Nicolás Ramírez Calderón
257084

**7.25.**
Consider the following alternative to this chapter's powerOf2 algorithm:

```
public static long otherPowerOf2( int n)
{
 if (n > 0)
 {
  Long lowerPower = otherPowerOf2( n-1);
  return lowerPower + lowerPower;
 }
 else
 {
  return 1;
 }
}
```

Estimate the execution time of this algorithm.

**R//**
La función otherPowerOf2 para un entero n dado se ejecuta n+1 veces, y las instrucciones return se ejecutan un numero similar de veces, nótese que se realizan 2 instrucciones dentro del if, mientras que solo 1 dentro del else, así se tiene 2n+1 como el tiempo T(n), para un n cualquiera para el que se ejecute otherPowerOf2. Un análisis más detallado se encuentra en el siguiente punto.

**7.26.**
Read the analysis of powerOf2's execution time, counting:

1. The n > 0 comparisons as well as the additions.
2. The n > 0 comparisons and the subtractions as well the additions.

Do these more complete analyses still suggest execution times more or less proportional to 2n, or do the times become even worse?

**R//**
La función otherPowerOf2 para un entero n dado se ejecuta n+1 veces, y las instrucciones return se ejecutan un numero similar de veces, nótese que además de la instrucción return se ejecuta una suma en la misma instrucción y que para cada llamado a la función se realiza un decrementote n y una asignación, por tanto considere que cada línea de la forma return lowerPower + lowerPower Consume 2 unidades de tiempo y que cada línea de la forma long lowerPower=otherPowerOf2 (n-1); consume 3 unidades de tiempo.

También considere que la evaluación de un if consume dos unidades de tiempo (una mientras se evalúa la expresión, y otra mientras se hace el jump a la instrucción adecuada o simplemente a la siguiente línea). Así se tiene.

```
public static long otherPowerOf2( int n) ……1
{
 if (n > 0) ……..……2
 {
  Long lowerPower = otherPowerOf2( n-1);……3
  return lowerPower + lowerPower;……..…2
 }
 else
 {
  return 1;……1
 }
}
```

La función se llama n+1 veces pero solo una ves se llama sin decremento, ni asignación, así se tiene 1, el bloque del primer if se ejecuta n veces se tiene (2+3+2)n la línea al interior del else se ejecuta 1 sola vez.

Finalmente el tiempo según estas estimaciones es:
T(n)=1+7n+1=7n+2
La función no es del orden 2n, un argumento claro en contra de esto es que la función se llama n+1 veces, y los tiempos de ejecución de las líneas dentro de la función son en el peor de los casos, constantes (son operaciones simples, sin llamadas a funciones, o instrucciones for, etc.) así el orden de la función es O(n), que esta muy por debajo de 2n.

**7.27.**
What function of x does the following algorithm compute (assume x is a natural number)?

```
public static int mystery( int x)
{
 if (x > 0)
 {
  return mystery ( x-1) + 2* x - 1;
 }
 else
 {
  return 0;
 }
}
```

Para un n dado se hace la operación 2*n-1, y a eso se le suma el resultado de mystery(n-1), que es 2*(n-1)-1, sumado con mystery(n-2) que de nuevo es 2*(n-2)-1, se tiene:

2*n-1+ 2*(n-1)-1+2*(n-2)-1…

Finalmente cuando x=0 se devuelve 0, convirtiendo a mystery en:

$$\sum_{0}^{n} 2n-1 = 2\sum_{0}^{n} n - n = 2\frac{n(n+1)}{2} - n = n^2 + n - n = n^2$$