江西理工大学 Jiangxi University of Science and Technology
信息工程学院 School of information engineering

高级算法分析与设计

Advanced Algorithm Analysis and Design

Lecture 07:

Recursion Algorithm (B)

Dr Ata Jahangir Moshayedi

EMAIL: ajm@jxust.edu.cn

Prof Associate ,
School of information engineering Jiangxi university of science and technology, China

LIVE Lecture series

Autumn _2022

高级算法分析与设计

Advanced Algorithm Analysis and Design

LECTURE 07: Recursion Algorithm(B)

Agenda

目录

CONTENTS

EXAMPLE

高级算法分析与设计

Advanced Algorithm Analysis and Design

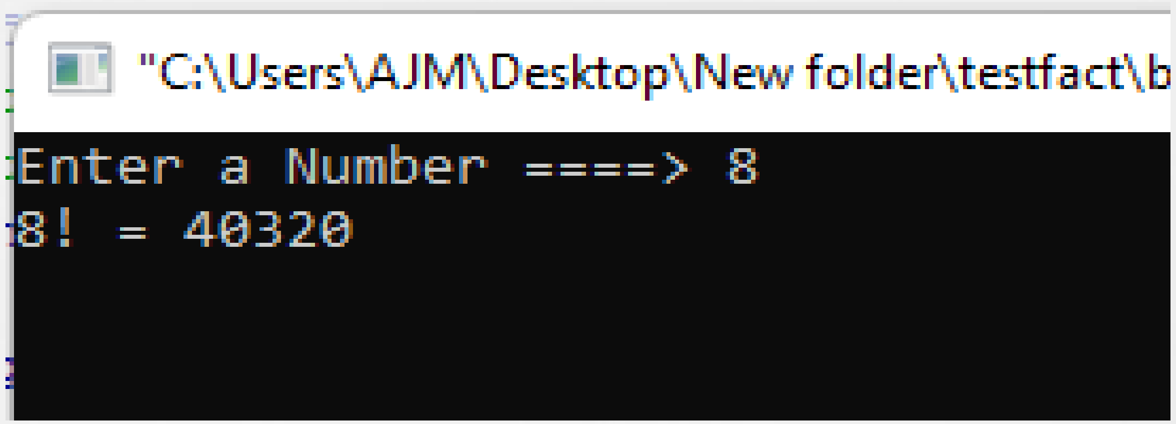
Example 1: Factorial

Factorial_1

```
// fact
#include<iostream>
#include<conio.h>
using namespace std;

long Fact(int n)
{
		if (n <= 1)
		 return 1;
		else
		 return n*Fact(n - 1);
}

int main()
{
		int n;
		cout << "Enter a Number ====> ";
		cin >> n;
		cout << n << "! = " << Fact(n);
		_getch();
		return 0;
}
```



"C:\Users\AJM\Desktop\New folder\testfact\b

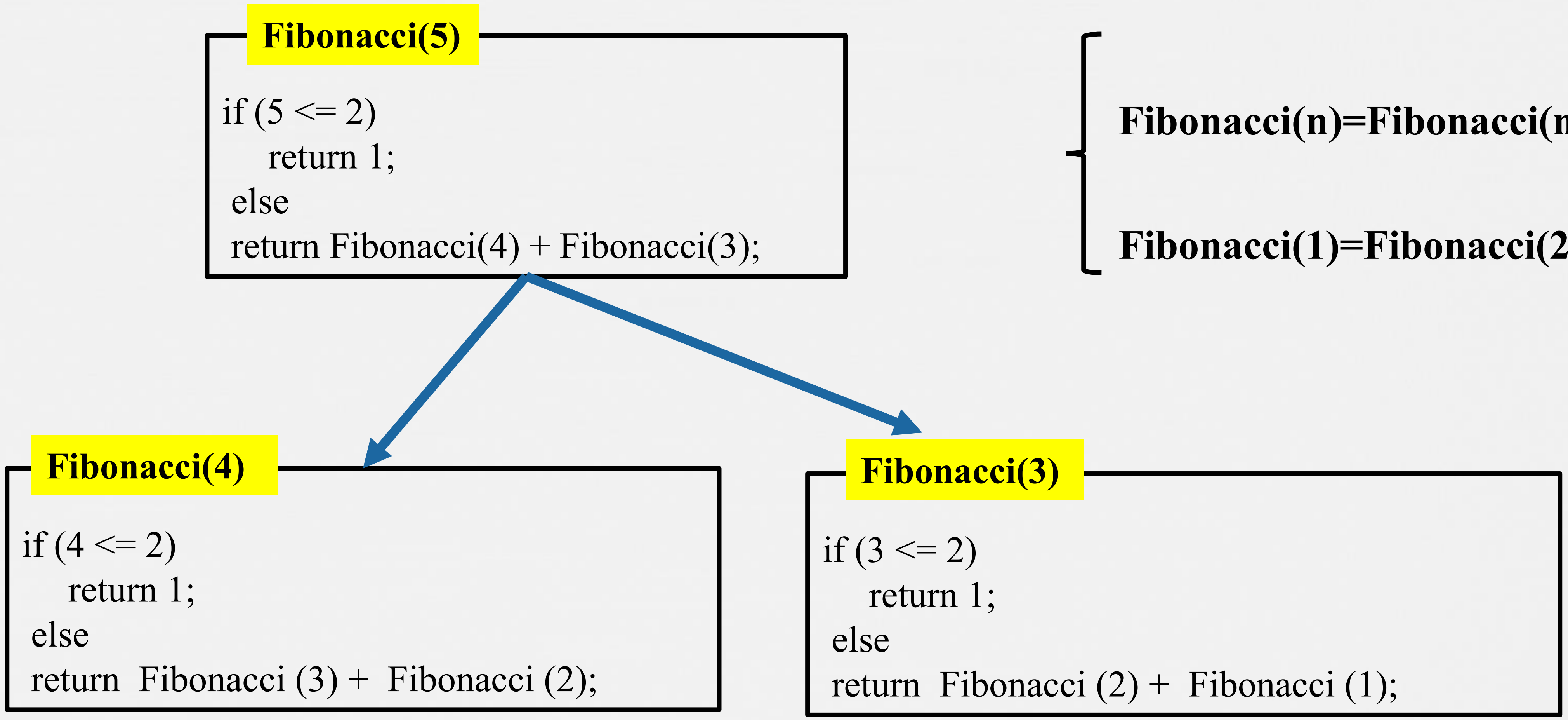
```
Enter a Number ====> 8
8! = 40320
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 2: Fibonacci

Fibonacci



Fibonacci(5)

```
if (5 <= 2)
    return 1;
 else
 return Fibonacci(4) + Fibonacci(3);
```

Fibonacci(4)

```
if (4 <= 2)
    return 1;
 else
 return  Fibonacci (3) +  Fibonacci (2);
```

Fibonacci(3)

```
if (3 <= 2)
    return 1;
 else
 return  Fibonacci (2) +  Fibonacci (1);
```

$$\begin{cases} \textbf{Fibonacci(n)=Fibonacci(n-1)+Fibonacci(n-2)} \\ \textbf{Fibonacci(1)=Fibonacci(2) =1} \end{cases}$$

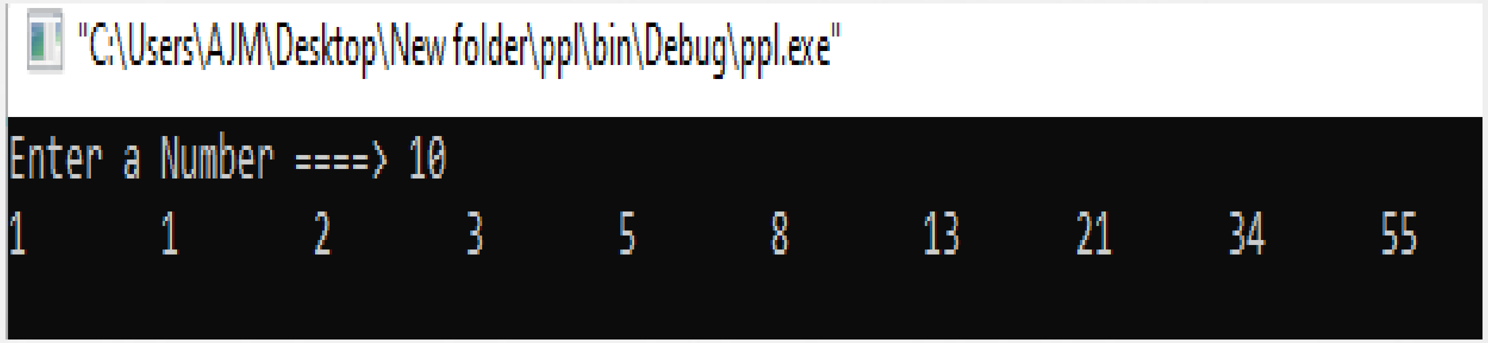
$$\begin{tikzcd}
 & & & & \mathrm{fib}(4) \arrow[dll] \arrow[dr] & & \\
 & & \mathrm{fib}(3) \arrow[dl] \arrow[dr] & & & \mathrm{fib}(2) \arrow[dl] \arrow[dr] & \\
 & \mathrm{fib}(2) \arrow[dl] \arrow[dr] & & \mathrm{fib}(1) & \mathrm{fib}(1) & & \mathrm{fib}(0) \\
\mathrm{fib}(1) & & \mathrm{fib}(0) & & & &
\end{tikzcd}$$

Fibo_01

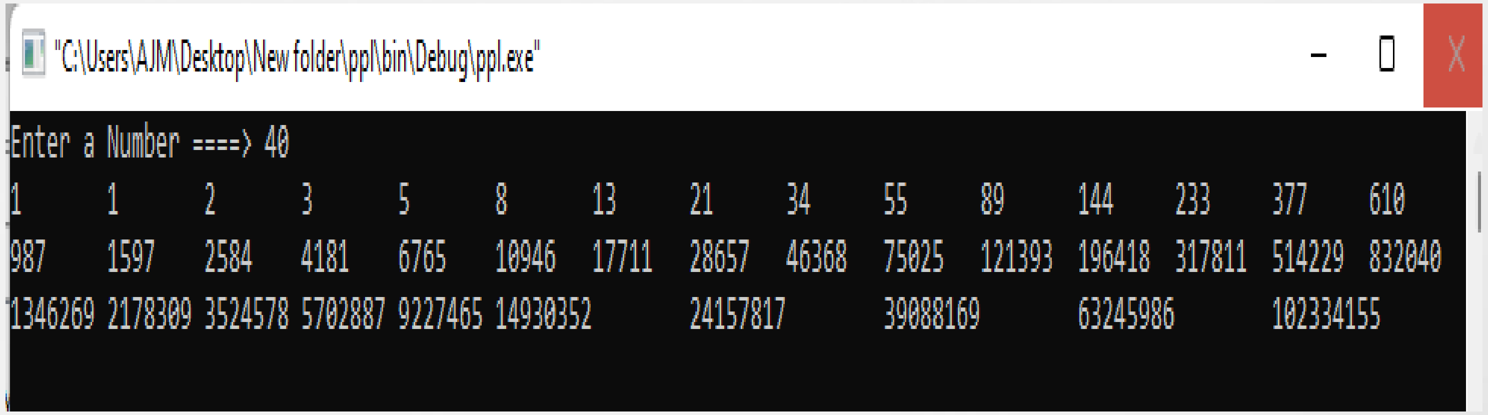
```
#include<iostream>
#include<conio.h>

using namespace std;

int Fiboo(int n)
{
            if (n <= 2)
            return 1;
            else
            return Fiboo(n - 1) + Fiboo(n - 2);
}
int main()
{
            int i,n;
            cout << "Enter a Number ====> ";
            cin >> n;
            for (i = 1; i <= n;i++)
            cout << Fiboo(i)<<"\t";
            _getch();
            return 0;
}
```

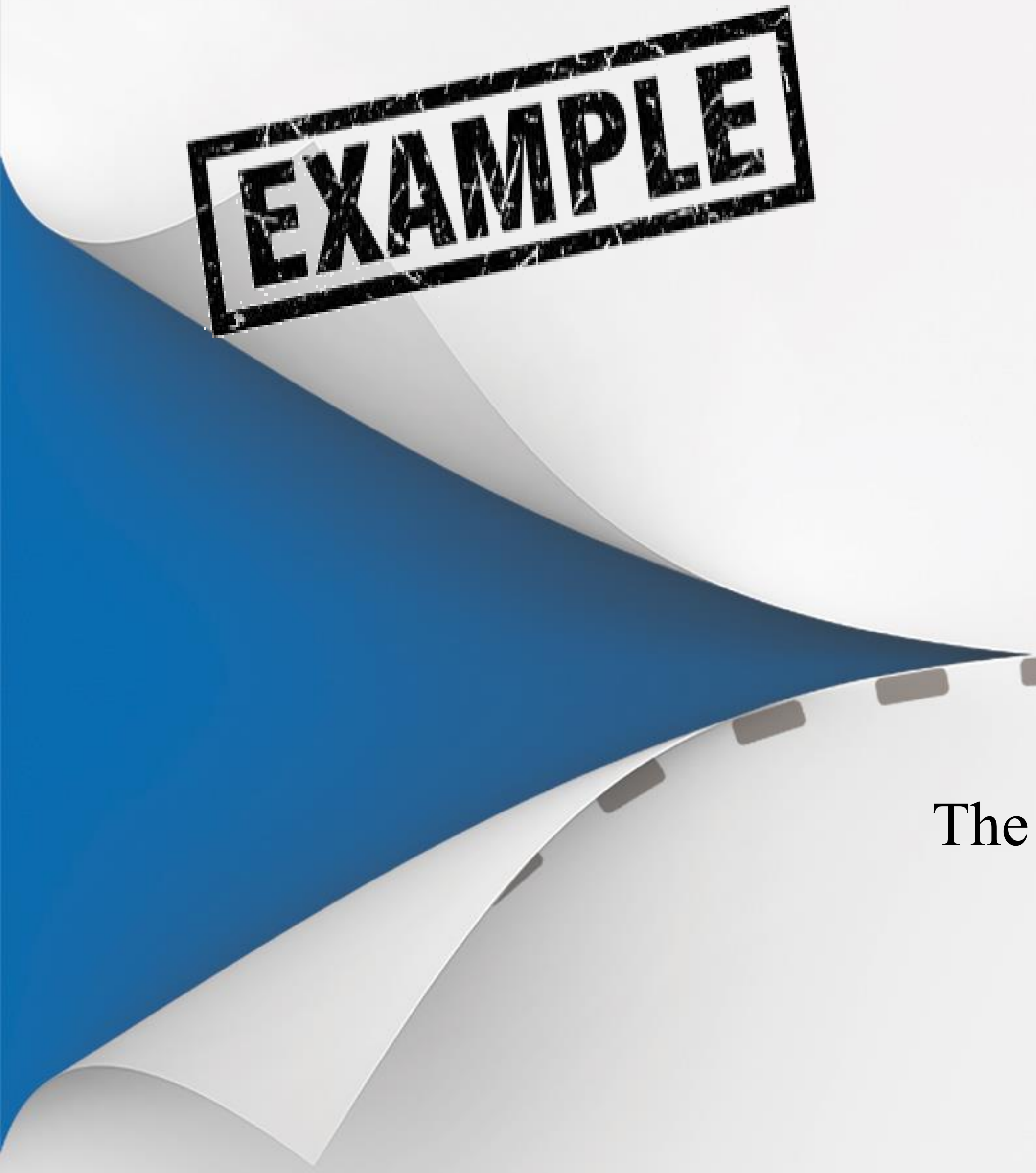


```
"C:\Users\AJM\Desktop\New folder\ppl\bin\Debug\ppl.exe"
Enter a Number ====> 10
1       1       2       3       5       8       13      21      34      55
```



```
"C:\Users\AJM\Desktop\New folder\ppl\bin\Debug\ppl.exe"
Enter a Number ====> 40
1       1       2       3       5       8       13      21      34      55      89      144     233     377     610
987     1597    2584    4181    6765    10946   17711   28657   46368   75025   121393  196418  317811  514229  832040
1346269 2178309 3524578 5702887 9227465 14930352        24157817        39088169        63245986        102334155
```

Have the delay for the last values for e.g for 40th should calculate more and pause the program



高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 3:

The greatest common divisor of two numbers in recursive

江西理工大學

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

The greatest common divisor of two numbers

```
int BMM(int a, int b){
        if (a%b == 0)
          return b;
        else
        return BMM(b, a%b);
}
```

```
BMM(640, 96)
if (640% 96 == 64==0)
   return b;
  else
 return BMM(96,64);
```

```
BMM(96, 64)
if (96% 64 == 32==0)
   return b;
  else
 return BMM(64,32);
```

```
BMM(640, 96)
if (64% 32 == 0==0)
   return b=32;
  else
 return BMM(32,0);
```

Example 3:
The greatest common divisor of two numbers in recursive

```
// ConsoleApplication3.cpp : Defines the entry point for the console application.

#include<iostream>
#include<conio.h>

using namespace std;

int BMM(int a, int b)
{
		if (a%b == 0)
			return b;
		else
			return BMM(b, a%b);
}

int main()
{
		cout << BMM(960, 260);
		_getch();
    return 0;
}
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 4: Segment algorithm

Example 4: Segment algorithm

```
// ConsoleApplication4.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include<iostream>
#include<conio.h>

using namespace std;

void Tajzeh(int n, int b)
{
if (n > 1)
if (n%b == 0) {
cout << endl << n << "\t" << b;
Tajzeh(n / b, b);
                                        }
else
Tajzeh(n, b + 1);
}
int main(){
Tajzeh(2000, 2);
_getch();
    return 0;
}
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 5: nested loop algorithm

Example 5: nested loop algorithm

```
#include<iostream>
Void main(void){
      Int i,j,n;
      cout <<“Enter number:”;
       cIn>>n;
         for(i=1;i<=n;i++){
            for(j=I ;j>=1;j--){
              cout << j<<“t”;
            cout << endl;
}
```

高级算法分析与设计

Advanced Algorithm Analysis and Design

Example 6: Printing the triangle Number pattern

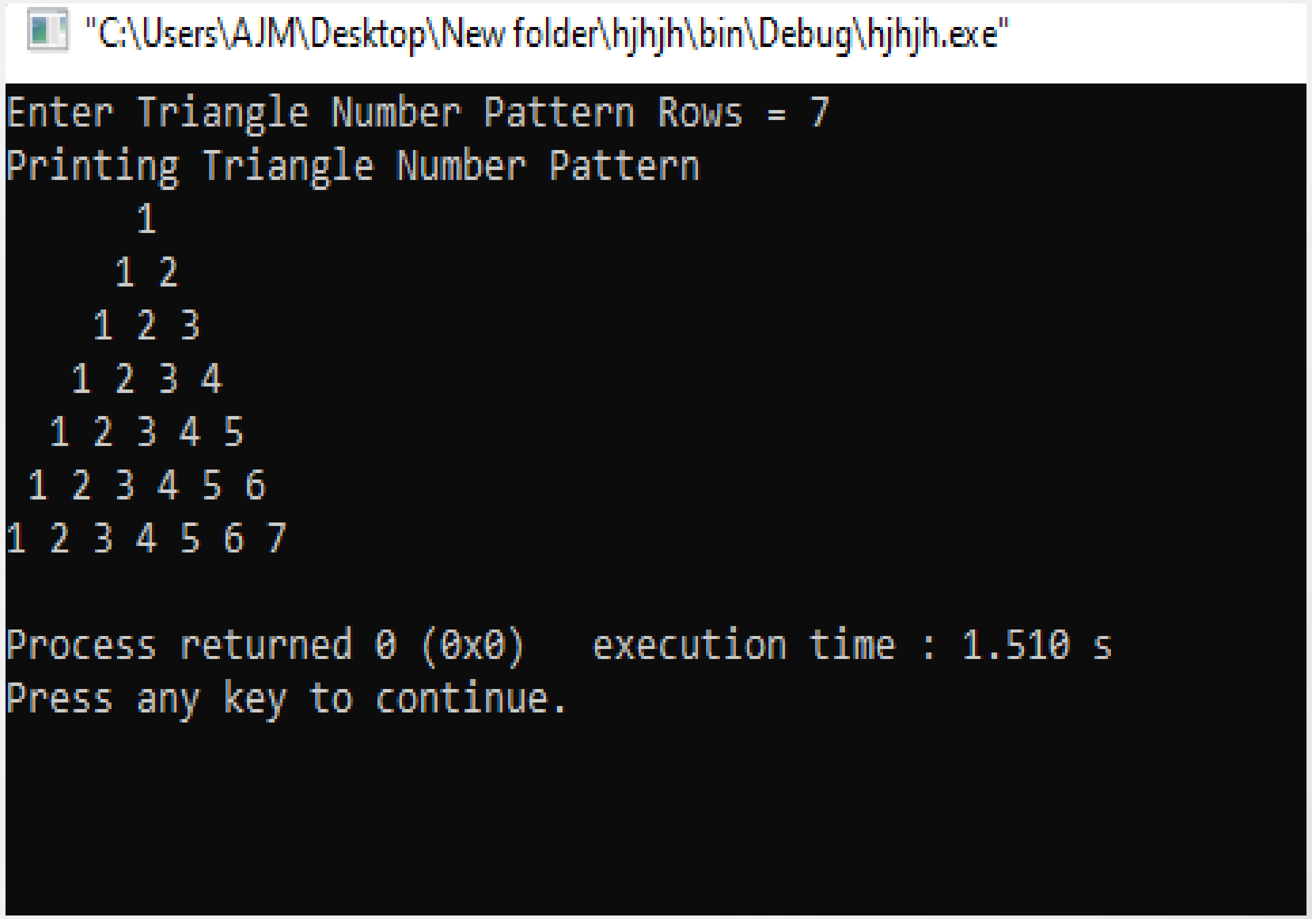
Printing the triangle Number pattern -1

```
#include<iostream>
using namespace std;
int main()
{
        int rows;

        cout << "Enter Triangle Number Pattern Rows = ";
        cin >> rows;

        cout << "Printing Triangle Number Pattern\n";

        for (int i = 1; i <= rows; i++)
        {
                for (int j = rows; j > i; j--)
                {
                        cout << " ";
                }
                for (int k = 1; k <= i; k++)
                {
                        cout << k << " ";
                }
                cout << "\n";
        }
}
```



"C:\Users\AJM\Desktop\New folder\hjhjh\bin\Debug\hjhjh.exe"

```
Enter Triangle Number Pattern Rows = 7
Printing Triangle Number Pattern
      1
     1 2
    1 2 3
   1 2 3 4
  1 2 3 4 5
 1 2 3 4 5 6
1 2 3 4 5 6 7

Process returned 0 (0x0)   execution time : 1.510 s
Press any key to continue.
```

Printing the triangle Number pattern -2

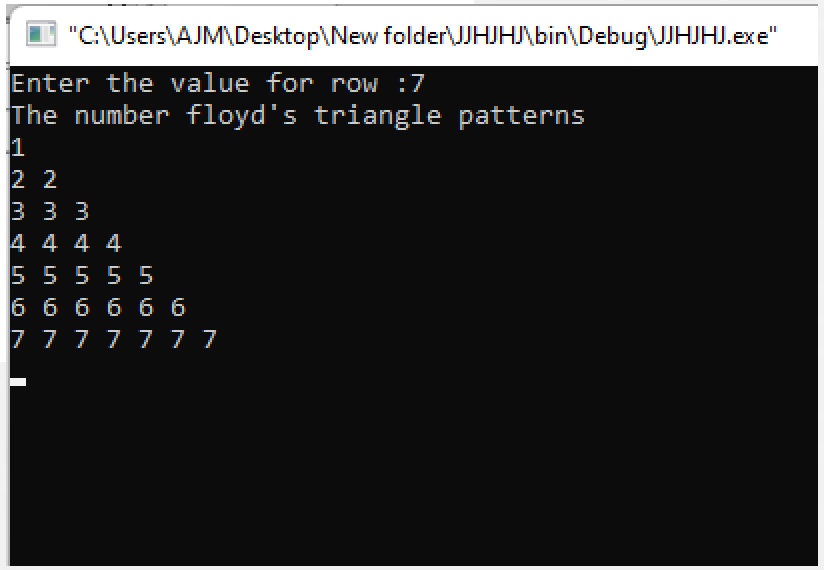
```
#include<iostream>
using namespace std;
int main()
{
int rows, i, j, k;
 cout << "Enter Rows = ";
 cin >> rows;
 cout << "Printing Triangle Number Pattern\n";
 i = 1;
 while (i <= rows){
   j = rows;
while (j > i){
cout << " ";
j--;
}
k = 1;
while (k <= i){
 cout << k << " ";
k++;
}
cout << "\n";
i++;
}
}
```

This program prints the triangle numbers pattern using a while loop.

Printing the triangle Number pattern -3

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
   int i,j,row;
   cout<<"Enter the value for row :";
   cin>>row;
   cout << "The number floyd's triangle patterns" << endl;
   for(i=1; i<=row; i++ ){
      for(j=1; j<=i; j++ ){
      cout<<i<<" ";
   }
   cout<< endl;
   }
   getch();
   return 0;
}
```

Another program out put format


"C:\Users\AJM\Desktop\New folder\JJHJHJ\bin\Debug\JJHJHJ.exe"

```
Enter the value for row :7
The number floyd's triangle patterns
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
```

高级算法分析与设计

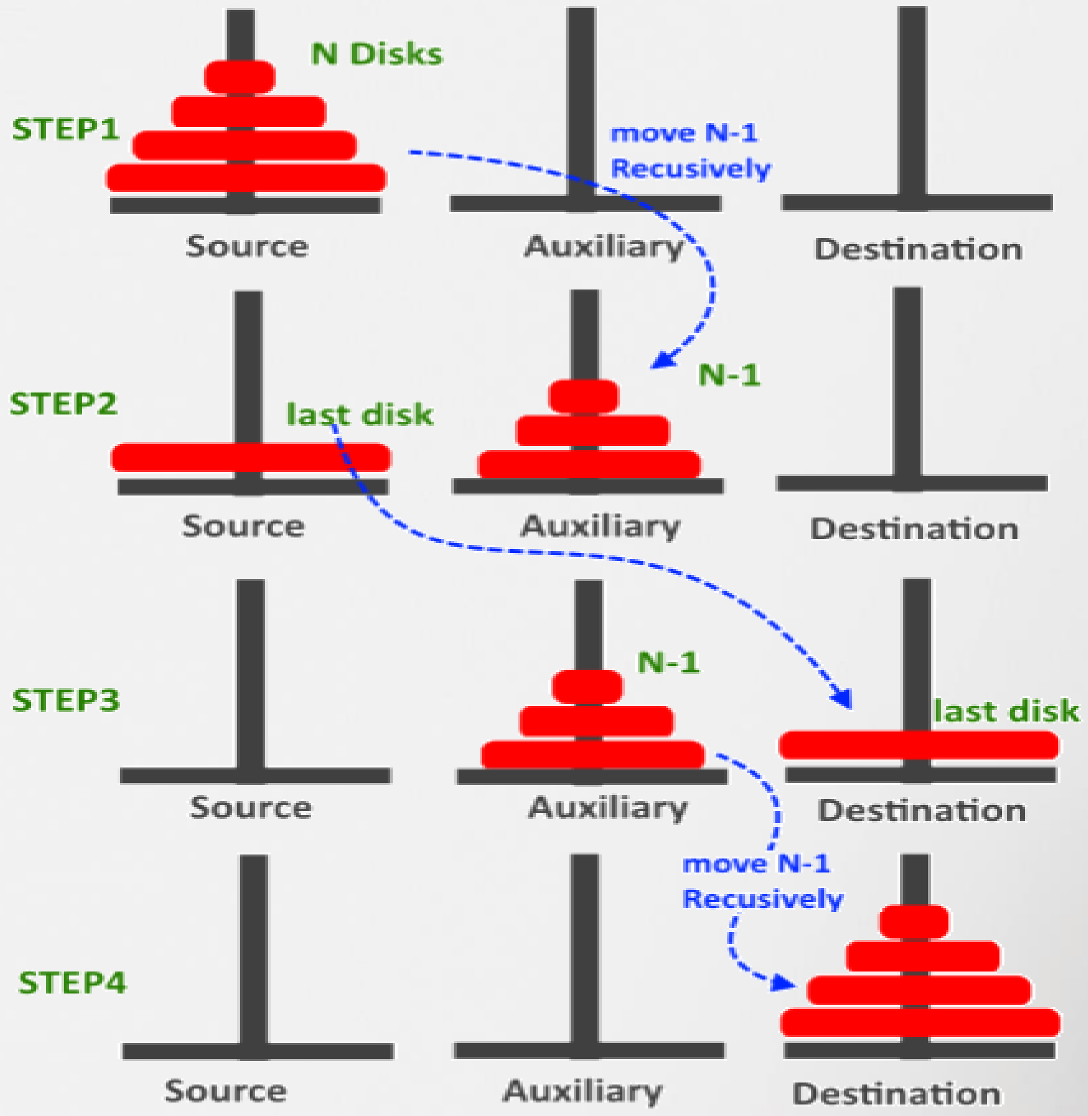
Advanced Algorithm Analysis and Design

Example 7:

Program for Tower of Hanoi Algorithm

Tower of Hanoi Algorithm

- Tower of Hanoi is a mathematical puzzle where we have three rods (**A**, **B**, and **C**) and **N** disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod **A**.
- The objective of the puzzle is to move the entire stack to another rod (here considered **C**), obeying the following simple rules:
- Only one disk can be moved at a time.
- 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.
- No disk may be placed on top of a smaller disk.



Tower of Hanoi Algorithm

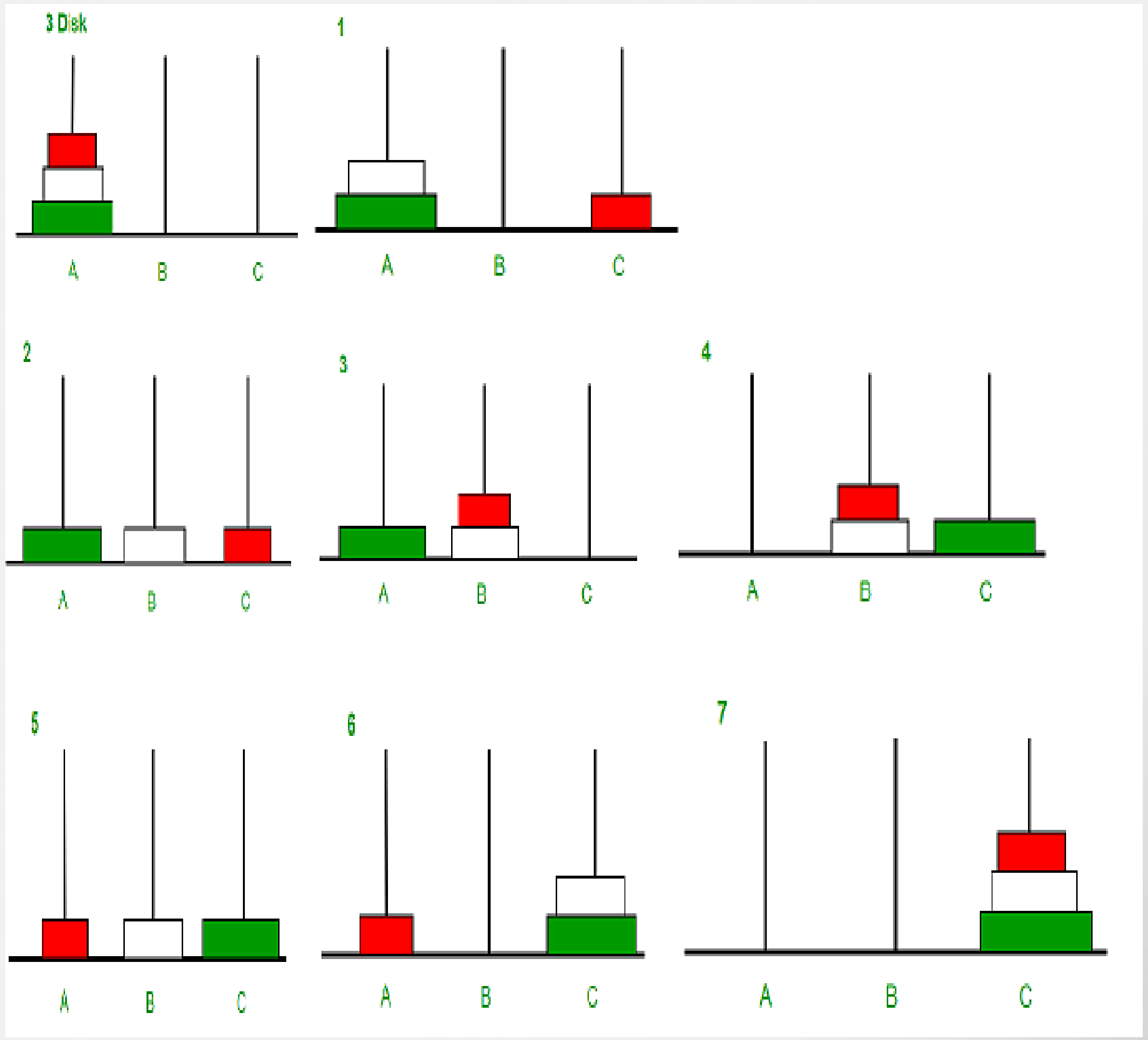
- **Approach:**
- Recursively Move N-1 disk from source to Auxiliary peg.
- Move the last disk from source to destination.
- Recursively Move N-1 disk from Auxiliary to destination peg.

- 方法:
- 递归地从源移动N-1个磁盘到辅助挂桩。
- 将最后一个磁盘从源移动到目标。
- 递归地从辅助设备移动N-1个磁盘到目标挂桩。

Tower of Hanoi using Recursion:

- The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:
  - Shift 'N-1' disks from 'A' to 'B', using C.
  - Shift last disk from 'A' to 'C'.
  - Shift 'N-1' disks from 'B' to 'C', using A.

- 其思想是使用递归使用helper节点到达目标。下面是这个问题的模式:
- 使用C将N-1个磁盘从"A"移到"B"。
- 把最后一张"A"盘移到"C"盘。
- 使用A将"N-1"磁盘从"B"移到"C"

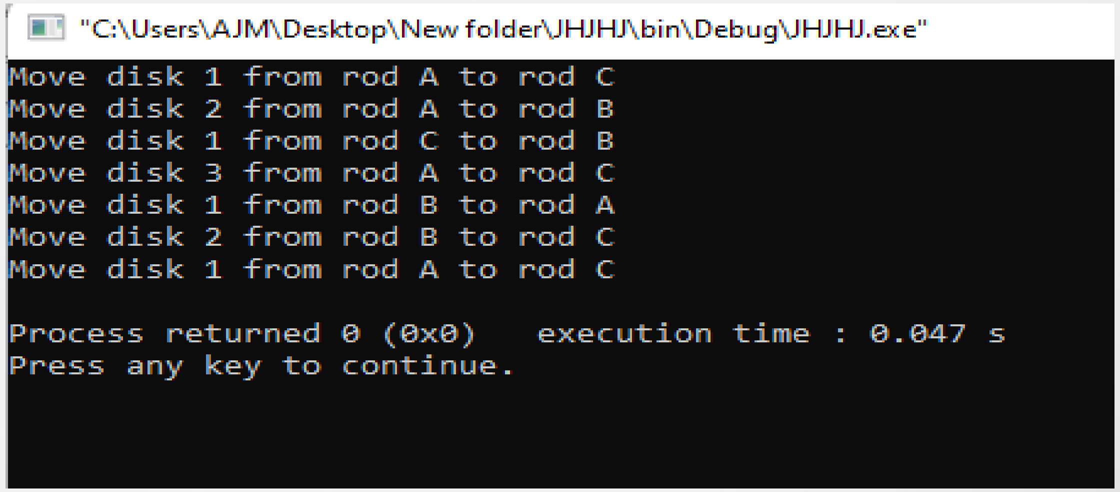


Tower of Hanoi using Recursion:

```
// solve tower of hanoi puzzle
#include <bits/stdc++.h>
using namespace std;

void towerOfHanoi(int n, char from_rod, char to_rod,char aux_rod)
{
        if (n == 0) {
        return;
        }
        towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
        cout << "Move disk " << n << " from rod " << from_rod
                << " to rod " << to_rod << endl;
        towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
// Driver code
int main()
{
        int N = 3;
        // A, B and C are names of rods
        towerOfHanoi(N, 'A', 'C', 'B');
        return 0;
}
```

- Follow the steps below to solve the problem:
  - Create a function **towerOfHanoi** where pass the **N** (current number of disk), **from_rod**, **to_rod**, **aux_rod.**
  - Make a function call for N – 1 th disk.
  - Then print the current the disk along with **from_rod** and **to_rod**
  - Again make a function call for N – 1 th disk.



"C:\Users\AJM\Desktop\New folder\JHJHJ\bin\Debug\JHJHJ.exe"

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

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Time complexity: O(2^N), There are two possibilities for every disk. Therefore, 2 * 2 * 2 * . . . * 2(N times) is 2^N
Auxiliary Space: O(N), Function call stack space

高级算法分析与设计
Advanced Algorithm Analysis and Design

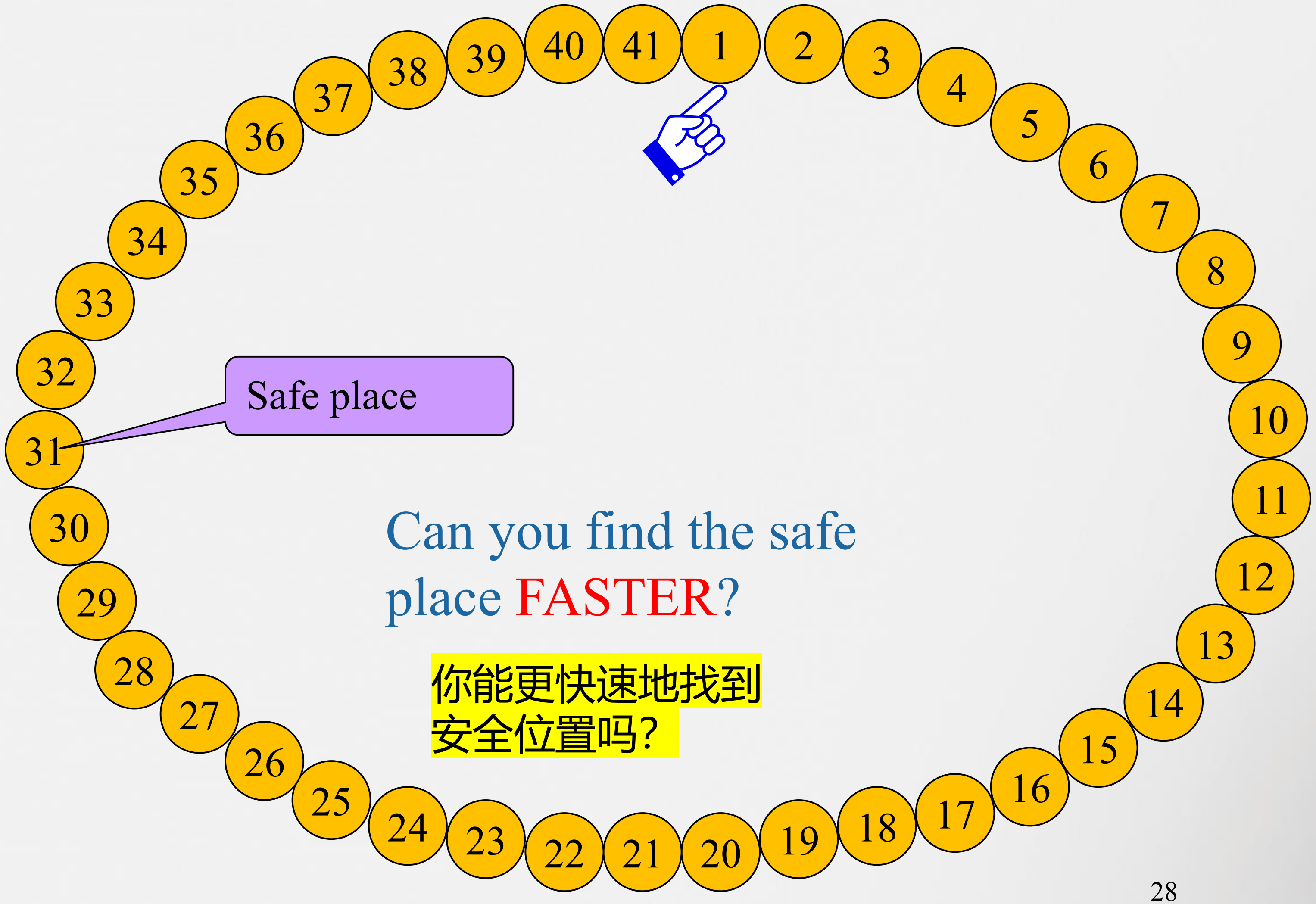
Example 8: **Joseph problem?**

约瑟夫问题?

Josephus problem
Can you find the safe place?

Flavius Josephus is a Jewish historian living in the 1st century. According to his account, he and his 40 comrade soldiers were trapped in a cave, surrounded by Romans. They chose suicide over capture and decided that they would form a circle and start killing themselves using a step of three. As Josephus did not want to die, he was able to find the safe place, and stayed alive with his comrade, later joining the Romans who captured them.

弗拉维乌斯·约瑟夫斯是生活在1世纪的犹太历史学家。根据他的描述，他和他的40名战友被困在一个山洞里，周围都是罗马人。他们选择了自杀而不是被捕，并决定围成一圈开始用三步制自杀。因为约瑟夫不想死，他找到了安全的地方，和他的同伴一起活了下来，后来加入了俘获他们的罗马人。



Can you find the safe place FASTER?

你能更快速地找到安全位置吗?

Josephus problem.

- There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed.
- The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of person n and a number k which indicates that k-1 persons are skipped and the kth person is killed in a circle.
- The task is to choose the place in the initial circle so that you are the last one remaining and so survive.
  - For example, if n = 5 and k = 2, then the safe position is 3. Firstly, the person at position 2 is killed, then the person at position 4 is killed, then the person at position 1 is killed. Finally, the person at position 5 is killed. So the person at position 3 survives.
  - If n = 7 and k = 3, then the safe position is 4.
  - The persons at positions 3, 6, 2, 7, 5, and 1 are killed in order, and the person at position 4 survives.

- 有n个人站成一圈等着被处决。从圆上的某一点开始计数，沿着固定的方向绕圆进行。在每个步骤中，跳过一定数量的人，然后执行下一个人。
- 消灭过程围绕着圆圈进行(随着被处决的人被移走，圆圈变得越来越小)，直到只剩下最后一个人，他被赋予了自由。给定总人数n和数字k这表示k-1个人被跳过第k个人被圆圈杀死。
- 任务是在最初的圆圈中选择一个地方，这样你就会是最后一个幸存下来的人。
  - 例如，如果n = 5, k = 2，那么安全位置是3。首先，2号位置的人被杀，然后4号位置的人被杀，然后1号位置的人被杀。最后，5号位置的人被杀。位置3的人存活下来。
  - 如果n = 7, k = 3，那么安全位置是4。
  - 位置3、6、2、7、5、1的人依次被杀，位置4的人幸存。

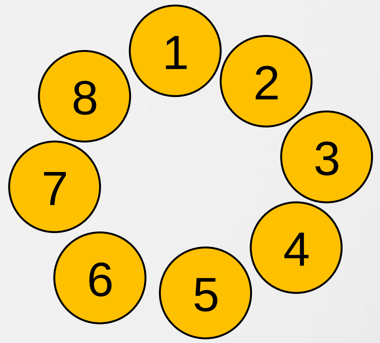
A simpler version

- Let's consider a similar problem 让我们考虑一个类似的问题
  - There are n person in a circle, numbered from 1 to n sequentially.

    一圈有n个人，按顺序从1到n编号。
  - Starting from the number 1 person, every 2nd person will be killed.

    从1号开始，每2个人就会被杀。
  - What is the safe place?

    什么叫做安全位置呢?
- The input is n, the output $f(n)$ is a number between 1 and n. 输入是n，输出是f（n），它是从1到n的数字
  - Ex: f(8) = ?



The first algorithm: simulation

- We can find *f*(*n*) using simulation. 我们可以用模拟的方法找到f(n)
  - Simulation is a process to imitate the real objects, states of affairs, or process.

    模拟是模拟真实物体、事物状态或过程的过程。
  - We do not need to “kill” anyone to find *f*(*n*).

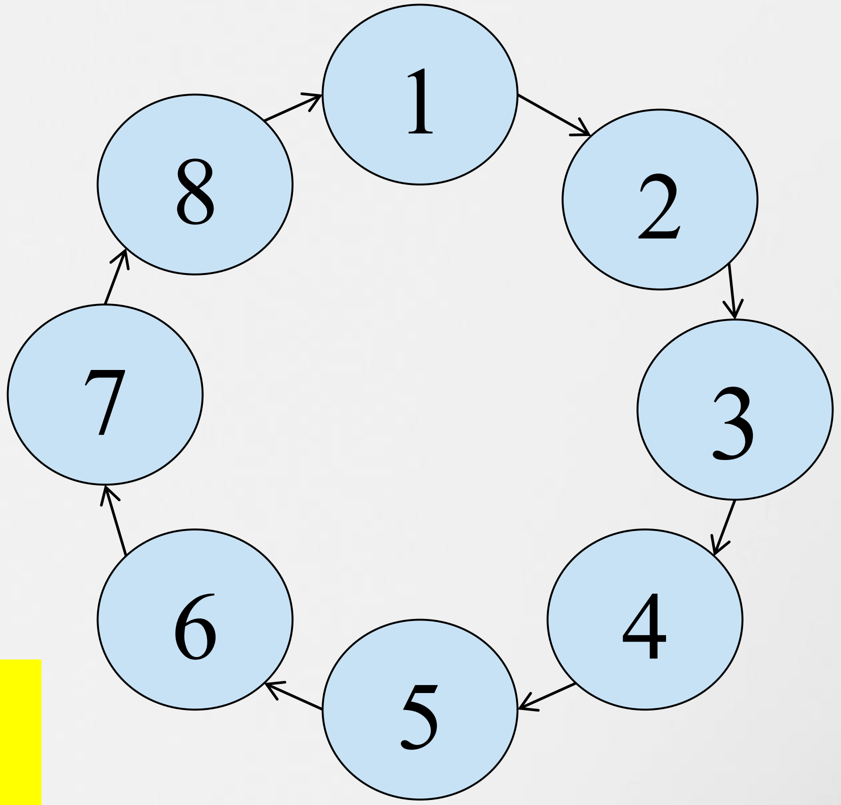
    我们不需要“杀死”任何人来找到f(n)。

- The simulation needs 模拟需求模拟需求
  - (1) a **model** to represents “n people in a circle” 表示“n人围一圈”的模型
  - (2) a way to **simulate** “kill every 2nd person” 一种模拟“杀死每2个人”的方法
  - (3) knowing when to stop 知道什么时候停止

Model *n* people in a circle

- We can use “data structure” to model it.
- 我们可以使用“数据结构”来建模。
- This is called a “circular linked list”.
- 这被称为“循环链表”。
  - Each node is of some “**struct**” data type
  - 每个结点都有一些“struct”数据类型
  - Each link is a “**pointer**”
  - 每个链接都是一个“指针”

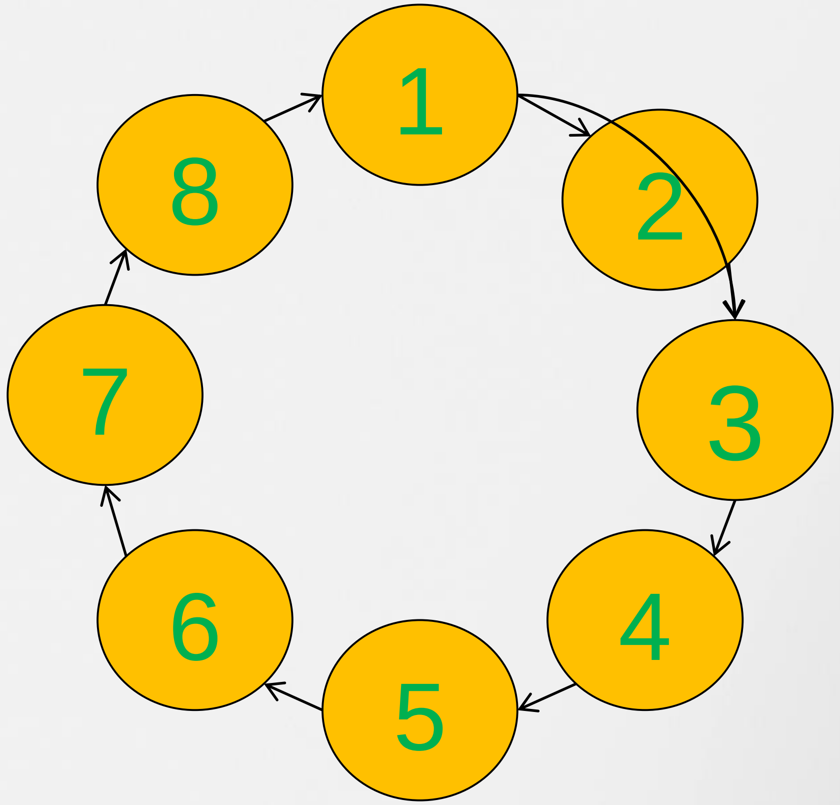
```
struct PIC {
    int ID;
    struct PIC *next;
}
```



$$\begin{tikzcd} & 8 \arrow[r] & 1 \arrow[r] & 2 \arrow[d] \\ 7 \arrow[ur] & & & 3 \arrow[d] \\ & 6 \arrow[ul] & 5 \arrow[l] & 4 \arrow[l] \end{tikzcd}$$

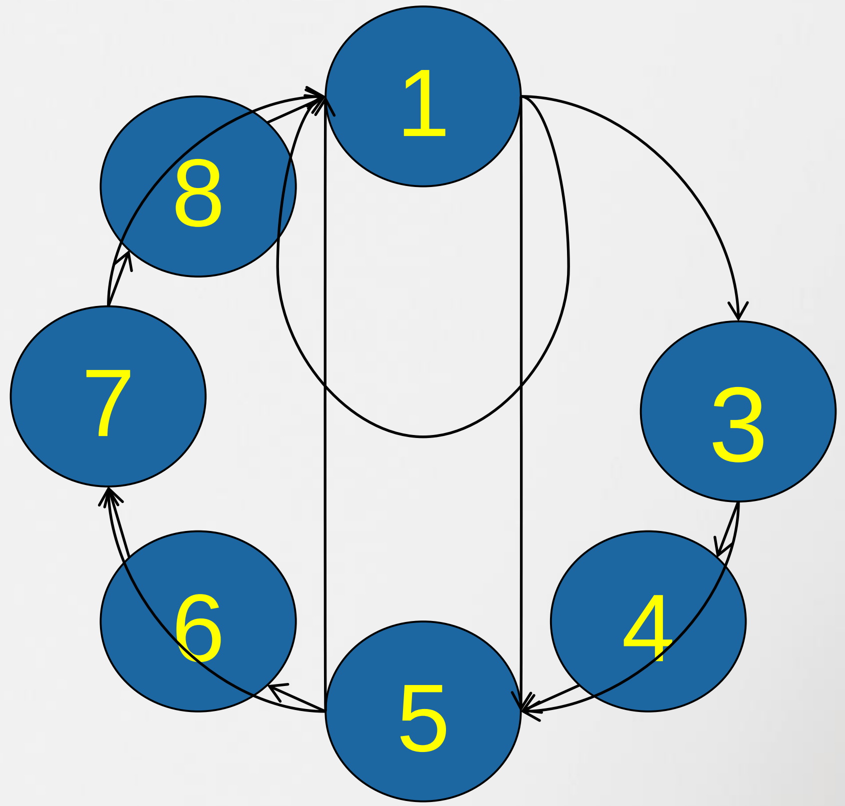
Kill every 2nd person

- Remove every 2nd node in the circular liked list.
- 删除循环列表中的每第二个节点。
  - You need to maintainthe circular linked structure after removing node 2
  - 移除节点2后，需要保持圆形链接结构
  - The process can continue until …
  - 这个过程可以持续到…



Knowing when to stop

- Stop when there is only one node left 当只剩下一个节点时停止
  - How to know that? 怎么知道什么时候只剩下一个节点呢?
  - When the *next is pointing to itself
  - 什么时候 *next指针指向它自己?
  - It's ID is f(n) 它的ID是f(n)
    - f(8) = 1



How fast can it compute f(n)?

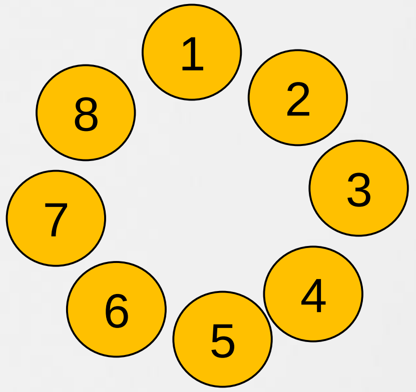
- Since the first algorithm removes one node at a time, it needs n-1 steps to compute f(n)
- 由于第一种算法每次删除一个节点，因此需要n-1步来计算f(n)
- The cost of each step (removing one node) is a constant time α (independent of n)
- 每一步(移除一个节点)的成本是一个常数时间□(与n无关)
- So the total number of operations to find f(n) is α(n-1)
- 所以求f(n)的总操作数是 α(n-1)
  - We can just represent it as O(n).
  - 我们可以把它表示为O(n)

The Second Algorithm
Can we do better?

- Simulation is a brute-force method. Faster algorithms are usually expected.
- 模拟是一种蛮力方法。通常需要更快的算法。
- The scientific approach 科学方法
  1. Observing some cases 观察一些案例
  2. Making some hypotheses (generalization) 作假设
  3. Testing the hypotheses 检验假设
  4. Repeat 1-3 until success 重复一到三步直到成功

Observing a case

- Let’s checkout the case n = 8.
  让我们来检验n=8的情况
- What have you observed? 你观察到了什么？
  - All even numbered people die 编号是偶数的人都会死亡
    - This is true for all kinds of n. 这对所有的n都成立
  - The starting point is back to 1 起点回到了1
    - This is only true when n is even 只有当n是偶数时才成立
  - Let’s first consider the case when n is even
    让我们先考虑n是偶数的情况



n is even

- For n=8, after the first “round”, there are only 4 people left.

  当n=8时，第一轮过后，只剩下4个人。

- If we can solve f(4), can weuse it to solve f(8)?

  如果我们能解出f(4)，我们能用它来解f(8)吗？

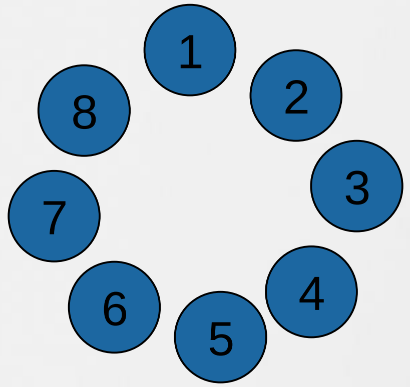
- If we renumber the remaining people,

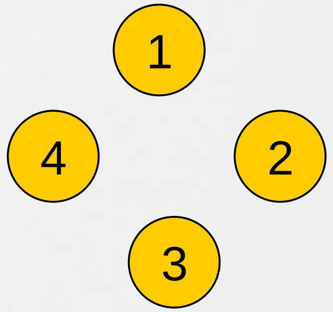
  1→1, 3→2, 5→3, 7→4, it becomes the n=4 problem.

  如果我们把剩下的人重新编号，1→1, 3→2, 5→3, 7→4，它又变成了n=4 时的问题

- So, if f(4)=x, f(8) =2x – 1

  所以，如果f(4)=x, f(8) =2x – 1





n is odd

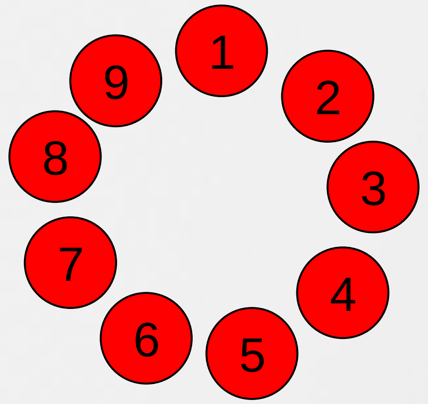
- Let’s checkout the case n=9

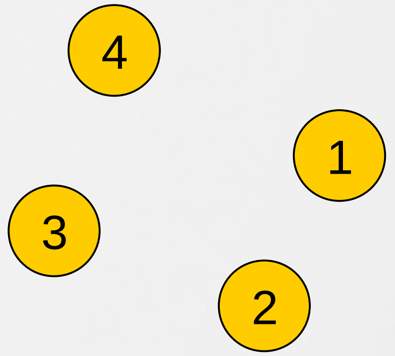
  让我们再来检验一下n=9时的情况
- The starting point becomes 3 起点变成了3
- If we can solve f(4), can we use it to solve f(9)?

  如果我们可以解决f(4)的问题，我们可以用它来解决f(9)的问题？
- If we renumber the remaining people, 3→1, 5→2, 7→3, 9→4, it becomes the n=4 problem.

  如果我们把剩下的人重新编号，3→1, 5→2, 7→3, 9→4, 它变成了n=4的问题
- So, if f(4)=x, f(9) =2x+1

  所以，如果f(4)=x, f(9) =2x+1





Recursive relation

- Hypothesis: If f(n)=x, f(2n)=2x–1.
  假设 If f(n)=x, f(2n+1)=2x+1.
  - How to prove or disprove it?
    怎么样去证明或者反驳它？
- This is called a recursive relation.
- 递归关系
  - You can design a recursive algorithm
    - Compute f(8) uses f(4);
    - Compute f(4) uses f(2);
    - Compute f(2) uses f(1);
    - f(1) =1. Why?

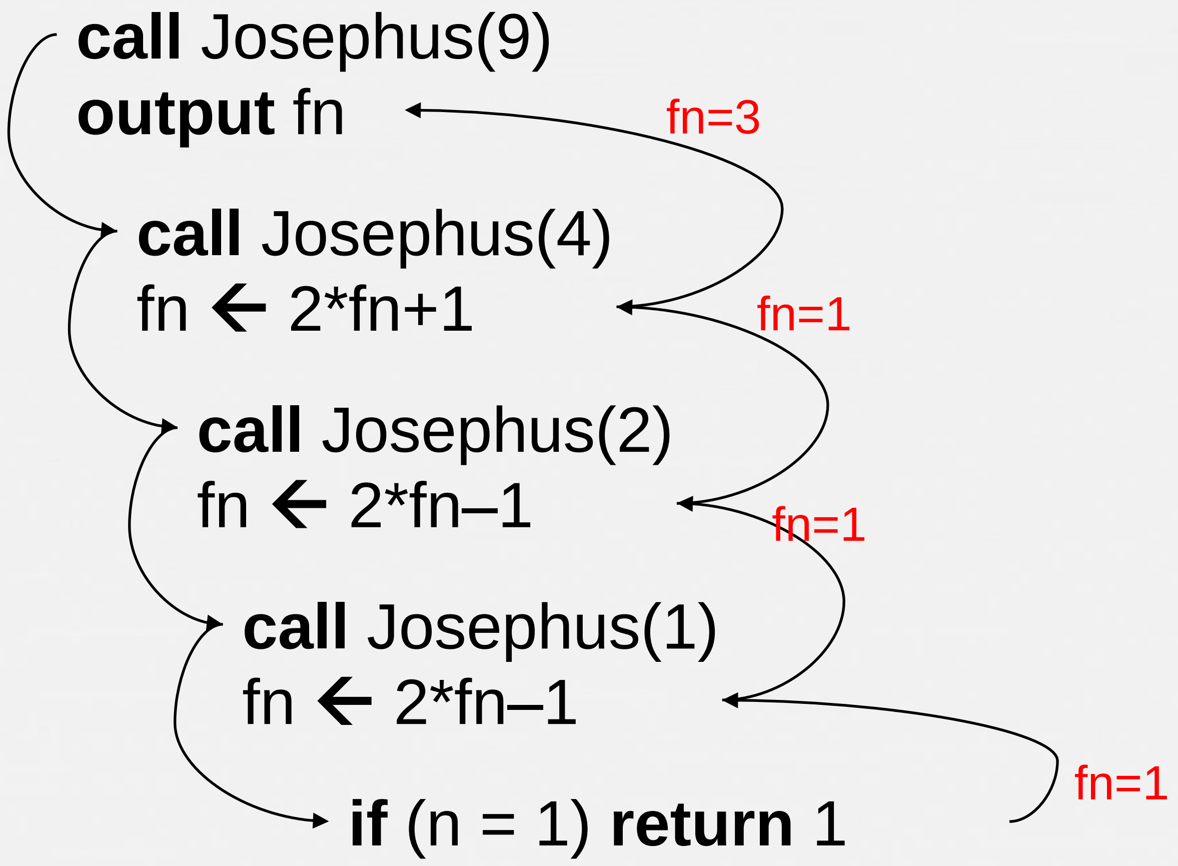
$$f(8)=2f(4)-1$$

$$f(4)=2f(2)-1$$

$$f(2)=2f(1)-1$$

$$f(1)=1$$

Recursive function



```
call Josephus(9)
output fn
    call Josephus(4)
    fn ← 2*fn+1
        call Josephus(2)
        fn ← 2*fn–1
            call Josephus(1)
            fn ← 2*fn–1
                if (n = 1) return 1
```

```
int Josephus (n) {
    /* base case */
   if (n == 1)  return 1;
   if (n%2 == 0) /*n is even*/
      return 2*Josephus (n/2)-1;
   else            /*n is odd*/
      return 2*Josephus((n–1)/2)+1;
}
```

How fast can it compute f(n)?

- *n* reduces at least half at each step. N每一步至少减少一半。
  - Need $\log_2(n)$ steps to reach n=1. 需要log2(n)步才能达到n=1。
- Each step needs a constant number of operations α. 每一步都需要固定数量的操作。
- The total number of operations is $\alpha\log_2(n)$ 操作总数是αlog2(n)

$$f(1) = 1$$

$$f(n) = \begin{cases} 2f(n/2) - 1 & \text{if n is even} \quad \text{n是偶数} \\ 2f((n-1)/2) + 1 & \text{if n is odd} \quad \text{n是奇数} \end{cases}$$

The Third Algorithm
Can we do better?

- Can we solve the recursion? 我们能解这个递归吗?

$$f(1) = 1$$

$$f(n) = \begin{cases} 2f(n/2) - 1 & \text{if } n \text{ is even} \\ 2f((n-1)/2) + 1 & \text{if } n \text{ is odd} \end{cases}$$

  - If we can, we may have a better algorithm to find f(n)

    如果可以，我们可能有更好的算法来求f(n)

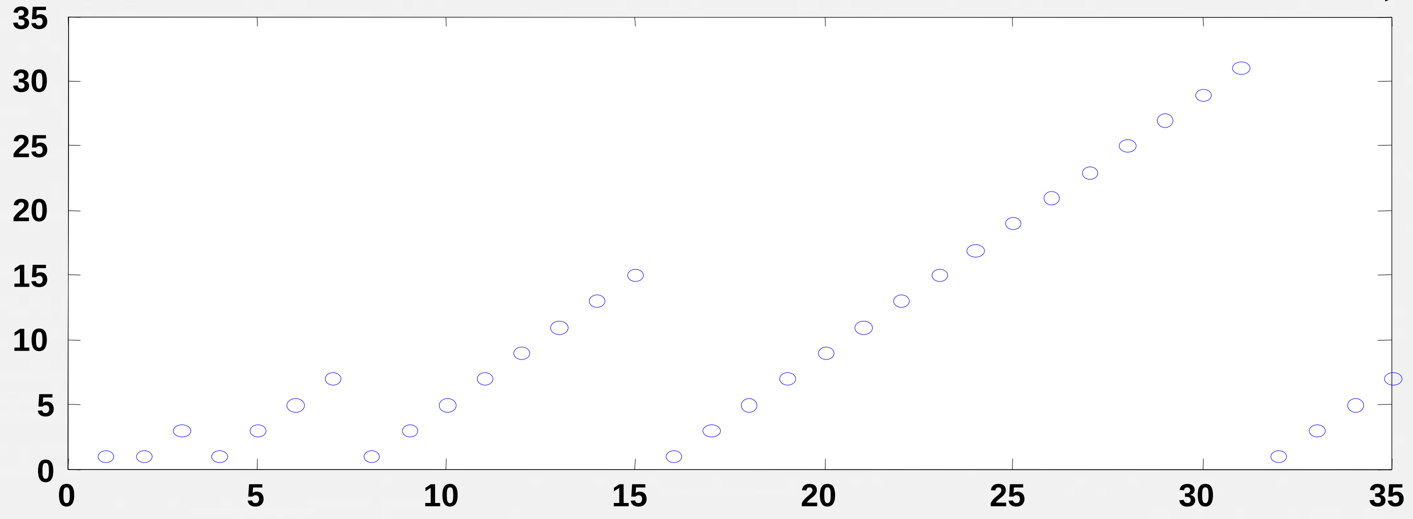
- Remember:
  - observation, hypotheses, verification

    观察，　假设，　验证

Let's make more observations

- n = 2, f(2) = 1
- n = 3, f(3) = 3
- n = 4, f(4) = 1
- n = 5, f(5) = 3

- n = 6, f(6) = 5
- n = 7, f(7) = 7
- n = 8, f(8) = 1
- n = 9, f(9) = 3



Have you observed the pattern of f(n)?

你观察到f(n)的规律了吗?

What are the patterns?

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
f(n)	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1
k	0	0	1	0	1	2	3	0	1	2	3	4	5	6	7	0

1. $f(1)=f(2)=f(4)=f(8)=f(16)=1$.

What are they in common? 它们有什么共同点?

They are power of 2, 2^m.

2.If we group the sequence [1], [2,3], [4,7],[8,15], f(n) in each group is a sequence of consecutive odd numbers starting from 1.

如果我们将序列[1]，[2,3]，[4,7]，[8,15]分组，
每组中的f(n)是从1开始的连续奇数序列。

$f(n)=2k+1$

3. Let k = n – <u>the first number in n’s group</u>
What is the pattern of k? 设k = n， n组的第一个数k的规律是什么?

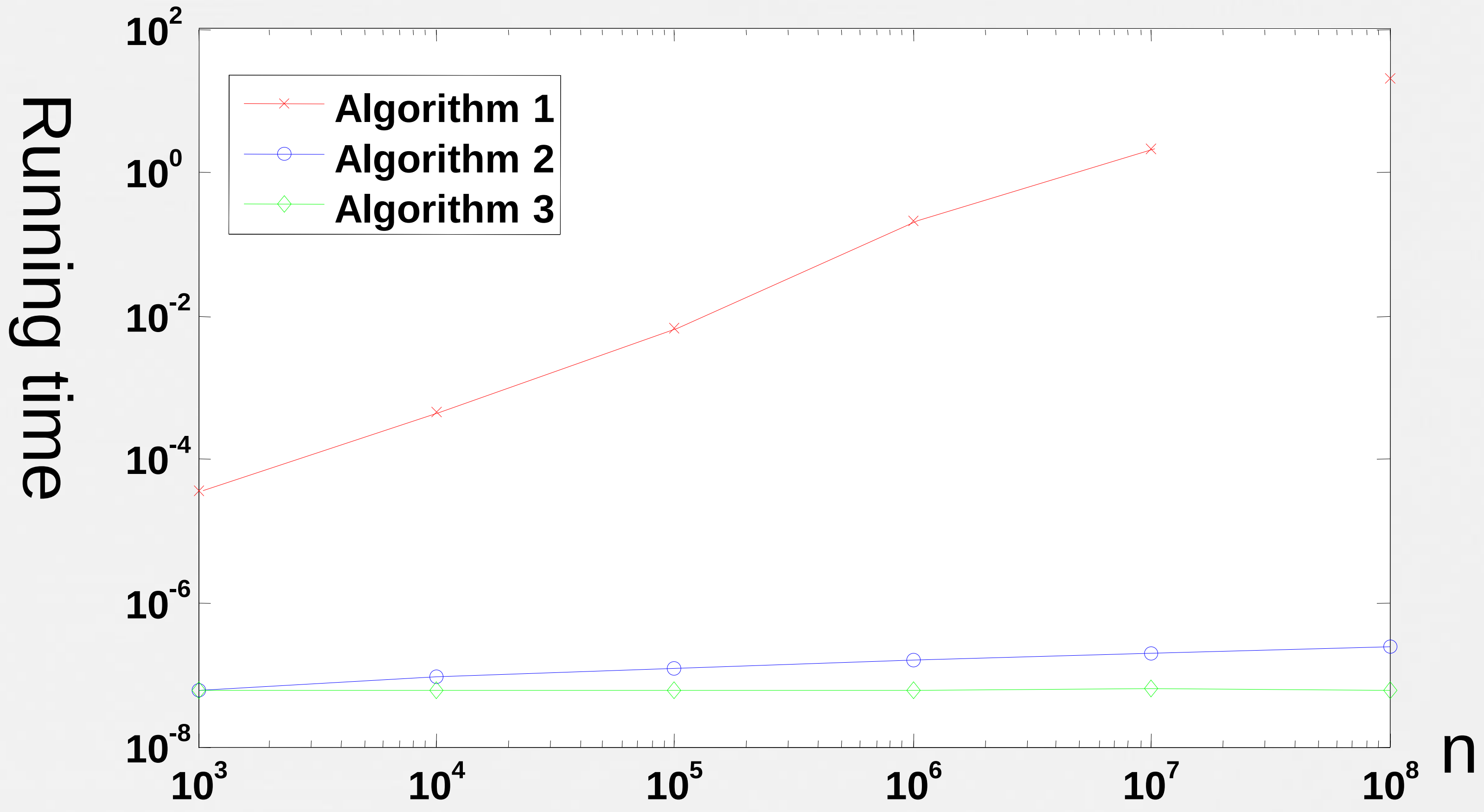
Guess the solution

- $f(n) = 2k+1$
  - k = n – the first number in n’s group K = n - n所在组的第一个数
  - The first number in n’s group is 2^m. n的第一个数是2m

$$2^m \leq n < 2^{m+1}$$

- How fast can you find m?
  - (1) use “binary search” on the bit pattern of *n* 对n的位模式使用“二进位搜索”
    - Since *n* has $\log_2(n)$ bits, it takes $\log_2(\log_2(n))$ steps.
    因为n有log2(n)位，它需要log2(log2(n))步。
  - (2) use “log2” function and take its integer part 使用“log2”函数并取其整数部分
    - It takes constant time only. (independent of *n*.)
    它只需要常数时间。(独立于n)

Running time of three algorithms



Josephus problem.

The problem has the following recursive structure.

该问题具有以下递归结构。

josephus(n, k) = (josephus(n - 1, k) + k-1) % n + 1 josephus(1, k) = 1

After the first person (kth from the beginning) is killed, n-1 persons are left.

当第一个人(从开始的第k个人)被杀死后，还剩下n-1个人。

- So we call Josephus(n – 1, k) to get the position with n-1 persons. 我们叫Josephus(n -1, k)得到n-1个人的位置。
- But the position returned by Josephus(n – 1, k) will consider the position starting from k%n + 1.

  但是Josephus(n - 1, k)返回的位置将考虑从k%n + 1开始的位置。
- So, we must make adjustments to the position returned by Josephus(n – 1, k).

  因此，我们必须对Josephus返回的位置(n - 1, k)进行调整。

Josephus problem.

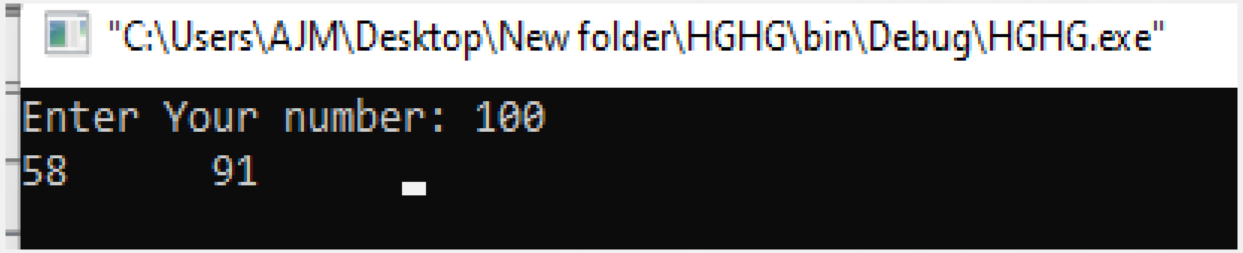
```
#include <iostream>
using namespace std;
int josephus(int n, int k)
{
    if (n == 1)
        return 1;
    else
        /* The position returned by josephus(n - 1, k)
        is adjusted because the recursive call
        josephus(n - 1, k) considers the
        original position k % n + 1 as position 1 */
        return (josephus(n - 1, k) + k - 1) % n + 1;
}
// Driver Program to test above function
int main()
{
    int n = 14;
    int k = 2;
    cout << "The chosen place is " << josephus(n, k);
    return 0;
}
```

- The following is a simple recursive implementation of the Josephus problem.
  下面是Josephus问题的一个简单递归实现。
- The implementation simply follows the recursive structure mentioned above.
- 实现简单地遵循上面提到的递归结构。

Output: The chosen place is 13
Time Complexity: O(n)
Auxiliary Space: O(n)

Try this code

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
                int a[1000] = { 0 }, i, j, k = 0, n;
                cout << "Enter Your number: ";
                cin >> n;
                do {
                 i = 0;
                 for (j = 0; j < n; j++)
                 if (a[j] == 0) {
                 k++;
                 if (k == 3) {
                 a[j] = 1;
                 k = 0;
                     }
                            }
                 else
                     i++;
                } while (i != n - 2);
                for (i = 0; i < n; i++)
                 if (a[i] == 0)
                 cout << i + 1 << "\t";
                _getch();
                return 0;
}
```



"C:\Users\AJM\Desktop\New folder\HGHG\bin\Debug\HGHG.exe"

```
Enter Your number: 100
58      91
```

Task 06

- Based on the code in this ppt
- 1. Please report your system configuration (Cpu model)
    1. Please run the example code C++ and python
    2. Write the Pseudocode for each program
    3. Report the time
    4. Calculate the Time complexity and Big O notation

1. Please carefully read the question and send your answer just based on question

2. Please Just use the share format in MOOC system.

3.Please send your task on time

1. 请仔细阅读问题，并根据问题发送您的答案
2. 请只使用MOOC系统中的共享格式。
3.请按时发送您的任务

Reference and study more

- The Josephus problem:
  - Graham, Knuth, and Patashnik, “Concrete Mathematics”, section 1.3
  - http://en.wikipedia.org/wiki/Josephus_problem
  - http://mathworld.wolfram.com/JosephusProblem.html

江西理工大学

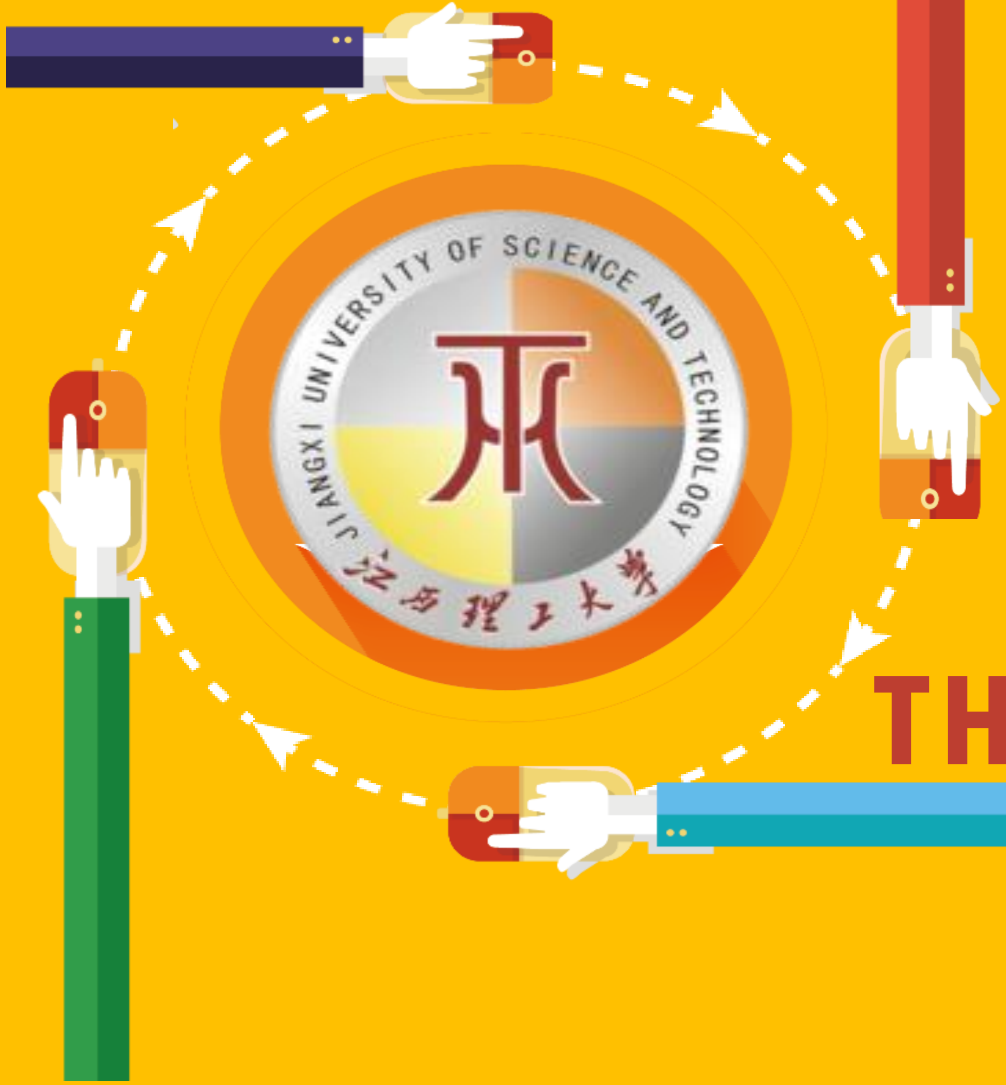
Jiangxi University of Science and Technology

信息工程学院

School of information engineering

高级算法分析与设计

Advanced Algorithm Analysis and Design



THANK YOU

"The beauty of research is that you never know where it's going to lead."

研究的美妙之处在于你永远不知道它会通向何方

RICHARD ROBERTS
Nobel Prize in Physiology or Medicine 1993

“BE HUMBLE. BE HUNGRY. AND ALWAYS BE THE HARDEST WORKER IN THE ROOM.”