Recursive Function

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.

Disadvantages of Recursion

- Consumes more storage space because the recursive calls along with automatic variables are stored on the stack.
- The computer may run out of memory if the recursive calls are not checked.
- Not more efficient in terms of speed and execution time.
- According to some computer professionals, recursion does not offer any concrete advantage over non-recursive procedures/functions.
- If proper precautions are not taken, recursion may result in non-terminating iterations.
- Recursion is not advocated when the problem can be solved through iteration.
- Recursion may be treated as a software tool to be applied carefully and selectively.

Example

```
#include<iostream>
using namespace std;
int gcd(int dividend, int divisor)
{  if(divisor)
          return gcd(divisor, dividend % divisor);
   else
          return dividend;     }
int main()
{  int a, b;
   cout << "Enter two numbers: ";                    cin >> a >> b;
   cout << "GCD of " << a << " and " << b << " is ";
   if (a > b)     cout << gcd(a,b);
   else    cout << gcd(b,a);
   return 0;}
```

```
int gcd(int dividend, int divisor)
{  if(divisor)
      return gcd(divisor, dividend % divisor);
  else
      return dividend;    }
```

For a = 62 and b = 8.

gcd(62,8) **dividend = 62, divisor = 8**

gcd(8,6) **dividend = 8, divisor = 6**

gcd(6,2) **dividend = 6, divisor = 2**

gcd(2,0) **dividend = 2, divisor = 0**

return 2

Data for gcd(62,8)
Return to Line 12
Data for main()

Data for gcd(8,6)
Return to Line 12
Data for main()

Data for gcd(6,2)
Return to Line 12
Data for main()

Data for gcd(2,0)
Return to Line 12
Data for main()

dividend = 2
Return to Line 12
Data for main()

Example

```
#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)

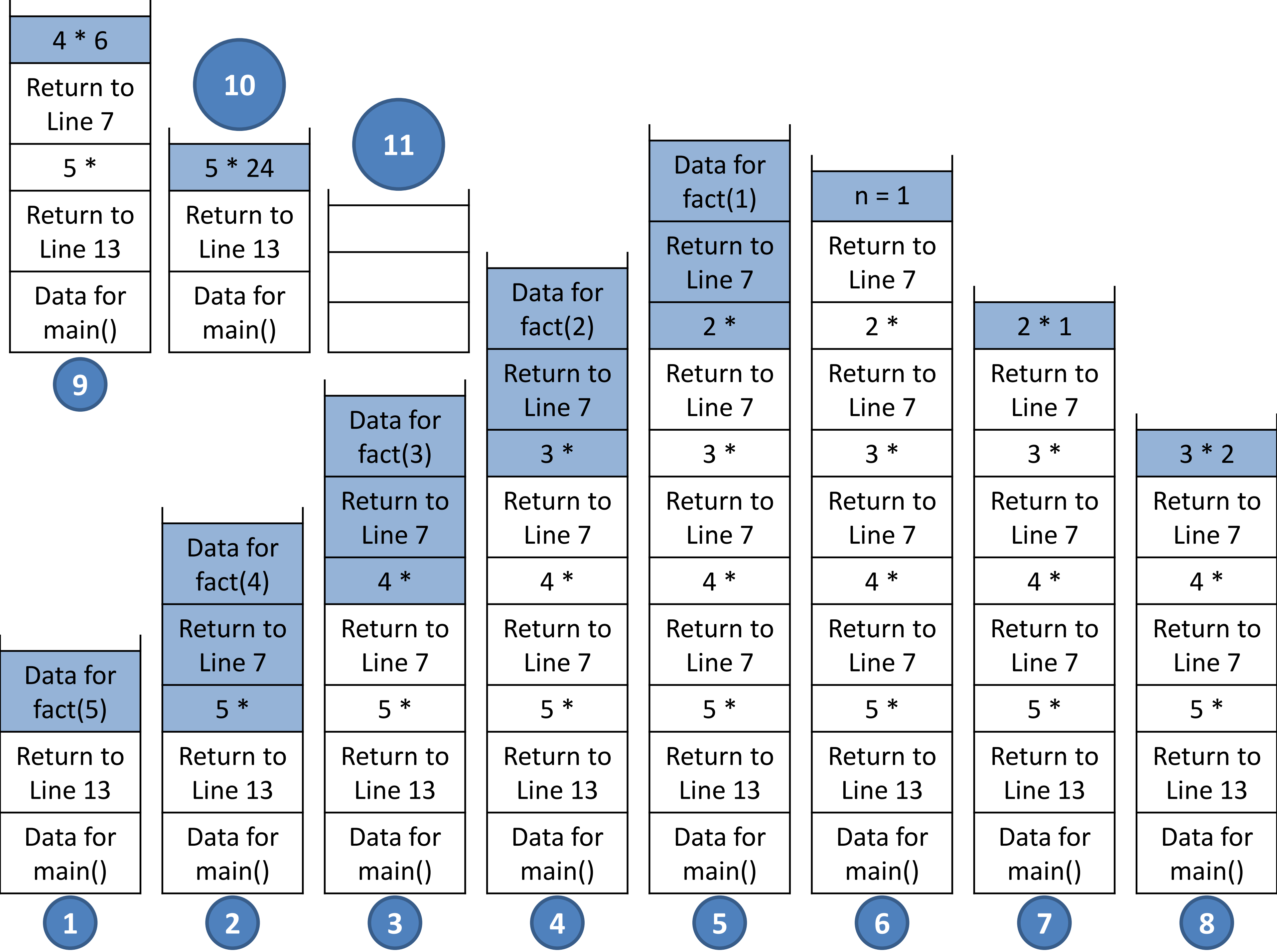
return 1

return 2

return 6

return 24

return 120



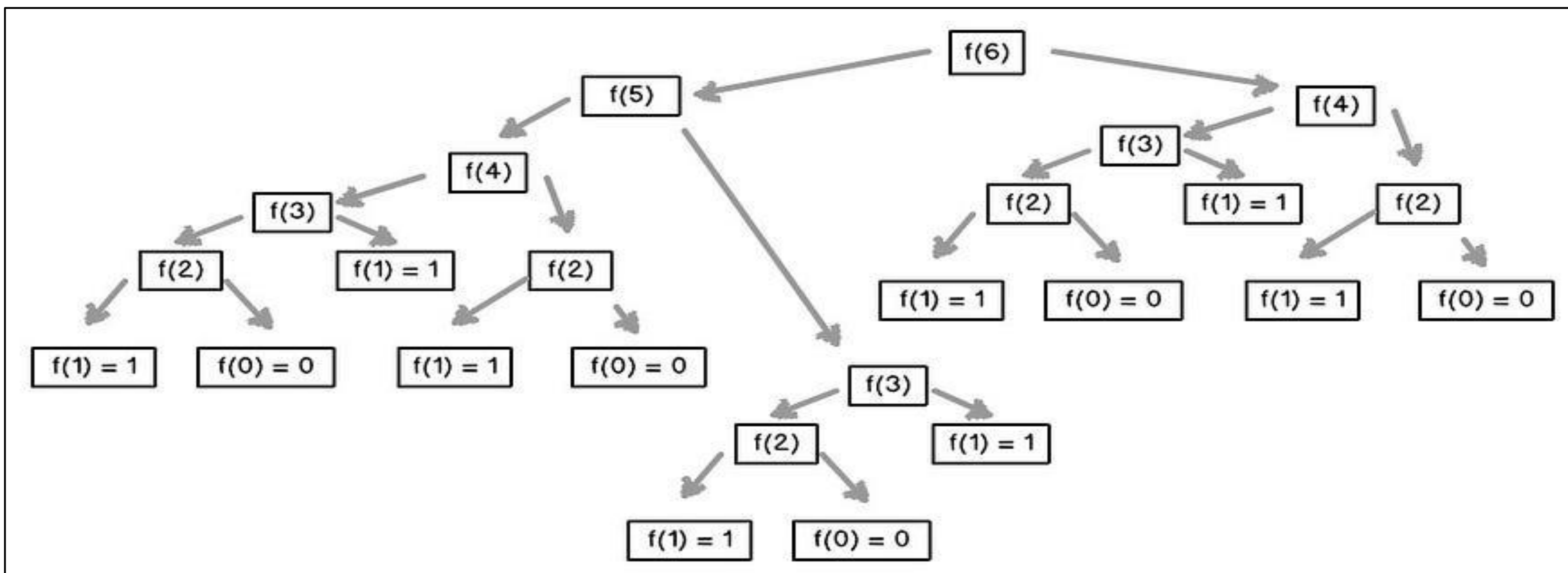
Data for fact(5)
Return to Line 13
Data for main()
1
Data for fact(4)
Return to Line 7
5 *
Return to Line 13
Data for main()
2
Data for fact(3)
Return to Line 7
4 *
Return to Line 7
5 *
Return to Line 13
Data for main()
3
Data for fact(2)
Return to Line 7
3 *
Return to Line 7
4 *
Return to Line 7
5 *
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
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

Fibonacci Series

Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233 and so on.

Pseudocode:

```
fibonacci(number)
if number < 2
        return number
else
        return fibonacci(number - 1) + fibonacci(number - 2)
```



Recursion vs Iteration

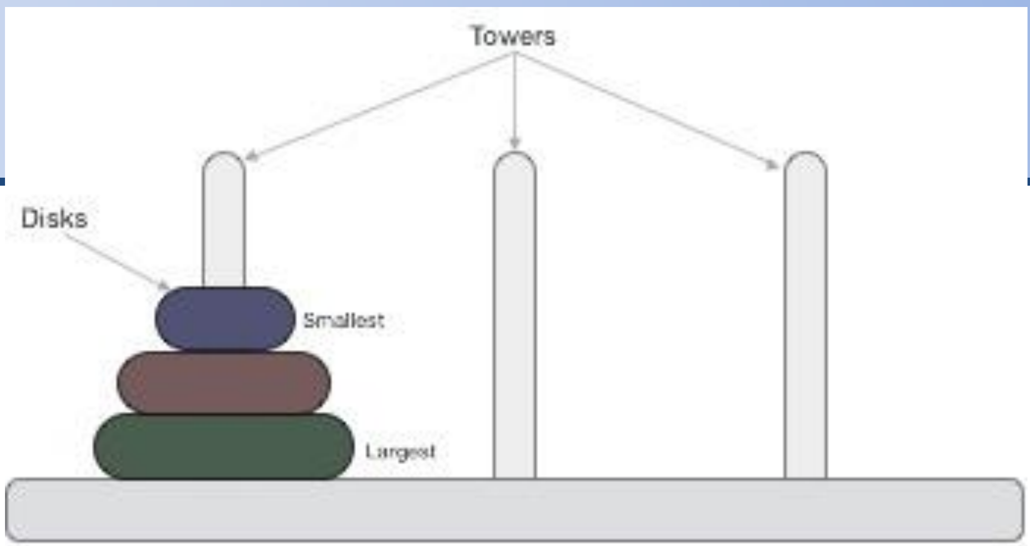
Iteration	Recursion
Process of executing a statement or a set of statements repeatedly, until some specified condition is specified.	Technique of defining anything in terms of itself.
Involves four clear-cut steps like initialization, condition, execution, and updating.	There must be an exclusive if statement inside the recursive function, specifying stopping condition.
Any recursive problem can be solved iteratively.	Not all problems have recursive solution.
Iterative counterpart of a problem is more efficient in terms of memory utilization and execution speed.	Recursion is generally a worse option to go for simple problems, or problems not recursive in nature

Towers of Hanoi

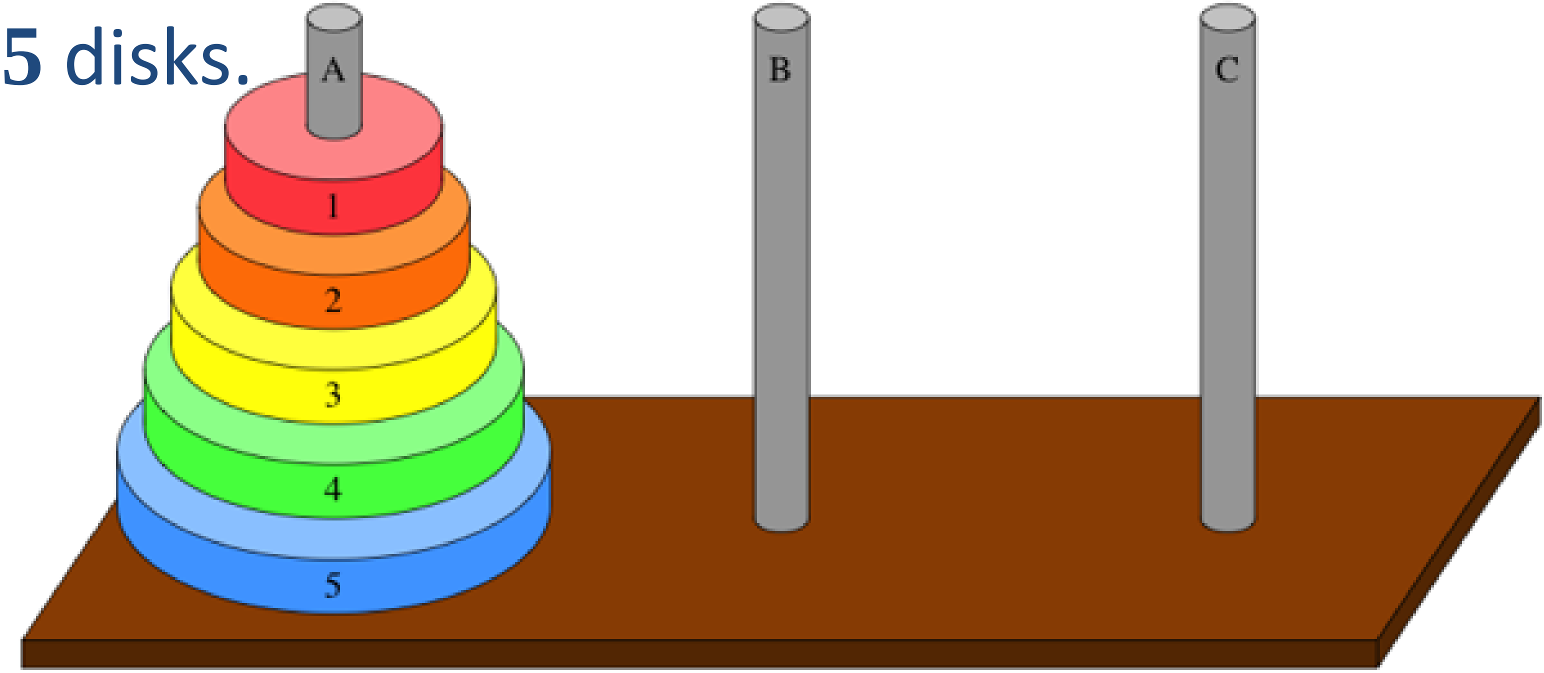
- Given:
  - A set of three pegs and
  - n disks, with each disk a different size.
- Let:
  - The pegs are named as A, B, and C, and
  - Disks are named as 1 (the smallest disk), 2, 3..., n (the largest disk).
- Initially, all n disks are on peg A, in order of decreasing size from bottom to top, so that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

Contd...

- For $n = 3$ disks.



- For $n = 5$ disks.

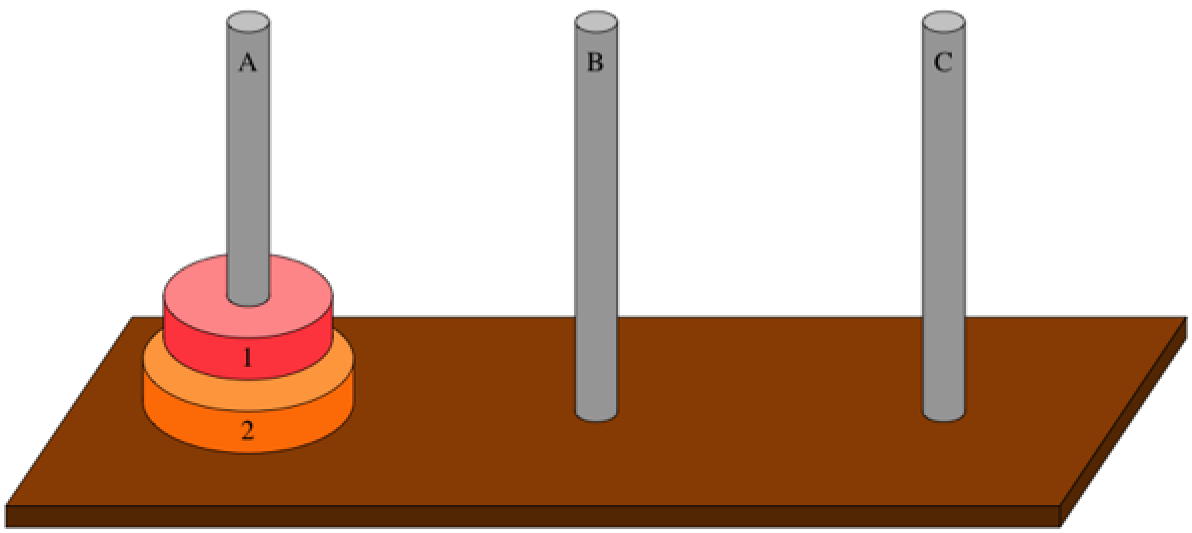


Contd…

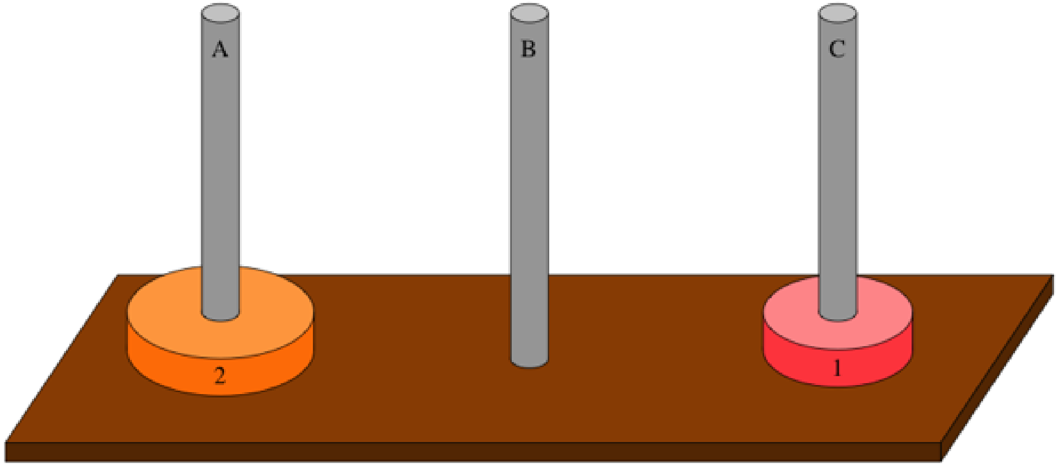
- The goal is to move all the disks to some another tower without violating the sequence of arrangement.
- A few rules to be followed are:
  - Only one disk can be moved among the towers at any given time.
  - Only the "top" disk can be removed.
  - No large disk can sit over a small disk.

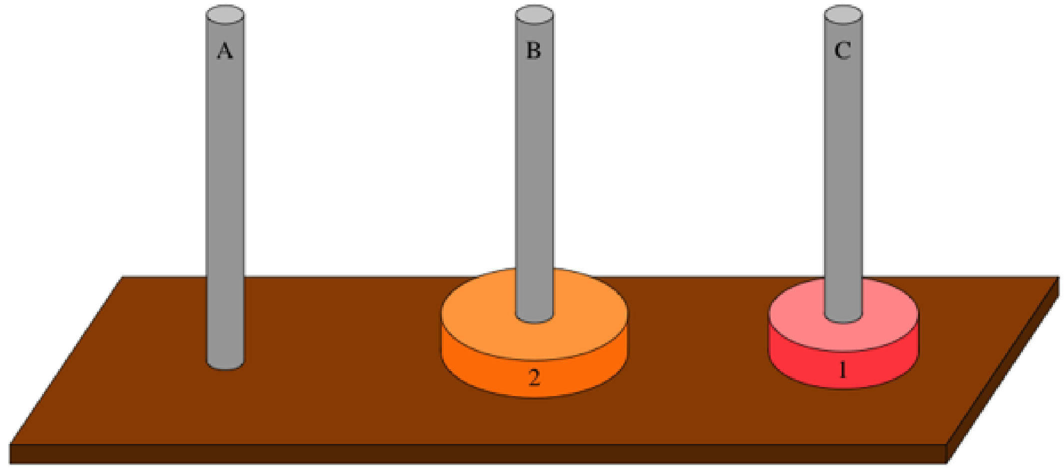
Recursive Solution

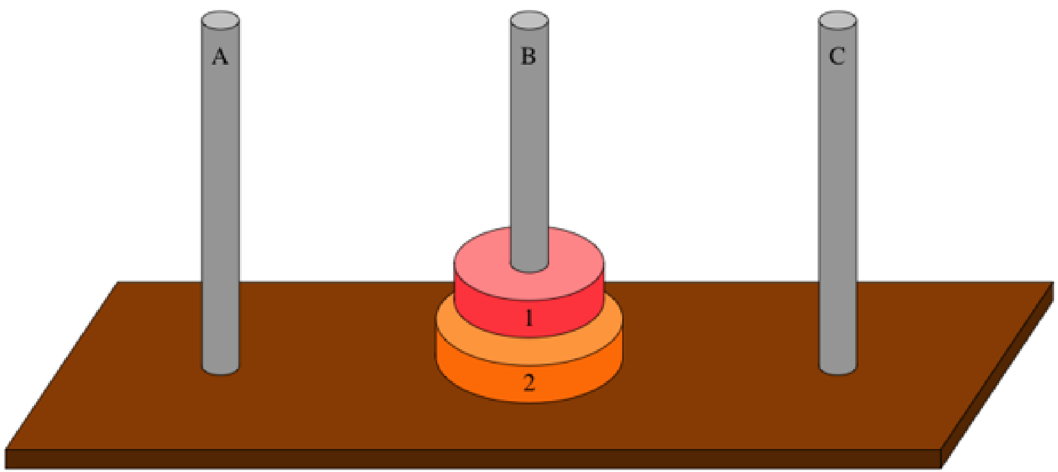
- Lets start with an easy case: one disk, that is, n = 1.
  - This is the base case, as disk 1 can be moved from any peg to any peg.
- What about n = 2?



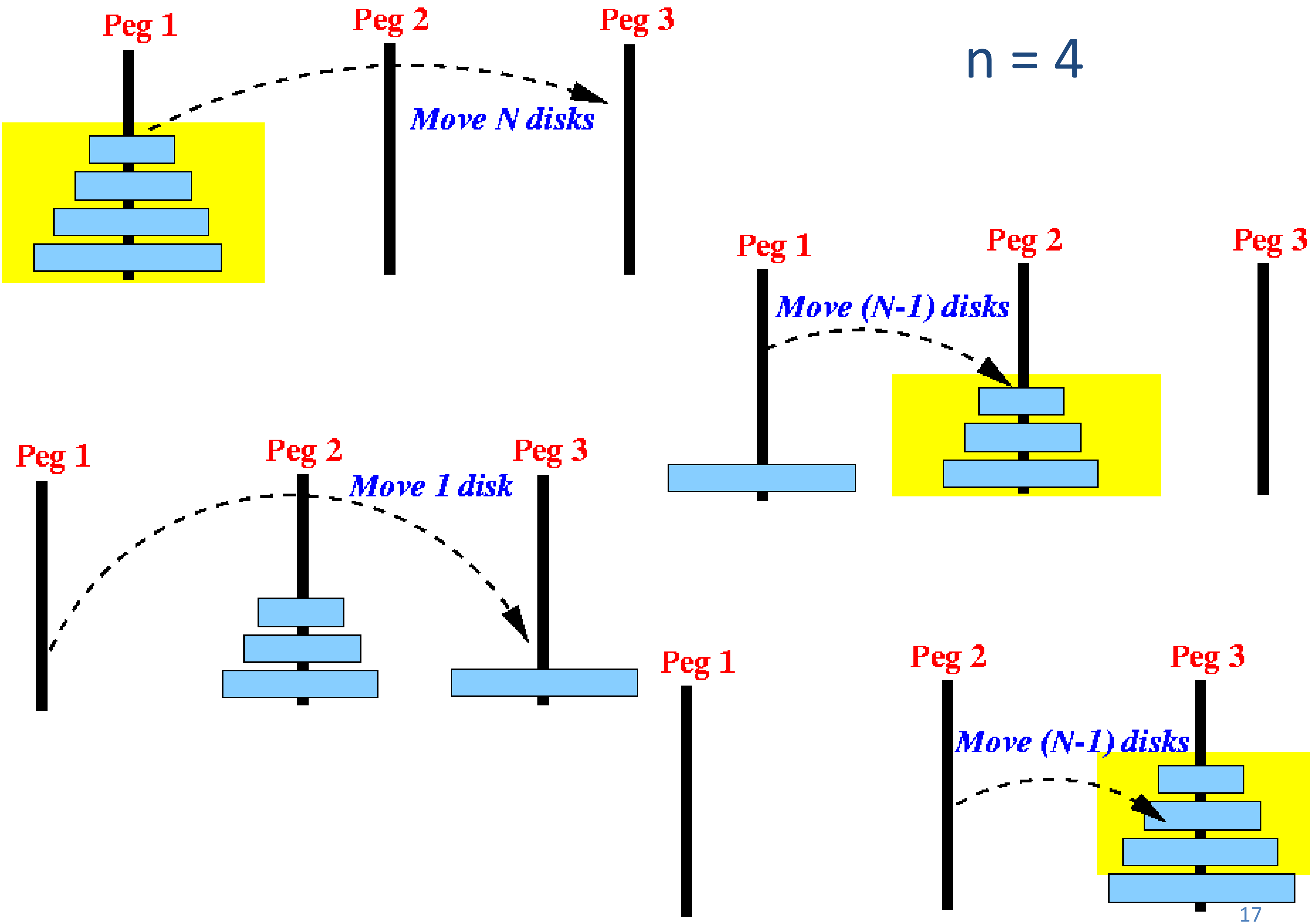
n = 2







$n = 4$



Peg 1
Peg 2
Peg 3
Move N disks
Peg 1
Peg 2
Peg 3
Move (N-1) disks
Peg 1
Peg 2
Peg 3
Move 1 disk
Peg 1
Peg 2
Peg 3
Move (N-1) 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)

Implementation

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

http://www.algomation.com/algorithm/towers-hanoi-recursive-visualization

For the visualization of tower of Hanoi.

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
```

Thankyou