CHAPTER 5 RECURSION

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Recursion is used to solve problems involving iterations in reverse order. It solves a problem by reducing it to an instance of the same problem with smaller input. Recursion is an alternative to each iteration in making a function executes repeatedly.

Properties of Recursion:

A recursive function can go infinite liked a loop, to avoid infinite running of recursive function. There are two properties that a recursive function must have:

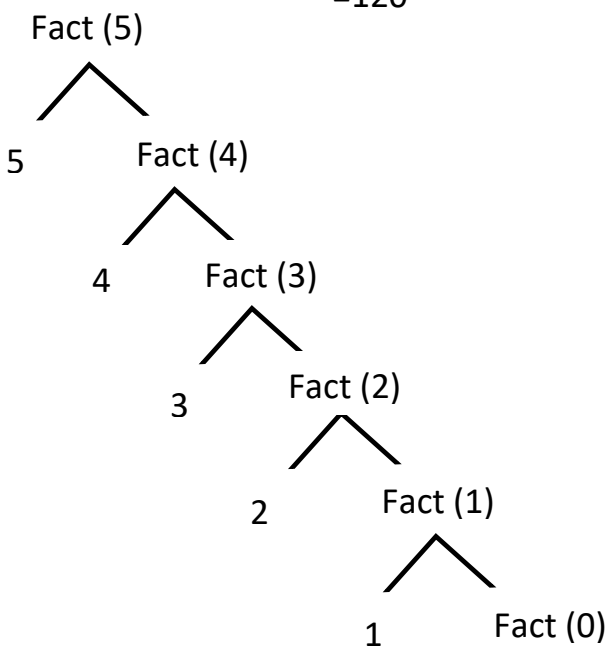
1. **Base Criteria:** There must be at least one base criteria or conditions such that when this condition is met the function stops calling itself recursively.
2. **Progressive Call:** The recursive calls should progress in such a way that each time a recursion call is made, it comes closer to the base case.

The classic example of recursive programming involves computing factorials. The factorial of a number is computed as that number times all of the numbers below it up to and including 1.

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

5!
=5*4!
=5*4*3!
=5*4*3*2!
=5*4*3*2*1!
=5*4*3*2*1*0!
=5*4*3*2*1
=5*4*3*2
=5*4*6
=5*24
=120

Recursion Tree for fact (5)



CHAPTER 5 RECURSION

Difference between Iteration and Recursive function:

Iteration	Recursion
• It is a process of executing statements repeatedly, until some specific condition is specified	• Recursion is a technique of defining anything in terms of itself
• Iteration involves four clear cut steps, initialization, condition, execution and updating	• There must be an base condition inside the recursive function specifying stopping condition
• The value of control variable moves towards the value in condition	• The function state converges towards the base case
• Any recursive problem can be solved iteratively	• Not all problems has recursive solution
• Iteration code tends to be bigger in size	• Recursion decrease the size of code
• An iteration does not use the stack so it's faster than recursion.	• It is usually much slower because all function calls must be stored in a stack to allow the return back to the caller functions.
• Iteration consumes less memory.	• Recursion uses more memory than iteration.

• E.g.
`int fib(int n)`
`{`
`  if( n <= 1 )`
`  return n`
`a = 0, b = 1`
`for( i = 2 to n )`
`{`
`  c = a + b`
`  a = b`
`  b = c`
`}`
`return c`
`}`
	• E.g.
	`int fib(int n)`
	`{`
	`  if(n <= 1)`
	`  {`
	`  return n;`
	`  }`
	`  return fib(n-1) + fib(n-2);`
	`}`

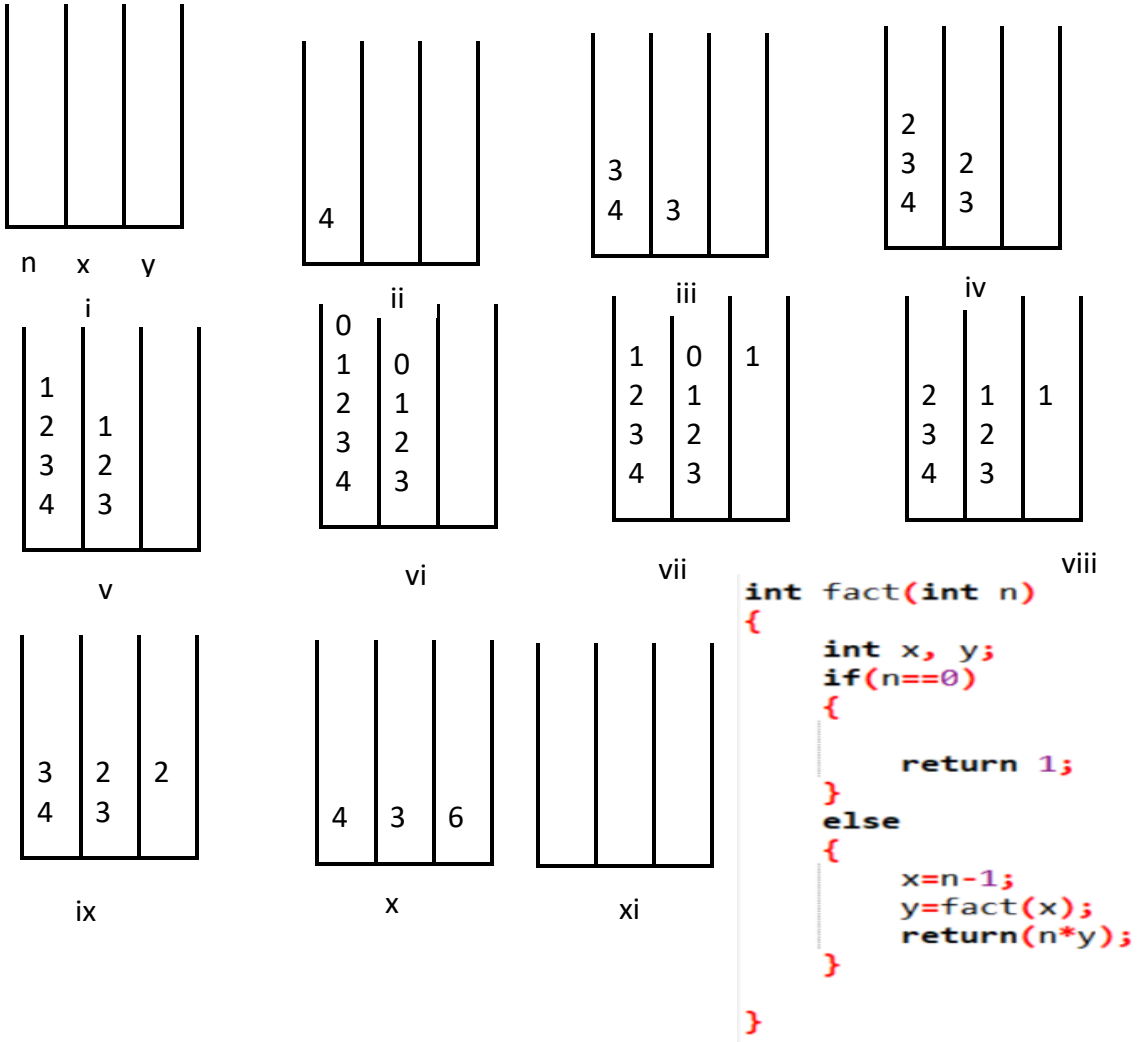
CHAPTER 5 RECURSION

Recursive Program Using Stack:

Recursive functions use something called “the call stack.” When a program calls a function, that function goes on top of the call stack.

Stack is used to keep the successive generations of local variables, the parameters and the returned values. This stack is maintained by the C system and lies inside and invisible to the users.

Each time that a recursive function is entered, a new allocation of its variables is pushed on top of the stack. Any reference to a local variables or parameter is through the current top of the stack. When the function returns, the stack is popped, the top allocation is freed, and the previous allocation becomes the current stack top to be used for referencing local variables. Figure below shows a snapshots of the stack as execution of the fact function proceeds.



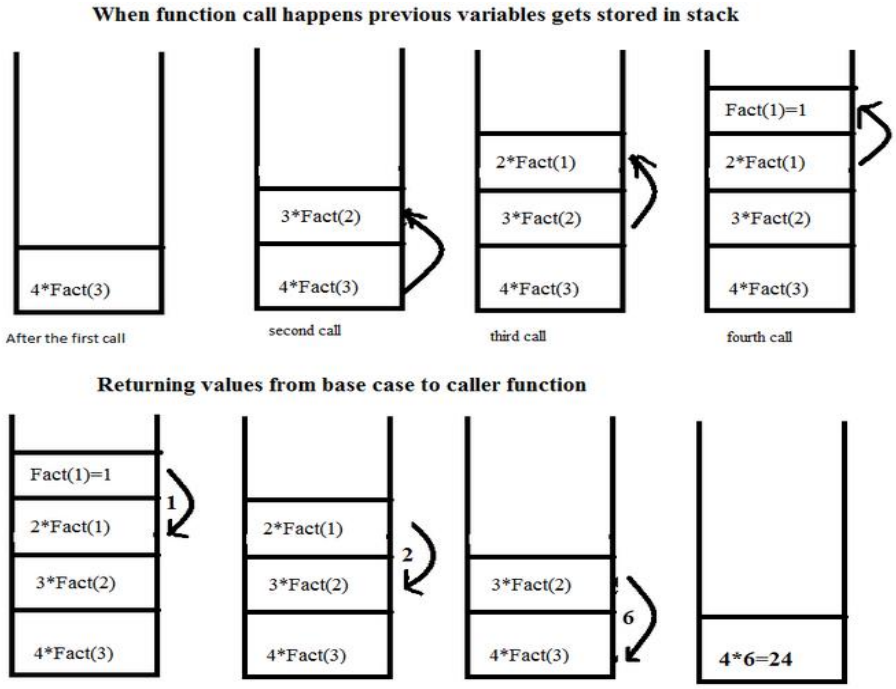

```
int fact(int n)
{
    int x, y;
    if(n==0)
    {

        return 1;
    }
    else
    {
        x=n-1;
        y=fact(x);
        return(n*y);
    }

}
```

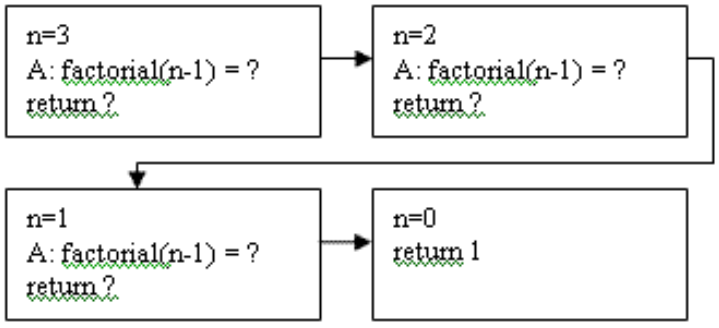
CHAPTER 5 RECURSION

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



Box Trace: It helps to understand how a recursive call works.

- Label each recursive call in the body of the recursive method.
- Represent each call to the method by a new box in which you note the method's local environment.
- Draw an arrow from the statement that initiates the recursive process to the first box.
- After you create the new box and arrow, start executing them body of the method.



CHAPTER 5 RECURSION

Time Complexity: we try to figure out the number of times a recursive call is being made. If n number of times a recursive call is made then the time complexity of recursive function is O (2^n) while iterative function is O (n).

Space Complexity: Space complexity is counted as what amount of extra space is required for a module to execute. In iterations, the compiler hardly requires any extra space. The compiler keeps updating the values of variables used in and space complexity is O (1). But in recursion, the system needs to store activation record each time a recursive call is made and space complexity is O (n).

Recursion Tree:

- Recursion tree is another method for solving the recurrence relations.
- A recursion tree is a tree where each node represents the cost of a certain recursive sub-problem.
- We sum up the values in each node to get the cost of the entire algorithm.

Types of Recursive Functions:

A recursive method is characterized based on:

- Whether the method calls itself or not (direct or indirect recursion)
- Whether there are pending operations at each recursive call (tail recursive or not)

Direct and Indirect Recursion:

Direct Recursion:

If a function calls itself, it's known as direct recursion. A function f1 is called direct recursive if it calls the same function say f1. E.g.

```
void directRecursiveFunction()
{
  // some code...
  directRecursiveFunction();
  // some code...
}
```

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

CHAPTER 5 RECURSION

Indirect Recursion:

When a function is mutually called by another function in a circular manner, the function is called an indirect recursion function. If the function f1 calls another function f2 and f2 calls f1 then it is indirect recursion (or mutual recursion). E.g.

```
void f1();
void f2();
void f1()
{
   // some code...
   f2();
   // some code...
}
void f2()
{
   // some code...
   f1();
   // some code...
}
```

```
void fun1(int a)
{
     if (a > 0)
     {
          printf("%d\n", a);
          fun2(a - 1);
     }
}
void fun2(int b)
{
     if(b > 0)
     {
          printf("%d\n", b);
          fun1(b - 3);
     }
}
```

Tail and Non-Tail Recursion:

Tail Recursion:

A recursive function is called the tail-recursive if the function makes recursive calling itself, and that *recursive call is the last statement executes by the function.* After that, there is no function or statement is left to call the recursive function.

```
void fun1( int num)
{
    if (num == 0)
        return;
    else
        printf ("\n Number is: %d", num);
    return fun1 (num - 1);      // recursive call at the end in the fun() function
}
```

Non-Tail / Head Recursion:

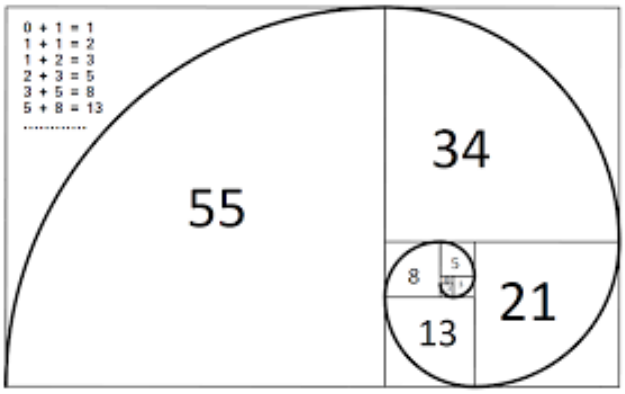
A function is called the non-tail or head recursive if a function makes a recursive call itself, the recursive call will be the first statement in the function. It means there should be no statement or operation is called before the recursive calls.

```
void head_fun (int num)
{
if ( num > 0 )
{
// Here the head_fun() is the first statement to be called
head_fun (num -1);
printf (" %d", num);
}
}
```

CHAPTER 5 RECURSION

Fibonacci Series:

Fibonacci series is a series of numbers formed by the addition of the preceding two numbers in the series. The first two terms are zero and one respectively. The terms after this are generated by simply adding the previous two terms.



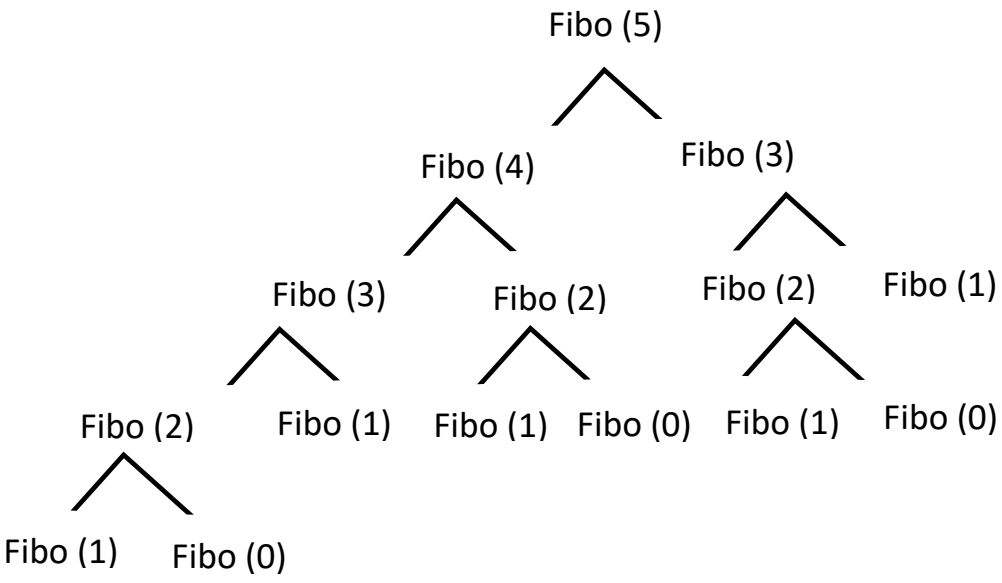
Iteration

```
int fibo(int n)
{
    int i, f1, f2, x;
    if(n==0 || n==1)
    {
        return n;
    }
    else
    {
        f1=0;
        f2=1;
        for (i=2; i<=n; i++)
        {
            x=f1;
            f1=f2;
            f2=x+f1;
        }
        return f2;
    }
}
```

Recursion

```
int fibo(int n)
{
    if (n==0 || n==1)
        return n;
    else
    return (fibo(n-1)+fibo(n-2));
}
```

Tree diagram for fibo (5)

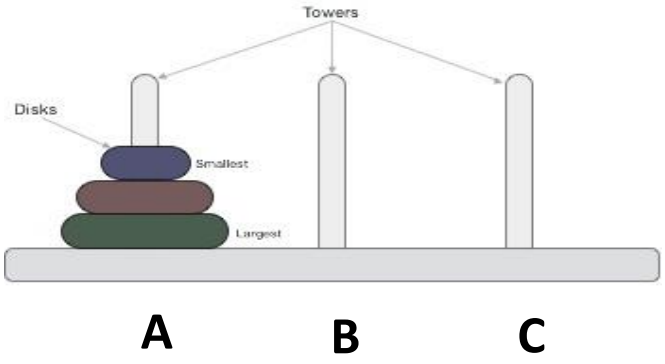


CHAPTER 5 RECURSION

Tower of Hanoi (TOH)

It is also called the problem of Benares Temple or Tower of Brahma or Lucas' Tower. The TOH puzzle was introduced to the west by the French mathematician Edouard Lucas in 1883. Numerous myths regarding the puzzle popped up almost immediately, including one about an Indian temple in Kashi Vishwanath containing a large room with three time-worn posts in it, surrounded by 64 golden disks.

It is a mathematical game or puzzle consisting of three towers (pegs) and a number of disks of various diameters, which can slide onto any tower.



Rules for TOH:

The objective to move all the disks from the peg A to peg C, using peg B as auxiliary. The rules to be followed are:

- Only the top disk on any peg may be moved to any other peg.
- Only one disk can be moved among the towers at any given time.
- Larger disk may never rest on a smaller one.

Recurrence Relation for TOH:

Base case : $H_1 = 1$ (for one disk)

Recursive case: $H_n = H_{n-1} + 1 + H_{n-1}$

$H_n = 2H_{n-1} + 1$
$= 2(2H_{n-2} + 1) + 1$
$= 4H_{n-2} + 2 + 1$
$= 4(2H_{n-3} + 1) + 2 + 1$
$= 2^{n-1} + 2^{n-2} + \ldots + 4 + 2 + 1$ $[\because a_n = ar^{n-1}]$
.
.
$= 2^n - 1$ $[\because S_n = \frac{a(r^n-1)}{(r-1)}, r \neq 1]$

CHAPTER 5 RECURSION

Algorithm for TOH:

To move n disks from A to C, using B as auxiliary:

1. Declare and initialize necessary variables.

   n = number of disks

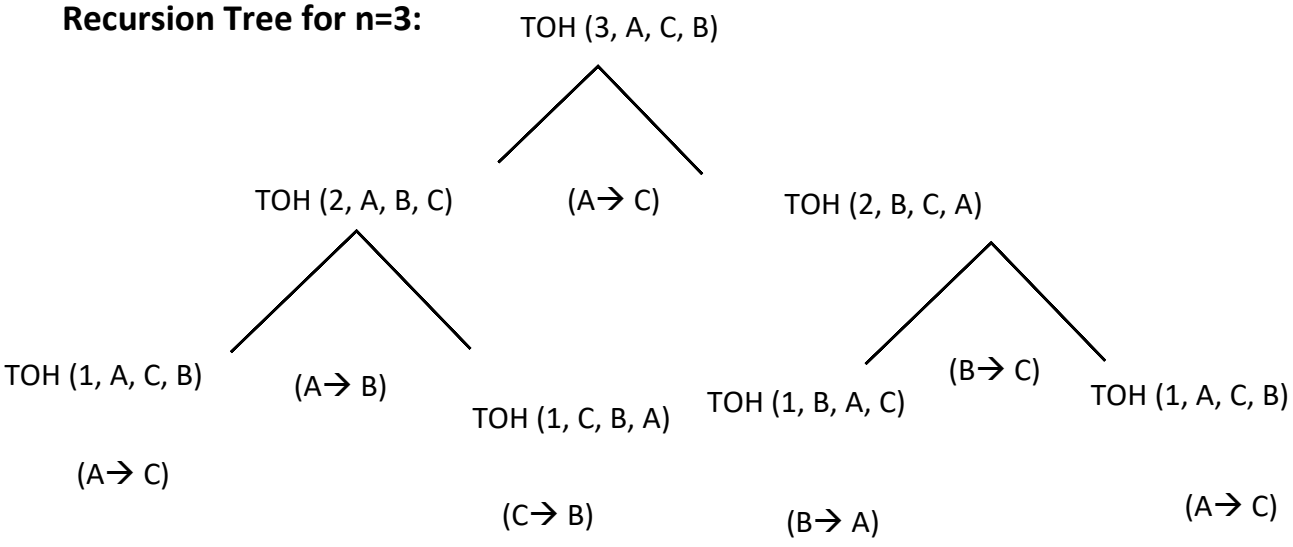
   A='A', B='B', C='C', for three pegs being used.

2. If n == 1,
   - Move the single disk from A to C and stop.
3. Otherwise
   - Move the top n-1 disks from A to B, using C as auxiliary.
   - Move the remaining disk from A to C.
   - Move the n-1 disks from B to C, using A as auxiliary.
4. stop

Algorithm for n=3 disks:

1. Move disk 1 from peg A to peg C
2. Move disk 2 from peg A to peg B
3. Move disk 1 from peg C to peg B
4. Move disk 3 from peg A to peg C
5. Move disk 1 from peg B to peg A
6. Move disk 2 from peg B to peg C
7. Move disk 1 from peg A to peg C

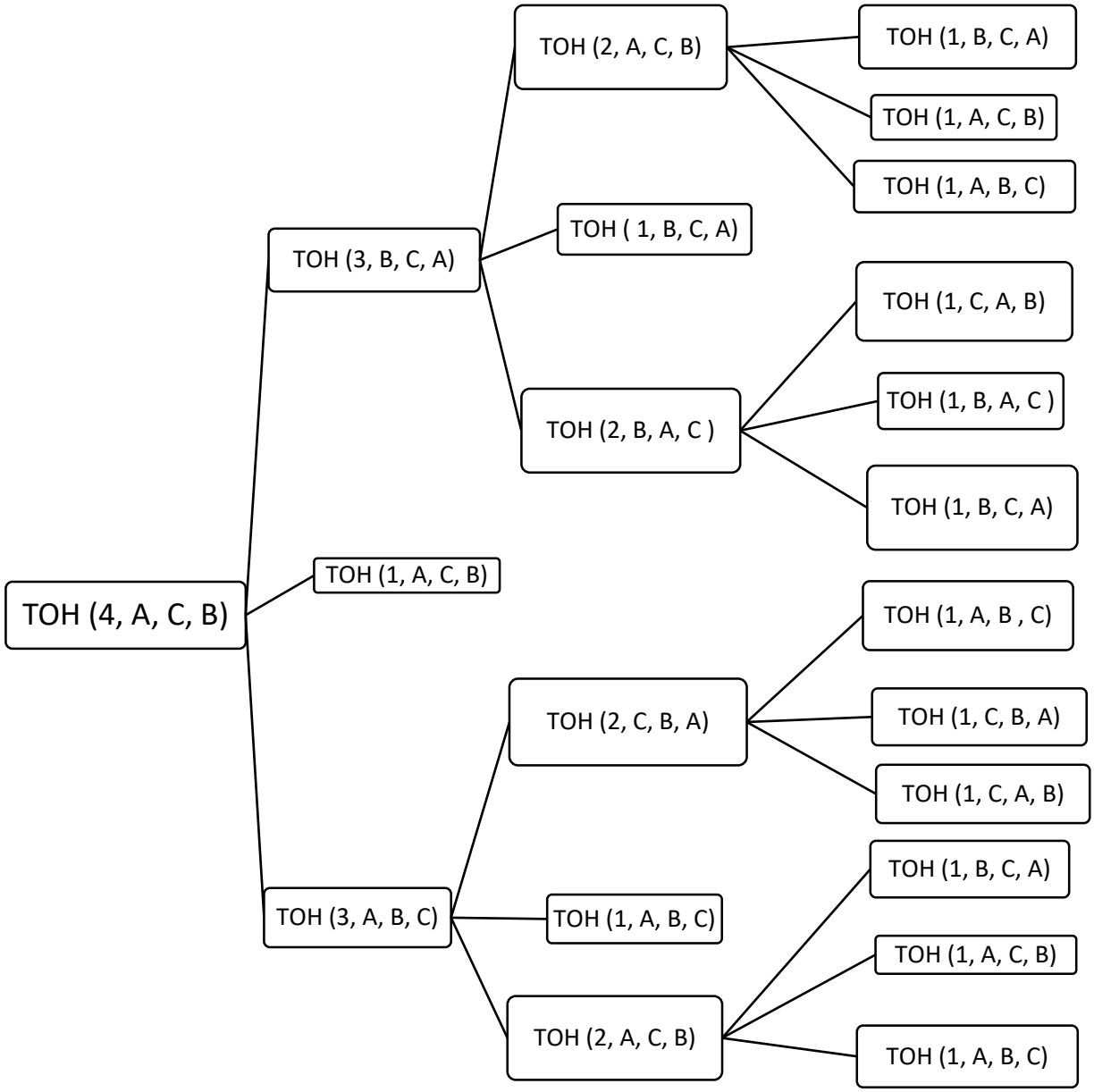
Recursion Tree for n=3:



CHAPTER 5 RECURSION

Algorithm for n=4 disks:

1. Move disk 1 from peg A to peg B
2. Move disk 2 from peg A to peg C
3. Move disk 1 from peg B to peg C
4. Move disk 3 from peg A to peg B
5. Move disk 1 from peg C to peg A
6. Move disk 2 from peg C to peg B
7. Move disk 1 from peg A to peg B
8. Move disk 4 from peg A to peg C
9. Move disk 1 from peg B to peg C
10. Move disk 2 from peg B to peg A
11. Move disk 1 from peg C to peg A
12. Move disk 3 from peg B to peg C
13. Move disk 1 from peg A to peg B
14. Move disk 2 from peg A to peg C
15. Move disk 1 from peg B to peg C



CHAPTER 5 RECURSION

Applications of Tower of Hanoi:

- It is used in psychological research on problem-solving.
- It is used in physical design of the game components.
- It is used as a backup rotation scheme when performing computer data backups where multiple tapes/media are involved.

Application of Recursion:

- The most important data structure 'Tree' doesn't exist without recursion we can solve that in iterative way also but that will be a very tough task.
- The mathematical problem can't be solved in general, but that can only be solved using recursion up to a certain extent.
- Sorting algorithms (Quick sort, Merge sort, etc.) uses recursion.
- All the puzzle games (Chess, Candy crush, etc.) broadly uses recursion.
- It is the backbone of searching, which is most important thing.
- This is the backbone of AI.