Práctica 2 - Divide and Conquer

NOTAS PRELIMINARES
Los objetivos de esta práctica son:

- Introducir la técnica de Divide & Conquer.
- Identificar los pasos requeridos para resolver problemas con Divide & Conquer.
- Desarrollar optimizaciones para alcanzar una mayor eficiencia de los algoritmos.

Para cada uno de los ejercicios, se debe indicar claramente cada una de las etapas del método Divide & Conquer, y justificar la complejidad temporal del algoritmo propuesto. Los ejercicios marcados con el simbolo * constituyen un subconjunto mínimo de ejercitación. Sin embargo, aconsejamos fuertemente hacer todos los ejercicios.

Ejercicio 1 *

Escriba un algoritmo de Divide & Conquer que determine si un arreglo es *más a la izquierda*, donde "más a la izquierda" significa que:

- La suma de los elementos de la mitad izquierda supera a la suma de 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.

Ejercicio 2 *

Tenemos un arreglo $a = [a_1, a_2, ..., a_n]$ 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 de Divide & Conquer eficiente (de complejidad de orden estrictamente menor que lineal en n) que resuelva el problema.

Ejercicio 3 *

Encuentre un algoritmo para calcular a^b en tiempo logarítmico en b. Piense cómo reutilizar los resultados ya calculados.

Ejercicio 4

Encuentre un algoritmo eficiente para calcular A^n, donde A es una matriz cuadrada.

Ejercicio 5 *

Suponga que se tiene un método `potencia(A, n)` que, dada un matriz cuadrada A de orden 4×4 y un número n, computa la matriz A^n. Dada una matriz cuadrada A de orden 4×4 y un número natural n que es potencia de 2 (es decir, $n = 2^k$ para alguna $k \geq 1$), desarrollar, utilizando la técnica de Divide & Conquer y el método `potencia`, un algoritmo que permita calcular:

$$A^1 + A^2 + \ldots + A^n$$

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 Divide & Conquer 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

Un *árbol rojo-negro*[1] es un árbol binario que cumple las siguientes propiedades:

1. Las hojas son negras.
2. Los nodos rojos sólo pueden tener hijos negros.

3. Todos los caminos desde la raíz hasta una hoja contienen el mismo número de nodos negros.

Dado un árbol binario y una función `color(nodo)` que toma un nodo del árbol y en $O(1)$ devuelve `negro` o `rojo`, se pide:

- Escribir un algoritmo que visite cada nodo a lo sumo una vez, y que utilice la técnica de Divide & Conquer, para determinar si el árbol es rojo-negro.
- Marcar claramente las distintas etapas del algoritmo.

[1] *Se utiliza para este ejercicio una versión levemente simplificada de la definición real de árboles rojo-negro. Estas estructuras de datos son árboles que se mantienen balanceados tras inserciones y eliminaciones de elementos.*

Ejercicio 8 *

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. *Pista*: Considerar hacer una modificación de un algoritmo de sorting.

Ejercicio 9

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 Divide & Conquer para rellenar todas las posiciones del tablero con figuras que ocupen 3 posiciones y tengan 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 $B[i, j]$).

Por ejemplo, si $n = 8$, $i_0 = 7$ y $j_0 = 3$, entonces la matriz B podra ser

$$\begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 4 & 4 \\ 1 & 5 & 5 & 2 & 3 & 6 & 6 & 4 \\ 7 & 5 & 8 & 8 & 9 & 9 & 6 & 10 \\ 7 & 7 & 8 & 11 & 11 & 9 & 10 & 10 \\ 12 & 12 & 13 & 13 & 11 & 14 & 15 & 15 \\ 12 & 16 & 16 & 13 & 14 & 14 & 17 & 15 \\ 18 & 16 & 0 & 19 & 20 & 17 & 17 & 21 \\ 18 & 18 & 19 & 19 & 20 & 20 & 21 & 21 \end{bmatrix}$$

Ejercicio 10

Se tiene una bolsa con una cantidad cualquiera de bolas de billar que, en apariencia, son todas iguales. Sin embargo, una de ellas es ligeramente más liviana que las demás. Por medio de una balanza de platos y realizando la menor cantidad de medidas, se desea determinar cuál es la bola diferente.

Vamos a asumir que las bolas estn numeradas de 1 a n, y que se cuenta con la siguiente función:

```
pesar(conj1: conj(nat), conj2: conj(nat)): int
```

Esta función devuelve -1 si `conj1` pesa más que `conj2`, 0 si pesan igual y 1 si `conj1` pesa menos que `conj2`.

- Escribir un algoritmo de complejidad temporal sublineal, de tipo Divide & Conquer para determinar el número de la bola más liviana (tenga en cuenta que la cantidad de bolas puede ser par o impar).
- Indicar la cantidad de pesadas necesarias en el peor caso en función de n, siendo n la cantidad total de bolas. Justifique su respuesta.