Tarea II

Para entregar el jueves 27 de agosto al inicio de la clase, excepto las preguntas 6 y 7. No hay que entregar la solución de las preguntas 8, 9 y 10.

1. (continuación de la Tarea I) Consideramos una cadena de longitud n donde en cada posición aperece 0, 1, 2 o 3. Sea a_n el número de diferentes cadenas que se pueden formar con un número par de 0's. Entonces:

   $$a_{n+1} = 2a_n + 4^n.$$

   Resuelva explícitamente esta recursión.

2. Decimos que la función $f(n)$ es **menor** que $g(n)$ si f es $O(g(n))$. Ordena las siguientes funciones (de menor a mayor):
   $nlog(n), (log(n))^2, 5n^2 + 7n, n^{5/2}, n!, 2^{n!}, 4^n, n^n + log(n), 5^{log(n)}, log(n!)$, $\sqrt{n}, \exp(n), 8n + 12, 10^n + n^{20}$.

3. Si $f(n)$ y $g(n)$ son $O(n^2)$, entonces ¿$f(n)/g(n)$ es $O(1)$? Demuéstralo o da un contraejemplo.

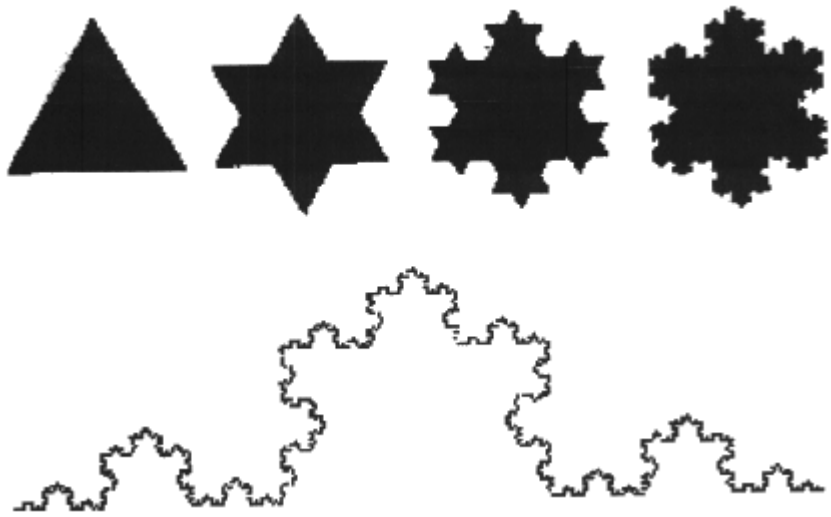
4. (esperar en resolverlo hasta el martes) Resuelva la siguiente recursión:

   $$T(n) = 2T(n/4) + \sqrt{n}.$$

   Verifica tu resultado con lo que dice el Teorema Master para este caso.

5. (continuación de la Tarea I; esperar en resolverlo hasta el martes) En la primera tarea vimos un método iterativo para obtener aproximaciones a números irracionales. Impleméntalo en Python y construye una(s) gráfica(s) informativa(s) que muestra(n) como mejora la aproximación cuando n crece
   (consulta `http://matplotlib.org/users/pyplot_tutorial.html`). Entregalo como IPython notebook.

6. (para entregar el 1 de septiembre) Escribe un programa recursivo que dibuja una curva de von Koch en IPython.
   Leer `https://en.wikipedia.org/wiki/Koch_snowflake`



7. (para entregar el 1 de septiembre) Supongamos que tenemos A, un arreglo $2^n \times 2^n$ ordenado, es decir:

$$A[i,j] \leq A[i,j+1] \quad A[i,j] \leq A[i+1,j], \forall i,j$$

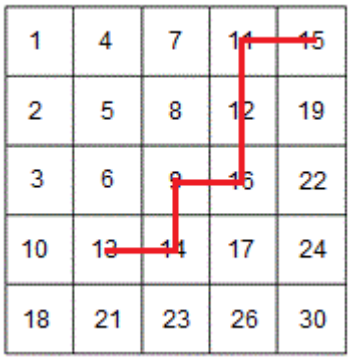
   Queremos ver si aparece el valor x en A.

   a) (no entregar) Una extensión de búsqueda binaria para este caso consiste en comparar $A[2^{n-1}, 2^{n-1}]$ con x y dependiendo de la respuesta buscar en subarreglos de A de manera recursiva. Escribe el algoritmo en Python y verifica que su complejidad (peor caso) cumple con

$$T(n) = 3T(n/2) + c$$

   Resuelva en $T(n)$.

   b) Existen varios algoritmos $O(n)$. Uno está basado en empezar en la celda de la esquina superior derecha y dependiendo de la comparación de la celda con el elemento que se busca, uno va un paso hacia abajo o un paso hacia la izquierda (ver la figura para la búsqueda de 13)



Escribe un algoritmo en Python que implementa esta idea. Verifica que es $O(n)$.

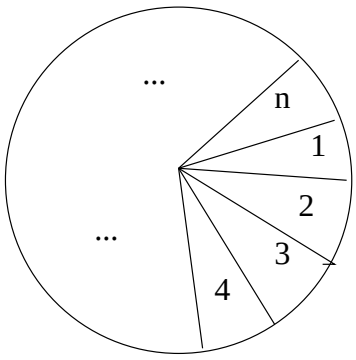
8. (no entregar) Considera la siguiente recursión:

$$T(n) = 2T(n/2) + nlog(n) \quad T(1) = 2$$

Resuélvala con la técnica general que vimos para ecuaciones lineales de diferencias. Puedes suponer que $n = 2^m$.
Verifica que es un ejemplo donde el Teorema Master (que veremos en la clase de martes) no aplica.

9. (no entregar) Define $a_n = \sum_{i=0}^{n} i^2$. Encuentra una recursión lineal para a_n y resuélvela para obtener una expresión explícita para a_n.

10. (no entregar) Dividimos un pastel en n rebanadas. Si tenemos $p > 2$ colores diferentes, define a_n como el número de maneras para asignar a cada rebanada un color tal que dos rebanadas colindantes no tengan el mismo color.



Demuestra que

$$a_n = (p-2)a_{n-1} + (p-1)a_{n-2}.$$

Resuelva la recursión.