Ejercicios Propuestos Tema 3

Martín “n3m1dotsys” Romera Sobrado

24 de mayo de 2021

1. Ejercicio 1

Calcular el coste del Algoritmo de Euclides

```java
public int mcd (int A, int B) {
    if (B == 0) return A;
    else return mcd(B, A % B);
}
```

En el mejor de los casos `B` será 0 y la respuesta será inmediata, sin embargho si entra en el bucle de llamadas recursivas del `else` es más dificil analizar cuantas vueltas va a dar el algortimo ya que la operación módulo (`%` en *Java*) puede dar un resultado entre 0 y `B`-1, según el valor de A en ese momento. Podemos analizar el peor caso del algoritmo que según el *teorea de Lamé* es tener como entrada dos número sucesivos de la *secuencia de Fibonacci*. Prblemos como haría el algoritmo para la entrada `mcd(55,34)`:

10	`mcd(55,34)`	= `mcd(34,21)`
9	`mcd(34,21)`	= `mcd(21,13)`
8	`mcd(21,13)`	= `mcd(13,8 )`
7	`mcd(13,8 )`	= `mcd(8 ,5 )`
6	`mcd(8 ,5 )`	= `mcd(5 ,3 )`
5	`mcd(5 ,3 )`	= `mcd(3 ,2 )`
4	`mcd(3 ,2 )`	= `mcd(2 ,1 )`
3	`mcd(2 ,1 )`	= `mcd(1 ,1 )`
2	`mcd(1 ,1 )`	= `mcd(1 ,0 )`
1	`mcd(1 ,0 )`	= **1**

Cuadro 1: Ejecución de `mcd(55,34)`

Vemos que recorre todas las posibles entradas de números sucesivos de la *secuencia de Fibonacci*, y que siendo este el peor caso teniendo en cuenta que incrementará una iteración cada vez que A (y B) lleguen al siguiente número de la secuencia, deducimos que el orden de ejecución será a la inversa de del orden de crecimiento de la *secuencia de Fibonacci*, que se encuentra en un orden logarítmico $O(\log n)$.

2. Ejercicio 2

Calcular el coste del algoritmo "multiplicación rusa"

```
public int multRusa(int A, int B:) {
    if (A == 1) return B;
    if (A % 2 != 0) return B + multRusa(A/2, B*2);
    else return multRusa(A/2, B*2);
}
```

Este método consiste en realizar la suma de los valores de B multiplicados por las potencias de 2 que componen a A. Por ejemplo, si tenemos que A es 42 y B es 50. La descomposición en base 2 de 42 sería

$$42 = 2^1 + 2^3 + 2^5 = 2 + 8 + 32$$

, de forma que con el método de multiplicación rusa sería:

$$42 * 50 = 2^1 \cdot 50 + 2^3 \cdot 50 + 2^5 \cdot 50 = 2 \cdot 50 + 8 \cdot 50 + 32 \cdot 50 = 2100$$

Cada producto representa una iteración en la que que se cumplía alguno de los dos if-s. De todas formas también se itera cuando no se realizan sumas. Una representación más afín del algoritmo podría ser la siguiente tabla:

2^0	multRusa(42,50) =		multRusa(21,100)
2^1	multRusa(21,100) =	100 +	multRusa(10,200)
2^2	multRusa(10,200) =		multRusa(5 ,400)
2^3	multRusa(5 ,400) =	400 +	multRusa(2 ,800)
2^4	multRusa(2 ,800) =		multRusa(1,1600)
2^5	multRusa(1,1600) =	1600	**end**

Cuadro 2: Ejecución de multRusa(42,50)

Tiene que hacer tantas iteraciones como bits se necesitan para codificar A en base 2, es decir:

$$T(n) = \log_2(n) \in O(\log n)$$

3. Ejercicio 3

Calcular el coste del algoritmo "potencia":

```
public int potencia(int B, int N) {
    if (N == 0) return 1;
    else return B * potencia(B, N-1);
}
```

Cada recursión `N` se decrementa en 1 hasta llegar a 0, de forma que es sencillo ver que para `N`$= n$ el tiempo de ejeción es:

$$T(n) = n + 1 \in O(n)$$

4. Ejercicio 4

Calcular el coste del algoritmo "potencia optimizada":

```
public int potencia2(int B, int N) {
    if (N == 0) return 1;
    int rec = potencia2(B,N/2);
    if (N%2 == 0) return rec*rec;
    else return B * rec * rec;
}
```

Este algoritmo en vez de reducir en 1 el valor de `N` se va reduciendo por la mitad, de forma que el tiempo de ejecución para un parametro `N` con valor n será:

$$T(n) = \log_2(n) + 1 \in O(\log n)$$

5. Ejercicio 5

Calcular el coste de invertir un número:

```
public int invertir(int n) return invertirAux(0,n);
public int invertirAux(int ac, n) {
    if (n == 0) return ac;
    return invertirAux(ac * 10 + (n % 10), n/10);
}
```

De forma similar algunos de los algoritmos que hemos analizado hasta ahora, parametro que controla el número de recursiones que hay que realizar es `n` el que se divide entre 10 cada llamada recursiva, de forma que el tiempo de ejecución es:

$$T(n) = \log_{10}(n) + 1 \in O(\log n)$$