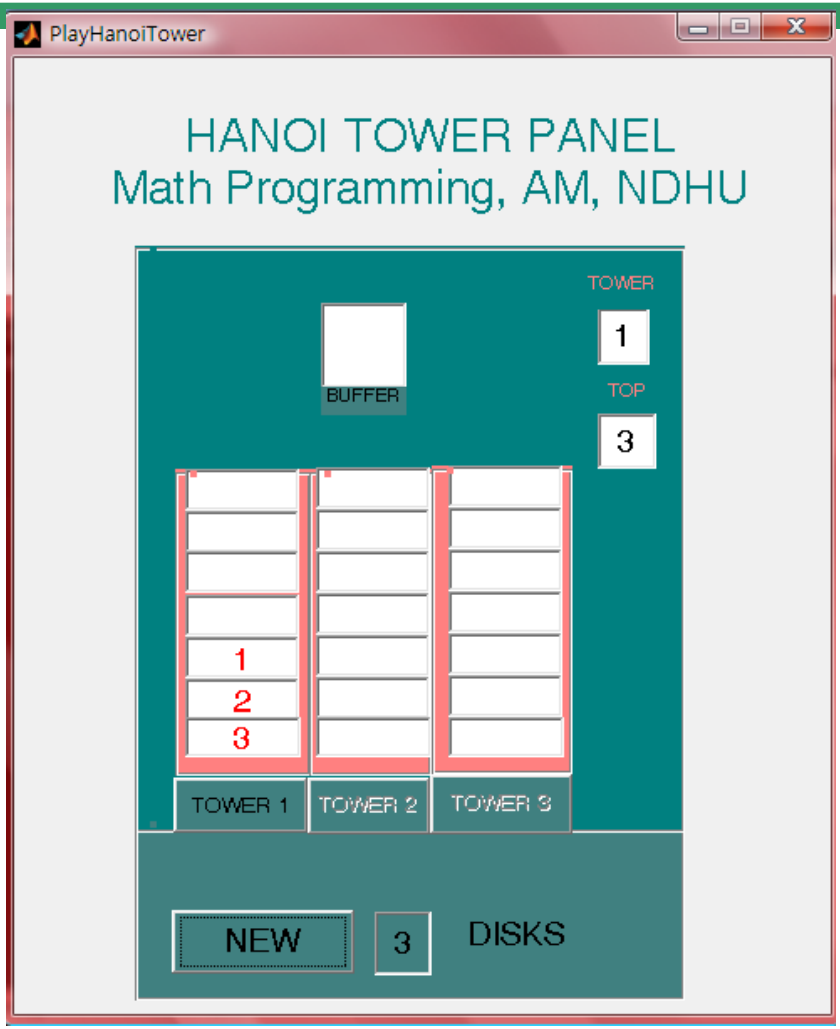
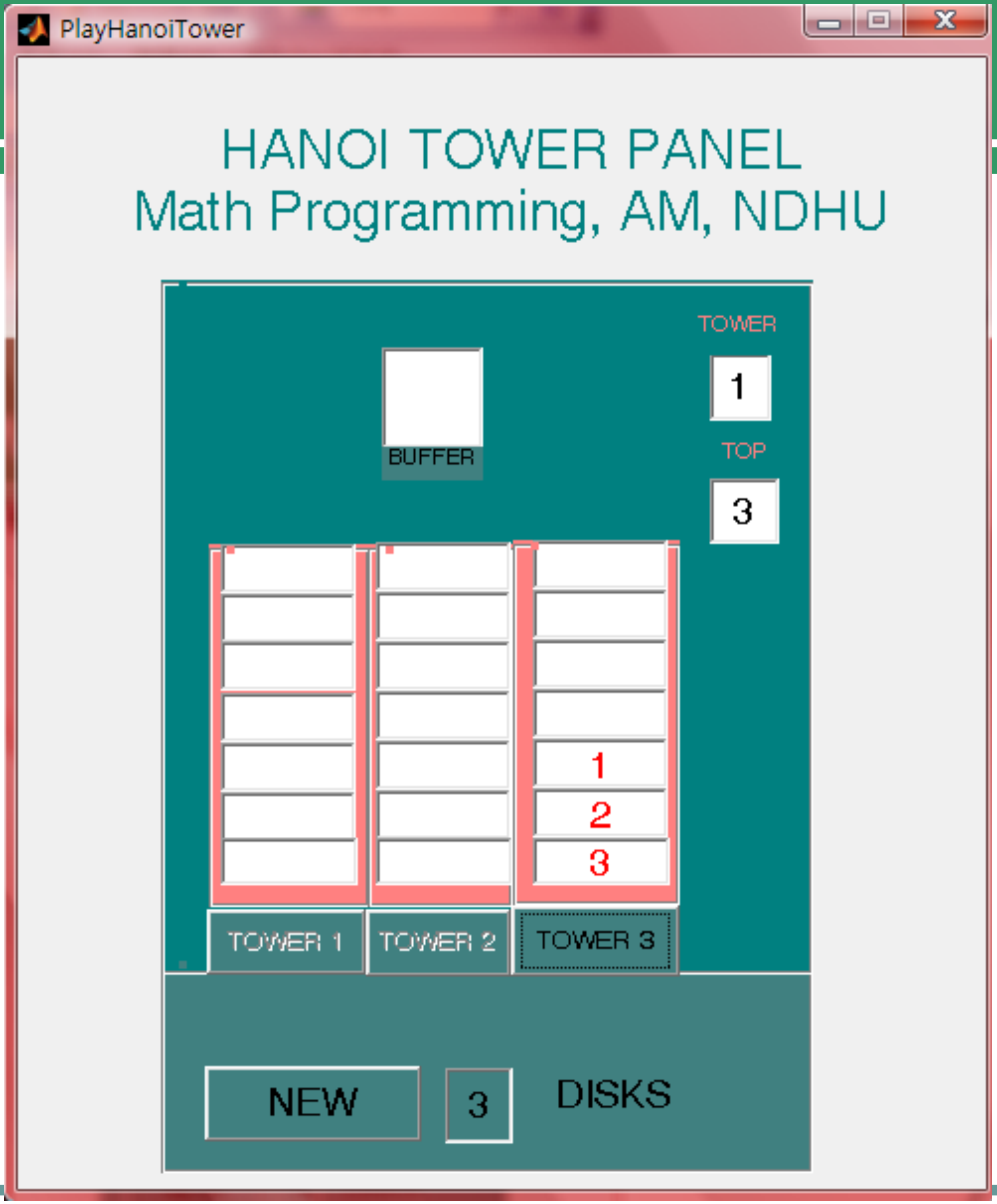
Lecture 8 Recursive programming

- Recursive programming
  - Hanoi tower
  - Laplacian expansions
  - Bareiss's standard fraction free Gaussian elimination
  - Binary to decimal translation
  - Decimal to binary translation

Hanoi Tower Problem



Target

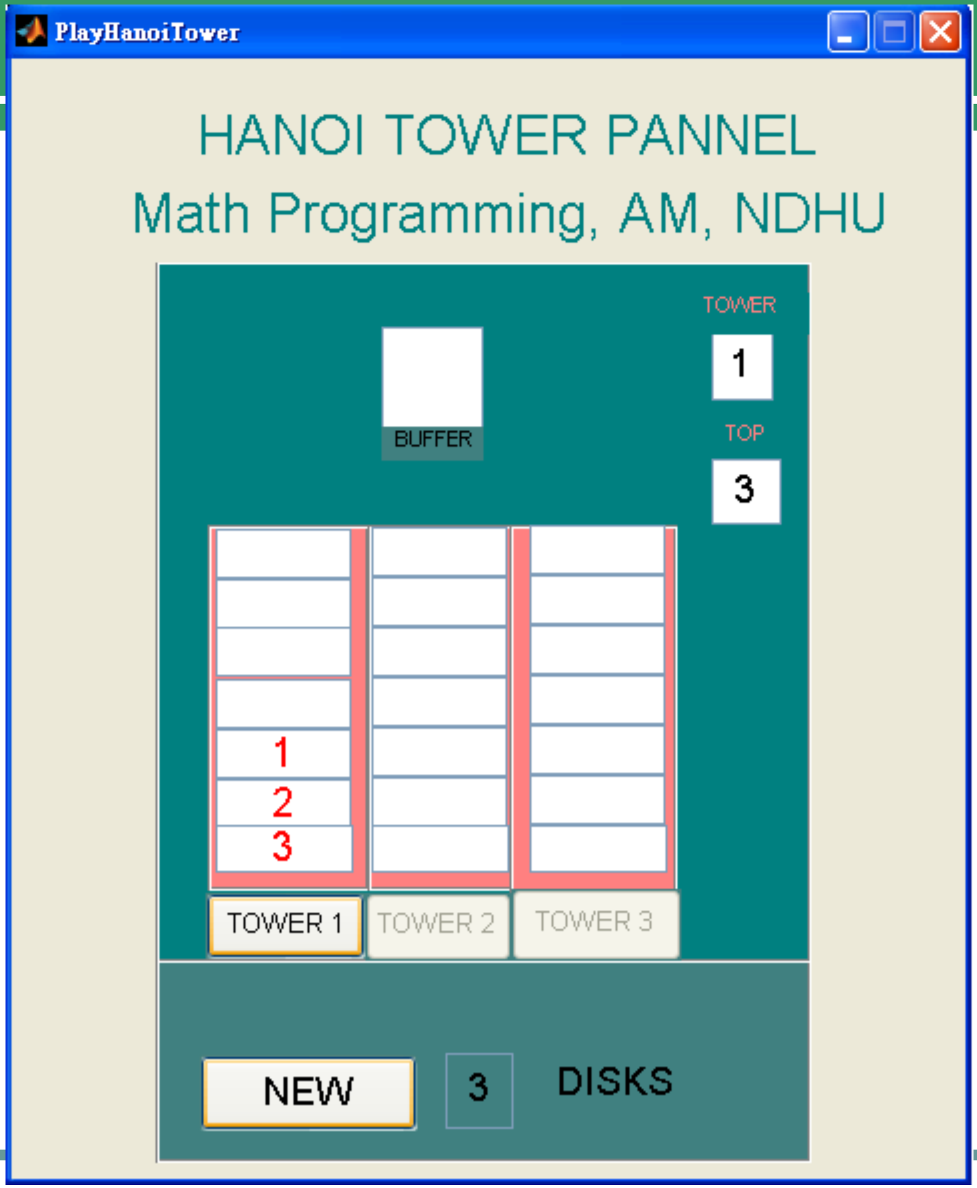

PlayHanoiTower
HANOI TOWER PANEL
Math Programming, AM, NDHU
TOWER
1
TOP
3
BUFFER
1
2
3
TOWER 1
TOWER 2
TOWER 3
NEW
3
DISKS

Hanoi Tower

PlayHanoiTower.fig
PlayHanoiTower.m

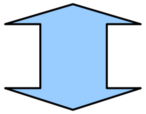
Valid movement

- A larger disk is inhibited to be placed on the top of a smaller disk.
- Initial state: all disks on the first tower
- Final state: all disks on the third tower

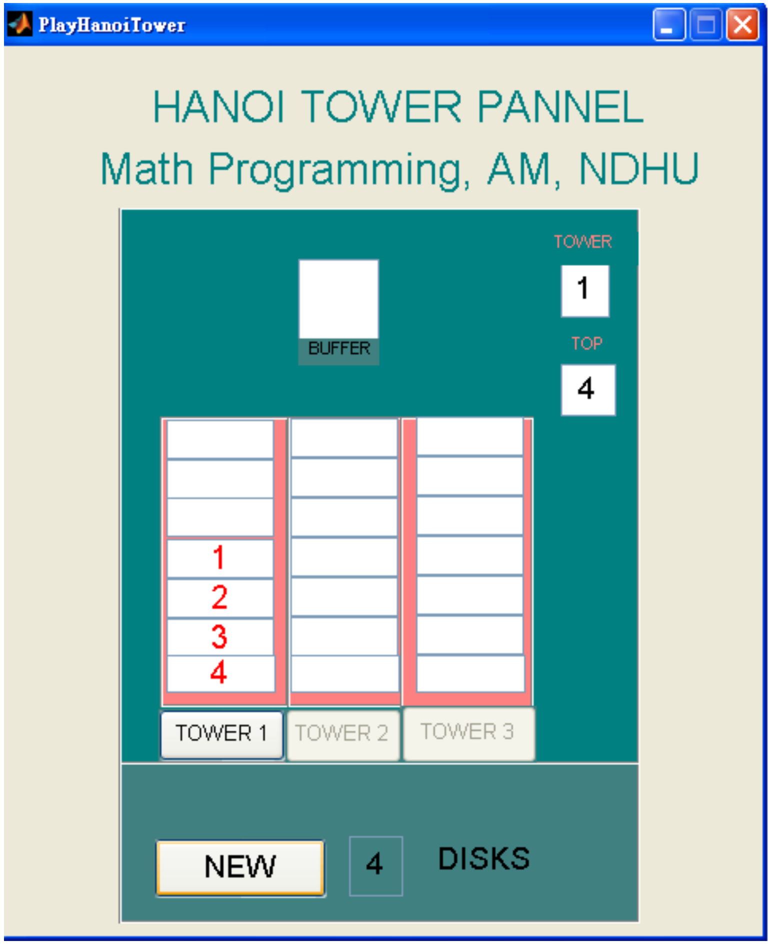

PlayHanoiTower
HANOI TOWER PANNEL
Math Programming, AM, NDHU
TOWER
1
BUFFER
TOP
3
1
2
3
TOWER 1
TOWER 2
TOWER 3
NEW
3
DISKS

Three steps for auto-play

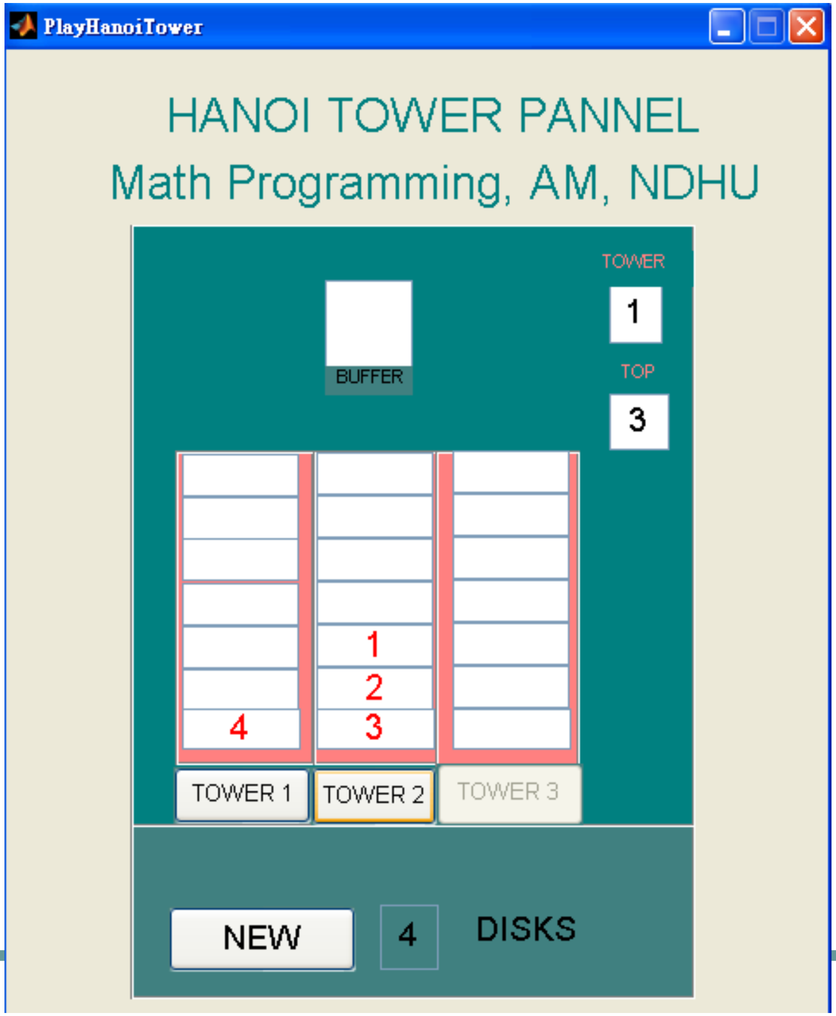
- Move n-1 objects from stack 1 to stack 2
- Move 1 object from stack 1 to stack 3
- Move n-1 objects from stack 2 to stack 3



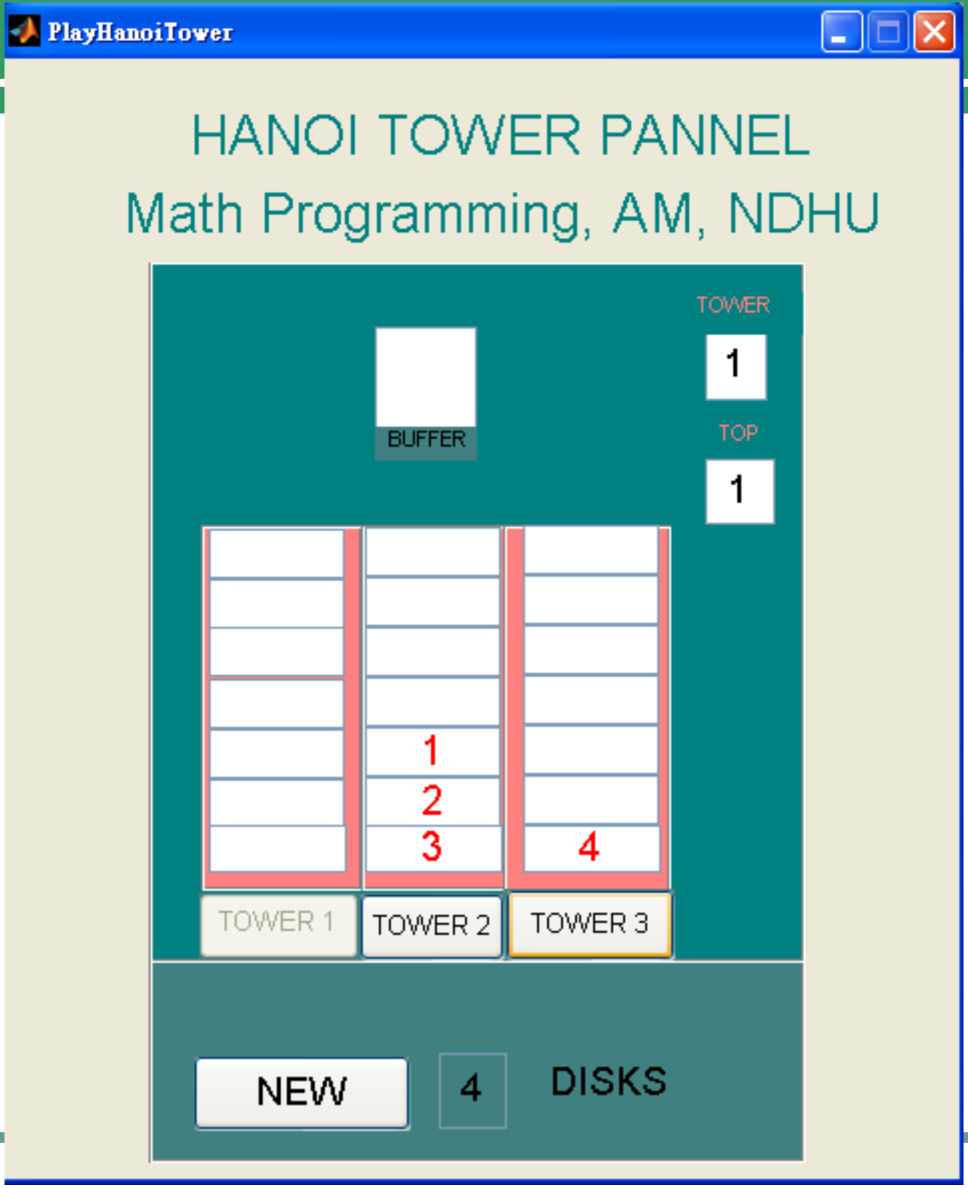
Initial state of 4-disk problem



Move 3 disks from stack 1 to 2

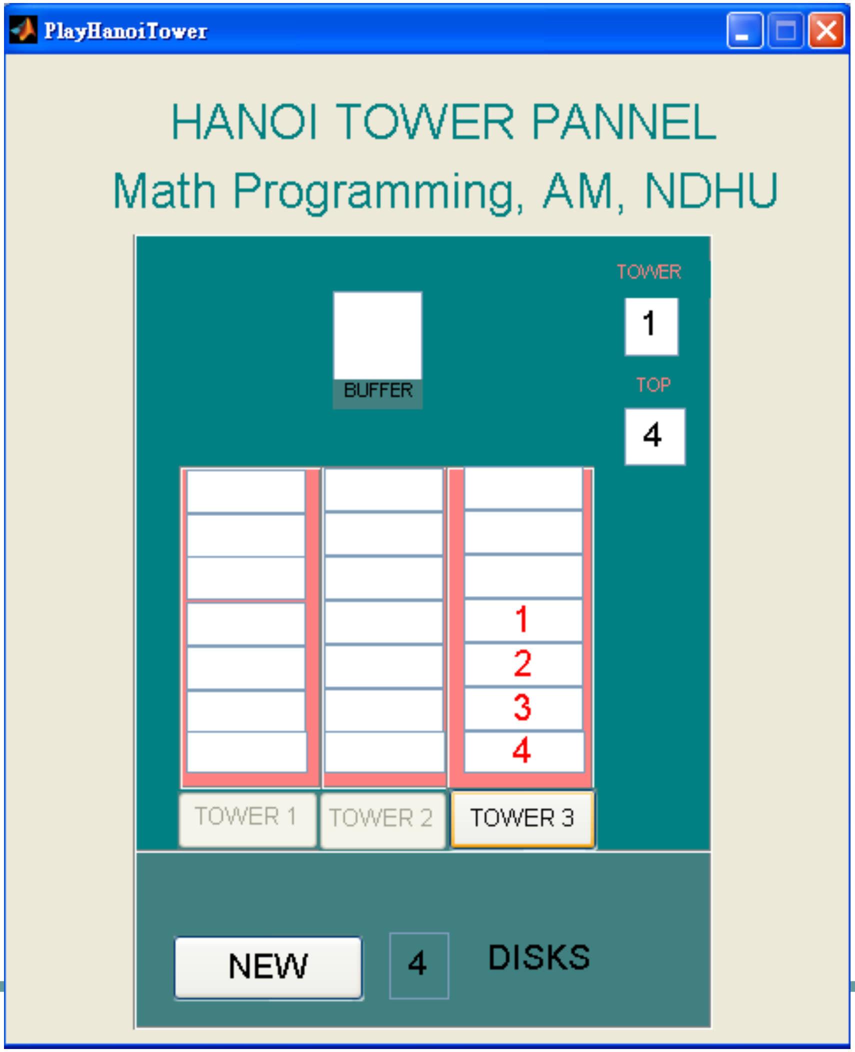


Move disk 4



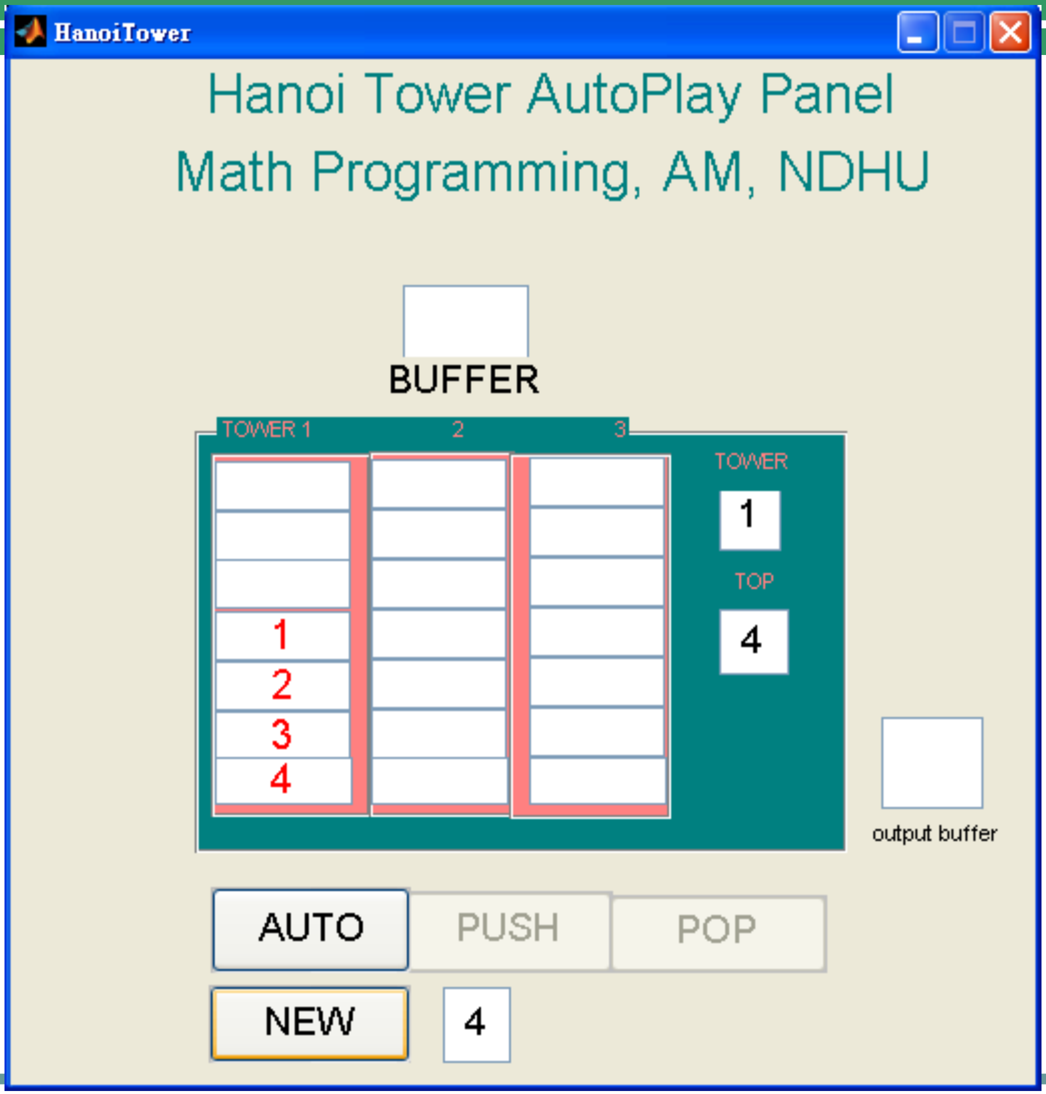
PlayHanoiTower
HANOI TOWER PANNEL
Math Programming, AM, NDHU
TOWER
1
BUFFER
TOP
1
1
2
3
4
TOWER 1
TOWER 2
TOWER 3
NEW
4
DISKS

Move 3 disks from stack 2 to 3

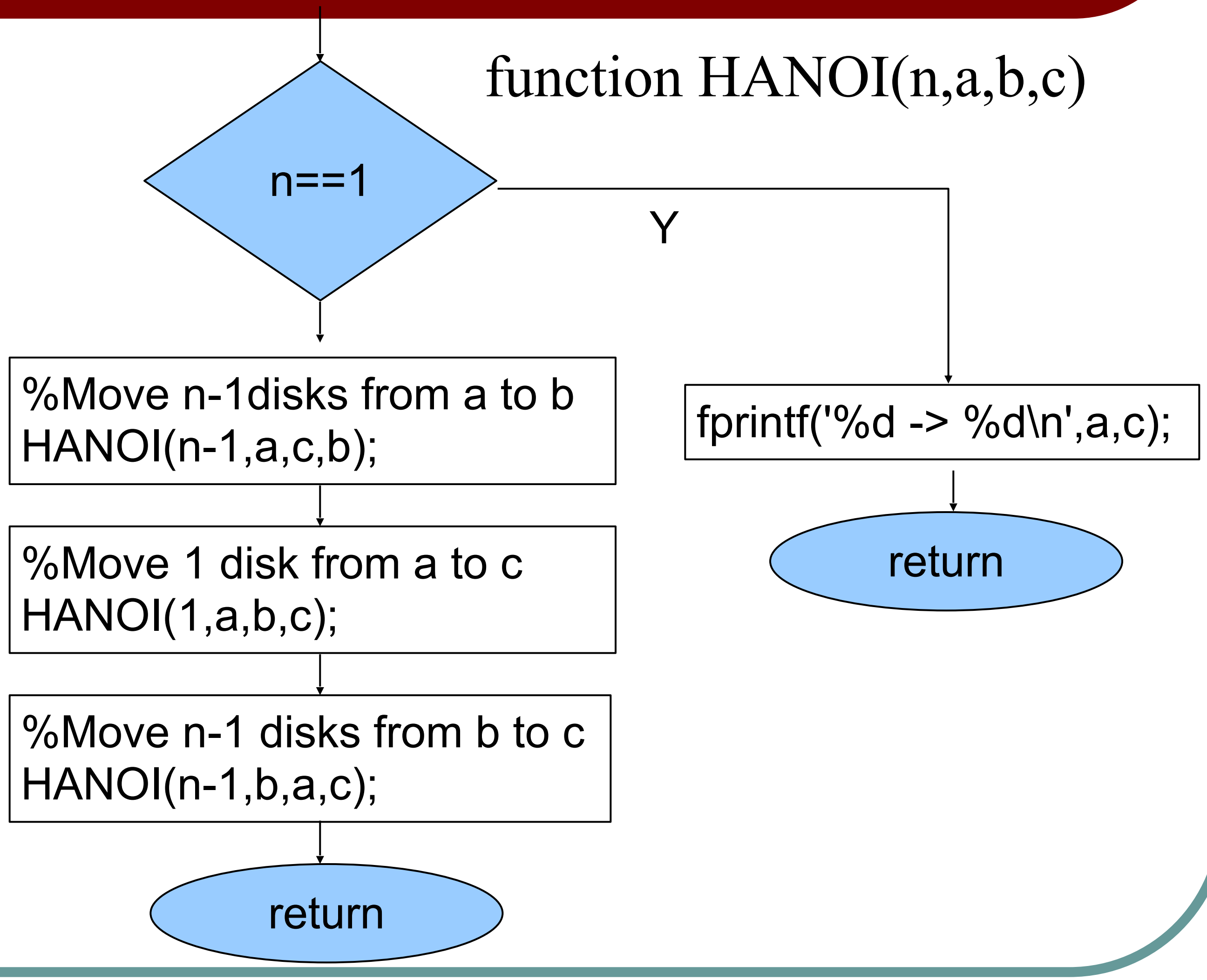


Hanoi-tower auto-play panel

HanoiTower.m
HanoiTower.fig


HanoiTower
Hanoi Tower AutoPlay Panel
Math Programming, AM, NDHU
BUFFER
TOWER 1
2
3
1
2
3
4
TOWER
1
TOP
4
output buffer
AUTO
PUSH
POP
NEW
4

Flow chart: move n disks from tower a to c

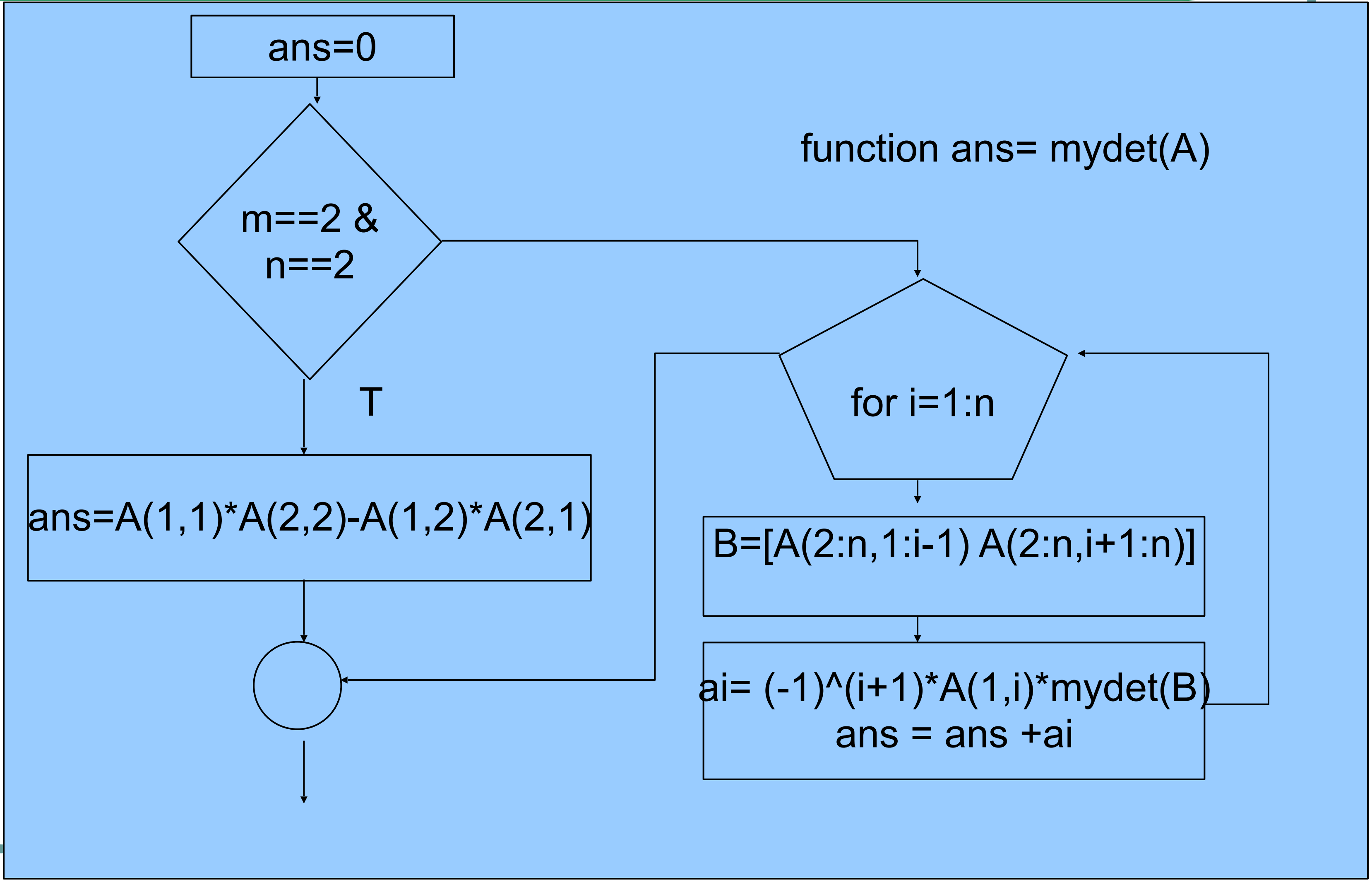


Recurrent relation of Laplacian expansion

- det(A) is decomposed to n sub-tasks
- Each calculates determinant of an (n-1)-by-(n-1) matrix $\widetilde{A}_{1i}$
- The problem size is reduced from n to n-1

$$Det(A) = \sum_{i=1}^{n} (-1)^{i+1} a_{1i} \det(\widetilde{A}_{1i})$$

Recursive programming based on Laplacian expansion



Drawbacks

- Computational complexity, O(n!)
- Time consuming
- Memory consuming
- If n >10, it results in intolerant computing time to evaluate determinant by recursive programming.
- An improvement by Bareiss's standard fraction free Gaussian elimination

Bareiss's standard fraction free Gaussian elimination

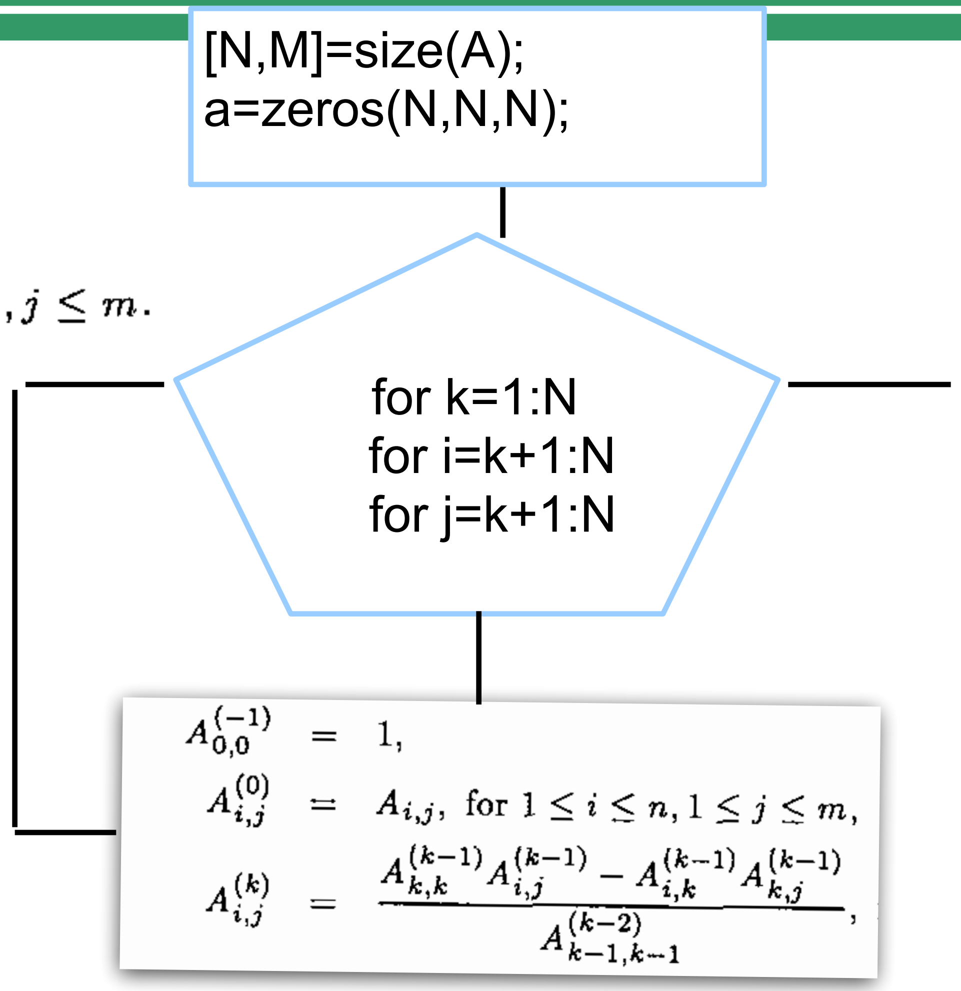
Bareiss' standard fraction free Gaussian elimination (Bareiss, 1968).

$$\begin{aligned} A_{0,0}^{(-1)} &= 1, \\ A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m, \\ A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}, \text{ for } 1 \le k < n, k < i, j \le m. \end{aligned}$$

It is well known that

$$A_{i,j}^{(k)} = \begin{vmatrix} A_{1,1} & \cdots & A_{1,k} & A_{1,j} \\ \vdots & & \vdots & \vdots \\ A_{k,1} & \cdots & A_{k,k} & A_{k,j} \\ A_{i,1} & \cdots & A_{i,k} & A_{i,j} \end{vmatrix}.$$

Thus when $m = n$, $\det(A) = A_{n,n}^{(n-1)}$, and when $A = \begin{pmatrix} M & b \\ I & 0 \end{pmatrix}$, for square for $1 \le k < n, k < i, j \le m$.



```
for k=1:N
for i=k+1:N
for j=k+1:N
```

$$A_{0,0}^{(-1)} = 1,$$

$$A_{i,j}^{(0)} = A_{i,j}, \text{ for } 1 \leq i \leq n, 1 \leq j \leq m,$$

$$A_{i,j}^{(k)} = \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}},$$

```
end
end
end
```

$$A_{0,0}^{(-1)} = 1,$$

$$A_{i,j}^{(0)} = A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m,$$

$$A_{i,j}^{(k)} = \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}},$$

```
if k==1
a(i,j,k)=A(k,k)*A(i,j)-A(i,k)*A(k,j);
end

if k==2
a(i,j,k)=(a(k,k,1)* a(i,j,1)-a(i,k,1)* a(k,j,1))/A(k-1,k-1);
end

if k>2
a(i,j,k)=(a(k,k,k-1)* a(i,j,k-1)-a(i,k,k-1)* a(k,j,k-1))/a(k-1,k-1,k-2);
end
```

```
[N,M]=size(A);
a=zeros(N,N,N);
for k=1:N
for i=k+1:N
for j=k+1:N
    if k==1
    a(i,j,k)=
    end
    if k==2
   a(i,j,k)=(
    end
    if k>2
    a(i,j,k)=
a(k-1,k-1,k
    end
end
end
end
a(N,N,N-1)
```

Bareiss' standard fraction free Gaussian elimination (Bareiss, 1968).

$$A_{0,0}^{(-1)} = 1,$$

$$A_{i,j}^{(0)} = A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m,$$

$$A_{i,j}^{(k)} = \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}, \text{ for } 1 \le k < n, k < i, j \le m.$$

It is well known that

$$A_{i,j}^{(k)} = \begin{vmatrix} A_{1,1} & \cdots & A_{1,k} & A_{1,j} \\ \vdots & & \vdots & \vdots \\ A_{k,1} & \cdots & A_{k,k} & A_{k,j} \\ A_{i,1} & \cdots & A_{i,k} & A_{i,j} \end{vmatrix}.$$

Thus when $m = n$, $\det(A) = A_{n,n}^{(n-1)}$, and when $A = \begin{pmatrix} M & b \\ I & 0 \end{pmatrix}$, for square

Recursive Programming

- Decimal to Binary representation
- Binary to decimal representation

Binary to decimal representations

- Problem statement

Let b be a vector of binary bits

$$b = [b_n, b_{n-1}, \ldots, b_1],$$

$$\text{where } b_n > 0$$

Translate it to a decimal number such that

$$a = b_n 2^{n-1} + b_{n-1} 2^{n-2} + \ldots + b_1 2^0$$

Decomposition

$$a = b_n 2^{n-1} + b_{n-1} 2^{n-2} + \ldots + b_1 2^0$$

$$= 2(b_n 2^{n-2} + b_{n-1} 2^{n-3} + \ldots + b_2 2^0) + b_1$$

$$b = [b_n, b_{n-1}, \ldots, b_1],$$

where $b_n > 0$

$$a = bin2dec(b)$$

$b1 = b(n)$

$b2 = \mathrm{b(1:n-1);}$

a can be calculated by

2 * bin2dec(b2) + b1

Halting condition

$$a = b_1$$

$$a = bin2dec(b)$$

$$if \;\; \mathrm{n} == 1$$

$$a = b$$

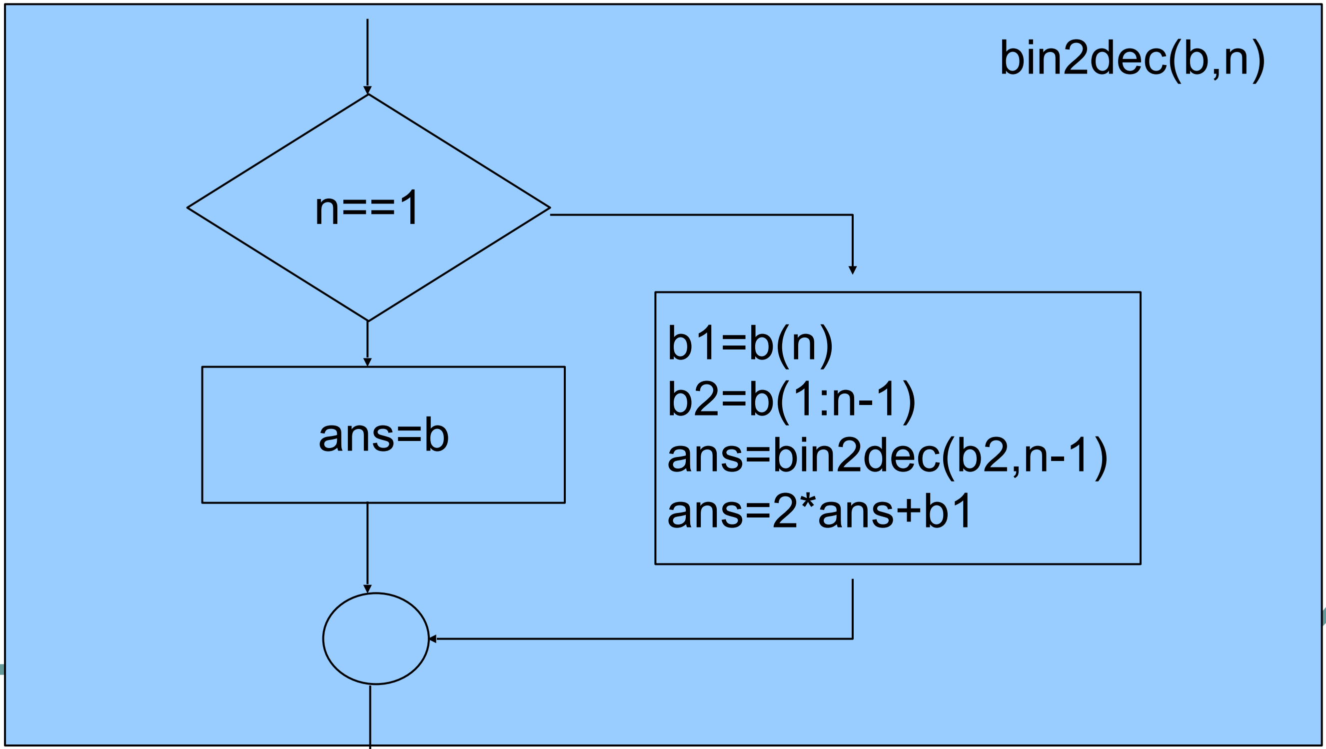
Recurrence relation

$$f(b;n) = 2 * f(b2;n-1) + b1 \text{ if n} > 1$$

$$\mathrm{b1 = b(n)}$$

$$\mathrm{b2 = b(1:n-1)}$$

$$f(b;1) = b$$



```
function ans=bin2dec(b,n)
    if n==1
        ans=b;
        return
    else
        b1=b(n);
        b2=b(1:n-1);
        ans=bin2dec(b2,n-1);
        ans=2*ans+b1;
    end
```

```
function ans=bin2dec(b)
    n = length(b);
    if n==1
        ans=b;
        return
    else
        b1=b(n);
        b2=b(1:n-1);
        ans=bin2dec(b2);
        ans=2*ans+b1;
    end
```

Decimal to binary representations

- Problem statement b = dec2bin(a)

Let a denote a decimal number
b denotes the binary representation of a

$$a = b_n 2^{n-1} + b_{n-1} 2^{n-2} + \ldots + b_1 2^0$$

$$b = [b_n, b_{n-1}, \ldots, b_1],$$

where $b_n > 0$

Recurrence relation

$$a1 = \mathrm{mod}(a,2)$$

$$a2 = \mathrm{floor}(a/2)$$

Use binary representation of *a2* to represent *a*

$$ans = [dec2bin(a2)\ \ \mathrm{a1}]\ \mathrm{if\ a} > 1$$

$$ans = a\ \mathrm{if}\ a \le 1$$

```
>> b1=[1 0 1]

b1 =

     1     0     1

>> b2=0

b2 =

     0

>> [b1 b2]

ans =

     1     0     1     0
```

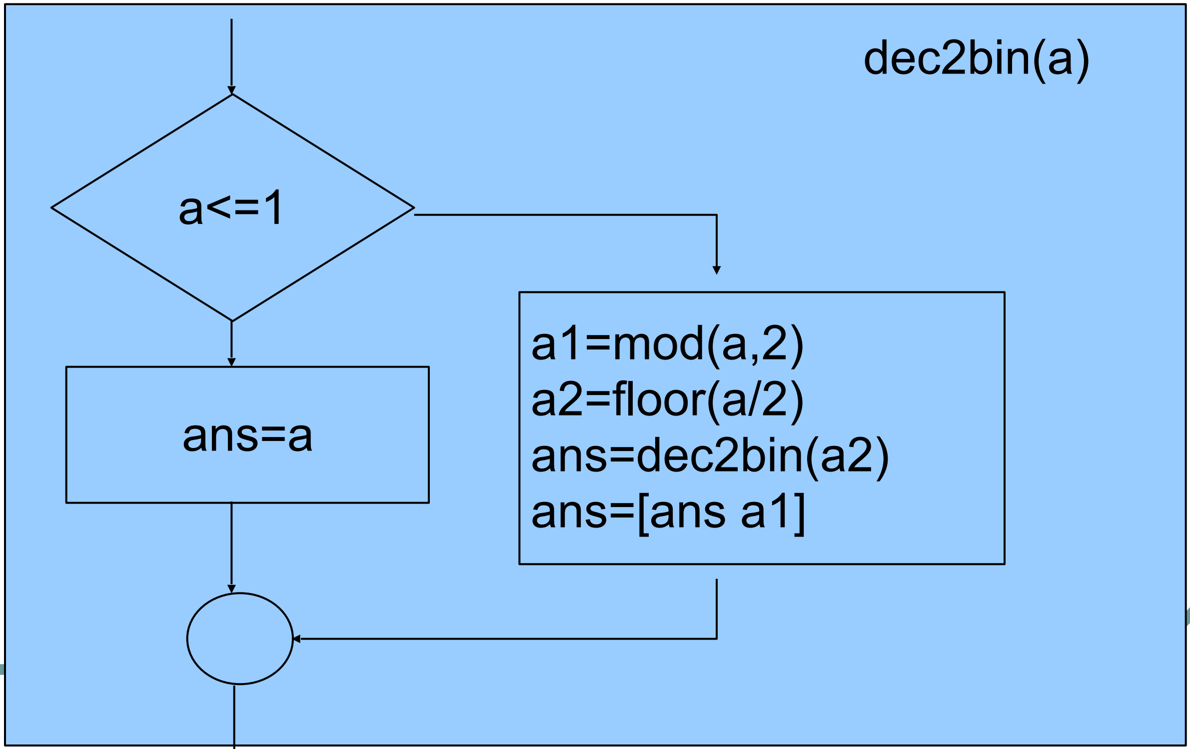
Recurrence relation

$a1 = \mathrm{mod}(a,2)$

$a2 = \mathrm{floor}(a/2)$

$ans = [dec2bin(a2)\ \ \mathrm{a1}]\ \mathrm{if\ a > 1}$

$ans = a \text{ if } a \le 1$



```
function ans=dec2bin(a)
    if a<=1
        ans=a;
        return
    else
        a1=mod(a,2);
        a2=floor(a/2);
        ans=dec2bin(a2);
        ans=[ans a1];

    end
```

```
>> a=53;
>> b=dec2bin(a)

b =

110101
```

```
>> bin2dec(b,6)

ans =

   53
```