# TEORÍA DE ALGORITMOS 1

## Guia de ejercicios: División y conquista

Por: ING. VÍCTOR DANIEL PODBEREZSKI
vpodberezski@fi.uba.ar

## Enunciados

Para cada ejercicio considere, además de resolver lo solicitado, expresar la relación de recurrencia, brindar pseudocódigo y calcular la complejidad temporal, espacial.

**Referencias**

- ⭐ Fácil
- ⭐⭐ Medio
- ⭐⭐⭐ Difícil

1. ⭐Un colaborador del laboratorio de “cálculo automatizado S.A” propone un nuevo método de multiplicación de matrices. Utiliza división y conquista partiendo la matrices en bloques de tamaño n/4 x n/4. Y el proceso de combinación de resultados llevara $\Theta(n^2)$. Se muestra vago en indicar la cantidad de subproblemas que creará cada paso. Le indican que este dato es fundamental para decidir si proseguir con esa línea de investigación o no. Actualmente utilizan el algoritmo de Strassen con una complejidad de $O(n\log_2(7))$. Siendo T (n) = aT (n/4) + Θ(n2), con a la información a determinar. ¿Cuál es la cantidad de subproblemas más grande que puede tomar la solución para que sea mejor que su algoritmo actual?

2. ⭐Se cuenta con un vector de “n” posiciones en el que se encuentran algunos de los primeros ”m” números naturales ordenados en forma creciente (m >= n). En el vector no hay números repetidos. Se desea obtener el menor número no incluido. Ejemplo: [1, 2, 3, 4, 5, 8, 9, 11, 12, 13, 14, 20, 22]. Solución: 6. Proponer un algoritmo de tipo división y conquista que resuelva el problema en tiempo inferior a lineal. Expresar su relación de recurrencia y calcular su complejidad temporal.

3. ⭐Se realiza un torneo con n jugadores en el cual cada jugador juega con todos los otros n-1. El resultado del partido solo permite la victoria o la derrota. Se cuenta con los resultados almacenados en una matriz. Queremos ordenar los jugadores como $P_1$, $P_2$, ..., $P_n$ tal que $P_1$ le gana a $P_2$, $P_2$ le gana a $P_3$, ..., $P_{n-1}$ le gana a $P_n$ (La relación “le gana a” no es transitiva). Ejemplo: $P_1$ le gana a $P_3$, $P_2$ le gana a $P_1$ y $P_3$ le gana a $P_2$. Solución: [P1, P3, P2]. Resolver por división y conquista con una complejidad no mayor a O(n log n).

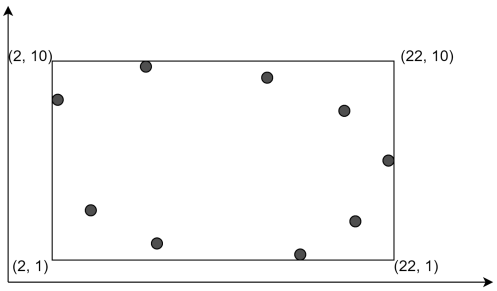
4. ⭐Para determinar si un número es primo existen varios algoritmos propuestos. Entre ellos el test de Fermat. Este es un algoritmo randomizado que opera de la siguiente manera: Dado un número entero “n”, seleccionar de forma aleatoria un número entero “a” coprimo a n. Calcular an-1 módulo n. Si el resultado es diferente a 1, entonces el número “n” es compuesto. La parte central de esta operatoria es la potenciación. Podríamos algorítmicamente realizarla de la siguiente manera:

   ```
   pot = 1
   Desde i=1 a n-1
     pot = pot * a
   ```

   En este caso se realizan o(n) multiplicaciones. Proponga un método usando división y conquista que resuelva la potenciación con menor complejidad temporal.

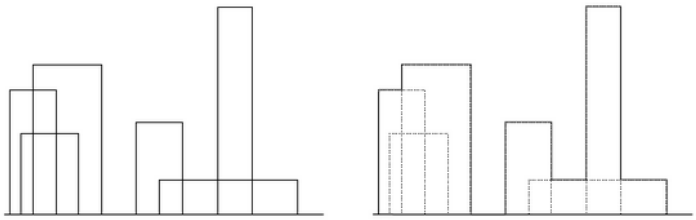
5. ⭐A raíz de una nueva regulación industrial un fabricante debe rotular cada lote que produce según un valor numérico que lo caracteriza. Cada lote está conformado por “n” piezas. A cada una de ellas se le realiza una medición de volumen. La regulación considera que el lote es válido si más de la mitad de las piezas tienen el mismo volumen. En ese caso el rótulo deberá ser ese valor. De lo contrario el lote se descarta. Actualmente cuenta con el proceso “A” que consiste en para cada pieza del lote contar cuántas de las restantes tienen el mismo volumen. Si alguna de las piezas corresponde al “elemento mayoritario”, lo rotula. De lo contrario lo rechaza. Un consultor informático impulsa una solución (proceso “B”) que considera la más eficiente: ordenar las piezas por volumen y con ello luego reducir el tiempo de búsqueda del elemento mayoritario. Nos contratan para construir una solución mejor (proceso “C”). Se pide:
   a. Exprese mediante pseudocódigo el proceso “A”.
   b. Explique si la sugerencia del consultor (proceso “B”) realmente puede mejorar el proceso. En caso afirmativo, arme el pseudocódigo que lo ilustre.
   c. Proponga el proceso “C” como un algoritmo superador mediante división y conquista. Explíquelo detalladamente y brinde pseudocódigo.

6. ⭐⭐Contamos con una lista ordenada de “n” coordenadas satelitales (latitud-longitud) que conforman un área con forma poligonal convexa. Queremos mostrar ese sector del mapa con el mayor tamaño posible en nuestra pantalla rectangular de la computadora. El programa que muestra el mapa acepta como parámetros 2 coordenadas para construir el rectángulo a mostrar: los correspondientes a los límites inferior izquierdo y superior derecho. Construya un algoritmo eficiente que resuelva el problema con complejidad O(logn). ¿Existe una solución de fuerza bruta que para “n” pequeños sea más eficiente? ¿para qué tamaño de n esto cambia?




   *Pista: profundizar en clases teóricas o apuntes*

7. ⭐⭐ Para la elaboración de un juego se desea construir un cielo nocturno de una ciudad donde se vea el contorno de los edificios en el horizonte. Cada edificio “ei“ está representado por rectángulos mediante la tripla (izquierda, altura, derecha). Dónde “izquierda” corresponde a la coordenada x menor, “derecha” la coordenada x mayor y altura la coordenada y. Todos los edificios inician en la coordenada 0 de y. Se cuenta con una lista de N edificios que llegan sin un criterio de orden específico. Se desea emitir como resultado el contorno representado como una lista de coordenadas “x” y sus alturas. Tenga en cuenta el siguiente ejemplo: Lista de edificios: (1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16) , (14, 3, 25), (19,18,22). Contorno: (1,11),(3,13),(9,0),(12,7),(16,3),(19,18),(22,3),(25,0). Presentar un algoritmo utilizando división y conquista que dado el listado de edificios retorna como resultado el contorno de la ciudad.



8. ⭐⭐Esta peculiar empresa se dedica a cubrir patios cuadrados de n*n metros (“n” es un número entero potencia de 2 y mayor o igual a 2). Cuenta con baldosas especiales que tienen forma en L (como se muestra en celeste en la imágen). Las baldosas no se pueden cortar. Todo patio cuenta con un único sumidero de agua de lluvia. Ocupa 1x1 metro y su ubicación depende del patio (Se muestra en la figura de ejemplo como un cuadrado negro). Nos piden que, dado un patio con un valor “n” y una ubicación del sumidero en una posición x,y desde la punta superior izquierda, determinemos cómo ubicar las baldosas. Presentar un algoritmo que lo resuelva utilizando división y conquista.

| | | | |
|---|---|---|---|
| (0,0) | (0,1) | (0,2) | (0,3) |
| (1,0) | (1,1) | (1,2) | (1,3) |
| (2,0) | **(2,1)** | (2,2) | (2,3) |
| (3,0) | (3,1) | (3,2) | (3,3) |

9. ⭐⭐Se realiza un experimento de conductividad de un nuevo material en aleación con otro. Se formaron muestras numeradas de 1 a n. A mayor número, mayor concentración del nuevo material. Además se realizaron “n” mediciones a diferentes temperaturas de conductividad para cada muestra. Los resultados fueron expresados en una matriz M de nxn. Se observa que un mismo material cuanto mayor temperatura tiene mayor conductividad. Además, cuanto mayor concentración a la misma temperatura, también mayor conductividad. En conclusión podemos, al analizar la matriz, ver dos progresiones. Cada fila tiene números ordenados de forma creciente y cada columna tiene números ordenados de forma creciente. Dada la matriz M, los experimentadores quieren encontrar en qué posición se encuentra un determinado número. Proponga una solución utilizando división y conquista.

10. ⭐⭐Todos los años la asociación de un importante deporte individual profesional realiza una preclasificación de los n jugadores que terminaron en las mejores posiciones del ranking

para un evento exclusivo. En la tarjeta de invitación adjuntan el número de posición en la que está actualmente y a cuantos rivales invitados superó en el ranking comparado el año pasado. Contamos con un listado que tiene el nombre del jugador y la posición del ranking del año pasado ordenado por el ranking actual. Ejemplo: LISTA: A,3 | B,4 | C,2 | D,8 | E,6 | F,5. Se puede ver que el jugador “A” superó al jugador “C”. El jugador “B” superó al jugador “C”. El jugador “C” no superó a ninguno de los invitados. Etc. Proponer una solución utilizando la metodología de división y conquista.

11. ⭐Una agencia gubernamental tiene un conjunto de "n" agentes dispersos por el país. Para una misión urgente requiere utilizar dos de ellos. Cada agente tiene una ubicación (x,y). Se dispone de un helicóptero para buscarlos. Generar una solución por división y conquista que indique cuáles son los 2 agentes más cercanos, cuál es su distancia y dónde debería ir el helicóptero a buscarlo.

12. ⭐Dado “L” un listado ordenado de “n” elementos y un elemento “e” determinado. Deseamos conocer la cantidad total de veces que “e” se encuentra en “L”. Podemos hacerlo en tiempo O(n) por fuerza bruta. Presentar una solución utilizando división y conquista que mejore esta complejidad.

13. ⭐Un conjunto de “n” personas votó de forma anónima entre un conjunto de “o” opciones (con o<<n). El resultado de la votación lo tenemos en un vector de n posiciones ordenado por opción seleccionada. Queremos determinar cuántos votos tuvo cada una de las opciones. Podemos hacerlo simplemente recorriendo el vector en O(n). Sin embargo, utilizando división y conquista se puede lograr en un tiempo inferior. Presentar y analizar una solución utilizando división y conquista que logre lo solicitado.

14. ⭐Una encuesta de internet pidió a personas que ordenen un conjunto de “n” películas comenzando por las que más les gusta a las que menos. Con los resultados quieren encontrar quienes comparten gustos entre sí. Nos solicitan generar un algoritmo, que basándose en el concepto de inversión, compare entre pares de personas y determine qué tan compatibles o incompatibles son. Proponer un algoritmo utilizando división y conquista que lo resuelva.

15. ⭐⭐Dentro de un país existen dos colonias subacuáticas cada una de ellas con “n” habitantes. Cada habitante tiene su documento de identidad único identificado por un número. Para una tarea especial se decidió seleccionar a aquella persona que vive en alguna de las colonias cuyo número de documento corresponda a la mediana de todos los números de documento presentes en ellas. Por una cuestión de protocolo no nos quieren dar los listados completos de documentos. Solo nos responden de cada colonia ante la consulta “Cual es el documento en la posición X de todos los habitantes de la isla ordenados de mayor a menor”. Utilizando esto, proponer un algoritmo utilizando división y conquista que resuelva el problema con la menor cantidad posibles de consultas. Analizar complejidad espacial y temporal.

16. ⭐ Se cuenta con un vector V de “n” elementos. Este vector visto de forma circular está ordenado. Pero no necesariamente en la posición inicial se encuentra el elemento más pequeño. Deseamos conocer la cantidad total de rotaciones que presenta “V”. Ejemplo: V = [6, 7, 9, 2, 4, 5] se encuentra rotado en 3 posiciones. Podemos hacerlo en tiempo O(n) por

fuerza bruta. Presentar una solución utilizando división y conquista que mejore esta complejidad.

17. ⭐⭐Sea S un conjunto de n puntos en el plano. Definimos que un punto p=(x,y) ∈ S es un punto máximo en S si no existe otro punto p'=(x',y') ∈ S tal que x ≤x' e y≤y'. Queremos obtener todos los puntos de S que sean máximos. Plantear un algoritmo de división y conquista para que lo resuelva. Analizar su complejidad computacional. (HINT: De ser requerido puede realizar un preorden del conjunto)

## Ejercicios resueltos

Cada ejercicio resuelto busca mostrar cómo se debe analizar, resolver y justificar la resolución del ejercicio, tanto en un trabajo práctico o en un parcial

1. ⭐ ⭐ Dado un vector A de “n” números enteros (tanto positivos como negativos) queremos obtener el subvector cuya suma de elementos sea mayor a la suma de cualquier otro subvector en A. Ejemplo: Array: [-2, -5, 6, -2, -3, 1, 5, -6]. Solución: [6, -2, -3, 1, 5]. Resolver el problema de subarreglo de suma máxima por división y conquista.

**Explicación**

Para resolver el ejercicio se partirá el array en 2 partes, y de forma recurrente invocar a la misma función (maxSubArray) hasta llegar al caso base, donde el array resultante contiene solo 1 elemento. En dicho caso se devuelve el mismo elemento.
Cada paso, además de dividir el array en dos partes y obtener el máximo sub array de cada parte, procederá a ejecutar la operación de merge, donde primero se buscará el máximo sub array considerando al elemento central del array. Para esto, se deben considerar 3 máximos, el máximo subarray de la parte izquierda del array, el máximo subarray de la parte derecha del array y la suma entre ambos máximos. Ese máximo nos indicará el máximo subarray consideran al elemento central. Luego el máximo subarray final se calcula como el máximo subarray considerando al elemento central y los máximos subarray de cada parte.

**Ecuación de recurrencia y análisis de complejidad**

$T(N) = 2.T(N/2) + O(n)$

$a = 2$
$b = 2$
$c = n$

La complejidad O(n) surge de calcular el máximo subarray considerando al elemento central, donde se recorren todos los elementos del array para poder calcularlo.

Aplicando el 2do caso del Teorema del Maestro obtenemos una complejidad temporal de $O(N.\log(N))$.

La complejidad espacial es O(N), dado que en cada recurrencia solo se calculan y almacenan los valores de los máximos calculados O(1). Luego, considerando que podemos tener N nodos en el peor de los casos, la complejidad espacial total resulta O(N).

**Pseudocódigo**

Python

```
function maxSubArray(array):

	# caso base
	if array.lenght == 1
		return array[0]

	# dividimos el array en 2
	lenght = array.lenght
	first_part = array[0:lenght/2]
	second_part = array[lenght/2:lenght]


	# calculamos la máxima suma partiendo del elemento intermedio, tanto a
	derecha como a izquierda

	curent_max_left = 0
	max_left = -inf
	for i = lenght/2 to 0:
		curent_max_left = curent_max_left + array[i]
		max_left = max(max_left, curent_max_left)

	curent_max_right = 0
	max_right = -inf
	for i = lenght/2 to lenght:
		curent_max_right = curent_max_right + array[i]
		max_right = max(max_right, curent_max_right)

	max_mid_subarray = max(max_left, max_right, max_left + max_right)


	# comparamos el máximo entre la suma del primer subarray, el segundo y el
	máximo subarray intermedio

	return max(
		maxSubArray(first_subarray),
		maxSubArray(second_subarray),
		max_mid_subarray
	)
```