Ejercicio 1

Estimar el orden de complejidad de los siguientes algoritmos, en función de N:

1.a

```
int sum = 0;
for (int n = N; n > 0; n /= 2)
for (int i = 0; i < n; i++)
     sum++;
```

Suma geométrica de 1 a 1/(2^logN). La razón es $\frac{1}{2}$. Es convergente podemos suponer de 1 a 1/infinito,
N+N/2+N/4+N/8+…+N/2∧log(N)= N(1+1/2+1/4+1/8+…)<N(1+1)=2N

1.b

```
int sum = 0;
for (int i = 1; i < N; i *= 2)
for(int j = 0; j < i; j++)
     sum++;
```

Es una suma geométrica de 1 a 2^(logN-1) con razón 2.
Sn=(an*r-a1)/(r-1)
1+2+4+8+ … +2^(logN-1)= SUM[k=0,logN-1](2∧k)=(2^(logN-1) * 2-1)/(2-1) = (2∧logN)-1=N-1

1.c

```
int sum = 0;
for (int i = 1; i < N; i *= 2)
for (int j = 0; j < N; j++)
     sum++;
```

N+N+N+…=NlogN

De los tres algoritmos cual debe tener un tiempo de ejecución mejor y cual lo tiene peor.

El mejor es el 1.b

Ejercicio 2

Un algoritmo de ordenación con una complejidad *O(n log n)* emplea 1 milisegundo en ordenar 1000 elementos. Asumiendo que el tiempo (*T(n)*) necesario para ordenar n elementos es directamente proporcional a *n log n*, esto es, *T(n) = cn log n*. Determinar una formula para calcular N elementos (*T(N)*), y estimar cuanto tiempo puede llevar ordenar 1.000.000 elementos.

T(n)=c n logn
C=T(N)/(N logN)

T(n)=(T(N)/(N logN)) * (n logn)= T(N) (n log n)/(N log N)

T(1000000) = T(1000) (1000000 log 1000000)/(1000 log 1000)= 1 (1000 2)= 2000 ms

Ejercicio 3

Un algoritmo cuadrático con un tiempo de procesamiento $T(n) = cn^2$ emplea T(N) segundos en procesar N elementos. Cuanto tiempo tardará en procesar 5000 elementos, si suponemos que para N=100 tarda *T(N) = 1ms*

T(N)=c n∧2
C=T(N)/n∧2 = 1/100∧2

T(5000) = c 5000∧2 = 5000∧2/100∧2 = 2500 ms