

# Algoritmos y Estructuras de Datos II

## Práctica 6 – Dividir y conquistar

### Notas preliminares

- Los objetivos de esta práctica son:
  - Introducir la técnica de Dividir y conquistar.
  - Identificar los pasos requeridos para resolver problemas con dicha técnica.
  - Desarrollar optimizaciones para alcanzar una mayor eficiencia de los algoritmos.
  - Aprender a calcular la complejidad de algoritmos recursivos.
- Los ejercicios marcados con el símbolo ★ constituyen un subconjunto mínimo de ejercitación. Sin embargo, aconsejamos fuertemente hacer todos los ejercicios.

---

### Ejercicio 1 ★

Escriba un algoritmo con dividir y conquistar que determine si un arreglo de tamaño potencia de 2 es *más a la izquierda*, donde “más a la izquierda” significa que:

- La suma de los elementos de la mitad izquierda superan los de la mitad derecha.
- Cada una de las mitades es a su vez “más a la izquierda”.

Por ejemplo, el arreglo  $[8, 6, 7, 4, 5, 1, 3, 2]$  es “más a la izquierda”, pero  $[8, 4, 7, 6, 5, 1, 3, 2]$  no lo es.

Intente que su solución aproveche la técnica de modo que complejidad del algoritmo sea estrictamente menor a  $O(n^2)$ .

### Ejercicio 2 ★

Tenemos un arreglo  $a = [a_1, a_2, \dots, a_n]$  de  $n$  enteros distintos (positivos y negativos) *en orden estrictamente creciente*. Queremos determinar si existe una posición  $i$  tal que  $a_i = i$ . Por ejemplo, dado el arreglo  $a = [-4, -1, 2, 4, 7]$ ,  $i = 4$  es esa posición.

Diseñar un algoritmo dividir y conquistar eficiente (de complejidad de orden estrictamente menor que lineal) que resuelva el problema. Calcule y justifique la complejidad del algoritmo dado.

### Ejercicio 3 ★

Encuentre un algoritmo para calcular  $a^b$  en tiempo logarítmico en  $b$ . Piense cómo reutilizar los resultados ya calculados. Justifique la complejidad del algoritmo dado.

### Ejercicio 4 ★

Calcule la complejidad de un algoritmo que utiliza  $T(n)$  pasos para una entrada de tamaño  $n$ , donde  $T$  cumple:

- |                              |                              |                             |
|------------------------------|------------------------------|-----------------------------|
| ■ $T(n) = T(n-2) + 5$        | ■ $T(n) = 2T(n-1)$           | ■ $T(n) = 2T(n-4)$          |
| ■ $T(n) = T(n-1) + n$        | ■ $T(n) = T(n/2) + n$        | ■ $T(n) = 2T(n/2) + \log n$ |
| ■ $T(n) = T(n-1) + \sqrt{n}$ | ■ $T(n) = T(n/2) + \sqrt{n}$ | ■ $T(n) = 3T(n/4)$          |
| ■ $T(n) = T(n-1) + n^2$      | ■ $T(n) = T(n/2) + n^2$      | ■ $T(n) = 3T(n/4) + n$      |

Intentar estimar la complejidad para cada ítem directamente y luego calcularla utilizando el teorema maestro de ser posible. Para simplificar los cálculos se puede asumir que  $n$  es potencia o múltiplo de 2 o de 4 según sea conveniente.

### Ejercicio 5 ★

Suponga que se tiene un método *potencia* que, dada una matriz cuadrada  $A$  de orden  $4 \times 4$  y un número  $n$ , computa la matriz  $A^n$ . Dada una matriz cuadrada  $A$  de orden  $4 \times 4$  y un número natural  $n$  que es potencia de

2 (i.e.,  $n = 2^k$  para algun  $k \geq 1$ ), desarrollar, utilizando la técnica de dividir y conquistar y el método *potencia*, un algoritmo que permita calcular

$$A^1 + A^2 + \dots + A^n.$$

Calcule el número de veces que el algoritmo propuesto aplica el método *potencia*. Si no es estrictamente menor que  $O(n)$ , resuelva el ejercicio nuevamente.

### Ejercicio 6 ★

Dado un árbol binario cualquiera, diseñar un algoritmo de dividir y conquistar que devuelva la máxima distancia entre dos nodos (es decir, máxima cantidad de ejes a atravesar). El algoritmo no debe hacer recorridos innecesarios sobre el árbol.

### Ejercicio 7 ★

La cantidad de parejas en desorden de un arreglo  $A[1 \dots n]$  es la cantidad de parejas de posiciones  $1 \leq i < j \leq n$  tales que  $A[i] > A[j]$ . Dar un algoritmo que calcule la cantidad de parejas en desorden de un arreglo y cuya complejidad temporal sea estrictamente mejor que  $O(n^2)$  en el peor caso. **Hint:** Considerar hacer una modificación de un algoritmo de sorting.

### Ejercicio 8 ★

Se tiene una matriz booleana  $A$  de  $n \times n$  y una operación *conjunciónSubmatriz* que toma  $O(1)$  y que dados 4 enteros  $i_0, i_1, j_0, j_1$  devuelve la conjunción de todos los elementos en la submatriz que toma las filas  $i_0$  hasta  $i_1$  y las columnas  $j_0$  hasta  $j_1$ . Formalmente:

$$\text{conjunciónSubmatriz}(i_0, i_1, j_0, j_1) = \bigwedge_{i_0 \leq i \leq i_1, j_0 \leq j \leq j_1} A[i, j]$$

1. Dar un algoritmo que tome tiempo estrictamente menor que  $O(n^2)$  que calcule la posición de algún false, asumiendo que hay al menos uno. Calcular y justificar la complejidad del algoritmo.
2. Modificar el algoritmo anterior para que cuente cuántos false hay en la matriz. Asumiendo que hay a lo sumo 5 elementos false en toda la matriz, calcular y justificar la complejidad del algoritmo.
3. Si obtuvo una complejidad  $O(n^2)$  en el punto anterior, mejore el algoritmo y/o el cálculo para obtener una complejidad menor.

### Ejercicio 9

Dados dos arreglos de naturales, ambos ordenados de manera creciente, se desea buscar, dada una posición  $i$ , el  $i$ -ésimo elemento de la unión de ambos. Dicho de otra forma, el  $i$ -ésimo del resultado de hacer merge ordenado entre ambos arreglos. Notar que no es necesario hacer el merge completo. Se puede asumir que cada natural aparece a lo sumo en uno de los arreglos, y a lo sumo una vez.

- a) Implementar la función

$$\text{iésimoMerge}(\text{in } A: \text{arreglo}(\text{nat}), \text{in } B: \text{arreglo}(\text{nat}), \text{in } i: \text{nat}) \rightarrow \text{nat}$$

que resuelve el problema planteado. La función debe ser de tiempo  $O(\log^2 n)$ , donde  $n = \text{tam}(A) = \text{tam}(B)$ .

- b) Calcular y justificar la complejidad del algoritmo propuesto.
- c) Intente resolver el mismo problema en tiempo  $O(\log n)$  (este ítem es bastante mas difícil, se incluye como desafío adicional).

### Ejercicio 10

Se tienen dos arreglos de  $n$  naturales  $A$  y  $B$ .  $A$  está ordenado de manera creciente y  $B$  está ordenado de manera decreciente. Ningún valor aparece mas de una vez en el mismo arreglo. Para cada posición  $i$  consideramos la diferencia absoluta entre los valores de ambos arreglos  $|A[i] - B[i]|$ . Se desea buscar el mínimo valor posible de dicha cuenta. Por ejemplo, si los arreglos son  $A = [1, 2, 3, 4]$  y  $B = [6, 4, 2, 1]$  los valores de las diferencias son 5, 2, 1, 3 y el resultado es 1.

- a) Implementar la función

$$\text{minDif}(\text{in } A: \text{arreglo}(\text{nat}), \text{in } B: \text{arreglo}(\text{nat})) \rightarrow \text{nat}$$

que resuelve el problema planteado. La función debe ser de tiempo  $O(\log n)$ , dónde  $n = \text{tam}(A) = \text{tam}(B)$ .

- b) Calcular y justificar la complejidad del algoritmo propuesto.

### Ejercicio 11

Se tiene un arreglo de números naturales  $A$ . Además se cuenta con estructuras adicionales sobre el arreglo que proveen la función

$$\text{aparece?}(\text{in } A: \text{arreglo}(\text{nat}), \text{in } i: \text{nat}, \text{in } j: \text{nat}, \text{in } e: \text{nat}) \rightarrow \text{bool}$$

que dado el arreglo  $A$ , índices  $i, j$  y un natural  $e$ , devuelve *true* si y sólo si  $e = A[k]$  para algún  $k$  tal que  $i \leq k \leq j$ . Además se sabe que *aparece?* toma tiempo  $O(\sqrt{j-i+1})$ , es decir, la raíz cuadrada del tamaño del intervalo de búsqueda.

Se desea encontrar un algoritmo sublineal que encuentra el índice de un elemento  $e$  en el arreglo  $A$ , asumiendo que tal elemento existe en el arreglo. El resultado de la función es justamente el índice  $i$  tal que  $A[i] = e$ .

- a) Implementar la función

$$\text{ubicar?}(\text{in } A: \text{arreglo}(\text{nat}), \text{in } e: \text{nat}) \rightarrow \text{nat}$$

que resuelve el problema planteado. La función debe ser de tiempo estrictamente menor a  $O(n)$ , dónde  $n = \text{tam}(A) = \text{tam}(B)$  (formalmente, la complejidad del algoritmo **no** debe pertenecer a  $\Omega(n)$ )

- b) Calcular y justificar la complejidad del algoritmo propuesto.

### Ejercicio 12 (Difícil)

Se tiene un tablero rectangular de  $n \times n$  posiciones, con  $n$  potencia de 2, donde una de las posiciones se encuentra inicialmente ocupada. Diseñar un algoritmo con la técnica de dividir y conquistar para rellenar todas las posiciones del tablero con figuras que ocupan 3 posiciones y tienen forma de  $L$ . Formalmente, podemos definir el problema de la siguiente forma: dado un valor  $n$  y un par de valores  $i_0, j_0$  ( $1 \leq i_0, j_0 \leq n$ ), se quiere encontrar una matriz  $B$  de tamaño  $n \times n$  tal que:

- $B[i_0, j_0] = 0$ ,
- Todos los valores entre 1 y  $(n^2 - 1)/3$  aparecen exactamente tres veces en  $B$ , y
- Para todo  $1 \leq i, j \leq n$  tal que  $(i, j) \neq (i_0, j_0)$ , ocurre que el conjunto

$$\{B[x, y] \mid 1 \leq x, y \leq n \text{ e } i-1 \leq x \leq i+1 \text{ y } j-1 \leq y \leq j+1\}$$

contiene exactamente tres elementos con el valor  $B[i, j]$  (uno de los cuales es  $B[i, j]$ ).

- Ningun entero aparece más de dos veces en la misma fila o columna.

Por ejemplo, si  $n = 4$ , entonces la matriz  $B$  podría ser

con  $i_0 = 1$  y  $j_0 = 1$

$$\begin{pmatrix} 0 & 1 & 2 & 2 \\ 1 & 1 & 4 & 2 \\ 3 & 4 & 4 & 5 \\ 3 & 3 & 5 & 5 \end{pmatrix}$$

con  $i_0 = 3$  y  $j_0 = 2$

$$\begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 5 & 5 & 2 \\ 3 & 0 & 5 & 4 \\ 3 & 3 & 4 & 4 \end{pmatrix}$$

con  $i_0 = 4$  y  $j_0 = 2$

$$\begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 4 & 4 & 2 \\ 3 & 3 & 4 & 5 \\ 3 & 0 & 5 & 5 \end{pmatrix}$$