Práctica 1

Medina Juárez Jesús Booz
Ríos Altamirano Alam Yael

Algoritmos básicos
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

```
int multiply(int x, int y){
 if(y == 0)
   return 0;
 int z = multiply(x, y/2);
 if(y%2 == 0)
  return 2*z;
 return x + 2*z;
}
```

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

$$2^c = y$$
$$\log_2 y = c$$

Complejidad del algoritmo es $O(\log_2 y + n^2)$ siendo n el número de dígitos de z

Impresiones de pantalla

```
5
5
```
```
$g++ -o main *.cpp
$main
25
```
```
5
25
```
```
$g++ -o main *.cpp
$main
125
```
```
3518
9875
```
```
$g++ -o main *.cpp
$main
34740250
```
```
8524
0
```
```
$g++ -o main *.cpp
$main
0
```

Segundo ejercicio

Primer algoritmo

Algoritmo para realizar exponenciación modular: $a^m mód\ n$, recibiendo los tres enteros a, m, n.

Código en C++

```
int expo(int a, int m, int n){
        if(n==1)
        return 0;
        int aux=1;
        for(int i = 0; i<m ; i++){
                aux=(aux*a)%n;
        }

        return aux;
}
```

Pruebas

```
2
5
6
```
```
$g++ -o main *.cpp
$main
2
```
```
4
5
6
```
```
$g++ -o main *.cpp
$main
4
```
```
100
34
7
```
```
$g++ -o main *.cpp
$main
2
```
```
111
23
2
```
```
$g++ -o main *.cpp
$main
1
```

Segundo algoritmo

Algoritmo para realizar exponenciación modular: $a^m mód\ n$ con complejidad logarítmica, recibiendo los tres enteros a, m, n. Implementación en Java 7

```
static int expmod(int a, int m, int n){
    int res = 1;
    a = a % n;
    while (m > 0){
        if((m & 1)==1)
            res = (res * a) % n;
        m = m >> 1; // m = m / 2
        a = (a * a) % n;
    }
    return res;
}
```

Pruebas

```
public class ExpModularLog {
    public static void main(String args[])
    {
        int a = 2;
        int m = 5;
        int n = 6;
```

```
$javac ExpModularLog.java
$java -Xmx128M -Xms16M ExpModularLog
2
```

```
public class ExpModularLog {
    public static void main(String args[])
    {
        int a = 4;
        int m = 5;
        int n = 6;
```

```
$javac ExpModularLog.java
$java -Xmx128M -Xms16M ExpModularLog
4
```

A comparación del algoritmo anterior este realiza un menor número de iteraciones ya que estamos aprovechando una de las reglas de multiplicación modular:

$$A \text{ ^ } 2\ mod\ C\ =\ (A\ *\ A)\ mod\ C\ =\ (\ (A\ mod\ C)\ *\ (A\ mod\ C)\ )\ mod\ C$$

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 a resultado de la primera descomposición binaria con potencia 1 y después ir dividiendo la potencia m a través de corrimientos de un bit y acumulamos en a el valor del módulo de la potencia correspondiente al corrimiento, en los casos en que m 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.

$$O(log_2 m)$$

Tercer ejercicio

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

Código en Python 2

```
def mcd(a, b):
        if a < b: #Identificar al mayor
                a, b = b, a
        while b != 0:
                r = a % b
                a, b = b, r
        return a
```

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 $O(\log n)$ 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

```
 RESTART: D:\ESCUELA\SextoSemestre\Algoritmos\Practicas\Primer_parcial\mcd_Eucli
des.py
55
89
1
>>>
 RESTART: D:\ESCUELA\SextoSemestre\Algoritmos\Practicas\Primer_parcial\mcd_Eucli
des.py
789
756381
3
>>>
 RESTART: D:\ESCUELA\SextoSemestre\Algoritmos\Practicas\Primer_parcial\mcd_Eucli
des.py
1368037
1367300
11
```