Depto. De Matemática y
Ciencia de la Computación

Trabajo 1.

Nombres:

Javier Gomez

Pedro Figueroa

Ignacio Sanhueza

Asignatura: Programación Avanzada

Depto. De Matemática y
Ciencia de la Computación

Depto. De Matemática y
Ciencia de la Computación

Objetivo principal.

Implementar algoritmos básicos de aritmética de polinomios: sumas, restas
y multiplicación de dos polinomios.

Objetivos segundarios.

- Implementar algoritmos clásicos de sumas, restas y multiplicación de polinomios de forma
  estable.
- Analizar y aplicar un algoritmo del tipo dividir-y-conquistar, especificamente el algoritmo
  de Karatsuba para la multiplicación de polinomios.
- Obtener una implementación eficaz de la multiplicación de polinomios aplicable para poli-
  nomios de cualquier grados, optimizada para la multiplicación de dos polinomios del mismo
  grado.

Depto. De Matemática y
Ciencia de la Computación

Para la programación.

Se utilizará el lenguaje C, con las siguientes librerías estándares.

- stdio.h

- stdlib.h

- math.h

- time.h

Además, para evitar cualquier problema de complejidad, utilizamos la función clock() para la medición de tiempo.

Para las variables a utiizar.

La estructura del polinomio es una Lista enlazada, el cual contiene la dirección de memoria del puntero, el coeficiente del polinomio de tipo “long”, donde no hay problemas de redondeo.

Se requieren crea una función para:

1) Recibir un polinomio ingresado manualemente (opcional).

2) Generación de polinomios aleatorios (de grado dado).

3) Escribir/imprimir un polinomio.

4) Crear una copia de un polinomio.

5) Sumar dos polinomios.

6) Diferencia entre dos polinomios.

7) Multiplicar polinomios de forma clásica, guardándolos en una tabla completa de productos de coeficientes.

8) Multiplicar polinomios con la formula clásico, reduciendo la utilización de memoria via una estrategia “reducir y conquistar” en una de las entradas.

9) Comparar los resultados de dos técnicas de multiplicación (comprobando que la diferencia de los resultados es 0).

10) Seleccionar/copirar parte de un polinomio (desde un grado mínimo hasta un grado máximo).

11) grado mínimo hasta un grado máximo). Multiplicar un polinomio por una potencia de x, o sumar dos polinomios multilplicando una de las entradas por una potencia de x [se puede programar solamente una de las dosfunciones o programar ambas].

12) Multiplicar polinomios con 1 paso de inducción para el algoritmo dividir-y-conquistar clásico (con 4 multiplicaciones de tamaño d/2).

13) Multiplicar polinomios con 1 paso de multiplicación de Karatsuba (dividir-y-conquistar).

14) Multiplicación de polinomios de cualquier grado, optimizada para la multiplicación de dos polinomios del mismo grado.

Depto. De Matemática y
Ciencia de la Computación

Estructura de dato Utilizada.

La estructura utilizada fue una lista enlazada, que contenía las variables del tipo long, coef (coeficiente) y grd (grado), conectados mediante su puntero nodeP *next.

```
typedef struct nodeP {
  /*
    Estructura linked list que permite almacenar coeficiente y grado
    de un polinomio de la forma a(x)=(asub(n-1)+A(x))
  */
  long coef;          // Coeficiente
  long grd;           // Grado
  struct nodeP *next; // Puntero de tipo nodeP que apunta al siguiente.
} node;
```

Decidimos trabajar con lista, por la cantidad de datos utlizada (polinomios de grado 10.000 en adelante), donde nos permite trabajar variando los largos de estas.

- El tamaño en memoria que admite long, para 64 bits, es de 8 bytes.
- El tamaño en memoria que utiliza un puntero, para 64 bits es de 8 bytes.

Formato utilizado en los polinomios.

El formato utilizado para los polinomios en el programa, es una pila donde el último es el más pequeño de los polinomios.

Depto. De Matemática y
Ciencia de la Computación

Estrategias utilizadas en las funciones 7), 8), 12), 13) y 14) (mutiplicación de polinomios sin contruír una tabla de todos los productos de coeficientes):

Multiplicar polinomios con la formula clásica, guardando la tabla completa de productos de coeficientes.

Depto. De Matemática y
Ciencia de la Computación

Multiplicar polinomios con la formula clásica, reduciendo la utilización de memoria via unaestrategia reducir-y-conquistar en una de las entradas.

Para este caso, se creó una función llamada ***polCoefC**, donde le enviamos el largo del polinomio (el grado), donde crea un polinomio con coeficientes 0 (polinomio referencia), pero con los grados que van hasta el grado más grande de este, y se retorna la pila completa.

Prosiguiendo la traza del programa para reducir y conquistar, nos encontramos con la función ***coefXPol**, el cual se el envía el coeficiente del polinomio, el grado del polinomio, su dirección de memoria y un polinomio que contiene la suma de lso grados de los polinomios que se desean multiplicar, con el fin de obtener un final dentro de los ciclo de la función un final. Refiriéndonos al ciclo de esta función, damos cuenta que se realiza mientras exista el polinomio, donde tendremos dos casos:
a) Primer caso, **si** la suma de los grados del polinomio sea igual al grado del polinomio referencia que este tenga, en una suma cíclica vamos agregandos los resultados de los polinomios con los coeficientes de estos multiplicados, donde los resultados se agregan al polinomio referencia.
b) segundo caso, **sino** avanza el polinomio referencia.
Hecho esto, se retorna el polinomio referencia con todos los coeficientes ordenados y operados (multiplicación) en la lista.

Para el caso final de este método de multiplicación de polinomios, tenemos la función ***RyC** (siglas de reducir y conquistar), donde recibe los dos polinomios que se desean multiplicar y un polinomio referencia, pero a su vez, con los grados del polinomio que debería resultar de la multiplicación de los dos polinomios ingresados a la función. Dentro d ella función se destaca un ciclo, donde **Mientras** el siguiente exista (cualquiera de los dos polinomios que se desea multiplicar), se va llenando el polinomio referencia, mediante la función ***coefXPol**, donde se le mandan los datos del primer polinomio, el nodo del otro polinomio y el polinomio referencia, hasta llegar al final de la lista. Si se llega al final, se opera también la misma función ***coefXPol** con los mismos datos, donde al final se retorna el polinomio referencia, multiplicado (con los coeficientes y grados operados entre los polinomios a multiplicar).

Algoritmo:

```
node *polCoefC(long grd) {
  int i = 0;
  node *final = NULL;
  for (i = 0; i <= grd; i++) {
    push(&final, 0, i);
  }
  return final;
}
node *coefXPol(long coef, long grd, node *p, node *r) {
  node *auxr = r;
  node *auxp = p;
  while (auxp) {
    if (auxp->grd + grd == auxr->grd) {
      auxr->coef = auxr->coef + auxp->coef * coef;
      auxp = auxp->next;
      auxr = auxr->next;
    } else {
      auxr = auxr->next;
    }
  }
  return r;
}

node *RyC(node *p1, node *p2, node *r) {
  node *aux1 = NULL;
  aux1 = p1;
  while (aux1->next) {
    r = coefXPol(aux1->coef, aux1->grd, p2, r);
    aux1 = aux1->next;
  }
  r = coefXPol(aux1->coef, aux1->grd, p2, r);
  return r;
}
```

Para hacer ingreso de la función ***RyC** se requiere P1, P2 y P3, donde P3=PolCoefC(P1->grd,P2->grd), donde P1 y P2 son polinomios ingresados o generados, y P3 es el polinomio referencia y grd, son los grados de este, ya mencionado en el tipo de dato utilizado en esta experiencia.

Depto. De Matemática y
Ciencia de la Computación

Multiplicar polinomios con 1 paso de inducción para el algoritmo dividir-y-conquistar clásico (con 4 multiplicaciones de tamaño d/2).

Para esta función se requieren diferentes funciones, unas de las primeras funciones es **splitPoly** donde recibe un polinomio y un grado de este (el grado mas grande el polinomio que está en la cabeza). Donde se generan dos arreglos de polinomios enlistados, por lo cual cada tamaño está dado por n/2 grados, con respecto a la entrada del polinomio(asumiendo que el tamaño de la entrada es n).

Dentro de esto está la función ***MultDivYConq** donde recibe dos polinomios y los largos de estos dos polinomios, donde se crea un polinomio referencia (polinomio con coefeicientes 0), y mediante la multiplicación **reducir y conquistar** se actualiza resultado con cada polinomio de largo n/2 con respecto al polinomio orginal. En este caso se hacen 4 multiplicaciones y se actualiza dentro del polinomio referencia.

Algortimo:

```
void splitPoly(node *head, node *split[2], long rango) {
  node *cdr = NULL;
  node *aux1 = NULL;
  node *aux2 = NULL;
  long corte;
  cdr = head;
  corte = rango;
  while (corte) { // Se avanza n/2 posiciones
    if (!aux1) {
      push(&aux1, cdr->coef, cdr->grd);
      split[0] = aux1;
    } else {
      push(&aux1->next, cdr->coef, cdr->grd);
      aux1 = aux1->next;
    }
    cdr = cdr->next;
    corte--;
  }
  while (cdr) { // Se avanza n/2 posiciones
    if (!aux2) {
      push(&aux2, cdr->coef, cdr->grd);
      split[1] = aux2;
    } else {
      push(&aux2->next, cdr->coef, cdr->grd);
      aux2 = aux2->next;
    }
    cdr = cdr->next;
    corte--;
  }
  return;
}
```

Depto. De Matemática y
Ciencia de la Computación

```
node *MultDivYConq(node *p1, node *p2, long n1, long n2) {
  node *cdr1[2];
  node *cdr2[2];
  node *aux1 = NULL;
  node *aux2 = NULL;
  node *aux3 = NULL;
  node *result = polCoefC(p1->grd + p2->grd);
  long mayor = 0;
  if (p1->next &&
      p2->next) { // Si existen ambas continuaciones se sigue con la recursiór
    splitPoly(p1, cdr1, n1 / 2);
    splitPoly(p2, cdr2, n2 / 2);
    result = RyC(cdr1[0], cdr2[0], result);
    result = RyC(cdr1[0], cdr2[1], result);
    result = RyC(cdr1[1], cdr2[0], result);
    result = RyC(cdr1[1], cdr2[1], result);

    return result;
  } else {
    result = RyC(p1, p2, result);
  }
}
```

Multiplicar polinomios con 1 paso de multiplicación de Karatsuba (dividir-y-conquistar) y Multiplicación de polinomios de cualquier grado, optimizada para la multiplicación de dos polinomios del mismo grado.

Para este caso usamos dos funciones, con el fin de que los dos polinomios ingresados tengas la misma cantidad de grados. Vale decir que para cortar algún tipo de polinomio que no tenga grado potencia de dos, poder llenar con 0 los valores, para poder operar el polinomio mediante la técnica **Karatsuba** en ellos, o en el peor de los casos **Reducir y Conquistar.**

Dentro de este algoritmo se destacan dos funciones: **cutin**, que recibe un polinomio y lo divide en dos términos tales que la primera parte tenga n términos.

Se utiliza la función **splitPolyK** dond eel polinomio es dividido en dos partes iguales

Posteriormente tenemos la función **Karatsuba**, donde se realizan algunos ajustes, con respecto al método de dividir y conquiistar, reduciendo el número de operaciones elementales de productos, algorítmicamente se denota de la siguiente forma:

$$
\begin{aligned}
a(x) \cdot b(x) = {} & (a_1(x) \cdot b_1(x))x^{2^k} \\
& + (a_0(x) \cdot b_0(x)) \\
& + \Big((a_0(x) + a_1(x)) \cdot (b_0(x) + b_1(x)) - a_1(x)b_1(x) - a_0(x)b_0(x) \Big) x^{2^{k-1}}
\end{aligned}
$$

Donde obtenemos 3 multiplicaciones con la mitad de los valores de cada polinomio y algunas sumas, lo que en su algortimo lo hace más eficiente en cantidad de operaciones elementales a utilizar.

En el algortimo ***Karatsuba**, se ingresan dos polinomios con sus largos respectivos. Dentro de este, tenemos una variable comparacón para limitar ciclos **pow2**, donde están las potencias de dos más utilizadas según capacidad de cómputo del PC (limitamos a la función ***Karatsuba** a solo realizar este tipo de múltiplicación a polinomios de grados con potencias de dos). Tenemos nuestro polinomio referencia (polinomios con el grado de las resultantes que tendría el producto entre lso dos polinomios utilizados y coeficientes 0), se declaran 4 variables auxiliares y una resultante, que contiene el polinomio referencia, donde este será retornado con los valores operados entre los polinomios.

Algoritmo:

```
node *karatsuba(node *p1, node *p2, long l1, long l2,
                int def) { // Primer paso de karatsuba
  long k = 0;
  long l = 0;
  long pow2[25] = {1,       2,       4,       8,       16,     32,     64,
                   128,     256,     512,     1024,    2048,   4096,   8192,
                   16384,   32768,   65536,   131072,  262144, 524288, 1048576,
                   2097152, 4194304, 8388608, 16777216};
  node *cdr1[2];
  node *cdr2[2];
  node *aux1 = NULL;
  node *aux2 = NULL;
  node *aux3[2];
  node *aux4[2];
  node *c1 = NULL;
  node *c2 = NULL;
  node *c3 = NULL;
  node *c4 = NULL;
  node *result = polCoefC(p1->grd + p2->grd);
  if (l1 < l2) {
    p1 = complete(p1, l2 - l1);
  } else if (l1 > l2) {
    p2 = complete(p2, l1 - l2);
  }
  l = p1->grd + 1;
  for (k = 0; k < 24; k++) {
    if (p1->grd + 1 > pow2[k] && p1->grd + 1 < pow2[k + 1]) {//Caso en el que el largo
      return RyC(p1, p2, result); //Ocupamos reducir y conquistar. uwu
    }
  }
  k = l / 2;
```


```
if (p1->next && p2->next &&
    def != 0) {           // Caso para el paso inductivo de Karatsuba
  splitPolyK(p1, cdr1, k); // A(x)1*X^(2^(k-1)) && A(x)0
  splitPolyK(p2, cdr2, k); // B(x)1*X^(2^(k-1)) && B(x)0
  c1 = polCoefC(k);
  c1 = karatsuba0(cdr1[0], cdr2[0], k, c1); //(A(x)1 * B(x)1)
  c2 = sumarPolinomios(c2, cdr1[0], 1);
  c2 = sumarPolinomios(c2, cdr1[1], 1);
  c3 = sumarPolinomios(c3, cdr2[0], 1);
  c3 = sumarPolinomios(c3, cdr2[1], 1);
  c4 = polCoefC(k);
  c4 = karatsuba0(c2, c3, k, c4); //((A(x)1 + A(x)0)*(B(x)1 + B(x)0))
  result = karatsuba0(cdr1[1], cdr2[1], k, result); //(A(x)0 * B(x)0)
  c4 = sumarPolinomios(c4, c1, -1);
  c4 = sumarPolinomios(c4, result, -1); //((A(x)1 + A(x)0)*(B(x)1 + B(x)0)) -
                                        //(A(x)1 * B(x)1) - (A(x)0 * B(x)0)
  aux1 = polCoefC(l + c1->grd);
  aux1 = coefXPol(1, l, c1, aux1); //(A(x)1 * B(x)1)^(2^k)
  aux2 = polCoefC(k + c4->grd);
  aux2 = coefXPol(1, k, c4, aux2); // c4^(2^(k-1))
  result = sumarPolinomios(result, aux1, 1);
  result = sumarPolinomios(result, aux2, 1);
  c1 = eliminar(c1);
  c2 = eliminar(c2);
  c3 = eliminar(c3);
  c4 = eliminar(c4);
  aux1 = eliminar(aux1);
  aux2 = eliminar(aux2);
  cdr1[0] = eliminar(cdr1[0]);
  cdr1[1] = eliminar(cdr1[1]);
  cdr2[0] = eliminar(cdr2[0]);
  cdr2[1] = eliminar(cdr2[1]);
  return elimSobrantes(result);

 } else { // Caso solo para primer paso de karatsuba
  if (p1->next && p2->next) {
    splitPolyK(p1, cdr1, k); // A(x)1*X^(2^(k-1)) && A(x)0
    splitPolyK(p2, cdr2, k); // B(x)1*X^(2^(k-1)) && B(x)0
    c1 = polCoefC(cdr1[0]->grd + cdr2[0]->grd);
    c1 = RyC(cdr1[0], cdr2[0], c1); //(A(x)1 * B(x)1)
    c2 = sumarPolinomios(c2, cdr1[0], 1);
    c2 = sumarPolinomios(c2, cdr1[1], 1);
    c3 = sumarPolinomios(c3, cdr2[0], 1);
    c3 = sumarPolinomios(c3, cdr2[1], 1);
    c4 = polCoefC(c2->grd + c3->grd);
    c4 = RyC(c2, c3, c4); //((A(x)1 + A(x)0)*(B(x)1 + B(x)0))
    result = RyC(cdr1[1], cdr2[1], result); //(A(x)0 * B(x)0)
    c4 = sumarPolinomios(c4, c1, -1);
    c4 =
        sumarPolinomios(c4, result, -1); //((A(x)1 + A(x)0)*(B(x)1 + B(x)0)) -
                                         //(A(x)1 * B(x)1) - (A(x)0 * B(x)0)
    aux1 = polCoefC(l + c1->grd);
    aux1 = coefXPol(1, l, c1, aux1); //(A(x)1 * B(x)1)^(2^k)
    aux2 = polCoefC(k + c4->grd);
    aux2 = coefXPol(1, k, c4, aux2); // c4^(2^(k-1))
    result = sumarPolinomios(result, aux1, 1);
    result = sumarPolinomios(result, aux2, 1);
    c1 = eliminar(c1);
    c2 = eliminar(c2);
    c3 = eliminar(c3);
    c4 = eliminar(c4);
    aux1 = eliminar(aux1);
    aux2 = eliminar(aux2);
    cdr1[0] = eliminar(cdr1[0]);
    cdr1[1] = eliminar(cdr1[1]);


   cdr2[0] = eliminar(cdr2[0]);
   cdr2[1] = eliminar(cdr2[1]);
   return elimSobrantes(result);
  }
  return elimSobrantes(RyC(p1, p2, result));
 }
}
```

Depto. De Matemática y
Ciencia de la Computación

Complejidad Algortimica.

Para la complejidad de algoritmo, damos cuenta que utilizaremos notación asintótica, con el fin de analizar la complejidad de este.

La complejidad de una múltiplicación de polinomio por **fuerza bruta** pertenece a:

- $O(n^2)$ operaciones.
- $O(n^2)$ memoria.

Del método **reducir y conquistar**, asociamos un problema de tamaño n a n-m=r, donde se repite iterativamente, hasta lograr m que sea cerca de 0, entonces asociamos un polinomio de grado n-1, a un molinomio de grado n-2, de esta manera reducimos el tamaño de la entrada.

El costo de multiplicar dos polinomios de grado n-1, se obtiene la recurrencia:

$$c_n = c_{n-1} + \text{ajustes}$$

Los ajustes vienen dado por escalar el polinomio vienene dado por 2n-1 multiplicaciones y 2n-3 sumas, para la primera multiplicación de coeficientes, existen

$$n^2 \text{ multiplicaciones y } (n-1)^2 \text{ sumas de coeficientes}$$

Si se reduce la primera entrada, el costo de multiplicar un polinomio de grado n-1, con un polinomio con m-1 coeficientes. Donde se sie analiza la recurrencia tenemos en cuenta que se tendrán m multiplicaciones y m-1 sumas de coeficientes. Entonces el coste de multplicar polinomios mediante esta técnica es de m*n y (n-1)(m-!) sumas de coeficientes. Si el grado de polinomio con m-1 coeficientes es n, entonces tendrémos:

$$n^2 \text{ multiplicaciones y } (n-1)^2 \text{ sumas de coeficientes}$$

Depto. De Matemática y
Ciencia de la Computación

Del método **Dividir y Conquistar**, se desea reducir el tamao n a n/m = r pequeño, donde se repite recursivamente hasta m que sea cerca de 1. Asociamos un producto de polinomio de grado n=2^(k) – 1, al producto de polinomio de grado 2^(k-1)-1. El costo de la recurrencia de dos polinomios del grado 2^(k) – 1, la recurrencia viene dada por:

$$c_k = 4c_{k-1} + \text{ajustes}$$

los ajustes vienen dado por:

$$\text{Ajustes: } (2^k - 1) + (2^k - 2) = 2^{k+1} - 3 \text{ sumas de coeficientes}$$

Si existe una multiplicación, obtendremos:

$$\text{Cuando } n = 2^k, \text{ da } n^2 \text{ multiplicaciones y } (n-1)^2 \text{ sumas de coeficientes}$$

El método **Karatsuba** tenemos una recurrencia:

$$c_k = 3c_{k-1} + \text{ajustes}$$

Aplicando los ajustes tendremos que:

$$\text{Ajustes: } 2(2^{k-1}) + 2(2^k - 1) + (2^k - 2) = 2^{k+2} - 4 \text{ sumas de coeficientes}$$

Para el caso de una multiplicación:

$$\text{Solución: } 3^k \text{ multiplicaciones y } 6 \cdot 3^k - 8 \cdot 2^k + 2 \text{ sumas de coeficientes}$$
(verificar por inducción)

$$\text{Cuando } n = 2^k, \text{ da } n^{1,58496\cdots} \text{ multiplicaciones y } 6n^{1,58496\cdots} - 8n + 2 \text{ sumas de coeficientes}$$

$$\text{Complejidad: } O(n^{\log_2 3})$$

Para el caso del programa, tomaremos solamente el costo de la multiplicación.

Depto. De Matemática y
Ciencia de la Computación

Complejidad Algorítmica de la Multiplicación de Polinomio con Diferentes Métodos.

Para la compilación del algoritmo, se utilizó un computador con las siguientes características:

- Marca: Aspire v13 Acer.

- CPU: Intel Core i5-4210u 1,7 Ghz – turbo boost to 2.7 Ghz.

- RAM: 6 Gb, en dos tarjetas, una de 4 de 2.

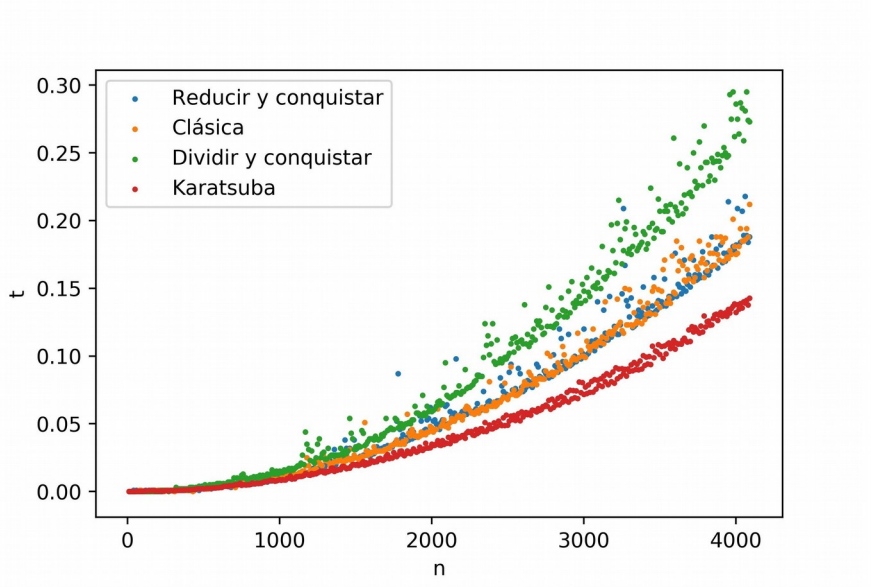
Para las gáficas y los análisis utilizaremos:

1) Spyder: IDE de iPython, que contienen las librerías:

- Numpy: para el trabajo con las tablas.
- Pandas: para la lectura de las tablas en formato csv.
- Matplotlib: para generar las imágenes de gráficos.

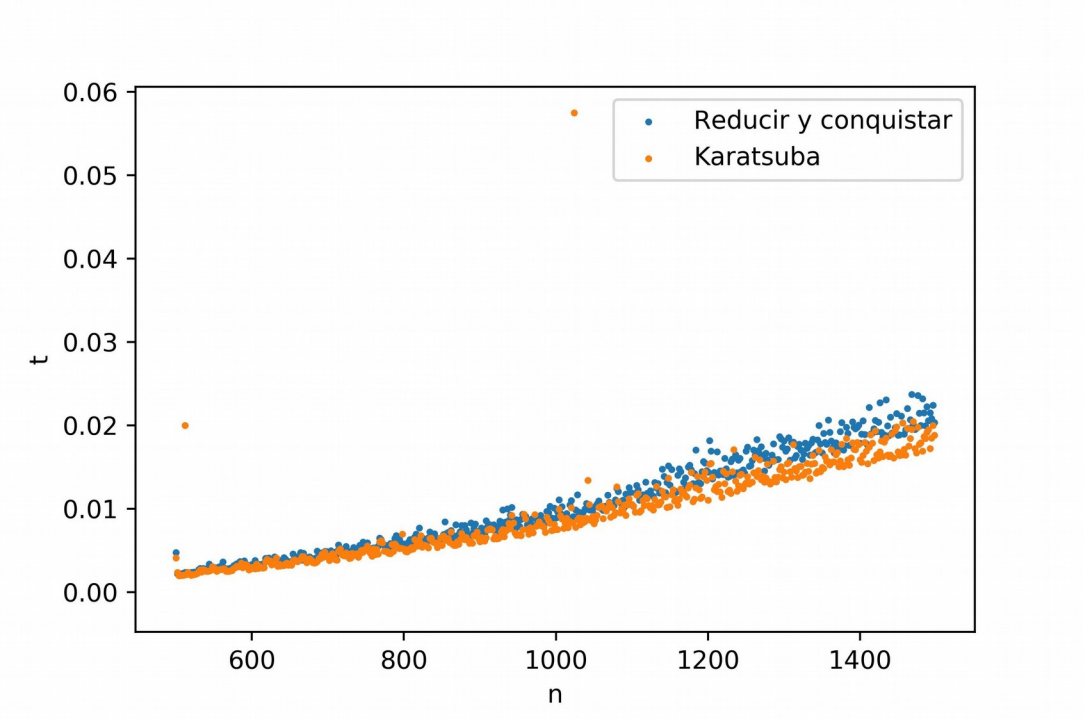
2) Para el análisis de datos utilizamos GeoGebra y Libre Office (Ubuntu).

Gráfica de los 4 métodos:



Depto. De Matemática y
Ciencia de la Computación

Gráfica de Múltiplicación Directa (Clásica Fuerza Bruta) y Karasuba (Dividir y Conquistar), para datos bajos del orden de 2^{12}.



Se toman Ciertos valores a evaluar para los casos de Fuerza Bruta y Dividir y Conquistar, con el fin de obtener la relación más cercana al dato teórico, para entradas n valores, y visto por su grado 2^(k) (números bajos).

Para ello se utilizó la plataforma geogebra y LibreOffice, para obtener la mejor definición de la función del gáfico, entendiendo este como una aproximación a los valorede conteo del proceso mientras se ejecuta el algoritmo, con el fin de comprender este análisis mediante las técnicas de conteo de procesos.

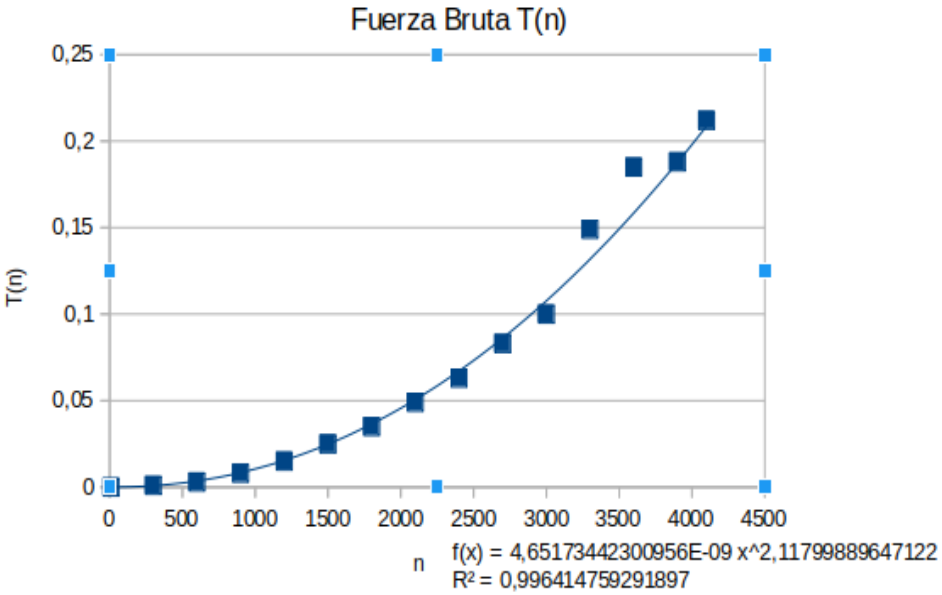
Depto. De Matemática y
Ciencia de la Computación

Fuerza Bruta:

Tabla de Datos para Fuerza Bruta:

n	T(n)
10	0
300	0,001
600	0,003
900	0,008
1200	0,015
1500	0,025
1800	0,035
2100	0,049
2400	0,063
2700	0,083
3000	0,1
3300	0,149
3600	0,185
3900	0,188
4100	0,212

Gáráfica:



Depto. De Matemática y
Ciencia de la Computación

Conclusiones de Fuerza Bruta:

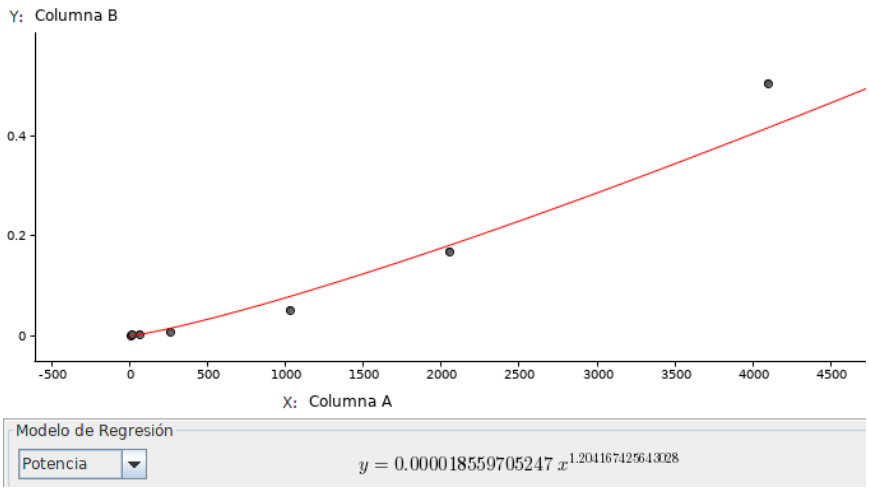
Donde Damos cuenta que la función Coincide en Puntos con n^2, lo que donde están la mayoría de los puntos puentos del conteo, se Comprueba con el valor de relación de las variables estadísticas que se muestra en el gráfico.

Karatsuba:

Tabla de Datos de Karastsuba (Entrada de datos potencias de Dos):

n	T(n)
4	2,3E-05
16	0,000227
64	0,003758
256	0,008594
1024	0,051957
2048	0,1686663
4096	0,505226

Y Obtuvimos una gáfica mediante la plataforma Geogebra:



Conclusión:

De la envolvente de la línea de tendencia obtuvimos el valor de la potencia de 1,2041674..., comparado con el valor teórico 1,58496, son bastante lógico en comparación a valor de conteo del proceso de multiplicación de los polinomios, puesto que en el mismo algoritmo que se fue construyendo este valor, se fue dividiendo el problema con respecto al grado de este, y con una ventaja al ser potencias de dos las que se dividiían.