

# เอกสารประกอบการอบรม ส่วนที่ 1 วิชาโครงสร้างข้อมูล

ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2 2/2566  
ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์  
ระหว่างวันที่ 18 มีนาคม – 3 เมษายน 2567

โดย

สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี  
มหาวิทยาลัยธรรมศาสตร์

# Recursion

# Recursion

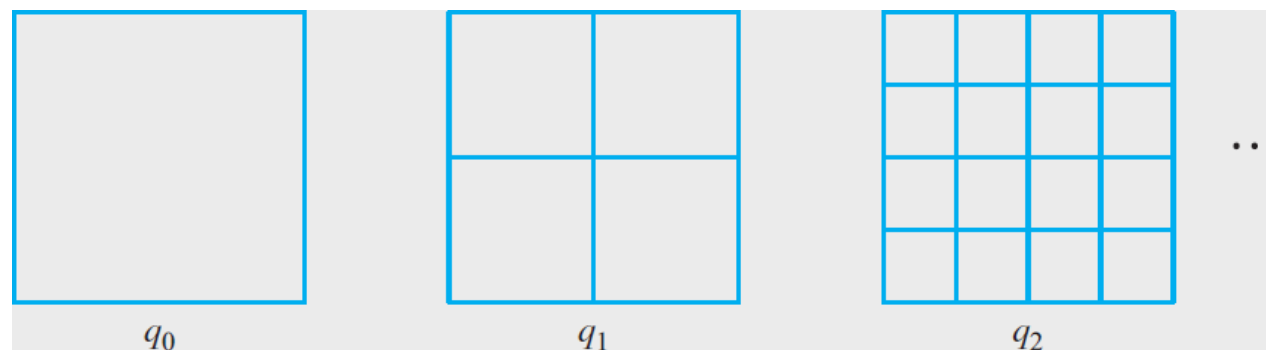
## นิยาม

**Recursion** คือการกำหนด object ใด ๆ ในเทอมที่ง่ายกว่าของตัวเอง โดยการทำงานจะต้องมีจุดจบที่แน่นอน

ตัวอย่าง Factorial, การคูณเลขสองจำนวน (Multiplication of natural number), Fibonacci sequence, Binary search

# ปัญหาที่ 1

- รูปขวามือแสดงการวาดรูปสี่เหลี่ยมจัตุรัสในลำดับที่  $q_0, q_1, q_2, \dots$
- จำนวนสี่เหลี่ยม (ที่ไม่ซ้อนทับกัน) มีจำนวนเท่าไร ในแต่ละขั้นตอนที่  $q_1, q_2, q_3, q_4, q_5$ ?



## ปัญหาที่ 2

ถ้าณิชนเริ่มต้นเข้าทำงานที่บริษัทแห่งหนึ่ง เริ่มต้นด้วยเงินเดือน 50,000 บาท และได้รับการการันตีว่าเงินเดือนจะขึ้น 10% ทุกครั้งปี ผ่านไป 3 ปี เงินเดือนของณิชนจะได้รับเท่าไร

1 4 9 16 ...

## ปัญหาที่ 3

ถ้าลำดับของเลขคี่คือ 1, 3, 5, 7, 9, ....

นิยามลำดับของ *S-numbers* ดังต่อไปนี้:

- ตัวเลขคือ 1
- ตัวที่สองคือ ผลบวกของตัวแรกกับตัวที่สองของเลขคี่
- ตัวที่สามคือ ผลบวกของตัวที่สองกับตัวที่สามของเลขคี่
- ตัวที่สี่คือ ผลบวกของตัวสามกับตัวที่สี่ของเลขคี่
- ฯลฯ

จงเขียน 7 ตัวแรกของ *S-numbers*

## โมเดลในการแทน ปัญหา

นิยามปัญหาในรูปแบบความสัมพันธ์ ซึ่งประกอบด้วย 2 ส่วน ได้แก่

- **base case:** เป็นส่วนที่อธิบายถึงจุดเริ่มต้น
- **recursive case:** อธิบายถึงค่าที่ได้ในขั้นตอนถัดมาโดยใช้ค่าของขั้นตอนก่อนหน้า

ลองกลับไปนิยามปัญหาก่อนหน้า

## 1. การนิยาม Factorial

$$5 \text{ factorial} = 5 * 4 * 3 * 2 * 1 = 120$$

$$4 \text{ factorial} = 4 * 3 * 2 * 1 = 24$$

$$3 \text{ factorial} = 3 * 2 * 1 = 6$$

:

$$0 \text{ factorial} = 1$$

เราสามารถเขียนนิยามของ Factorial โดยสรุปได้ดังนี้

$$f(n) = \begin{cases} 1 & ; n = 0 \\ n * f(n - 1) & ; n > 0 \end{cases}$$



## การหาค่าผลลัพธ์ (5!)

Bottom-up	
0!	1
1!	$1*1 = 1$
2!	$1*2 = 2$
3!	$2*3 = 6$
4!	$6*4 = 24$
5!	$24*5 = 120$

Top-down	
5!	$5*4!$
	$5*(4*3!)$
	$5*4*(3*2!)$
	$5*4*3*(2*1!)$
	$5*4*3*2*(1*0!)$
	$5*4*3*2*1*1 = 120$

# การหาค่า Factorial แบบ Bottom-up

รับค่าจำนวนเต็ม  $n$   
และ return ค่า  $n!$

$$f(n) = \begin{cases} 1 & ; n = 0 \\ n * f(n - 1) & ; n > 0 \end{cases}$$

```
int iterativeFactorial (int n ){  
    int fact = 1, x;  
    for ( x = 1; x <= n; x++)  
        fact = fact * x;  
    return(fact);  
}
```

# การหาค่า factorial แบบ Top-down



Recursive  
function

รับค่าจำนวนเต็ม  $n$   
และ return ค่า  $n!$

$$f(n) = \begin{cases} 1 & ; n = 0 \\ n * f(n - 1) & ; n > 0 \end{cases}$$

```
int recursiveFactorial ( int n ) {  
  
    int result;  
    if (n == 0) return 1;  
    else  
        result = n * recursiveFactorial (n - 1)  
    return ( result);  
}
```

## 2. การหาค่าผลบวกของเลขจำนวนเต็ม n จำนวนแรก

$$sum(n) = \begin{cases} 0 & ; n = 0 \\ n + sum(n - 1) & ; n > 0 \end{cases}$$

### Iteration (Bottom-up)

```
int sum (int n)
{
    int sumn = 0, i;
    for (i = 1; i <= n; i++ )
        sumn = sumn + i;
    return(sumn)
}
```

### Recursive (Top-down)

```
int sum (int n)
{
    if (n == 0) return (0);
    return(n + sum(n-1))
}
```



# สรุปหลักการเขียน อัลกอริทึมแบบ recursive

หา base case ซึ่งจะเป็นจุดจบของการทำงาน  
แบบ recursive เช่น

factorial  $\Rightarrow 0! = 1$

นิยามกรณี recursive โดย


- มองหาลักษณะการทำงานที่ซ้ำๆกันเป็นรูปแบบ และแบ่งงานเป็นงานย่อยที่สุดที่หาคำตอบได้ง่าย
- มองหาความสัมพันธ์กับ step ก่อนหน้า ว่าผลที่ได้อันใหม่จะคำนวณจากผลใน step ก่อนหน้าได้อย่างไร

### 3. การหาค่าลำดับของ Fibonacci

**Fibonacci sequence** คือ ชุดของเลขจำนวนเต็มที่ค่าในลำดับใด ๆ จะมีค่าเท่ากับค่าในลำดับสองตัวก่อนหน้ารวมกัน ยกเว้นในลำดับที่ 0 และ 1

Example

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...



# นิยามของ Fibonacci Sequence

ก่อนเขียนโปรแกรม เราจะนิยาม Fibonacci sequence แบบ recursive ก่อน

$$F(n) = \begin{cases} 1 & \text{if } n = 1 \text{ or } n = 2 \\ F(n-1) + F(n-2) & \text{if } n > 2. \end{cases}$$

**Base case** of the definition

**Recursive part** of the definition

## การหาค่าของลำดับ Fibonacci แบบ recursive

$$F(n) = \begin{cases} 1 & \text{if } n = 1 \text{ or } n = 2 \\ F(n-1) + F(n-2) & \text{if } n > 2. \end{cases}$$

### Iteration (Bottom-up)

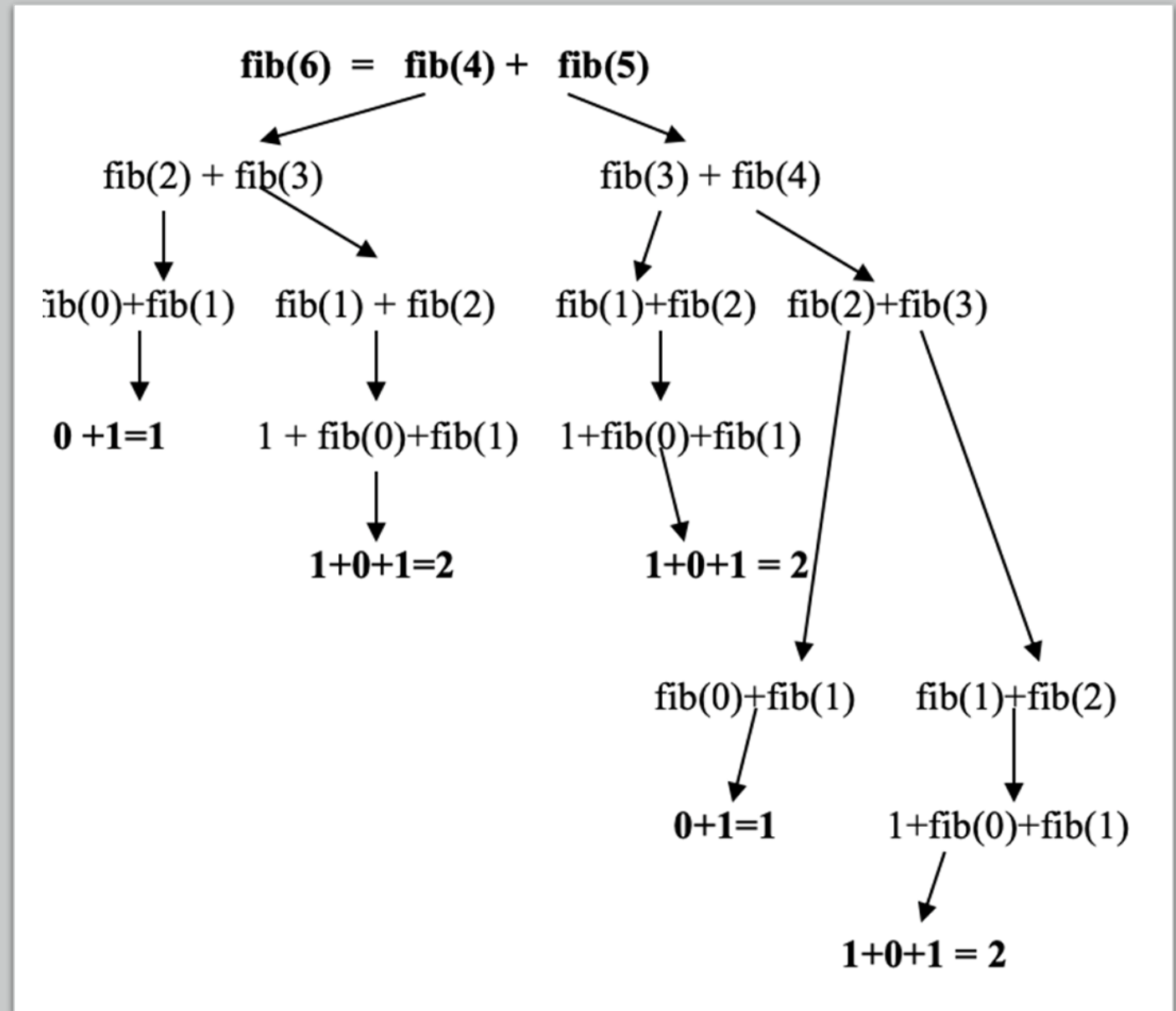
```
int fib (int n)
{
    int result[3] = {0, 1, 1}, i;
    for (i = 3; i <= n; i++) {
        result[0] = result[1] + result[2];
        result[2] = result[1];
        result[1] = result[0];
    }
    return(result[0]);
}
```

### Recursive (Top-down)

```
int fib (int n)
{
    if (n <= 1)
        return ( n );
    return(fib(n-1) + fib(n - 2));
}
```



## ตัวอย่างการคิด



## 4. การค้นหาแบบไบนารี (Binary Search)

Binary Search จะใช้กับข้อมูลอินพุตที่ได้เรียงลำดับเรียบร้อยแล้ว

ตัวอย่าง การหาว่ามีข้อมูลเลข 64 อยู่ใน array ของตัวเลขข้างล่างนี้หรือไม่

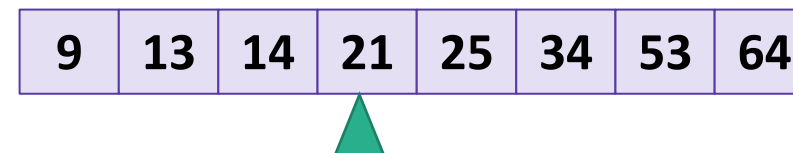
9	13	14	21	25	34	53	64
---	----	----	----	----	----	----	----

- ถ้าใช้การเปรียบเทียบแบบธรรมดาตั้งแต่ตัวแรก จะต้องทำการเปรียบเทียบ 8 ครั้ง จึงจะเจอ
- การใช้การค้นหาแบบ binary จะช่วยให้จำนวนครั้งของการเปรียบเทียบน้อยลง เพราะการเปรียบเทียบแต่ละครั้งสามารถตัดข้อมูลกึ่งหนึ่งออกจากการพิจารณาได้ ทำให้ลดจำนวนข้อมูลที่จะต้องทำการเปรียบเทียบลงมาก

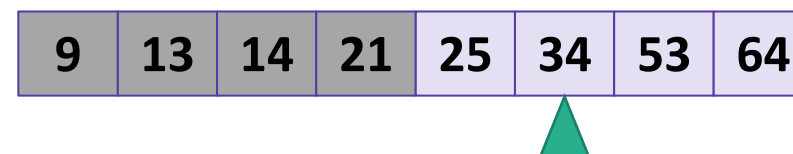


# ตัวอย่างการค้นหา แบบไบนารี

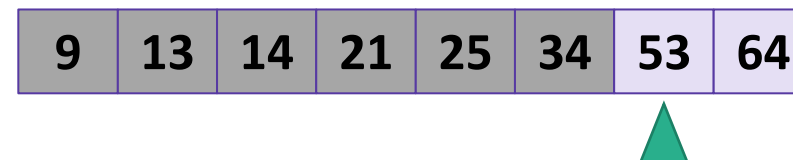
รอบที่ 1



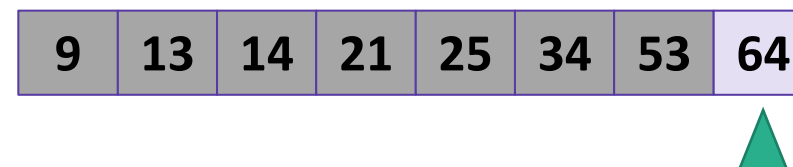
รอบที่ 2



รอบที่ 3



รอบที่ 4



## การค้นหาแบบไบนารี แบบ recursive

สำหรับการค้นหาข้อมูล  $x$  ใน  
array  $a$  ที่มีข้อมูลที่เรียงลำดับ  
แล้วจาก  $a[\text{low}]$  ถึง  $a[\text{high}]$

```
int binsrch (int a[], int x, int low, int high)
{
    int mid;

    if ( low > high)
        return(-1);
    mid = (low + high) / 2;
    if (x == a[mid])
        result = mid;
    else if ( x < a[mid])
        result = binsrch(a, x, low, mid-1);
    else
        result = binsrch(a, x, mid+1,high);
    return(result);
}
```

คุณสมบัติของ  
อัลกอริทึมแบบ  
Recursive

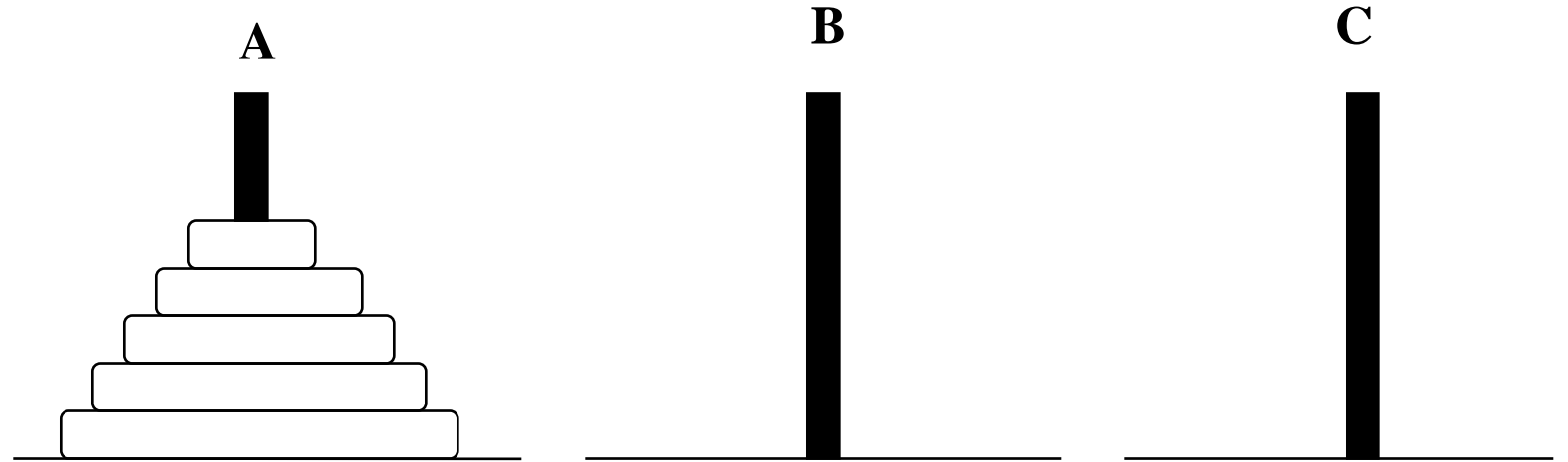
---

จะต้องมีส่วนการทำงานที่เป็น **base case**  
ให้การทำงานมีจุดสิ้นสุด

---

จะต้องมีส่วนที่เป็น **recursive part** เขียน  
ให้อยู่ในรูปที่เรียกตัวเองได้ ซึ่งส่วนนี้คือการ  
แบ่งงานเป็นส่วนย่อยๆ ที่มีความสัมพันธ์กัน

# The Tower of Hanoi Problem



สมมติมีเสาทั้งหมด 3 ต้นให้ชื่อว่า ต้น A, B และ C และมีจานทั้งหมด 5 ใบที่มีเส้นผ่าศูนย์กลางต่างกันวางซ้อนกันอยู่บนเสาต้นแรก โดยให้จานใบใหญ่กว่าจะอยู่ใต้ใบที่เล็กกว่าเสมอ ดังรูป

# กติกาการเล่น



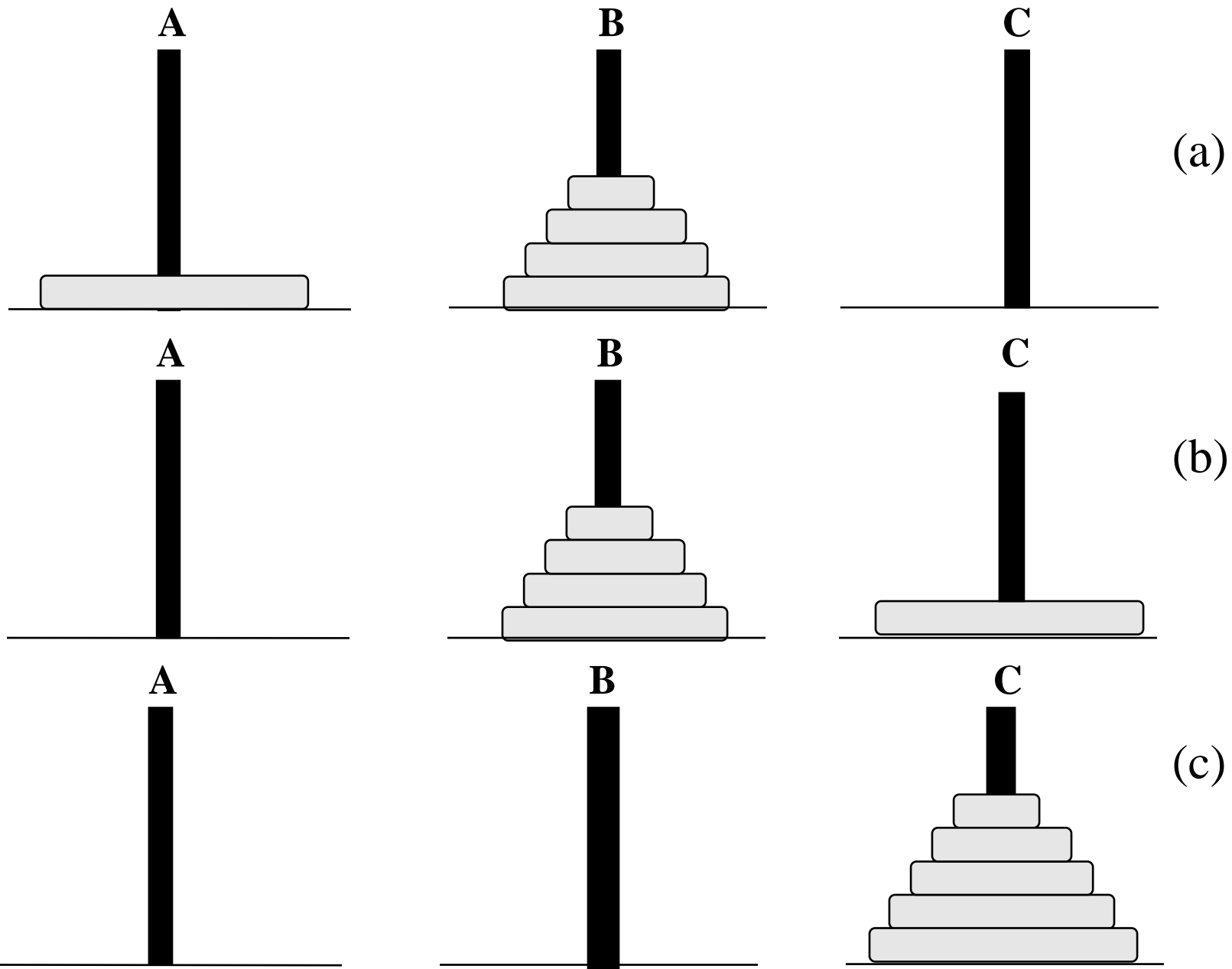
เป้าหมาย: เราต้องการย้ายจานทั้ง 5 ใบจากต้น A ไปต้น C โดยใช้เสาต้น B เป็นต้นพัก โดยมีเงื่อนไขในการย้าย ดังนี้



ในการย้ายครั้งหนึ่งๆ เราสามารถย้ายได้เฉพาะจานใบบนสุดเท่านั้น



จานใบใหญ่กว่าไม่สามารถอยู่บนจานที่เล็กกว่า

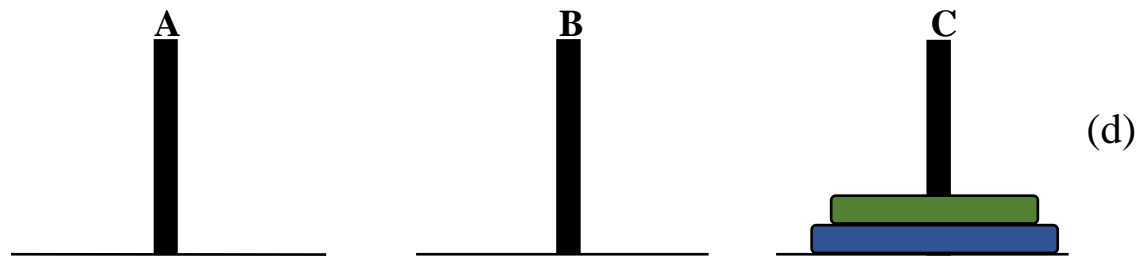
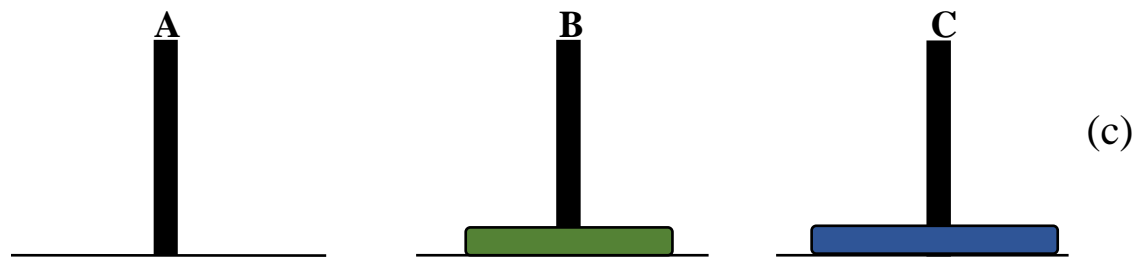
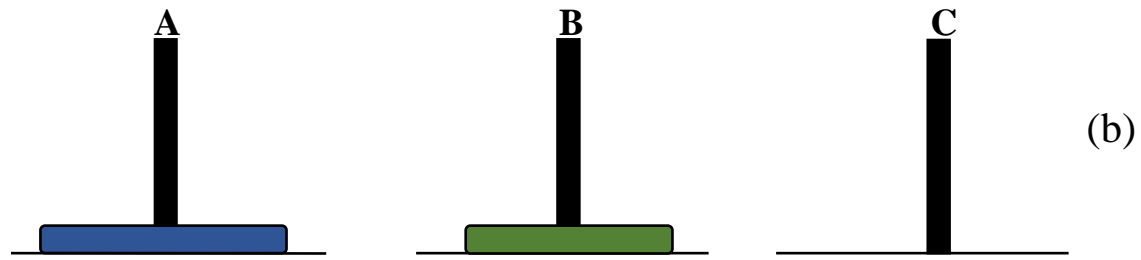
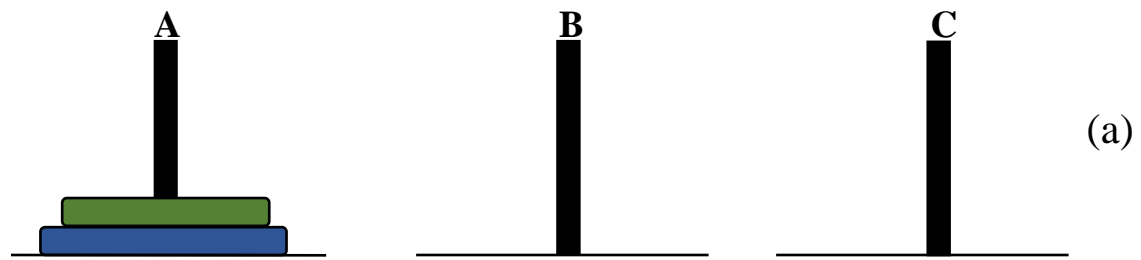






พักผ่อนเกมกันก่อนที่....

<https://www.mathsisfun.com/games/towerofhanoi.html>



## การลองแก้ปัญหาด้วย วิธี recursion

ถ้าสามารถแก้ปัญหาสำหรับจำนวน  $n-1$  ได้ ก็จะสามารถแก้ปัญหาจำนวน  $n$  ได้ พยายามเขียนปัญหา  $n$  ให้อยู่ในรูปของปัญหา  $n-1$  ได้

ขั้นตอนการแก้ปัญหา: ให้  $n$  แทนจำนวนงาน

1) ถ้า  $n = 1$  ย้ายงานใบเดียวนี้จากเสาต้น A ไปต้น C

2) ถ้า  $n > 1$

ย้าย  $n-1$  ใบจากเสาต้น A ไปต้น B โดยใช้เสาต้น C เป็นต้นพัก

ย้ายใบที่เหลือจากเสาต้น A ไปรอไว้ที่ต้น C

ย้าย  $n-1$  ใบจากเสาต้น B ไปต้น C โดยใช้เสาต้น A เป็นต้นพัก

ฉะนั้น Base case สำหรับกรณีนี้คือเมื่อ  $n = 1$

# อัลกอริทึมสำหรับ Tower of Hanoi

```
void towers(int n, char frompeg, char topeg, char auxpeg)
{
    // Base case
    if (n == 1) {
        printf("move disk 1 from peg %c => %c\n", frompeg, topeg);
        return;
    }
    // Recursive case
    // move top n-1 disks from A to B using C as auxiliary
    towers(n-1, frompeg, auxpeg, topeg);

    // move remaining disk from A to C
    printf("move disk %d from peg %c => %c\n", n, frompeg, topeg);

    // move n-1 disk from B to C using A as auxiliary
    towers(n-1, auxpeg, topeg, frompeg);
}
```