Divide y vencerás:

Las torres de Hanoi.

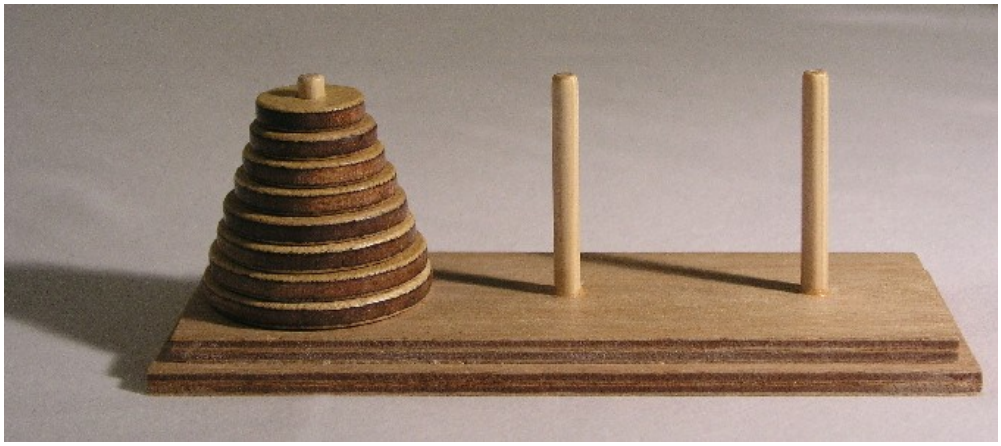
Ángel Alberto Hamilton López
alu0100888102
alu0100888102@ull.edu.es

Introducción.

Para esta práctica se nos pide programar en java/c++ (java en mi caso) y analizar el algoritmo recursivo para resolver el juego de “Las torres de Hanoi”.

Este juego consiste en lo siguiente: Se dispone de 3 torres y N discos de tamaños decrecientes colocados en una de las torres, y se nos pide que pasemos todos los discos de una torre a otra siguiendo las siguientes reglas:

- Solo se puede mover un disco cada vez.
- Cada disco ha de empezar y terminar su movimiento en alguna de las torres.
- No se puede colocar un disco sobre otro más pequeño.



El algoritmo.

El algoritmo más eficiente para resolver este problema forma parte de los llamados algoritmos “divide y vencerás” y es el siguiente:
Nota: llamamos a los discos con números del 1 a N, siendo 1 el menor disco y N el mayor.

```
resuelve( D = numero_del_disco, torre_de_origen, torre_de_destino, torre_auxiliar){
        si ( D = 1)
                mover (D, torre_de_destino)
        en otro caso {
                resuelve(D-1, torre_de_origen, torre_auxiliar, torre_de_destino)
                mover (D, torre_de_destino)
                resuelve(D-1, torre_auxiliar, torre_de_destino, torre_de_origen)
        }
}
```

Este algoritmo divide el problema en subproblemas, que consisten en mover un disco D a una torre de destino. Para hacer esto, tenemos que quitarle los que tenga encima y luego volvérselos a colocar, por lo que el problema se reduce a tener que mover el disco D-1 al poste auxiliar, luego el disco D al destino, y por último el disco D-1 al destino. Claro que D-1 puede tener otros discos encima, por lo que la división se repite de manera recursiva hasta llegar al disco 1, el cual siempre es libre de moverse. Este algoritmo resuelve el problema en *2^n-1* movimientos, donde *n* es el número de discos

Análisis del algoritmo.

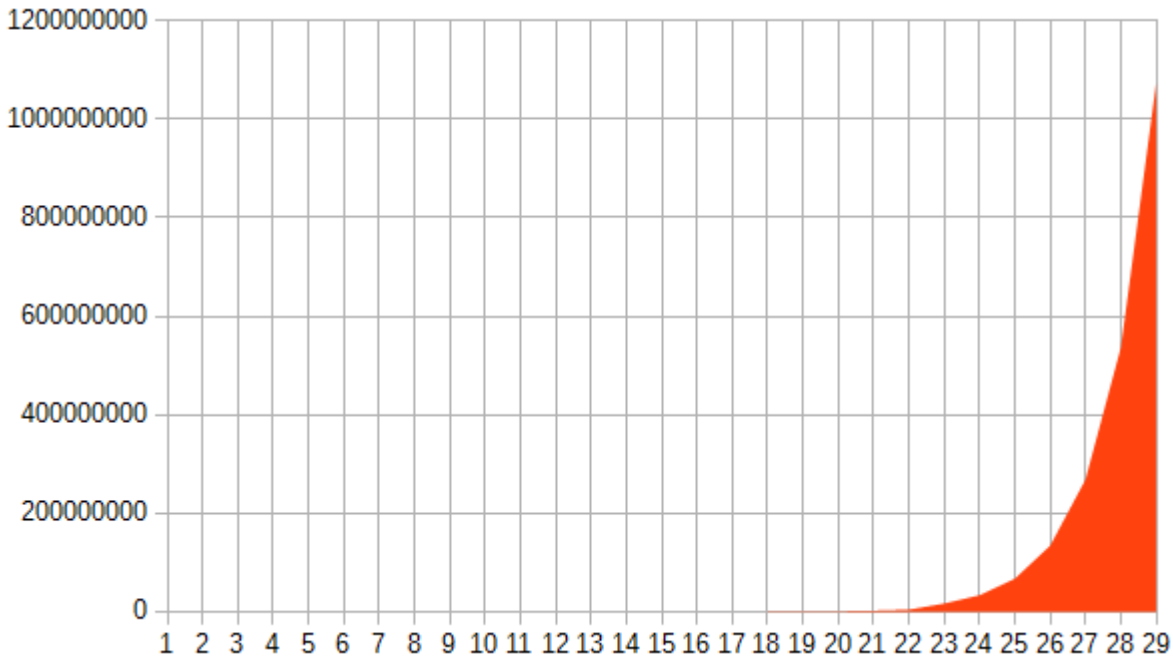
El programa desarrollado recibe como parámetro el número de discos con el que se quiere resolver el problema y aplica el algoritmo previamente explicado para resolverlo, pudiendo mostrar el proceso por pantalla de forma opcional. Siendo un algoritmo con complejidad exponencial, cabría esperar que el numero de movimientos necesarios se disparara así como el tiempo necesario. Para comprobarlo se recogieron los datos del número de pasos y el tiempo de ejecución del algoritmo para 1, 2, 3… hasta 29 discos.

Tabla de datos.

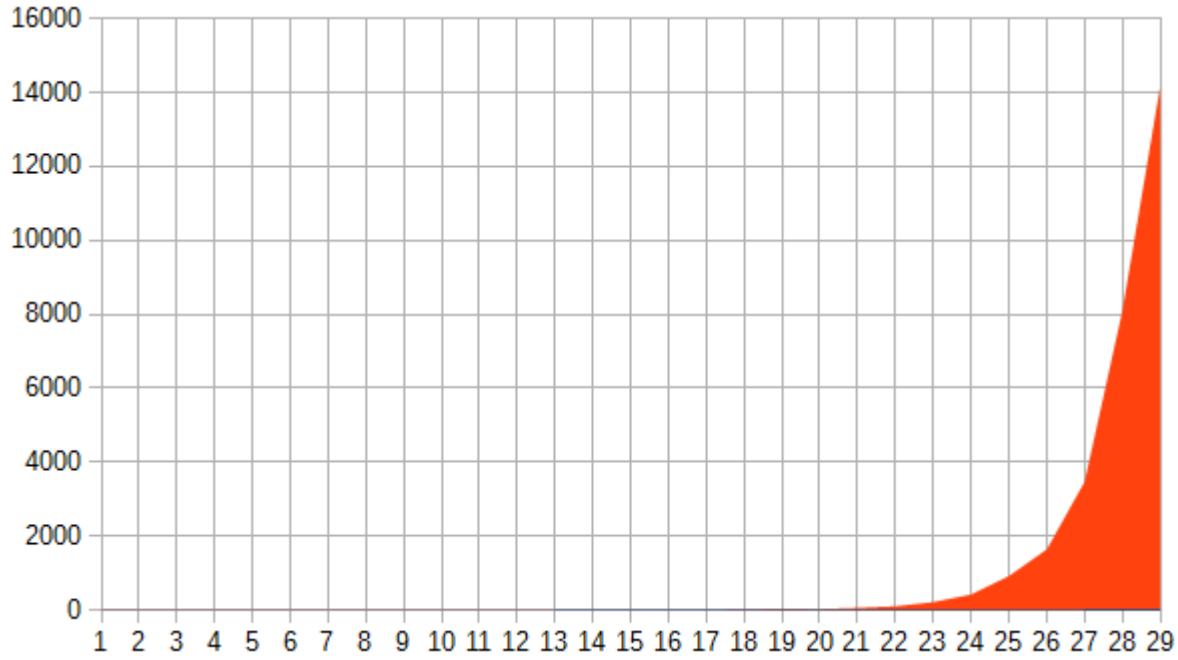
Discos	Tiempo	Pasos
1	0,029311	1
2	0,015633	3
3	0,026869	7
4	0,055204	15
5	0,108453	31
6	0,31168	63
7	0,506603	127
8	0,83929	255
9	1,244767	511
10	1,984886	1023
11	1,888158	2047
12	4,003969	4095
13	1,34345	8191
14	2,614109	16383
15	3,863762	32767
16	5,107063	65535
17	10,240995	131071
18	22,228476	262143
19	33,820248	524287
20	24,95348	1048575
21	57,742447	2097151
22	102,261441	4194303
23	217,13128	16777215
24	422,08309	33554431
25	920,563038	67108863
26	1641,351677	134217727
27	3472,032552	268435455
28	8102,38113	536870911
29	14191,59157	1073741823

Nótese que existe cierto error en el tiempo, pues al estar trabajando con fracciones de milisegundos, cualquier mínimo cambio en el software externo (como que haya muchas tareas en la CPU en ese momento) hace que varíe el resultado.

Gráfica del número de movimientos en función del número de discos.



Gráfica del tiempo (en ms) en función del número de discos.



Conclusiones del análisis.

Observando las gráficas se puede ver perfectamente como la complejidad exponencial del algoritmo afecta tanto al número de movimientos como al tiempo de ejecución, los cuales se doblan por cada disco nuevo que se añade. Podemos concluir que es un algoritmo poco eficiente, y que sólo es práctico cuando trabajamos con pocos discos.

También cabe destacar, a modo de curiosidad, que hasta los 25 discos la resolución tarda menos de 1 segundo y se podría considerar “práctico” el algoritmo, sin embargo pasa de aproximadamente 1 segundo a 14 en 4 discos. Para 40 discos, la solución tardaría aproximadamente 4 horas en completarse a esta velocidad de computación, y para los 64 discos de la leyenda que originó el problema se necesitarían más de 190 días de cálculos.

Recursos externos.

https://en.wikipedia.org/wiki/Tower_of_Hanoi
http://estdatosgrupoa16.blogspot.com.es/2009/02/las-torres-de-hanoi-recursivo.html
https://recursividad.wordpress.com/2009/12/02/las-torres-de-hanoi-ejemplo-de-recursividad/