**Cálculo Simbólico**

**Documentación práctica 1**

## Aritmética entera

## Resultados de la comparación de los algoritmos.

### 1 numero \* 1 numero -🡪 desde 1 dig hasta 1024 dig.

|  |  |  |  |
| --- | --- | --- | --- |
| Numero Digitos | Escuela en ms | Karatsuba en ms | Modular en ms |
| 1 | 0.00124 | 0.00865 | 0.1 |
| 10 | 0.1 | 0.2 | 0.4 |
| 32 | 0.1 | 0.7 | 1.8 |
| 64 | 0.2 | 1.1 | 4.6999 |
| 128 | 0.5 | 2.7 | 29.3 |
| 256 | 1.7 | 3.1 | 194.5 |
| 512 | 26.9 | 30.6 | 11520.5 |
| 1024 | 28.4 | 46.7 | 11403.5 |
| 2048 |  |  |  |
| 4096 |  |  |  |

### 1 numero grande \* (1 numero que siempre sea un dig) -🡪 el grande desde 1 dig hasta 1024 dig.

|  |  |  |  |
| --- | --- | --- | --- |
| Numero Digitos del numero grande | Escuela en ms | Karatsuba en ms | Modular en ms |
| 1 | 0.000045 | 0.000324 | 0.00045 |
| 10 | 0.000048 | 0.3 | 0.5 |
| 32 | 0.0346 | 0.6 | 1.5 |
| 64 | 0.056 | 1.1 | 5.3 |
| 128 | 0.1 | 2.8 | 26.2 |
| 256 | 0.1 | 3.2 | 191.3 |
| 512 | 0.1 | 27.5 | 11500.3 |
| 1024 | 0.5 | 55.3 | 11434.7 |
| 2048 |  |  |  |
| 4096 |  |  |  |

### 1 numero \* 1 numero -🡪 desde 100 dig hasta 2000 dig de 100 en 100.

|  |  |  |  |
| --- | --- | --- | --- |
| Numero Digitos | Escuela en ms | Karatsuba en ms | Modular en ms |
| 100 | 0.9 | 4.2 | 30.6 |
| 200 | 1.0 | 3.4 | 182.7 |
| 300 | 2.2 | 9.4 | 1443.0 |
| 400 | 3.1 | 10.9 | 1407.1 |
| 500 | 6.3 | 9.3 | 1388.4 |
| 600 | 7.8 | 28.0 | 10977.7 |
| 700 | 12.5 | 28.1 | 11023.0 |
| 800 | 15.6 | 28.1 | 10968.4 |
| 900 | 18.8 | 28.0 | 10974.7 |
| 1000 | 23.4 | 28.1 | 10960.6 |
| 1100 | 29.6 | 84.3 | 87299.3 |
| 1200 | 35.9 | 82.6 | 87268.1 |
| 1300 | 42.1 | 84.3 | 522841.3 |
| 1400 | 48.3 | 84.3 | 87392.9 |
| 1500 | 56.2 | 84.2 | 446023.8 |
| 1600 | 62.4 | 85.8 | 87349.3 |
| 1700 | 71.7 | 84.3 | 87780.5 |
| 1800 | 79.5 | 85.8 | 87449.1 |
| 1900 | 88.9 | 85.8 | 87595.8 |
| 2000 | 99.8 | 84.2 | 87759.2 |

### 1 numero grande \* (1 numero que siempre sea un dig) -🡪 el grande desde desde 100 dig hasta 2000 dig de 100 en 100.

|  |  |  |  |
| --- | --- | --- | --- |
| Numero Digitos | Escuela en ms | Karatsuba en ms | Modular en ms |
| 100 | 0.1 | 4.2 | 31.3 |
| 200 | 0.1 | 3.1 | 183.5 |
| 300 | 0.1 | 9.0 | 1443.8 |
| 400 | 0.1 | 8.9 | 1447.1 |
| 500 | 0.1 | 9.0 | 1446.2 |
| 600 | 0.1 | 28.0 | 11439.2 |
| 700 | 0.1 | 27.9 | 11427.3 |
| 800 | 0.1 | 27.7 | 11416.6 |
| 900 | 0.1 | 27.8 | 11416.7 |
| 1000 | 0.1 | 27.6 | 11418.5 |
| 1100 | 0.1 | 82.4 | 90902.0 |
| 1200 | 0.1 | 83.5 | 90951.9 |
| 1300 | 0.1 | 83.8 | 90924.9 |
| 1400 | 0.2 | 83.5 | 90849.1 |
| 1500 | 0.2 | 83.4 | 90844.5 |
| 1600 | 0.2 | 83.8 | 90900.6 |
| 1700 | 0.2 | 84.0 | 90919.7 |
| 1800 | 0.2 | 84.7 | 91015.9 |
| 1900 | 0.2 | 85.3 | 91132.7 |
| 2000 | 0.2 | 85.4 | 90929.4 |

## Documentación del código usado.

**cambiaBase**

public static java.util.ArrayList<java.lang.Integer> **cambiaBase**(java.util.ArrayList<java.lang.Character> num)

Convierte el ArrayList de entrada que es un número en Hexadecimal a un número en base 16^2 de la siguiente forma:

1.Recorre el ArrayList de entrada y convierte cada 4 cifras de Hexadecimal en un dígito de nuestro numero en base 2^16, cada posición contendrá un numero 'num\_i' menor que 16^4=65536 el cual multiplica a la base elevada a la posición que ocupa, o sea: num\_0\*65536^0 + num\_1\*65536^1 + num\_2\*65536^2.....num\_list.size\*65536^list.size.

2.Devuelve un ArrayList cuyas posiciones contienen cada uno de los dígitos del número en base 2^16.

**Parametros:**

num - cadena de entrada que contiene el numero en Hexadecimal que se introduce

**Returns:**

ArrayList con el numero en base 2^16, en cada una de las posiciones hay un dígito del número.

**leer\_numero\_hexadecimal**

public static java.util.ArrayList<java.lang.Character> **leer\_numero\_hexadecimal**()

Lee de un fichero de entrada o de la entrada estándar un número en hexadecimal el cual después de transformarse a base 2^16 será usado para las operaciones correspondientes.

**mostrar\_numero**

public static void **mostrar\_numero**(java.util.ArrayList<java.lang.Integer> vector\_enteros)

Imprime el número obtenido del ArrayList que se le pasa por parámetro empezando por el final.

**Parametros:**

vector\_enteros - ArrayList el cual se va a mostrar por pantalla.

**suma\_escuela**

public static java.util.ArrayList<java.lang.Integer> **suma\_escuela**(java.util.ArrayList<java.lang.Integer> num1, java.util.ArrayList<java.lang.Integer> num2)

Suma los dos números en base 2^16 pasados por parámetro mediante el algoritmo suma\_escuela siguiendo los siguientes pasos:

1.Se comprueba el número de dígitos de ambos números y se completa con ceros a la izquierda el de menor tamaño hasta que este alcance un número de dígitos igual al otro número. Si los dos números son iguales no se hace nada. 2.Ir recorriendo los dos números desde el digito menos significativo hasta el más significativo e ir sumando ambos dígitos introduciendo en el ArrayList 'result' el digito con el resultado de hacer modulo base 2^16 de la suma, añadiendo además el acarreo, el cual se obtiene del cociente de dividir la suma de ambos dígitos entre la base 2^16.

**Parametros:**

num1 - ArrayList con el primer número en base 2^16 que se va a sumar mediante suma\_escuela.

num2 - ArrayList con el segundo número en base 2^16 que se va a sumar mediante suma\_escuela.

**Returns:**

ArrayList con el resultado de sumar los dos número en base 2^16 mediante suma\_escuela.

**mult\_escuela**

public static java.util.ArrayList<java.lang.Integer> **mult\_escuela**(java.util.ArrayList<java.lang.Integer> num1, java.util.ArrayList<java.lang.Integer> num2)

Multiplica los dos números que se pasan como parámetro por medio del algoritmo mult\_escuela siguiendo los siguientes pasos:

1.Se añaden ceros en un número de posiciones de 'result' igual al tamaño del número que se va a multiplicar.

2.Se va recorriendo el primer y segundo número multiplicando todos los factores de num1 por cada uno de los factores de num2 y almacenando en 'result' el resultado de esta multiplicación haciendo modulo con la base 2^16 y sumando también el acarreo.

3.El acarreo se calcula como el cociente del resultado de la multiplicación entre la base 2^16.

**Parametros:**

num1 - ArrayList con el primer número en base 2^16 que se va a multiplicar mediante mult\_escuela.

num2 - ArrayList con el segundo número en base 2^16 que se va a multiplicar mediante mult\_escuela.

**Returns:**

ArrayList con el resultado de multiplicar los dos número en base 2^16 mediante mult\_escuela.

**elimina\_ceros\_izq**

public static java.util.ArrayList<java.lang.Integer> **elimina\_ceros\_izq**(java.util.ArrayList<java.lang.Integer> num1)

Elimina los ceros de la izquierda que se han introducido durante las operaciones como dígitos nulos.

**Parametros:**

num1 - ArrayList el cual se va a modificar quitándole los ceros.

**Returns:**

num1 el ArrayList una vez eliminados los ceros

**resta\_escuela**

public static java.util.ArrayList<java.lang.Integer> resta\_escuela(java.util.ArrayList<java.lang.Integer> num1, java.util.ArrayList<java.lang.Integer> num2)

Resta los dos números en base 2^16 pasados por parámetro mediante el algoritmo resta\_escuela siguiendo los siguientes pasos:

1.Se comprueba el número de dígitos de ambos números y se completa con ceros a la izquierda el de menor tamaño hasta que este alcance un número de dígitos igual al otro número.

2.Ir recorriendo los dos números desde el digito menos significativo hasta el más significativo e ir restando ambos introduciendo en el ArrayList 'result' un digito con el resultado de hacer modulo base 2^16 de la resta de ambos digitos.

3.El acarreo, el cual será siempre igual a 1 se sumará al siguiente digito del sustraendo antes de realizar la resta de ambos dígitos.

**Parametros:**

num1 - ArrayList con el primer número en base 2^16 que se va a restar mediante resta\_escuela (minuendo).

num2 - ArrayList con el segundo número en base 2^16 que se va a restar mediante resta\_escuela (sustraendo).

**Returns:**

ArrayList con el resultado de restar los dos número en base 2^16 mediante resta\_escuela (diferencia).

**getUnsignedInt**

public static long **getUnsignedInt**(int x)

**calcula\_m\_karatsuba**

public static java.util.ArrayList<java.lang.Integer> **calcula\_m\_karatsuba**(java.util.ArrayList<java.lang.Integer> num1, java.util.ArrayList<java.lang.Integer> num2)

Hace una llamada a mult\_karatsuba para obtener el resultado de multiplicar los dos números en base 2^16 pasados por parámetro, además antes de esto obtiene el valor de m. Para todo ello se siguen los siguientes pasos:

1.Se calcula cuál de los dos números tiene un número mayor de dígitos.

2.Se calcula m siendo su valor el logaritmo en base 2 del número de dígitos de la cifra que se va a multiplicar que sea mayor o un número mayor a esta.

3.Completar los dos números con ceros hasta llegar al tamaño de m.

4.Hacer una llamada a mult\_karatsuba con los dos números ya completados con ceros y el valor de m obtenido.

**Parametros:**

num1 - ArrayList con el primer número en base 2^16 que se va a multiplicar mediante mult\_karatsuba pasándose como parámetro desde esta función a mult\_karatsuba en la cual se realizará la multiplicación.

num2 - ArrayList con el segundo número en base 2^16 que se va a multiplicar mediante mult\_karatsuba pasándose como parámetro desde esta función a mult\_karatsuba en la cual se realizará la multiplicación.

**Returns:**

ArrayList con el resultado de multiplicar los dos número en base 2^16 mediante mult\_karatsuba.

**mult\_karatsuba**

public static java.util.ArrayList<java.lang.Integer> **mult\_karatsuba**(java.util.ArrayList<java.lang.Integer> a, java.util.ArrayList<java.lang.Integer> b, int m)

Multiplica los dos números que se pasan como parámetro por medio del algoritmo mult\_karatsuba siguiendo los siguientes pasos:

1.El algoritmo es un algoritmo recursivo por tanto tiene un caso base para cuando m es cero en el cual se almacena en el resultado el digito y el acarreo de multiplicar los dos dígitos menos significativos de a y de b.

2.Si no se ha entrado en el caso base se almacenan en a0 y a1 las partes menos significativa y más significativa respectivamente de a haciendo lo mismo para b0 y b1 para b.

3.Se calculan t1, t2 y t3 haciendo tres llamadas recursivas a la función con sus correspondientes parámetros.

4.Se calcula el resultado final el cual se devuelve recursivamente a las llamadas anteriores.

**Parametros:**

a - ArrayList con el primer número en base 2^16 que se va a multiplicar mediante mult\_karatsuba.

b - ArrayList con el segundo número en base 2^16 que se va a multiplicar mediante mult\_karatsuba.

m - valor que va a servir para reducir hasta un caso base de cuando m sea cero.

**Returns:**

ArrayList con el resultado de multiplicar los dos número en base 2^16 mediante mult\_karatsuba.

**digitos\_menos\_sinificativos**

public static void **digitos\_menos\_sinificativos**(java.util.ArrayList<java.lang.Integer> parte, java.util.ArrayList<java.lang.Integer> num)

Copia en el ArrayList 'parte' los dígitos de la parte menos significativa del ArrayList 'num', es decir desde la posición 0 hasta la mitad del tamaño que tenga 'num' menos 1.

**Parametros:**

parte - ArrayList con la parte menos significativa del número.

num - ArrayList con el número que se va a dividir.

**digitos\_mas\_sinificativos**

public static void **digitos\_mas\_sinificativos**(java.util.ArrayList<java.lang.Integer> parte, java.util.ArrayList<java.lang.Integer> num)

Copia en el ArrayList 'parte' los dígitos de la parte más significativa del ArrayList 'num', es decir desde la mitad del tamaño que tenga 'num' hasta la última posición.

**Parametros:**

parte - ArrayList con la parte más significativa del número.

num - ArrayList con el número que se va a dividir.

**mostrar\_numero\_de\_izq\_dcha**

public static void **mostrar\_numero\_de\_izq\_dcha**(java.util.ArrayList<java.lang.Integer> vector\_enteros)

Muestra por pantalla los dígitos de un número de izquierda a derecha.

**Parametros:**

vector\_enteros - ArrayList que se va a mostrar.

**guarda\_vector\_numeros\_primos**

public static java.util.ArrayList<java.lang.Integer> guarda\_vector\_numeros\_primos(java.util.ArrayList<java.lang.Integer> num1,java.util.ArrayList<java.lang.Integer> num2)

Lee de un fichero los números primos que serán usados en las operaciones correspondientes.

**Parametros:**

num1 - ArrayList usado en los cálculos.

num1 - ArrayList usado en los cálculos.

**Returns:**

ArrayList con los primos obtenidos.

**cambio\_base\_modular**

public static java.util.ArrayList<java.lang.Integer> **cambio\_base\_modular**(java.util.ArrayList<java.lang.Integer> num, java.util.ArrayList<java.lang.Integer> primos)

Pasa cada uno de los números de 'num' a su correspondiente valor haciendo módulo con un número primo el cual hace de base.

**Parametros:**

num - ArrayList con los números con los cuales se va a hacer módulos con los correspondientes números primos.

primos - ArrayList con los números que se van a usar como bases.

**Returns:**

ArrayList con el resultado del cambio de base.

**multiplicacion\_modular**

public static java.util.ArrayList<java.lang.Integer> **multiplicacion\_modular**(java.util.ArrayList<java.lang.Integer> num1,java.util.ArrayList<java.lang.Integer> num2,

java.util.ArrayList<java.lang.Integer> primos)

Multiplica los dos números que se pasan como parámetro haciendo modulo con su correspondiente número primo.

**Parametros:**

num1 - ArrayList con el primer número que se va a multiplicar.

num2 - ArrayList con el segundo número que se va a multiplicar.

primos - ArrayList con los números primos los cuales se van a usar como bases para hacer el módulo.

**Returns:**

ArrayList con el resultado de multiplicar los dos número en base 2^16 módulo con su correspondiente número primo.

**calcular\_c\_mult\_modular**

public static java.util.ArrayList<java.util.ArrayList<java.lang.Integer>> **calcular\_c\_mult\_modular**(java.util.ArrayList<java.lang.Integer> primos\_relativos)

Calcula una matriz con los Cij´s correspondientes a los primos relativos que se pasan como parámetro.

**Parametros:**

primos\_relativos - ArrayList con los números primos que se van a usar en los cálculos.

**alg\_euclides**

public static java.util.ArrayList<java.util.ArrayList<java.lang.Integer>> **alg\_euclides**(java.util.ArrayList<java.lang.Integer> primos\_relativos)

Calcula mediante el algoritmo extendido de Euclides los inversos de los primos relativos que se le pasan por parámetro.

**Parametros:**

primos\_relativos - ArrayList con los números a los cuales se les va a calcular su inverso.

**Rueturns:**

Una matriz con los correspondientes inversos calculados.

**mostrar\_numero\_base\_dec**

public static void **mostrar\_numero\_base\_dec**(java.util.ArrayList<java.lang.Integer> vector\_enteros)

Muestra el número que se le pasa por parámetro en base decimal.

**Parametros:**

vector\_enteros - ArrayList con el número que se va a mostrar.

**pasar\_base\_mod\_a\_normal**

public static java.util.ArrayList<java.lang.Integer> **pasar\_base\_mod\_a\_normal**(java.util.ArrayList<java.lang.Integer> x,java.util.ArrayList<java.util.ArrayList<java.lang.Integer>> cij,java.util.ArrayList<java.lang.Integer> m)

Pasa un número de su base a la base normal que se está usando para las operaciones.

**Parametros:**

x - ArrayList con los Xi´s que se van a usar en las operaciones.

cij - Una matriz con los Cij´s que se van a usar el las operaciones.

m - ArrayList con los Mi´s que se van a usar en las operaciones.

**Returns:**

ArrayList con el resultado de pasar a base normal modular.

**leer\_numero**

public static int **leer\_numero**()

Lee un número desde la entrada estándar

**Returns:**

num número que se ha leído.

**main**

public static void **main**(java.lang.String[] args)