Práctica 3 - Análisis de Algoritmos

1. Sea F_n la sucesión de Fibonacci:

$$\begin{aligned} F_1 &= 1 \\ F_2 &= 1 \\ F_{n+2} &= F_{n+1} + F_n \end{aligned}$$

Desarrollar fórmulas para las siguientes sumas:

$$\sum_{i=1}^{n} F_{2i-1} \qquad \sum_{i=1}^{n} F_{2i}$$

2. Encontrar una forma cerrada para la siguiente sumatoria:

$$\sum_{i=0}^{n} a + bi$$

3. ¿Cuáles de los siguientes enunciados son verdaderos? Probar las respuestas.

1. $n^2 \in O(n^3)$
2. $n^2 \in \Omega(n^3)$
3. $2^n \in \Theta(2^{n+1})$
4. $n! \in \Theta((n+1)!)$

4. Demostrar que $f \in \Theta(g)$ si y solo si $f \in O(g)$ y $g \in O(f)$

5. Sean $f, g : \mathbb{N} \to \mathbb{R}$ asintóticamente no negativas y $h(n) = f(n) + g(n)$, demostrar que

$$h(n) \in \Theta(\text{máx}(f(n), g(n)))$$

6. Dadas $f, g : \mathbb{N} \to \mathbb{R}$, demostrar las siguientes propiedades de las notaciones asintóticas:

1. O y Ω son transitivas
2. f asintóticamente no negativa $\Rightarrow f(n) \in \Theta(f(n))$
3. $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$
4. $f(n) \in O(g(n)) \Rightarrow \forall k \in \mathbb{R}^+ \cdot kf(n) \in O(g(n))$
5. $f(n) \in \Omega(g(n)) \Rightarrow \forall k \in \mathbb{R}^+ \cdot kf(n) \in \Omega(g(n))$

7. Sean a, b $\in \mathbb{R}$ constantes, b positivo, probar que

1. $(n+a)^b \in \Theta(n^b)$
2. $b^n \in \Theta(b^{n+a})$

8. Dadas las siguientes definiciones en Haskell, calcular el Work de cada una de ellas.

```
num_1 0 = 1
num_1 n = sqrt(num_1 (n - 1))  +  n

num_2 0 = 1
num_2 n = 2 * (num_2 (div n 2))  +  1
```

9. Dadas las siguientes definiciones en Haskell que implementan distintos algoritmos para multiplicar, escribir las ecuaciones correspondientes a Work:

```
producto1 n 1 = n
producto1 n m = n  +  producto n (m − 1)

producto2 n 1 = n
producto2 n m = case (even m) of
                    True  → producto2 (2 ∗ n) (div m 2)
                    False → n  +  producto (2 ∗ n) (div m 2)
```

10. Escribir las recurrencias de Work y Span de las siguientes funciones.

```
data Tree a = E | L a | N (Tree a) (Tree a)

split [] = ([], [])
split [x] = ([x], [])
split (x:y:xs) = let (ys, zs) = split xs
                 in (x:ys, y:zs)

toTree [] = E
toTree [x] = L x
toTree ls@(x:y:xs) = let (ys, zs) = split ls
                         (l, r) = toTree ys (|||) toTree zs
                         in Node l r
```