**SRM UNIVERSITY, ANDHRA PRADESH**

**Data Structures Lab Project Report**

**Submitted in partial fulfillment for the award of the degree in**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

**Beecha Venkata Naga Hareesh|AP21110010839**

**Under the guidance of**

**Dr. Nitul Dutta**

**PROJECT REPORT**

**1)Towers of Hanoi**

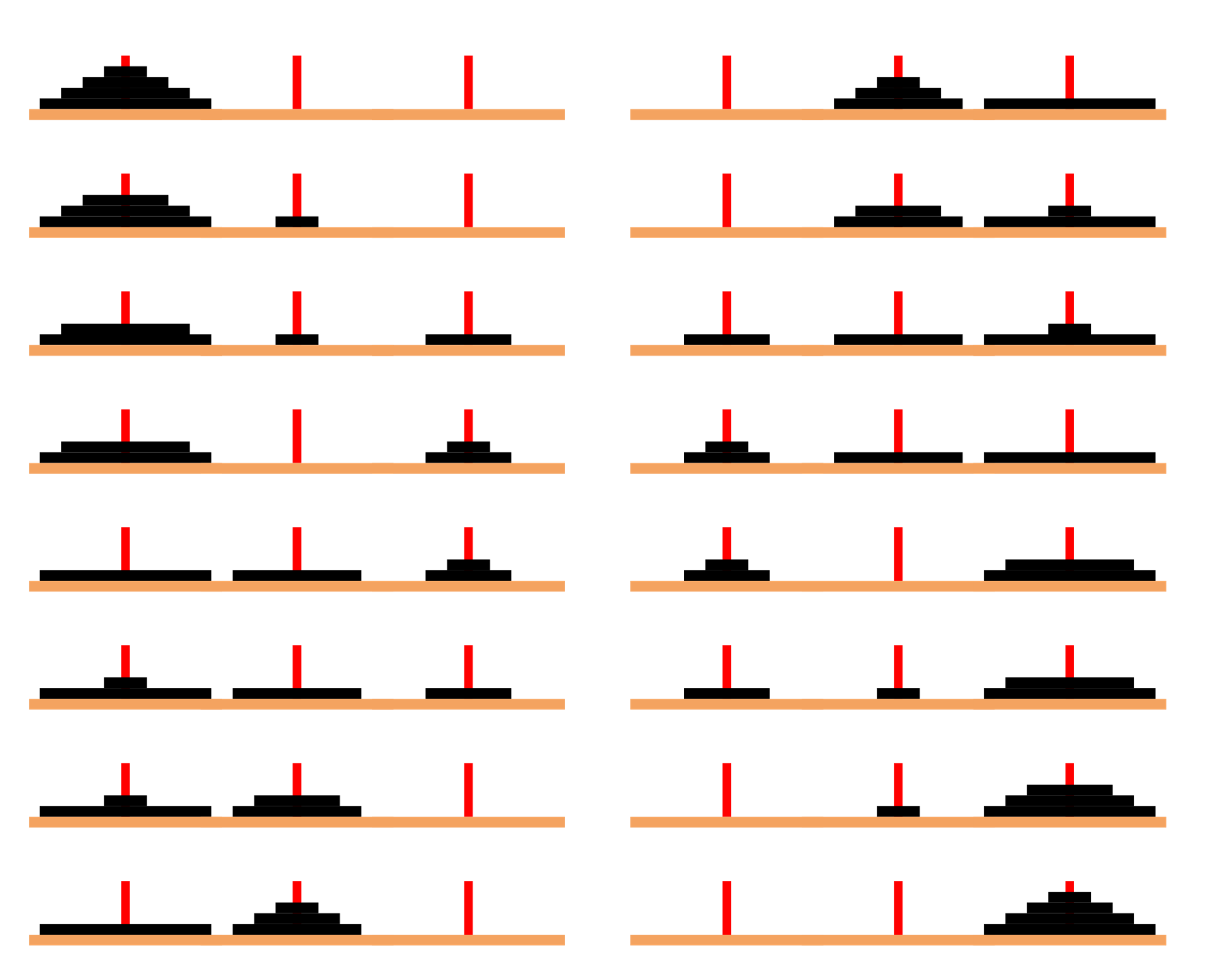
**Description:**

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. the objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1)Only one disk can be moved at a time.

2)Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e., a disk can only be moved if it is the uppermost disk on a stack.

3)No disk may be placed on top of a smaller disk.



**ALGRORITHM**

**STEP 1: - START**

**STEP 2: -** Initialize an integer n representing number of disks.

**STEP 3: -** Create 3 stacks for source, destination, and auxiliary. Similarly, Create 3 variables  s as ‘A’,  d as  ‘B’,  a                as ‘C’.

**STEP 4: -** Check if the number of disks mod 2 is 0, store d in a temporary variable. After that, update d as a              and a as temporary variable.

**STEP 5: -** Firstly, create a variable for the total number of moves and update it as (n\*n) -1.

**STEP 6: -** Traverse from n to 1 and push the current value in source stack.

**STEP 7: -** Traverse from 1 to total moves and check if current index mod 3 is 1, pop the top from source  and         destination stack. If the popped top of source stack is equal to INT\_MIN, push the popped top of destination to source.

**STEP 8: -** Else If the popped top of destination stack is equal to INT\_MIN, push the popped top of source stack to destination stack.

**STEP 9: -** Else If the popped top of source stack is greater than the popped top of destination stack, push both the tops in source stack, else push both in destination stack. After that, print the traversal.

**STEP 10: -** Similarly, check if current index mod 3 is 2, pop the top from source, and auxiliary stack. If the     popped top of source stack is equal to INT\_MIN, push the popped top of the auxiliary stack to     source stack.

**STEP 11: -** Else If the popped top of the auxiliary stack is equal to INT\_MIN, push the popped top of source       stack to auxiliary stack.

**STEP 12: -** Else If the popped top of the auxiliary stack is equal to INT\_MIN, push the popped top of source stack to auxiliary stack.

**STEP 13: -** Similarly, check if current index mod 3 is 0, pop the top from auxiliary and destination stack. If the popped top of the auxiliary stack is equal to INT\_MIN, push the popped top of destination stack to auxiliary stack.

**STEP 14: - STOP**

**Time And Space Complexity.**

Time Complexity:

if we assume n as number of discs Then The time complexity is O(2^n), because that is the number of iterations done in the only loops present in the code, while remaining code run in constant time

Space Complexity:

The space complexity can be split up into two parts:

The "towers" themselves (stacks) have a O(n) space complexity

The auxiliary space has a O(1) space complexity as there are no other vectors, and the call stack has a fixed size (no dynamic recursion) So the overall space complexity is O(n).

**2)** **THE 4-QUEENS PROBLEM**

**Description:**

The N Queen Problem is one of the best problems used to teach backtracking and of course recursion. Backtracking is a general algorithm which finds all complete solutions to a problem by building over partial solutions. In this process, the problem might reach to a partial solution which may not result into a complete solution. Such partial solutions are effectively rejected by the backtracking algorithm.

Shape, square

Description automatically generated

**ALGRORITHM**

**STEP 1: - START**

**STEP 2: -** Place the queens col­umn wise, start from the left most column

**STEP 3: -** If all queens are placed:  
                     return true and print the solu­tion matrix.  
                     Else  
                      Try all the rows in the cur­rent column.

**STEP 4: -**Try all rows in the current column. Perform the following steps:

a) if the queens can be placed safely in this row, then mark the (row, column) as part of the solution and recursively check if the queen is perfectly placed or not.

  b) Place queen in (row, column) if leads to the solution then

return true.

c) if queen placed doesn't lead to solution, then use backtrack and go to step a) to try other probabilities

**STEP 5: -**If all rows have been tried and nothing worked, return false.

**STEP 6: - STOP**

**Time and Space Complexity**

Time Complexity:

The QUEEN method takes O(N) time as it iterates through our array every time.

For each invocation of the place method, there is a loop which runs for O(N) time.

In each iteration of this loop, there is QUEEN invocation which is O(N) and a recursive call with a smaller argument.

If we add all this up and define the run time as T(N). Then T(N) = O(N) + N\*T (N-1)

1). If you draw a recursion tree using this recurrence, the final term will be

something like n^3+n!0(1). By the definition of Big O, this can be reduced to O(n!)

running time.

Space complexity:

For this algorithm it is O(N). The algorithm uses an auxiliary array of length N to store just N position

**COMPARATION BETWEEN THE TWO PROJECTS**

1. STACK is the most preferred data structure for solving the Tower of Hanoi problem as we have to access the top disk of the tower every time for checking with some condition statement or to move the disk from the top of tower to another tower or the other way around, as stack always keep track of top element and follows LIFO (Last In First Out) principle, Due to this stack proves to be the efficient data structure over here.
2. ARRAY is the preferred data structure for solving the 4-queens problem as we have to perform multiple backtracking, ARRAY proves to be the efficient data structure for recursive calls and backtracking process.