > n/2.  
Necesitamos implementar un algoritmo capaz de decidir si un vector dado contiene  
un elemento mayoritario (no puede haber más de uno) y calcular su tiempo de  
ejecución. Sin suponer que el vector se encuentra ordenado, una forma de plantear la  
solución a este problema aparece indicada en [WEI95]. La idea consiste en  
encontrar primero un posible candidato, es decir, el único elemento que podría ser  
mayoritario, y luego comprobar si realmente lo es. De no serlo, el vector no  
admitiría elemento mayoritario, como le ocurre al vector [1,1,2,2,3,3,3,3,4].  
Para encontrar el candidato, suponiendo que el número de elementos del vector  
es par, vamos a ir comparándolos por pares (a[1] con a[2], a[3] con a[4], etc.). Si  
para algún k = 1,3,5,7,... se tiene que a[k] = a[k+1] entonces copiaremos a[k] en un  
segundo vector auxiliar b. Una vez recorrido todo el vector a, buscaremos  
recursivamente un candidato para el vector b, y así sucesivamente. Este método  
obecede a la técnica de Divide y Vencerás pues en cada paso el número de  
elementos se reduce a menos de la mitad.  
Vamos a considerar tres cuestiones para implementar un algoritmo basado en  
esta idea: (i) su caso base, (ii) lo que ocurre cuando el número de elementos es  
impar, y (iii) cómo eliminar el uso recursivo del vector auxiliar b para no aumentar  
excesivamente la complejidad espacial del método.  
i) En primer lugar, el caso base de la recursión ocurre cuando disponemos de un  
vector con uno o dos elementos. En este caso existe elemento mayoritario si los  
elementos del vector son iguales.  
ii) Si el número de elementos n del vector es impar y mayor que 2, aplicaremos la  
idea anterior para el subvector compuesto por sus primeros n–1 elementos.  
Como resultado puede que obtengamos que dicho subvector contiene un  
candidato a elemento mayoritario, con lo cual éste lo será también para el vector  
completo. Pero si la búsqueda de candidato para el subvector de n–1 elementos  
no encuentra ninguno, escogeremos como candidato el n-ésimo elemento.  
iii)Respecto a cómo eliminar el vector auxiliar b, podemos pensar en utilizar el  
propio vector a para ir almacenando los elementos que vayan quedando tras  
cada una de las pasadas.

**EL TORNEO DE TENIS**Necesitamos organizar un torneo de tenis con n jugadores en donde cada jugador  
ha de jugar exactamente una vez contra cada uno de sus posibles n–1 competidores,  
y además ha de jugar un partido cada día, teniendo a lo sumo un día de descanso en  
todo el torneo.

Dado cualquier n>1, implementaremos un algoritmo para construir un cuadrante  
de partidas del torneo que permita terminarlo en n–1 días si n es par, o en n días  
si n es impar.  
En el primer caso suponemos que n es una potencia de 2. El caso más simple se  
produce cuando sólo tenemos dos jugadores, cuya solución es fácil pues basta  
enfrentar uno contra el otro.  
Si n>2, aplicaremos la técnica de Divide y Vencerás para construir la tabla  
pedida suponiendo que tenemos calculada ya una solución para la mitad de los  
jugadores, esto es, que tenemos relleno el cuadrante superior izquierdo de la tabla.  
En este caso los otros tres cuadrantes no son difíciles de rellenar, como puede  
observarse en la siguiente figura, y en donde se han tenido en cuenta la siguientes  
consideraciones para su construcción:  
1. El cuadrante inferior izquierdo debe enfrentar a los jugadores de número  
superior entre ellos, por lo que se obtiene sumando n/2 a los valores del  
cuadrante superior izquierdo.  
2. El cuadrante superior derecho enfrenta a los jugadores con menores y mayores  
números, y se puede obtener enfrentando a los jugadores numerados 1 a n/2  
contra (n/2)+1 a n respectivamente en el día n/2, y después rotando los valores  
(n/2)+1 a n cada día.  
3. Análogamente, el cuadrante inferior derecho enfrenta a los jugadores de mayor  
número contra los de menor número, y se puede obtener enfrentando a los  
jugadores (n/2)+1 a n contra 1 a n/2 respectivamente en el día n/2, y después  
rotando los valores 1 a n cada día, pero en sentido contrario a como lo hemos  
hecho para el cuadrante superior derecho.

**LA SUBSECUENCIA DE SUMA MÁXIMA**Dados n enteros cualesquiera a1,a2,...,an, necesitamos encontrar el valor de la  
expresión: 

que calcula el máximo de las sumas parciales de elementos consecutivos. Como  
ejemplo, dados los 6 enteros (–2,11,–4,13,–5,–2) la solución al problema es 20  
(suma de a2 hasta a4).

Solución:

Utilizaremos ahora la técnica de Divide y Vencerás para intentar mejorar la  
eficiencia de los algoritmos anteriores, y lo haremos siguiendo una idea de  
[BEN89]. Para ello, dividiremos el problema en tres subproblemas más pequeños,  
sobre cuyas soluciones construiremos la solución total.  
En este caso la subsecuencia de suma máxima puede encontrarse en uno de tres  
lugares. O está en la primera mitad del vector, o en la segunda, o bien contiene al  
punto medio del vector y se encuentra en ambas mitades. Las tres soluciones se  
combinan mediante el cálculo de su máximo para obtener la suma pedida.  
Los dos primeros casos pueden resolverse recursivamente. Respecto al tercero,  
podemos calcular la subsecuencia de suma máxima de la primera mitad que  
contenga al último elemento de esa primera mitad, y la subsecuencia de suma  
máxima de la segunda mitad que contenga al primer elemento de esa segunda  
mitad. Estas dos secuencias pueden concatenarse para construir la subsecuencia de  
suma máxima que contiene al elemento central de vector. Esto da lugar al siguiente  
algoritmo