Tarea 4

Problema 1.

Definitivamente esto depende de la plataforma, de la máquina y de la función implementada. Lo mejor es obtener el resultado teórico. Como el proporcionado por el comando ulimit

```
cesar-mac: P01 $ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) 6144
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 256
pipe size            (512 bytes, -p) 1
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 266
virtual memory          (kbytes, -v) unlimited
```

Notemos que stacksize nos proporciona información sobre el tamaño de la pila, es decir, aproximadamente 8 MB. Si por ejemplo la función tiene una variable n de tipo entero (4 bytes), tendremos a lo más aproximadamente 2000000 de recursiones.

Aplicaré esto a la función recursiva del problema:

```
double Composicion ( double x, double (* funcion)( double ), int n ){
   return (n==0)?x:Composicion( (*funcion)(x), &*(funcion), n-1 );
}
```

Esto es: aproximadamente 8 bytes de cada uno de los 2 doubles y 4 bytes del int. Esto es entonces 14 bytes por recursión. Que serían aproximadamente: 585000 recursiones.

Tarea 4

También se puede usar un algoritmo similar al siguiente como método alternativo para calcularlo:

```
int MaxRecursion ( int n ){
   printf("Recursion: %d\n", n);
   return MaxRecursion(n+1);
}
```

Ejecutado en dos máquinas se obtuvieron los siguientes resultados decepcionantes:

Procesador	**SO**	**Recursiones**
AMD Athlon64 X2Dual Core Processor 5200+	Ubuntu	523838
2.4 Ghz Intel Core 2 Duo	Leopard	>67000000

En el segundo caso no terminé la ejecución por que me dió flojera...