# Práctica 12: Algoritmos Divide y Vencerás (Divide & Conquer). Multiplicación de enteros grandes

## Objetivo

El objetivo principal de esta práctica consiste en desarrollar un ejemplo de Algoritmos Divide y Vencerás (*Divide & Conquer*) para multiplicar enteros positivos grandes. Para ello se detalla a continuación una introducción al problema, los requisitos funcionales mínimos y opcionales, ejemplos de uso y los criterios de evaluación.

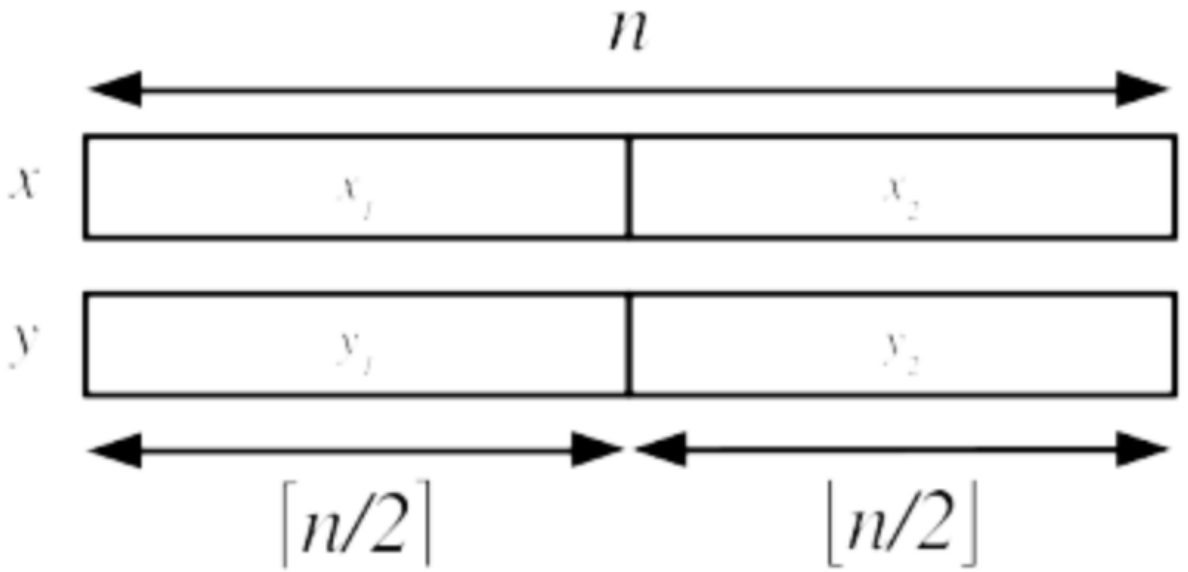
## Introducción

La multiplicación de dos números, dependiendo de la longitud de los números, puede utilizar distintos algoritmos, unos más eficientes que otros. Los algoritmos de multiplicación para realizar manualmente tiene usualmente una complejidad de $O(n^2)$, pero en 1960 Anatoly Karatsuba descubrió que se podía obtener una mejor complejidad.

## Algoritmo de Karatsuba

Para calcular el producto de dos números grandes *x* e *y* el algoritmo de Karatsuba aplica una fórmula que nos permite usar solo tres multiplicaciones de números más pequeños, cada uno con más o menos la mitad de los dígitos de *x* e *y*, más algunas sumas y desplazamientos de dígitos.

Para aplicar una estrategia Divide y Vencerás los números a multiplicar *x* e *y* podemos suponer que están representados como cadenas de *n* dígitos en alguna base *B*, y se pueden puede dividir los dos números dados en partes casi iguales de la manera siguiente:



y el producto de *x* e *y* se puede realizar como:

$$xy = z_2 B^{2\lfloor n/2 \rfloor} + z_1 B^{\lfloor n/2 \rfloor} + z_0,$$

donde

$$z_2 = x_1 y_1,$$
$$z_0 = x_0 y_0,$$
$$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0.$$

Nótese que ahora son necesarias tres multiplicaciones de números más pequeños que los originales. Cada una de estas multiplicaciones se han de calcular llamando recursivamente al algoritmo de Karatsuba, lográndose un orden de complejidad de $O(n^{\log(3)/\log(2)})$. La recursión finalizará cuando los números a operar sean los suficientemente pequeños para poder ser multiplicados directamente.

## Clase BigInt

Para facilitar la realización de esta práctica habrá de utilizarse la clase BigInt (BIG INTEGERS) desarrollada en C++ y publicada en GeeksforGeeks. Dicha clase define un nuevo tipo de datos `BigInt` que almacenar enteros positivos grandes haciendo uso de cadenas std::string en C++. Además en ella se implementa múltiples operaciones básicas sobre este nuevo tipo de datos, incluyendo suma, resta y multiplicación, aunque la multiplicación implementada es la clásica con orden de complejidad $O(n^2)$. También ha de tenerse en cuenta lo siguiente:

- La clase BigInt almacena los números en base 10 ($B$ = 10), donde cada dígito (0-9) está almacenado en un carácter. Además, por motivos de eficiencia, los dígitos están ordenados en la cadena desde el menos significativo al más significativo
- Para dividir un número BigInt en dos partes casi iguales, así como para multiplicarlo por números potencias de base 10, resulta mas eficiente utilizar los modificadores de la clase std::string directamente
- El algoritmo de Karatsuba, una vez que los números a multiplicar son suficientemente pequeños, por ejemplo un dígito, podría utilizar el `operator*` de la propia clase BigInt
- El `operator>>` para leer un número BigInt no funciona correctamente, si se desea utilizar habrá de corregirse inspirándose en el constructor que tiene como parámetro un `string`

### Requisitos funcionales mínimos

Para superar la práctica, los requisitos funcionales mínimos exigidos son:

- Implementar el algoritmo Divide y Vencerás de Karatsuba mediante recursividad para multiplicar dos enteros positivos grandes, no necesariamente de igual número de dígitos, representados por la clase BigInt

- Obtener el resultado de multiplicar dos enteros positivos grandes utilizando el operador `*` sobrecargado en la clase BigInt
- La clase BigInt debe ser ajustada lo necesario para adaptarse al estilo exigido en las prácticas, Google C++ Style Guide
- El programa realizado ha de ejecutarse leyendo desde la entrada estandar los dos números a multiplicar, cada uno en una línea diferente, y emitiendo su resultado a la salida estandar
- Si no se indica nada o se indica la opción `-k` en la línea de comandos el programa utilizará el algoritmo de Karatsuba para realizar la multiplicación. Si se indica la opción `-m` el programa utilizará el utilizando el operador `*` de la clase BigInt
- El programa, después de emitir el resultado, debe emitir por pantalla el tiempo que ha necesitado en la ejecución del algoritmo seleccionado

## Requisitos funcionales opcionales

Dentro de la evaluación global de la práctica, se valorarán positivamente hasta tres de los siguientes requisitos optativos:

- Implementación híbrida del algoritmo de Karatsuba, de forma que cuando el número de dígitos de los valores *x* e *y* a multiplicar sean menor o igual que un valor cota determinado utilice el operador `*` de la clase BigInt, supuestamente mas eficiente para números pequeños. Llamando al programa con la opción `-n <cota>` en la línea de comandos, donde `<cota>` es el valor de cota del cambio
- Generación aleatoria de los números *x* e *y* enteros positivos a multiplicar. Llamando al programa con la opción `-r <dígitos>` en la línea de comandos, donde `<dígitos>` es el el número de dígitos que tiene que tener cada número
- Realización de un documento con una tabla de tiempos de cálculo empleados en la ejecución utilizando números con distintas cantidades de dígitos y comparando con distintos algoritmos o diferentes cotas de la implementación híbrida del algoritmo de Karatsuba
- Implementación de algún otro algoritmo de multiplicación de números enteros grandes
- Modificación de la clase BigInt para realizar un empaquetado más eficiente de los números enteros, y cambio por tanto de la base *B*

## Ejemplos de prueba

Se dispone de una carpeta Ejemplos de prueba con ficheros que contienen los números y resultado para dos multiplicaciones distintas (*n1* x *n2* y *n3* x *n4*), si su programa se llama `multbigint` y cumple las especificaciones de entrada y salida comentadas puede probarlo de la siguiente manera:

```
cat n1.dat n2.dat | ./multbigint | diff - n1_x_n2.dat
cat n3.dat n4.dat | ./multbigint | diff - n3_x_n4.dat
```

También puede utilizar la siguiente web para poder generar números enteros aleatorios de diferentes dimensiones, el resultado de la multiplicación de dos de ellos con las opciones `-m` y `-k` debe ser el mismo.

## Evaluación

Se evaluará positivamente los siguientes aspectos:

- Funcionamiento correcto de la modificación solicitada: si no se consigue realizar la modificación la práctica será valorada con un cero.
- Presentación en el laboratorio: el grado de funcionamiento de la práctica, y si desarrolla requisitos opcionales o mejoras significativas.
- Código subido en la tarea correspondiente a la práctica: buen diseño, estructura de clases y limpieza.

Teniendo superados los aspectos anteriormente descritos, una evaluación orientativa sería la siguiente:

- Si se desarrollan los requisitos funcionales mínimos: hasta 5 puntos.
- Si se desarrolla además un requisito funcional optativo u otra mejora significativa: hasta 7 puntos.
- Si se desarrolla además dos requisitos funcionales optativos: hasta 8,5 puntos.
- Si se desarrolla además tres o más requisitos funcionales optativos: hasta 10 puntos.

# Profesor responsable

Si tienen alguna duda más sobre la práctica, el enunciado o la corrección, pueden hacerla al correo electrónico del profesor Patricio García Báez.