-Ejercicio 9

a. Exprese la función del tiempo de ejecución de cada uno de los siguientes algoritmos, resuélvela y calcule el orden.
b. Compare el tiempo de ejecución del método ‘rec2’ con el del método ‘rec1’.
c. Implemente un algoritmo más eficiente que el del método ‘rec3’. (es decir, que el T(n) sea menor).

```java
static public int rec2(int n){
        if (n <= 1)
                return 1;
        else
                return (2 * rec2(n-1));
}
```

Expresamos…
cte n<= 1
T(n-1) + cte n>1

n = 8
1° T(n-1) 2° T(n-2) k-esimo T(n - k)

Igualamos en el caso base
n-k = 2 (cuando n sea 2 va a ser la última vez que entra a la recursión)
n - 2 = k
Obtenemos la función entera
T (n) = n - (n - 2) → O (n)

```java
static public int rec1(int n){
        if (n <= 1)
                return 1;
        else
                return (rec1(n-1) + rec1(n-1));
}
```

Expresamos…
n <= 1 cte
n > 1 2T(n-1)

1° 2T(n-1)

2° 2 [2T(n-2)] = 4T(n-2)

3° 4(2T(n-2)) = 8T(n-3)

K-esimo 2^k T(n-k)
Igualamos en el caso base
n-k = 2 (cuando n sea 2 va a ser la última vez que entra a la recursión)
n - 2 = k

Obtenemos la funcion
$T(n) = 2^{n-2} T(n-(n-2))$ → $O(2^n)$

```
static public int rec3(int n){
        if  ( n == 0 )
                return  0;
        else {
                if ( n == 1 )
                        return 1;
                else
                        return (rec3(n-2) * rec3(n-2));
        }
}
```

Expresamos…
n = 0 cte
n = 1 cte
n > 1 2T(n - 2)

Paso 1: 2T(n-2)

Paso 2: 2(2T(n-4)) = 4T(n-4)

Paso 3: 2(4T(n-6)) = 8T(n-6)

Paso k: $2^k(n - 2k)$

Igualamos en el caso base:
n - 2k = 2
n - 2 = 2k
(n-2)/2 = k
n/2 - 2/2 = k
(n/2) -1 = k

Obtenemos la función
$T(n) = 2^{(n/2)-1}(n - 2((n/2) -1))$ —> $O(2^n)$

```java
static public int potencia_iter(int x, int n){
        int potencia;
        if  (n == 0)
                potencia = 1;
        else {
                if  (n == 1)
                        potencia  =  x;
                else{
                        potencia  = x;
                        for (int  i  = 2 ; i <=  n ; i++) {
                                potencia *= x ;
                        }
                }
        }
        return  potencia;
}
```

n = 0	cte
n = 1	cte
n > 1	cte + $\sum_{i=2}^{n}$ cte → cte + ($\sum_{i=1}^{n}$ - $\sum_{i=1}^{1}$)cte → cte +(n - 1)cte
	→ o(n)

```java
static public int potencia_rec( int x, int n){
        if( n == 0 )
                return 1;
        else{
                if( n == 1)
                        return x;
                else{
                        if ( (n  %  2 ) == 0)
                                return  potencia_rec (x * x, n / 2 );
                        else
                                return potencia_rec (x * x, n / 2) * x;
                }
        }
}
```

n = 0	cte
n = 1	cte
n % 2 = 0	T(n/2) caso 1
n % 0 != 0	Tn(n/2) caso 2

1° $T(n/2)$
2° $T(n/4)$
3° $T(n/8)$
k- esimo $T(n/ 2^k)$

Igualamos en el caso base

caso 1:
$n/ 2^k = 0$
$n = 0$

$T(n) = (0/2^k)$

caso 2:
$n/ 2^k = 1$
$n = 2^k$
$\log_2 (n) = k$
$T(n) = (n/2^{\log(n)})$ $\rightarrow O(\log_2(n))$

☹️