95.12 Algoritmos y Programación II

Práctica 6: diseño de algoritmos

Notas preliminares

- El objetivo de esta práctica es no sólo ejercitar el diseño de algoritmos, fundamentalmente a través de la técnica de divide and conquer, sino además afianzar y consolidar los conceptos de análisis de algoritmos y complejidad.
- Los ejercicios marcados con el símbolo ♣ constituyen un subconjunto mínimo de ejercitación. No obstante, recomendamos fuertemente realizar todos los ejercicios.

Ejercicio 1 ♣

(a) Enumerar las tres etapas de un algoritmo divide & conquer y explicar en qué consiste cada una.

(b) Para cada etapa, identificar por lo menos un algoritmo conocido cuya implementación de dicha etapa sea no trivial.

Ejercicio 2 ♣

Diseñar un algoritmo recursivo para calcular a^n, en donde $n \in \mathbb{N}$ y $a \in \mathbb{R}$. La complejidad de peor caso del algoritmo debe ser una $O(\log n)$.

Ejercicio 3 ♣

Sean $A[1 \dots n]$ y $B[1 \dots m]$, con $n \geq m$, dos arreglos de puntos en el plano.

(a) Proponer un algoritmo que corra en tiempo $O(n \log m)$ y encuentre la cantidad de puntos que A y B tienen en común. Por ejemplo, dados

$$A = \langle (4,2), (1,7), (5,10), (\pi, 0) \rangle$$

$$B = \langle (0,0), (4,2), (5,10) \rangle$$

el algoritmo debe devolver 2.

(b) Calcular la complejidad espacial del algoritmo desarrollado.

Ejercicio 4

Dados k arreglos ordenados, cada uno de ellos conteniendo n elementos, nos interesa combinarlos en un único arreglo ordenado de kn elementos.

(a) Supongamos que, para ello, utilizamos el algoritmo de mezcla que utiliza MergeSort, de la siguiente manera: en primera instancia, se mezclan los dos primeros arreglos, luego se mezcla el resultado con el tercero y así sucesivamente. ¿Cuál es la complejidad temporal de peor caso de este algoritmo?

(b) Proponer un algoritmo más eficiente y que utilice la técnica de dividir y conquistar para resolver el problema.

Ejercicio 5

Se tienen n bolillas indistinguibles entre las cuales hay una ligeramente más pesada que las demás. Todas las restantes registran el mismo peso.

(a) Formular un algoritmo que implemente la técnica de dividir y conquistar y que corra en tiempo estrictamente inferior que $O(n)$ para encontrar la bolilla pesada. Asumir para ello que se dispone de una función `másPesado` que corre en tiempo constante y que, dados dos subconjuntos arbitrarios de bolillas, indica cuál de éstos es el más pesado (o si ambos pesan lo mismo).

(b) Supongamos ahora que `másPesado` corre en tiempo $O(\log m_1 + \log m_2)$, siendo m_i la cantidad de bolillas en el subconjunto i pasado como argumento a dicha función. Analizar cómo impacta este cambio en la complejidad temporal del algoritmo desarrollado.

Ejercicio 6

Proponer un algoritmo que, dados dos números naturales n y k, encuentre la parte entera de la raíz k-ésima de n. En otras palabras, el algoritmo debe computar $\lfloor \sqrt[k]{n} \rfloor$. Además, el algoritmo debe correr en tiempo estrictamente inferior que $O(n)$ (asumiendo que las operaciones aritméticas involucradas toman tiempo constante).

Por ejemplo, si $n = 345651$ y $k = 7$, el algoritmo debe devolver 6 pues $\sqrt[7]{345651} \approx 6{,}1835$.

Ejercicio 7

Las *matrices de Hadamard* $H_0, H_1, \ldots$ se definen de la siguiente manera:

- H_0 es la matriz $[1]$ (de dimensión 1×1)
- Para $k > 0$, H_k es la matriz de dimensión $2^k \times 2^k$ dada por

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$$

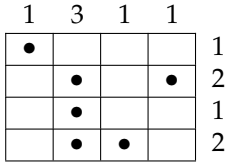
Dado un vector v de dimensión $n = 2^k$, proponer un algoritmo para calcular el producto $H_k v$ en tiempo estrictamente inferior que $O(n^2)$.

Ejercicio 8 ♣

Dada una secuencia $s = \langle s_1, \ldots, s_n \rangle$ de números enteros distintos ordenados ascendentemente, proponer un algoritmo de complejidad temporal estrictamente menor que $O(n)$ para determinar si en s existe algún i tal que $i = s_i$. Por ejemplo, para la secuencia $\langle -5, -1, 2, 3, 5, 8, 15 \rangle$, el algoritmo debe responder afirmativamente dado que el 5-ésimo elemento es exactamente el número 5.

Ejercicio 9

El célebre artista Donaldo Noot, durante la confección de su última obra de arte, se topó con el siguiente problema: en su lienzo, dividido en $n \times n$ sectores, debe pintar exactamente r_i de ellos en la i-ésima fila y exactamente c_j en la j-ésima columna ($0 \leq r_i, c_j \leq n$). Al estar atravesando un delicado momento económico, Donaldo no puede darse el lujo de desperdiciar cartuchos de témpera. La siguiente figura muestra un lienzo de 4×4 pintado exitosamente:



(a) Para asisitir a Donaldo, proponer un algoritmo que permita pintar el lienzo de forma óptima (i.e., utilizando una cantidad de cartuchos mínima). Asumimos que se requiere un cartucho por cada sector pintado. También asumimos que existe siempre una forma de pintar el lienzo satisfaciendo las restricciones impuestas.

(b) Calcular la complejidad temporal de peor caso del algoritmo diseñado, obteniendo para ello una expresión asintótica en términos de n y k, siendo k la cantidad total de sectores que deben pintarse (notar que $k = \sum r_i = \sum c_j$).

Ejercicio 10

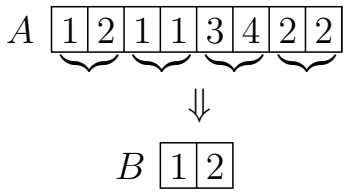
Decimos que un arreglo $A[1 \dots n]$ contiene un elemento *mayoritario* si existe $m \in A$ tal que $|A|_m > n/2$ (i.e., la cantidad de apariciones de m en A es estrictamente mayor que $n/2$).

(a) Escribir un algoritmo que corra en tiempo $O(n \log n)$ y que, dado A, determine si este elemento existe y lo retorne en caso afirmativo. A los efectos de obtener soluciones generales para este problema, **no puede asumirse** que existe una relación de orden entre los elementos de A.

(b) Supongamos que n es par. Sea B un arreglo construido de la siguiente manera:

- Considerar los elementos $A[2i+1]$ y $A[2i+2]$ para cada $i = 0 \dots \frac{n-2}{2}$.
- Si éstos son iguales, agregar uno de ellos a B.

La siguiente figura muestra un ejemplo de construcción de B:



(i) Probar que, si m es mayoritario en $A[1 \dots n]$ y n es par, entonces m es mayoritario en B.

(ii) Probar que la recíproca del punto anterior no es cierta (i.e., que B tenga elemento mayoritario no necesariamente implica que A también lo tenga).

(iii) Probar que, si m es mayoritario en A, entonces $m = A[n]$ o bien m es mayoritario en $A[1 \dots n-1]$.

(iv) A partir de todo lo anterior, formular un algoritmo que encuentre, si existe, el elemento mayoritario de A.
Pista: el ítem previo es útil para cuando n es impar.

(v) Calcular la complejidad temporal del algoritmo diseñado. ¿Es eficiente?

(vi) Analizar si este algoritmo utiliza la técnica de dividir y conquistar.

Ejercicio 11

Una *tabla de Young* es una matriz de $m \times n$ en la que los elementos de cada fila se encuentran ordenados ascendentemente de izquierda a derecha y los elementos de cada columna se encuentran ordenados ascendentemente de arriba hacia abajo. Algunos elementos de una tabla de Young pueden ser $\star$, a quienes entenderemos como elementos inexistentes y tales que cualquier otro elemento es estrictamente menor que ellos. De esta manera, una tabla de Young puede utilizarse para almacenar $r \leq mn$ elementos.

(a) Dibujar dos tablas de Young de 4×4 distintas que contengan los elementos $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$.

(b) Proponer un algoritmo que corra en tiempo $O(m+n)$ para remover el mínimo elemento de una tabla de Young de $m \times n$.

Ejercicio 12 ♣

(a) Diseñar un algoritmo que reciba un arreglo $A[1 \dots n]$ de números y encuentre y devuelva la suma máxima σ_A de alguno de sus subarreglos en tiempo $O(n \log n)$. Puesto en términos más formales, el algoritmo debe calcular lo siguiente:

$$\sigma_A = \text{máx} \left\{ \sum_{i \le k \le j} A[k] \,/\, 1 \le i \le j \le n \right\}$$

Por ejemplo, si $A = \langle 3, -4, 5, -1, 5, 6, 10, -9, -2, 8 \rangle$, el algoritmo debe devolver $\sigma_A = 25$. Esta suma corresponde al subarreglo $\langle 5, -1, 5, 6, 10 \rangle$.

(b) Modificar el algoritmo anterior para que corra en tiempo $O(n)$.
Pista: utilizar la técnica de generalización de funciones.

Ejercicio 13

(a) Dado un arreglo $A[1 \dots n]$ de números arbitrarios, nos interesa calcular la mínima diferencia δ_A entre dos elementos cualesquiera de A. Puesto en términos más formales, nos interesa computar:

$$\delta_A = \text{mín} \{ |A[i] - A[j]| \,/\, 1 \le i < j \le n \}$$

Dar un algoritmo que implemente la técnica de dividir y conquistar para encontrar este valor.

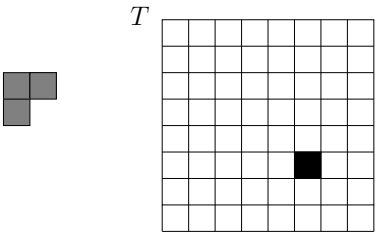
(b) Calcular la complejidad temporal de peor caso, y comentar informalmente qué sucede en el caso promedio.

Ejercicio 14

Diseñar un algoritmo que, dados dos arreglos $A_1[1 \dots n]$ y $A_2[1 \dots n]$ ordenados ascendentemente, encuentre la mediana[1] de $A_1 \cup A_2$ en tiempo $\Theta(\log n)$.

Ejercicio 15 ♣

Un *tromino* es una pieza en forma de L construida a partir de tres cuadrados adyacentes de 1×1. Se requiere cubrir por completo con trominos un tablero T de $2^n \times 2^n$ casilleros en donde exactamente uno de ellos se encuentra pintado. Los trominos deben cubrir todos los casilleros a excepción del pintado, y sin solaparse. La figura que sigue muestra un tromino y un tablero T de 8×8 casilleros.



(a) Proponer un algoritmo que utilice la técnica de dividir y conquistar para solucionar este problema.

(b) Calcular la complejidad temporal de peor caso del algoritmo encontrado.

Ejercicio 16

La ciudad Algoritmópolis, cuna de eminentes programadores, enfrenta una grave crisis de escasez de agua corriente. Al estar próxima a un cordón montañoso donde el agua de lluvia suele acumularse, los

[1] La mediana de $A[1 \dots k]$ es el elemento $\lceil k/2 \rceil$-ésimo en el arreglo que resulta de ordenar a A.

ingenieros de la ciudad decidieron implementar un mecanismo para almacenar transitoriamente el agua en ciertos *puntos de recolección*, desde donde luego se canalizará a la red urbana.

Cuando la lluvia cae sobre el cordón montañoso, se desliza a lo largo de las pendientes de las cumbres hasta caer en un *punto de recolección*, que es esencialmente un punto con altitud inferior a todos sus puntos vecinos. La figura muestra un cordón con sus altitudes en donde todos los puntos de recolección aparecen identificados.

Dado un arreglo $A[1 \dots n]$ conteniendo las n altitudes del cordón montañoso, todas distintas entre sí, proponer un algoritmo eficiente que utilice la técnica de dividir y conquistar para encontrar un punto de recolección y ayudar a los ingenieros de Algoritmópolis a completar su crítica tarea.

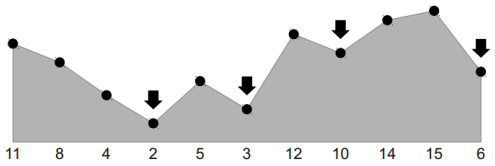


Figura 1: un cordón montañoso y sus puntos de recolección

Ejercicio 17 ♣

Un arreglo $A[1 \dots 2n+1]$ de números se dice *serpenteante* siempre que

$$A[1] \leq A[2] \geq A[3] \leq A[4] \geq \cdots \leq A[2n] \geq A[2n+1]$$

(a) Dado un arreglo arbitrario $A[1 \dots 2n+1]$, proponer un algoritmo que utilice la técnica de dividir y conquistar para convertir a A en un arreglo serpenteante.
Pista: separar el escenario donde n es par del escenario donde n es impar.

(b) Calcular la complejidad temporal del algoritmo desarrollado. ¿Es eficiente?

Ejercicio 18

Se tiene un arreglo $A[1 \dots n]$ de números enteros en el que cierta posición i, $1 \leq i \leq n$, verifica las siguientes propiedades, siendo $k = A[i] > 0$:

- $A[i \dots n]$ contiene únicamente múltiplos consecutivos de k, no necesariamente en orden.
- $A[1 \dots i-1]$ contiene únicamente números consecutivos arrancando desde 1 y sustituyendo cada eventual múltiplo m de k por $-m$. Al igual que en el caso anterior, estos números tampoco se disponen en un orden particular.

Nos interesa computar $S(A) = \sum A[j]$, la suma de los números de A. A modo ilustrativo, el arreglo A de la figura es tal que $i = 8$, $k = 3$ y $S(A) = 40$:

-3	2	4	7	1	-6	5	3	12	6	9

Observar además que, en este caso, $A[1 \dots i-1]$ contiene los números enteros consecutivos entre 1 y 7 en un orden arbitrario, a excepción de los dos múltiplos de $k = 3$, 3 y 6, cuyos lugares corresponden a -3 y -6 respectivamente. En cuanto a $A[i \dots n]$, notar que aparecen allí los primeros cuatro múltiplos de 3, 3, 6, 9 y 12, también en un orden arbitrario.

Proponer un algoritmo **eficiente** que, dados A y k, permita calcular $S(A)$. Explicar claramente el criterio utilizado para determinar que se trata de un algoritmo eficiente.

Pista: dado $m \in \mathbb{N}$, la cantidad de múltiplos de m entre 1 y N viene dada por $\lfloor N/m \rfloor$.