21.12.2023

COMP 303
ANALYSIS OF ALGORITHMS
PROJECT REPORT

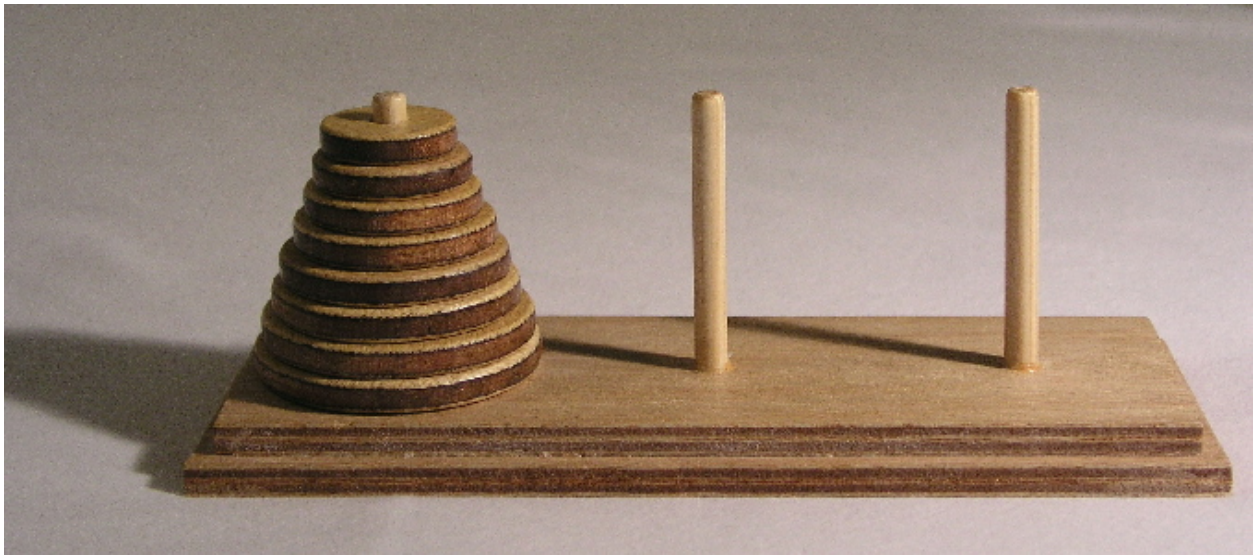
Students: Mehmet Fatih Çapal - 042101115
Mustafa Eren Soyhan - 042101008
Rahiq Al-huslan - 042101140
Mevlüt Korkmaz - 042101057

Instructor: Yassine Drias

1.Introduction

Tower of Hanoi Problem:

The Tower of Hanoi is a classic problem in mathematics and computer science especially in the study of algorithms and their analysis. The problem was invented by Édouard Lucas, a french mathematician in 1883. The game involves 3 towers and a number of disks of different sizes which can slide onto these towers. These disks are arranged in ascending order, largest at the bottom to the smallest at the top, thereby creating a pyramid or conical shape. The difficulty of the game is due to moving an entire stack of disks to another tower whilst following certain rules.



1.Tower of Hanoi Puzzle(1)

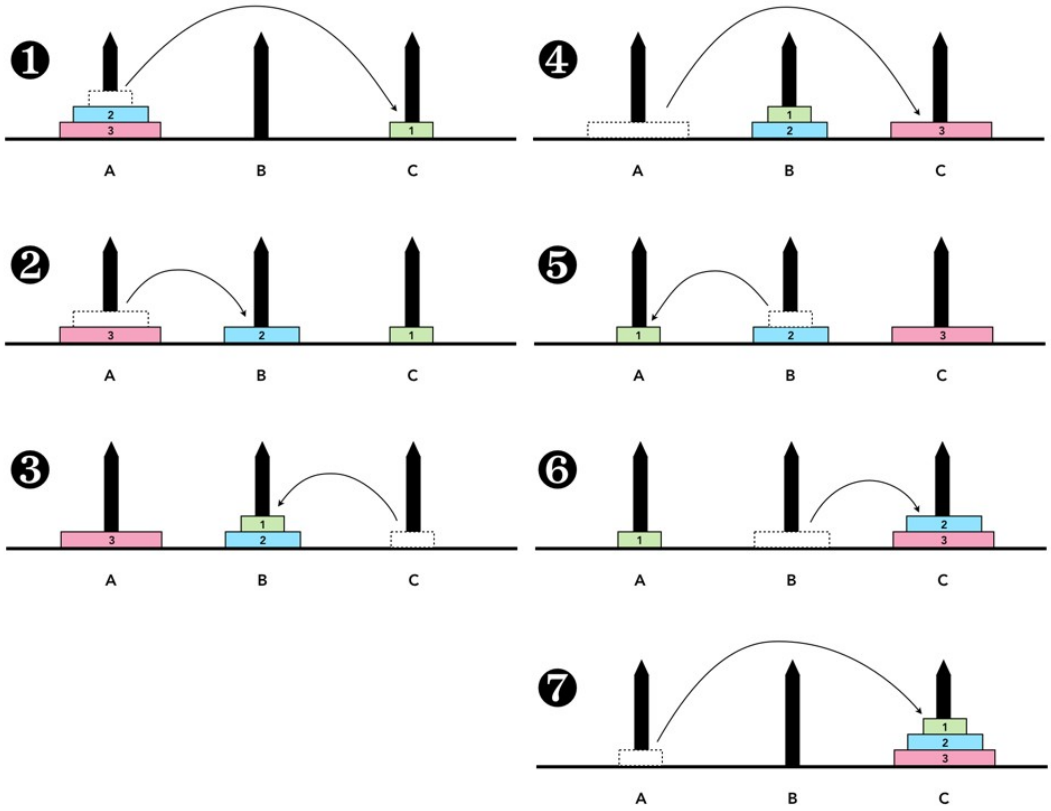
The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. You're only able to move one disk at a time.
2. The highest disk from one stack is removed with each move, and it is then placed either atop another stack or on an empty rod.
3. A disk that is smaller than it cannot be stacked on top of another disk.

The rules are simple but as the number of disks increases the complexity and strategic depth of the problem also increases exponentially.

Importance of The Tower of Hanoi:

While the puzzle itself is a fun and challenging game, it has gained significance in a number of fields, including mathematics, computer science, and psychology. In mathematics, the Tower of Hanoi is used as an example of a problem that can be solved using recursion, a fundamental concept in computer science. In computer science, the puzzle has been used to illustrate algorithms for solving problems, including sorting and searching.(2)



2. Solution of a Tower of Hanoi with Three Disks(3)

The tower of hanoi's step by step approach to problem-solving is a good example to show for complex task management, where a larger problem is broken down into smaller, more manageable components just like the classic strategy move, divide and conquer.

2.Recursive and Iterative Approach

Recursive Function:

```
def _move_recursive(self, n, source, target, auxiliary):
    if n > 0 and not self.stop_flag:
        self._move_recursive(n - 1, source, auxiliary, target)
        disk = self.positions[source].pop()
        self.positions[target].append(disk)
        self.movements.append(f"Move disk {disk} from {source} to {target}")
        self.update_movements()
        self.master.after(0, self.draw_towers)
        self._move_recursive(n - 1, auxiliary, target, source)
```

3.The Code of The Recursive Approach

def _move_recursive(self, n, source, target, auxiliary):

This is the definition of the recursive function. The function takes 4 parameters:

n: The number of disks to be moved.
source: The source tower/rod from which disks are to be moved.
target: The target tower/rod to which disks are to be moved.
auxiliary: An auxiliary tower/rod that is used during the recursive moves.

if n > 0 and not self.stop_flag:

This condition checks whether the number of disks n is greater than 0 and a stop flag (self.stop_flag) is not set. The stop flag is used to terminate the recursive process.

self._move_recursive(n - 1, source, auxiliary, target)

This is a recursive call to _move_recursive with n - 1 disks. It moves the top n-1 disks from the source tower to the auxiliary tower, using the target tower as an auxiliary.

disk = self.positions[source].pop()
self.positions[target].append(disk)
self.movements.append(f"Move disk {disk} from {source} to {target}")

This block of code represents the actual movement of a disk from the source tower to the target tower. It updates the disk positions, records the movement in a list (self.movements), and appends a corresponding movement string.

self.update_movements()
self.master.after(0, self.draw_towers)

These lines update the movements and change the GUI animation respectively.

self._move_recursive(n - 1, auxiliary, target, source)

This is another recursive call to _move_recursive to move the n-1 disks from the auxiliary tower to the target tower, using the source tower as an auxiliary.

Iterative Function:

```
def _move_iterative(self, n, source, target, auxiliary):
    s, t, a = source, target, auxiliary
    total_moves = 2**n - 1
    for i in range(1, total_moves + 1):
        if self.stop_flag:
            break

        if i % 3 == 1:
            self.move_disk(s, t)
        elif i % 3 == 2:
            self.move_disk(s, a)
        else:
            self.move_disk(a, t)
        self.update_movements()
        self.master.after(0, self.draw_towers)
```

4. The Code of The Iterative Approach

```
def _move_iterative(self, n, source, target, auxiliary):
s, t, a = source, target, auxiliary
total_moves = 2**n - 1
```

This function starts by initializing variables s, t, and a with the values of source, target, and auxiliary. These variables are used to represent the three towers in the Tower of Hanoi problem. The variable total_moves is calculated as (2^n)-1, that is the total number of moves needed to solve the Tower of Hanoi problem with n disks.

```
for i in range(1, total_moves + 1):
    if self.stop_flag:
        break
    if i % 3 == 1:
        self.move_disk(s, t)
    elif i % 3 == 2:
        self.move_disk(s, a)
    else:
        self.move_disk(a, t)
        self.update_movements()
        self.master.after(0, self.draw_towers)
```

The function then enters a loop that iterates over the total number of moves((2^n)-1) needed to solve the problem. Inside the loop:

The code checks if the stop_flag is set, and it has the same purpose that is used in the recursive function. It terminates the algorithm.

The loop uses the modulo operator (%) to determine the type of move to perform based on the current iteration (i). If i % 3 == 1, it performs a move from the source tower (s) to the target tower (t), if i % 3 == 2, it performs a move from the source tower (s) to the auxiliary tower (a), and if i % 3 == 0, it performs a move from the auxiliary tower (a) to the target tower (t).

After each move, it updates the movements and it updates the animation of the towers in the GUI application using self.master.after(0, self.draw_towers).

3.GUI And GUI's Functionalities

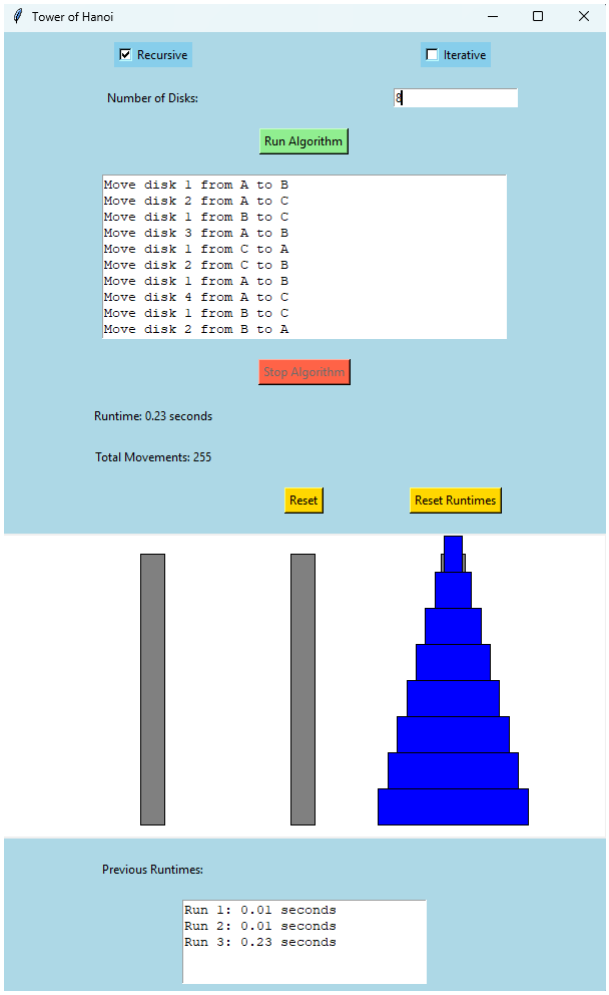
This is the graphical user interface that we used to show the code's functions.

We used libraries Tkinter to show our GUI using the Tk toolkit, while also using time to handle time related tasks and threading to allow for the creation of threads and the application to run long-running tasks.

On the top side there are 2 options to choose. Those clickable buttons let users choose which approach to solve the problem.

Below that user can decide how many disks to use in the initial tower. After the user decides the approach path and disk number, the program can be executed with the green "Run Algorithm" button.

The white rectangle below the green button shows every single relocation that disks made between the three towers.



5.GUI of The Algorithm

"Runtime" and "Total Movements" sections show the runtime of the algorithm and total movements of the disks that are made until the program terminates. Program can terminate when (2^n)-1 moves are completed or the user can terminate it manually.

The red "Stop Algorithm" button stops the whole process when clicked and the "Reset" button reverses the GUI into its starting state(except previous runtimes). "Reset Runtimes" button clears all the previously recorded runtimes in the last white rectangle area.

Lastly the biggest white rectangle area shows the animation of the algorithm.

4.Performance of The Algorithm

These both approaches have their own advantages and disadvantages. Recursive approach is simpler to implement and easier to read, but it tends to be slower. On the other hand, the iterative way is faster than the recursive approach but it has more complex logic and the implementation might be harder.

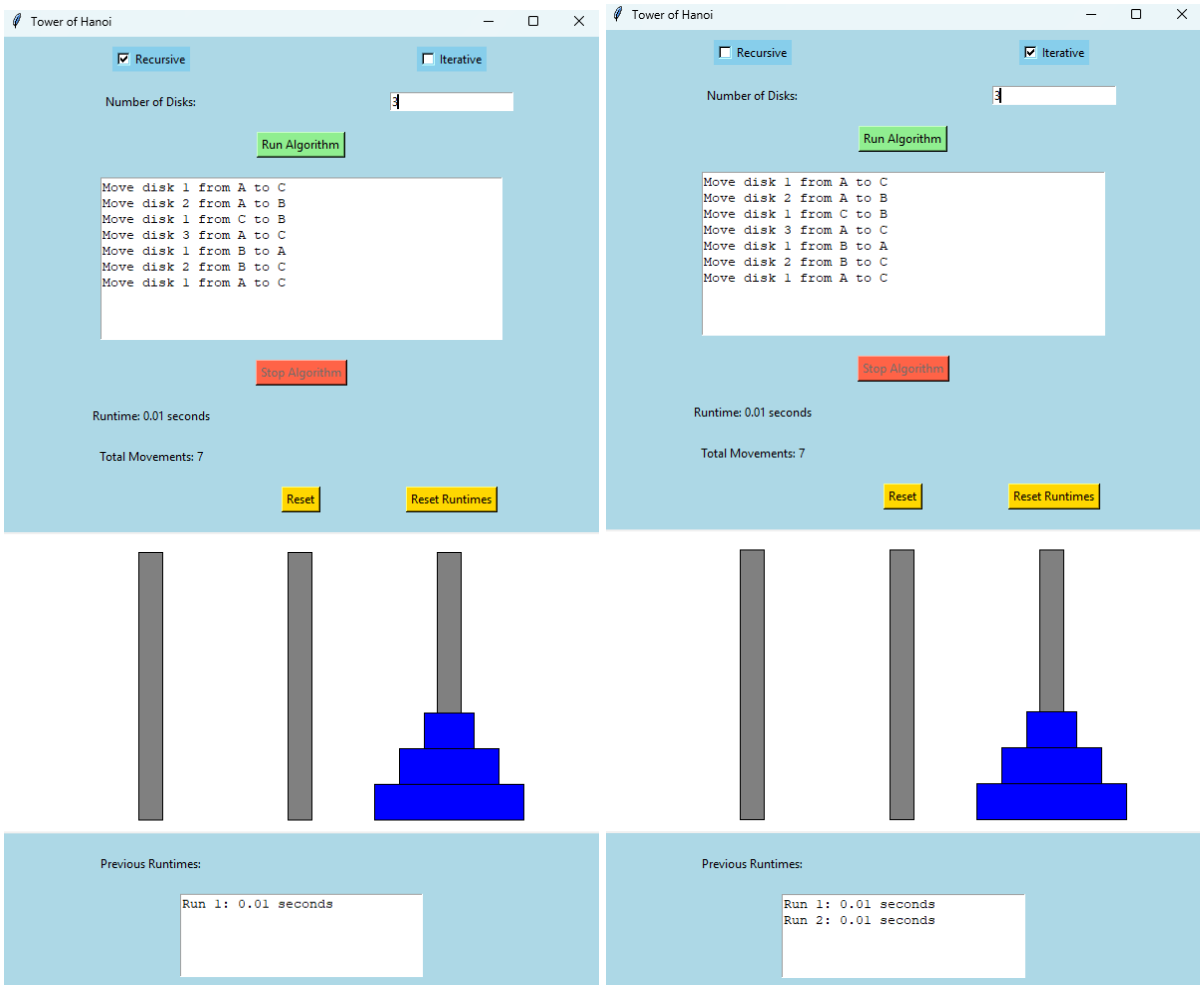
Performance Comparison:

Recursive Approach's time complexity is O(2^n)

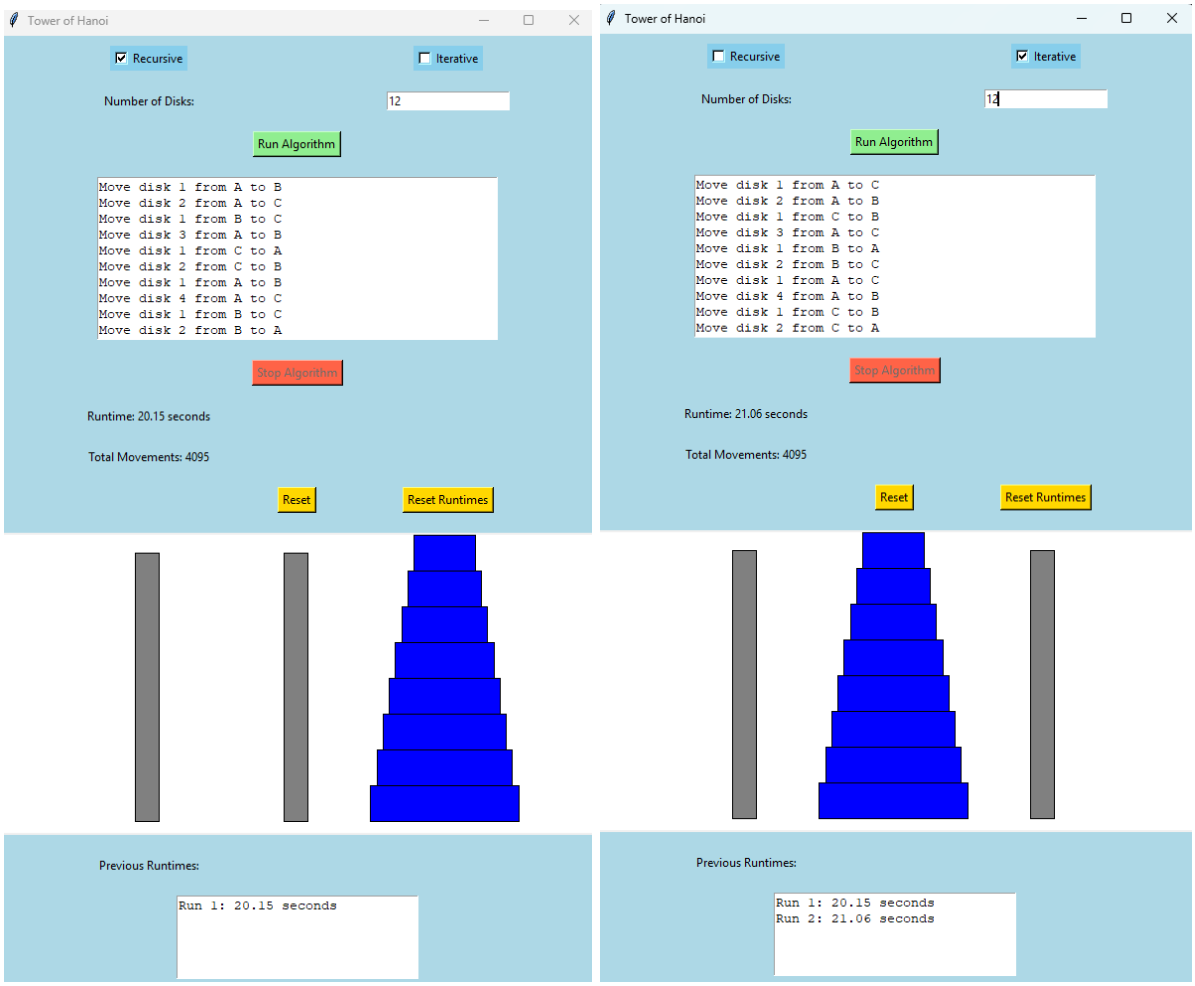
Each recursive call involves two additional recursive calls, leading to an exponential growth in the number of function calls.

Iterative Approach's time complexity is O(2^n).

The iterative approach uses a loop to perform the required moves, and the total number of moves is proportional to the number of disks (n).



6.Recursive and Iterative Runtimes When There are 3 Disks



7.Recursive and Iterative Runtimes When There are 12 Disks

Conclusion of The Performance Comparison:

For small values of n it seems there are no differences between those two approaches. But normally when n is a big number, the iterative approach tends to be the faster choice due to lower overhead(4). Instead of that,in our tries, we saw that most of the times the recursive way is slightly faster when n is a big number. The reason for this might be caused by the programming language/libraries or the user implementation error.

5.Conclusion

In our project, we have successfully implemented both recursive and iterative approaches to successfully solve the Tower of Hanoi problem. We designed a GUI which is capable of showing the desired variables, towers, disks and reset buttons allowing the users to type needed inputs. The application we created, animates every step(total steps are (2^n)-1) of the algorithm's moves and shows all the relevant steps clearly. It also saves and displays the previous runtimes to the user for comparison of runtimes. While coding the algorithm the iterative approach part was harder than the recursive approach implementation.This was due to problems we faced implementing the iterative ordering method in python properly. Even though runtimes of each algorithm can vary due to the system's computational power and workload, the recursive and iterative algorithms have the same time complexity which is O(2^n). So when the algorithm is executed their runtimes will show up very close to each other. All in all, we saw the application of the recursive divide and conquer method in comparison performance to iterative methods in long-running tasks and complex applications such as tower of hanoi problems with a high number of disks.

References

1. Tower of Hanoi (2023) Wikipedia. Available at: https://en.wikipedia.org/wiki/Tower_of_Hanoi#cite_ref-oeis_1-0 (Accessed: 21 December 2023).

2. Shumail Zafar, 2023 Available at: https://www.quora.com/Where-does-tower-of-Hanoi-carry-great-significance

3.Spqr (2020) *Recursion – the towers of Hanoi (iii)*, *The Craft of Coding*. Available at: https://craftofcoding.wordpress.com/2020/06/23/recursion-the-towers-of-hanoi-iii/ (Accessed: 21 December 2023).

4. *Difference between recursion and iteration* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/ (Accessed: 21 December 2023).