Práctico 6

Divide & Vencerás (Divide & Conquer)

EJERCICIO 1 (I)

Un cuadrado latino de orden n es una matriz cuadrada de enteros tal que cada fila y cada columna contiene una permutación de (*1, 2, … , n*). Por ejemplo:

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

Una forma de construir un cuadrado latino de orden *n* es con el siguiente algoritmo que genera las permutaciones en forma circular en la matriz A:

```
void cuadLatino (int **a,  int n){
  for (int i = 1; i <= n; i++)
     for (int j = 1; j <= n; j++)
          a[i-1][j-1] = ((i+j-1) % n) + 1;
}
```

a) Calcule el orden del algoritmo dado.
b) Escriba un algoritmo que aplique la estrategia de D&C para construir un cuadrado latino de orden *n*, sabiendo que n es una potencia de 2. Este algoritmo no tiene por qué generar un cuadrado latino como el de la figura.
c) Calcule el orden de su algoritmo.

EJERCICIO 2 (I)

En una planta se fabrican piezas elementales (no fabricadas a partir de ninguna otra pieza) y piezas compuestas. Las piezas compuestas se fabrican por composición de otras tres piezas, cada una de las cuales puede ser a su vez compuesta o elemental. Las piezas se identifican por un número de 1..*n*. Para representar la relación entre piezas se definió la siguiente estructura de datos:

```
struct pieza {
      bool elemental;
      int pieza1, pieza2, pieza3; // solo si no es elemental
}
typedef pieza* catalogo; // un vector de piezas
```

El índice del vector es el identificador de la pieza. Los campos *pieza1*, *pieza2*, *pieza3* son los identificadores de las piezas componentes.
Por otro lado, se define el *nivel* de una pieza como sigue:

- si la pieza es elemental su *nivel* es 0
- si la pieza es compuesta, su *nivel* es igual al máximo *nivel* de las componentes más 1.

Uno de los requerimientos de la planta es, dada una pieza a fabricar, determinar todas las piezas (elementales o no) con las que debe contar para poder construir la pieza dada, y saber además el orden en que deben ser hechas las mismas. Una forma de satisfacer este requerimiento es proporcionar una lista de parejas (pieza, nivel) ordenadas por nivel.

Escriba un algoritmo que utilice la técnica de D&C para resolver el requerimiento planteado.

EJERCICIO 3 (I)

Se da una lista de coordenadas (x,y), ordenadas de manera que el valor en x para el punto i sea menor que el valor en x para el punto $i + 1$. Se debe determinar (utlizando D&C) la menor de todas las distancias euclideas entre pares de puntos de la lista.

EJERCICIO 4 (I)

Se tienen n matrices M_1, ..., M_n de índices naturales y elementos de tipo r. Consideramos definidas las siguientes funciones:

filas (M_i) = "número de filas de la matriz M_i"
columnas (M_i) = "número de columnas de la matriz M_i"

Llamaremos $M_{i\,..\,j}$ a la matriz resultado del producto $M_i \cdot \ldots \cdot M_j$, con $i \leq j$.

El costo de calcular $M_{i\,..\,i+1}$ es el número de productos necesarios es decir,

costo ($M_{i..i+1}$) = filas (M_i) · columnas (M_i) · columnas (M_{i+1}),

siendo columnas (M_i) = filas (M_{i+1}).

El costo de calcular $M_{i..j}$ (siendo $j > i+1$) depende de la asociación de las matrices $M_i...M_j$ que se utilice en el cálculo, pues el producto de matrices es asociativo y por lo tanto para calcular $M_{i..j}$ se puede elegir la asociación que involucre menos operaciones.

Por ejemplo, sean las matrices M_1 de dimensión 10×20, M_2 de 20×30 y M_3 de 30×30, resulta:
costo $((M_1 \cdot M_2) \cdot M_3) = 10 \cdot 20 \cdot 30 + 10 \cdot 30 \cdot 30 = 15000$, mientras que
costo $(M_1 \cdot (M_2 \cdot M_3)) = 20 \cdot 30 \cdot 30 + 10 \cdot 20 \cdot 30 = 24000$.

Se desea un algoritmo para calcular la asociación que tiene el mínimo costo de obtener $M_{1..n}$

Se pide: a) Elegir una estructura de datos adecuada para almacenar una asociación de matrices.

b) Desarrollar un algoritmo, usando D&C, que calcule y retorne la asociación que tiene el mínimo costo de obtener $M_{1..n}$. ¿Cuál es el orden de ejecución del algoritmo?.

EJERCICIO 5 (I)

Usted es director de un torneo y necesita organizar un esquema de todos contra todos entre $n = 2.k$ jugadores. En este torneo cada uno juega exactamente un juego diario. Después de n-1 días debe haber tenido lugar un encuentro entre todo par de jugadores.

Proporcione un algoritmo que imprima quienes juegan cada día (considere los jugadores numerados del 0 al n-1).

EJERCICIO 6 (R)

Suponga que para representar polinomios de grado n se utiliza un vector de $n + 1$ elementos (indexado de 0 a n), donde en la posición i del vector se almacena el coeficiente de x^i en el polinomio. Por ejemplo, el vector [1,–2,0,2,–5] representa el siguiente polinomio de grado 4: $-5x^4 + 2x^3 - 2x + 1$.

Se quiere, dados n números reales, obtener un polinomio normalizado que tenga a dichos números como raíces. Un polinomio normalizado es aquel en el que el coeficiente del término de mayor grado es 1. Recordar que dadas las raíces $\alpha_1, \alpha_2 \ldots, \alpha_n$, el polinomio normalizado es $(x - \alpha_1)\cdot(x - \alpha_2)\cdot\ldots\cdot(x - \alpha_n)$. Lo que se quiere es obtener ese polinomio en la representación definida en el párrafo anterior.

Suponga dados los siguientes algoritmos:

`Grado_1`: dada α, devuelve un polinomio normalizado de grado 1 que tiene a α como única raíz, es decir, el polinomio $x - \alpha$. No realiza operaciones básicas.

`Mult_1`: dados, $i \geq 1$, un polinomio de grado 1 y un polinomio de grado i, devuelve el polinomio correspondiente a la multiplicación de los polinomios dados. Realiza i operaciones básicas.

`Mult_i`: dados, $i \geq 2$ y dos polinomios de grado i, devuelve el polinomio correspondiente a la multiplicación de los polinomios dados. Realiza $i \cdot \log i$ operaciones básicas.

Una posible solución al problema es la obtenida por el siguiente algoritmo:

```
float * Polin (float * raices)
{
     float * p = new float[2], * parcial = new float[n + 1];
     Grado_1 (raices[0], parcial);

     for (int i = 2; i <= n; i++)
     {
         Grado_1 (raices[i - 1], p);
         Mult_1 (i - 1, p, parcial);
     }

     delete [] p;
     return parcial;
}
```

a) Calcule el orden del tiempo de ejecución del algoritmo dado en función de n.
b) Escriba un algoritmo basado en Divide & Conquer que resuelva el mismo problema.
c) Calcule el orden del tiempo de ejecución de su algoritmo, suponiendo n potencia de 2.

EJERCICIO 7 (R)

Sean A y B dos matrices cuadradas de dimensiones n x n, el objetivo del problema es encontrar la matriz C (de dimensión n x n), resultado de multiplicar A por B.
Sea c_{ij} el elemento que se encuentra en la fila i y columna j de C, la manera convencional de calcularlo es mediante el producto escalar de la i-ésima fila de A por la j-ésima columna de B,

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Otro método, no convencional para calcular este producto es el de dividir cada matriz en cuatro submatrices de dimensiones $\frac{n}{2} x \frac{n}{2}$:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

donde por ejemplo, $A_{11} = \begin{bmatrix} a_{11} & \dots & a_{1\frac{n}{2}} \\ \dots & \dots & \dots \\ a_{\frac{n}{2}1} & \dots & a_{\frac{n}{2}\frac{n}{2}} \end{bmatrix}$

y calcular el producto como, $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

donde:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

a) Calcular la **cantidad de multiplicaciones** que realiza el método convencional.
b) Diseñar un algoritmo basado en *D&C* que calcule el producto de matrices siguiendo el método **no** convencional.
c) Calcular la **cantidad de multiplicaciones** que realiza su algoritmo.

EJERCICIO 8 (R)

Sean A y B dos naturales de *n* dígitos:

$$A = a_n a_{n-1} \dots a_1 \qquad B = b_n b_{n-1} \dots b_1$$

Sea C el producto de A por B, el método convencional es el que calcula C de la siguiente manera

$$C = \sum_{i=1}^{n} A b_i 10^{i-1}$$

Un método no convencional consiste en dividir cada natural en dos partes de *n* / 2 dígitos:

$A = A_1 10^{n/2} + A_2$ $\qquad$ $B = B_1 10^{n/2} + B_2$ donde, por ejemplo, $A_1 = a_n a_{n-1} \dots a_{(n/2)+1}$

y calcular el producto como:

$$C = (A_1 10^{n/2} + A_2)(B_1 10^{n/2} + B_2) = A_1 B_1 10^n + (A_1 B_2 + A_2 B_1) 10^{n/2} + A_2 B_2$$

a) Calcular la cantidad de multiplicaciones entre dos dígitos que realiza el método convencional.
b) Escribir un algoritmo basado en *D&C* que calcule el producto de naturales siguiendo el método no convencional.
c) Calcular la cantidad de multiplicaciones entre dos dígitos que realiza su algoritmo.

EJERCICIO 9 (R)

Considere dos arreglos de enteros de tamaño *n* ordenados en forma creciente y sin elementos repetidos entre los 2.*n* elementos.

a) Escribir un algoritmo, utilizando D&C para encontrar la mediana del los 2.*n* elementos.
b) Calcule el orden de ejecución de su algoritmo para el peor caso.

Notas:

- La mediana es el elemento que ocuparía el lugar *n* si los 2.*n* elementos estuviesen ordenados en la misma secuencia.
- No se pueden utilizar estructuras de datos auxiliares (únicamente se podrán utilizar variables enteras y de índices de arreglo).

EJERCICIO 10 (C)

Dado el ejercicio 19 del práctico 1 (función de Fibonacci), determinar el tiempo de ejecución del algoritmo.