**Presented By:**

**Sivan Saar**

**026524983**

**Software engineering**

**Course:**

**Software engineering seminar**

**Isaac Aviv**

**Semester B 2010**

# Table of contents

[Summary 3](#_Toc261385465)

[Introduction – Tower of Hanoi 4](#_Toc261385466)

[The recursive solution for 3-pegs Hanoi Tower 5](#_Toc261385467)

[Introduction 5](#_Toc261385468)

[Method of action 5](#_Toc261385469)

[Analysis of the algorithm 6](#_Toc261385470)

[Algorithm complexity: 7](#_Toc261385471)

[4-peg- Tower of Hanoi 8](#_Toc261385472)

[Introduction 8](#_Toc261385473)

[Method of action 9](#_Toc261385474)

[Analysis of the algorithm 11](#_Toc261385475)

[Code Implementation 12](#_Toc261385476)

[Recursive tree 14](#_Toc261385477)

[Algorithm Complexity 18](#_Toc261385480)

[The classical linear hybrid Hanoi tower 21](#_Toc261385481)

[Introduction 21](#_Toc261385482)

[Method of action 21](#_Toc261385483)

[Analysis of the algorithm 23](#_Toc261385484)

[Code implementation 26](#_Toc261385485)

[Recursive tree 28](#_Toc261385486)

[4-peg colored tower 32](#_Toc261385490)

[Introduction 32](#_Toc261385491)

[Method of action 33](#_Toc261385492)

[Analysis of the algorithm 33](#_Toc261385493)

[Code implementation 34](#_Toc261385494)

[Recursive tree 35](#_Toc261385495)

[Resources 38](#_Toc261385498)

[Articles: 38](#_Toc261385499)

[On-line resources 39](#_Toc261385500)

# Summary

The tower of Hanoi puzzle, invented by Edouard Lucas in 1883.

Many variations have been proposed as exercises and challenges over the past 125 years.

In this seminar I will shortly review the basic "Towers of Hanoi" problem with 3 towers as a preface to the three main problems:

1. Tower of Hanoi with four pegs
2. The classical linear hybrid tower of Hanoi
3. The four peg colored Hanoi tower

I will talk about the solution for the basic 3-pegs Hanoi problem, and then I will show the basic idea, the algorithm and the minimal amount of steps it takes to solve the problems mentioned above.

# Introduction – Tower of Hanoi

The tower of Hanoi is a mathematical game that was invented by French mathematician Edouard Lucas in 1883.

Lucas described the problem as a legend that tells about an old Vietnamese temple that contained three towers on which 64 disks were stacked, each one of them was in a different size, arranged all on one of the three towers, in ascending size, order from top to bottom.

The task of the monks was to transfer the N disks to one of the other two towers in accordance with two rules:

1. Only one disk can be moved at a time.
2. A disk cannot be placed directly on top of a smaller disk.

The legend tells that when the monks finish their job it will be the end of the world.

If the legend were true, and if the monks were able to move disks at a rate of one per second, using the smallest number of moves, it would take 264-1 second, 18,446,744,073,709,551,615 moves that will take about 600 years. (see [10] )

# The recursive solution for 3-pegs Hanoi Tower

## Introduction

There are two optimal solutions for the game, one is recursive and the other is iterative.

The recursive solution is easy to understand but difficult to implement, on the other hand, the iterative solution is easy to implement but is difficult to understand.

In this paper I will focus on the recursive solution only.

## Method of action

Like any other recursive problem, we need to rely on solution of the problem in a smaller dimension.

In the problem of Hanoi tower it is done in the following manner:

If we need to move N disks, first we move N-1 disks to the buffer peg, and then we have only one disk so we move it from the initial peg to target peg.

After that we use again the principal of solving the problem on a smaller dimension, and we move the N-1 disks from the buffer peg to the target peg. (see [9])

## Analysis of the algorithm

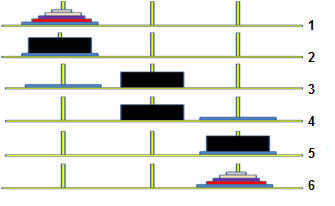


Figure 1: step for the solution of the 3-peg tower of Hanoi

Pseudo code for the algorithm:

Hanoi\_Tower(disks, initial\_peg, buffer\_peg, target\_peg)

1. If (disks == 1)
   1. Move one disk from initial\_peg to target\_peg
2. Else
   1. Hanoi\_Tower(disks-1, initial\_peg, target\_peg, , buffer\_peg)

/\* step 2-3 from figure 1 \*/

* 1. Move one disk from initial\_peg to target\_peg
  2. Hanoi\_Tower(disks-1, buffer\_peg, initial\_peg, target\_peg)

/\* step 4-5 from figure 1 \*/

## Algorithm complexity:

Required 2n-1 moves to move all the disks from initial peg to target peg.

By means of mathematical induction, it is easily proven that the above procedure requires the minimal number of moves possible, and that the produced solution is the only one with this minimal number of moves.

Because this is not the main subject of this work I’m not going to show the mathematical induction solution.

According to the algorithms number of step, it’s easy to show that the complexity is O(2n). (see [9] )

# 4-peg- Tower of Hanoi

## Introduction

The first four pegs version was proposed in 1970 by Henry Dudeney in the first chapter of his famous book, “The Canterbury Puzzles”.   
In order to entertain the pilgrims on their way to Canterbury.

The problem is as follows: a pilgrim is challenged by the reeve to move a stack of cheese of varying sizes from the first of four stools to another stool without ever placing a cheese on one of the smaller diameter. Also only one piece of cheese can be moved from one peg to another at a given time. (see [2]).

It is clear that it had the same rules as the tower of Hanoi problem, only with four pegs instead of three.

But wean while 3-peg problem has unique moving order, 4-peg problem has not because one more buffer peg means optimal move selection and therefore different moving orders.

J.S.Frame propose one algorithm to solve this problem in “America Mathematic Monthly” in 1941, and in 2004, the algorithm is proved by YangKai and XuChuan using mathematical induction in “Acta Scientiarum Naturalium Universities Pekinensis” ( see [3]).

And afterward LiueDuo and DaiYiqi provide mathematic prove of the minimum moving steps about even more general problem with multi-pegs (see [4]).

## Method of action

Like I mentioned before in this problem we have two buffers pegs so the problem seem to be more difficult, but now I’m going to show that there is a pattern to solve the algorithm.

The algorithm begins with the selection of an integer R(n), such that 1=< R(n) =< n, where n is the total number of disks.

When there is only one disk, move it from peg 1 to peg 4 and stop.

For n>1 the algorithm proceeds recursively using the following three steps:

1. Move the smallest disks n-R(n) from 1 to 2 with the help of buffers 3 and 4.
2. Move the rest disks from 1 to 4 with 3 as buffer, using the 3-peg recursive solution.
3. Move the n-R(n) disks from 2 to 4 with the help of buffer 1 and 3.

(see [1]).

We can describe it in a formula like that:

But what is R(n) and how we can calculate his value?

R(n) is actually the core of the whole problem, it’s the number of disks we need to move from one peg to another (like describe above) and each time he had a different value depend on n (see [1]).

To solve the problem with minimum steps we need to calculate R(n) like that:

(see [1], [3] ).

And the minimum step is:

(see [1], [4] ).

With the help of the formulas I mention before we could describe the way the algorithm works for some data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | TH(n) | R(n) | Moving Steps | Correlation Items |
| 1 | TH(1,1,2,3,4) | 1 | 1 | TH(1,1,2,3,4) = 1,4 |
| 2 | TH(2,1,2,3,4) | 1 | 3 | TH(1,1,3,4,2)  TH3(1,1,3,4)  TH(1,2,1,3,4) |
| 3 | TH(3,1,2,3,4) | 2 | 5 | TH(1,1,3,4,2)  TH3(2,1,3,4)  TH(1,2,1,3,4) |
| 4 | TH(4,1,2,3,4) | 2 | 9 | TH(2,1,3,4,2)  TH3(2,1,3,4)  TH(2,2,1,3,4) |
| 5 | TH(5,1,2,3,4) | 2 | 13 | TH(3,1,3,4,2)  TH3(2,1,3,4)  TH(3,2,1,3,4) |
| 6 | TH(6,1,2,3,4) | 3 | 17 | TH(3,1,3,4,2)  TH3(3,1,3,4)  TH(3,2,1,3,4) |
| 7 | TH(7,1,2,3,4) | 3 | 25 | TH(4,1,3,4,2)  TH3(3,1,3,4)  TH(4,2,1,3,4) |
| 8 | TH(8,1,2,3,4) | 3 | 33 | TH(5,1,3,4,2)  TH3(3,1,3,4)  TH(5,2,1,3,4) |
| 9 | TH(9,1,2,3,4) | 3 | 41 | TH(6,1,3,4,2)  TH3(3,1,3,4)  TH(6,2,1,3,4) |

(see [1] ).

## Analysis of the algorithm

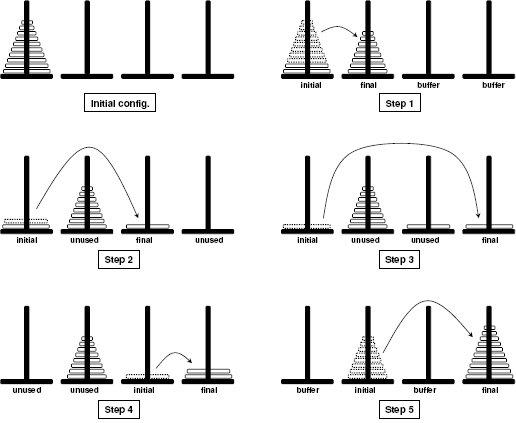


Figure 2: step for the solution of the 4-peg tower of Hanoi

**Pseudo code:**

TH(N, 1, 2, 3, 4)

1. If (N==1)
   1. Move the disk from 1 to 4

/\* figure 2- step 3 \*/

1. Else
   1. TH(N-R(N), 1, 3, 4, 2)

/\* figure 2- step 1 \*/

* 1. TH3(R(n), 1, 3, 4)

/\* figure 2- step 2-4 \*/

* 1. TH(N-R(N), 2, 1, 3, 4)

/\*figure 2- step 5 \*/

## Code Implementation

#include "stdio.h"

#include <math.h>

// the recursive function for 3-peg tower of hanoi

void TH3(int n, int f, int t, int b)

{

if(n!=0)

{

TH3(n-1, f, b, t); // recursive call

printf( "TH3 - Move the disk on top from %d to %d\n", f, t);

TH3(n-1, b, t, f); // recursive call

}

}

// the func that solve the 4-peg tower of hanoi problem

void TH(int n, int f, int b1,int b2, int t)

{

int r=0;

if (n == 1)

{

printf( "TH - Move the disk on top from %d to %d\n", f, t);

}

else

{

r=(int)floor((sqrt((double) 8\*n+1)-1)/2); // calaulate the value or r for the curent n

TH(n-r,f,b2,t,b1); // recursive call

TH3(r,f,t, b2); // call to 3-peg recursive func

TH(n-r,b1,f,b2,t); // recursive call

}

}

void main()

{

int number=0;

// get the number of disks to move

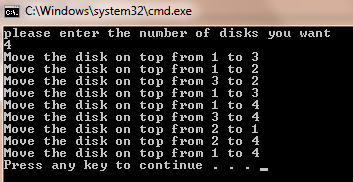
printf("please enter the number of disks you want\n");

scanf("%i",&number);

TH(number,1,2,3,4); // call for 4-peg TH func wite “number” of disks to move from tower 1 to tower 4 with the help of tower 2 and 3

}

The output of this program for running with 4 disks is:



After three steps we get the following result:

We can see that we moved the N-R(n) disks to the buffer tower.

Now we need to move the disks that stay on the source tower to the destination tower, we can see that it happens after three more steps and we get the following result:

Finally we need to move the N-R(n) disks (from step 1) from the buffer tower to the destination tower, we can see that it happens after three more steps and we get the following result:

In conclusion, we can see that the algorithm works.

## Recursive tree

In order to understand the algorithm we will follow other the running with the help of use in the recursive tree.

We run the algorithm for TH(4,1,2,3,4), it means that we need to move 4 disks from tower 1 to tower 4 with the help of tower 2 and tower 3.

The recursive tree is built from three main levels, as described below:

For convenience purposes, I separated the tree into three levels:

### First level:

Second level:

### Third level:

## Algorithm Complexity

The number of steps is depending on the number of steps that take to move the disks in the algorithm’s three main stages.

Stage 1: number of steps it takes to move the R(n) disks from source tower to buffer tower.

Stage 2: number of steps it takes to move the other disks from source tower to destination tower.

Stage 3: the number of steps it takes to move the R(n) disks from buffer tower to destination tower.

We could describe it in a formula like that:

T(n,4) = 2T(n-R(n) , 4) + T(R(n) ,3)

In this solution we use the formula above to calculate the number of disks to move from source tower to destination tower.

With this number ( R(n) ) we could calculate the minimum steps to move all the disks from source tower to destination tower, we can do it with the help of the formula bellow

The table bellow shows for each number of disks (that needs to move) the number of R(n)s that we could choose and the number of moves for each one of them

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Number of steps | | | | | | Number of disks |
| R(n)=6 | R(n)=5 | R(n)=4 | R(n)=3 | R(n)=2 | R(n)=1 |
| **1** | **1** | **1** | **1** | **1** | **1** | 1 |
| **3** | **3** | **3** | **3** | **3** | 3 | 2 |
| **5** | **5** | **5** | **5** | 5 | 7 | 3 |
| **9** | **9** | **9** | 9 | 9 | 15 | 4 |
| **13** | **13** | 17 | 13 | 13 | 31 | 5 |
| **17** | 33 | 21 | 17 | 21 | 63 | 6 |
| 65 | 37 | 25 | 25 | 29 | 127 | 7 |
| 69 | 41 | 33 | 33 | 45 | 255 | 8 |
| 73 | 49 | 49 | 41 | 61 | 511 | 9 |
| 81 | 57 | 57 | 57 | 93 | 1023 | 10 |
| 89 | 97 | 65 | 73 | 125 | 2047 | 11 |
| 97 | 105 | 81 | 89 | 189 | 4095 | 12 |
| 193 | 113 | 113 | 121 | 253 | 8191 | 13 |
| 201 | 129 | 129 | 153 | 381 | 16383 | 14 |
| 209 | 145 | 145 | 185 | 509 | 32767 | 15 |
| 225 | 225 | 177 | 249 | 765 | 65535 | 16 |
| 241 | 241 | 241 | 313 | 1021 | 131071 | 17 |
| 257 | 257 | 273 | 377 | 1533 | 262143 | 18 |
| 449 | 289 | 305 | 505 | 2045 | 524287 | 19 |
| 465 | 321 | 369 | 633 | 3069 | 1048575 | 20 |
| 481 | 481 | 497 | 761 | 4093 | 2097151 | 21 |
| 513 | 513 | 561 | 1017 | 6141 | 4194303 | 22 |
| 545 | 545 | 625 | 1273 | 8189 | 8388607 | 23 |
| 577 | 609 | 753 | 1529 | 12285 | 16777215 | 24 |
| 961 | 673 | 1009 | 2041 | 16381 | 33554431 | 25 |
| 993 | 993 | 1137 | 2553 | 24573 | 67108863 | 26 |
| 1025 | 1057 | 1265 | 3065 | 32765 | 134217727 | 27 |
| 1089 | 1121 | 1521 | 4089 | 49149 | 268435455 | 28 |
| 1153 | 1249 | 2033 | 5113 | 65533 | 536870911 | 29 |
| 1217 | 1377 | 2289 | 6137 | 98301 | 1073741823 | 30 |
| 1985 | 2017 | 2545 | 8185 | 131069 | 2147483647 | 31 |
| 2049 | 2145 | 3057 | 10233 | 196605 | 4294967295 | 32 |
| 2113 | 2273 | 4081 | 12281 | 262141 | 8589934591 | 33 |
| 2241 | 2529 | 4593 | 16377 | 393213 | 17179869183 | 34 |
| 2369 | 2785 | 5105 | 20473 | 524285 | 34359738367 | 35 |
| 2497 | 4065 | 6129 | 24569 | 786429 | 68719476735 | 36 |

(see [5] )

We can see from the table that our formula to calculate R(n) seems to be true and formal prove could find in reference [4].

The solution grows sub-exponentially, at the rate of O()(see[7]) .

# The classical linear hybrid Hanoi tower

## Introduction

In this problem there are three pegs arranged in a row (for definiteness, we number the pegs 1,2,3 with 2 being the middle peg) and there are n disks of distinct sizes.

At the beginning all n disks arranged in usual small on large order on one of the three pegs.

Each disk is colored black or white. All disks movements must obey the usual classical Hanoi rules, but black disks have the additional restriction of having to obey the linear Hanoi rule of not being able to travel directly from one end peg to other (see [6]).

## Method of action

We can see that the CLHH problem is in fact three problems in one, because we depend not only on disks colorings but on whether we are moving our disk.

There are three possible movements:

1. End to end
2. End to mid
3. Mid to end

The algorithm starts in calling to the end to end function that works like that:

If (color of the biggest disk == black)

1. Move all n-1 disks from source peg to the destination peg (with the help of midToend and endTomid function).
2. Move the n disk to the middle peg.
3. Move all n-1 disks from distention peg to source peg
4. Move n disk from middle peg to source peg
5. Move all n-1 peg from source peg to destination peg

Else if (color of the biggest disk == white)

1. Move all n-1 disks from source peg to the middle peg (with the help of midToend and endTomid function).
2. Move n disk from source peg to destination peg
3. Move al n-1 disks from middle disks to destination peg.

The aim of midToend function is to move the disks from the middle peg to one of the end peg (1 or 3 depend who is the “from” peg).

The aim of endTomid function is to move the disks from one of the end peg (1 or 3) to the middle peg.

Noticed that the middle peg is always number 2, and if x is a end peg so the other peg is 4-x (if x=1 then the other end peg is 4-1=3 and if x=3 then the other end peg is 4-3=1)(see [6]).

We will use this in the Implement of the algorithm.

## Analysis of the algorithm

For the nth black disk:

For the nth white disk:

Pseudo code of the algorithm:

endToend ( N, FROM)

{

if (N>0)

if (N == BLACK)

{

endtoend (N-1, FROM);

"move disk N from tower FROM to tower 2"

endtoend (N-1, 4-FROM);

"move disk N from tower 2 to tower 4-FROM"

endtoend( N-1, FROM);

}

else if (N color == WHITE)

{

endtomid (N-1, FROM);

"move disk N from tower FROM to tower 4-FROM"

midtoend (N-1, 4-FROM);

}

}

midToend (N, TO)

{

if (N>0)

{

midtoend (N-1, 4-TO);

"move disk N from tower 2 to tower TO"

endtoend (N-1, 4-TO);

}

}

endTomid (N , FROM)

{

if (N>0)

{

endtoend (N-1, FROM);

"move disk N from tower FROM to tower 2"

endtomid (N-1, 4-FROM);

}

}

## Code implementation

#include "stdio.h"

#define BLACK 1

#define WHITW 2

void midtoend (int n, int TO);

void endtomid (int n, int FROM);

void endtoend (int n, int FROM);

int color[10];

void main()

{

int number=0;

printf("please enter the number of rings you want to move \n");

scanf("%i", &number);

for (int i=1; i<=number; i++)

{

if ( (i%2) == 0)

color[i] = 2;

else

color[i] = 1;

}

endtoend(number, 1);

}

void endtoend (int n, int FROM)

{

if (n>0)

if (color[n]== BLACK)

{

endtoend (n-1, FROM);

printf("move ring %d from tower %d to tower 2\n", n ,FROM );

endtoend (n-1, 4-FROM);

printf("move ring %d from tower 2 to tower %d\n",n,4-FROM );

endtoend( n-1, FROM);

}

else

{

endtomid (n-1, FROM);

printf("move ring %d from tower %d to tower %d\n",n FROM,4- FROM );

midtoend (n-1, 4-FROM);

}

}

void midtoend (int n, int TO)

{

if (n>0)

{

midtoend (n-1, 4-TO);

printf("move ring %d from tower 2 to tower %d\n", n ,TO );

endtoend (n-1, 4-TO);

}

}

void endtomid (int n, int FROM)

{

if (n>0)

{

endtoend (n-1, FROM);

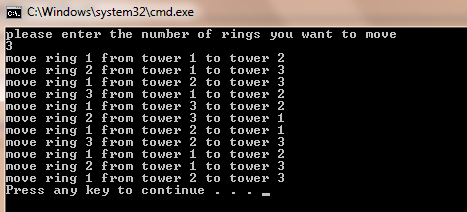
printf("move ring %d from tower %d to tower 2\n", n ,FROM );

endtomid (n-1, 4-FROM);

}

}

The output for 3 disks is:



## Recursive tree

In order to understand the algorithm we will follow it with the help of a recursive tree.

We run the algorithm for endToend(4,1), it means that we need to move 4 disks from tower 1 to tower 3.

The recursive tree is built from five main levels, as described below:

For convenience purposes, I separated the tree into three levels:

### First level:

**3**

**1**

**2**

**4**

### Second level:

**3**

**4**

**1**

**2**

### Third level:

**3**

**1**

**2**

# 4-peg colored tower

## Introduction

In this version of Hanoi tower there are four pegs arranged in a row, labeled A to D.

A stack of n black disks is initially on peg A and a stack of n white disks is initially on peg B.

The white disks and the black disks are arranged in the usual, small to large, order. In other words, each tower creates a tower of Hanoi with one color.

The goal is to create one Hanoi tower on peg C with 2n disks in a way that each white disk will be put on top of a black disk interchangeably.

## Method of action

When there is only one disk on peg A and one disk on peg B, move the one from peg A to peg C, and then, move the disk from peg B to peg C as well.

For n>2 the algorithm proceeds recursively using the following four steps:

1. Move the smallest disks n-2 from A and B to D with the help of buffer C.
2. Move the black disks from A to C.
3. Move the white disks from B to C.
4. Move the n-2 disks from D to C with the help of buffer B.

See reference[8].

## Analysis of the algorithm

**Pseudo code:**

Colored\_tower(N, A, B, C, D)

1. If (N==2)
   1. Move disk 2 from A to C
   2. Move disk 1 from B to C
2. Else
   1. Colored\_tower (N-2, A, B, D, C)
   2. Move disk N from A to D
   3. Move disk N-1 from B to D
   4. Hanoi\_Tower(N-2, D,C,B)

See reference [8]

## Code implementation

#include "stdio.h"

void Hanoi ( int n , char from , char to , char help ) {

if(n==1)

printf("Move disk 1 (white), from %c to %c\n",from,to);

else

{

Hanoi(n-1,from,help,to);

if (n%2)

printf("Move disk %d (white), from %c to %c\n",n,from,to);

else

printf("Move disk %d (black), from %c to %c\n",n,from,to);

Hanoi(n-1,help,to,from);

}

}

void colored\_tower(int n,char from1,char from2,char to,char help)

{

if (n==2){

printf("Move disk 2 (black), from %c to %c\n",from1,to);

printf("Move disk 1 (white), from %c to %c\n",from2,to);

}

else

{

colored\_tower(n-2,from1,from2, help,to);

printf("Move disk %d (black), from %c to %c\n",n,from1,to);

printf("Move disk %d (white), from %c to %c\n",n-1, from2,to);

Hanoi(n-2,help,to,from2);

}

}

void main()

{

int n=0;

printf("please enter the number of rings you want to move \n");

scanf("%i", &n);

colored\_tower ( n , 'A' ,'B', 'C' , 'D' );

}

**The output of 4 disks:**

## 

## Recursive tree

In order to understand the algorithm we will follow it with the help of a recursive tree.

We run the algorithm for CT(4,A,B,C,D). It means that we have two black disks on peg A and two white disks on peg B; we need to move all 4 disks to peg C.

The recursive tree is built from 4 main levels, as described below:

For convenience purposes, I separated the tree into two levels:

## First level:

**4**

**3**

**1**

**2**

## Second level:

**3**

**2**

**1**

# Resources

## Articles:

[1] J.Wang and J.Liu and G.Yue and L.Shao, A Non-recursive Algorithm for 4-Peg Hanoi Tower

[2] K.Paul, Variations on the Four-Post Tower of Hanoi Puzzle

[3] K. Yang and C. Xu, The Preliminary Probe of 4-Peg Hanoi Tower

[4] D. Liu and Y.Q. Dai, Study of Hanoi Tower Problem with Multi Pegs

[5] Man-Keun Lee, The graph for the Tower of Hanoi with four pegs

[6] S.Minsker, Another brief recursion excursion to Hanoi

[7] D.Berend and A.Sapir, Complexity of the Path Multi-Peg Tower of Hanoi.

[8] Dr Y.Aviv, Hanoi tower problem with two colors.

## On-line resources

[9] <http://en.wikipedia.org/wiki/Tower_of_Hanoi>

[10] <http://www.cut-the-knot.org/recurrence/hanoi.shtml>