1. Calcular el coste del Algoritmo de Euclides:

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

$C_B(n)$: $O(1)$
$C_{NR}(n)$: $O(1)$
a:
b: $*$

2. Calcular el coste del algoritmo “multiplicación rusa”:

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

$C_B(n)$: $O(1)$
$C_{NR}(n)$: $O(1)$
a: 1 $(Solo\ hay\ una\ llamada)$
b: $*2$
n: $= A$

Siempre hay que buscar primero cual es el tamaño del problema.

n: El tamaño es A

* A simple vista, vemos que el problema se reduce en 2

$$O(\log_2(n)\ \cdot C_{NR}(N) + C_B(n) \rightarrow O(\log_2(n))$$

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)
        }
}
```

$C_B(n)$: $O(1)$
$C_{NR}(n)$: $O(1)$
a: 1 $(Solo\ hay\ una\ llamada)$
b: $1(caso\ donde\ menos\ se\ reduce$
n: $= N$

Siempre hay que buscar primero cual es el tamaño del problema.

n: Se hace la operación: B^N, a realizar de la siguiente manera: $B_N \cdot B_{N-1} \cdot B_{N-2} \cdot \ldots \cdot (B_{N=0} = 1)$ es por lo tanto, recursión por sustracción, y el tamaño del problema se reduce en una unidad en cada nueva llamada.

$$O(n \cdot C_{NR}(1) + C_B(1)) = O(n \cdot= O(1) + O(1)) = \boldsymbol{O(n)}$$

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);
        }
}
```

$C_B(n)$: $O(1)$
$C_{NR}(n)$: $O(1)$
a:1
b: 2
n: = N

Siempre hay que buscar primero cual es el tamaño del problema.

n: Se hace la operación: B^N, de nuevo, el tamaño del problema es N.

* A simple vista se ve que el problema se reduce a la mitad en cada nueva llamada, por lo que:

$$O(\log_2(n)\ \cdot C_{NR}(N) + C_B(n) \rightarrow O(\log_2(n)\ \cdot O(1) + O(1) = O(\log_2(n)$$

5. Calcular el coste de invertir un número:

```
public int invertir(int n) {
        return invertirAux(0,n);
}

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

$C_B(n)$: $O(1)$
$C_{NR}(n)$: $O(1)$
a:
b: $*$
n: $= N$

Siempre hay que buscar primero cual es el tamaño del problema.

n: El tamaño es n, ya que, depende del número de caracteres que tenga la cadena.

* A simple vista, vemos que el problema se reduce en 10

$$O(\log_{10}(n)\ \cdot C_{NR}(N) + C_B(n) \rightarrow O(\log_{10}(n)$$