Algorithm Design and Analysis LAB Introduction

YAO ZHAO, WWY

Lab tutor & SAs

Lab teacher: wwy wangwy@mail.sustech.edu.cn

SA:

Tan Yajing (Wed12)	12332416@mail.sustech.edu.cn
Sun Kaixuan (Wed56)	12432697@mail.sustech.edu.cn
Zhou Xingyu (Thu34)	12432730@mail.sustech.edu.cn

Office hour: Thu 13:00~15:00, Teaching Building 3, 505

Knowledge, Ability and Skill Requirements

- Algorithmic Learning : Proficiency in the presentation, solution and proof of algorithmic problems described in textbook.
- Algorithmic Description: Ability to describe the process of algorithms using flow chart, pseudocode or other methods.
- Algorithmic Implementation: Ability to Implement Algorithms.
- Algorithmic Design and Analysis: Ability to design algorithms for given problems and prove their correctness, analyze their time and space complexity

Content

- Explanation of key knowledge points
- Do some exercises
- Solutions of lab assignments and exercises or other extended questions
- Collect and explain the most concentrated problems
- Experience sharing

Requirements Of Theory Assignments

- Mr. Shi will assign theory assignments at irregular intervals. Please submit your theory assignments on time.
- Theory assignments will be posted on **BB worksite:**

  Algorithm Design and Analysis SE1&2 Spring 2025.
- You can write assignments by hand, in word, or markdown, but submit a pdf file.
- **Only one DDL. No late submission is allowed. (Submissions later than DDL will not be accepted.). If exceed the DDL, the score of the assignment will be 0.**
- Please **submit your assignment in advance** so as not to submit fail or exceed DDL.
- **Don't copy**, you **can't copy online reference answers**, you **can't copy other students' homework**.
- **Assignment:20%**

Requirements Of Lab Assignments

- Due to the free withdrawal of the first three weeks, the DDL of all lab and theory assignments in the first two weeks will start from the fourth week. (that is, **The DDL will not be earlier than Mar. 11**.) But all the students have the same criteria. Since the students who are expected to take this course, they should submit lab and theory assignments on time. They should not delay submission time due to the late course selection.
- Only one lab assignment a week, only one DDL. Each assignment contains two questions, except for the first three weeks. **Late submissions are not allowed**. (Submissions later than DDL will not be accepted.). **If exceed the DDL, the score will be 0**.
- Please **submit your assignment in advance** so as not to submit fail or exceed DDL.
- All important notices are sent on BB platform.

Introduction to lab assignments

- This course is intended to judge the lab assignments online, which requires that you should strictly comply with the requirements of the question. We will supply a complete and clear description of the question, as well as the format of input and output.
- In order to encourage students to **do the right thing right the first time**, the second question will be penalized if you submit your code **more than two times**. If your first two submissions can pass all the test cases, you will get the full score; otherwise, according to the number of submissions, the score will **be deducted 5 points per submission**. The final grade is the best grade you have achieved.
- An example:
  - ✓ If you pass 60% test cases at first submit or the second submit, you can get 60.
  - ✓ At the third submit, you pass 95% cases, you will get 95-5 = 90.
  - ✓ At the fourth submit, you pass 90% cases, you will get 90-10 = 80. Finally, you will get 90.
- If we find some problems in our own code or system, we will fix these problems as soon as possible, the number of submissions of all participants will be reset to 0 accordingly.

Plagiarism Policy

- **From Spring 2022**, the plagiarism policy applied by the Computer Science and Engineering department is the following:
  - * **If an undergraduate assignment is found to be plagiarized, the first time the score of the assignment will be 0.**
  - * **The second time the score of the course will be 0.**
  - * **If a student does not sign the Assignment Declaration Form or cheats in the course, including regular assignments, midterms, final exams, etc., in addition to the grade penalty, the student will not be allowed to enroll in the two CS majors through 1+3, and cannot receive any recommendation for postgraduate admission exam exemption and all other academic awards.**
- As it may be difficult when two assignments are identical or nearly identical who actually wrote it, the policy will **apply to BOTH students**, unless one confesses having copied without the knowledge of the other.

What is OK, and what isn't OK?

- It's OK to work on an assignment with a friend, and think together about the program structure, share ideas and even the global logic. At the time of actually writing the code, you should write it alone.
- It's OK to use in an assignment a piece of code found on the web, as long as you indicate in a comment where it was found and don't claim it as your own work.
- It's OK to help friends debug their programs (you'll probably learn a lot yourself by doing so).
- It's OK to show your code to friends to explain the logic, as long as the friends write their code on their own later.
- **It's NOT OK to take the code of a friend, make a few cosmetic changes (comments, some variable names) and pass it as your own work.**

No excuse will be accepted once plagiarism is discovered!

Assignment 0

Please submit **Assignment 0** (**Assignment Declaration Form**) to BB **before the deadline: Mar.10, 2025 23:55 pm**, otherwise, you will lose all points for class performance.

Recursion

YAO ZHAO

Factorial

The "Hello, World" for recursion is the *factorial*function, which is defined for positive integers *n* by the equation:

$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

The following relationship can be observed:

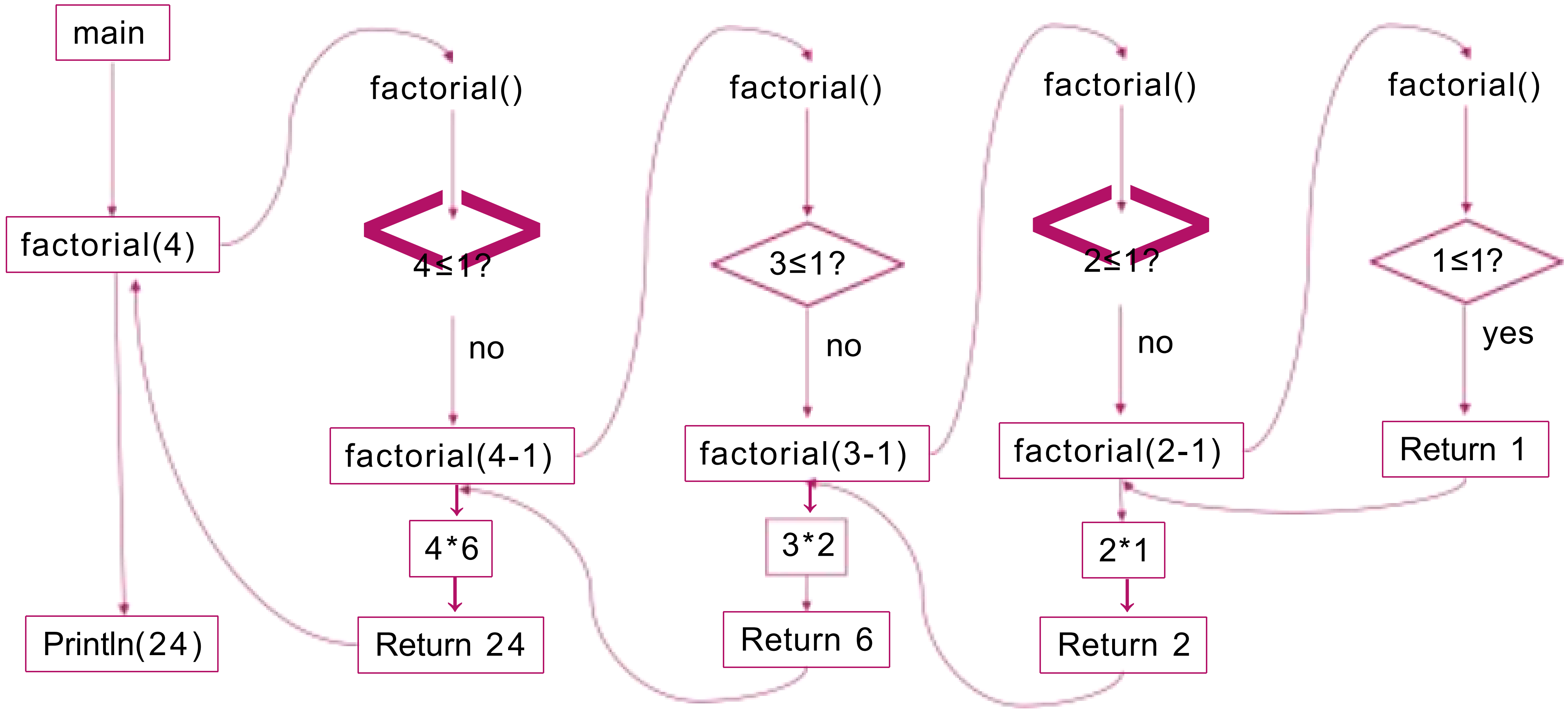
$n! = n \cdot (n-1)!$

We can implement the same problem by recursion.

```
public static long factorial(long n) {
    if (n <= 1) /   testfor base case
        return 1; /   base cases: 0! = 1 and1! = 1
    else /   recursion step
        return n * factorial(n - 1);
}
```

Factorial: Trace this computation

```
public static void main(String[] args) {
        System.out.println(factorial(4));
}
```



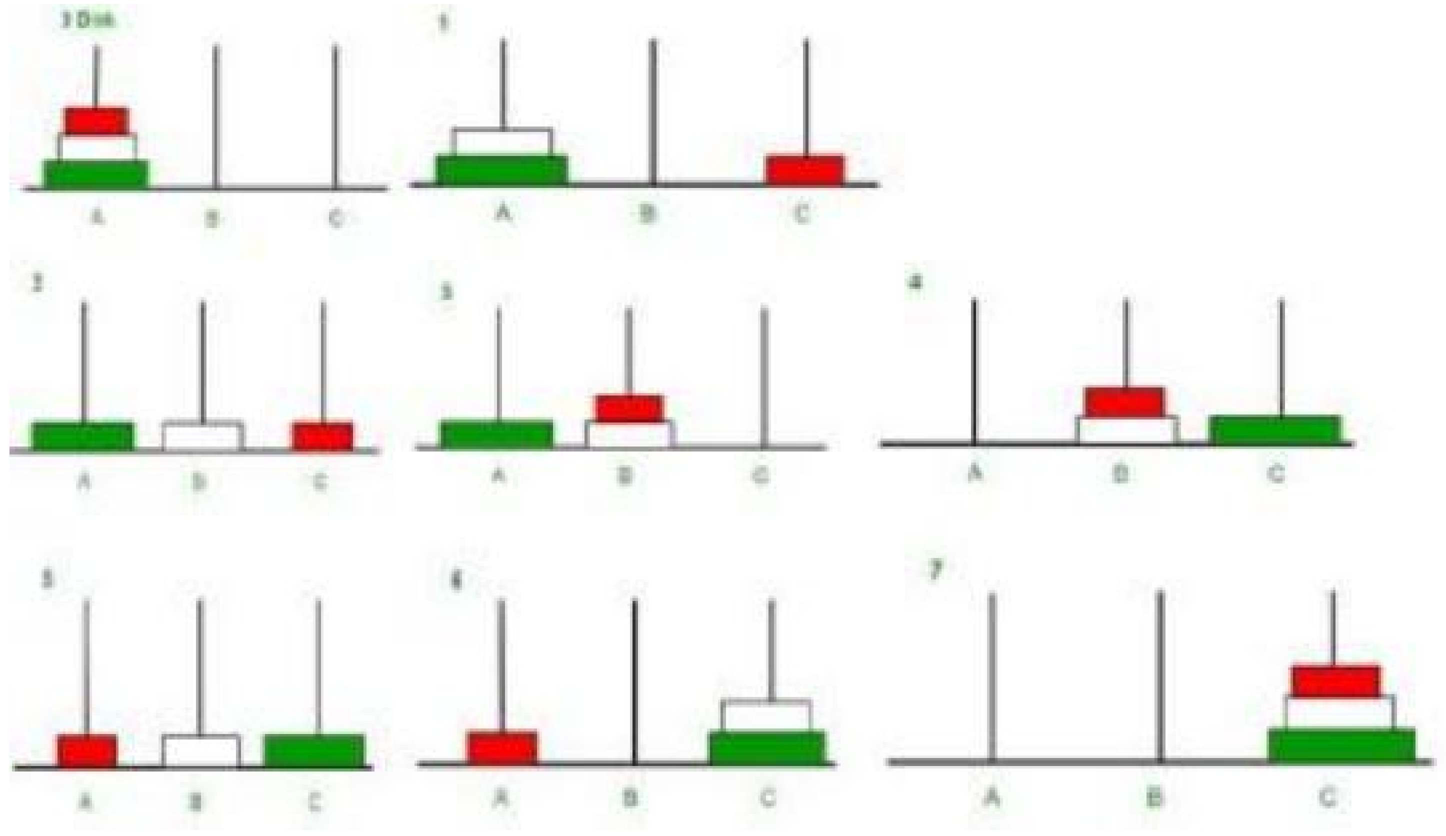
Tower of Hanoi

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:

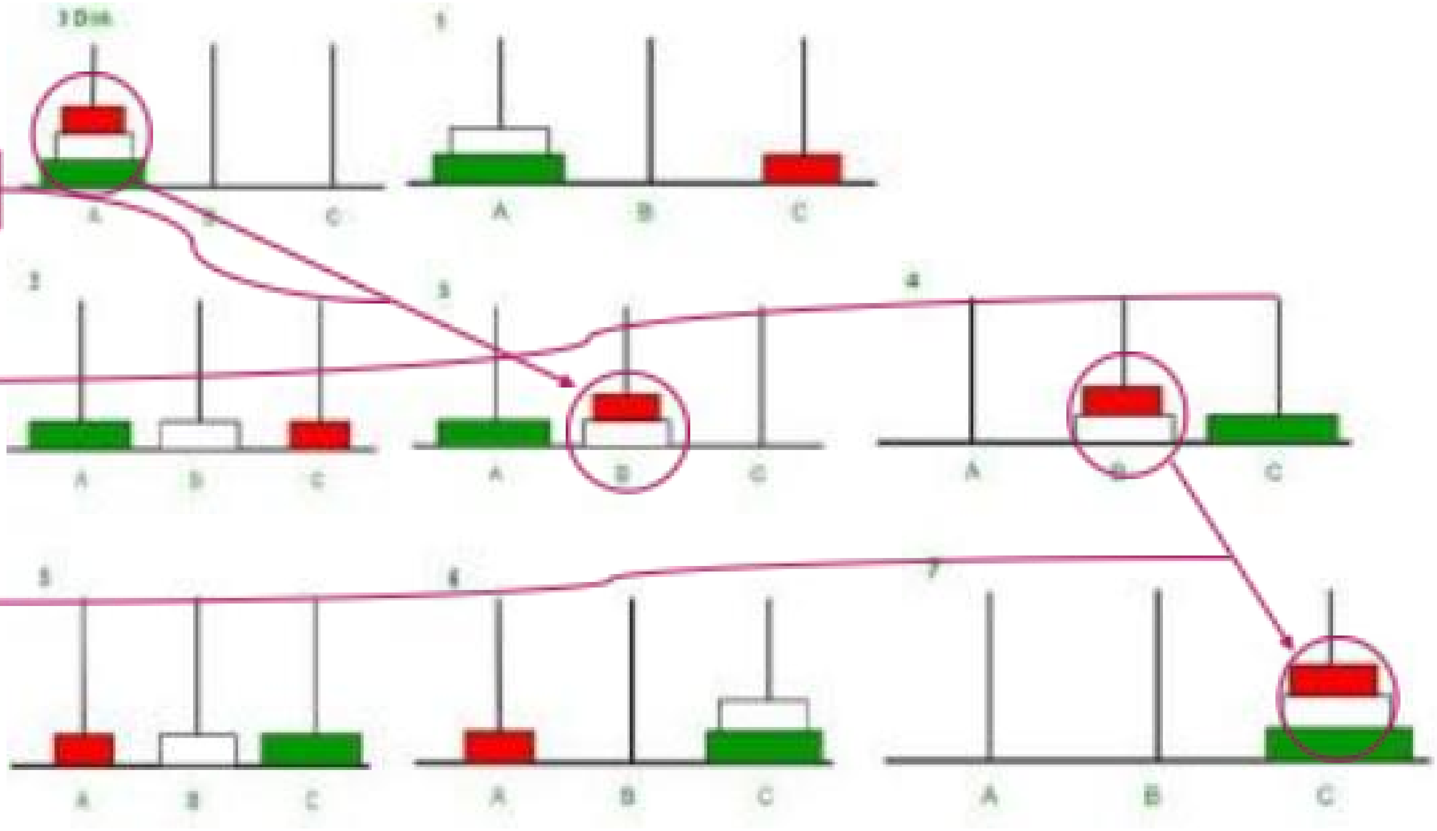
1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e., a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Tower of Hanoi: Sample n = 3

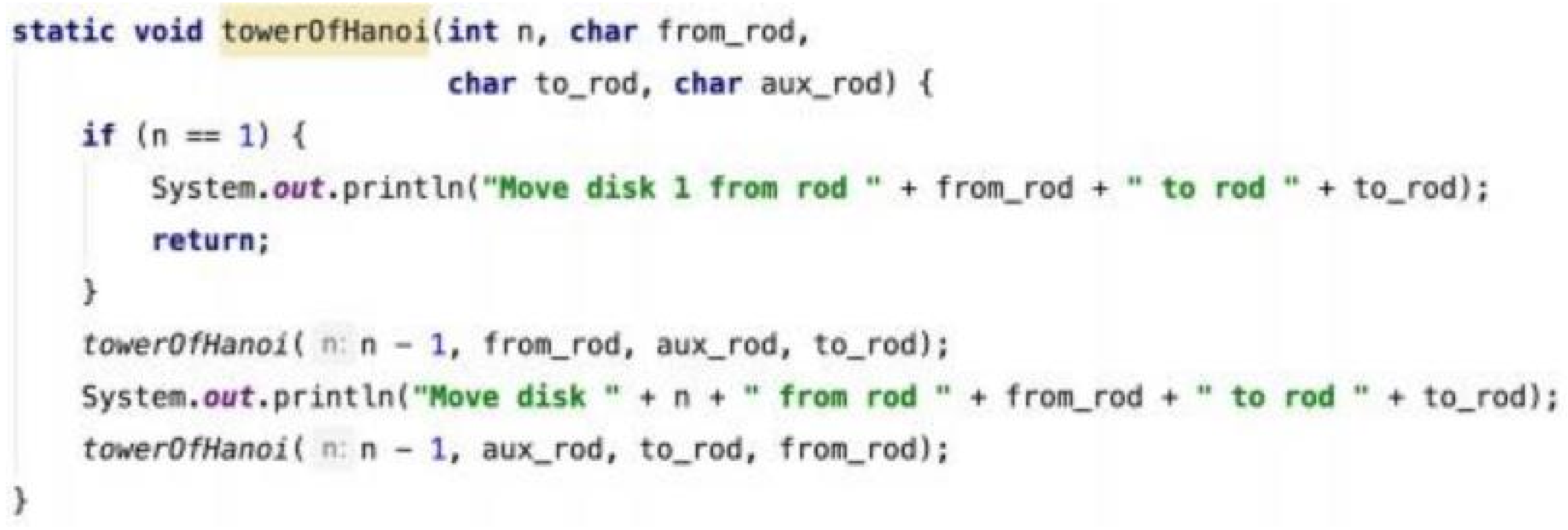
1.Disk 1 moved from A to C
2.Disk 2 moved from A to B
3.Disk 1 moved from C to B
4.Disk 3 moved from A to C
5.Disk 1 moved from B to A
6.Disk 2 moved from B to C
7.Disk 1 moved from A to C



Tower of Hanoi: Sample n = 3


1~3. n-1 disks move from A to B
4. Disk n moved from A to C
5~7. n-1 disks move from B to C
3 Disk
A
B
C

Tower of Hanoi: Code



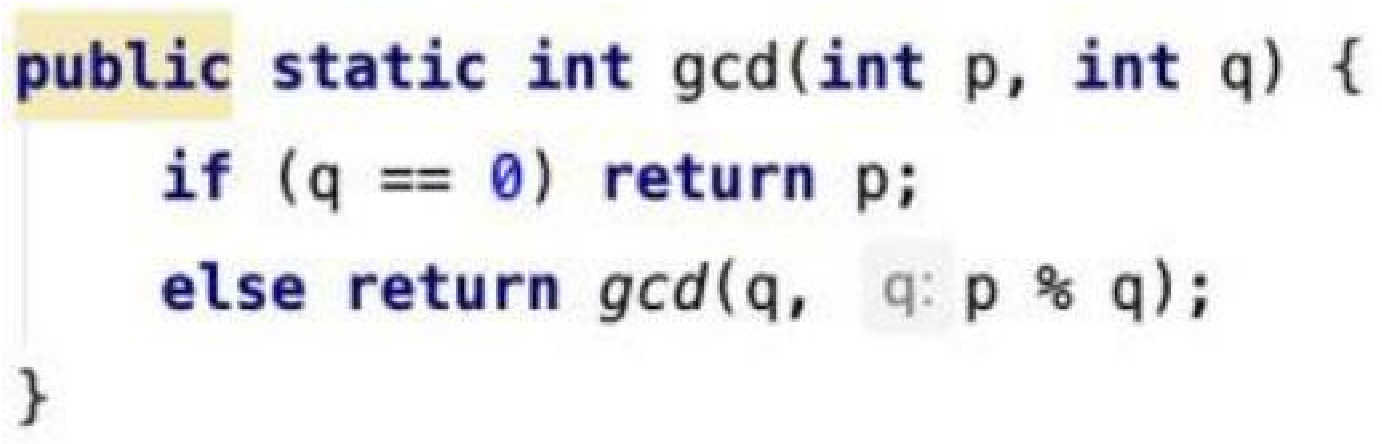
```
static void towerOfHanoi(int n, char from_rod,
                         char to_rod, char aux_rod) {
    if (n == 1) {
        System.out.println("Move disk 1 from rod " + from_rod + " to rod " + to_rod);
        return;
    }
    towerOfHanoi( n: n - 1, from_rod, aux_rod, to_rod);
    System.out.println("Move disk " + n + " from rod " + from_rod + " to rod " + to_rod);
    towerOfHanoi( n: n - 1, aux_rod, to_rod, from_rod);
}
```

https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/
https://www.youtube.com/watch?v=YstLjLCGmgg

Practice: trace the computation for the tower of Hanoi like P.3

Euclid's algorithm

The greatest common divisor (gcd) of two positive integers is the largest integer that divides evenly into both of them. For example, the gcd(102, 68) = 34.
We can efficiently compute the gcd using the following property, which holds for positive integers p and q:
If p > q, the gcd of p and q is the same as the gcd of q and p % q.



```
public static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, q: p % q);
}
```

Pitfalls of recursion

Code1:

```
public static double harmonic(int n) {
    return harmonic(n-1) + 1.0/n;
}
```

Pitfalls of recursion

Pitfall 1: ***Missing base case***

The recursive function **harmonic** supposed to compute harmonic numbers, but is missing a base case.
If you call this function, it will repeatedly call itself and never return.

Pitfalls of recursion

```
public static double harmonic(int n) {
    if (n == 1) return 1.0;
    return harmonic(n) + 1.0/n;
}
```

Pitfalls of recursion

Pitfall 2: ***No guarantee of convergence***

The recursive function **harmonic** goes into an infinite recursive loop for any value of its argument (except 1).
This is another common problem which is to include within a recursive function a recursive call to solve a subproblem that is not smaller than the original problem.

Pitfalls of recursion

```
public static double harmonic(int n) {
    if (n == 1) return 1.0;
    return harmonic(n-1) + 1.0/n;
}
```

Pitfalls of recursion

Pitfall 3:*Excessive memory requirements*

The recursive function **harmonic** correctly computes the nth harmonic number.
However, calling it with a huge value of n (50000 for example)will lead to
a **StackOverflowError**.
If a function calls itself recursively an excessive number of times before returning,
the memory required by Java to keep track of the recursive calls may be prohibitive.

Pitfalls of recursion

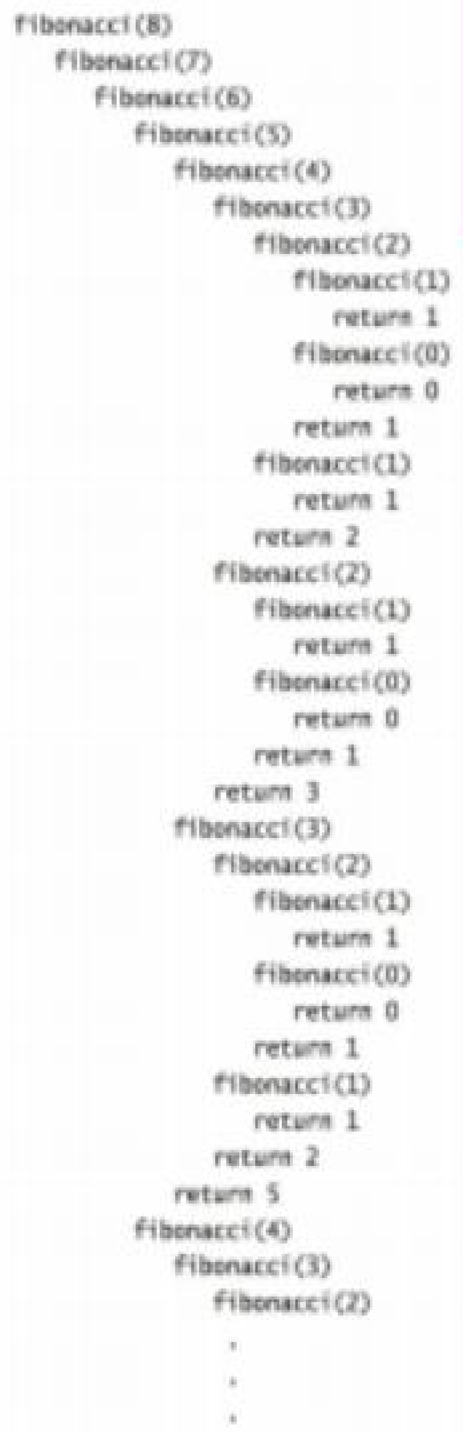
```
public static long fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Pitfalls of recursion

Pitfall 4: ***Excessive recomputation.***

This program is spectacularly inefficient. To see why it is futile to do so, consider what the function does to compute fibonacci(8) = 21. It first computes fibonacci(7) = 13 and fibonacci(6) = 8. To compute fibonacci(7), it recursively computes fibonacci(6) = 8 *again* and fibonacci(5) = 5. Things rapidly get worse. The number of times this program computes fibonacci(1) when computing fibonacci(n) is precisely F_n.

The temptation to write a simple recursive program to solve a problem must always be tempered by the understanding that a simple program might require exponential time (unnecessarily), due to excessive recomputation.



```
fibonacci(8)
  fibonacci(7)
    fibonacci(6)
      fibonacci(5)
        fibonacci(4)
          fibonacci(3)
            fibonacci(2)
              fibonacci(1)
                return 1
              fibonacci(0)
                return 0
              return 1
            fibonacci(1)
              return 1
            return 2
          fibonacci(2)
            fibonacci(1)
              return 1
            fibonacci(0)
              return 0
            return 1
          return 3
        fibonacci(3)
          fibonacci(2)
            fibonacci(1)
              return 1
            fibonacci(0)
              return 0
            return 1
          fibonacci(1)
            return 1
          return 2
        return 5
      fibonacci(4)
        fibonacci(3)
          fibonacci(2)
           .
           .
           .
```

Memoization(记忆化)

In computing, memoization or memoisation is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again. Memoization has also been used in other contexts (and for purposes other than speed gains), such as in simple mutually recursive descent parsing. Although related to caching, memoization refers to a specific case of this optimization, distinguishing it from forms of caching such as buffering or page replacement. In the context of some logic programming languages, memoization is also known as tabling. (wikipedia)

How to use memoization to avoid ***Excessive recomputation?***

Problem: Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

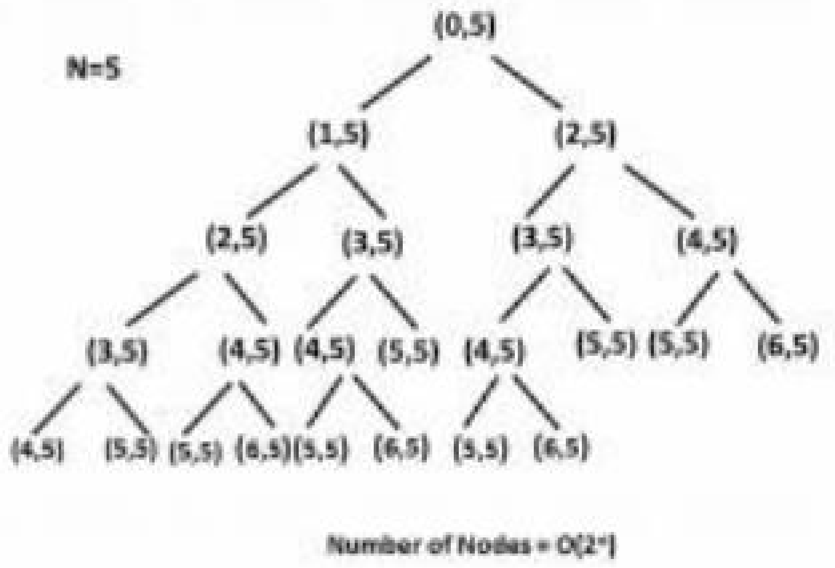
```
Example 1:
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps

Example 2:
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

Source： LeetCode　　　　Link： https://leetcode-cn.com/problems/climbing-stairs

Simple Recursion

$$\text{climbStairs}(i, n) = \text{climbStairs}(i+1, n) + \text{climbStairs}(i+2, n)$$

```
public class ClimbStairs {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        System.out.println(climbStairs(0,n));
    }
    public static int climbStairs(int i, int n) {
        if (i > n) {
            return 0;
        }
        if (i == n) {
            return 1;
        }
        return climbStairs(i + 1, n) + climbStairs(i + 2, n);
    }
}
```



Memoization Recursion

```
public class ClimbStairs {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int memo[] = new int[n + 1];
        System.out.println(climbStairs(0, n, memo));
    }
    public static int climbStairs(int i, int n, int memo[]) {
        if (i > n) {
            return 0;
        }
        if (i == n) {
            return 1;
        }
        if (memo[i] > 0) {
            return memo[i];
        }
        memo[i] = climbStairs(i + 1, n, memo) + climbStairs(i + 2, n, memo);
        return memo[i];
    }
}
```

The result of each step is stored in the **memo** array, and each time the function is called again, we return the result directly from the **memo** array.

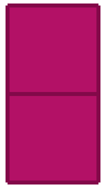
With the help of the **memo** array, we get a repaired recursion tree whose size is reduced to n.

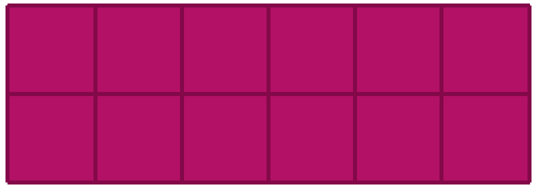
Recursion Practice

Recursion Practice:

1. trace the computation for the tower of Hanoi like P.3

2. Find the number of ways a 2*n rectangle can be tiled with rectangular tiles of size 2*1


2*1


2*
6

Practice (Optional):

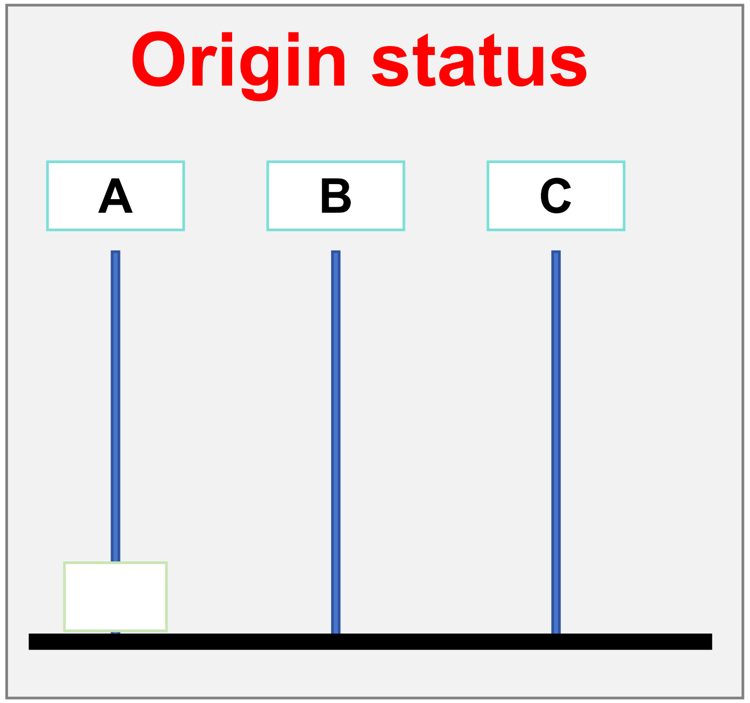
3. Enter a string and print out all permutations of the characters in the string.(0.1 points)
Example:
Input: abc
Output: abc, acb, bac, bca, cab, cba

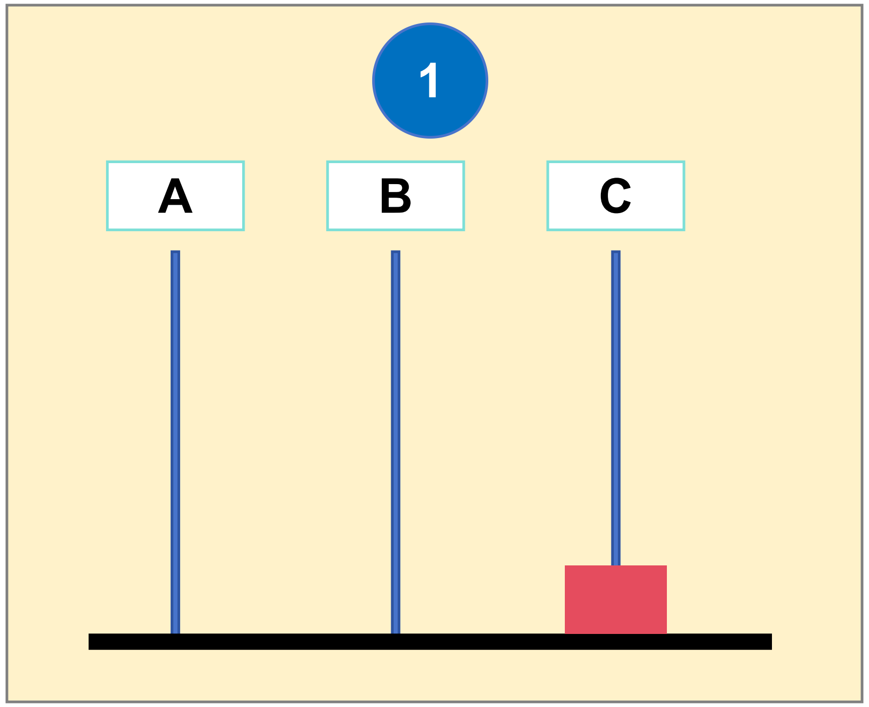
4. Enter a string and print out all combinations of the characters in the string.(0.1 points)
Example:
Input: abc
Output: a, b, c, ab, bc, ac, abc

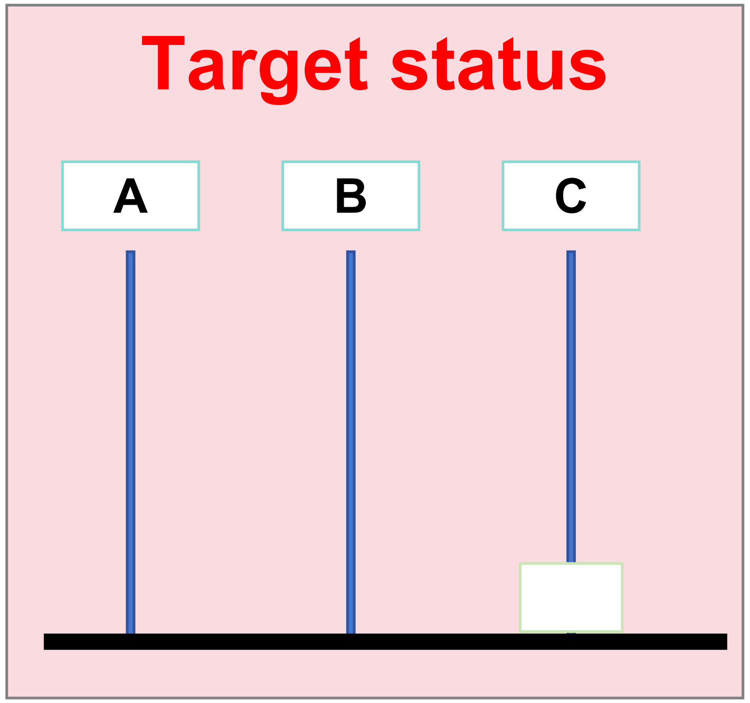
The practice will be checked in this lab class or the next lab class by teachers or SAs by email or face-to-face or Enterprise WeChat or etc..DDL Mar.2 23:59.
Question 3 and 4 are optional, just describe ideas, don ’ t have to write code.

This practice will contribute **1 mark** to your overall grade. Late submissions within 2 weeks after the deadline (Mar.2 23:59)will incur a 20% penalty, meaning that you can only get 80% of the score.No acception after Mar.16 23:59.

Tower of Hanoi（from A to C）: Sample n = 1

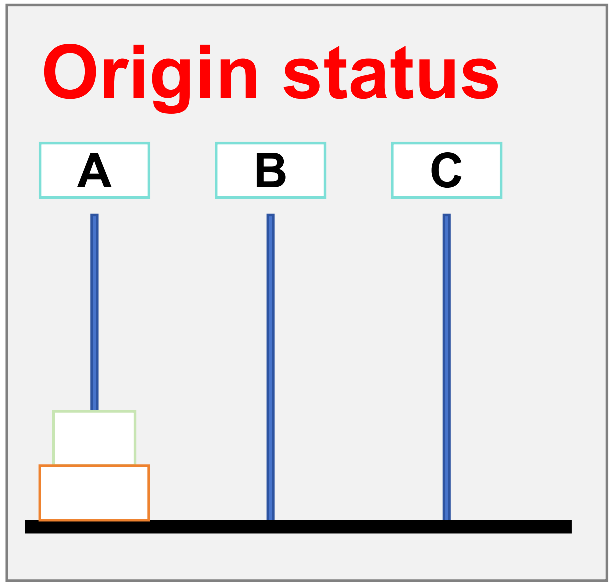


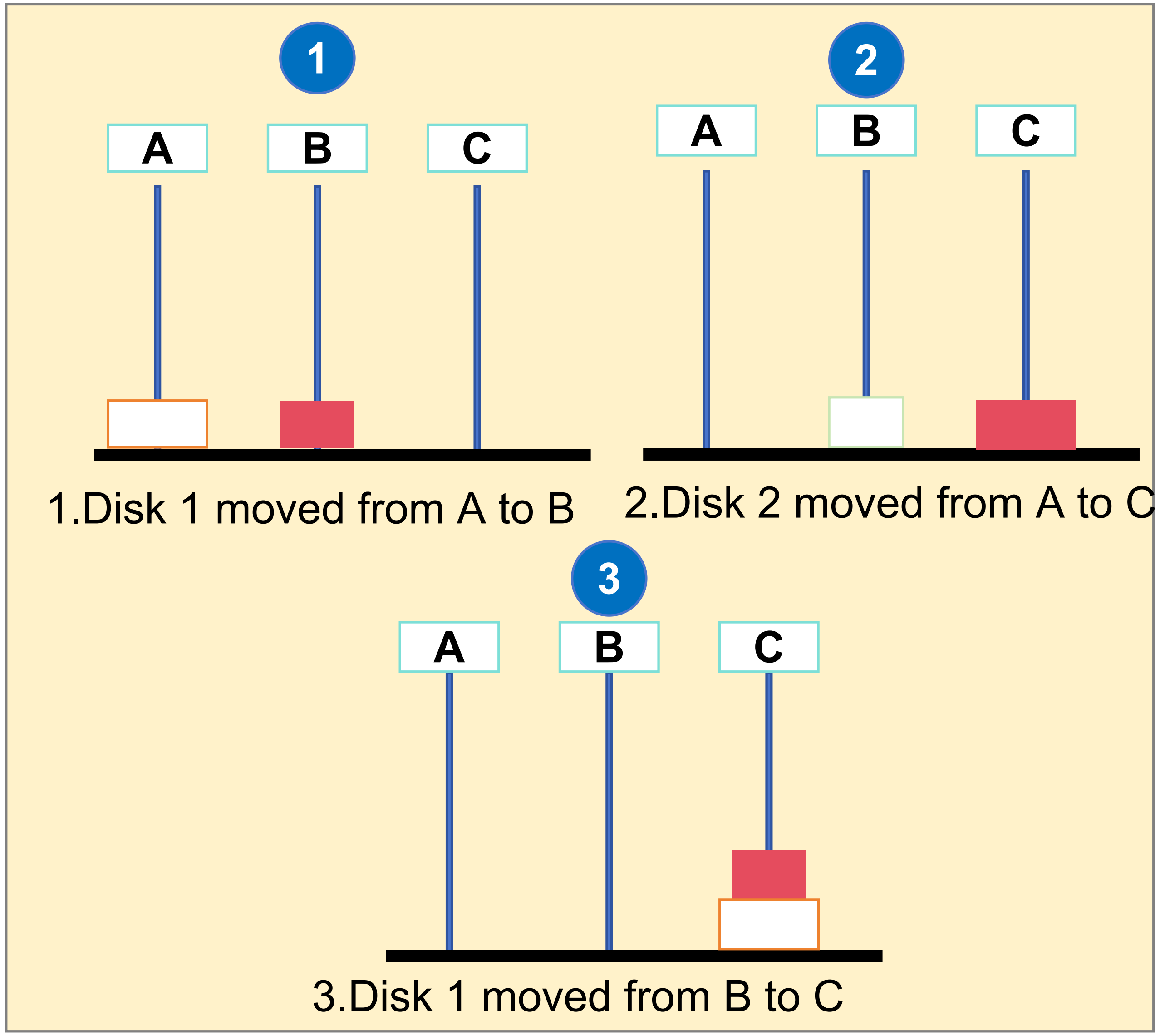




1.Disk 1 moved from A to C

Tower of Hanoi（from A to C）： Sample n = 2

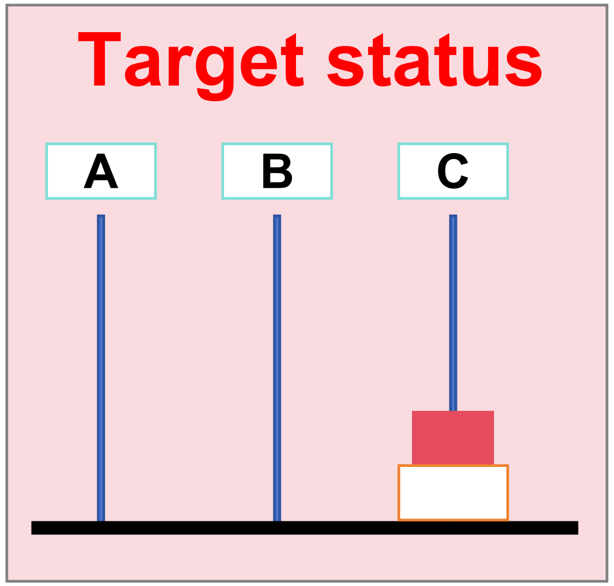




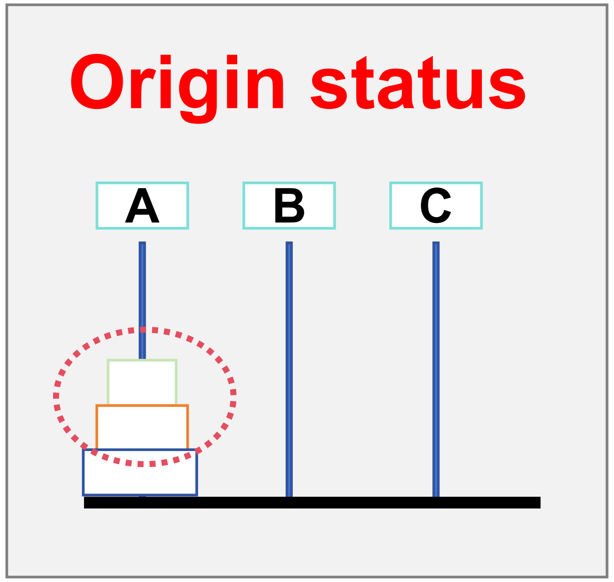
1.Disk 1 moved from A to B

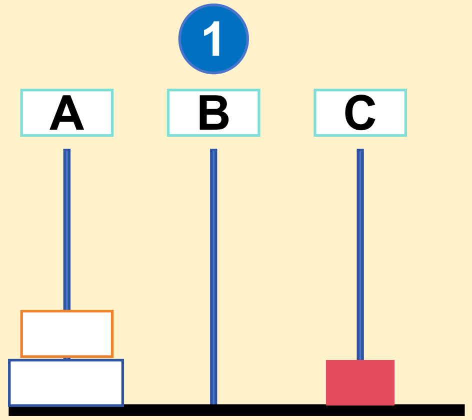
2.Disk 2 moved from A to C

3.Disk 1 moved from B to C

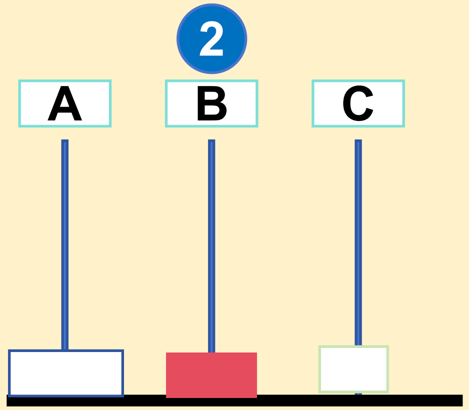


Tower of Hanoi（from A to C）: Sample n = 3 (Page1/3)

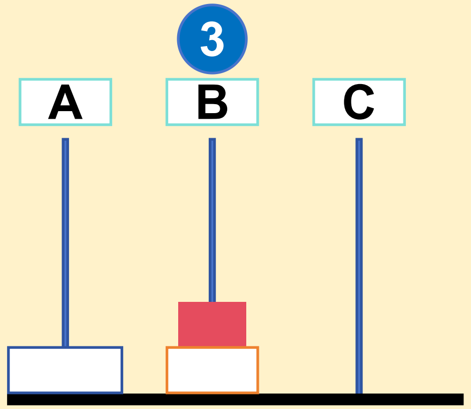




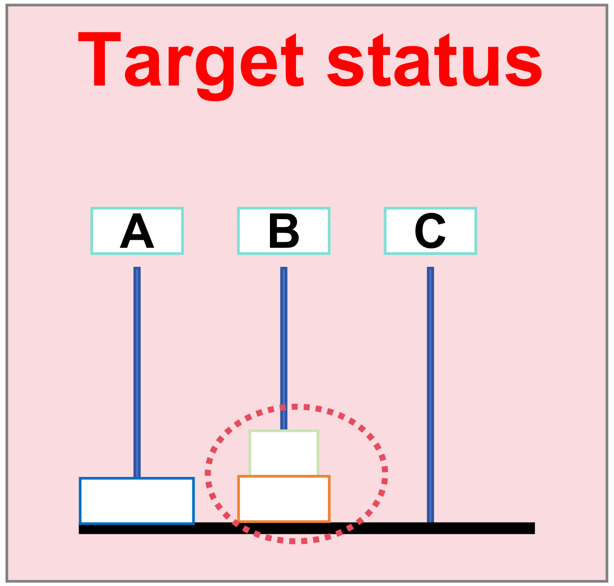
1.Disk 1 moved from A to C



2.Disk 2 moved from A to B

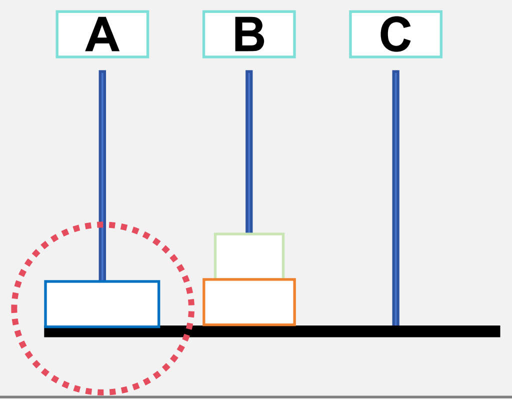


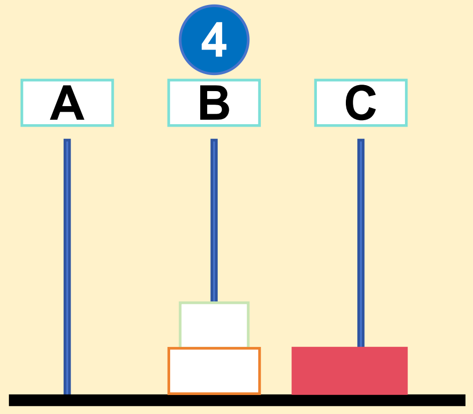
3.Disk 1 moved from C to B



Tower of Hanoi（from A to C）： Sample n = 3 (Page2/3)

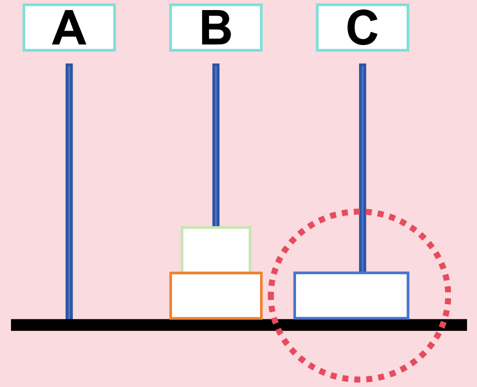
Origin status





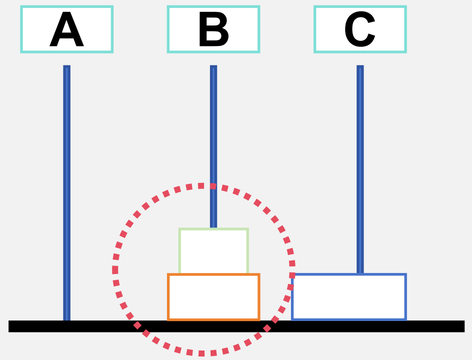
4.Disk 3 moved from A to C

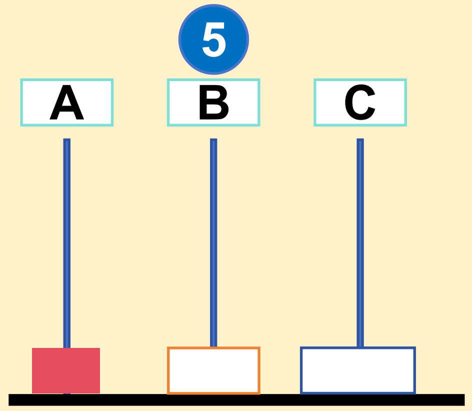
Target status



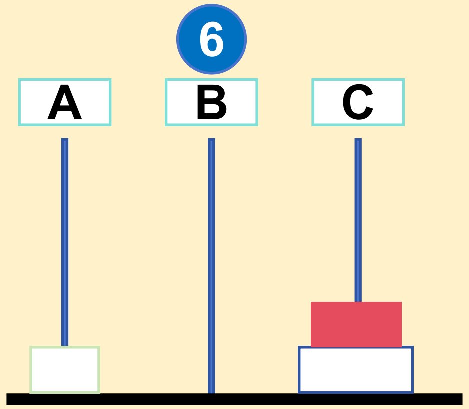
Tower of Hanoi（from A to C）： Sample n = 3 (Page1/3)

Origin status

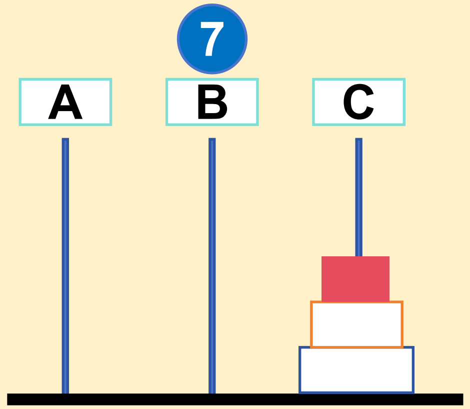




5.Disk 1 moved from B to A

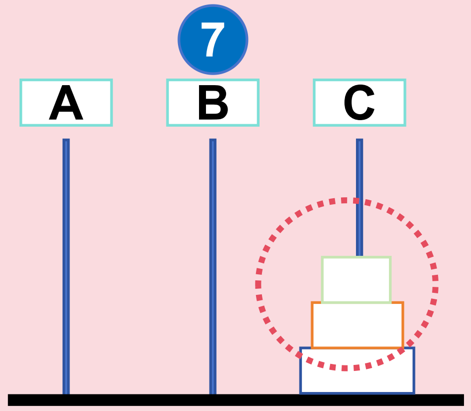


6.Disk 2 moved from B to C

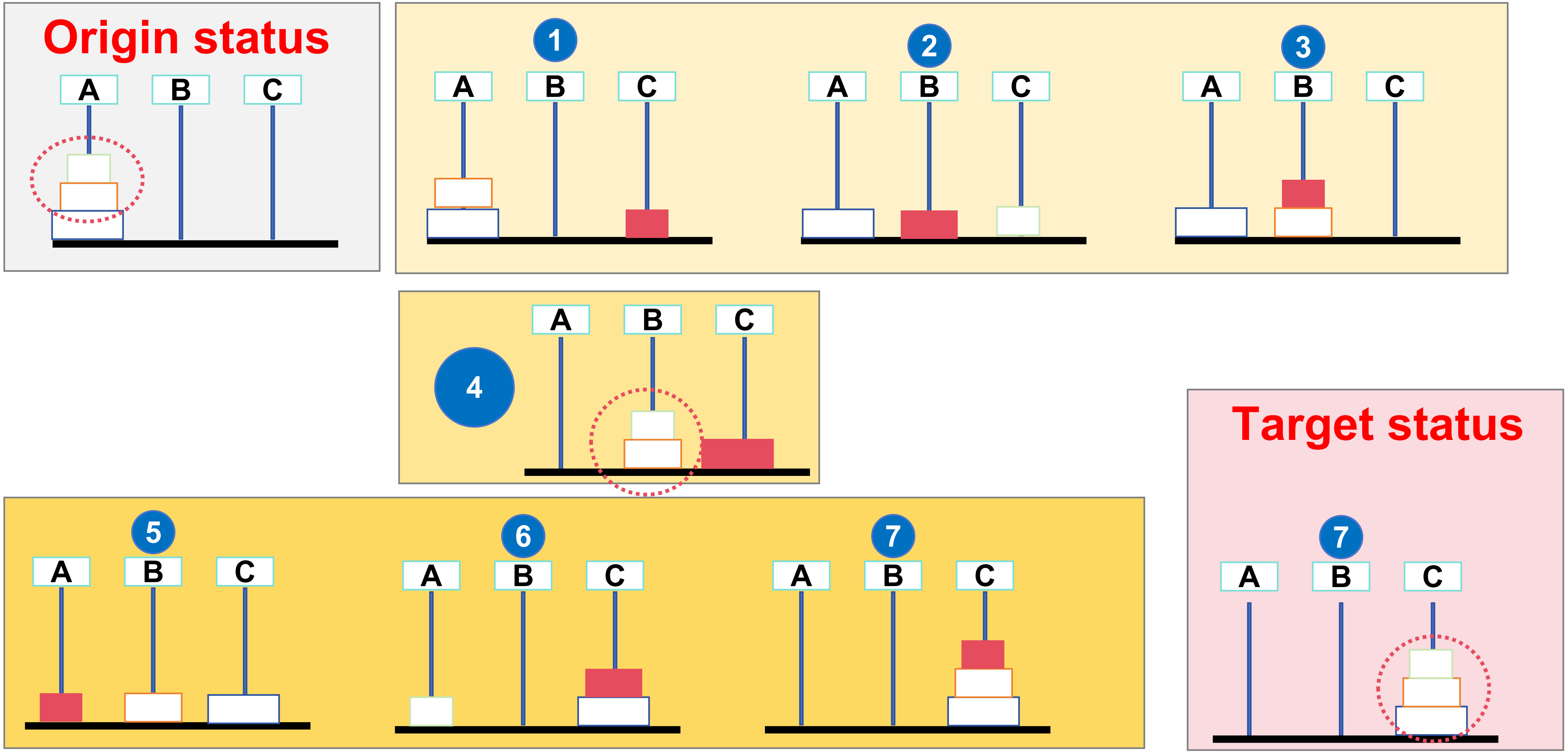


7.Disk 1 moved from A to C

Target status



Tower of Hanoi（from A to C）: Sample n = 3 overview


Origin status
A
B
C
1
2
3
4
5
6
7
Target status
7