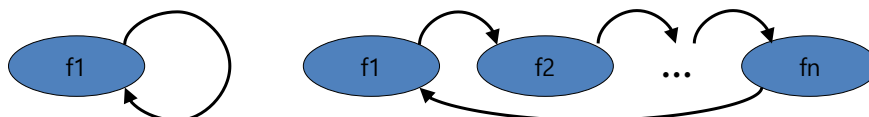




RECURSION คืออะไร

- Recursion คือ ปรากฏการณ์ที่มีการวนกลับไปอ้างอิงถึงตัวเองซ้ำแล้วซ้ำเล่า ในภาษาซีนั้นการเวียนเกิดจะเกิดขึ้นกับ Function
- Recursive Function คือ Function ที่เรียกตัวมันเอง หรือ เป็นส่วนหนึ่งของ cycle ในลำดับการเรียกฟังก์ชัน function calls.



- ตัวอย่าง Recursive Function เราสามารถ Implement การ
คุณได้ด้วยการบวก

โจทย์ที่เหมาะสมกับการใช้ Recursive Functions

- เมื่อมีกรณีอย่างน้อย 1 กรณีที่สามารถหาคำตอบได้ทันที (simple case)
- ในขณะที่กรณีอื่น ๆ สามารถกำหนดได้ใหม่ (redefine) ในรูปที่ง่ายขึ้นกว่าเดิม
- ปัญหาสามารถถูกลดรูปลงสู่ simple case

If this is a simple case

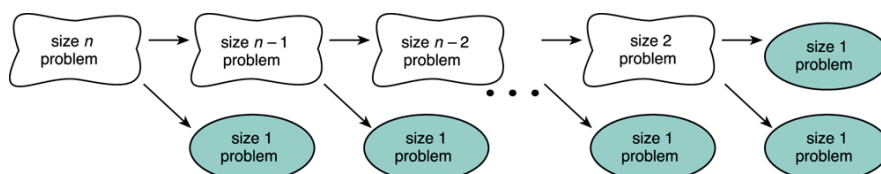
solve it

else

redefine the problem using recursion

การแบ่งปัญหออกเป็นปัญหาย่อย

- สมมติให้ปัญหาที่มีขนาดข้อมูลเป็น 1 สามารถหาคำตอบได้ทันที (i.e., the simple case).
- เราสามารถแบ่งปัญหออกเป็นสองส่วนคือ ปัญหาขนาดข้อมูล 1 และปัญหาของขนาดข้อมูล $n-1$



Recursive Function : Multiply

```
#include <stdio.h>

int multiply(int,int);

int main(){
    int m, n;

    printf("Enter two integers: ");

    scanf("%d %d", &m, &n);

    printf("%d multiply %d is %d.\n", m, n, multiply(m,n));

    return 0;
}
```

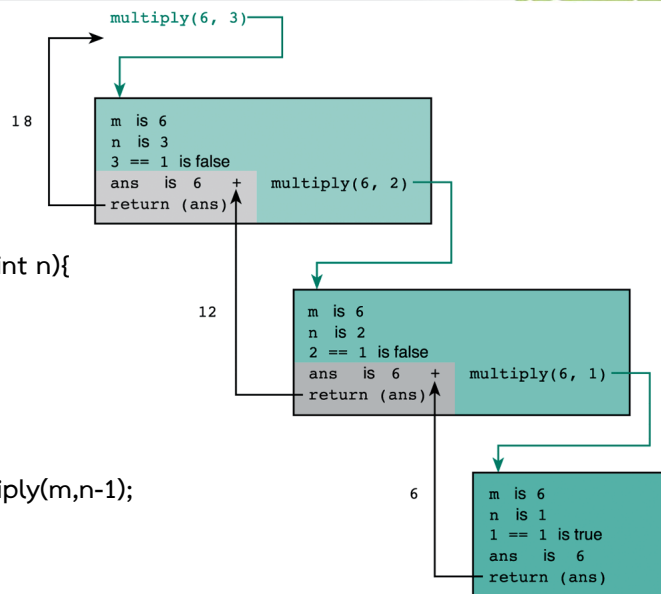
```
int multiply(int m, int n){
    int ans;

    if (n == 1)
        return m;
    else
        ans = m+multiply(m,n-1);
    return (ans);
}
```

Recursive Function : Multiply

```
int multiply(int m, int n){
    int ans;

    if (n == 1)
        return m;
    else
        ans = m+multiply(m,n-1);
    return (ans);
}
```



Terminating Condition

- Recursion functions จะต้องมีเงื่อนไขหยุดการทำงานอย่างน้อย 1 เงื่อนไขเสมอ โดยปกติ จะเป็นเงื่อนไขของกรณีพื้นฐาน simple case
- หากไม่มี terminating condition ฟังก์ชันอาจทำงานไม่หยุดเลย เช่น ฟังก์ชันการคูณ if statement “if (n == 1) ...” คือ terminating condition.

How to Trace Recursive Functions

- การ trace และ debug ฟังก์ชัน recursive ไม่ง่าย !!!
- เช่น ถ้ามีการเรียกตัวเองนับร้อยครั้ง การ set break point หรือการ trace แบบ step-by-step จะน่าเหนื่อยมาก
- วิธีการที่นิยมคือ การเพิ่มคำสั่ง print เข้าไปเพื่อดูผลของการทำงานในแต่ละ recursive steps.

```
multiply(int m, int n)
{
    int ans;

    printf("Entering multiply with m = %d, n = %d\n", m, n);

    if (n == 1)
        ans = m;      /* simple case */
    else
        ans = m + multiply(m, n - 1); /* recursive step */
}
```

Recursive Function : Factorial

```
#include <stdio.h>

long factorial(long);

int main(){
    int n;

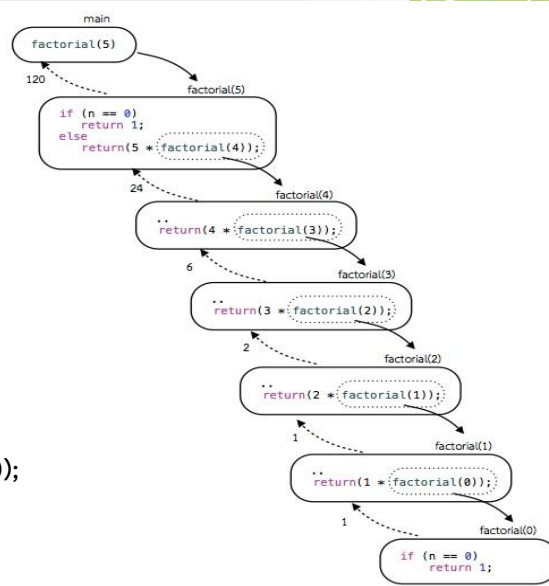
    printf("Enter integers: ");
    scanf("%d",&n);
    printf("%2d! = %ld\n",n,factorial(n));

    return 0;
}
```

```
long factorial(long n){
    if (n == 0)
        return 1;
    else
        return(n*factorial(n-1));
}
```

Recursive Function : Factorial

```
long factorial(long n){
    if (n == 0)
        return 1;
    else
        return(n*factorial(n-1));
}
```

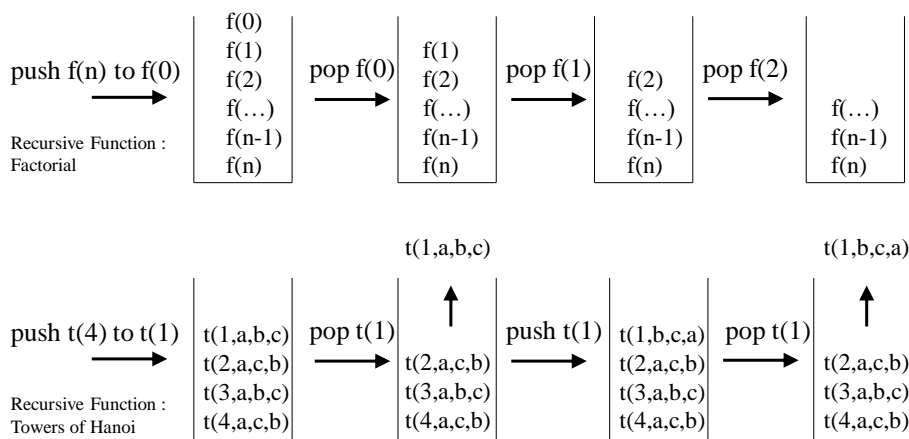


อ้างอิงภาพจาก <http://macfeteria.com/wp-content/uploads/2013/06/recursive.jpg>

Recursive Steps

- ภาษา C เก็บรักษาค่าตัวแปรโดยใช้ Stack
- Stack คือ โครงสร้างข้อมูลที่จัดการข้อมูลในรูปแบบ Last In, First Out (LIFO)
- Stack มี 2 operations คือ push และ pop
- ทุกครั้งที่มีการเรียกฟังก์ชัน สถานะปัจจุบันของผู้เรียก (caller function) ได้แก่ parameters, local variables, and memory address เป็นต้น จะถูก push ลงใน stack
- เมื่อฟังก์ชันที่ถูกเรียกประมวลผลเสร็จ ค่าสถานะต่าง ๆ ของผู้เรียก (caller function) จะถูกนำกลับมาประมวลผลโดยการ pop จาก stack.

Recursive Steps



มีการ push และ pop ฟังก์ชัน t(1) จำนวน 2 ครั้ง เนื่องจาก ฟังก์ชัน t(2) มีการเรียก t(1) 2 ครั้ง ซึ่งถ้าดูต่อไปก็จะพบว่า t(2) ถูก push และ pop โดยฟังก์ชัน t(3) ด้วย

Recursive Function : Fibonacci

- ลำดับฟีโบนัชชี คือจำนวนต่าง ๆ ที่อยู่ในลำดับจำนวนเต็มดังต่อไปนี้
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...
- ชื่อของจำนวนฟีโบนัชชีตั้งขึ้นเพื่อเป็นเกียรติแก่นักคณิตศาสตร์ชาวที่มีชื่อเสียงอิตาลี ชื่อ เลโอนาร์โดแห่งปิซา (Leonardo de Pisa) ซึ่งเป็นที่รู้จักกันในนามฟีโบนัชชี (Fibonacci) ผู้ค้นพบจำนวนฟีโบนัชชีในต้นศตวรรษที่ 13
- หากเขียนให้อยู่ในรูปของสัญลักษณ์ ลำดับ F_n ของจำนวนฟีโบนัชชี นิยามขึ้นด้วยความสัมพันธ์เวียนเกิด ดังนี้

$$F_n = F_{n-1} + F_{n-2}$$

โดยกำหนดค่าเริ่มแรกให้ $F_0 = 0 ; F_1 = 1$

3	2	
	1	1
5		8

Recursive Function : Fibonacci

```
#include <stdio.h>
```

```
long fib(long);
```

```
int main(){
```

```
    int n;
```

```
    printf("Enter integers: ");
```

```
    scanf("%d",&n);
```

```
    printf("F[%2d] = %ld
```

```
    \n",n,fib (n));
```

```
    return 0;
```

```
}
```

```
long fib(long n){
```

```
    int ans;
```

```
    if (n == 1 || n == 2)
```

```
        return 1;
```

```
    else
```

```
        ans = fib(n-1)+fib(n-2);
```

```
    return (ans);
```

```
}
```

Recursive Function : gcd

- หาร่วมมาก (ห.ร.ม) (อังกฤษ: greatest common divisor : gcd) คือ จำนวนเต็มที่ยิ่งใหญ่ที่สุดที่สามารถนำไปหารจำนวนตั้งแต่สองจำนวนขึ้นไปพร้อมกันได้ลงตัวทั้งหมด
- คูณร่วมน้อย (ค.ร.น) (อังกฤษ: least common multiple : lcm) คือ จำนวนเต็มที่ยิ่งน้อยที่สุดที่สามารถหารด้วยจำนวนตั้งแต่สองจำนวนขึ้นไปได้ลงตัวทั้งหมด
- ยูคลิด (Euclid) เป็นนักคณิตศาสตร์ชาวกรีก ซึ่งมีชีวิตอยู่ประมาณ 325 – 265 ปีก่อนคริสต์ศักราช ได้กล่าวถึงการหารร่วมมาก หรือ ห.ร.ม. ของจำนวนนับสองจำนวน ที่มีค่ามากได้อย่างรวดเร็วด้วยวิธีที่เรียกว่า ขั้นตอนวิธีแบบยูคลิด

ขั้นตอนวิธีแบบยูคลิด

ถ้าต้องการหา ห.ร.ม ของ 1500 และ 2050

ขั้นตอนที่ 1 นำตัวเลขที่มีค่าน้อยหารตัวเลขที่มีค่ามาก คือ $2050 \div 1500$ ได้เศษ 550

ขั้นตอนที่ 2 นำเศษที่ได้จากการหารรอบแรกมาหารตัวหารตัวแรก คือ $1500 \div 550$ ได้เศษ 400

ขั้นตอนที่ 3 ทำซ้ำ ขั้นตอนที่ 2 โดยนำเศษที่ได้จากการหารรอบก่อนหน้ามาหารตัวหารก่อนหน้าเช่นกัน คือ $550 \div 400$ ได้เศษ 150

ขั้นตอนที่ 4 นำ 150 มาหารตัวหาร คือ $400 \div 150$ ได้เศษ 100

ขั้นตอนที่ 5 นำ 100 มาหารตัวหาร คือ $150 \div 100$ ได้เศษ 50

ขั้นตอนที่ 6 นำ 50 มาหารตัวหาร คือ $100 \div 50$ ได้เศษ 0

เมื่อทำการหารไปเรื่อย ๆ จนได้เศษ = 0 หรือเรียกว่าหารลงตัวนั่นเอง ตัวหารสุดท้ายที่ได้คือคำตอบของ ห.ร.ม ซึ่งในที่นี้ตัวสุดท้าย คือ 50

Recursive Function : gcd

```
#include <stdio.h>


int GCD(int m, int n);

int main()
{
    int m, n;
    printf("Enter two integers: ");
    scanf("%d %d", &m, &n);
    printf("G.C.D of %d and %d is
%d.\n", m, n, GCD(m,n));
    return 0;
}
```

```
int GCD(int m, int n)
{
    if (m%n == 0)
        return n;
    else
        GCD(n, m%n);
}
```

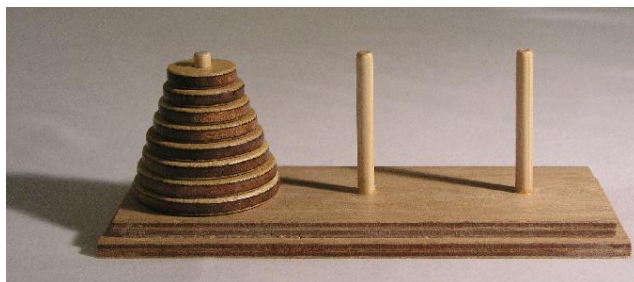
การหา ค.ร.น. โดยวิธียุคลิด

- เมื่อกำหนด a และ b เป็นจำนวนนับสองจำนวน ห.ร.ม. ของ a และ b คูณ ค.ร.น. ของ a และ b จะเท่ากับ $a \times b$ ดังนั้น อาจหา ค.ร.น. ของ a และ b โดยใช้สูตรต่อไปนี้

$$\text{ค.ร.น. ของ } a \text{ และ } b = \frac{a \times b}{\text{ห.ร.ม. ของ } a \text{ และ } b}$$


Towers of Hanoi

- ทาวเวอร์ออฟฮานอย (Tower of Hanoi) เป็นเกมคณิตศาสตร์ ประกอบด้วยหมุด 3 แท่ง และ จานกลมแบนขนาดต่าง ๆ ซึ่งมีรูตรงกลางสำหรับให้หมุด
- เป้าหมายของเกมคือ พยายามย้ายกองจานทั้งหมดไปไว้ที่อีกหมุดหนึ่ง โดยการเคลื่อนย้ายจานจะต้องเป็นไปตามกติกาคือ
- สามารถย้ายจานได้เพียงครั้งละ 1 ใบ
- ไม่สามารถวางจาน ไว้บนจานที่มีขนาดเล็กกว่าได้



Towers of Hanoi

ขั้นตอนวิธีเวียนเกิด

- ตั้งชื่อหมุดทั้งสาม A,B,C
- สมมุติมีจานทั้งหมดจำนวน n ใบ
- ติดเบอร์ให้กับจานจากเล็กที่สุดให้เป็น “จาน 1” ไปจนถึงใหญ่ที่สุดคือ “จาน n ”

ต้องการย้ายจานทั้งหมดจำนวน n ใบ จากหมุด A ไปยังหมุด C

- หากย้ายจานจำนวน $n-1$ ใบจาก A ไปไว้ที่ B ก่อน จะทำให้เหลือ “จาน n ” เพียงใบเดียวที่หมุด A
- ย้าย “จาน n ” จาก A ไปไว้ที่ C
- ย้ายจาน $n-1$ ใบจาก B ไปที่ C ซึ่งจานทั้งหมดจะอยู่บน “จาน n ”

Towers of Hanoi

- จำนวนครั้งในการเคลื่อนย้ายจาน n ใบ เพื่อแก้ปัญหา มีจำนวนครั้งเท่ากับ $2^n - 1$



คำตอบสำหรับจาน 4 ใบ



คำตอบสำหรับจาน 3 ใบ

Towers of Hanoi

```
void tower(int n, char fr, char tr, char ar){
    if (n == 1){
        printf("\n Move disk 1 from %c to %c", fr, tr);
        return;
    }
    tower(n - 1, fr, ar, tr);
    printf("\n Move disk %d from %c to %c", n, fr, tr);
    tower(n - 1, ar, tr, fr);
}
```

Towers of Hanoi

```
#include <stdio.h>

void tower(int, char, char, char);

int main(){
    int n;

    printf("Enter the number of disks : ");
    scanf("%d", &n);

    printf("The sequence of moves involved in
the Tower of Hanoi are :\n");

    tower(n, 'A', 'C', 'B');

    return 0;
}

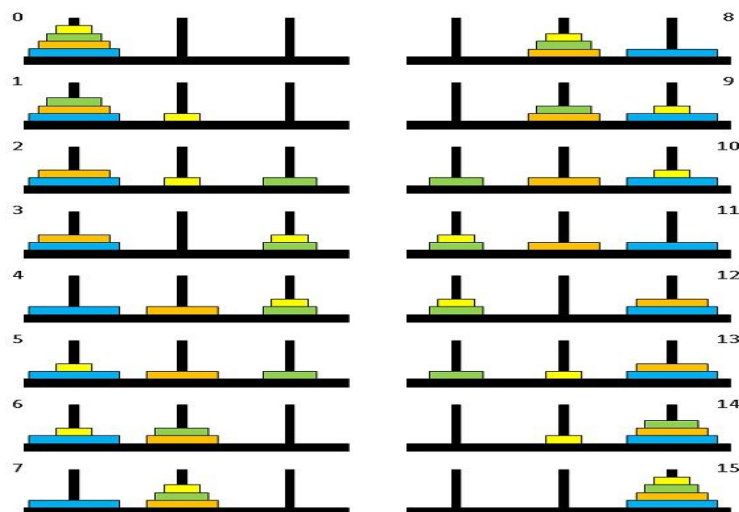
void tower(int n, char fr, char tr, char ar){
    if (n == 1){
        printf("\n Move disk 1 from %c to
%c", fr, tr);
        return;
    }
    tower(n - 1, fr, ar, tr);
    printf("\n Move disk %d from %c to
%c", n, fr, tr);
    tower(n - 1, ar, tr, fr);
}
```

Towers of Hanoi

```
C:\Users\phit-apha\Documents\hanoi.exe
Enter the number of disks : 4
The sequence of moves involved in the Tower of Hanoi are :

Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
```

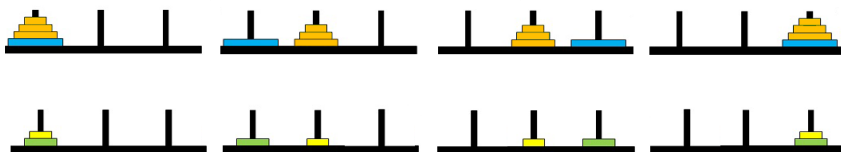
Towers of Hanoi



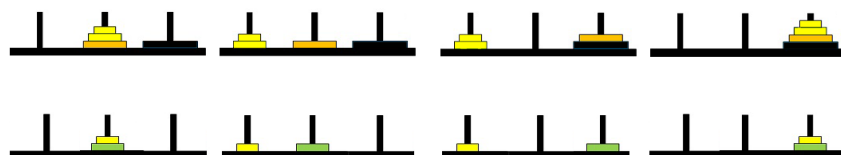
อ้างอิงภาพ : เกียรติหน้า , <https://pantip.com/profile/268499>

Towers of Hanoi

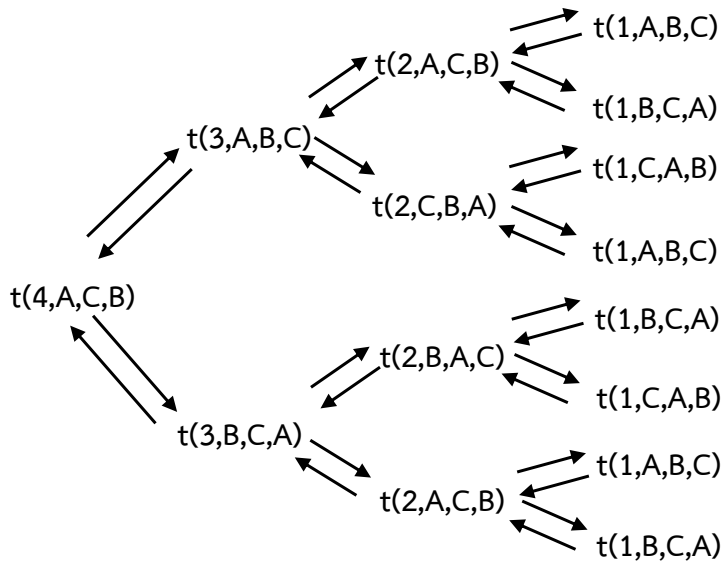
เราสามารถมองปัญหาของ n จาน แบบเดียวกับ ปัญหา 2 จาน คือ ย้ายจานจำนวน $n-1$ ใบจาก A ไป B , ย้ายจานใบที่ n จาก A ไป C , ย้ายจาน $n-1$ ใบจาก B ไป C



ในกรณีเมื่อจานใบที่ n ไปถึงจุดหมายแล้วก็ไม่ต้องคำนึงถึงจานใบที่ n อีกต่อไป จากนั้นเราสามารถมองปัญหาของ $n-1$ จาน แบบเดียวกับ ปัญหา 2 จาน แต่ ตำแหน่งเสาเริ่มต้นเปลี่ยนไป ทำให้ต้องตั้งค่าตำแหน่งเสาใหม่ก่อนคิดด้วย



Towers of Hanoi



แหล่งข้อมูล

- Recursion, ดร.อัศรา ประโยชน์ ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ
- Math M.1 - Rinda, นางรินดา กรุดเนียม โรงเรียนบรรหารแจ่มใสวิทยา 6 อำเภอสามชูก จังหวัดสุพรรณบุรี
- หอคอยฮานอย, <https://th.wikipedia.org/wiki/%E0%B8%AB%E0%B8%AD%E0%B8%84%E0%B8%AD%E0%B8%A2%E0%B8%AE%E0%B8%B2%E0%B8%99%E0%B8%AD%E0%B8%A2>
- ภาษาซีฉบับภาษาชาวบ้าน , กวินวิษณุ พุ่มสาขา ศูนย์เทคโนโลยีเพื่อการเรียนการสอน โรงเรียนสตรีอ่างทอง