北京交通大學
BEIJING JIAOTONG UNIVERSITY

递归程序设计

——栈的重要应用

本节内容

- 3.1 递归的定义
- 3.2 菲波那切数列（Fibonacci）
  - 递归的执行过程
  - 递归运行栈与递归树
  - 递归程序的效率分析
- 3.3 递归程序的设计方法
- 3.4 设计举例

北京交通大學
BEIJING JIAOTONG UNIVERSITY

递归的定义

递归举例——像中像

- 如果A、B两面镜子相对而置。A镜中有B镜中的像，B镜中有A镜中的像，如此反反复复，就会产生一连串的“像中像”。

递归函数：直接或间接调用自己的函数

(1) 直接递归

```
Function Sort(...)
.
.
if(condition)
    call Sort(...)
.
.
end
```

(2) 间接递归

```
Function Sort(...)
.
.
if(condition)
    call Search (...)
.
.
end
```

```
Function Search(...)
.
.
if(condition)
    call Sort (...)
.
.
end
```

递归的定义

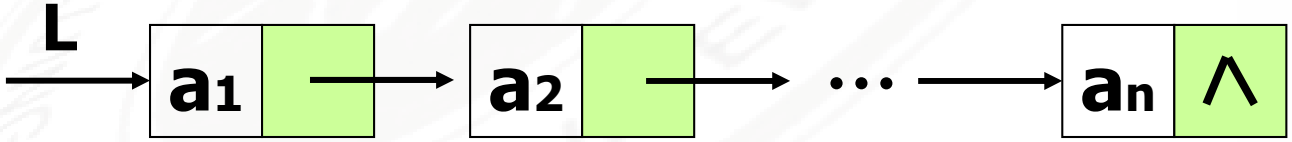
- 递归定义的数学函数

阶乘函数:

$$Fact(n)=\begin{cases}1 & 若n=0\\ n\cdot Fact(n-1) & 若n>0\end{cases}$$

- 具有递归特性的数据结构

单链表

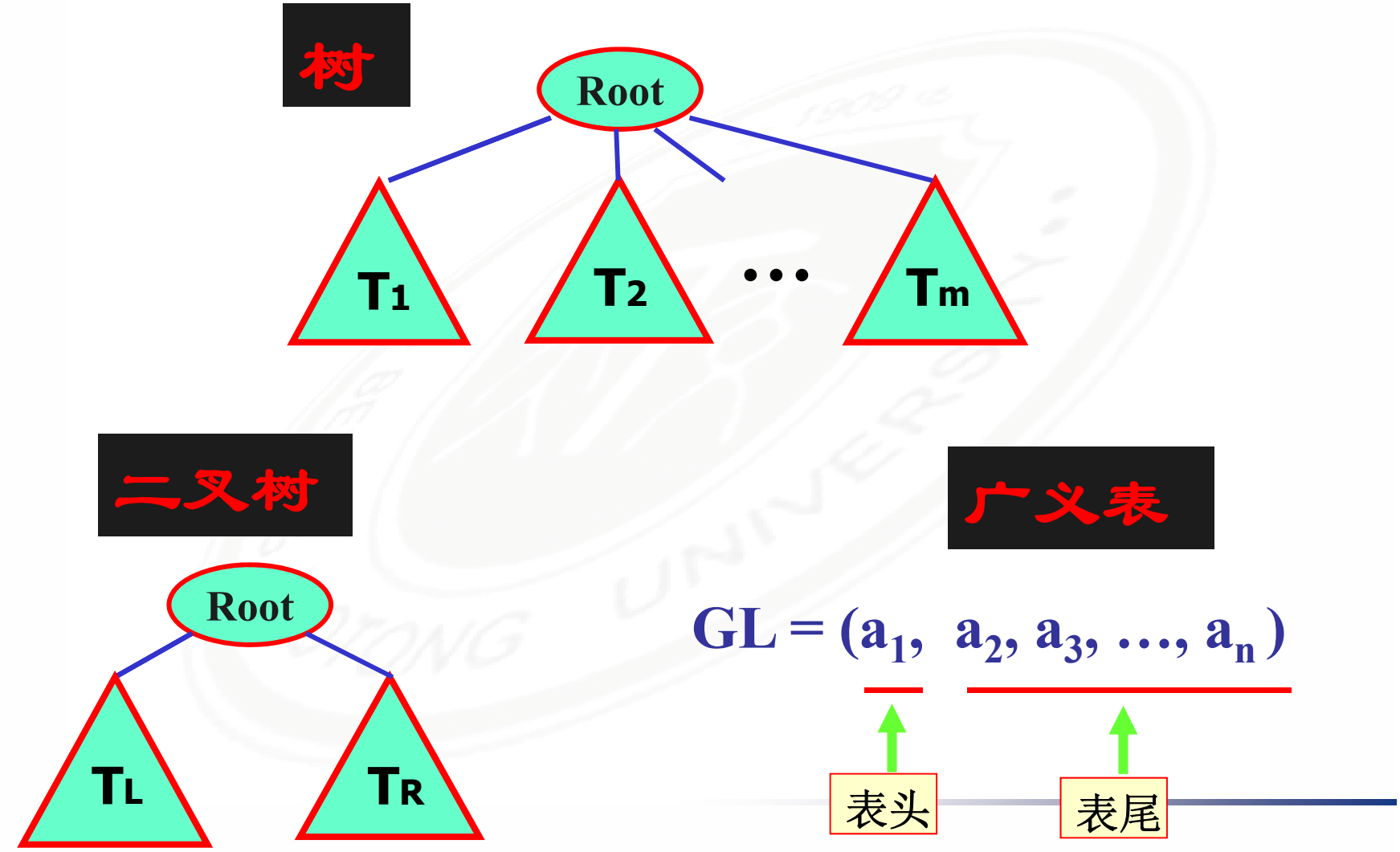


字符串

S="$a_1\ a_2 \ldots\ a_n$"

递归的定义

- 具有递归特性的数据结构（续）



本节内容

- 3.1 递归的定义
- 3.2 菲波那切数列（Fibonacci）
  - 递归的执行过程
  - 递归运行栈与递归树
  - 递归程序的效率分析
- 3.3 递归程序的设计方法
- 3.4 设计举例

菲波那切数列（Fibonacci）

- 菲波那切数列（Fibonacci）
  - 斐波那契数列的发现者，是意大利数学家列昂纳多·斐波那契。
  - 这个级数与大自然植物的关系极为密切。松果、凤梨、树叶的排列、某些花朵的花瓣数（典型的有向日葵花瓣），蜂巢，蜻蜓翅膀，等等
  - 此级数中任何相邻的两个数，次第相除，其比率都最为接近0.618034……极限就是所谓的"黄金分割数"。
- 兔子繁殖问题
  - 如果兔子在出生两个月后，就有繁殖能力。一对兔子每个月能生出一对小兔子。假设所有兔子都不死，那么一年以后可以繁殖多少对兔子？

月份	兔子对数
1	1
2	
3	
4	
5	
6	

所经过的月数	1	2	3	4	5	6	7	8	9	10	11	12
兔子对数	1	1	2	3	5	8	13	21	34	55	89	144

菲波那切数列（Fibonacci）

菲波那切级数的数学表达式

- 通项公式

$$\mathbf{Fib}(n)=\frac{\sqrt{5}}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n}-\left(\frac{1-\sqrt{5}}{2}\right)^{n}\right]$$

- 递推公式

2阶Fibonaci数列:

$$Fib(n)=\begin{cases}0 & \text{若}n=0\\ 1 & \text{若}n=1\\ Fib(n-1)+Fib(n-2) & \text{其它}\end{cases}$$

菲波那切数列的迭代实现

- 打印出前40位菲波那切级数。

```
int main ( )
{
    int i;
    int a[40];
    a[0]=0;
    a[1]=1;
    printf ("%d ",a[0]) ;
    printf ("%d ",a[1]) ;
    for (i = 2;i < 40;i++)
    {
        a[i] = a[i-1] + a[i-2];
        printf ("%d ",a[i]) ;
    }
    return 0;
}
```

菲波那切数列的递归实现

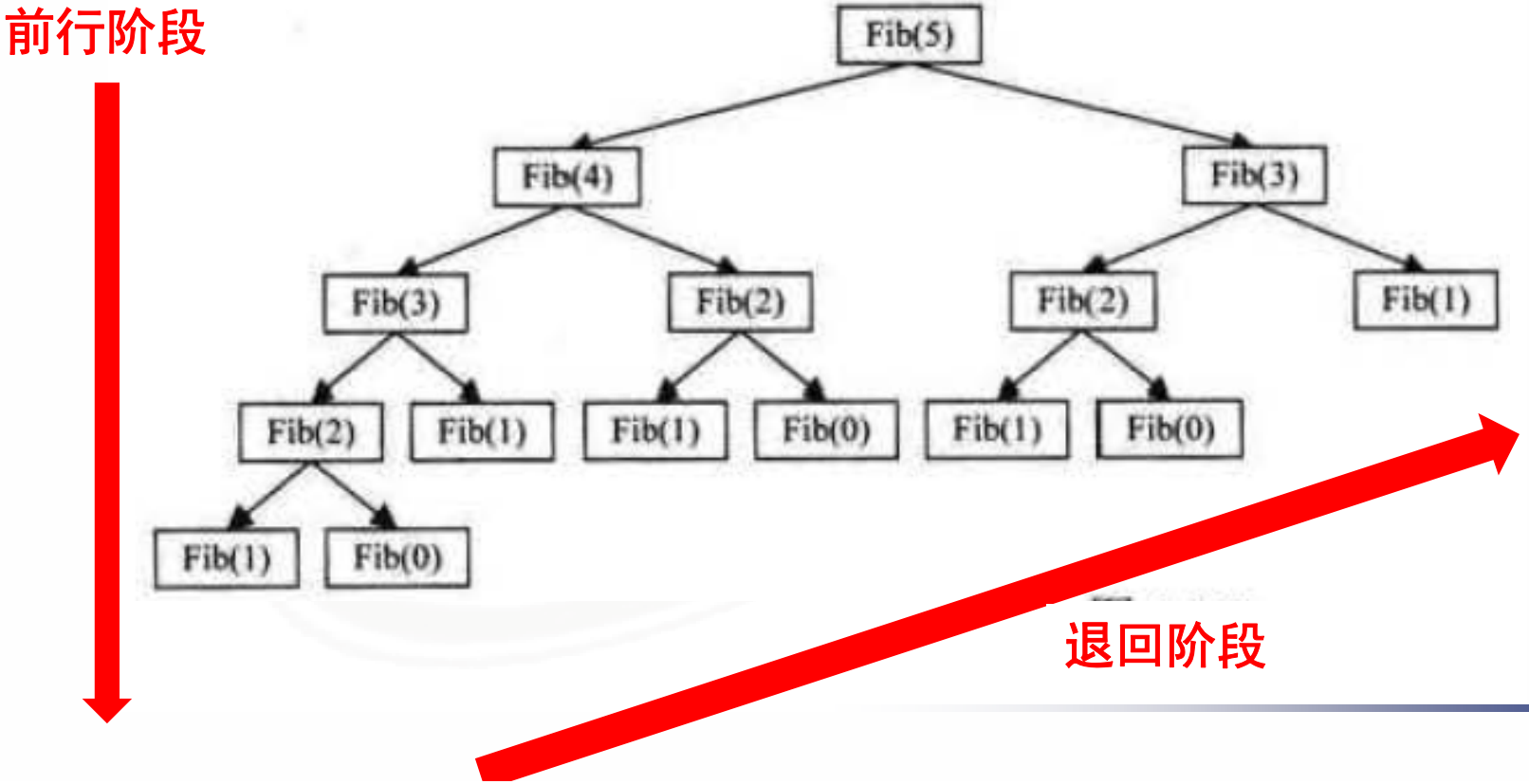
- 用递归实现

```
/* 斐波那契的递归函数 */
int Fbi(int i)
{
    if( i < 2 )
        return i == 0 ? 0 : 1;
    return Fbi(i-1) + Fbi(i-2);/*这里Fbi就是函数自己，它在调用自己 */
}
int main()
{
    int i;
    for(int i = 0;i < 40;i++)
        printf("%d ", Fbi(i));
    return 0;
}
```

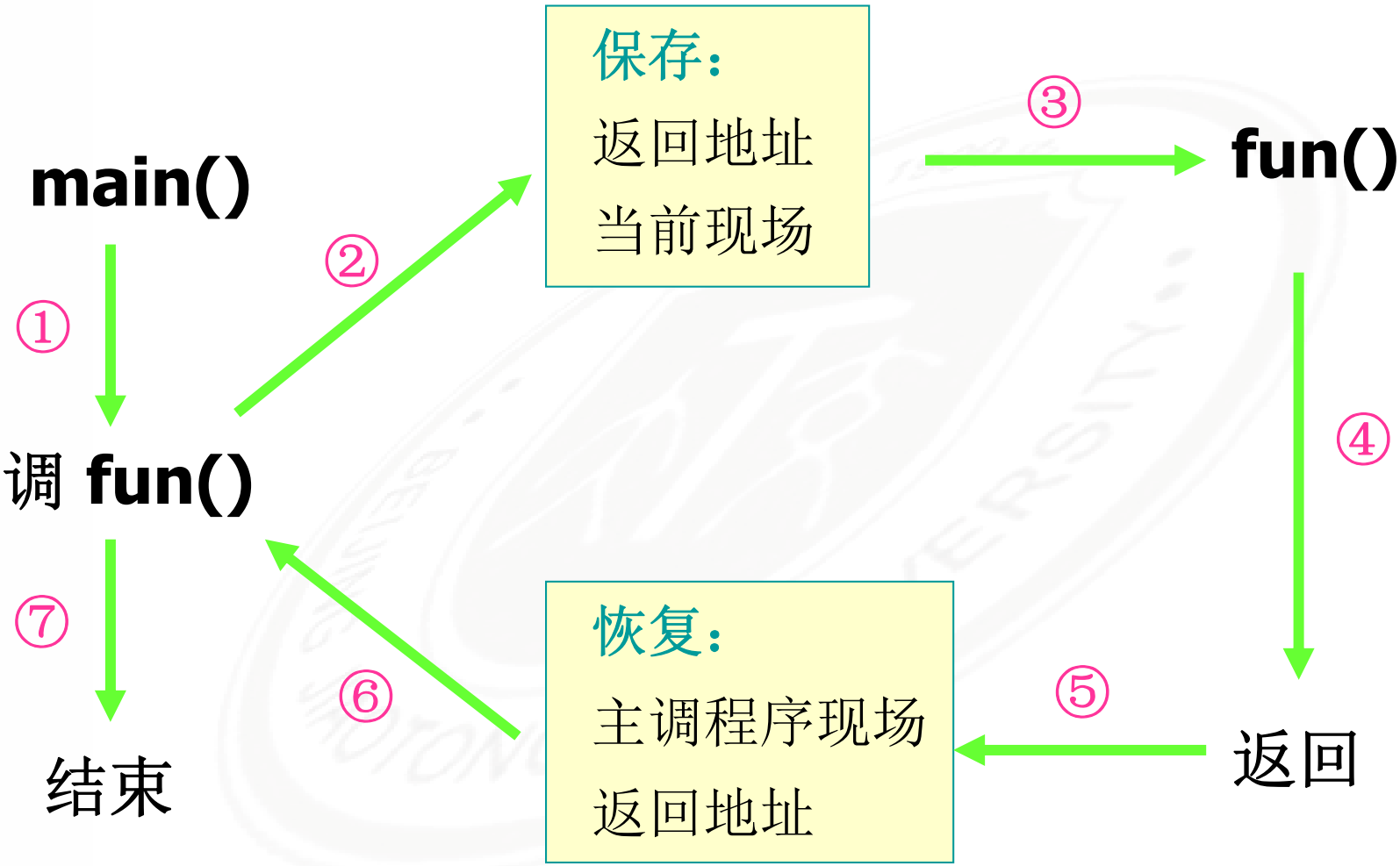
菲波那切数列的递归实现

递归的执行过程

- 在前行阶段，每一层递归，函数的局部变量、参数值以及返回地址都被压入栈中。在退回阶段，位于栈顶的局部变量、参数值和返回地址被弹出，用于返回调用层次中执行代码的其余部分。



回顾--函数调用过程



回顾--函数调用过程

调用前，系统完成：

(1)将实参,返回地址等传递给被调用函数

(2)为被调用函数的局部变量分配存储区

(3)将控制转移到被调用函数的入口

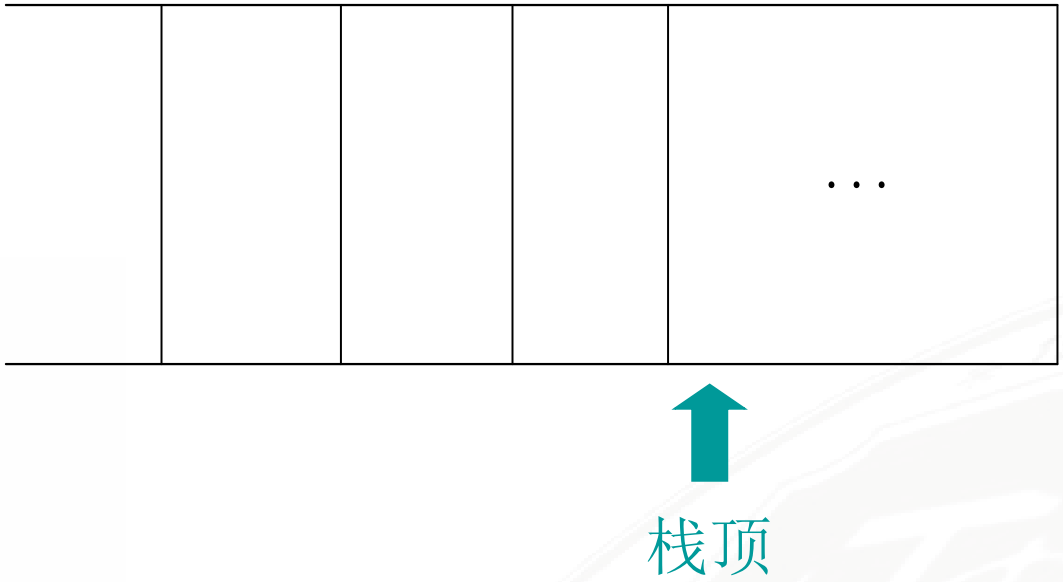
调用后，系统完成：

(1) 保存被调用函数的计算结果

(2) 释放被调用函数的数据区

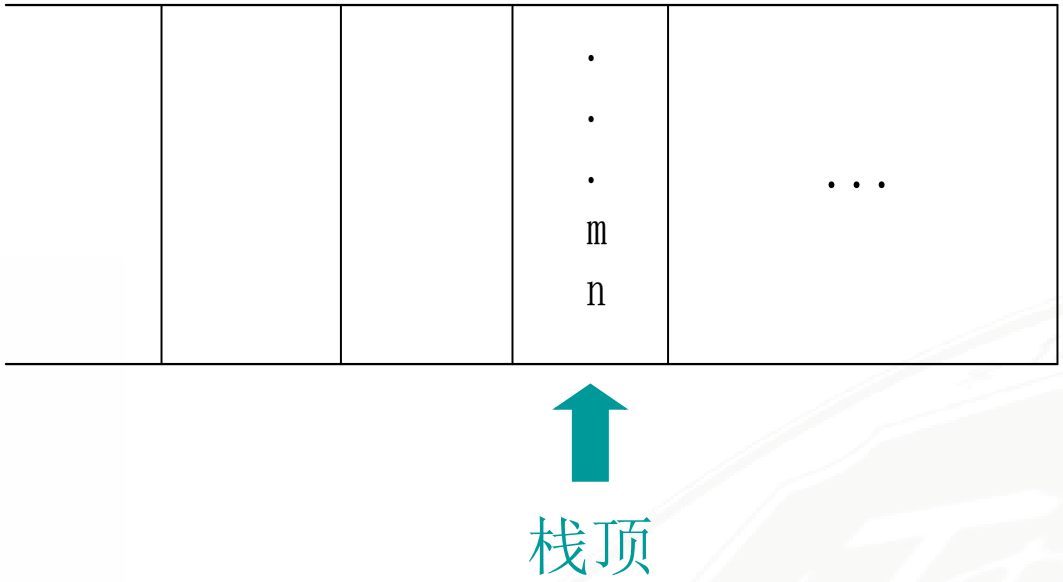
(3) 依照被调用函数保存的返回地址将控制转移到调用函数

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```



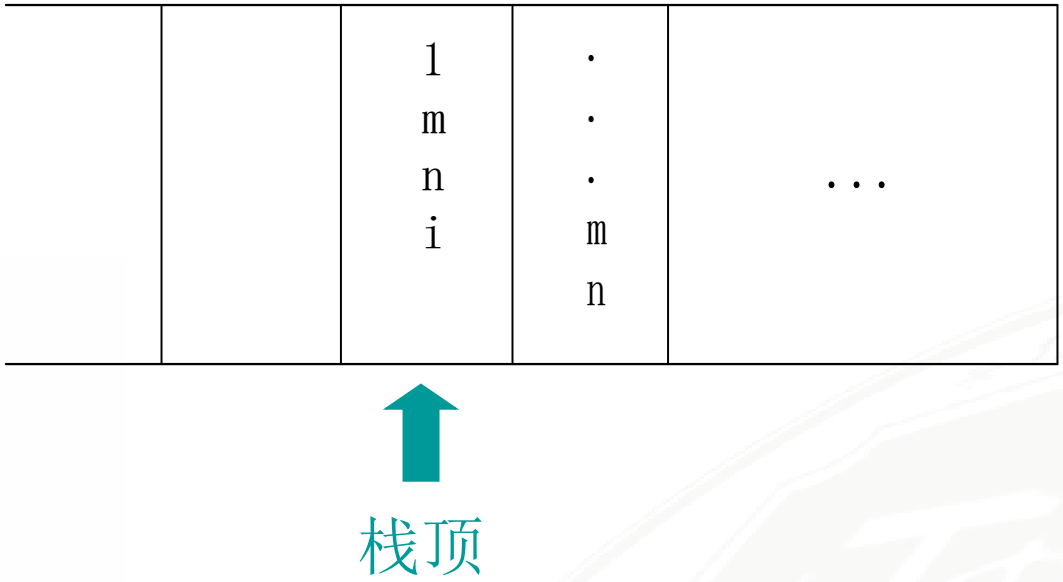

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```

函数调用过程



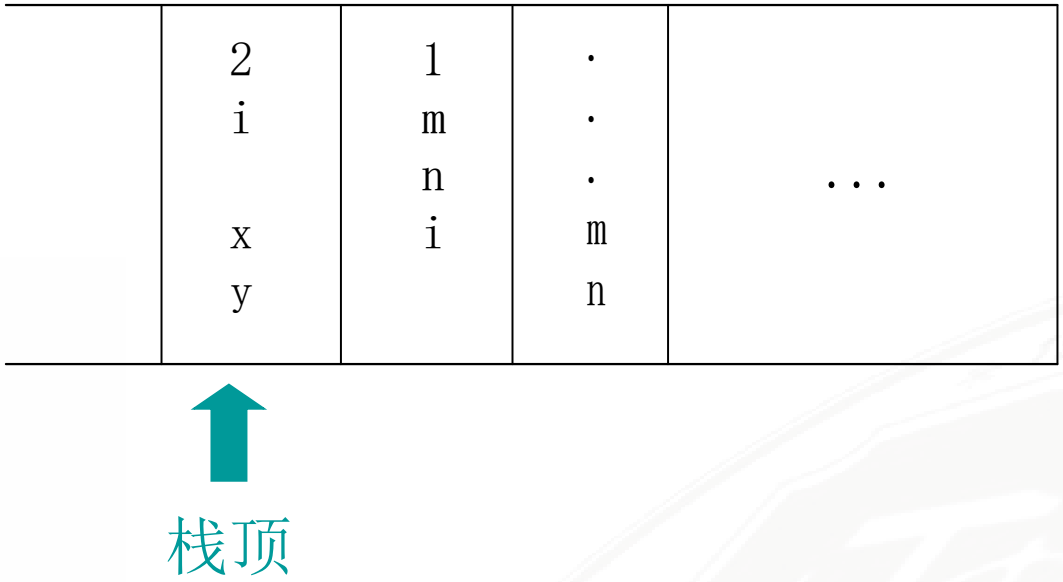
函数调用过程

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```



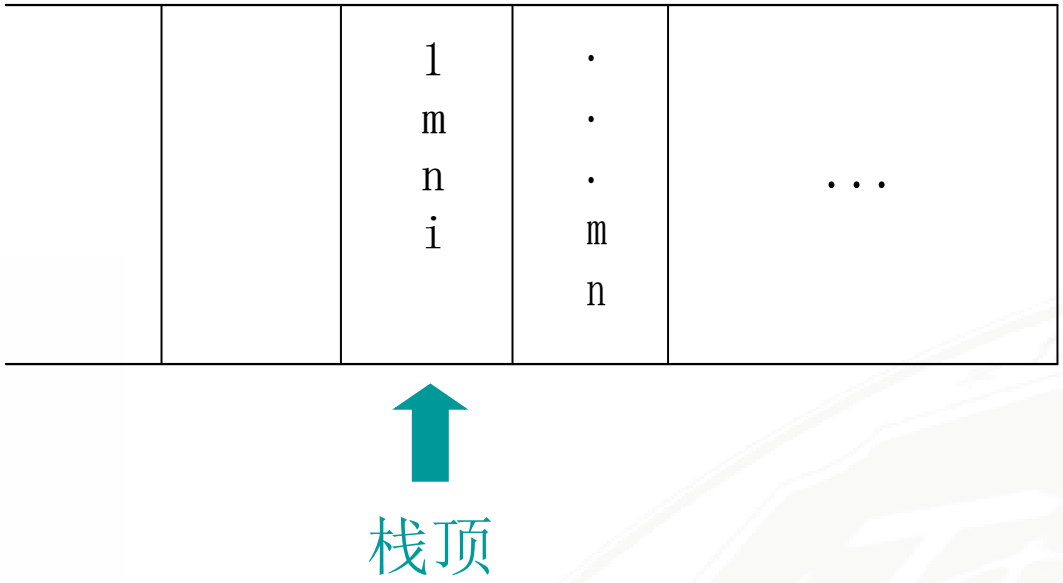
函数调用过程

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```



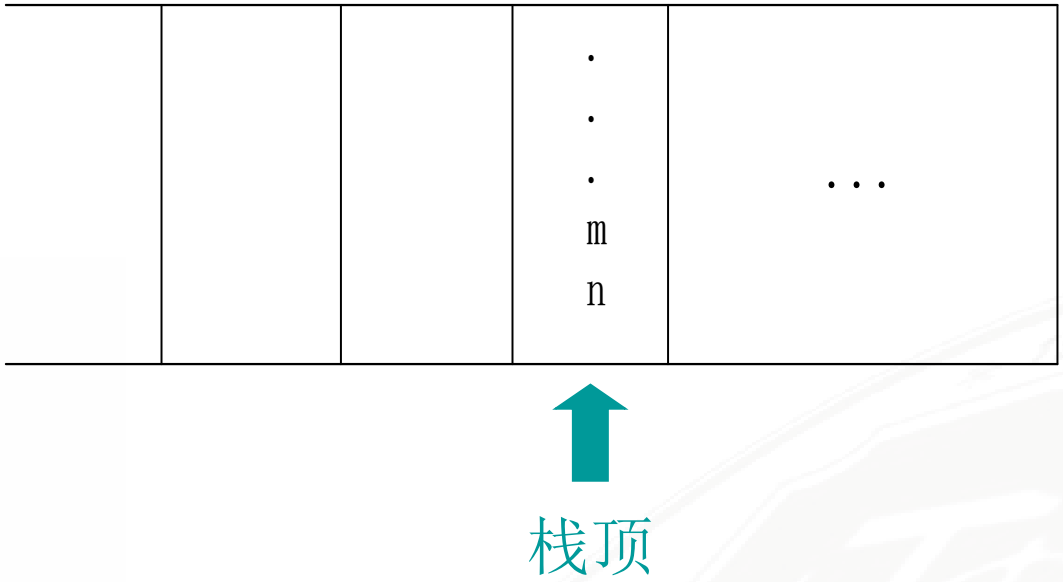
函数调用过程

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```



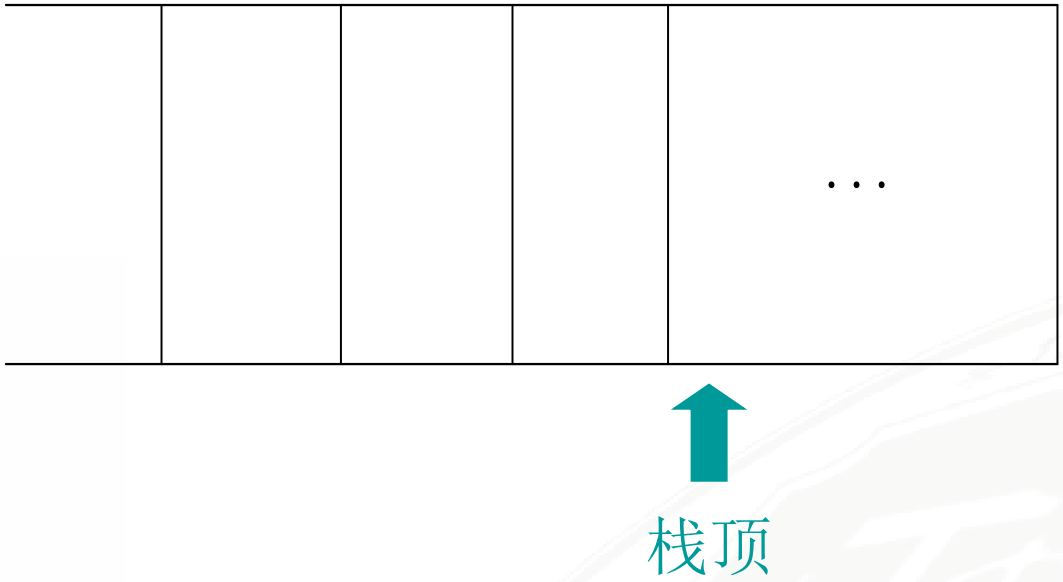
函数调用过程

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```



函数调用过程

```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```




```
int first(int s,int t);
int second(int d);
int main(){
   int m,n;
   ...
   first(m,n);
   1:...
}

int first(int s,int t){
   int i;
   ...
   second(i);
   2:...
}

int second(int d){
   int x,y;

   ...
}
```

递归函数调用的实现

“层次”

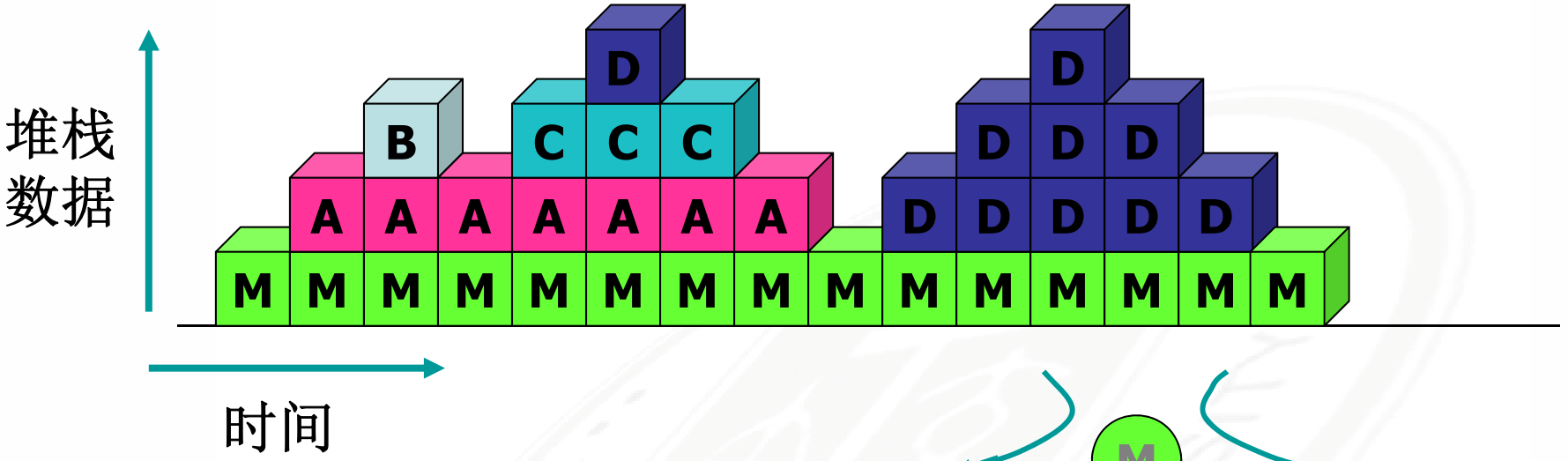
主函数	0层
第1次调用	1层
第 i 次调用	i 层

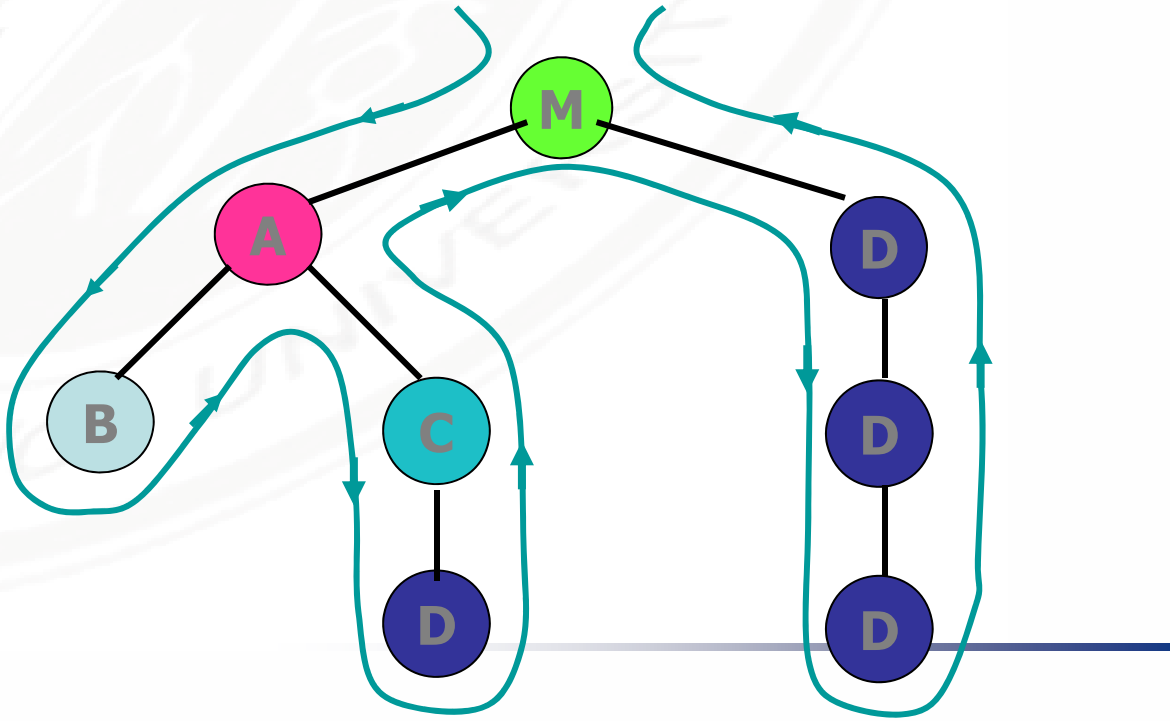
“递归工作栈”

“工作记录” ⇒ 实际参数,局部变量,返回地址

“活动记录”

递归工作栈与递归树


堆栈
数据
时间
M M M M M M M M M M M M M M M
A A A A A A A
B
C C C
D
D D D
D D D D D
D


M
A
B
C
D
D
D
D
D

递归程序的效率分析

- 菲波那切数列迭代算法的时间复杂度

```
int main ( )
{
    int i;
    int a[40];
    a[0]=0;
    a[1]=1;
    printf ("%d ",a[0] ) ;
    printf ("%d ",a[1] ) ;
    for (i = 2;i < 40;i++)
    {
        a[i] = a[i-1] + a[i-2];
        printf ("%d ",a[i] ) ;
    }
    return 0;
}
```

$O(n)$

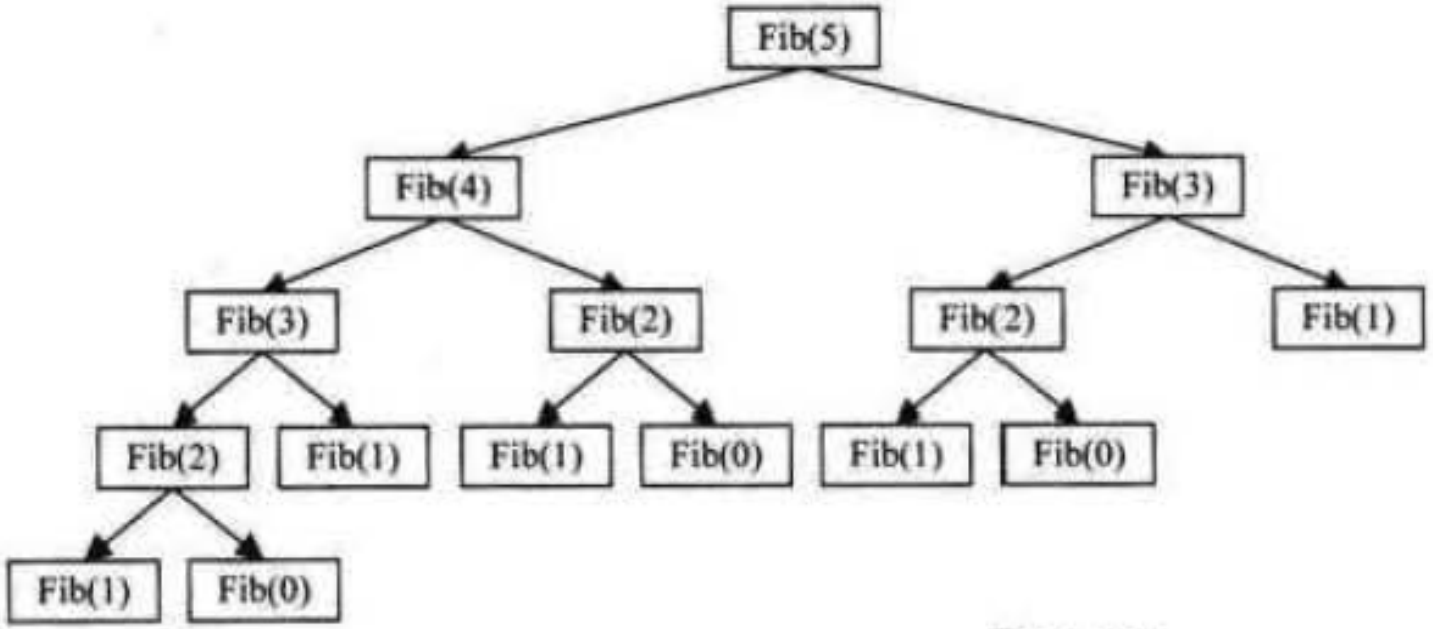
递归程序的效率分析

- 菲波那切数列递归算法的时间复杂度

```
/* 斐波那契的递归函数 */
int Fbi(int i)
{
    if( i < 2 )
        return i == 0 ? 0 : 1;
    return Fbi(i-1) + Fbi(i-2);/*这里Fbi就是函数自己，它在调用自己 */
}
int main()
{
    int i;
    for(int i = 0;i < 40;i++)
        printf("%d ", Fbi(i));
    return 0;
}
```

递归程序的效率分析

- 菲波那切数列递归算法的时间复杂度



递归调用的次数：$C(n)=2^n-1$

迭代和递归算法的对比

迭代算法的特点

- 使用循环结构
- 程序结构较复杂，不易读懂，容易出错
- 不需要反复调用函数和占用额外内存

递归算法的特点

- 使用选择结构
- 结构更清晰、简洁、易懂
- 大量的递归调用会建立函数的副本，耗费大量的时间和内存

根据实际问题，选择合适的实现方式。

本节内容

- 3.1 递归的定义
- 3.2 菲波那切数列（Fibonacci）
  - 递归的执行过程
  - 递归运行栈与递归树
  - 递归程序的效率分析
- 3.3 递归程序的设计方法
- 3.4 设计举例

递归算法的设计方法

递归算法求解问题的基本思想

对于一个复杂的问题，把原问题分解成若干个相对简单且同类的子问题，这样原问题就可以地推得到解。

适宜用递归求解问题的充分必要条件

（1）子问题与原始问题为同样的事

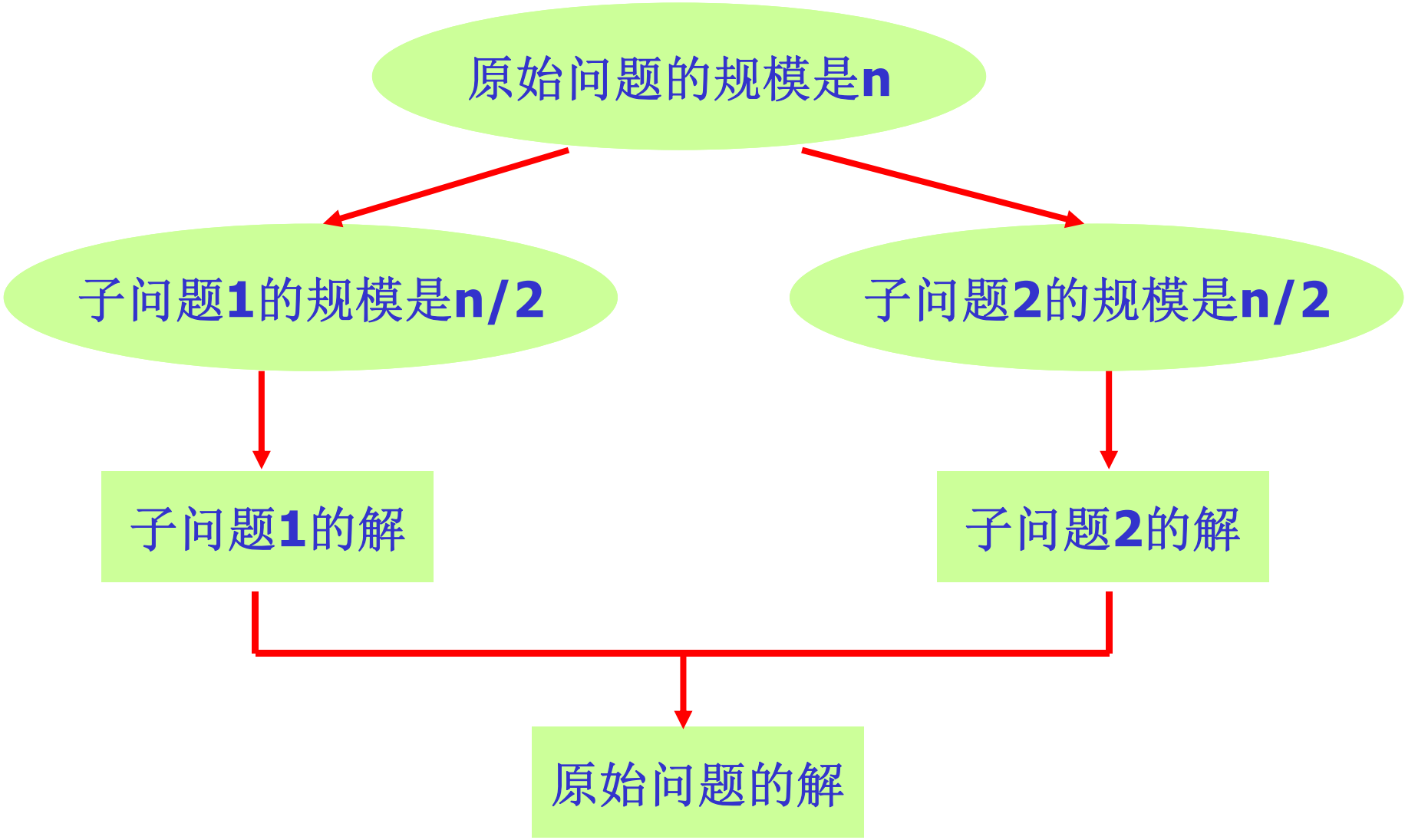
（2）不能无限制地调用本身，须有一个出口，化简为非递归状况处理

常用的方法有：分治法，减治法

分治法(Divide and Conquer)

1. 将问题的实例划分为同一个问题的几个较小的实例，最好拥有同样的规模。
2. 对这些较小的实例求解（一般使用递归方法，但在问题规模足够小时，有时也会使用一些其他方法）。
3. 如果有必要，合并这些较小问题的解，以得到原始问题的解。

分治法(Divide and Conquer)



分治法(Divide-and-Conquer)

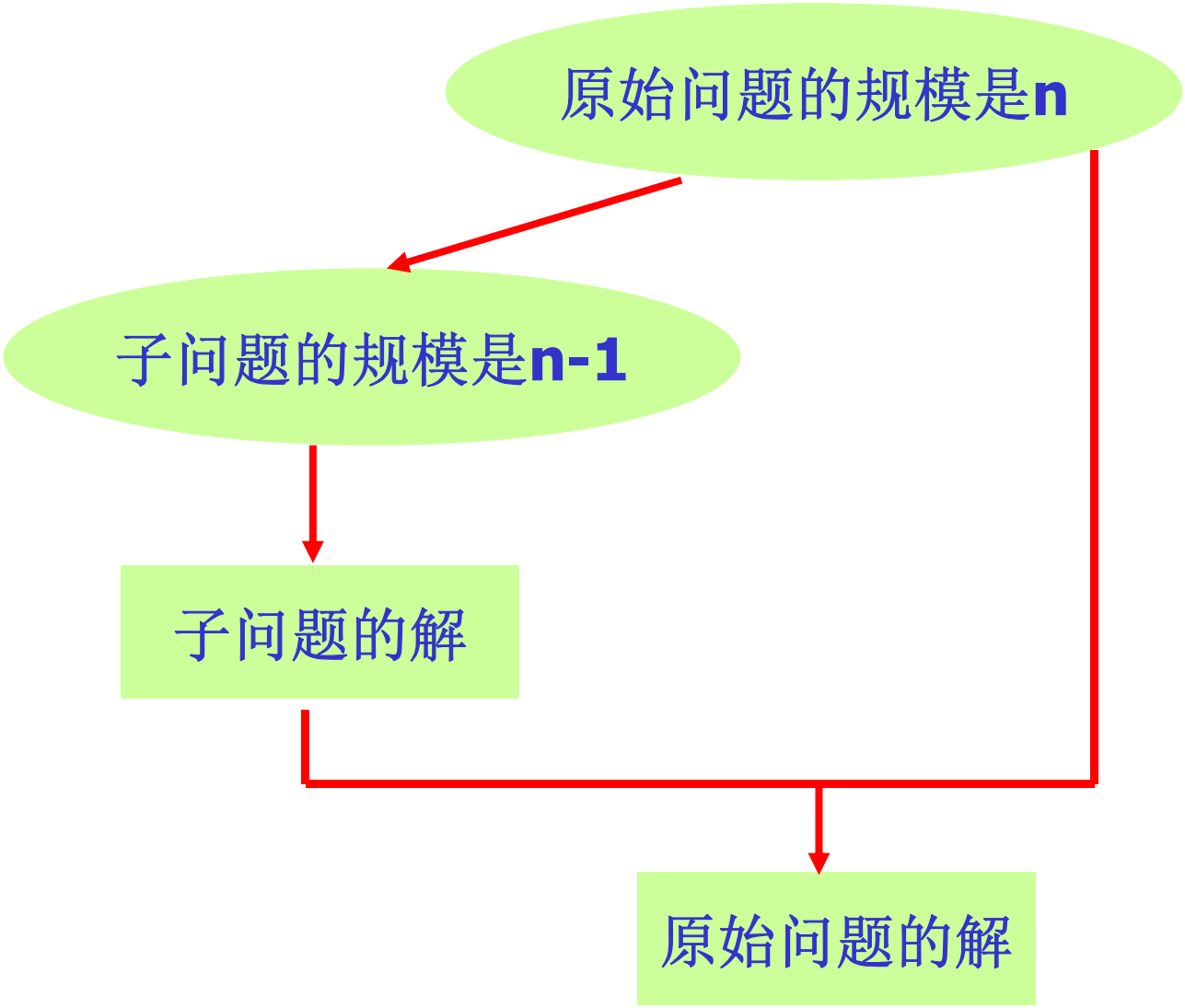
分治法的典型应用

- 二叉树的遍历及应用
- 广义表的递归算法
- 折半查找
- 快速排序
- 合并排序

减治法(Decrease-and-Conquer)

- 减治法利用了一种关系：**一个问题给定实例的解和同样问题较小实例的解之间的关系。**
- 一旦建立了这样一种关系，就可以**自顶至下**（递归地），也可以**自底至上**（非递归地）来运用
- 减治法有3种主要的变种：
  - 减去一个常量
  - 减去一个常数因子
  - 减去的规模是可变的

减治法(Decrease-and-Conquer)



减治法(Decrease-and-Conquer)

减治法的典型应用

- 插入排序
- 图的深度优先搜索
- 拓扑排序
- 二叉排序树的查找和插入
- 单链表的逆序打印或逆置

Anany Levitin，*Introduction to The Design & Analysis of Algorithms(影印版)*，清华大学出版社

递归程序书写方式

```
Function<name> (parameter list)
if(initial condition)
    return (initial value)
else
    return (<name>(parameter exchange));
end
```

参数表

初始条件

参数交换

例：求n个数的阶乘

$$n!=n*(n-1)!$$

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

初始条件

切忌想得太深太远！！！

本节内容

- 3.1 递归的定义
- 3.2 菲波那切数列（Fibonacci）
  - 递归的执行过程
  - 递归运行栈与递归树
  - 递归程序的效率分析
- 3.3 递归程序的设计方法
- 3.4 设计举例

实验6 汉诺塔

汉诺塔传说

- 在魔法世界的圣殿里，有一块黄铜板上插着三根宝石针。据说创世神在创造世界的时候，在A针上从下到上地穿好了由大到小的64片金片，这就是所谓的汉诺塔。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：一次只移动一片，不管在哪根针上，小片必须在大片上面。僧侣们预言，当所有的金片都从创世神穿好的那根针上移到C针上时，世界就将在一声霹雳中消灭，而汉诺塔、庙宇和众生也都将同归于尽。

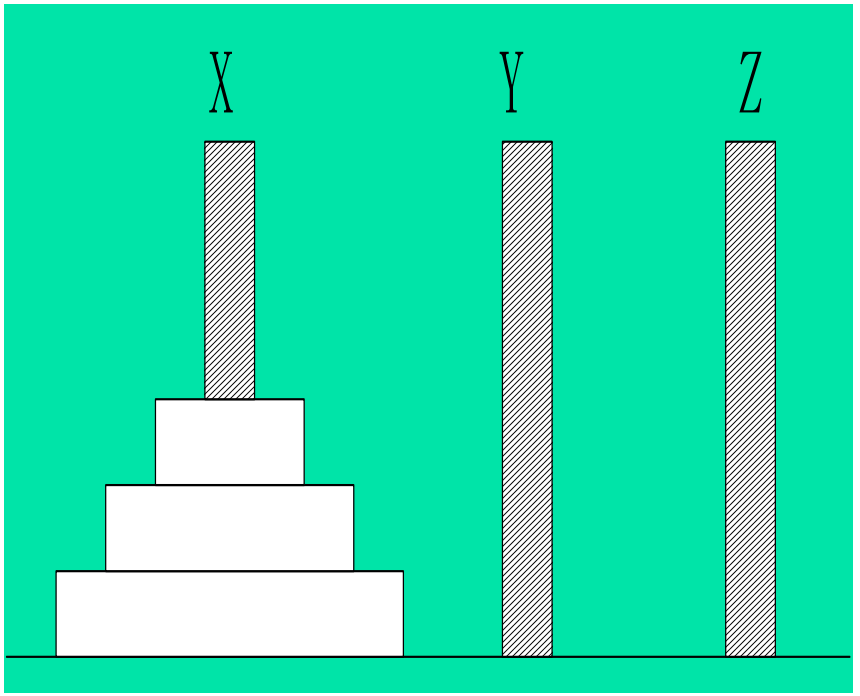
Hanoi塔问题

规则:

(1) 每次只能移动一个圆盘

(2) 圆盘可以插在X, Y和Z中的任一塔座上

(3) 任何时刻不可将较大圆盘压在较小圆盘之上



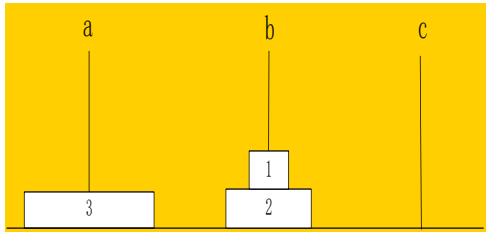
Hanoi塔问题

```
void hanoi(int n, char x, char y, char z)
    //将塔座x上由小到大且自上而下编号为1至n的n个圆盘按规
   // 则搬到塔座z上, y可用作辅助塔座
{
      if(n==1) move(x,1,z);          //将编号为1的圆盘从x搬到z
      else{
         hanoi(n-1,x,z,y);           //将x上编号为1至n-1的圆盘移到y,
                                     //z作辅助塔
         move(x,n,z);                //将编号为n的圆盘从x移到z
         hanoi(n-1,y,x,z);           //将y上编号为1至n-1的圆盘移到z,
                                     //x作辅助塔
      }
}
```

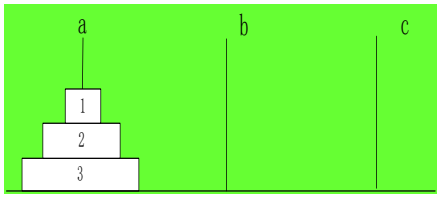
0

1

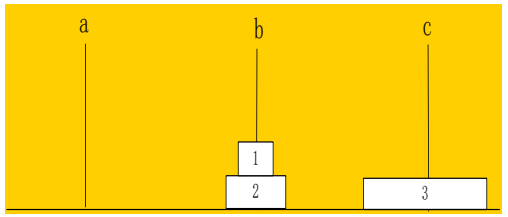
Hanoi(2,a,c,b)



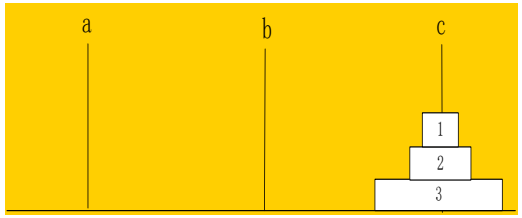
Hanoi(3,a,b,c)



move (a,3,c)



Hanoi(2,b,a,c)

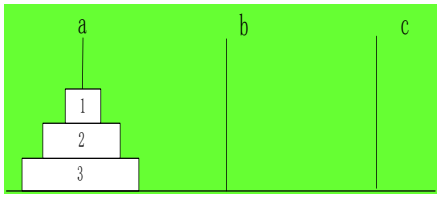


0

1

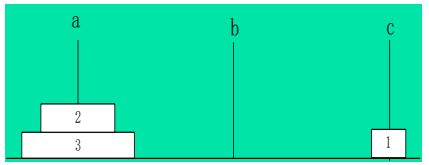
2

Hanoi(3,a,b,c)

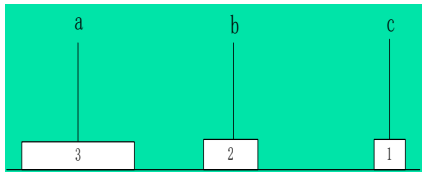


Hanoi(2,a,c,b)

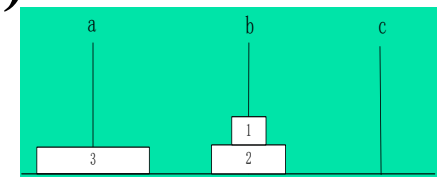
Hanoi(1,a,b,c)



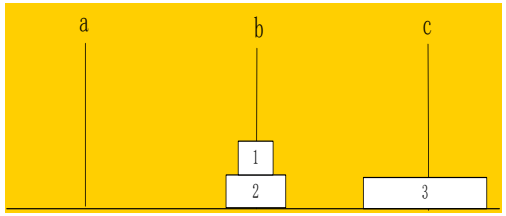
move(a,2,b)



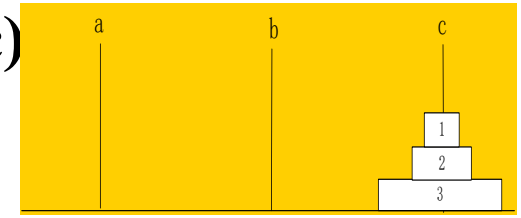
Hanoi(1,c,a,b)



move (a,3,c)



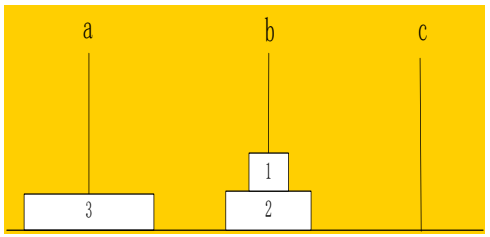
Hanoi(2,b,a,c)



0 1 2 3

Hanoi(1,a,b,c) → move(a,1,c)

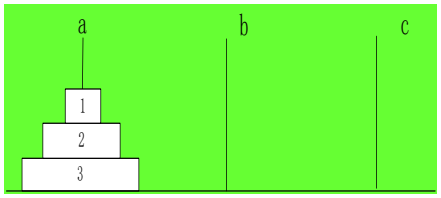
Hanoi(2,a,c,b)



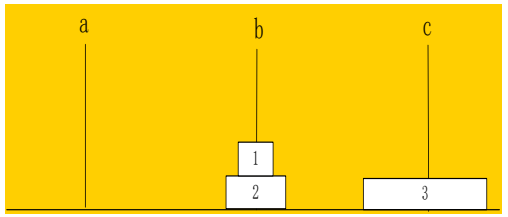
move(a,2,b)

Hanoi(1,c,a,b) → move(c,1,b)

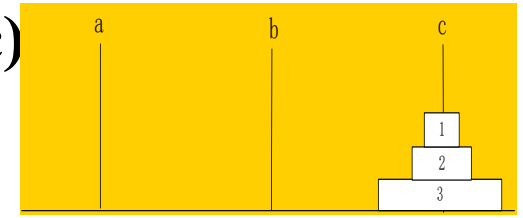
Hanoi(3,a,b,c)



move (a,3,c)

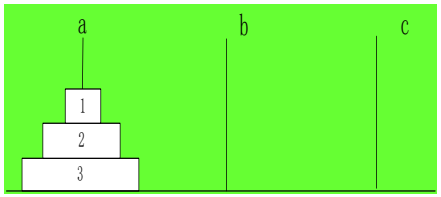


Hanoi(2,b,a,c)



0 1 2 3

Hanoi(3,a,b,c)



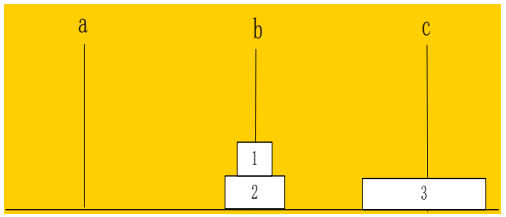
Hanoi(2,a,c,b)

Hanoi(1,a,b,c) → move(a,1,c)

move(a,2,b)

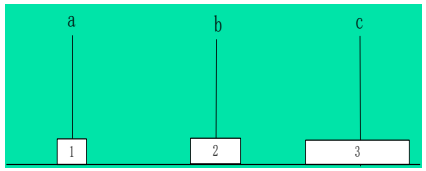
Hanoi(1,c,a,b) → move(c,1,b)

move (a,3,c)

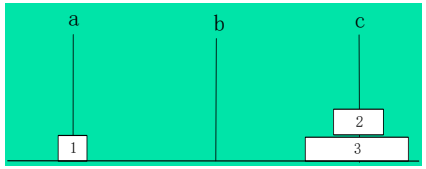


Hanoi(2,b,a,c)

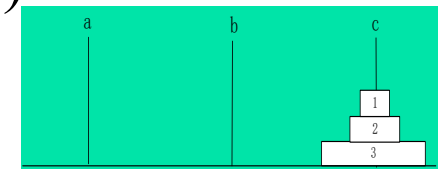
Hanoi(1,b,c,a)

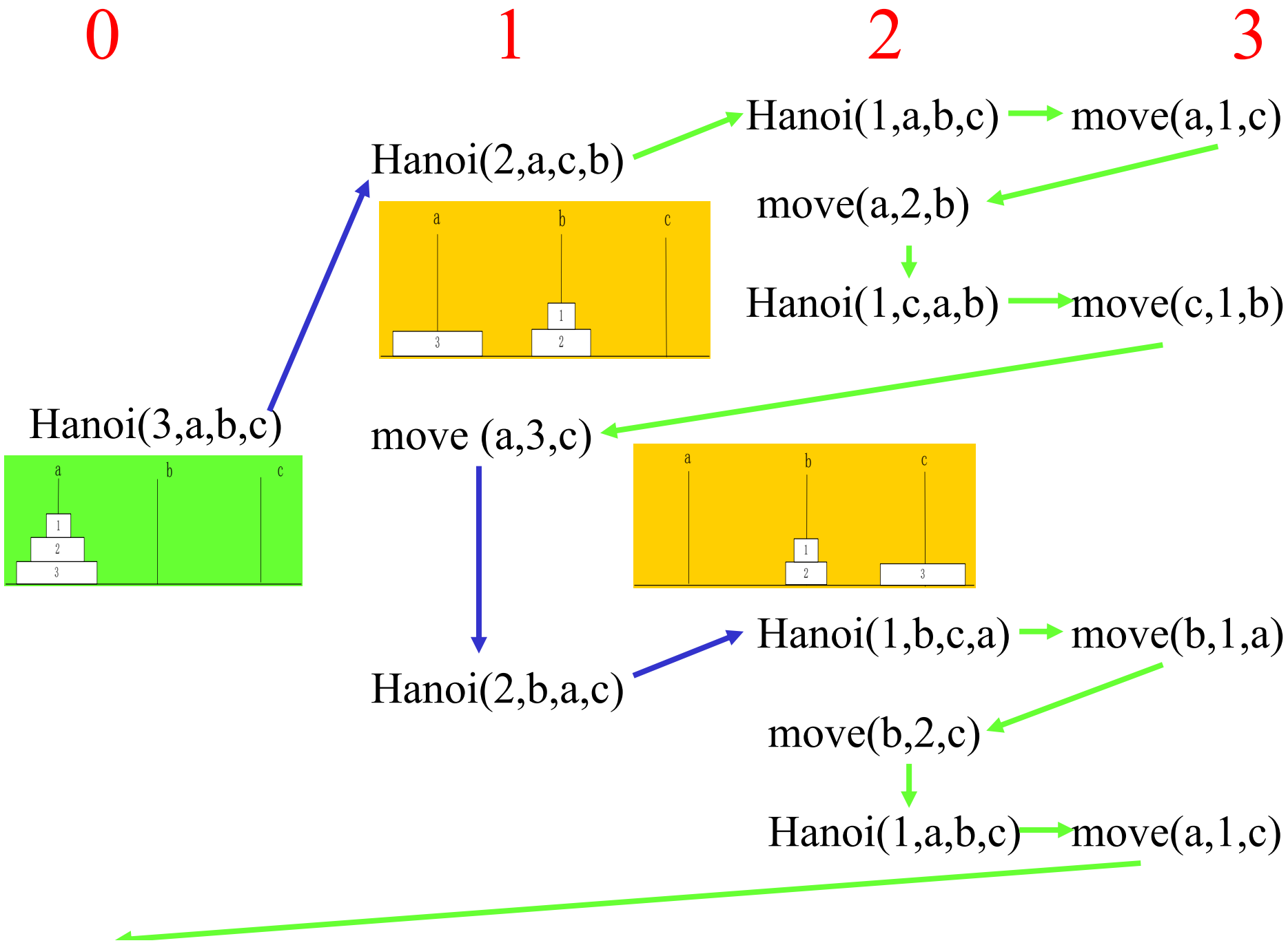


move(b,2,c)



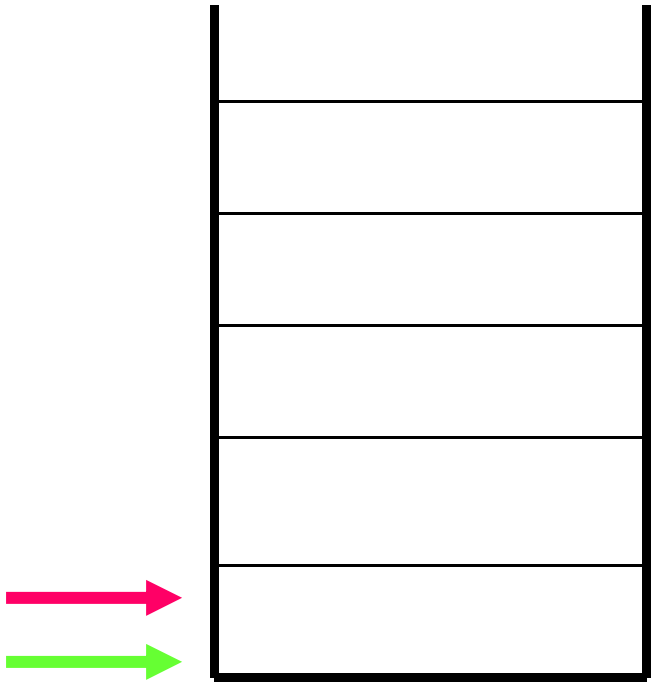
Hanoi(1,a,b,c)




0
1
2
3
Hanoi(3,a,b,c)
Hanoi(2,a,c,b)
Hanoi(1,a,b,c)
move(a,1,c)
move(a,2,b)
Hanoi(1,c,a,b)
move(c,1,b)
move (a,3,c)
Hanoi(2,b,a,c)
Hanoi(1,b,c,a)
move(b,1,a)
move(b,2,c)
Hanoi(1,a,b,c)
move(a,1,c)
a
b
c
1
2
3

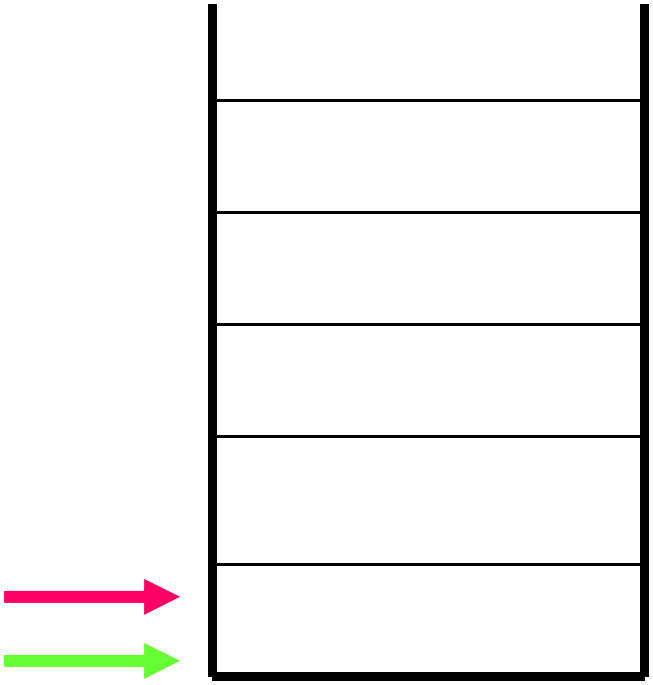
```
void main (void)
{
    int n;
    unsigned char a,b,c;

    n=3;
    a=1;   b=2;   c=3;
    hanoi(n, a, b, c);
0:
    .
    .
    .
}
```

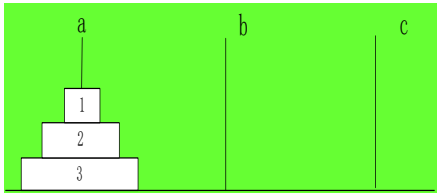
0 层



```
void main (void)
{
    int n;
    unsigned char a,b,c;

    n=3;
    a=1;   b=2;   c=3;
    hanoi(n, a, b, c);
0:
    .
    .
    .
}
```

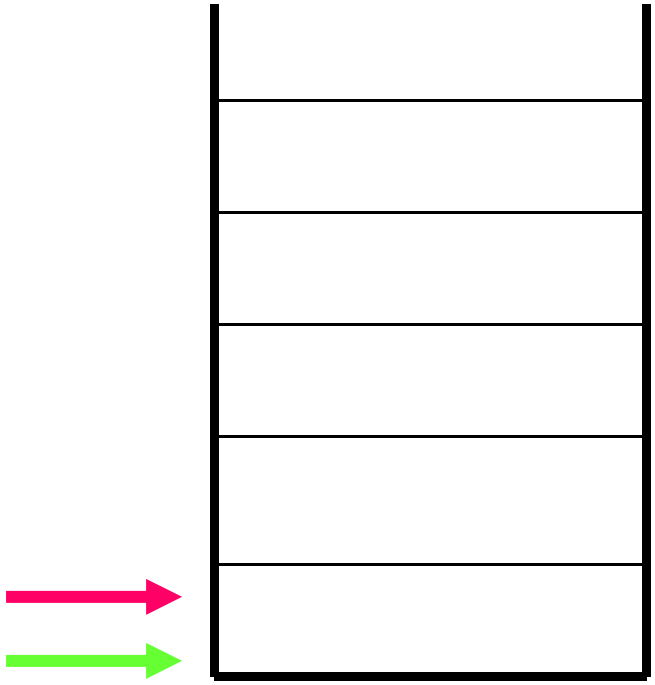
0 层

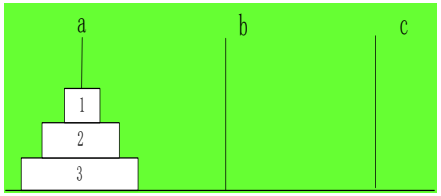




```
void main (void)
{
    int n;
    unsigned char a,b,c;

    n=3;
    a=1;   b=2;   c=3;
    hanoi(n, a, b, c);
0:
    .
    .
    .
}
```

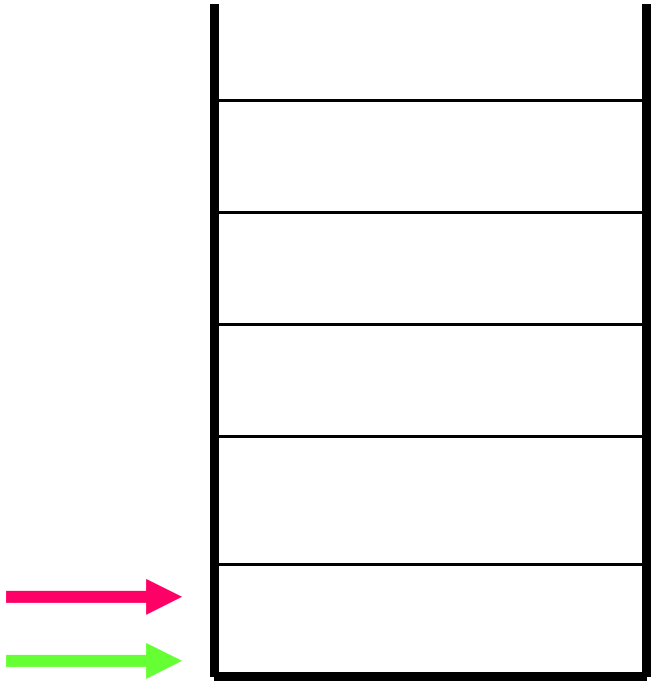
0 层

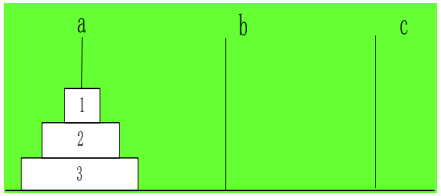




```
void main (void)
{
    int n;
    unsigned char a,b,c;

    n=3;
    a=1;   b=2;   c=3;
    hanoi(n, a, b, c);
0:
    .
    .
    .
}
```

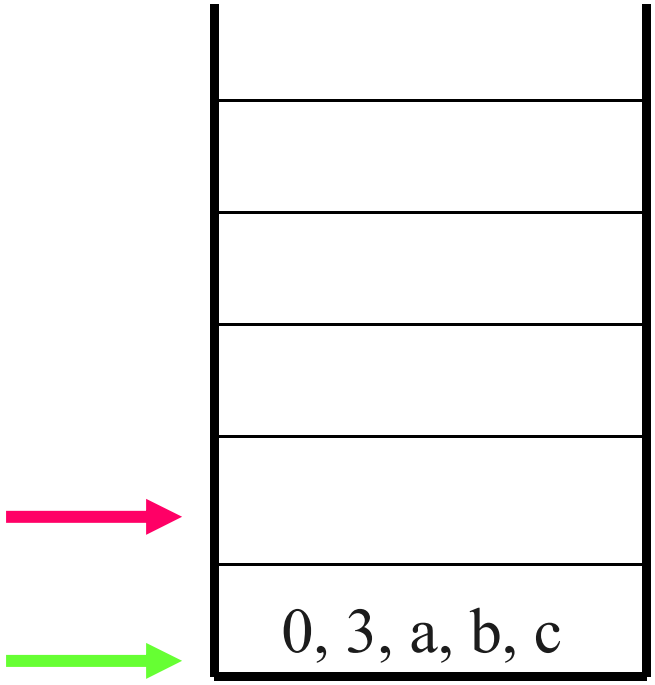
0 层

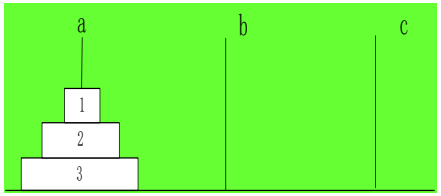




```
void main (void)
{
    int n;
    unsigned char a,b,c;

    n=3;
    a=1;   b=2;   c=3;
    hanoi(n, a, b, c);
0:
    .
    .
    .
}
```

0 层

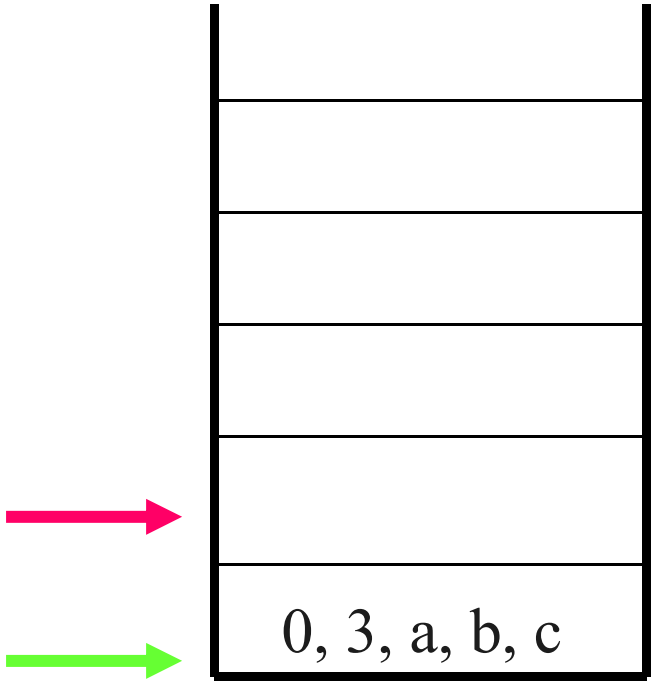


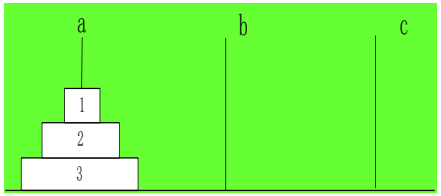


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

1 层

3, a, b, c

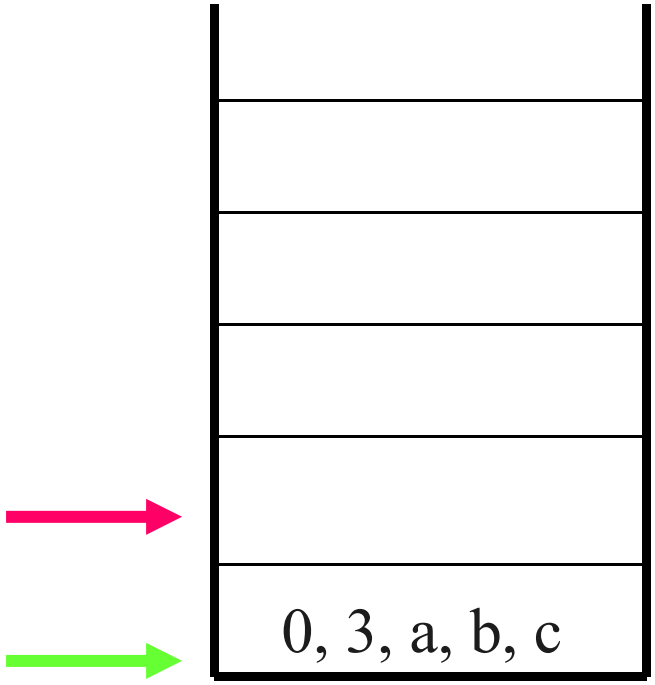


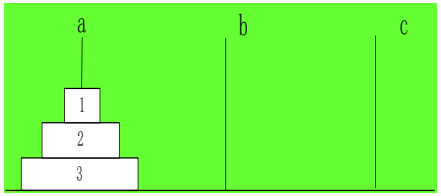


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

1 层

3, a, b, c

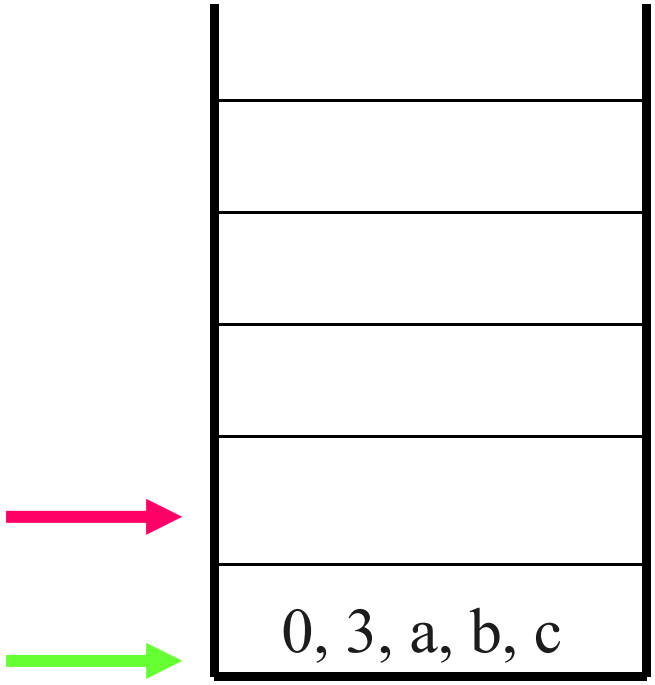


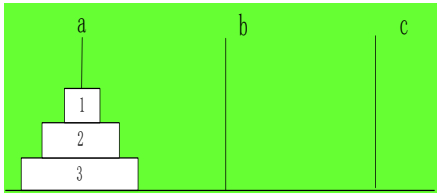


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

1 层

3, a, b, c



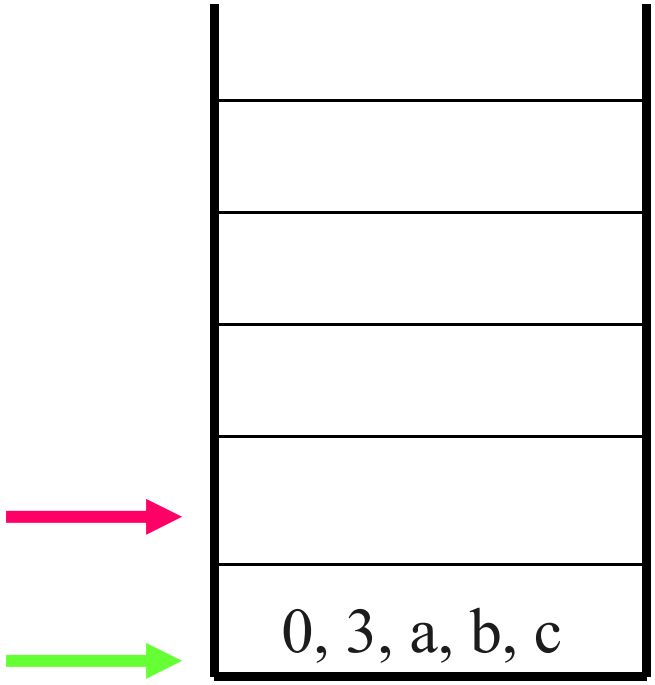


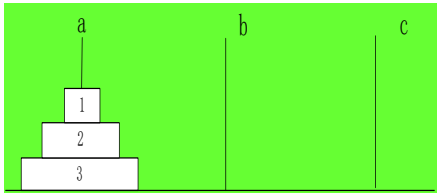
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);
6     move(x,n,z);
7     hanoi(n-1,y,x,z);
8      }
9  }
```

2, a, c, b

1 层

3, a, b, c



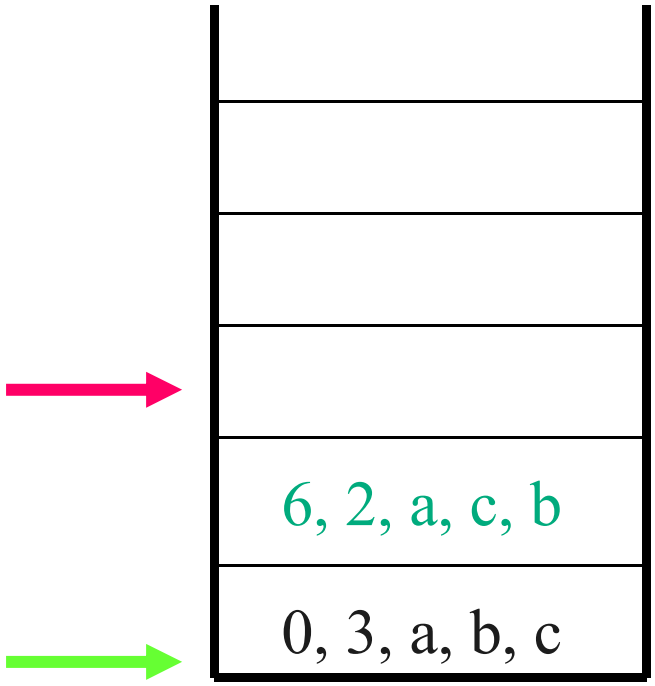


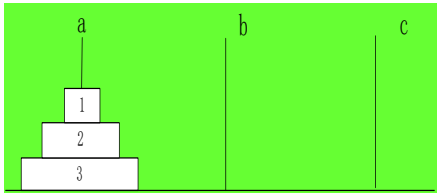
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);
6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2, a, c, b

1 层

3, a, b, c

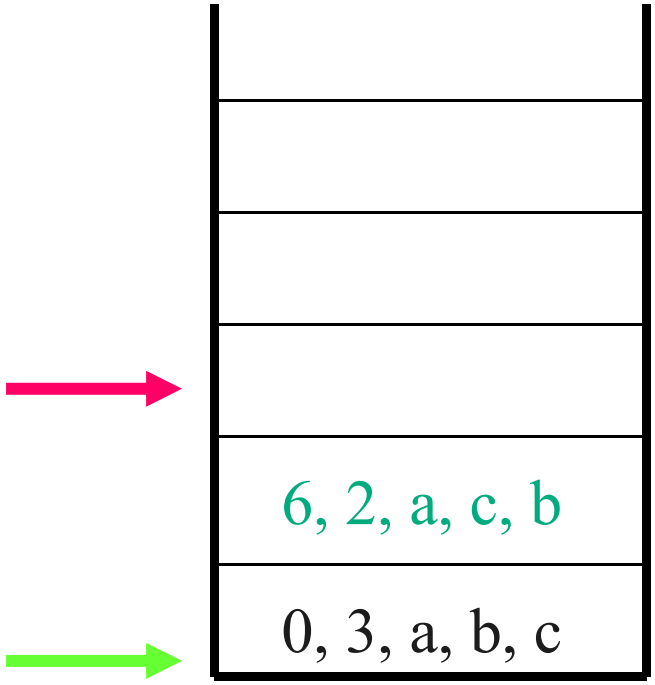


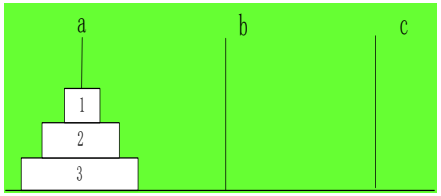


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, a, c, b

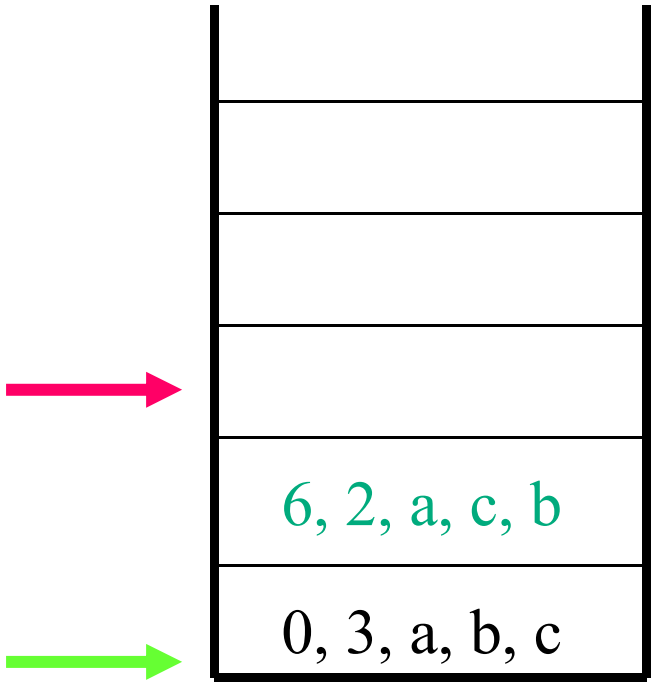


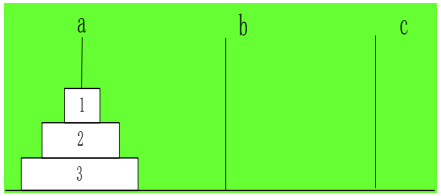


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, a, c, b

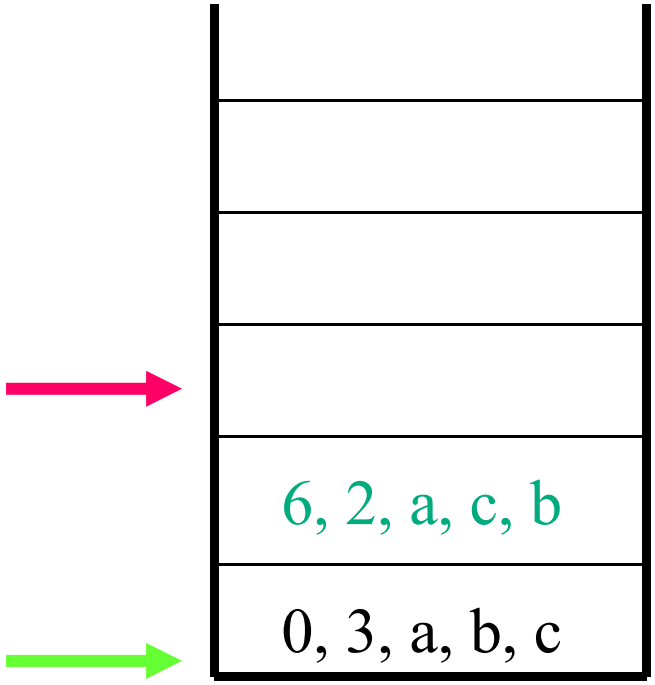


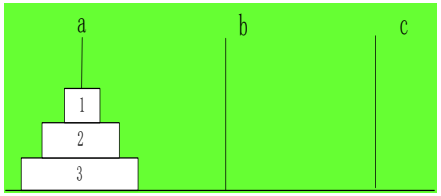


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, a, c, b



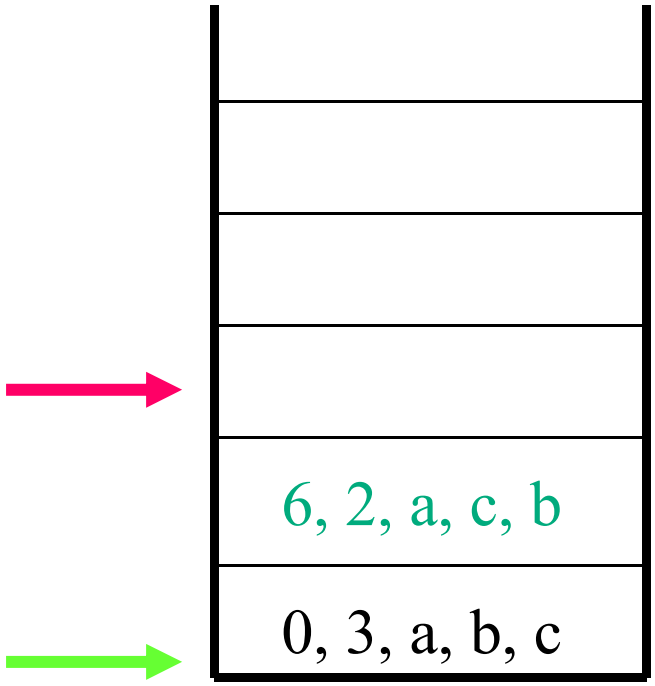


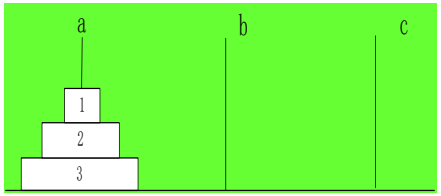
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);
6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

1, a, b, c

2 层

2, a, c, b



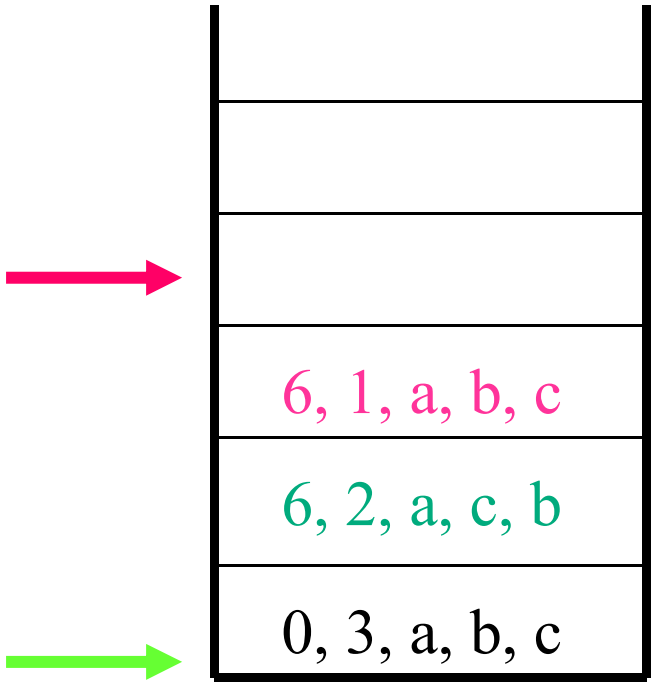


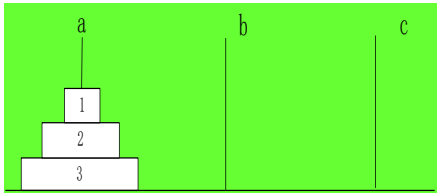
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);
6     move(x,n,z);
7     hanoi(n-1,y,x,z);
8      }
9  }
```

1, a, b, c

2 层

2, a, c, b

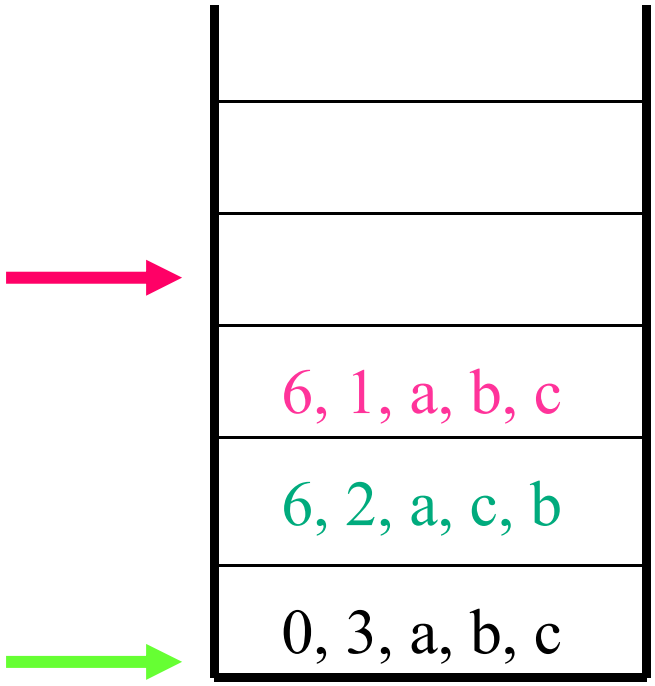


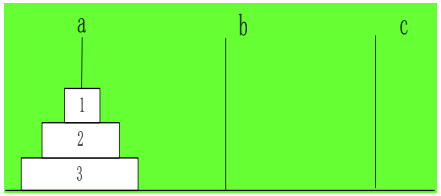


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c

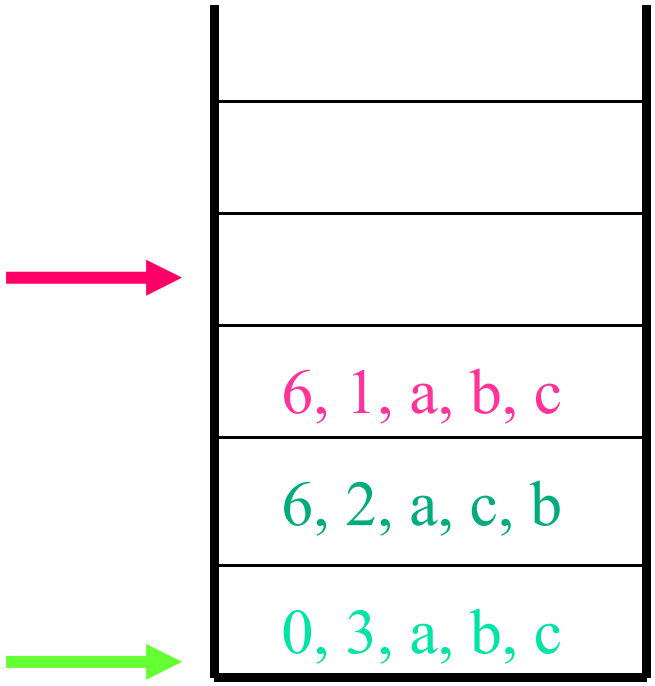


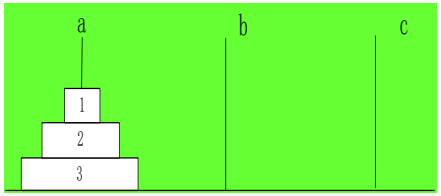


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c



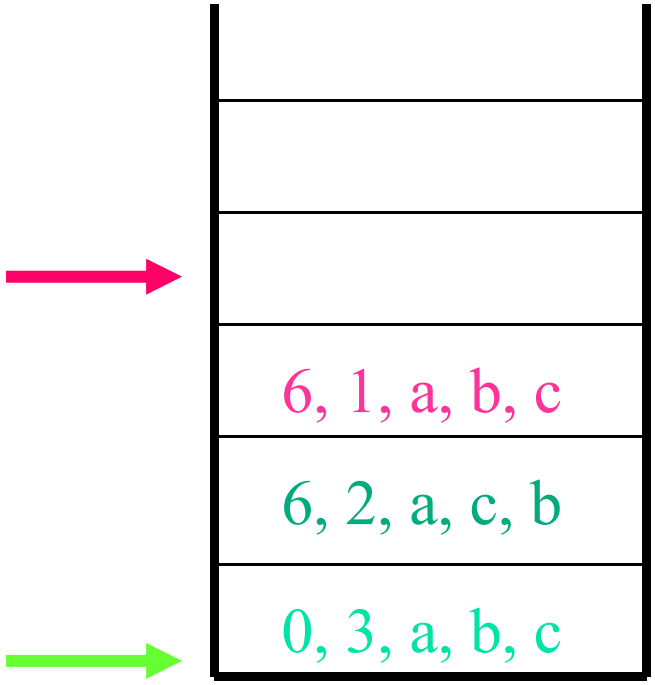


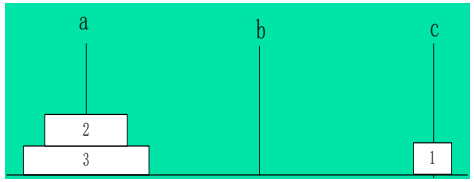
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c

a, 1, c

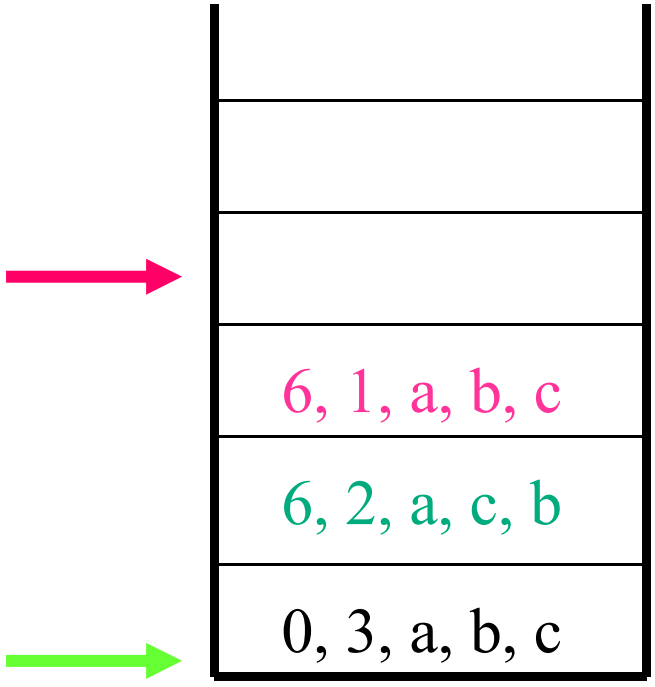


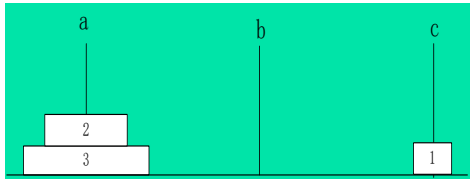


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c



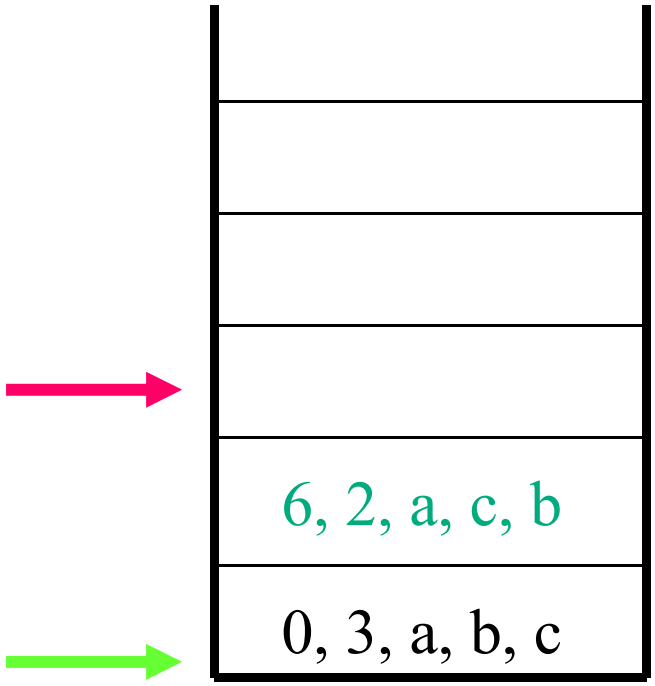


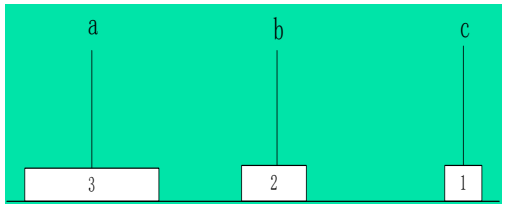
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

a, 2, b

2 层

2, a, c, b



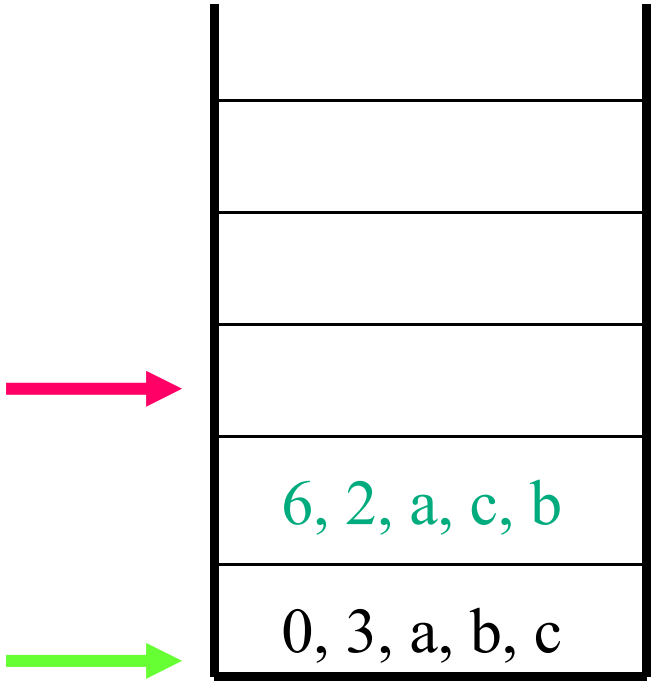


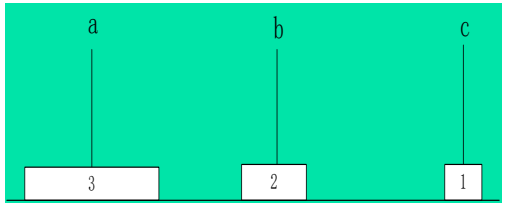
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

1,c,a,b

2 层

2, a, c, b



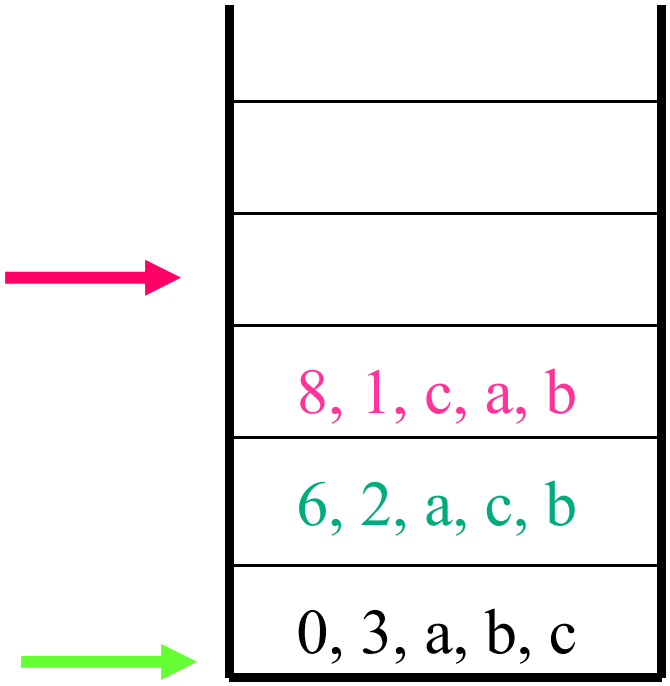


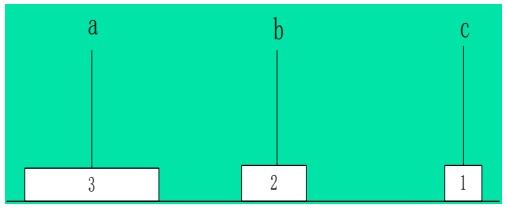
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

1,c,a,b

2 层

2, a, c, b

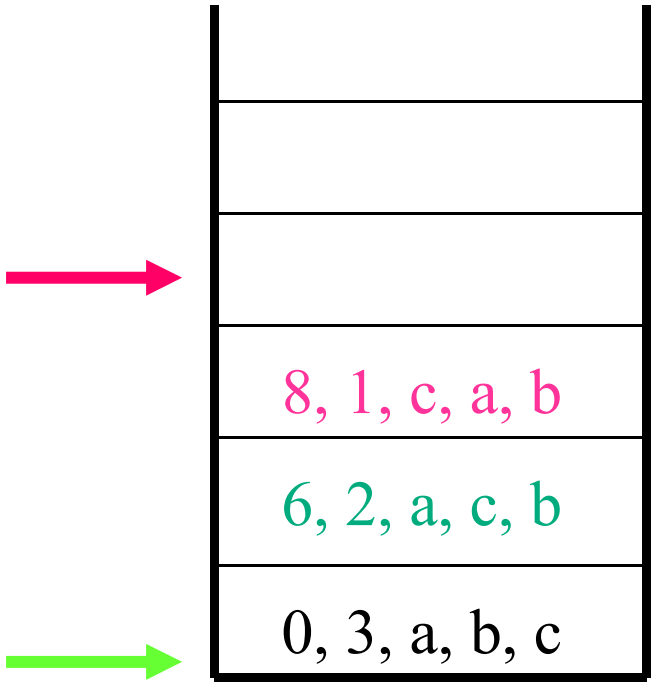


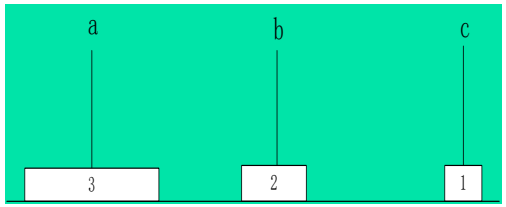


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, c, a, b

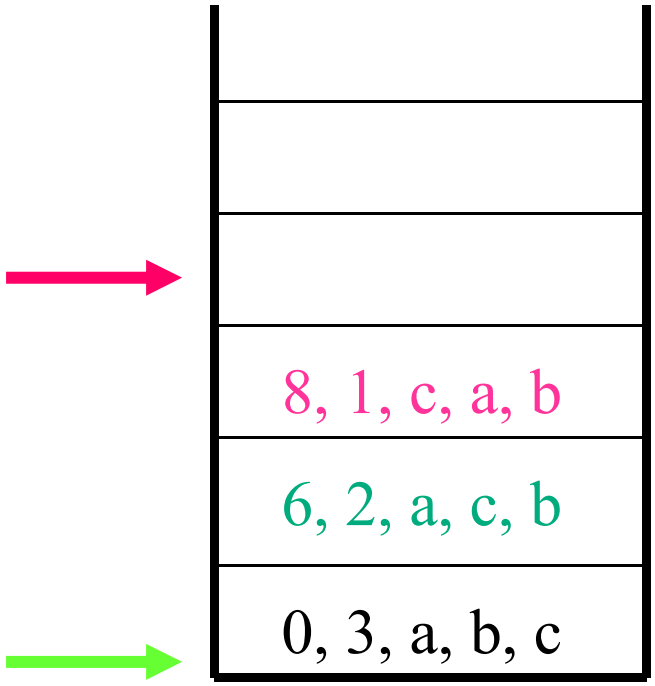


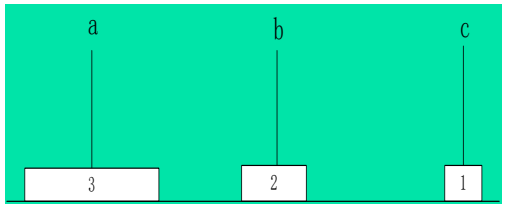


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, c, a, b



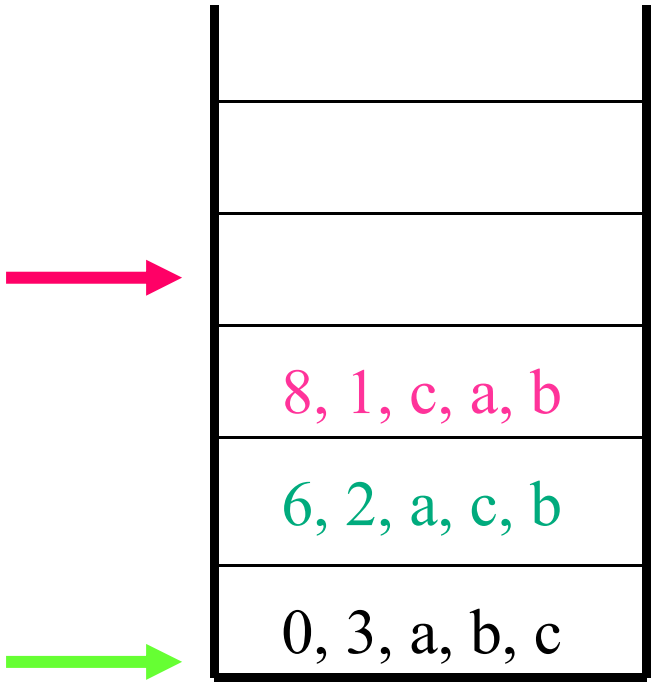


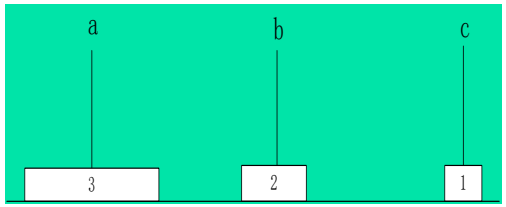
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);
6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

c, 1, b

3 层

1, c, a, b



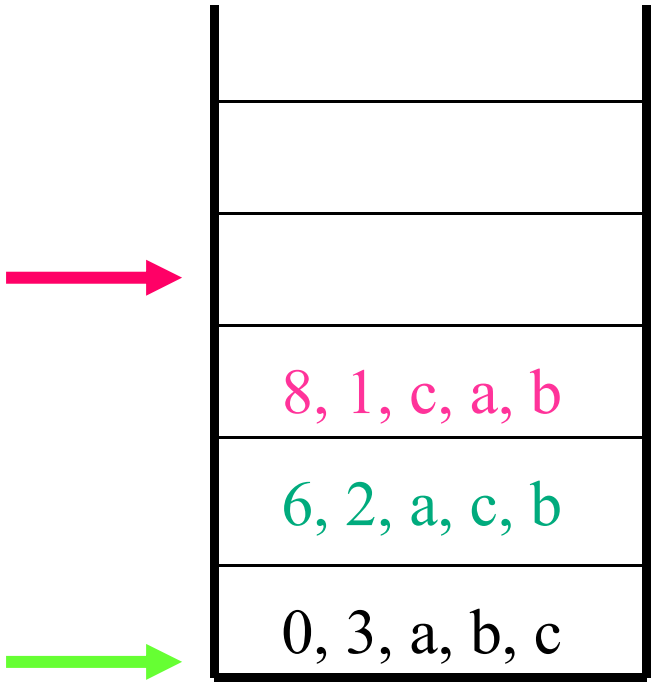


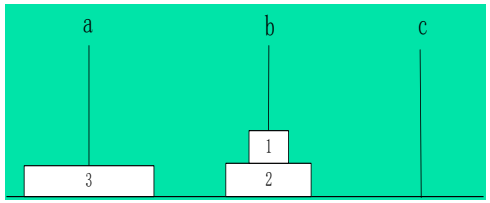
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);
6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

c, 1, b

3 层

1, c, a, b

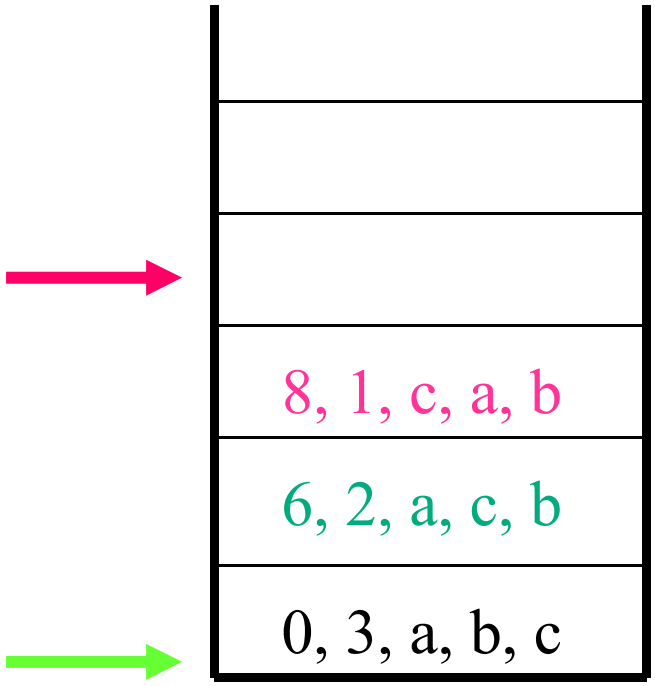


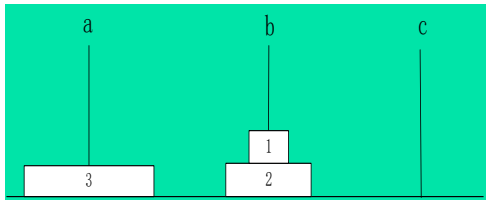


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, c, a, b

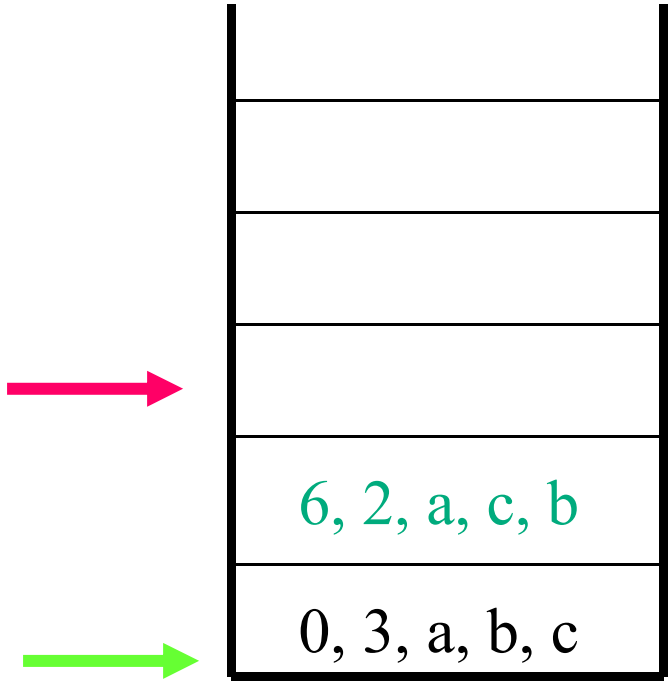


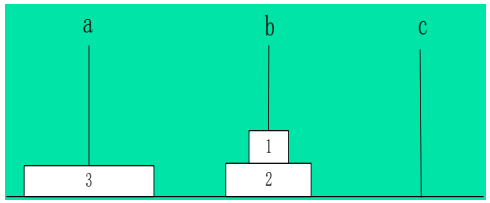


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

2 层

2, a, c, b

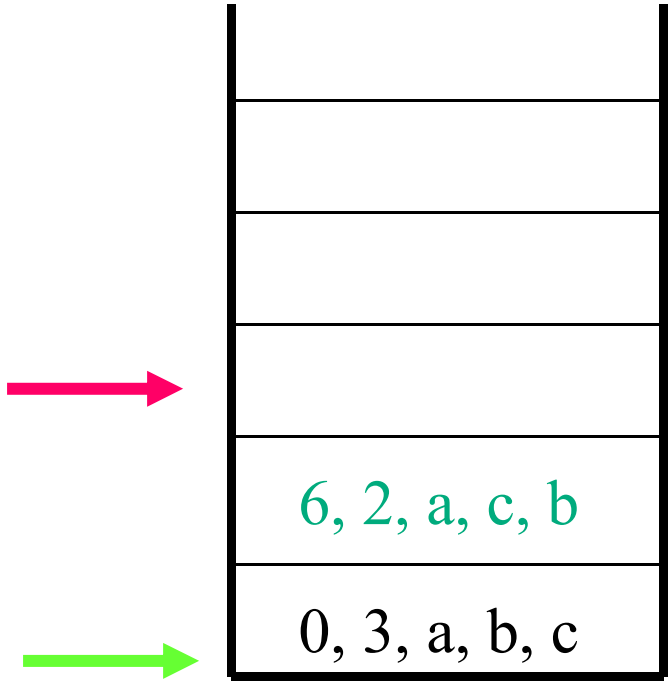


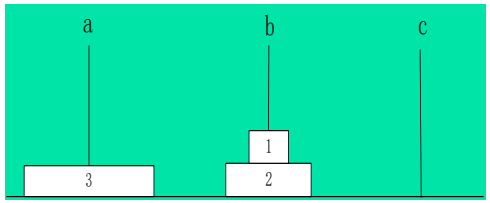


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, a, c, b



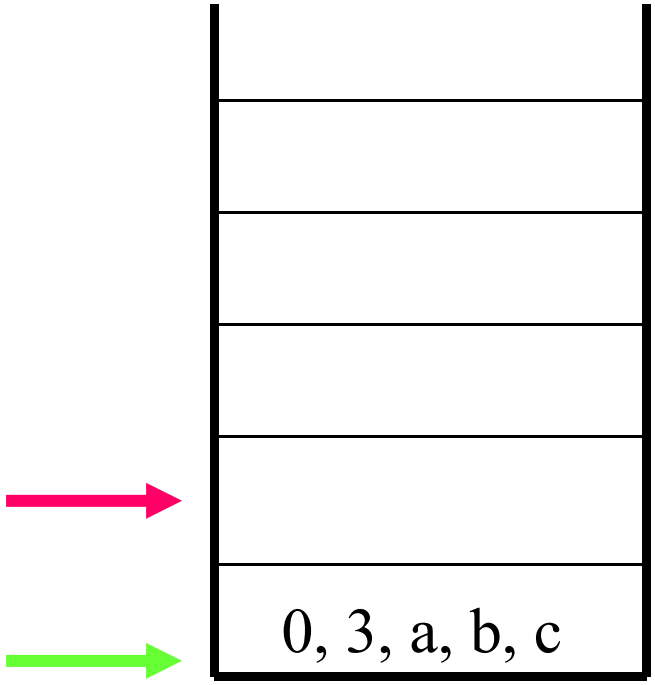


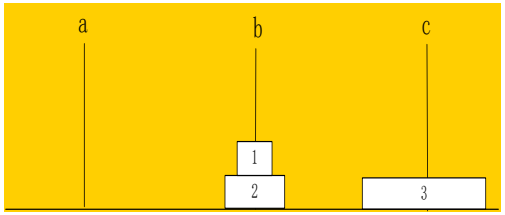
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

a, 3, c

1 层

3, a, b, c



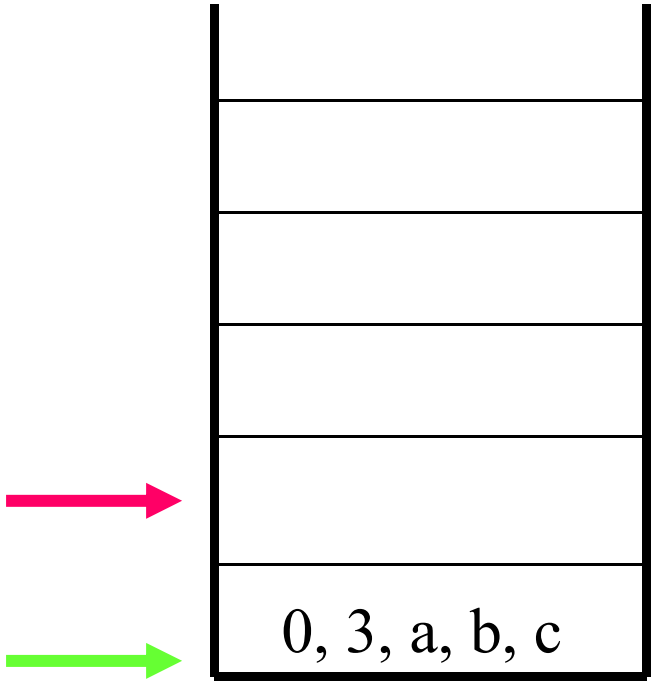


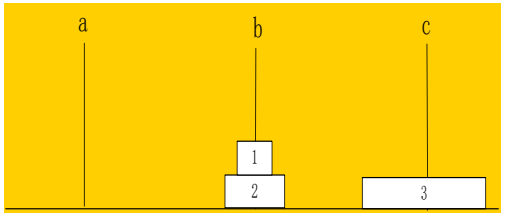
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

2, b, a, c

1 层

3, a, b, c



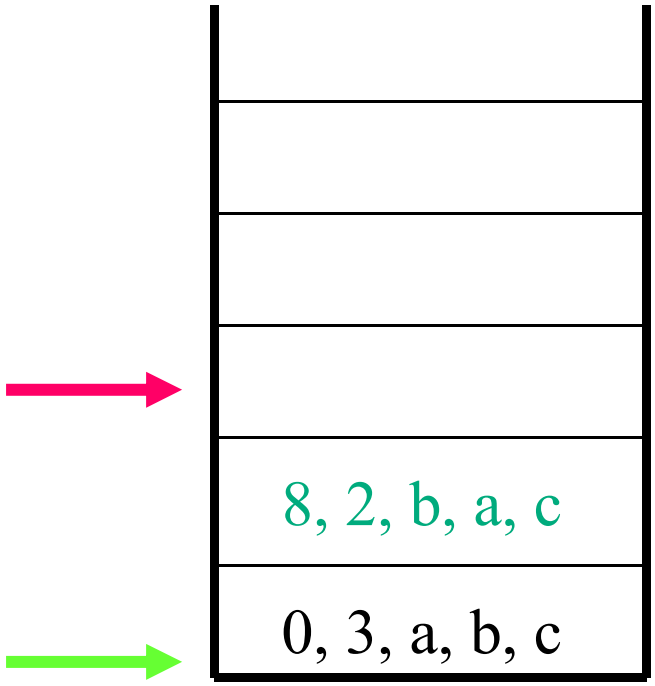


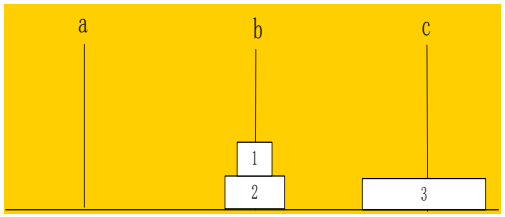
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

2, b, a, c

1 层

3, a, b, c

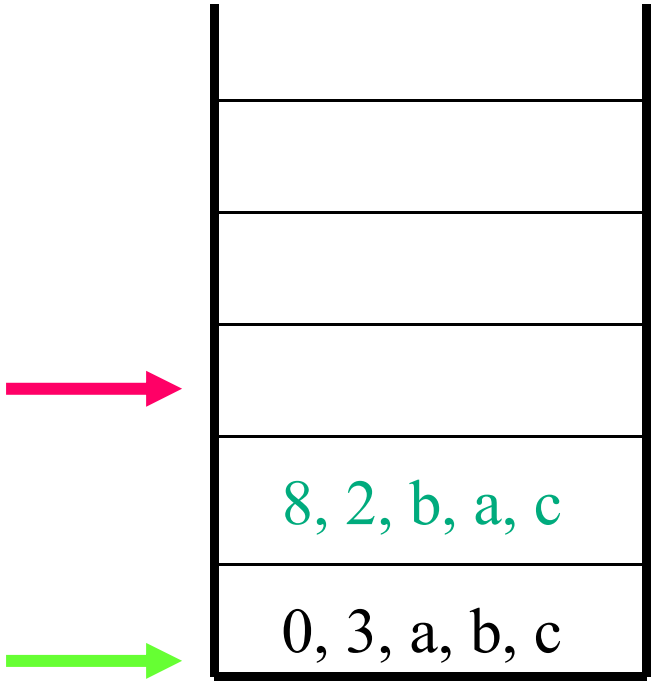


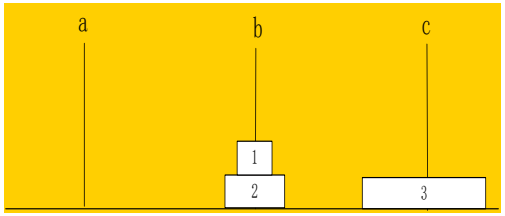


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, b, a, c

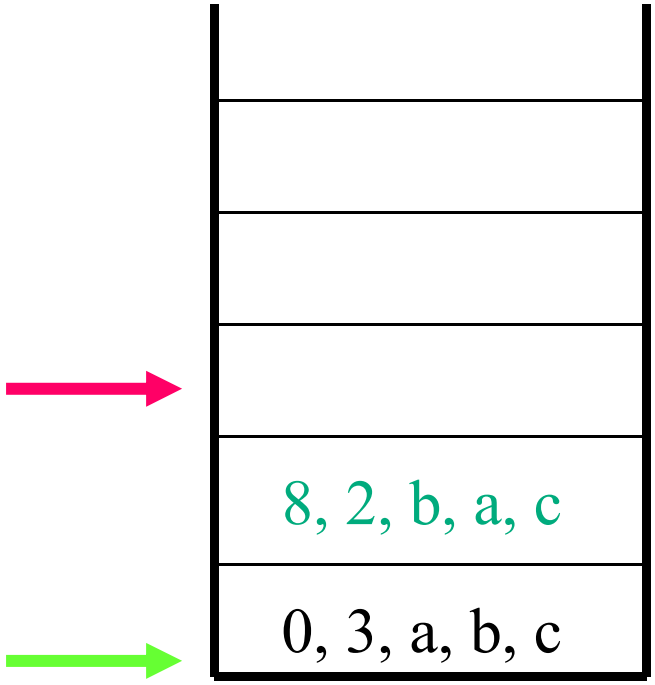


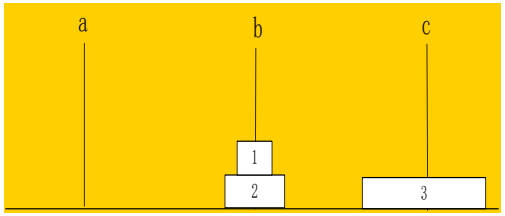


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, b, a, c

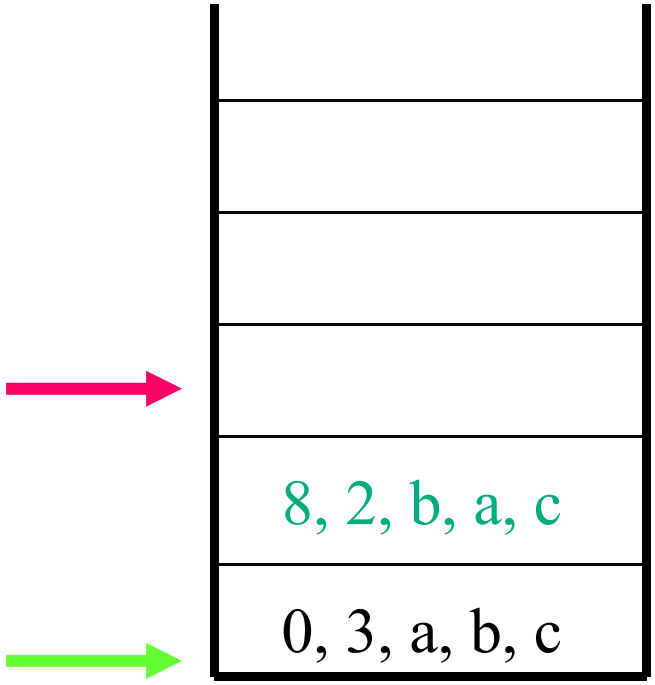


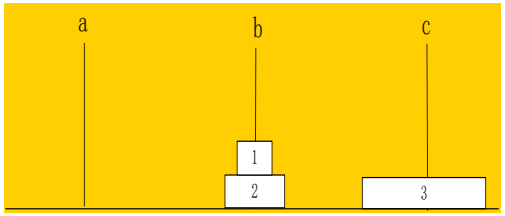


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, b, a, c



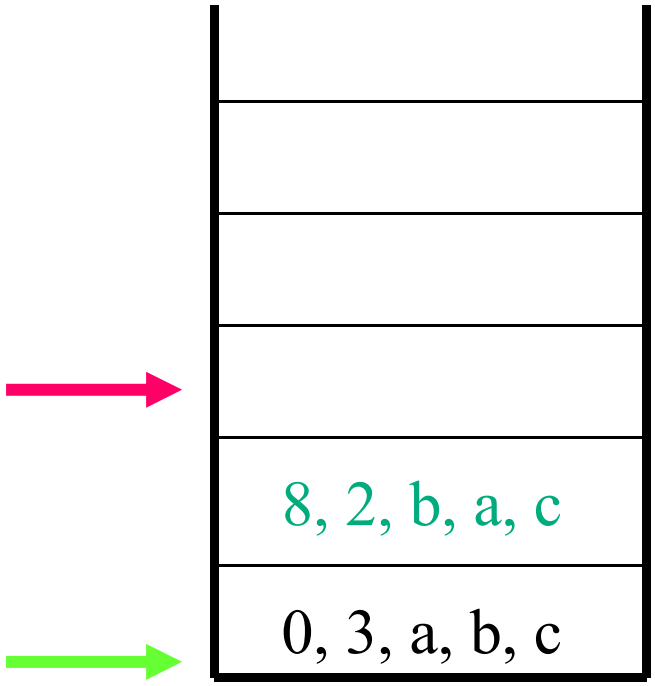


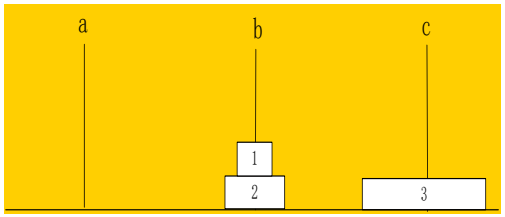
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);
6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

1, b, c, a

2 层

2, b, a, c



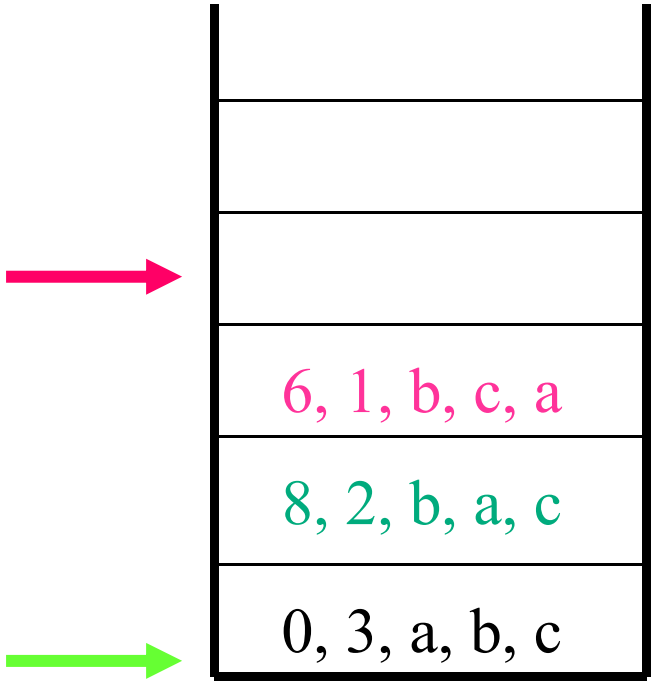


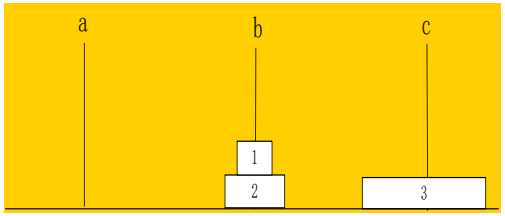
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);
6     move(x,n,z);
7     hanoi(n-1,y,x,z);
8      }
9  }
```

1, b, c, a

2 层

2, b, a, c

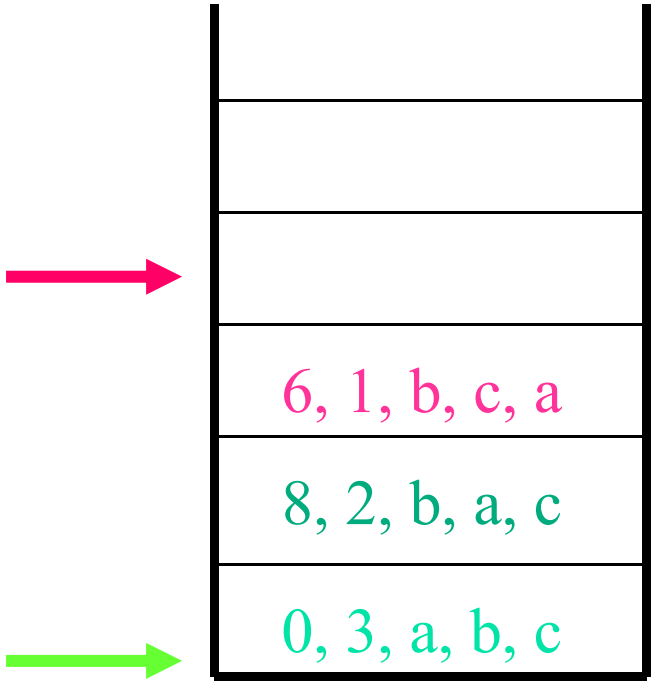


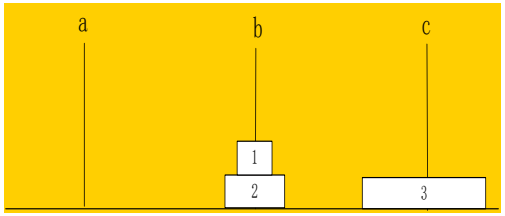


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, b, c, a

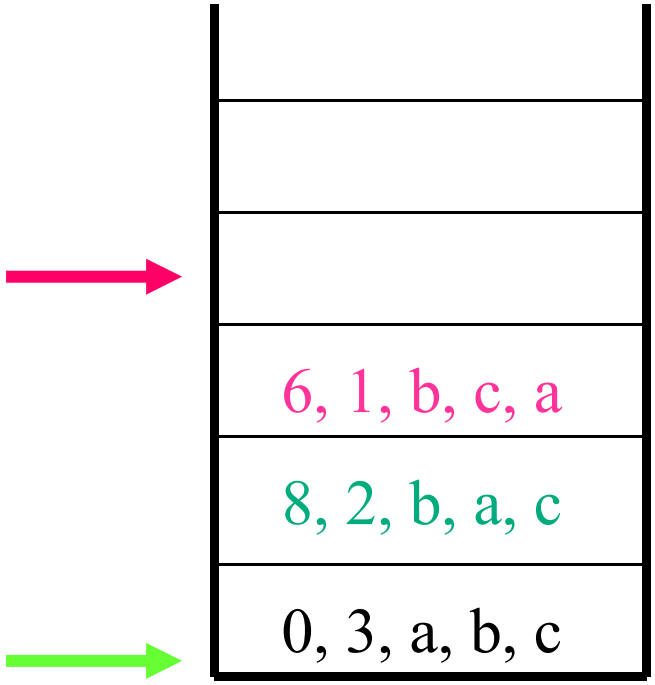


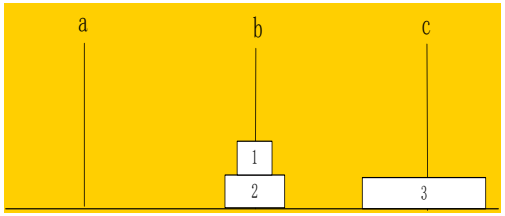


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, b, c, a



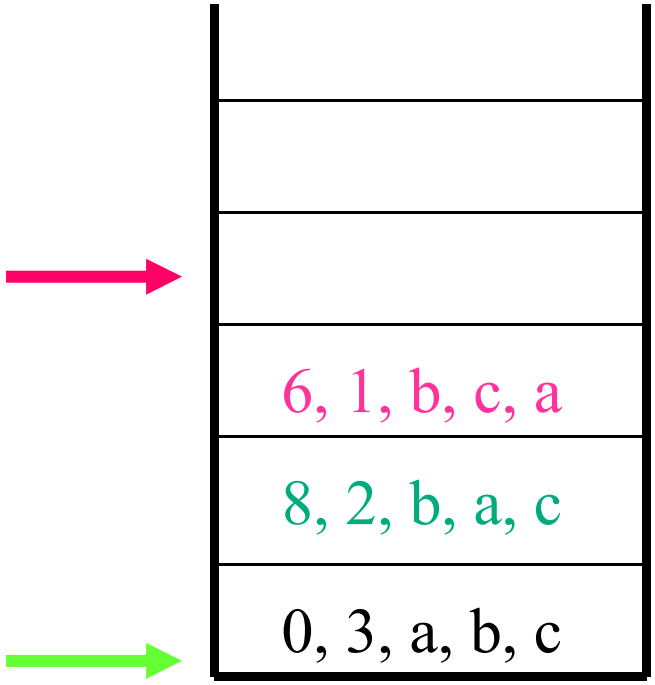


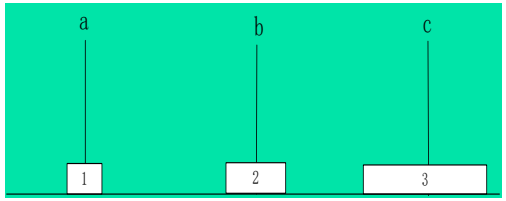
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, b, c, a

b, 1, a

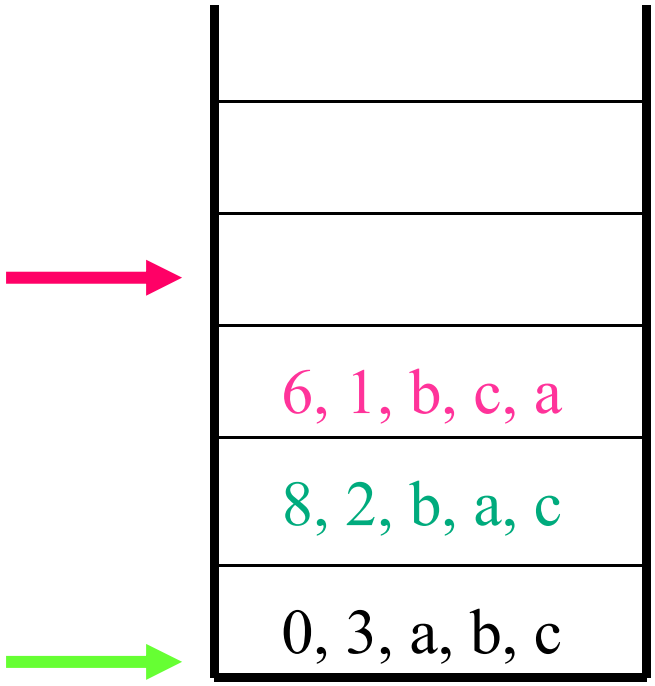


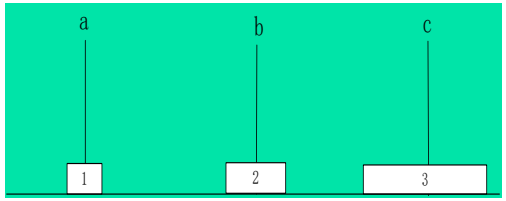


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, b, c, a



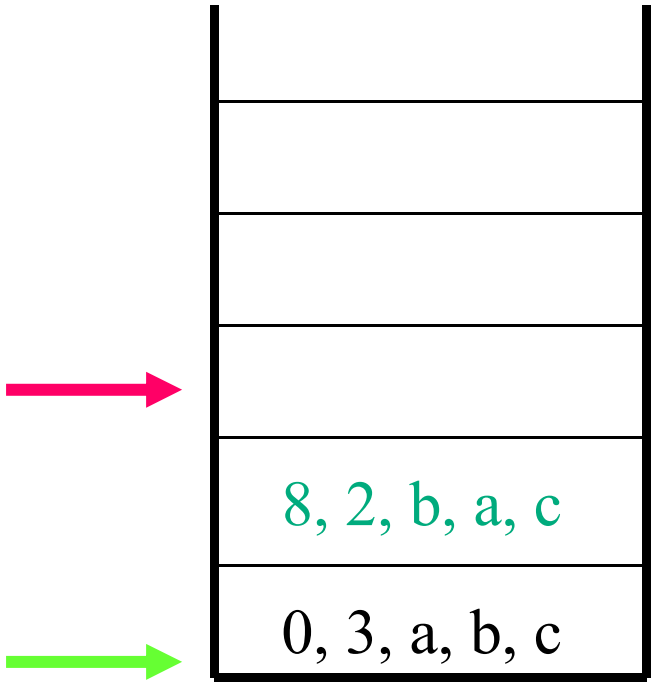


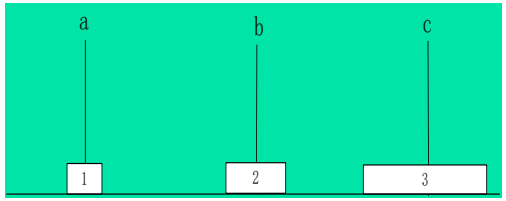
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

b,2,c

2 层

2, b, a, c



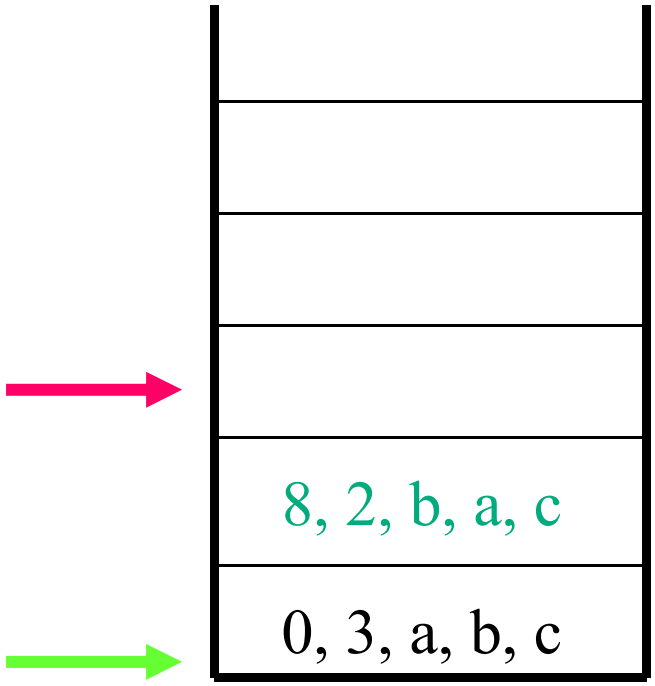


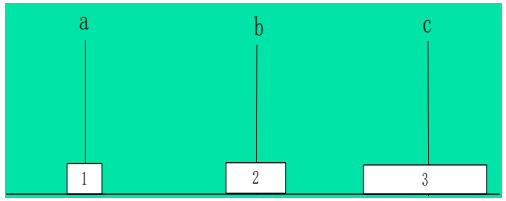
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

1,a,b,c

2 层

2, b, a, c



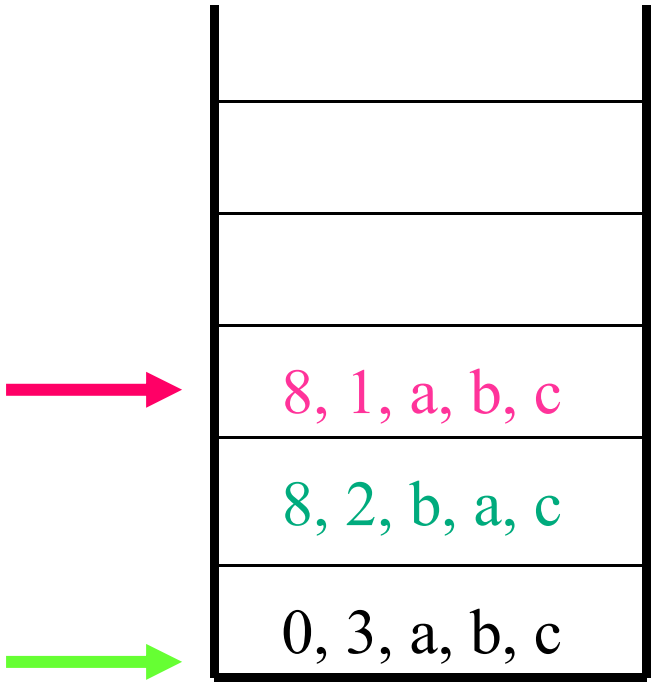


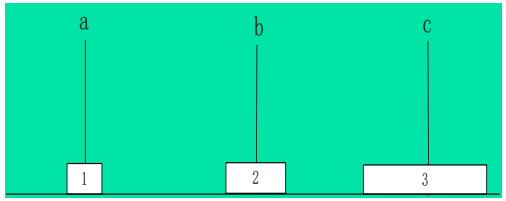
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

1,a,b,c

2 层

2, b, a, c

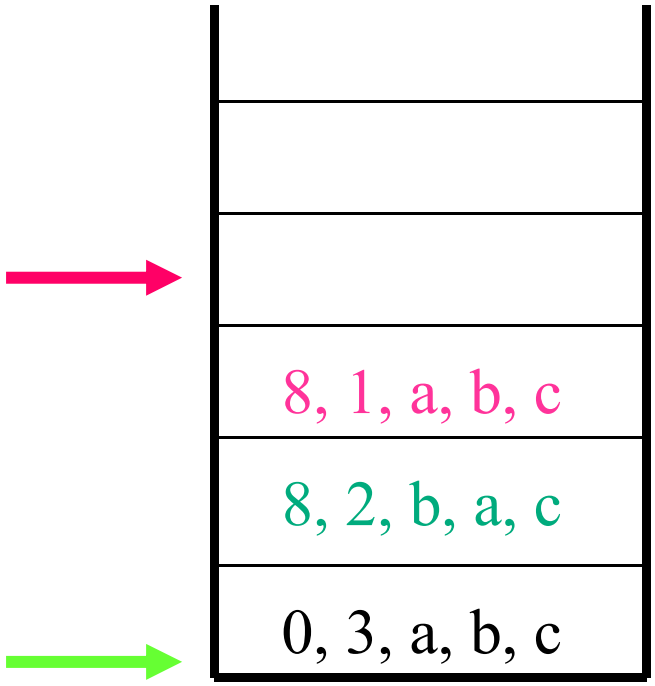


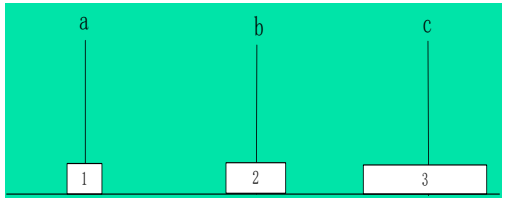


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c

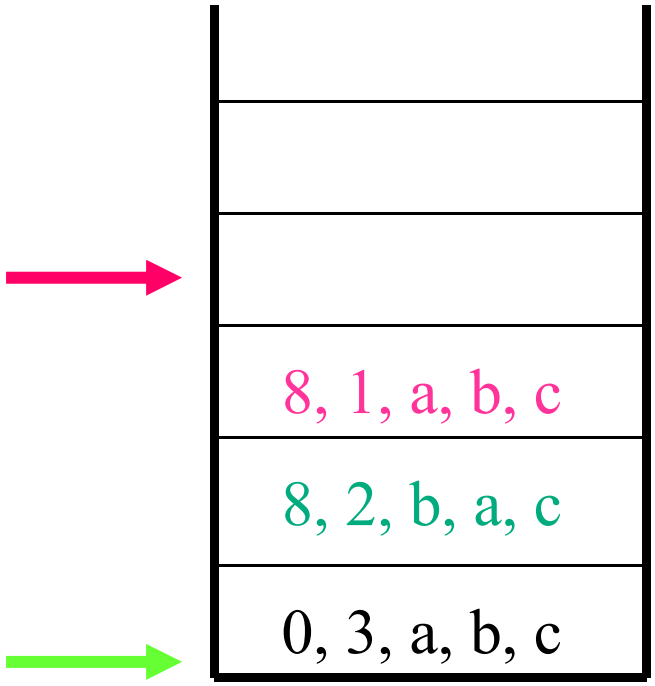


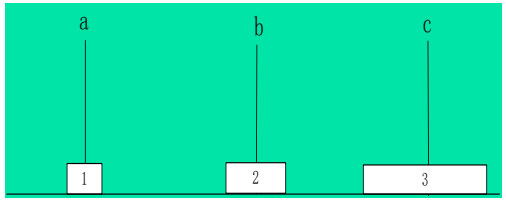


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);


6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c



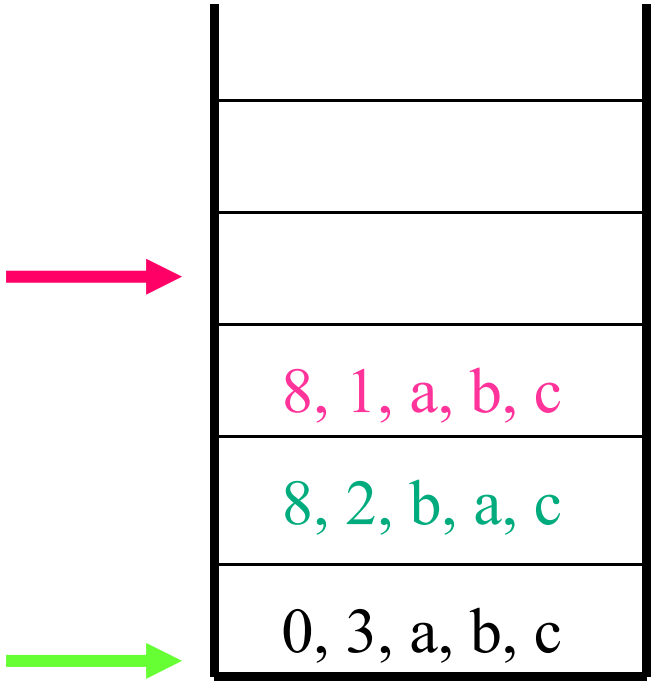


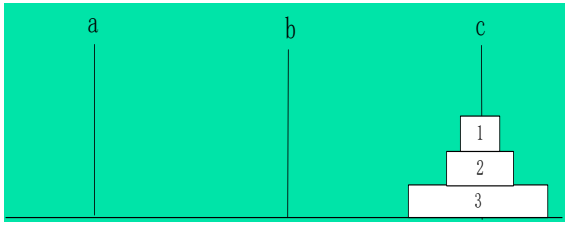
```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3          move(x,1,z);
4      else{
5          hanoi(n-1,x,z,y);
6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

a, 1, c

3 层

1, a, b, c

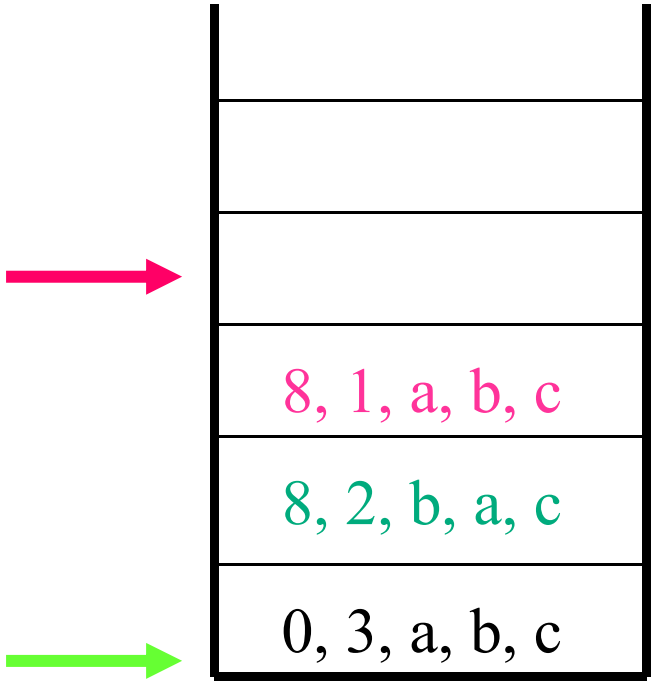


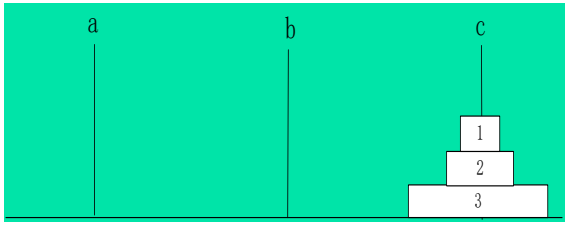


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

3 层

1, a, b, c

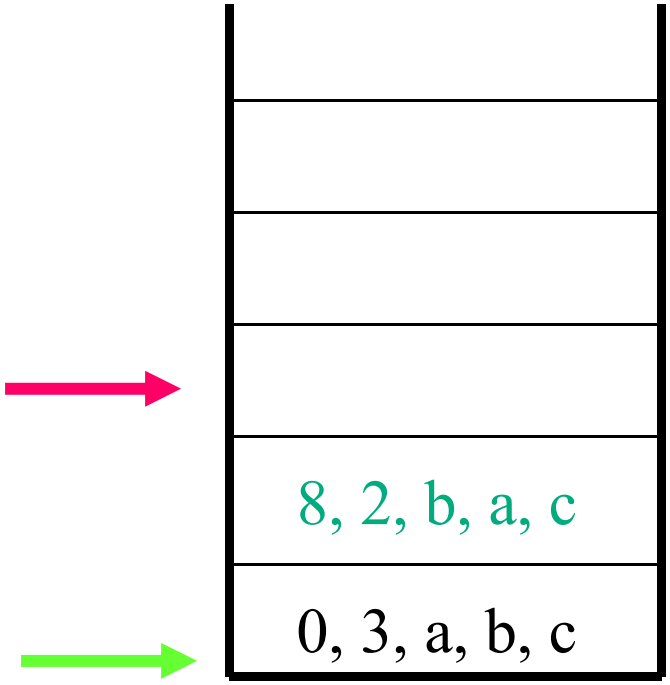


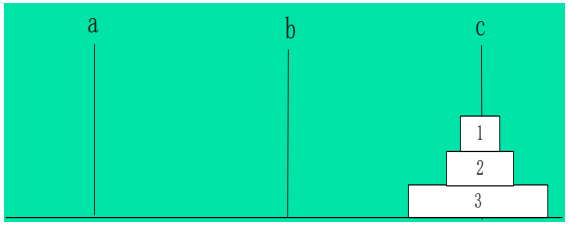


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

2 层

2, b, a, c

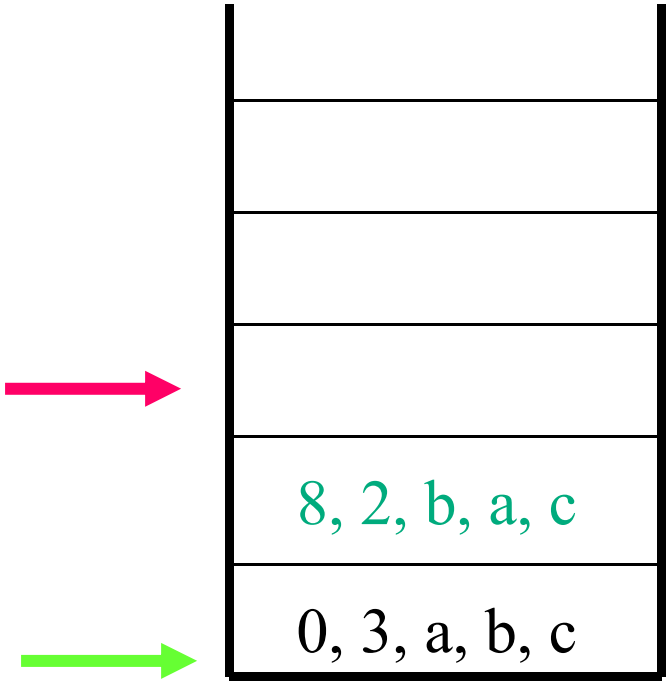


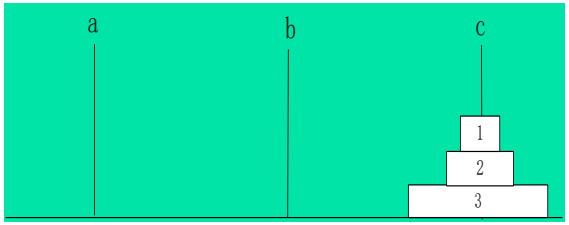


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

2 层

2, b, a, c

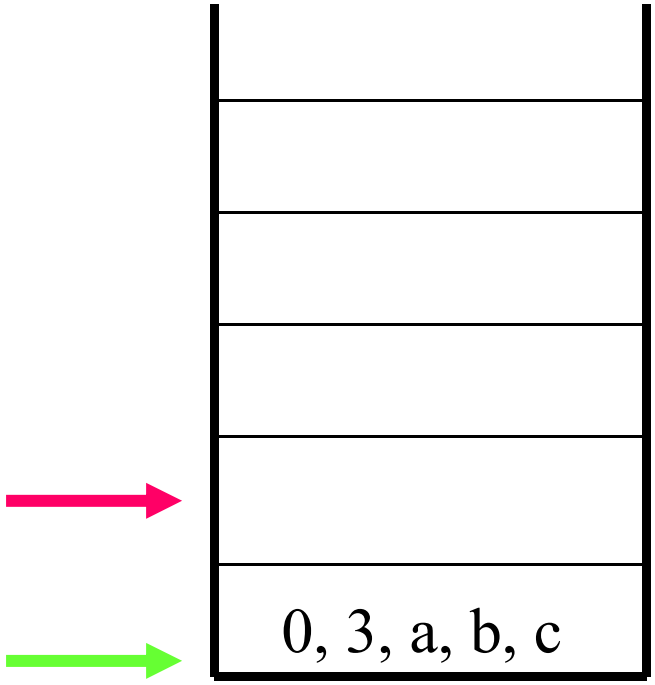


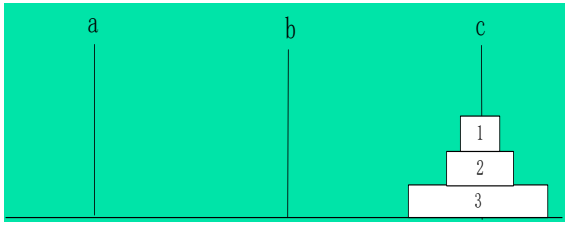


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8      }
9  }
```

1 层

3, a, b, c

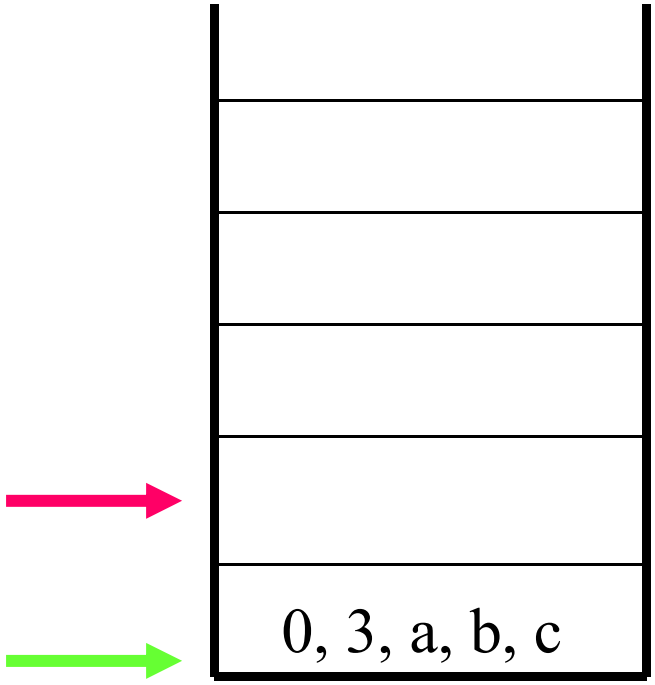


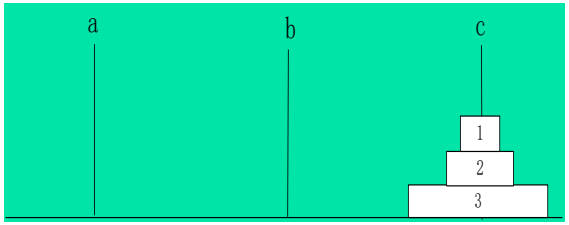


```
void hanoi(int n, char x, char y, char z)
1  {
2      if(n==1)
3         move(x,1,z);
4      else{
5         hanoi(n-1,x,z,y);

6      move(x,n,z);
7      hanoi(n-1,y,x,z);
8       }
9  }
```

1 层

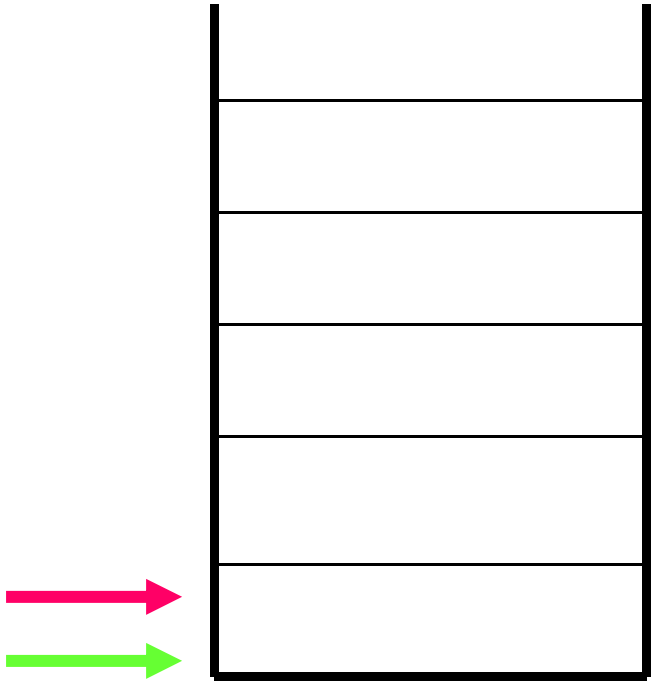
3, a, b, c

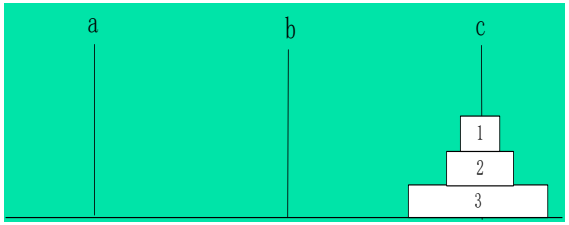




```
void main (void)
{
    int n;
    unsigned char a,b,c;

    n=3;
    a=1;   b=2;   c=3;
    hanoi(n, a, b, c);
0:
    .
    .
    .
}
```

0 层





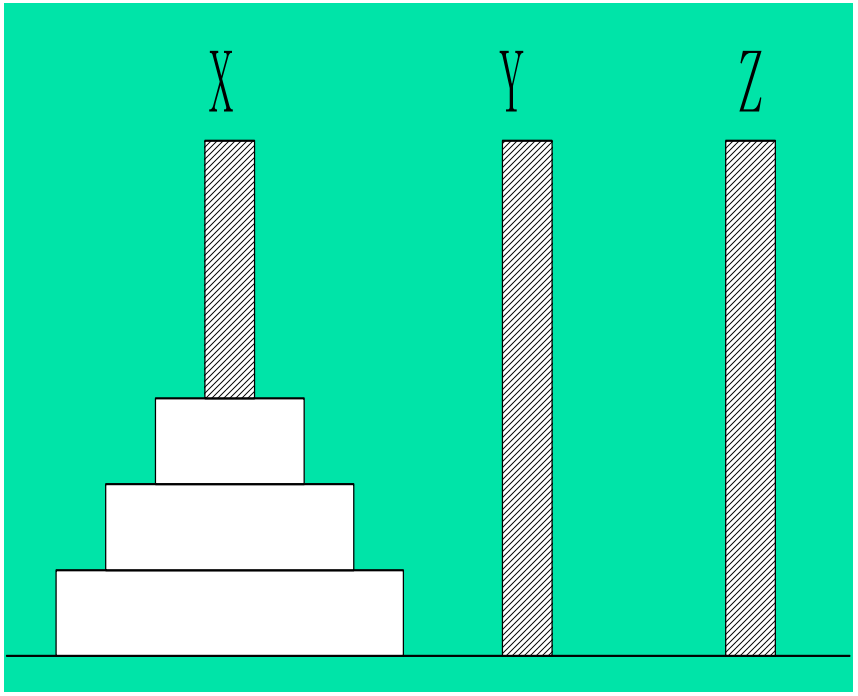
思考题

如果盘子个数为n, 求最小移动次数F(n) ？

(1) n-1, X→Y (Z) **F(n-1)次**

(2) 1, X→Y **1次**

(3) n-1, Y→Z (X) **F(n-1)次**

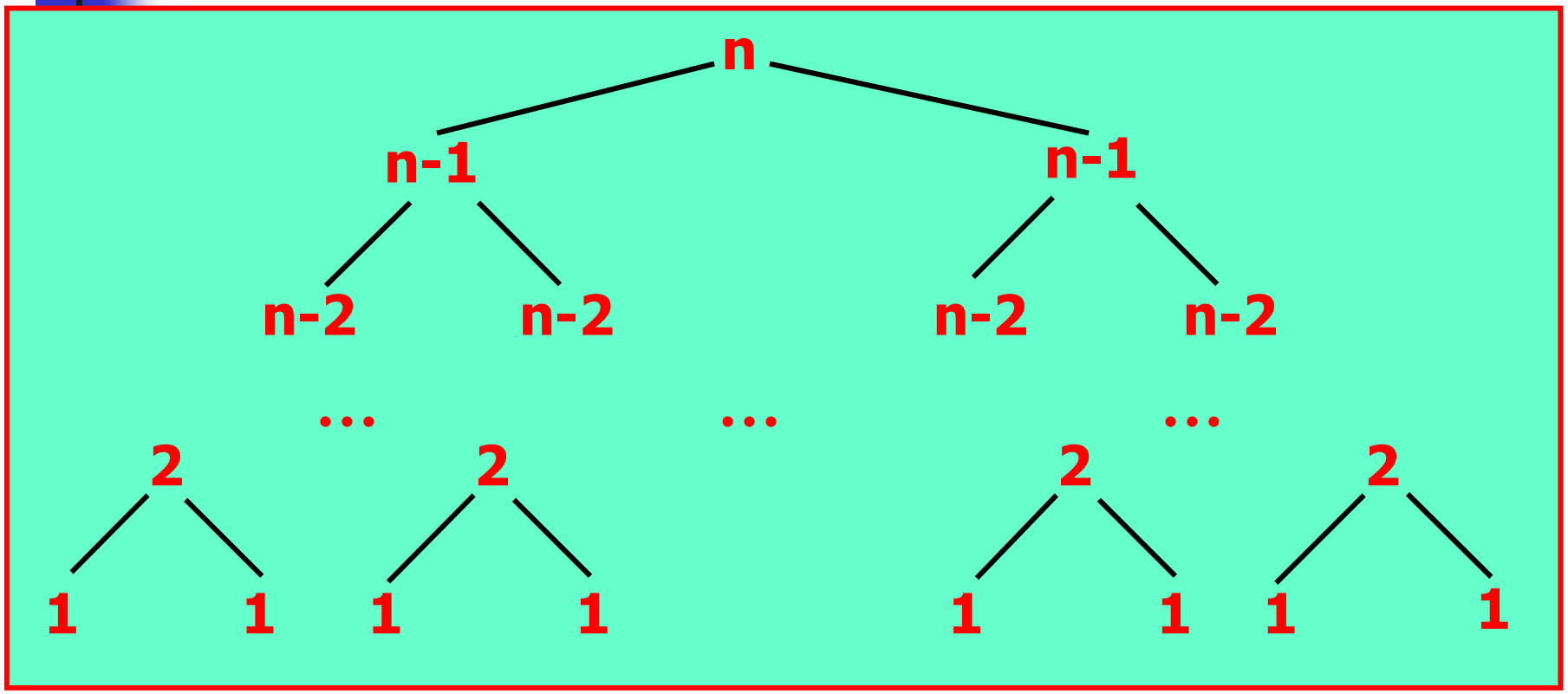


$$F(n)=2F(n-1)+1$$

$$=2(2F(n-2)+1)+1$$

$$=2^n-1$$

Hanoi塔的递归调用树



递归调用的次数：

$$C(n)=\sum_{l=0}^{n-1}2^i=2^n-1$$