



CS-2001

DATA STRUCTURE

Dr. Hashim Yasin

**National University of Computer
and Emerging Sciences,
Faisalabad, Pakistan.**

APPLICATION OF STACKS

Application of Stacks

3

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

Tower of Hanoi

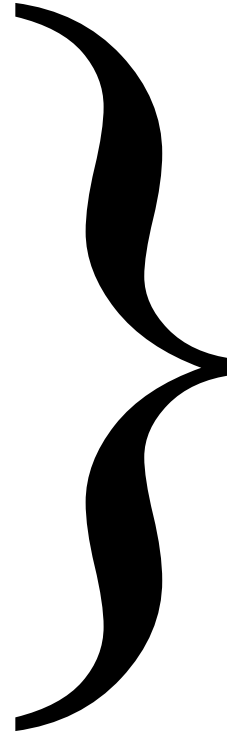
4

- 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.

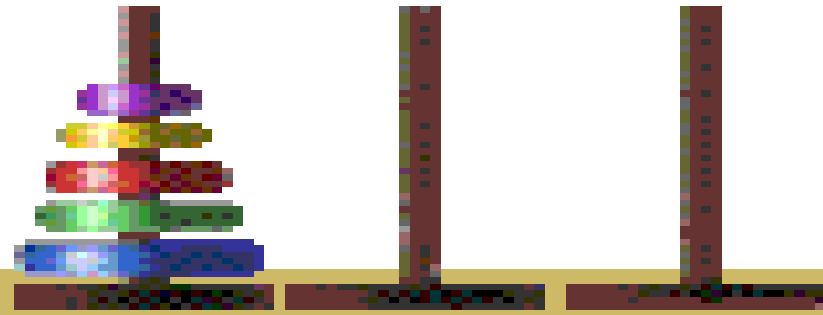
| | |
|----|--------------|
| 64 | $2^{64} - 1$ |
|----|--------------|

| | |
|-----|-----------|
| n | $2^n - 1$ |
|-----|-----------|

RECURSION

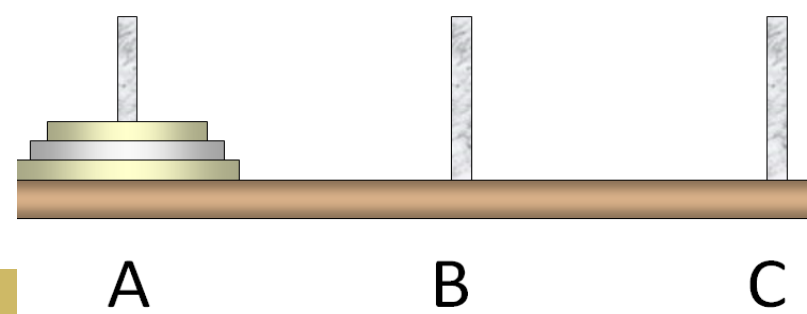
Recursion

7



- 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



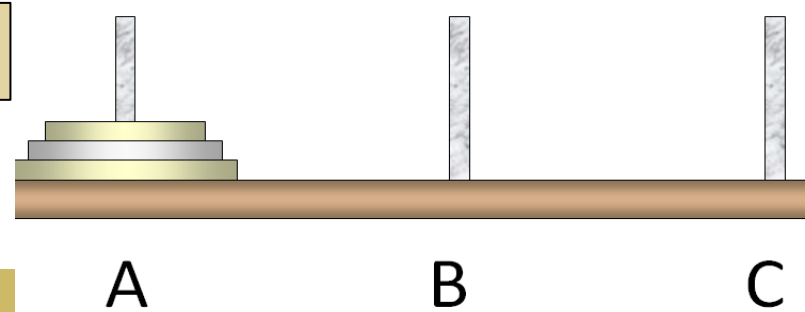
8

```
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.

moveDiscs(3,1,3,2)

Recursion



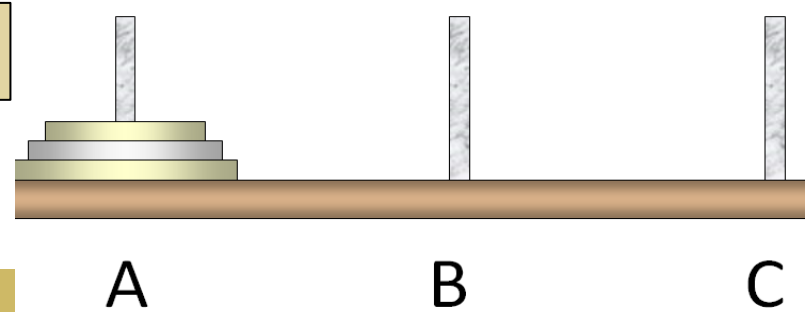
9

```
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)

moveDiscs(3,1,3,2)

Recursion



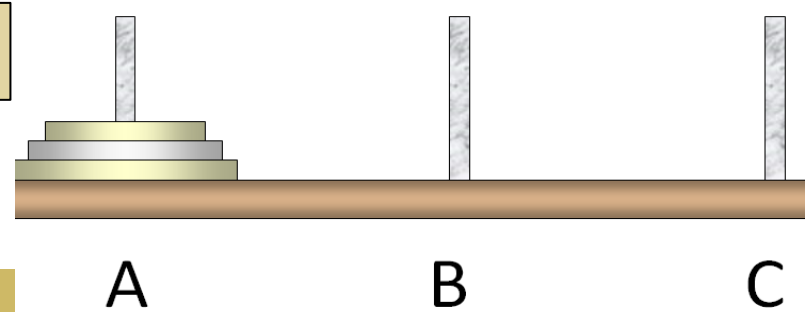
10

```
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)
```

moveDiscs(3,1,3,2)

Recursion



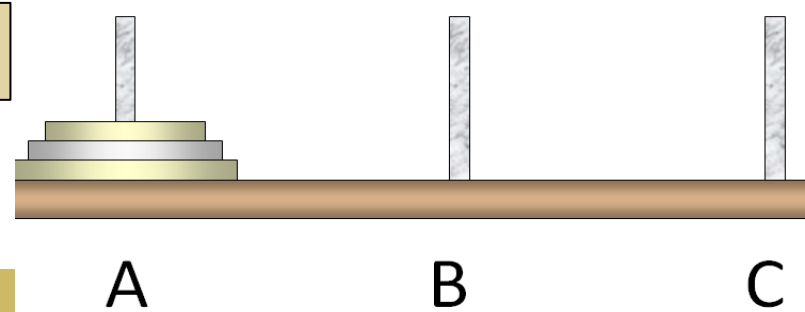
11

```
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)
```

moveDiscs(3,1,3,2)

Recursion



12

```
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)

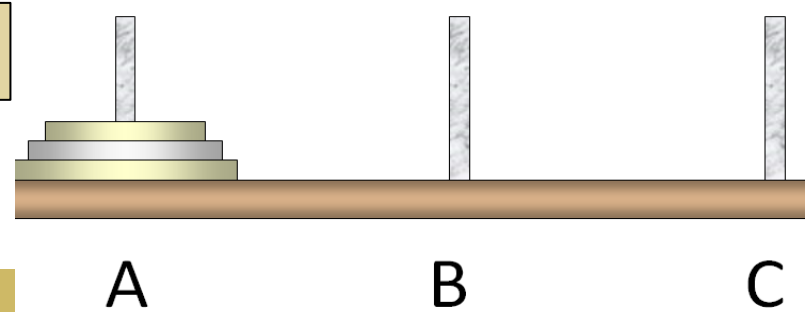
L2 moveDiscs(2, 1, 2, 3)

L3 moveDiscs(1, 1, 3, 2)

L4 moveDiscs(0, 1, 2, 3) → 

moveDiscs(3,1,3,2)

Recursion



13

```
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)

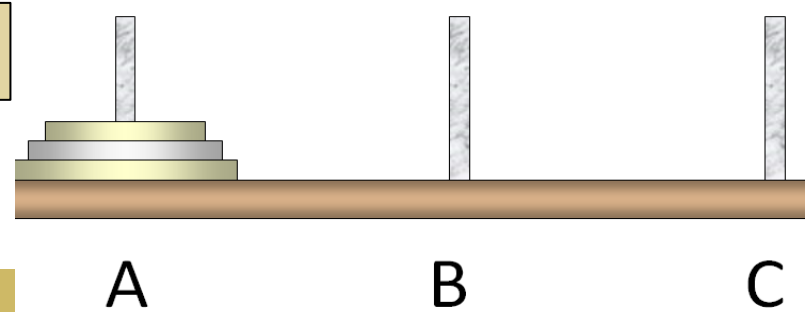
L4 moveDiscs(0, 1, 2, 3) → 

① **move 1 -> 3**

L4 moveDiscs(0, 2, 3, 1) → 

moveDiscs(3,1,3,2)

Recursion



14

```
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

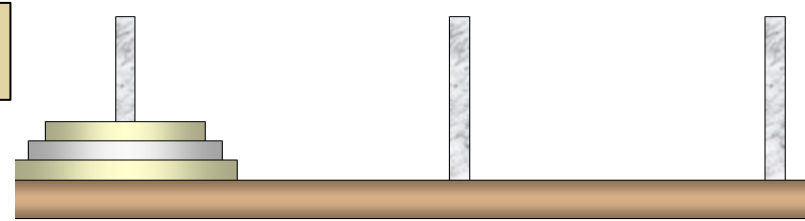
① move 1 -> 3

L4 moveDiscs(0, 2, 3, 1) → STOP

② move 1 -> 2

moveDiscs(3,1,3,2)

Recursion



15

```
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) →



① move 1 -> 3

L4 moveDiscs(0, 2, 3, 1) →

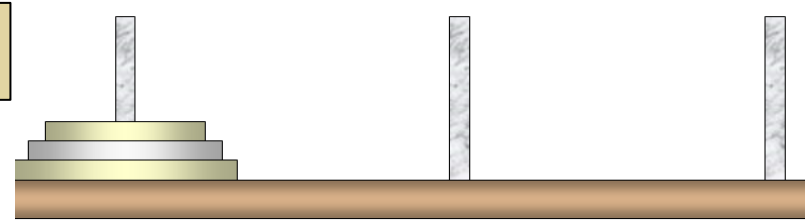


② move 1 -> 2

L3 moveDiscs(1, 3, 2, 1)

moveDiscs(3,1,3,2)

Recursion



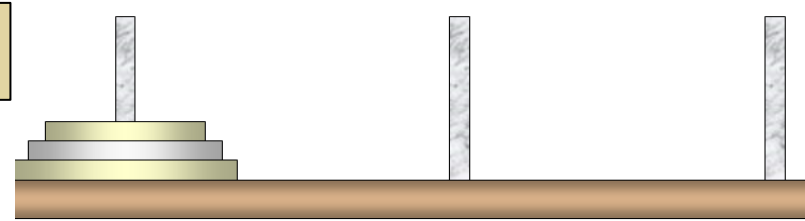
16

```
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  
      ① move 1 -> 3  
      L4 moveDiscs(0, 2, 3, 1) → STOP  
    ② move 1 -> 2  
    L3 moveDiscs(1, 3, 2, 1)  
      L4 moveDiscs(0, 3, 1, 2) → STOP
```


moveDiscs(3,1,3,2)

Recursion



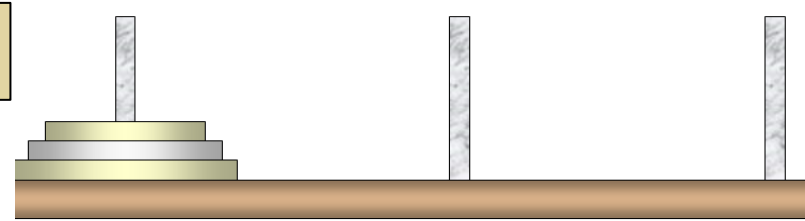
17

```
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  
      ① move 1 -> 3  
      L4 moveDiscs(0, 2, 3, 1) → STOP  
    ② move 1 -> 2  
    L3 moveDiscs(1, 3, 2, 1)  
      L4 moveDiscs(0, 3, 1, 2) → STOP  
      ③ move 3 -> 2  
      L4 moveDiscs(0, 1, 2, 3) → STOP
```

moveDiscs(3,1,3,2)

Recursion



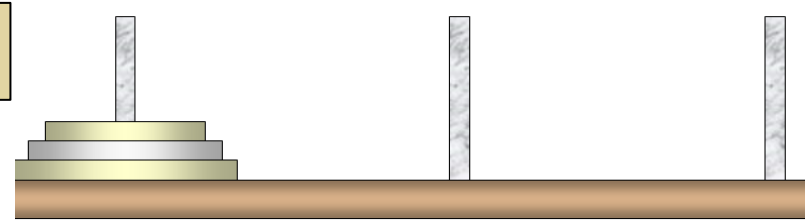
18

```
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  
      ① move 1 -> 3  
      L4 moveDiscs(0, 2, 3, 1) → STOP  
    ② move 1 -> 2  
    L3 moveDiscs(1, 3, 2, 1)  
      L4 moveDiscs(0, 3, 1, 2) → STOP  
      ③ move 3 -> 2  
      L4 moveDiscs(0, 1, 2, 3) → STOP  
    ④ move 1 -> 3
```

moveDiscs(3,1,3,2)

Recursion



19

```
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);  
    }  
}
```

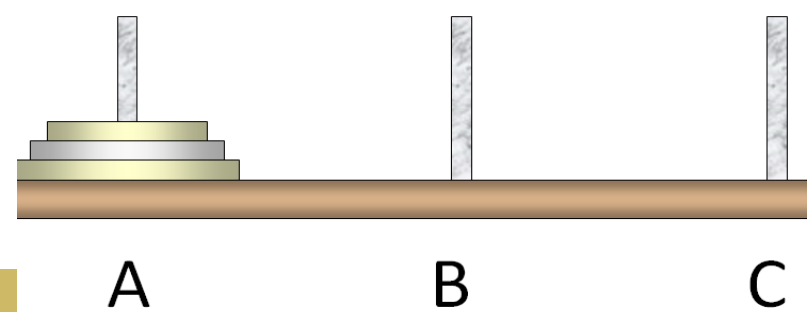
```
④ move 1 -> 3  
L2 moveDiscs(2, 2, 3, 1)  
L3 moveDiscs(1, 2, 1, 3)  
L4 moveDiscs(0, 2, 3, 1) → STOP  
⑤ move 2 -> 1  
L4 moveDiscs(0, 3, 1, 2) → STOP  
⑥ move 2 -> 3  
L3 moveDiscs(1, 1, 3, 2)  
L4 moveDiscs(0, 1, 2, 3) → STOP  
⑦ move 1 -> 3  
L4 moveDiscs(0, 2, 3, 1) → STOP
```

```
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
      ① move 1 -> 3
      L4 moveDiscs(0, 2, 3, 1) → STOP
    ② move 1 -> 2
    L3 moveDiscs(1, 3, 2, 1)
      L4 moveDiscs(0, 3, 1, 2) → STOP
      ③ move 3 -> 2
      L4 moveDiscs(0, 1, 2, 3) → STOP
    ④ move 1 -> 3
  L2 moveDiscs(2, 2, 3, 1)
    L3 moveDiscs(1, 2, 1, 3)
      L4 moveDiscs(0, 2, 3, 1) → STOP
      ⑤ move 2 -> 1
      L4 moveDiscs(0, 3, 1, 2) → STOP
    ⑥ move 2 -> 3
    L3 moveDiscs(1, 1, 3, 2)
      L4 moveDiscs(0, 1, 2, 3) → STOP
      ⑦ move 1 -> 3
      L4 moveDiscs(0, 2, 3, 1) → STOP
```

RECURSION TREE

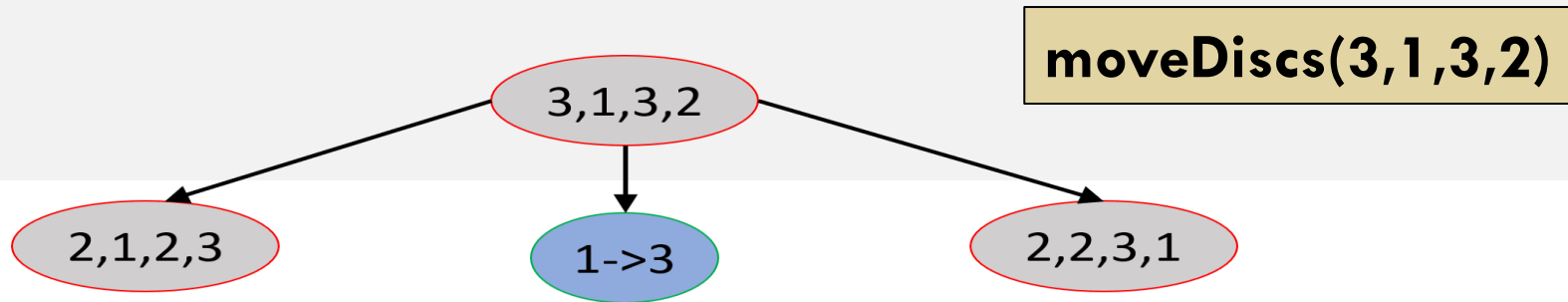


Recursion Tree

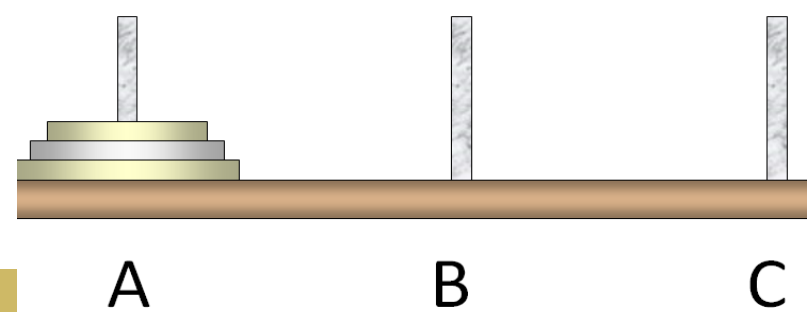


22

```
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);  
    }  
}
```



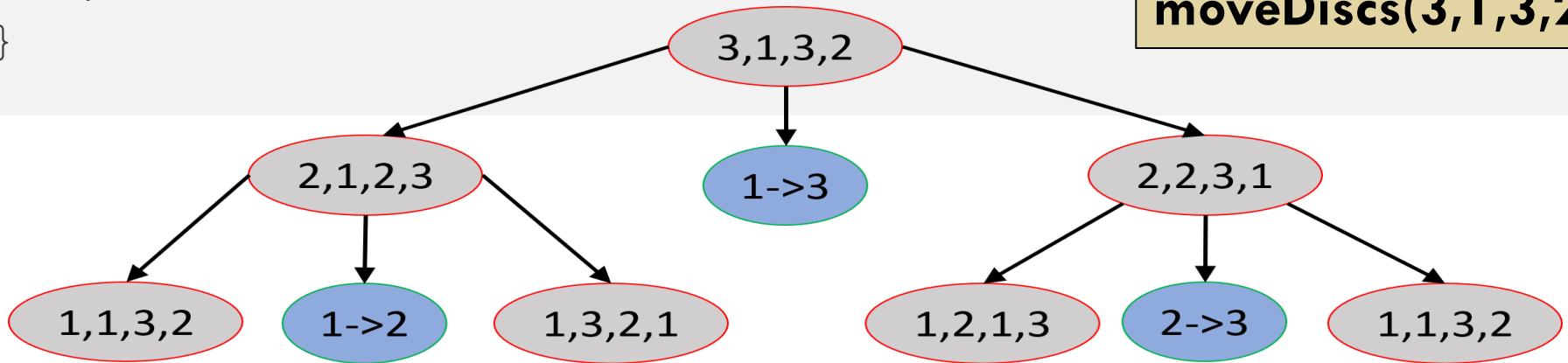
Recursion Tree



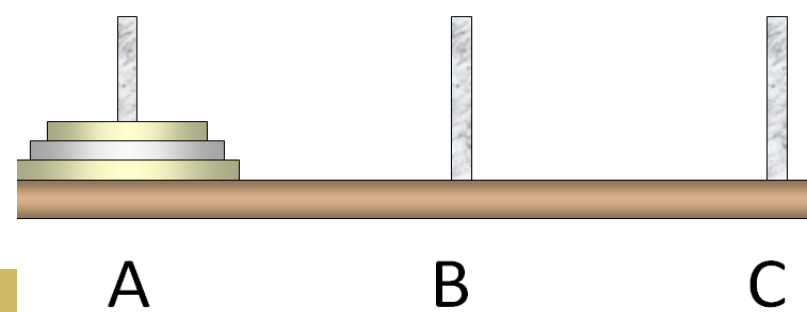
23

```
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)



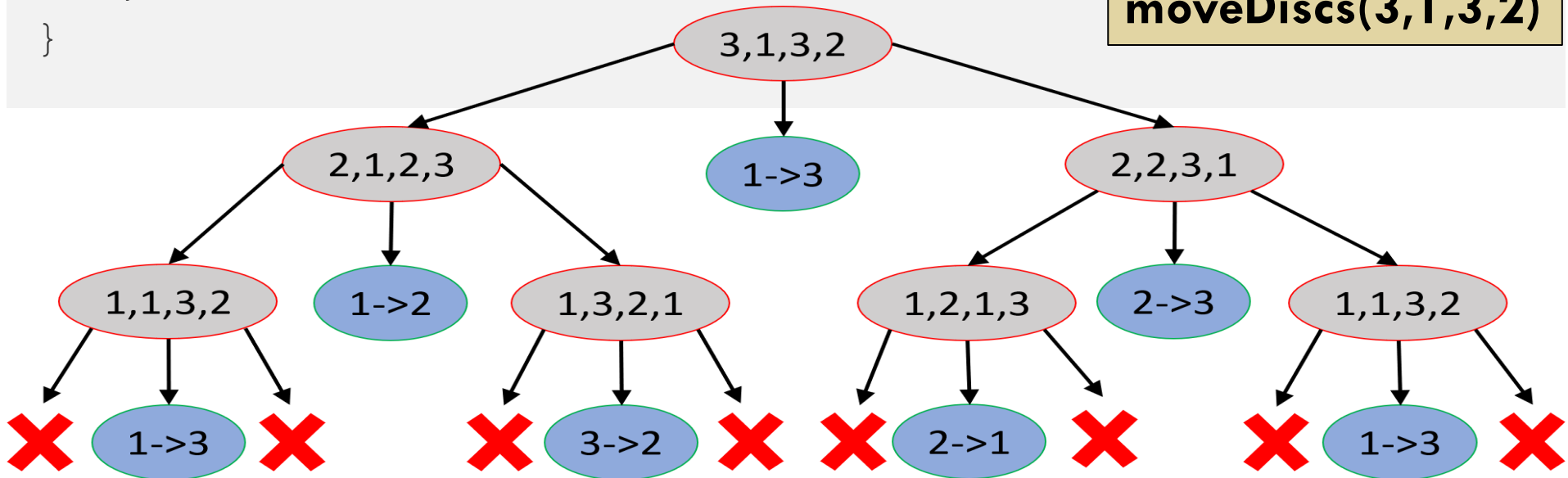
Recursion Tree



24

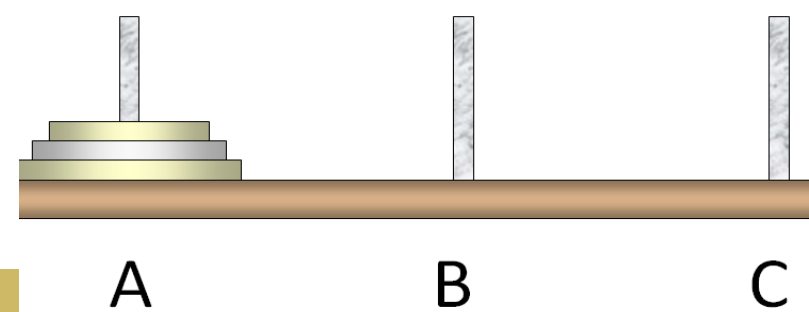
```
void moveDiscs(int N, int from, int to, int using) {  
    if (N > 0) {  
        moveDiscs(N-1, from, using, to);  
        cout << "move " << from << " -> " << to << endl;  
        moveDiscs(N-1, using, to, from);  
    }  
}
```

moveDiscs(3,1,3,2)



Recursion Tree

25

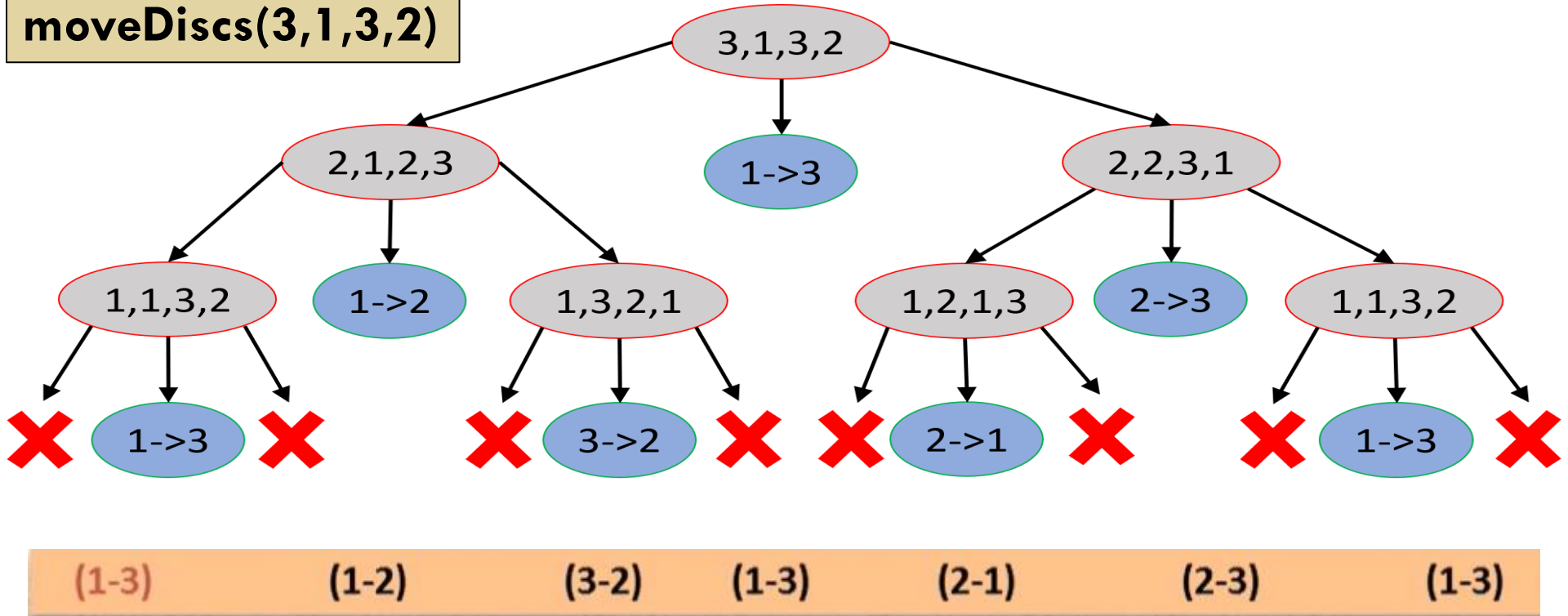


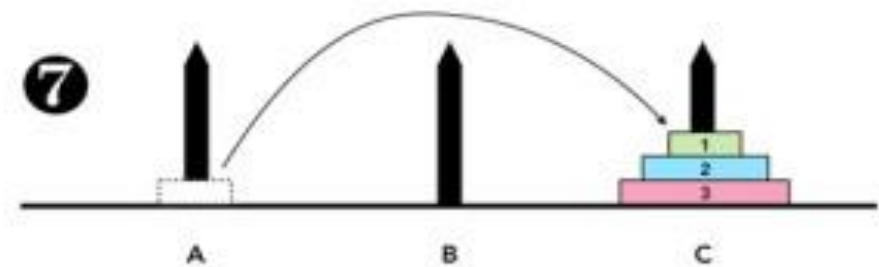
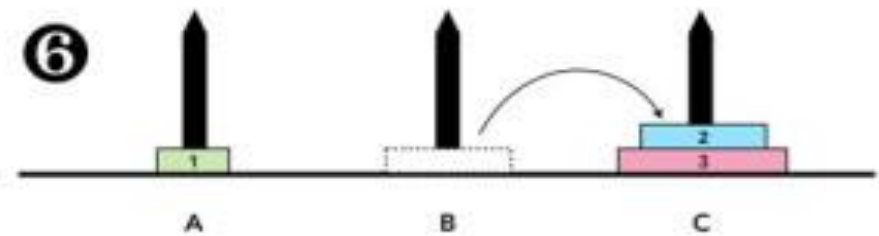
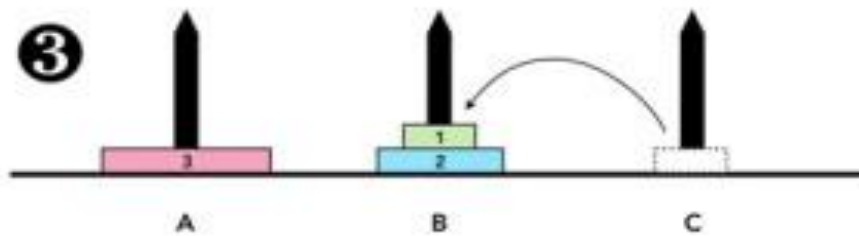
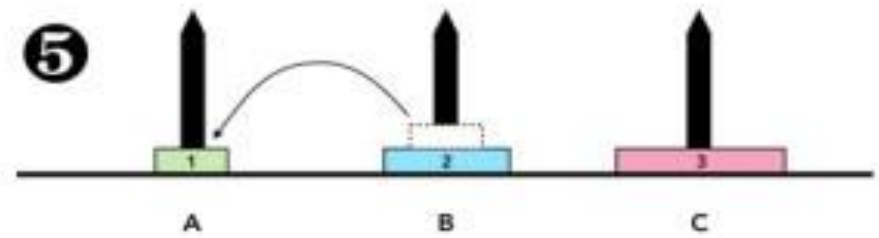
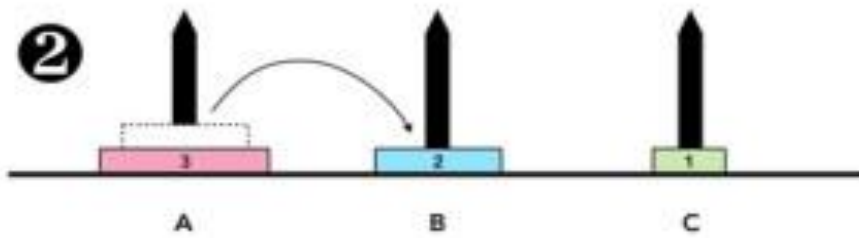
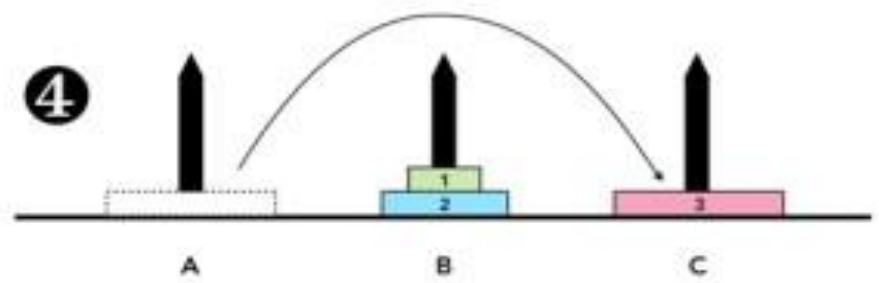
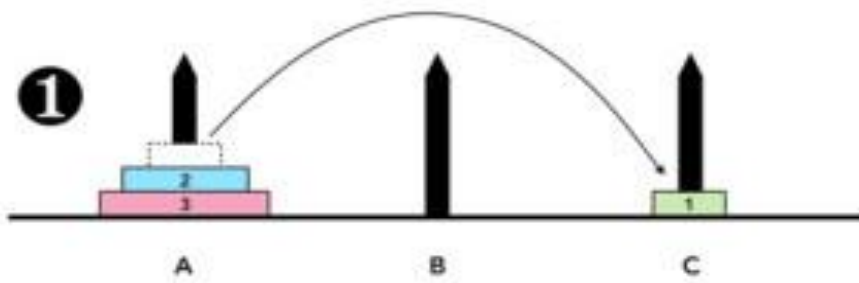
A

B

C

moveDiscs(3,1,3,2)





(1-3)

Dr Hashim Yasin (1-2)

(3-2)

CS-2001 Data Structure (1-3)

(2-1)

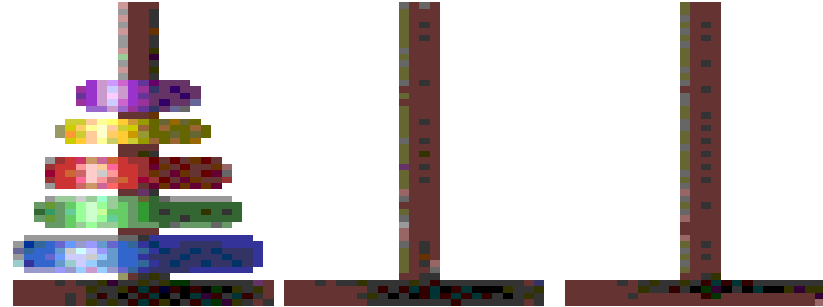
(2-3)

(1-3)

Tower of Hanoi: Recursive Solution

27

```
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

28

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