CS-218
DATA STRUCTURE

Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

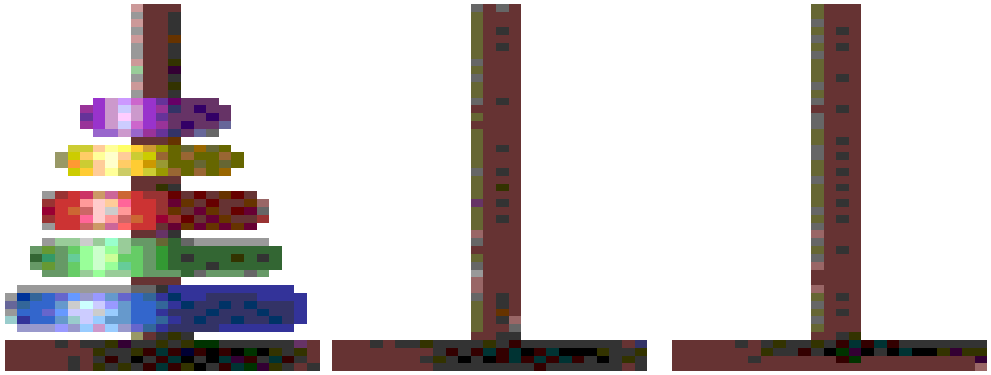
APPLICATION OF STACKS

Application of Stacks

- Tower of Hanoi
- Expressions
  - Infix: A+B-C
  - Postfix: AB+C-
  - Prefix: -+ABC
- Recursion

Tower of Hanoi

- GIVEN: Three poles
  - a set of discs on the first pole,
  - discs of different sizes,
  - the smallest discs at the top
- GOAL: move all the discs from the left pole to the right one.
- CONDITIONS: only one disc may be moved at a time.
  - A disc can be placed either on an empty pole or on top of a larger disc.



The Tower of Hanoi

Discs	Moves
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255

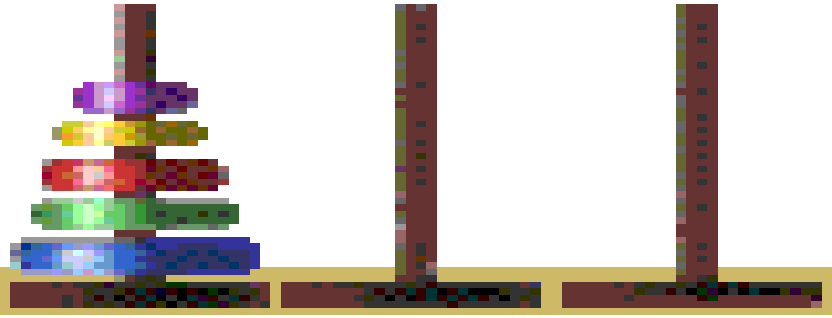
This is called a recursive function.

64	$2^{64} - 1$

n	$2^n - 1$

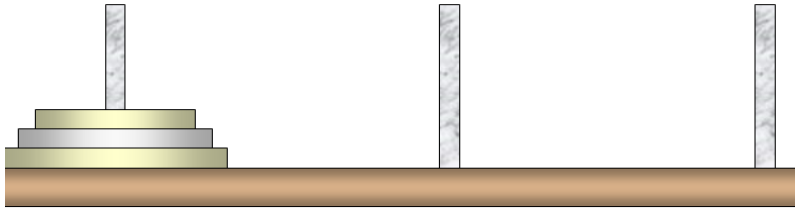
RECURSION

Recursion



- We can solve the **Towers of Hanoi** problem for a stack of discs of height n, by trying to solve it for a stack of height $n-1$.
- To move *n* discs from tower A to tower C, using tower B as the intermediary, the algorithm would look like this:
  - ✓ Move $n-1$ discs from A to B.
  - ✓ Move one disc from A to C.
  - ✓ Move $n-1$ discs from B to C.

Recursion

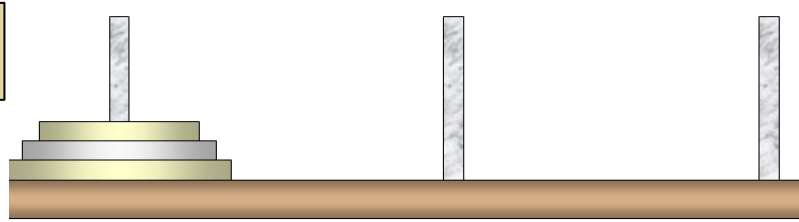



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

If the function above is called as **moveDiscs(3,1,3,2)**, it would move 3 discs from tower 1 (A), to tower 3 (C), using tower 2 (B) as the intermediary.

Recursion

moveDiscs(3,1,3,2)

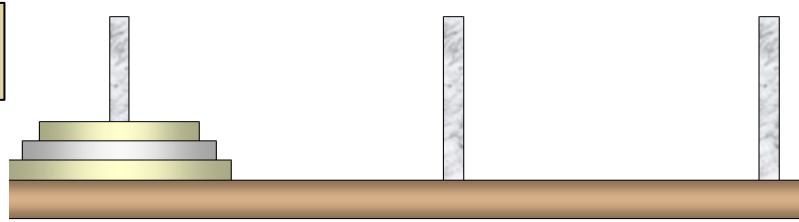



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

```
L1 moveDiscs(3, 1, 3, 2)
  L2 moveDiscs(2, 1, 2, 3)
    L3 moveDiscs(1, 1, 3, 2)
      L4 moveDiscs(0, 1, 2, 3) → STOP
```

Recursion

moveDiscs(3,1,3,2)

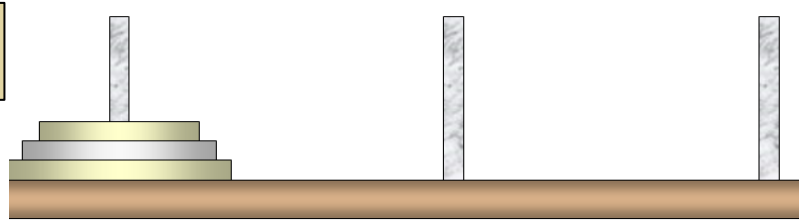



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

```
L1 moveDiscs(3, 1, 3, 2)
  L2 moveDiscs(2, 1, 2, 3)
    L3 moveDiscs(1, 1, 3, 2)
       L4 moveDiscs(0, 1, 2, 3) → STOP
       ❶ move 1 -> 3
       L4 moveDiscs(0, 2, 3, 1) → STOP
    ❷ move 1 -> 2
    L3 moveDiscs(1, 3, 2, 1)
       L4 moveDiscs(0, 3, 1, 2) → STOP
       ❸ move 3 -> 2
       L4 moveDiscs(0, 1, 2, 3) → STOP
    ❹ move 1 -> 3
```

Recursion

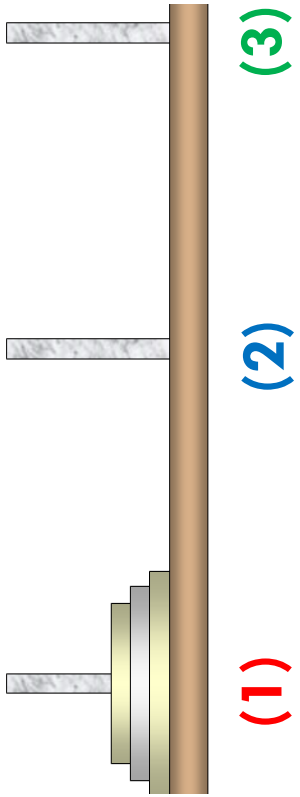
moveDiscs(3,1,3,2)




```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

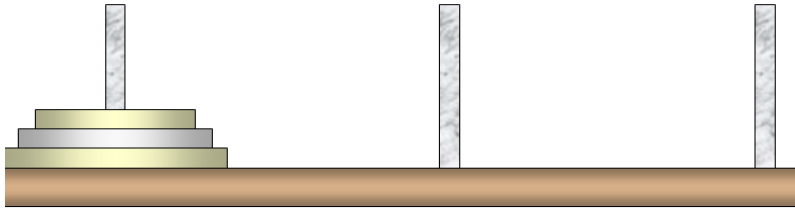
```
    ❹ move 1 -> 3
L2 moveDiscs(2, 2, 3, 1)
  L3 moveDiscs(1, 2, 1, 3)
      L4 moveDiscs(0, 2, 3, 1) → STOP
      ❺ move 2 -> 1
      L4 moveDiscs(0, 3, 1, 2) → STOP
  ❻ move 2 -> 3
  L3 moveDiscs(1, 1, 3, 2)
      L4 moveDiscs(0, 1, 2, 3) → STOP
      ❼ move 1 -> 3
      L4 moveDiscs(0, 2, 3, 1) → STOP
```

```
L1 moveDiscs(3, 1, 3, 2)
   L2 moveDiscs(2, 1, 2, 3)
      L3 moveDiscs(1, 1, 3, 2)
         L4 moveDiscs(0, 1, 2, 3) → STOP
         ❶ move 1 -> 3
         L4 moveDiscs(0, 2, 3, 1) → STOP
      ❷ move 1 -> 2
      L3 moveDiscs(1, 3, 2, 1)
         L4 moveDiscs(0, 3, 1, 2) → STOP
         ❸ move 3 -> 2
         L4 moveDiscs(0, 1, 2, 3) → STOP
      ❹ move 1 -> 3
   L2 moveDiscs(2, 2, 3, 1)
      L3 moveDiscs(1, 2, 1, 3)
         L4 moveDiscs(0, 2, 3, 1) → STOP
         ❺ move 2 -> 1
         L4 moveDiscs(0, 3, 1, 2) → STOP
      ❻ move 2 -> 3
      L3 moveDiscs(1, 1, 3, 2)
         L4 moveDiscs(0, 1, 2, 3) → STOP
         ❼ move 1 -> 3
         L4 moveDiscs(0, 2, 3, 1) → STOP
```



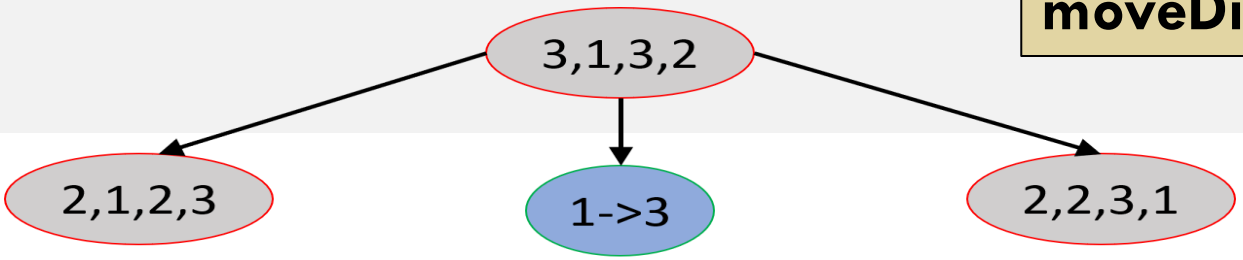
RECURSION TREE

Recursion Tree

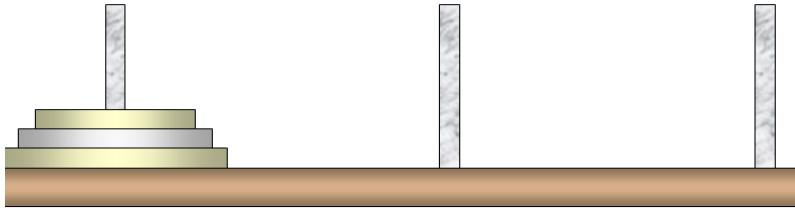



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

moveDiscs(3,1,3,2)

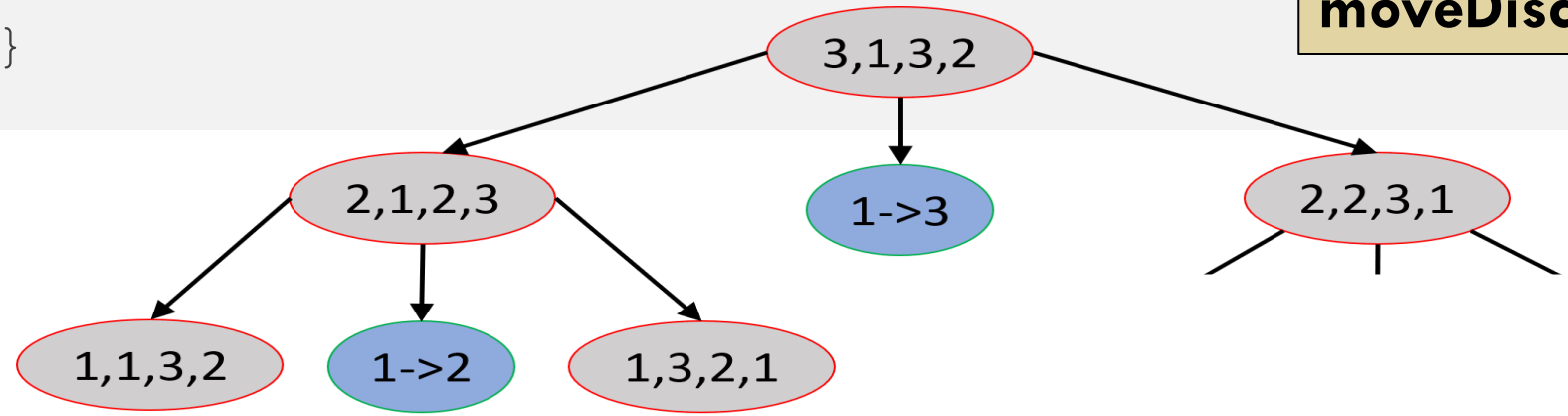


Recursion Tree

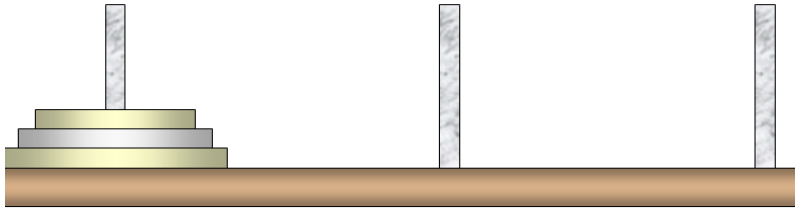



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

moveDiscs(3,1,3,2)

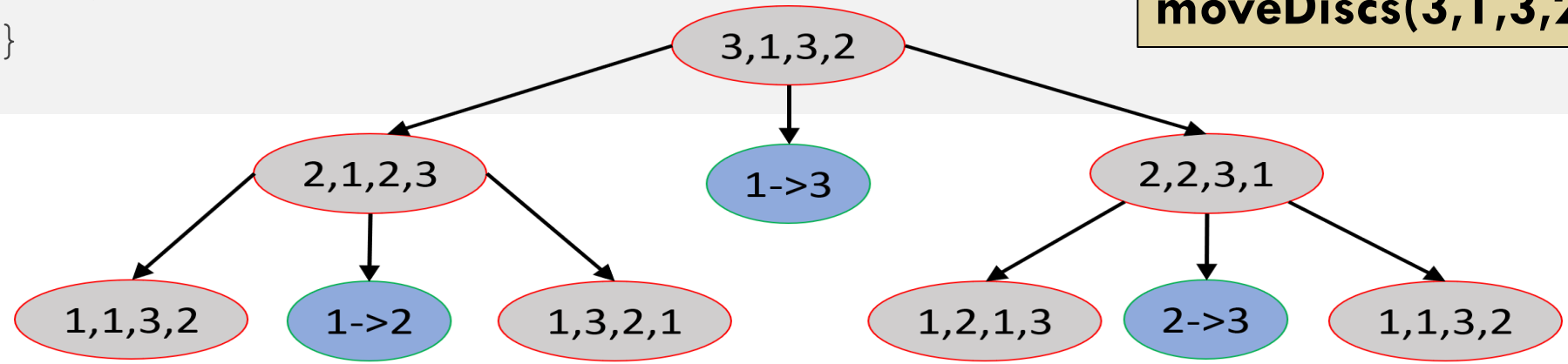


Recursion Tree

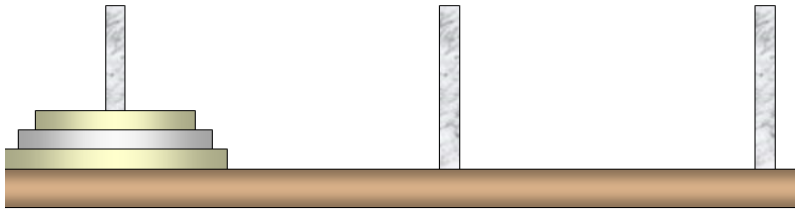



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

moveDiscs(3,1,3,2)

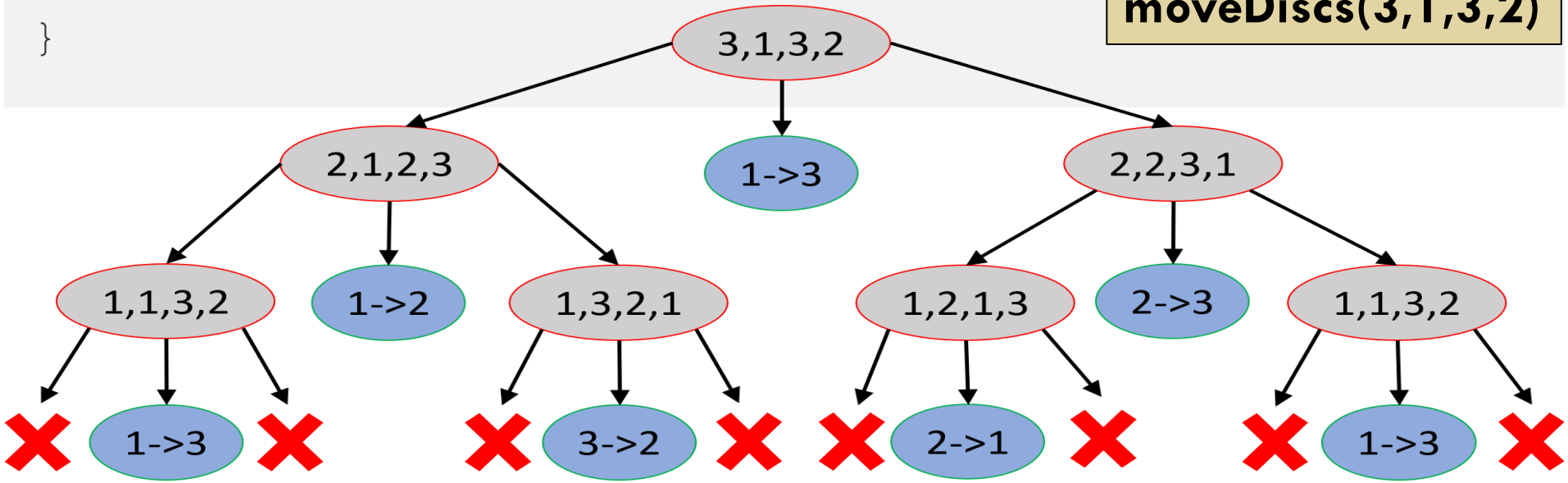


Recursion Tree

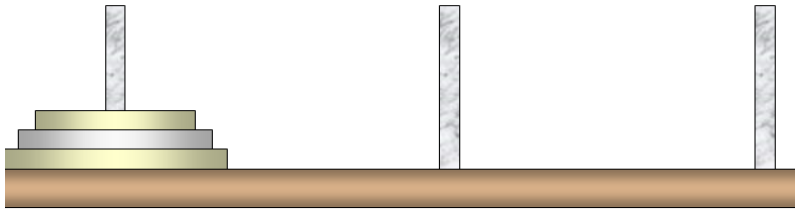



```
void moveDiscs(int N, int from, int to, int using){
    if (N > 0) {
        moveDiscs(N-1, from, using, to);
        cout << "move " << from << " -> " << to << endl;
        moveDiscs(N-1, using, to, from);
    }
}
```

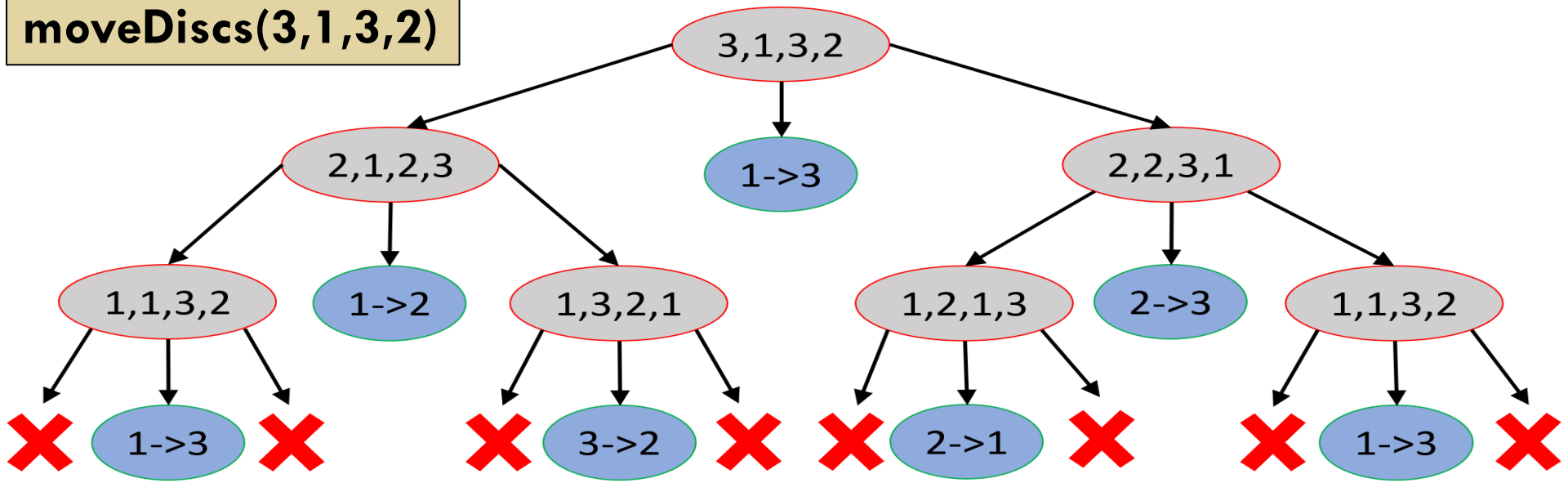
moveDiscs(3,1,3,2)



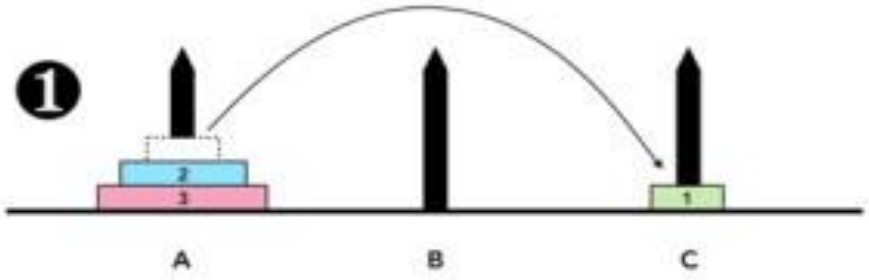
Recursion Tree

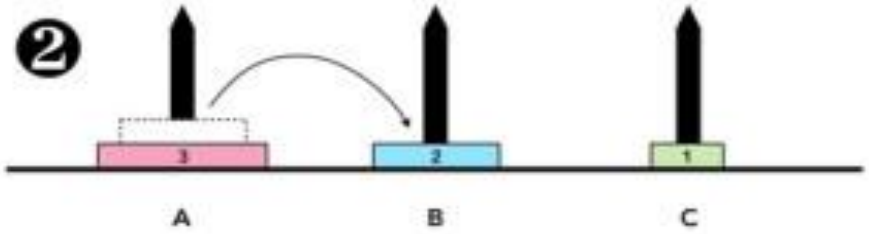


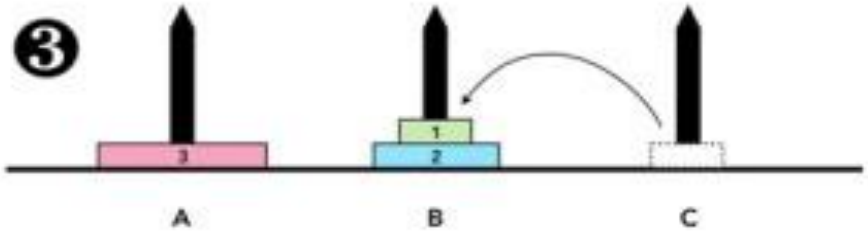
moveDiscs(3,1,3,2)

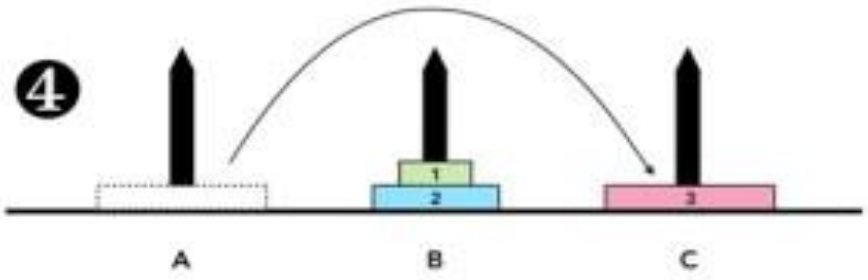


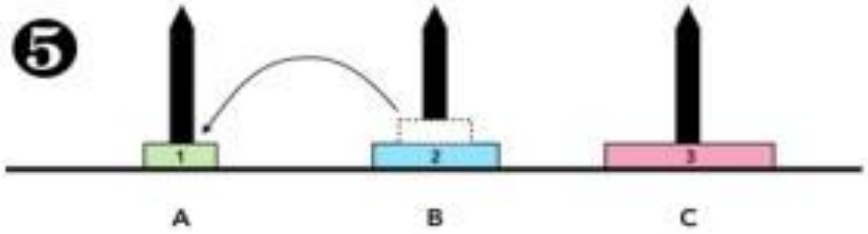
(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

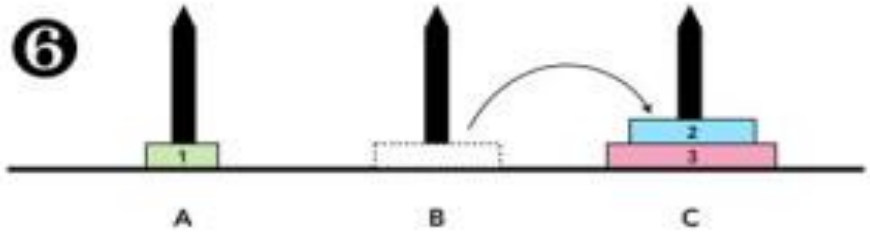


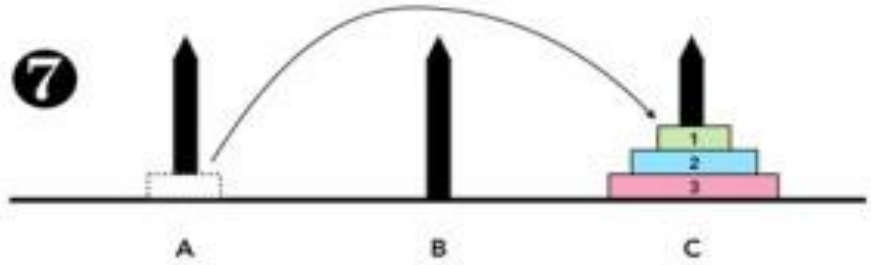






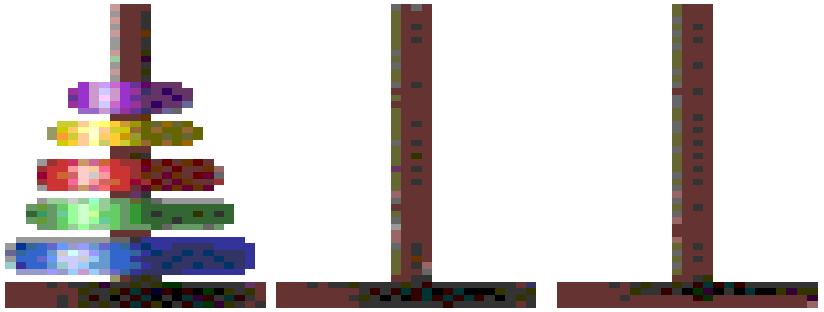






(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

Tower of Hanoi: Recursive Solution



```
void hanoi (int discs,
                Stack fromPole,
                Stack toPole,
                Stack aux) {
Disc d;
if( discs > 0) {
        hanoi(discs-1, fromPole, aux, toPole);
        d = fromPole.pop();
        toPole.push(d);
        hanoi(discs-1,aux, toPole, fromPole);
}
```

Reading Materials

- Nell Dale – Chapter#4
- Schaum's Outlines – Chapter#6
- D. S. Malik – Chapter#7