Lecture 4
Recursive Functions

Recursive Function

- A function that calls itself.
- Compiler internally uses stack to implement (or execute) any recursive function.
- Recursion occurs when a function is called by itself repeatedly.

Example:

```
function1()
{
  …
  function1();
  …
}
```

Contd…

- Recursion is an elegant programming technique.
- Not the best way to solve a problem, due to the following reasons:
  - Requires stack implementation.
  - Utilizes memory inefficiently, as every recursive call allocates a new set of local variables to a function.
  - Slows down execution speed, as function calls require jumps, and saving the current state of program onto stack before jump.
- Although an inefficient way, but
  - Too handy to solve several problems.
  - Easier to implement.

Example: Factorial

```
#include<iostream>
using namespace std;
int fact(int n)
{  if (n == 1)
       return n;
   else
      return (n * fact(n-1));
}
int main()
{  int n;
   cout << "Enter a number: ";
   cin >> n;
   cout << "Factorial of " << n << " is " << fact(n);
   return 0;            }
```

For n = 5.

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

fact(5)

5 * fact(4)

4 * fact(3)

3 * fact(2)

2 * fact(1)

return 1

return 2

return 6

return 24

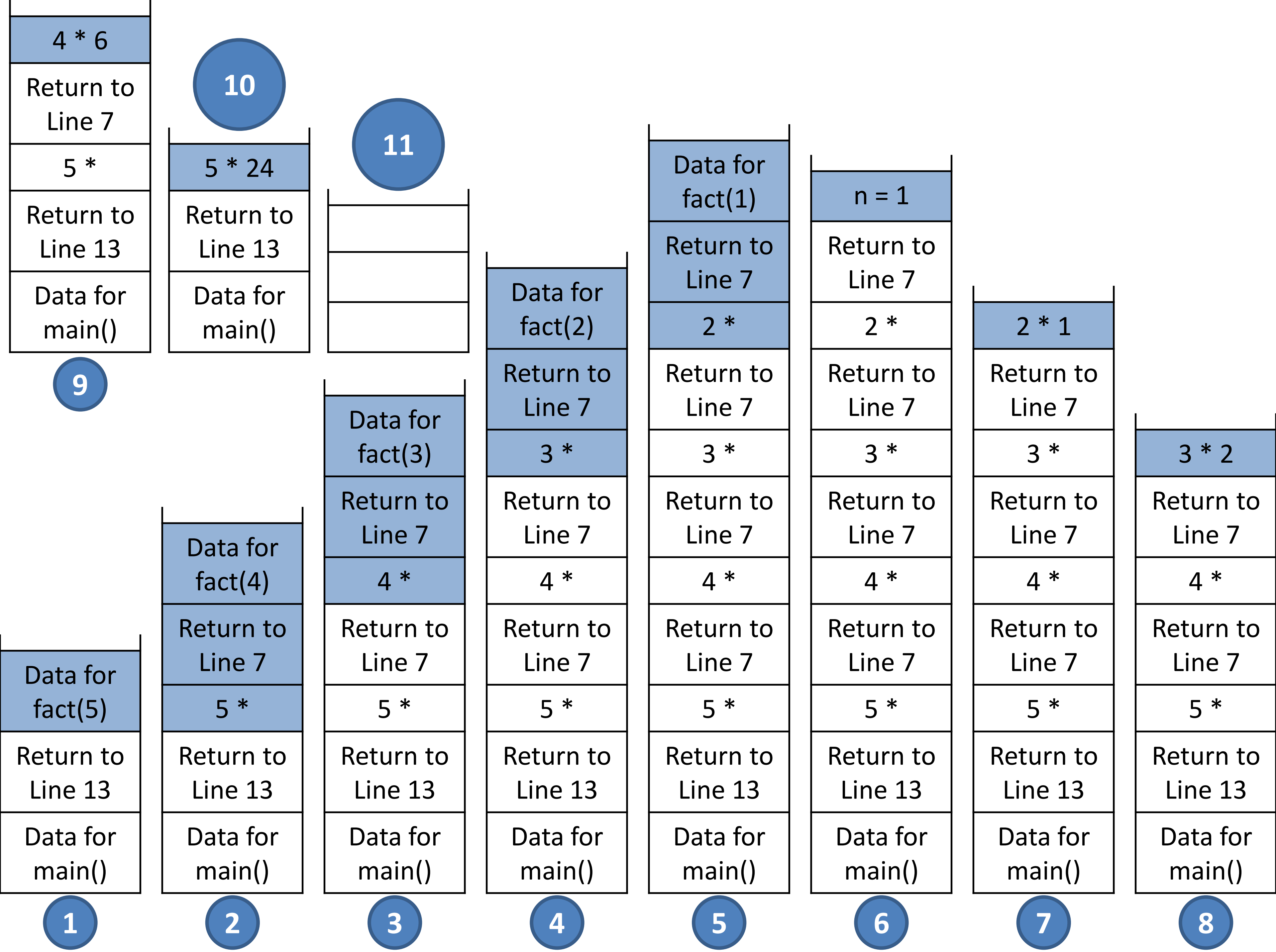
return 120

Recurrence Relation:

$T(n)=1$ for $n=0$

$T(n)=1$ for $n=1$

$T(n)=T(n-1) + 1$ for $n>1$

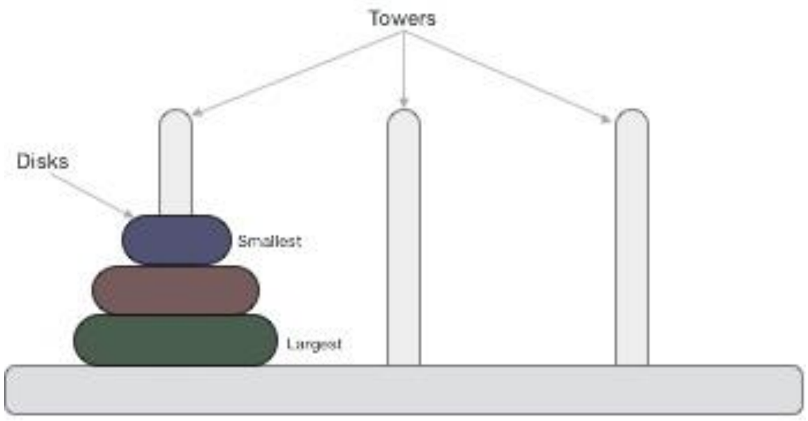

4 * 6
Return to Line 7
5 *
Return to Line 13
Data for main()
9
10
5 * 24
Return to Line 13
Data for main()
11
Data for fact(5)
Return to Line 13
Data for main()
1
Data for fact(4)
Return to Line 7
5 *
Return to Line 13
Data for main()
2
Data for fact(3)
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
3
Data for fact(2)
Return to Line 7
3 *
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
4
Data for fact(1)
Return to Line 7
2 *
Return to Line 7
3 *
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
5
n = 1
Return to Line 7
2 *
Return to Line 7
3 *
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
6
2 * 1
Return to Line 7
3 *
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
7
3 * 2
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
8

Towers of Hanoi

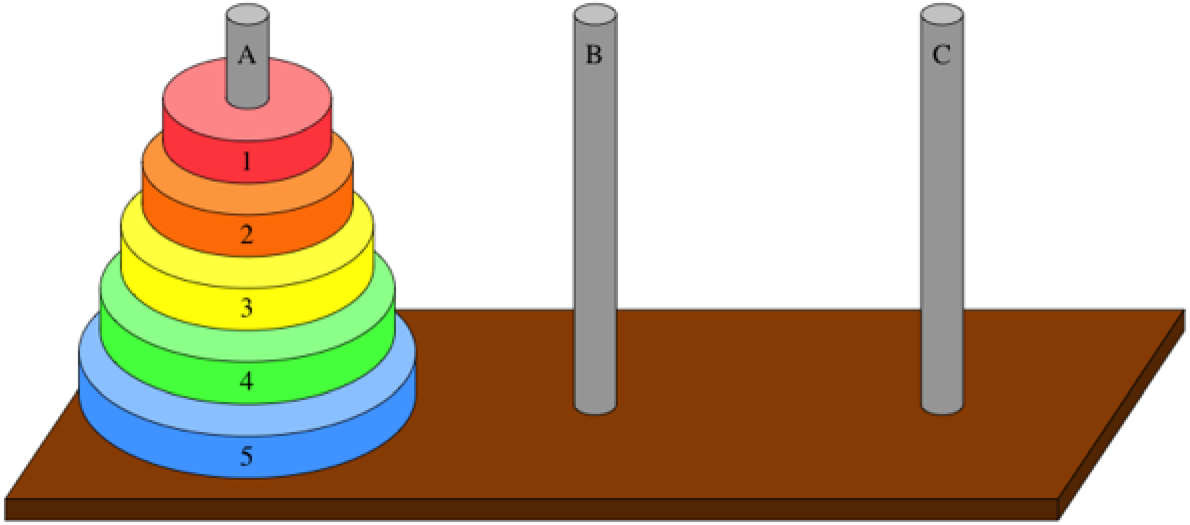
- Given:
  - A set of three pegs and
  - $\boldsymbol{n}$ disks, with each disk a different size.
- Let:
  - The pegs are named as $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$, and
  - Disks are named as $\mathbf{1}$ (the smallest disk), $\mathbf{2}$, $\mathbf{3}$..., $\boldsymbol{n}$ (the largest disk).
- Initially, all $\boldsymbol{n}$ disks are on peg $\boldsymbol{A}$, in order of decreasing size from bottom to top, so that disk $\boldsymbol{n}$ is on the bottom and disk $\mathbf{1}$ is on the top.

Contd...

- For $n = 3$ disks.



- For $n = 5$ disks.

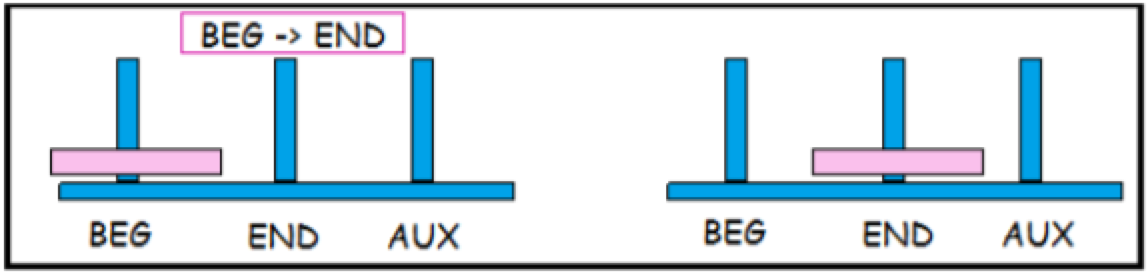


Contd…

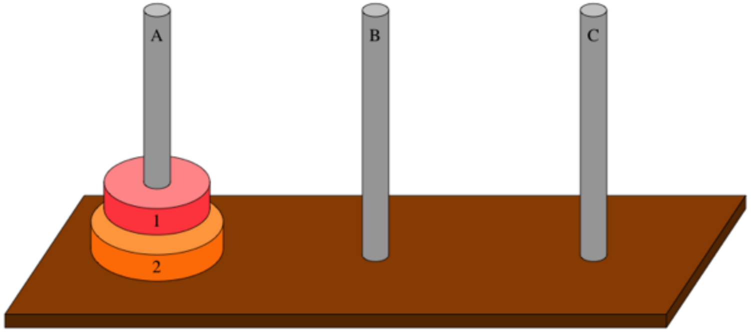
- The goal is to move all the disks to some another tower without violating the sequence of arrangement.
- A few rules to be followed are:
  - Only one disk can be moved among the towers at any given time.
  - Only the "top" disk can be removed.
  - No large disk can sit over a small disk.

Recursive Solution

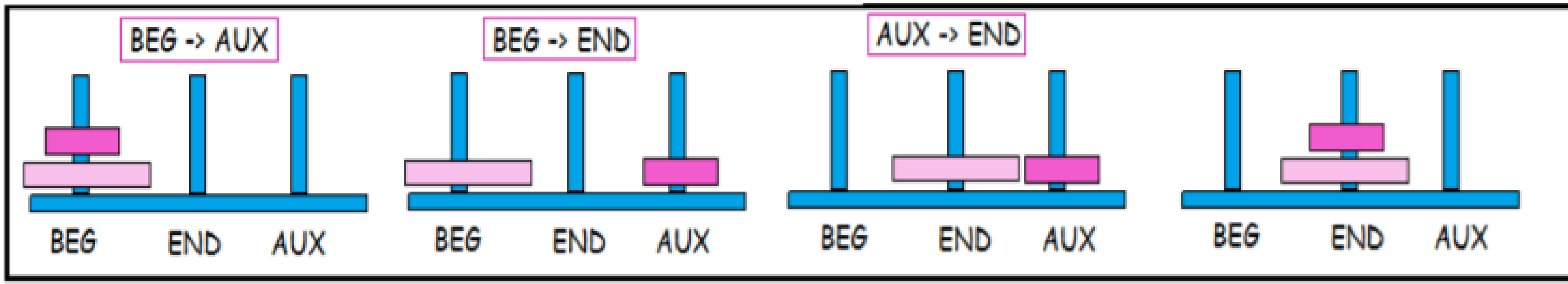
- Lets start with an easy case: one disk, that is, n = 1.
  - This is the base case, as disk 1 can be moved from any peg to any peg.



- What about n = 2?

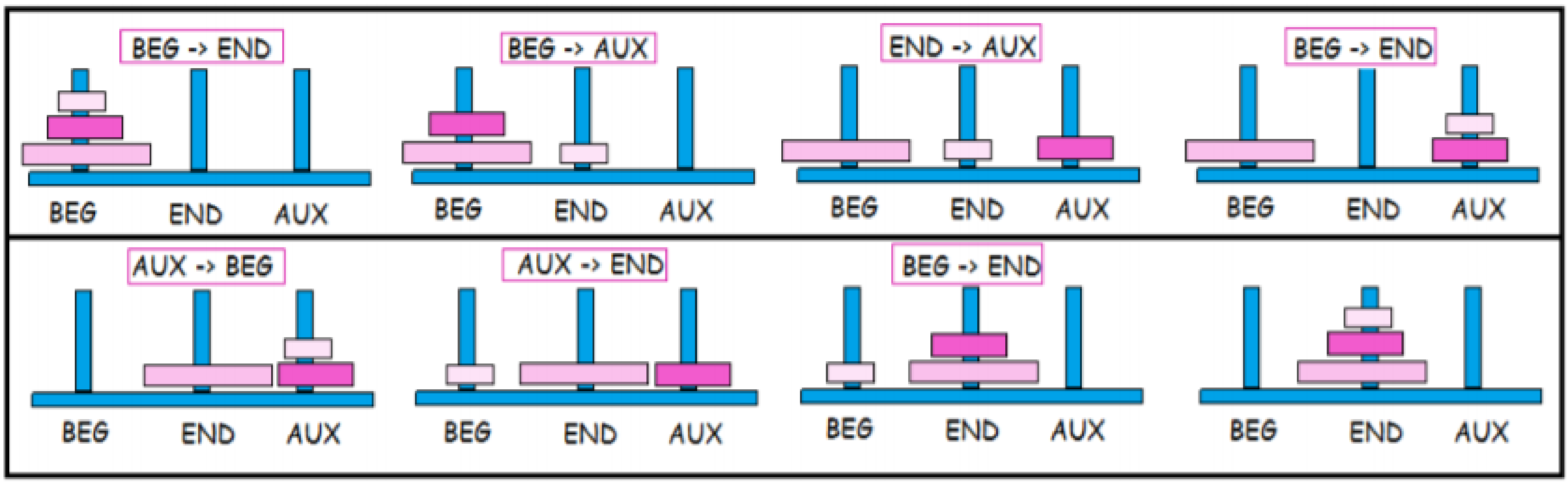


n = 2



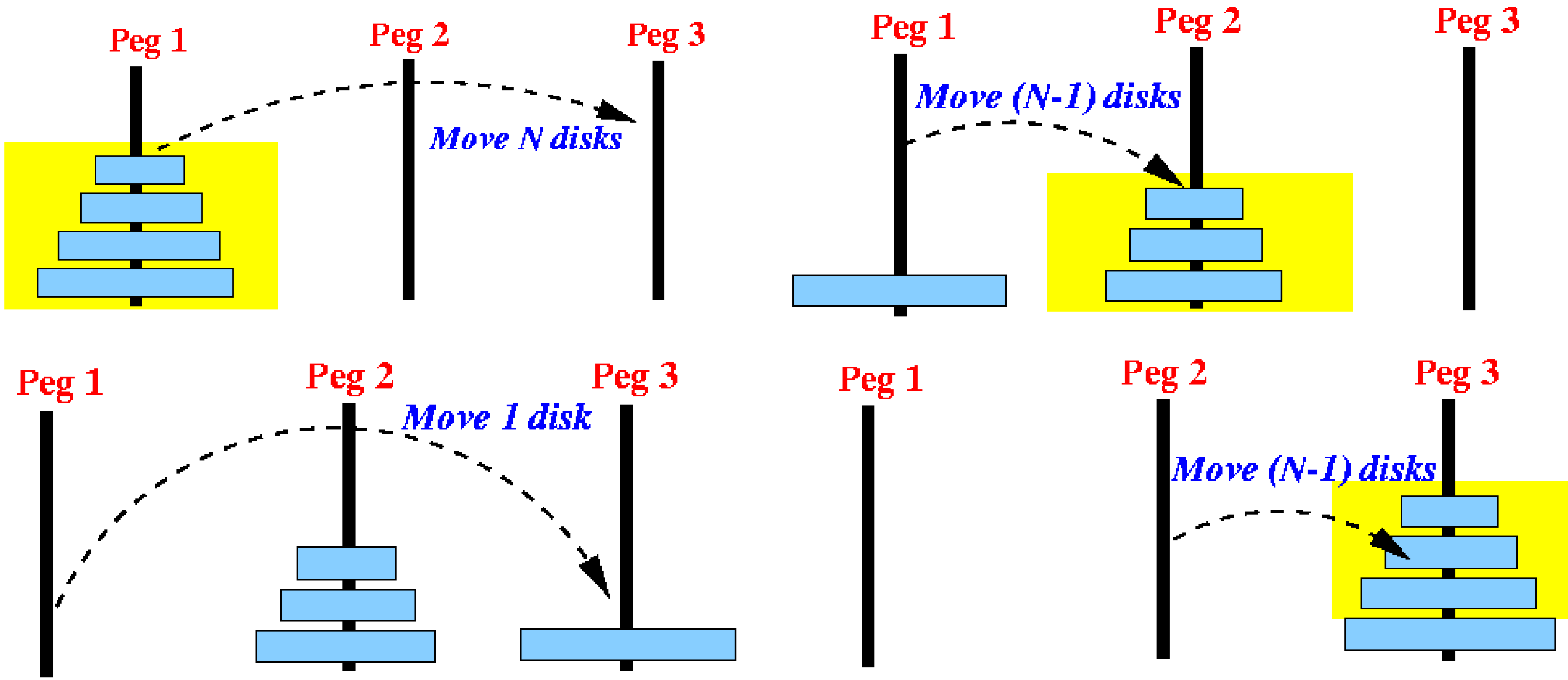
BEG -> AUX
BEG END AUX
BEG -> END
BEG END AUX
AUX -> END
BEG END AUX
BEG END AUX

n = 3

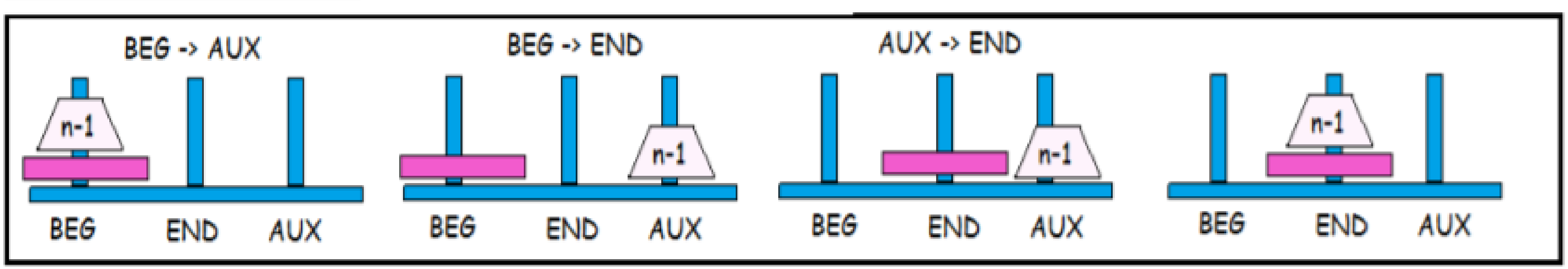


BEG -> END
BEG END AUX
BEG -> AUX
BEG END AUX
END -> AUX
BEG END AUX
BEG -> END
BEG END AUX
AUX -> BEG
BEG END AUX
AUX -> END
BEG END AUX
BEG -> END
BEG END AUX
BEG END AUX

n = 4



For n Disks



Contd…

- To move n disks from the source pole, to the destination pole, using an auxiliary pole:
  1. If n == 1, move the disk to the destination pole and stop.
  2. Move the top n – 1 disks to an auxiliary pole, using the destination pole.
  3. Move the remaining disk to the destination pole.
  4. Move the n – 1 disks from the auxiliary pole to the destination pole using the source pole.

Algorithm

- towerOfHanoi(n, source, dest, aux)

1. If n == 1
2. Print [Move disk 1 from source to dest]
3. Else
4. towerOfHanoi(n-1, source, aux, dest)
5. Print [Move disk n from source to dest]
6. towerOfHanoi(n-1, aux, dest, source)

Recurrence Relation:

$T(n)=1$ for $n=1$

$T(n)= 2\,T(n-1) + 1$ for $n > 1$

Implementation

```
#include <iostream>
using namespace std;
void towers(int num, char frompeg, char topeg, char auxpeg)
{   if (num == 1)
    {   printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
         return;    }
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);    }
 int main()
{   int num;
     printf("Enter the number of disks : ");
     scanf("%d", &num);
     printf("The sequence of moves involved in the Tower of Hanoi are :\n");
     towers(num, 'A', 'C', 'B');
     return 0; }
```

Output

```
Enter the number of disks : 2
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg B
 Move disk 2 from peg A to peg C
 Move disk 1 from peg B to peg C
```

Contd…

```
Enter the number of disks : 4
The sequence of moves involved in the Tower of Hanoi are :

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

Thank You