75.04 Algoritmos y Programación II

Práctica 5: divide & conquer

Notas preliminares

- El objetivo de esta práctica es no solamente ejercitar el diseño de algoritmos a través de la técnica de división y conquista 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

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 4

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 5 ♣

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 6

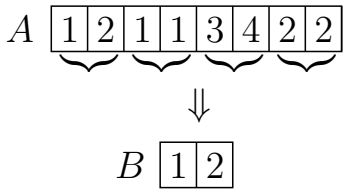
Decimos que un arreglo $A[1 \dots n]$ contiene un elemento *mayoritario* si existe $m \in A$ tal que $|A|_m > n/2$ (i.e., la cantidad de apariciones de m en A es estrictamente mayor que $n/2$).

(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:



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

(b) 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).

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

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

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

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

Ejercicio 7 ♣

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

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

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

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

Ejercicio 8

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

$$\boldsymbol{\delta}_A = \text{mín} \left\{ |A[i] - A[j]| \,/\, 1 \le i < j \le n \right\}$$

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

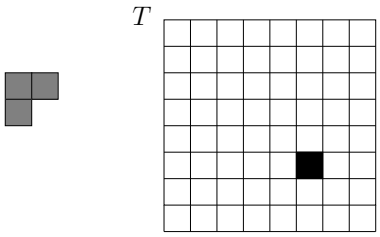
(b) Calcular la complejidad temporal de peor caso, y comentar informalmente qué sucede en el caso promedio.

Ejercicio 9

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 10 ♣

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.

[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.