RETO 1

Participantes:
Rubén Mogica Garrido
Arturo Cortes Sánchez

1. *Usando la notación O, determinar la eficiencia de los siguientes segmentos de código:*

```
1   int n,j; int i=1; int x=0;//O(1)
2
3   do{                         //O(n)
4           j=1;                //O(1)
5           while (j<=n){       //O(log n)
6                   j=j*2;      //O(1)
7                   x++;        //O(1)
8           }
9           i++;
10  }while (i<=n);
```

Eficiencia total = O(n log(n))

Explicación:
El segmento presenta dos bucles y varias operaciones simples de las que suponemos que su eficiencia es $O(1)$.

El primer bucle, el do-while es de eficiencia O(n) ya que i incrementa de uno en uno con cada iteración, llegando a hacer por tanto n iteraciones.

El bucle while anidado, tiene un crecimiento de $O(\log(n))$. Esto es debido a que el entero j que determina las iteraciones del bucle, y su crecimiento viene determinado en la linea j=j*2 que nos indica que llegara pronto a cumplir el requisito del bucle (j<=n) al duplicarse en cada iteración.

Sabiendo esto, y aplicando la regla del producto por ser bucles anidados, tenemos que la eficiencia total del código es de $O(n\log(n))$

```
1   int n,j; int i=2; int x=0;      //O(1)
2   do{                             //O(n)
3           j=1;                    //O(1)
4           while(j<=i){            //O(log i) = O(log n)
5                   j=j*2;          //O(1)
6                   x++             //O(1)
7           }
8           i++;                    //O(1)
9   }while (i<=n)
```

Eficiencia total = O(n log(n))

Explicación:
Este caso es igual al anterior excepto en el bucle anidado, donde j se comparará con i en vez de con

n a la hora de iterar. El crecimiento de j seguiría siendo logarítmico, donde nos tenemos que fijar en este caso es si el cambiar n por i afectaría a la eficiencia. Aunque en la práctica este bucle sea más eficiente que el anterior, examinando el peor caso como estamos haciendo ahora, este bucle anidado se ejecutará log n veces, siendo la eficiencia por tanto $O(\log n)$.

En conclusión, volvemos a tener que la eficiencia total del código es de $O(n\log(n))$

2. *Para cada función $f(n)$ y cada tiempo t de la tabla siguiente, determinar el mayor tamaño de un problema que puede ser resuelto en un tiempo t (suponiendo que el algoritmo para resolver el problema tarda $f(n)$ microsegundos, es decir, $f(n)x10^{-6}sg$)*

f(n)/t	1 sg	1 h.	1 semana	1 año	1000 años
$\log_2(n)$	2^(10^6)	2^(36*10^8)	2^(6048*10^8)	2^(3.1536*10^13)	2^(3.1536*10^16)
n	10^6	36*10^8	6048*10^8	3.1536*10^13	3.1536*10^16
$n\log_2(n)$	62746.12646	1.33378×10^8	1.77631×10^10	7.97634×10^11	641136862352842
n^3	10^2	1532	8456	31593	315938
2^n	19	31	39	44	54
$n!$	9	12	13	16	18

Explicación: Para empezar, necesitamos los datos con la eficiencia de n para poder calcular los demás campos. Para ello, tomamos el dato del enunciado de que se tardan $f(n)\cdot 10^{-6}sg$ para resolver el problema, por lo que, al ser una función lineal, podemos calcular con facilidad el numero de ejecuciones posibles en cada apartado de tiempo.

Para calcular $\log_2(n)$, usamos su función inversa, 2^n , y los datos de n para obtener los datos. Es decir, la fórmula para calcular $\log_2(n)$ sería $f(n)=2^n$.

Lo mismo ocurre con 2^n , solo que en este caso la función inversa será el logaritmo en base 2, por lo que la fórmula es $f(n)=\log_2 n$

Para $n\log_2(n)$ hemos realizad una búsqueda en internet hasta encontrar que para calcular la función inversa de $n\log_2(n)$ hay que hacer uso la “Función W de Lambert”, resultando en la siguiente formula $f(n)=\frac{n\log(2)}{W(n\log(2))}$

En el caso de n^3 podemos usar también su función inversa, que en este caso seria la raíz cubica de n. La fórmula entonces es $f(n)=\sqrt[3]{n}$

Para el caso de $n!$ hay que usar algo distinto, pues no se puede usar una función inversa para el factorial. En este caso, al ser una eficiencia muy mala que dará pocas ejecuciones, podemos ir calculando a mano distintos valores hasta llegar a el n que sea igual o justo anterior a la eficiencia de la tabla n. Este número que encontremos será el numero de ejecuciones con una eficiencia de $n!$