**Práctica 1**

*Algoritmos básicos*

**Unidad académica:** Análisis de Algoritmos

**Profesor a cargo:** Dra. Sandra Díaz Santiago

**Grupo:** 3CV1

**Realizada por:**

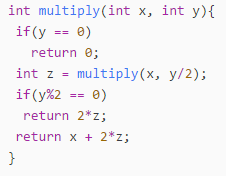
* Medina Juárez Jesús Booz
* Ríos Altamirano Alam Yael

**Sesión de laboratorio:** 21 de febrero del 2018  
**Fecha de entrega:** 27 de febrero del 2018

## Primer ejercicio

### Ejercicio 1

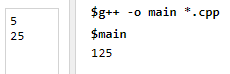
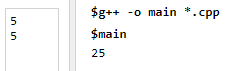
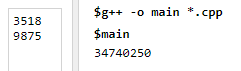
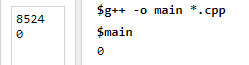
Algoritmo para multiplicar dos números



Analizando por el comportamiento del algoritmo que la complejidad del algoritmo puede ser despejada de la fórmula por propiedades de logaritmos:

Complejidad del algoritmo es siendo n el número de dígitos de z

Impresiones de pantalla

### Ejercicio 2

## Segundo ejercicio

### Primer algoritmo

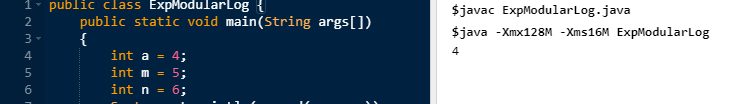
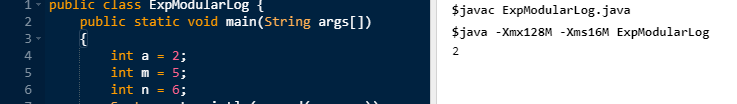
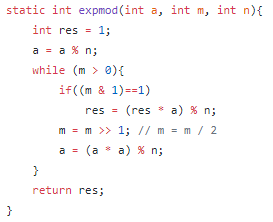
Algoritmo para realizar exponenciación modular: , recibiendo los tres enteros

Código en C++

|  |  |
| --- | --- |
|  |  |

### Segundo algoritmo

Algoritmo para realizar exponenciación modular: con complejidad logarítmica, recibiendo los tres enteros Implementación en Java 7



A comparación del algoritmo anterior este realiza un menor número de iteraciones ya que estamos aprovechando una de las reglas de multiplicación modular: , en el caso de que se trate de una potencia **par**, y una forma de separar una potencia impar es escribiéndola en binario.

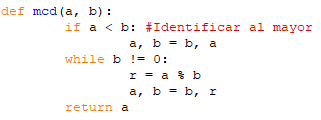
Lo que básicamente se hace en el algoritmo es asignar a la variable resultado de la primera descomposición binaria con potencia 1 y después ir dividiendo la potencia a través de corrimientos de un bit y acumulamos en el valor del módulo de la potencia correspondiente al corrimiento, en los casos en que sea un numero par haremos uso de la propiedad antes explicada para acumular el resultado en **res.**

Se trata de un algoritmo de complejidad logarítmica debido a que el número de operaciones que realiza es la cantidad de veces que se puede dividir entre 2 a la entrada.

## Tercer ejercicio

Algoritmo de Euclides para obtener el máximo común divisor de dos números

Código en Python 2



El algoritmo es correcto porque devuelve siempre el mismo máximo común divisor para las mismas entradas, y este valor es siempre el esperado.

La complejidad del algoritmo es ya que al sólo calcular los cocientes dependeremos de un valor por debajo del tamaño del más grande de los números.

También es posible analizar la complejidad del algoritmo partiendo de la serie de Fibonacci, analizando que para dos valores, el número de elementos que cubra de la serie de Fibonacci el mayor de los números será la complejidad.

Impresiones de pantalla

