Práctica: Producto de Polinomios - Divide y Vencerás

Diseño y Análisis de Algoritmos

1. Objetivo

Se trata de realizar un estudio comparativo de algoritmos que realizan el producto de polinomios. El estudio deberá realizarse para problemas de diferente tamaño de forma que quede reflejado en ellos el comportamiento de cada algoritmo. El estudio debe describir la arquitectura sobre la que se desarrolla la experimentación y presentar una tabla y una gráfica en los que se muestren los resultados obtenidos.

2. Especificaciones

Se generarán aleatoriamente los polinomios sobre los que se realiza el estudio. El estudio comparativo debe realizarse con con los siguientes algoritmos:

1. **Algoritmo 1 (Algoritmo clásico)**: Dados dos polinomios $p(x)$ y $q(x)$ de grados $N-1$ cada uno de ellos (N coeficientes p_i, q_j), calcular el producto $r(x)$ de acuerdo a la siguiente ecuación:

$$\sum_{i=0}^{2n-2}[\sum_{j=0}^{i} p_j q_{i-j}]x^i$$

2. **Algoritmo 2 (Algoritmo Divide y Vencerás)**: Dados dos polinomios $p(x)$ y $q(x)$ de grados $N-1$ cada uno de ellos (N coeficientesp_i, q_j), se propone implementar un procedimiento que utilice la técnica *Divide y Vencerás* dividiendo cada polinomio en dos polinomios con $\frac{N}{2}$ coeficientes cada uno.
   Por ejemplo, para el polinomio $p(x) = p_0+p_1x+\cdots+p_{N-1}x^{N-1}$,Dados dos polinomios $p(x)$ y $q(x)$ de grados $N-1$ cada uno de ellos (N coeficientesp_i, q_j), definimos:
   $p_l(x) = p_0 + p_1x + \cdots + p_{\frac{N}{2}-1}x^{\frac{N}{2}-1}$
   $p(x) = p_{\frac{N}{2}} + p_{\frac{N}{2}+1}x + \cdots + p_{N-1}x^{\frac{N}{2}-1}$

   dividiendo $q(x)$ de la misma forma, tenemos:
   $p(x) = p_l(x) + x^{\frac{N}{2}}p_h(x)$
   $q(x) = q_l(x) + x^{\frac{N}{2}}q_h(x)$

   Ahora, en términos de polinomios más pequeños, el producto viene dado por:
   $p(x)q(x) = p_l(x)q_l(x) + (p_l(x)q_h(x) + q_l(x)p_h(x))x^{\frac{N}{2}} + p_h(x)q_h(x)x^N$.

En este caso habríamos pasado a tener que computar cuatro productos para obtener el producto de los polinomios de partida. Obsérvese que para realizar el producto anterior son necesarios sólo tres productos de polinomios manipulando la expresión anterior del siguiente modo:
$r_l(x) = p_l(x)q_l(x)$, $r_h(x) = p_h(x)q_h(x)$ y $r_m(x) = (p_l(x) + p_h(x))(q_l(x) + q_h(x))$ puede obtenerse el producto $p(x)q(x)$ de la forma:

$$p(x)q(x) = r_l(x) + (r_m(x) - r_l(x) - r_h(x))x^{\frac{N}{2}} + r_h(x)x^N$$

Esta aproximación divide y vencerás computa una multiplicación de polinomios de tamaño N resolviendo tres subproblemas de tamaño $\frac{N}{2}$, realizando además algunas sumas de polinomios para obtener la combinación de los productos.

3. Implementación

Considérese la siguiente jerarquía de clases para elaborar el producto de polinomios. Debe añadirse a la clase *Monomio* los operadores que se consideren necesarios. Tomando como base el patrón de diseño *Estrategia*, modifica y extiende la clase *Polinomio* de manera que a partir de ella puedan derivarse comportamientos distintos que se correspondan con el Algoritmo 1 y 2 respectivamente. Instancia objetos de estas clases a la hora de realizar el estudio comparativo.

```
1  class Monomio {
2      int Coeficiente;
3      int Exponente;
4  public:
5      Monomio(); // Constructor por defecto
6      Monomio(int coef, int exp); // Constructor con Coeficiente y Exponente
7      int Evaluar(int x) const; // Evalua un monomio en un punto
8      int getCoeficiente() const; // Devuelve el Coeficiente
9      int getExponente() const; // Devuelve el Exponente
10     void setCoeficiente(int c); // Asigna un valor al Coeficiente
11     void setExponente(int e); // Asigna un valor al Exponente
12     ....
13 };
14
15 //Operadores de insercion y extraccion
16 ostream& operator<<(ostream &sout, const Monomio &s);
17 istream& operator>>(istream &sin, Monomio &r);
18
19
20 // Suma dos monomios de mismo exponente
21 Monomio operator+(const Monomio &x, const Monomio &y);
22
23 class Polinomio {
24     int Grado; // Grado del polinomio
25     int Terminos; // Numero de terminos en el polinomio
26 public:
27     // Constructores
28     Polinomio(); // constructor por defecto
29     Polinomio(int coef[], const int tam); // constructor con vector de coeficientes
30     ...
31 };
```