جامعة هواري بومدين للعلوم والتكنولوجيا

USTHB

Tower of Hanoi

SURNAME: Bouldja	**SURNAME:** Boutmedjet
NAME: Idir	**NAME:** Abd Elmoudjib
NUMBER: 222231355509	**NUMBER:** 232331338812
GROUP: 1	**GROUP:** 2
SURNAME: Louni	**SURNAME:** Lisri
NAME: Mohammed Said	**NAME:** Akram
NUMBER: 232331499716	**NUMBER:** 232331200815
GROUP: 1	**GROUP:** 2

6th December 2025

Contents

1 Introduction

The Tower of Hanoi is a well-known mathematical puzzle and recreation with a fascinating history. The namesake of the problem is an actual octagonal tower in Hanoi, Vietnam, built in 1812 and, when including its flag mast, standing 41 metres high.

The puzzle is also called the Lucas Tower after Édouard Lucas (1842–1891) of Saint Louis, France, who created it in 1883 and published it under the anagrammatic pseudonym *Professeur N. Claus de Siam, Mandarin du Collège de Li-Sou-Stian.* It is also sometimes referred to as the Tower of Brahma, after a fictional legend that Lucas invented to accompany the puzzle [**lucas1883**]. By the time the puzzle reached Ball (1892), the legend had become heavily embellished, and later Gardner (1965), a popular writer of mathematical recreations, appeared confused about some of its details.

For this reason, we cite the original description from the leaflet that accompanied the puzzle:

> *D'après une vieille légende indienne, les brahmes se succèdent depuis bien longtemps, sur les marches de l'autel, dans le Temple de Bénarès, pour exécuter le déplacement de la Tour Sacrée de Brahma, aux soixante-quatre étages en or fin, garnis de diamants de Golconde. Quand tout sera fini, la Tour et les brahmes tomberont, et ce sera la fin du monde !*

In other words, the Holy Tower of Brahma is said to be located in a temple in the Indian city of Benares. The tower consists of 64 disks, each made of pure gold and decorated with diamonds, mounted on a cylindrical stele. The priests of the temple are tasked with transferring the disks from the initial stele to one of the two remaining steles, although the legend does not specify which one, nor how frequently the disks are moved. When the final disk is placed, the priests will collapse, the tower will fall, and the world will end. Gardner (1965, p. 58) even stated that before the task is completed, "the temple will crumble into dust and the world will vanish in a clap of thunder."

The mathematical problem, therefore, is to determine whether we ought to be concerned about when this supposed end of the world might occur.

2 Theoretical Overview

Formal Definition of the Problem

The classical Tower of Hanoi problem consists of three vertical pegs, named A, B, and C, and $n \geq 1$ disks of different sizes. Initially, all the disks are placed on the source peg, stacked from largest at the bottom to smallest at the top, forming a tower.

The objective is to transfer the entire tower from the source peg to the destination peg by performing only allowed moves. An allowed move is defined as transferring the top disk of any peg to the top of another peg without violating the following rules:

- Only one disk may be moved at a time;
- Only the uppermost disk on a peg may be moved;

- A larger disk may never be placed on top of a smaller disk.

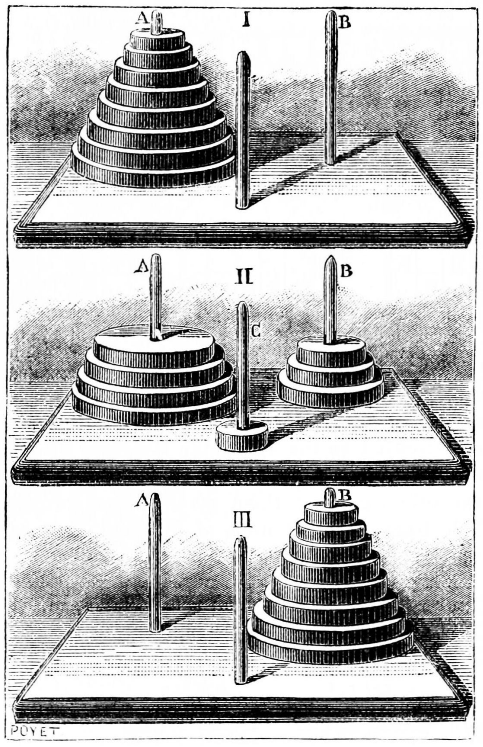

Figure 1: Initial, middle, and final configurations of the Tower of Hanoi.

Design of the Data Structures

The Tower of Hanoi is modelled using simple structures: **stacks** for pegs and **integers** for disks.

Each peg is a LIFO stack, allowing only the top disk to be moved in line with the puzzle rules. Disks are represented by integers, with smaller values denoting smaller disks, facilitating size comparisons to ensure that a larger disk is never placed atop a smaller one.

Initial State of the Stacks

Figure 2 shows the initial configuration of the stacks at the start of the puzzle.

Final State of the Stacks

Figure 3 illustrates the final configuration once all disks have been transferred to the destination peg according to the rules of the puzzle.

3 Solving The Problem

General Solution

a) Case of a Single Disk

Simply move the single disk from the source peg to the destination peg.

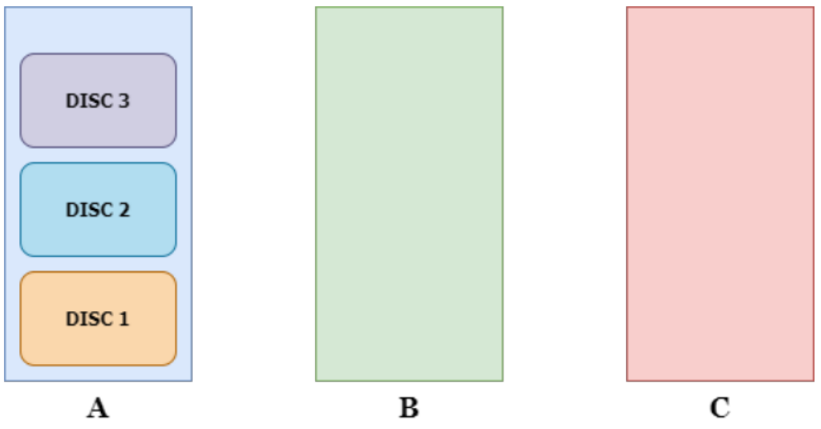

Figure 2: Initial state of the stacks representing the Tower of Hanoi.

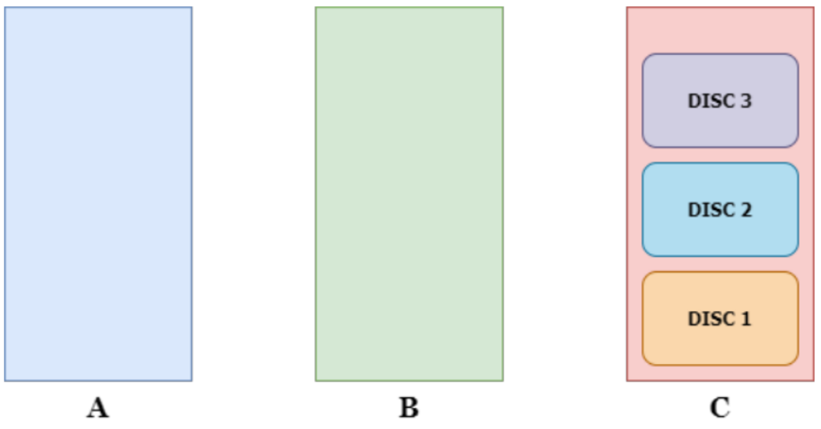

Figure 3: Final state of the stacks representing the completed Tower of Hanoi.

b) Case of Multiple Disks

1. Move the top $n-1$ disks onto the auxiliary peg, leaving the largest disk on the source peg.
2. Move the largest disk onto the destination peg.
3. Finally, move the $n-1$ disks from the auxiliary peg onto the destination peg, stacking them on top of the largest disk.

c) Optimal Number of Moves

The minimal number of moves required to solve the puzzle with n disks is

$$M(n) = 2^n - 1.$$

This can be seen by induction:

Base case: $n = 1$ requires 1 move, which satisfies $2^1 - 1 = 1$.

Induction step: Assume n disks require $2^n - 1$ moves. For $n + 1$ disks, move the top n disks to the auxiliary peg ($2^n - 1$ moves), move the largest disk to the destination peg (1 move), and then move the n disks onto it ($2^n - 1$ moves). Total moves:

$$(2^n - 1) + 1 + (2^n - 1) = 2^{n+1} - 1.$$

Hence, by induction, the formula holds for all $n \geq 1$.

3.1 Recursive Method

Algorithm Presentation:

```
PROCEDURE Hanoi_Recursive(n, src, dest, aux)
BEGIN
    // Base case: if there are no disks, do nothing
    If (n = 0) Then
        Return ;
    EndIf ;

    // Move n-1 disks from source to auxiliary peg
    Hanoi_Recursive_Function(n - 1, src, aux, dest) ;

    // Move the nth disk from source to destination
    Write("Move disk ", n, " from ", src, " to ", dest) ;

    // Move the n-1 disks from auxiliary to destination peg
    Hanoi_Recursive_Function(n - 1, aux, dest, src) ;
END ;
```

Complexity Analysis

The recursive solution follows a divide-and-conquer approach, where the problem of moving n disks is reduced to moving $n - 1$ disks twice, plus one move for the largest disk.

Time Complexity:

Let $T(n)$ denote the number of moves required for n disks. The recursive relation is:

$$T(n) = 2 \cdot T(n-1) + 1, \quad T(1) = 1$$

Solving this recurrence gives:

$$T(n) = 2^n - 1$$

Therefore, the time complexity of the recursive method is $\mathbf{O(2^n)}$.

Space Complexity:

The maximum depth of the recursion is n, which corresponds to the call stack usage. Each recursive call requires constant space, so the space complexity is $\mathbf{O(n)}$.

3.2 Iterative Method

Algorithm Presentation:

```
ALGORITHM Hanoi_Iterative
Var
    src, aux, dest : Stack ;
    n, total_moves, i : integer ;
    temp : character ;
BEGIN
    // Create stacks for source, auxiliary, and destination pegs
    src  <- Create_Stack(n) ;
    aux  <- Create_Stack(n) ;
    dest <- Create_Stack(n) ;

    // Push disks onto source peg (largest at bottom, smallest at top)
    For i <- n To 1 Do
        Push(src, i) ;
    EndFor ;

    // Compute total number of moves
    total_moves <- 2^n - 1 ;

    // Swap auxiliary and destination pegs if n is even
    If (n MOD 2 = 0) Then
        temp <- dest ;
        dest <- aux ;
        aux <- temp ;
    EndIf ;

    // Perform moves iteratively
    For i <- 1 To total_moves Do
        If (i MOD 3 = 1) Then
            Move_Disk(src, dest) ;
        ElseIf (i MOD 3 = 2) Then
            Move_Disk(src, aux) ;
        Else
            Move_Disk(aux, dest) ;
        EndIf ;
    EndFor ;
END.

FUNCTION Create_Stack(capacity : integer) : Stack
BEGIN
```

```
    // Returns a new empty stack of given capacity
END ;

PROCEDURE Push(s : Stack, disk : integer)
BEGIN
    // Push disk onto stack
END ;

FUNCTION Pop(s : Stack) : integer
BEGIN
    // Remove and return top element of stack
    // Return -1 if stack is empty
END ;

FUNCTION Peek(s : Stack) : integer
BEGIN
    // Return top element of stack without removing it
    // Return -1 if stack is empty
END ;

PROCEDURE Move_Disk(from, to : Stack)
Var
    f, t : integer
BEGIN
    f <- Peek(from) ;
    t <- Peek(to) ;

    If (f = -1) Then
        Pop(to) ;
        Push(from, t) ;
    ElseIf (t = -1) Then
        Pop(from) ;
        Push(to, f) ;
    ElseIf (f < t) Then
        Pop(from) ;
        Push(to, f) ;
    Else
        Pop(to) ;
        Push(from, t) ;
    EndIf ;

    // Increment move counter
```

```
END ;
```

Complexity Analysis of Iterative Method

The iterative solution uses three stacks to represent the pegs and performs disk moves based on a fixed sequence determined by the parity of n and the modulo operation. Each move is executed explicitly without recursion.

Time Complexity:

The total number of moves is the same as in the recursive approach:

$$T(n) = 2^n - 1$$

Each move involves a constant number of operations (peek, pop, push), so the time complexity is

$$\mathbf{O(2^n)}.$$

Space Complexity:

The iterative method requires three stacks, each capable of storing up to n disks. No recursion is used, so the call stack does not grow. Therefore, the space complexity is

$$\mathbf{O(n)}$$

for each stack, giving overall stack usage of $3 \cdot O(n) = O(n)$.

4 Experimental Setup

This section presents the environment, methodology, and tools used to benchmark the recursive and iterative implementations of the Tower of Hanoi problem.

Hardware and Software Configuration

Experiments were conducted on an Intel Core i3 laptop running Arch Linux.

Benchmarking Methodology

The benchmarks were performed using the `main.c` program, which measures:

- Execution time of the recursive algorithm `hanoi_recursive(n,'A','C','B')`.
- Execution time of the iterative algorithm `hanoi_iterative(n,'A','B','C')`.
- Total number of moves performed (`move_count`).

The tested values of n (number of disks) were: 1,3,5,10,12,14,16,18,20,22,24,25. Execution times were computed using `clock()` and converted to seconds as:

$$t = \frac{\text{end} - \text{start}}{\text{CLOCKS\_PER\_SEC}}$$

The table below summarises the measured results.

Experimental Results

Table 1: Execution times and move counts for recursive and iterative algorithms

n	Recursive Time (s)	Iterative Time (s)	Move Count
1	0.000001	0.000003	1
3	0.000001	0.000002	7
5	0.000001	0.000001	31
10	0.000002	0.000015	1023
12	0.000019	0.000031	4095
14	0.000024	0.000101	16383
16	0.000208	0.000602	65535
18	0.000299	0.001622	262143
20	0.002895	0.006647	1048575
22	0.004837	0.029614	4194303
24	0.045021	0.125130	16777215
25	0.076152	0.208974	33554431

5 Comparison and Analysis

This section compares the recursive and iterative implementations of the Tower of Hanoi algorithm from both theoretical and empirical perspectives, focussing strictly on the differences between them.

Complexity Comparison

Both algorithms perform exactly $2^n - 1$ moves and therefore share the same time complexity:

$$T(n) = \Theta(2^n).$$

The key difference concerns space:

- **Recursive:** requires $O(n)$ space from the call stack.
- **Iterative:** requires $O(n)$ space for three explicit stacks (one per peg).

Thus, asymptotically, both use linear space, but *the recursive structure is built-in and optimized*, whereas the iterative version manages its own data structures manually.

Empirical Differences

Despite identical asymptotic time complexity, measurements show clear differences:

- For small n (up to ~ 12), the two implementations run at indistinguishable speeds.
- For moderate n (from ~ 14 upward), the recursive version becomes consistently faster.

- At $n = 25$, the recursive version is roughly three times faster than the iterative version.

This difference grows with n because each move in the iterative version incurs more overhead.

Why the Recursive Version Is Faster

The recursive implementation benefits from several structural advantages. It uses the call stack, which is lightweight, tightly packed, and optimised for push/pop operations, whereas the iterative version uses three user-managed stacks with additional bounds checks, index manipulation, and conditional logic.

Per move, the recursive method performs only a few function calls and returns, whilst the iterative method executes multiple stack operations, comparisons, and parity checks. Branching overhead is also higher in the iterative version, which must select the appropriate stack pair for each move, whereas recursion follows a fixed call pattern.

Finally, memory locality favours recursion: stack frames are contiguous and cache-friendly, whilst the iterative method accesses multiple scattered arrays, causing more cache misses. These factors collectively increase the per-move cost of the iterative implementation.

Interpretation

The recursive and iterative algorithms have the same theoretical complexity, but their *effective constant factors* differ substantially. The recursive implementation leverages runtime-optimised mechanisms (stack frames, predictable control flow), whilst the iterative implementation must simulate recursion through explicit structures, increasing the operational overhead of each move.

As a result:

$$\text{Recursive is faster in practice, despite identical } \Theta(2^n) \text{ time complexity.}$$

The experiments confirm that theoretical complexity alone is insufficient for performance prediction: when two algorithms share the same asymptotic class, their implementation strategy and per-operation cost become decisive.

Practical Limits on the Number of Disks

In practice, both implementations are limited by exponential runtime rather than memory. Since each algorithm performs $2^n - 1$ moves, the computation becomes infeasible once this value exceeds roughly 10^9 operations. This places the practical limit at:

$$n \approx 30.$$

The recursive version has an additional constraint from stack depth, but this limit occurs far beyond any usable range (tens of thousands), well after runtime has already made the problem impossible to compute. Therefore, both approaches are effectively limited by time, and their usable range is the same:

$$\boxed{n_{\max} \approx 30}.$$

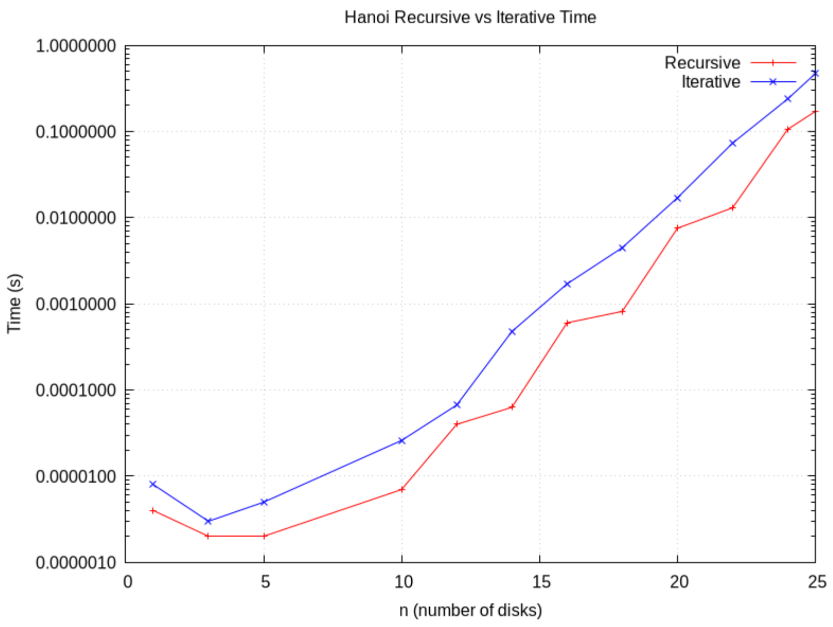

Figure 4: Execution time comparison for recursive and iterative algorithms.

6 Conclusion

During this project, we analysed the classical Tower of Hanoi problem and implemented two solutions in C: one using a recursive approach and the other using an iterative approach. Theoretical studies showed that both methods run in $O(2^n)$ time, whilst the recursive method requires $O(n)$ stack space.

The experimental results confirmed the exponential growth predicted by theory: execution time increased rapidly with n, and both algorithms produced exactly $2^n - 1$ moves. However, the recursive implementation consistently outperformed the iterative one for all tested values of n, due to lower overhead and more efficient use of the call stack.

In summary, even though both approaches share identical theoretical complexity, the recursive formulation proved more efficient in practice for this problem. This highlights the need to consider both theoretical behaviour and implementation overhead when selecting an algorithm.

7 Group Contributions

- Abd Elmoudjib Boutmedjet: Implementation of the recursive algorithm and benchmarking.
- Akram Lisri: Implementation of the iterative algorithm and testing.
- Mohammed Said Louni: First half of the report, designing of the project architecture.
- Idir Bouldja: Second half of the report, preparation of the figures and graph plotting.