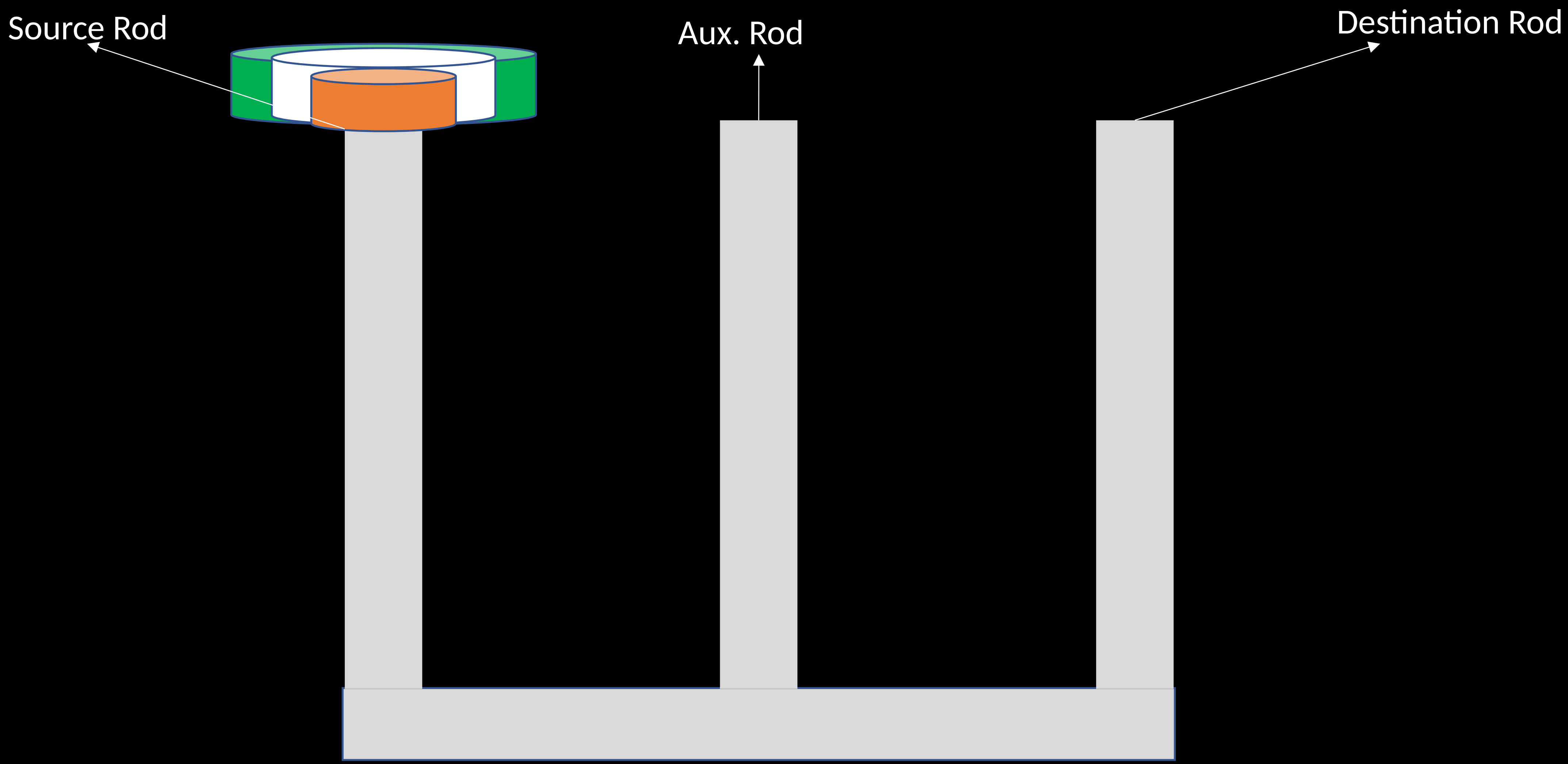
Tower Of Hanoi Algorithm

Name :- Vamsi Krishna Vallabhaneni

NetID :- tw9714

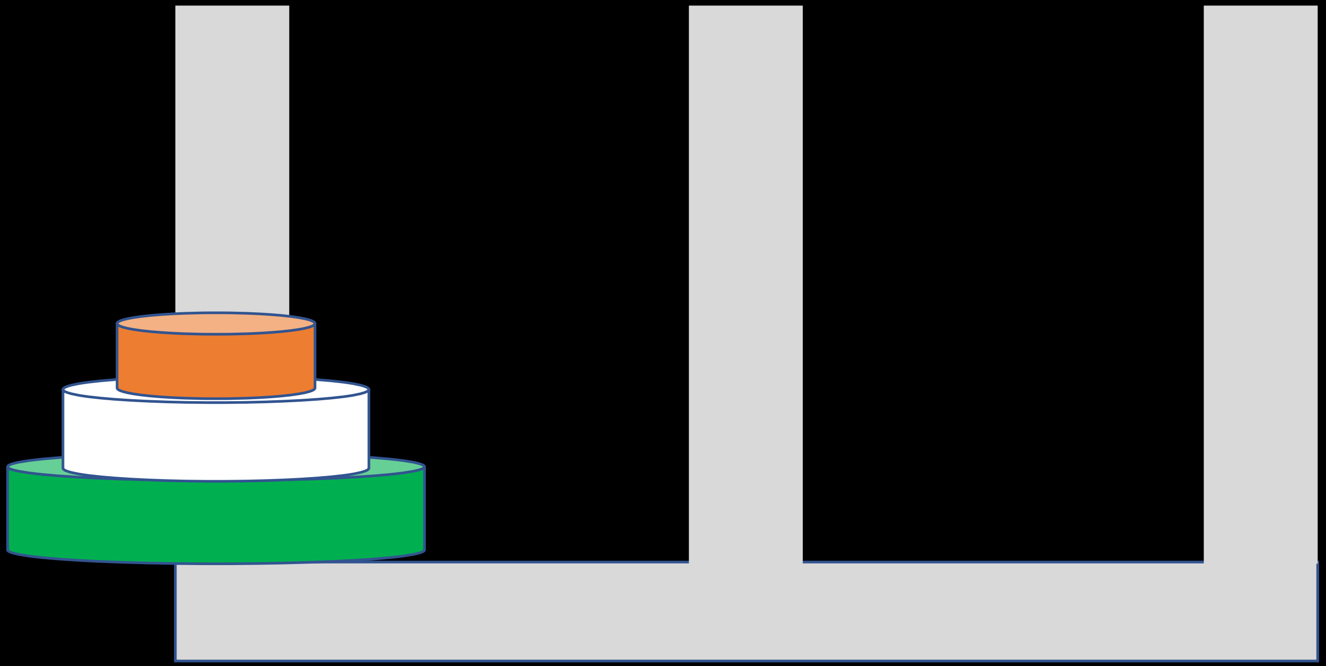
Class :- CS-601-01

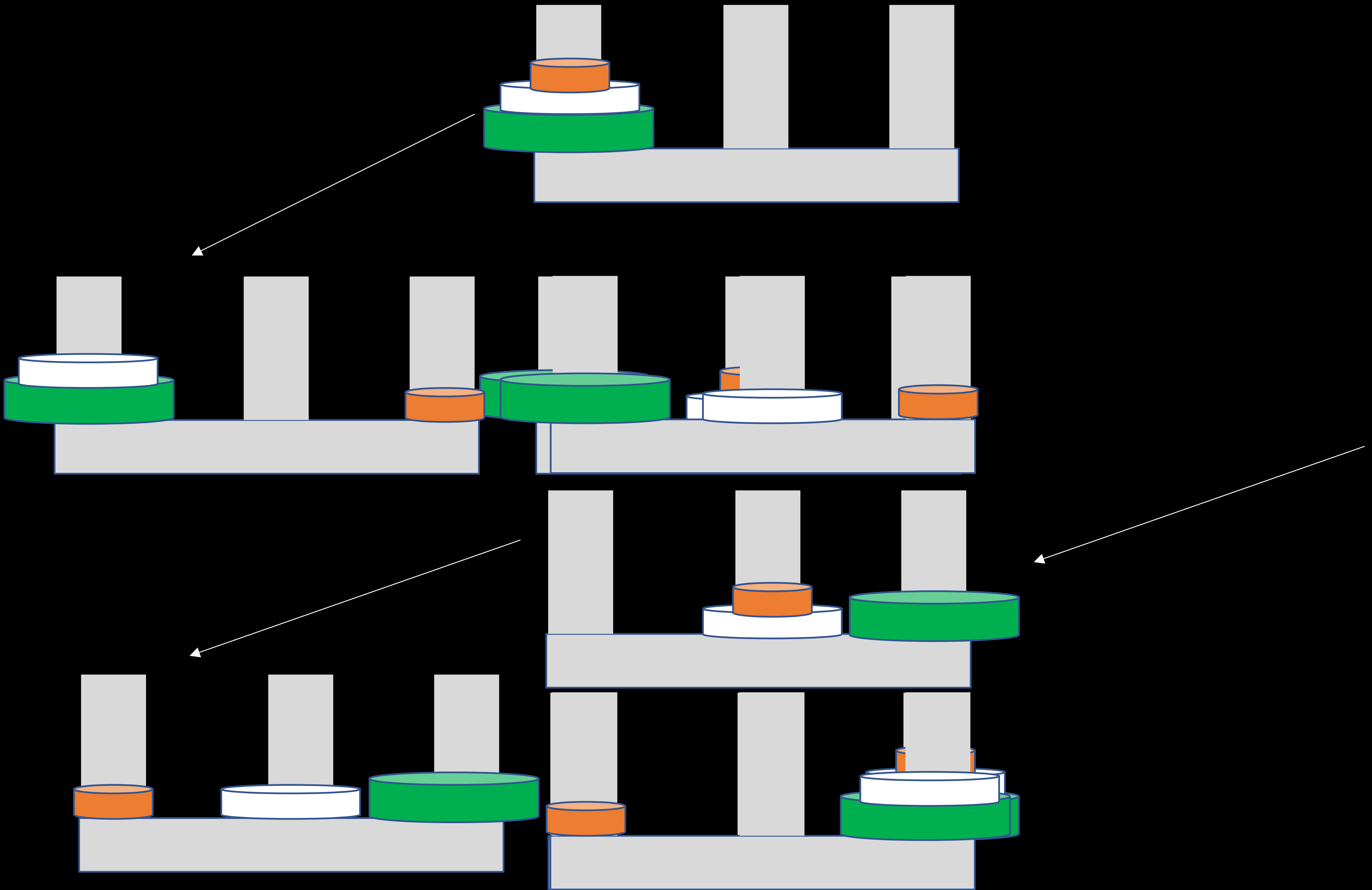


- There are three rods – source rod, destination rod, auxiliary rod
- Our objective is to move the disks from source rod to destination rod, such that they are in the same order
- You can only move 1 disk at a time
- A larger disk cannot be on top of a smaller disk at any point in the process

Strategy for solving a Tower of Hanoi Problem

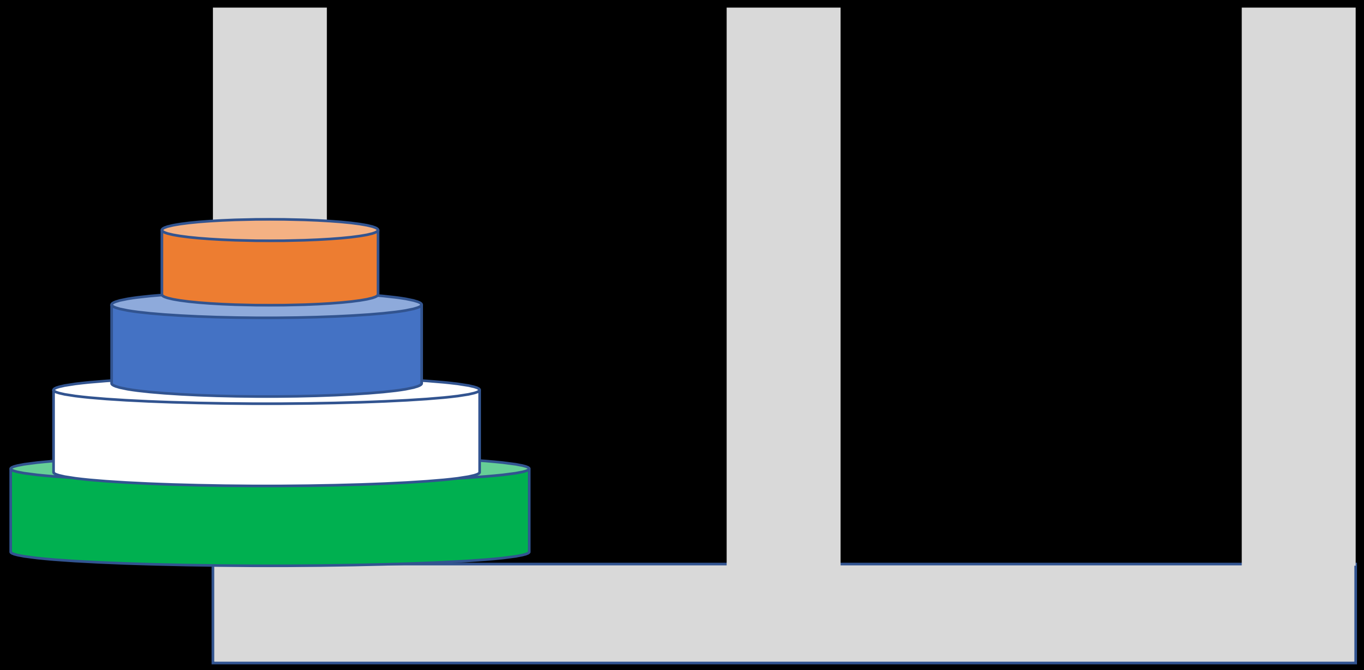
- Move the the top 2 disks from source rod to aux. rod
- Move the largest disk to the destination rod
- Move the disks in aux. rod to destination rod

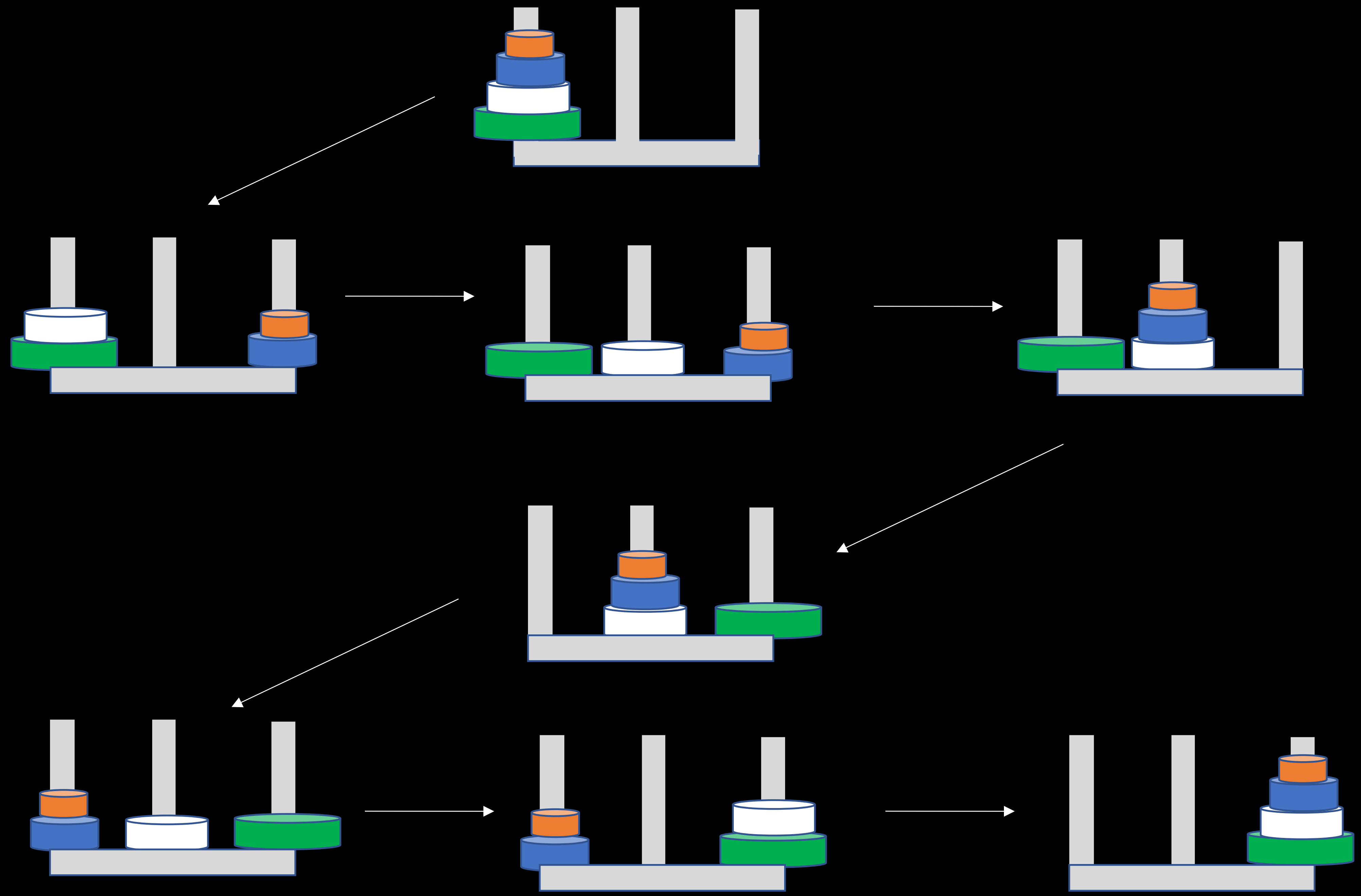




Using the strategy for 4 disks

Move the the top 3 disks from source rod to aux. rod
Move the largest disk to the destination rod
Move the disks in aux. rod to destination rod





Using the strategy for n disks

- Move the the top n-1 disks from source rod to aux. rod
- Move the largest disk to the destination rod
- Move the disks in aux. rod to destination rod

Algorithm for Tower of Hanoi

// hanoi(n, src, dest, aux), n = no. of disks, src = source rod,
dest = destination rod, aux = auxiliary rod

1. start
2. If n=1,
   then move disk from src to dest
3. else,
   call the recursive function hanoi(n-1, src, aux, dest)
   then move disk from src to dest
   call the recursive function hanoi(n-1,aux,dest,src)
4. end

Pseudocode for Tower of Hanoi

The function '***hanoi(n, src, dest, aux)'*** prints the sequence of steps required to move 'n' disks from 'src' to 'dest'.

```
hanoi(n, src, dest, aux)
{
    if(n==0)
    return;
    else
    hanoi(n-1, src, aux)
    print(src, ”->”, dest)
    hanoi(n-1,aux,dest)
}
```

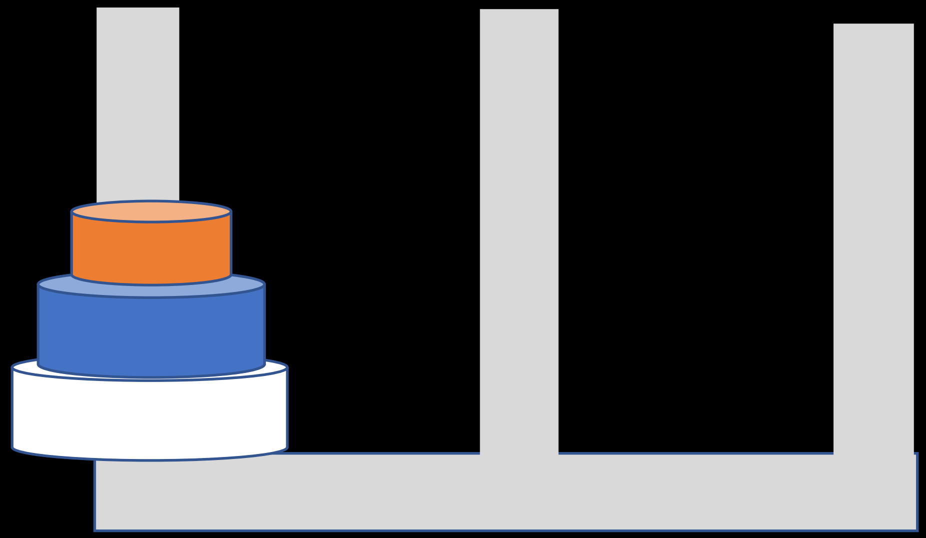
Analyzing the Algorithm for n=3, src=1, dest =3, aux=2

hanoi(3,1,3,2)

hanoi(2,1,2,3)
1->3
hanoi(2,2,3,1)

hanoi(1,1,3,2)
1->2
hanoi(1,3,2,1)
hanoi(1,2,1,3)
2->3
hanoi(1,1,3,2)

1->3
3->2
2->1
1->3



Tower of Hanoi Recurrence

move from src to dest if = 1

hanoi(n,src,dest,aux) =

hanoi(n,src,aux,dest) if n>1

move from src to dest

hanoi(n,aux,dest,src)

Obtaining the Recurrence Relation for Tower of Hanoi

- **Step 1:** Size of problem is *n*
- **Step 2:** Primitive operation is to move the disk from one rod to another rod
- **Step 3:** Every call makes two recursive calls with a problem size of n – 1. And each call corresponds to one primitive operation, so recurrence for this problem can be set up as follows:

  $T(n) = 2T(n - 1) + 1$ **…(1)**

Solving the Recurrence Relation

$T(n) = 2T(n - 1) + 1$ **…(1)**

- Let us solve this recurrence using forward and backward substitution:
- Substitute n by n – 1 in Equation (1),

$T(n - 1) = 2T(n - 2) + 1$,

- By putting this value back in Equation (1),

$T(n) = 2[2T(n - 2) + 1] + 1$

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

$= 2^2T(n - 2) + (2^2 - 1)$ **…(2)**

- Similarly, replace n by n – 2 in Equation (1),

$T(n - 2) = 2T(n - 3) + 1$,

Solving the Recurrence Relation Continuation

- From Equation (2),

  $T(n) = 2^2[\ 2T(n – 3) + 1] + 2 + 1$

  $= 2^3T(n – 3) + 2^2 + 2 + 1$

  $= 2^3T(n – 3) + (2^3 – 1)$

- In general,

  $T(n) = 2^kT(n – k\ ) + (2^k – 1)$

Solving the Recurrence Relation Continuation

- Base condition $n – k = 1$
- $k = n-1$
- By putting $k = n – 1$,

$T(n) = 2^{n–1}[T(1)] + (2^{n-1} – 1)$

- $T(1)$ indicates problem of size 1. To shift 1 disk from source to destination rods take only one move, so $T(1) = 1$.

$$
\begin{aligned}
T(n) &= 2^{n–1} + (2^{n-1} – 1) \\
&= 2^{n–1} + 2^{n-1} – 1 \\
&= 2.2^{n-1} – 1 \\
&= 2^{n-1+1} – 1 \\
&= 2^{n} – 1
\end{aligned}
$$

e.g. For 3 disks it will take $2^3 – 1 = 8 - 1 = 7$ steps

Time Complexity of the Algorithm

Time Complexity of Towers of Hanoi Algorithm is

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

Understanding How the Algorithm Increases Rapidly with Increasing Disks

- Now let us assume that each disk operation takes 1 sec to be executed .
- Then, the time taken for a problem with 3 disks =

  $(2^3 – 1)$ x 1sec = 7 sec
- The time taken for a problem with 16 disks =

  $(2^{16}– 1)$ x 1sec = 65535 sec = 18.2 hours
- The time taken for a problem with 32 disks =

  $(2^{32}– 1)$ x 1sec = 4,294,967,296 sec = 136.2 years

Therefore, we can see how increasing the number of disks gradually increases the time taken to solve the problem at a rapid rate. Hence, we can say that algorithms with a time complexity of 2^n have a high growth rate.