



RIZAL TECHNOLOGICAL UNIVERSITY  
COLLEGE OF ENGINEERING ARCHITECTURE AND TECHNOLOGY

# DATA STRUCTURE AND ALGORITHM



# RECURSION

- ❑ What is Recursion?
- ❑ Types of Recursion
- ❑ Applications of Recursion
- ❑ Recursion and Memory
- ❑ Recursion vs Iteration
- ❑ What is Backtracking
- ❑ Algorithm and Backtracking

# Objectives:

- ❑ Define what recursion is.
- ❑ Differentiate iteration from recursion.
- ❑ Apply recursion in different mathematical applications.
- ❑ Define backtracking and its relation to algorithm.

# RECURSION

- technique in programming in which the function calls itself directly or indirectly
- has the capability to save condition it was in or the particular process it was serving when calling itself
- The design of a recursive method consists of the following elements:
  - one or more **stopping conditions** that can be directly evaluated for certain arguments.
  - one or more **recursive steps**, in which a current value of the method can be computed by repeated calling of the method with arguments that will eventually arrive at a stopping condition

\*\*\* a recursive method must always have a **base case** a condition when it does not make a further call to itself. It prevents infinite recursion which results to the end of the program.

\*\*\* a **recursive expression** is a function, algorithm or sequence of instruction that loops back to the beginning of itself until it detects that some condition has been satisfied.



# Two Classes of Recursion

- **Direct Recursion** – recursion wherein procedures directly invoke themselves
- **Indirect Recursion** – recursion wherein a chain invocation of procedures is permitted resulting in a closed circle (ex. Procedure A calls procedure B, which calls procedure C, which calls procedure A)

```
void directRec()  
{  
    // code  
    directRec();  
    // code  
}
```

```
void indirectRecA()  
{  
    // code  
    indirectRecB();  
    // code  
}  
  
void indirectRecB()  
{  
    // code  
    indirectRecC();  
    // code  
}  
  
void indirectRecC()  
{  
    // code  
    indirectRecA();  
    // code  
}
```

# Types of Recursion

- **Linear Recursion.** is a recursion where only one call is made from within the function each time it runs. e.g. Factorial
- **Tail Recursion.** is a recursion in which the last operation of the function is a recursive call. e.g. Function for computing the Greatest Common Divisor (GCD) also known as Greatest Common Factor.
- **Binary Recursion.** is a recursive function that calls itself twice in the run of the function. e.g. mathematical combinations.

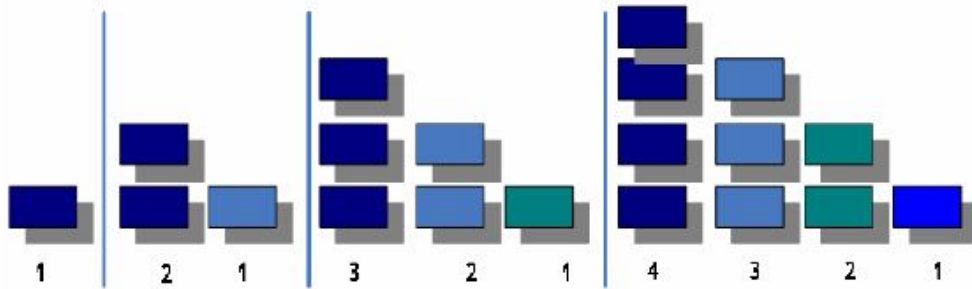
# Types of Recursion

- **Exponential Recursion.** is a recursion that numerous calls are made in the function leading to exponential growth in the number of recursive call. e.g. computation of all permutations of a data set.
- **Nested Recursion.** it is said that one of the arguments in a recursive function is the recursive function itself that makes it grow extremely fast. e.g. Ackermann's function where it takes two natural numbers as arguments and yields another natural number that makes it grow rapidly.
- **Mutual Recursion.** is a recursion that doesn't necessarily need to call itself. For example, function A calls function B which calls function C which in turn calls function A. An example application would be the one that determines which integer is odd or even.

# Applications of Recursion: Triangular Numbers

- the sum of 1 to n numbers arranged in equilateral triangle
- seen by row where each row is longer than the previous row
- the sum of consecutive integers

Example:  $n = 4$



Total =  $1 + 2 + 3 + 4 = 10$



# Applications of Recursion: Triangular Numbers

## Recursive Algorithm

- There must be at least one case (the base case), for a small value of  $n$  that can be solved directly
- A problem of a given size  $n$  can be split into one or more smaller versions of the same problem
- Identify the base case and provide a solution to it
- Create a strategy to split the problem into smaller versions of itself while making progress toward the base case
- Merge the solution that will solve the larger problem

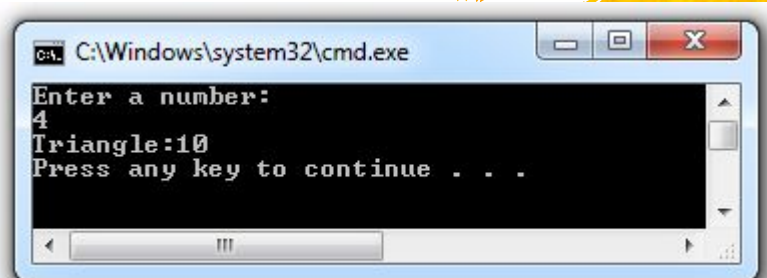
# Applications of Recursion: Triangular Numbers

```
//method that sums up all columns from height of n to a height of 1
```

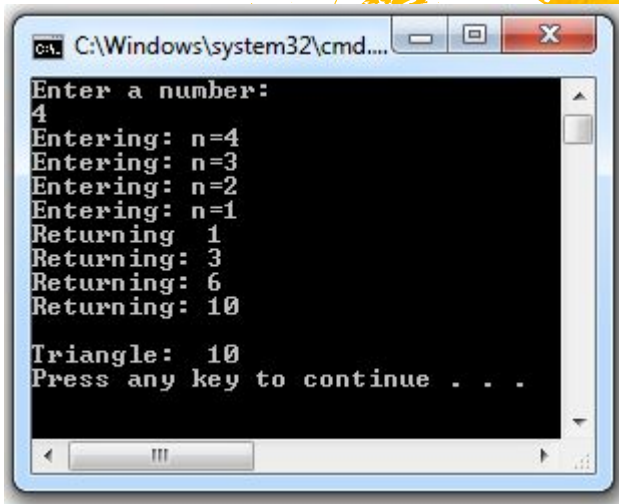
```
public static int triangle(int n)
{
    if(n==1)
        return 1;
    else
        return (n + triangle(n-1));
}
```

```
static int triangle(int n)
{
    System.out.println("Entering: n=" + n);

    if (n==1) // base case
    {
        System.out.println("Returning 1");
        return 1;
    }
    else
    {
        int temp = n + triangle(n-1);
        System.out.println("Returning: " + temp);
        return temp;
    }
}
```



```
C:\Windows\system32\cmd.exe
Enter a number:
4
Triangle:10
Press any key to continue . . .
```



```
C:\Windows\system32\cmd.exe
Enter a number:
4
Entering: n=4
Returning: 3
Returning: 6
Returning: 10
Triangle: 10
Press any key to continue . . .
```

# Applications of Recursion: Factorial

- It is the product of a series of consecutive positive integers from 1 to a given number,  $n$
- Expressed by the symbol  $!$

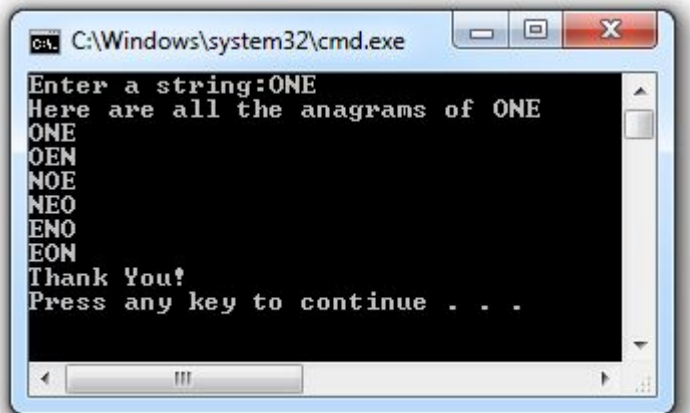
<b>n</b>	<b>n!</b>		
1	1	1	1
2	$2 \times 1$	$= 2 \times 1!$	$= 2$
3	$3 \times 2 \times 1$	$= 3 \times 2!$	$= 6$
4	$4 \times 3 \times 2 \times 1$	$= 4 \times 3!$	$= 24$
5	$5 \times 4 \times 3 \times 2 \times 1$	$= 5 \times 4!$	$= 120$
6	etc	etc	

# Applications of Recursion: Anagram

- It forms a real English word or not using the same letters, no more, no less

## Algorithm to Anagram a word:

- Anagram the rightmost  $n-1$  letters
- Rotate all  $n$  letters
- Repeat these steps  $n$  time



```
C:\Windows\system32\cmd.exe
Enter a string:ONE
Here are all the anagrams of ONE
ONE
OEN
NOE
NEO
ENO
EON
Thank You!
Press any key to continue . . .
```

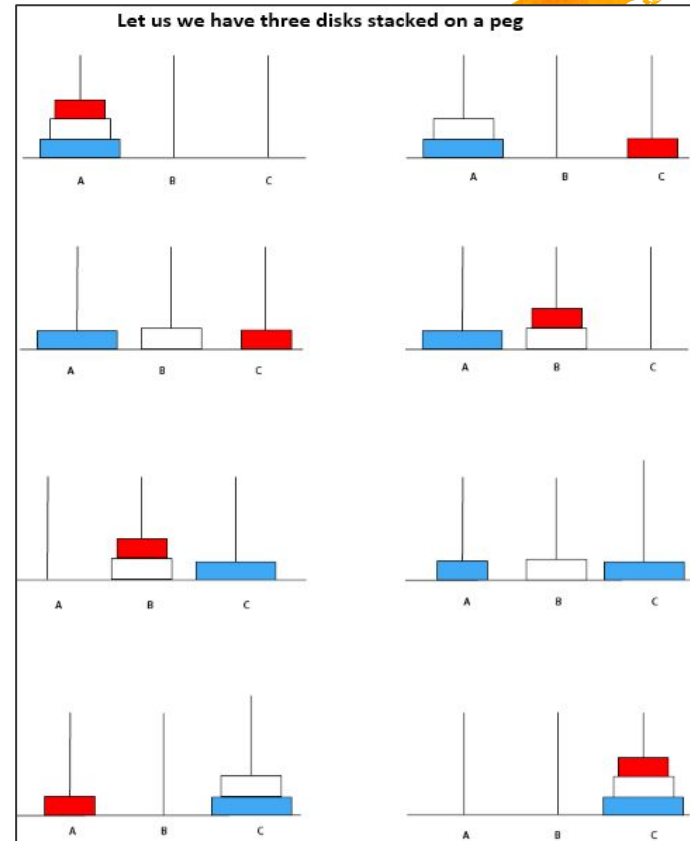


# Applications of Recursion: Tower of Hanoi

- Invented by a French mathematician named Edouard Lucas in 1833.
- Its objective is to transfer the entire tower to the last peg, moving only one disk at a time without moving a larger one onto a smaller.

## Algorithm

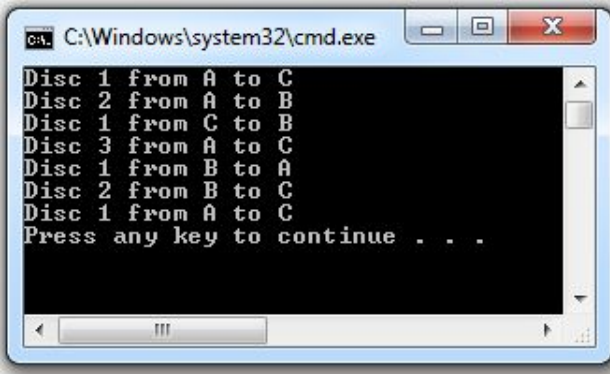
- Move the smaller disc from the top  $n-1$  from A to B
- Move the remaining (larger) disc from A to C
- Move the smaller disc from B to C



# Applications of Recursion: Tower of Hanoi

```
// Java recursive function to solve tower of hanoi puzzle
class towersofHanoi
{
    static void towerHanoi(int topN, char from, char inter, char to)
    {
        if(topN==1)
            System.out.println("Disc 1 from " + from + " to " + to);
        else
        {
            towerHanoi(topN-1, from, to, inter);
            System.out.println("Disc " + topN + " from " + from + " to " + to);
            towerHanoi(topN-1, inter, from, to);
        }
    }

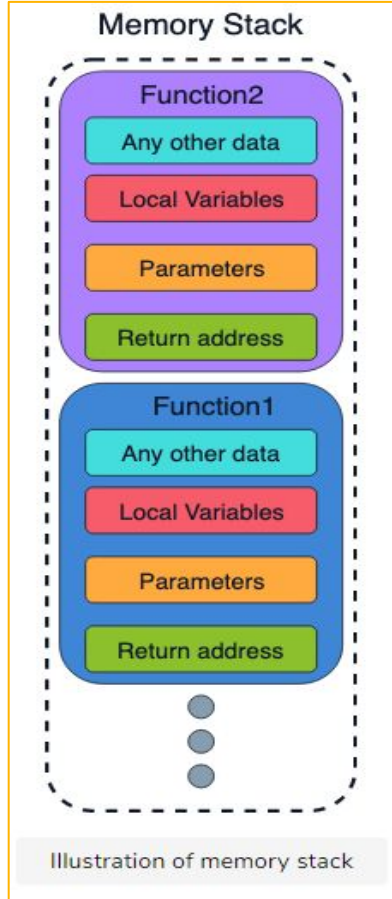
    public static void main(String args[])
    {
        int n = 3; // Number of disks
        towerHanoi(n, 'A', 'B', 'C'); // A, B and C are names of rods
    }
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the Java program for 3 disks. The output shows the sequence of moves: Disc 1 from A to C, Disc 2 from A to B, Disc 1 from C to B, Disc 3 from A to C, Disc 1 from B to A, Disc 2 from B to C, and Disc 1 from A to C. The prompt "Press any key to continue . . ." is visible at the bottom of the window.

```
C:\Windows\system32\cmd.exe
Disc 1 from A to C
Disc 2 from A to B
Disc 1 from C to B
Disc 3 from A to C
Disc 1 from B to A
Disc 2 from B to C
Disc 1 from A to C
Press any key to continue . . .
```

# Recursion and Memory

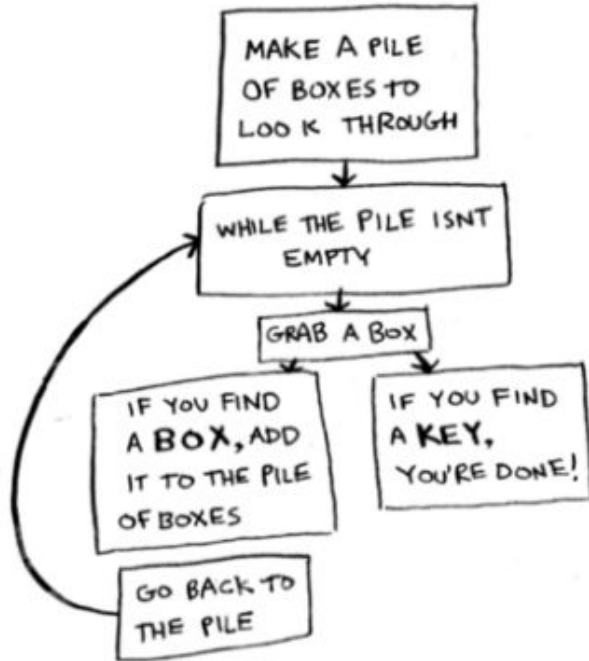


There are two storage areas involved: the **stack** and the **heap**. The stack is where the current *state* of a method call is kept (ie local variables and references), and the heap is where objects are stored.

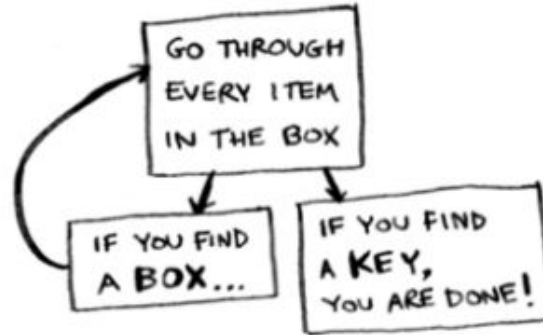
- When a function is called, its memory is allocated on a **stack**.
- When a function executes, it adds its **state** data to the **top** of the stack.
- When the function exits, this data is removed from the stack.

# Recursion vs Iteration

Iterative Approach



Recursive Approach





# Recursion vs Iteration

- Recursion uses **selection structure**.
- **Infinite recursion** occurs if the recursion step does not reduce the problem in a manner that converges on some condition (**base case**) and Infinite recursion can crash the system.
- Recursion terminates when a **base case** is recognized.
- Recursion is usually **slower than iteration** due to the overhead of maintaining the stack.
- Recursion uses **more memory than iteration**.
- Recursion makes the **code smaller**.

- Iteration uses **repetition structure**.
- An infinite loop occurs with iteration if the loop condition test never becomes false and Infinite looping uses CPU cycles repeatedly.
- An iteration **terminates** when the **loop condition fails**.
- An iteration does not use the **stack** so it's **faster than recursion**.
- Iteration consumes **less memory**.
- Iteration makes the **code longer**.

# Recursion vs Iteration

```
public class RecursionExample {  
    public static void main(String args[]) {  
        RecursionExample re = new RecursionExample();  
        int result = re.factorial(4);  
        System.out.println("Result:" + result);  
    }  
    public int factorial(int n) {  
        if (n==0) {  
            return 1;  
        }  
        else {  
            return n*factorial(n-1);  
        }  
    }  
}
```

```
public class IterationExample {  
    public static void main(String args[]) {  
        for(int i = 1; i <= 5; i++) {  
            System.out.println(i + " ");  
        }  
    }  
}
```

# Recursion: Pros and Cons

## Pros

- Can reduce time complexity.
- Adds clarity and reduces the time needed to write and debug code.
- Is better at tree traversal

## Cons

- Uses more memory.
- Can be slow.

# Backtracking Algorithms

**Backtracking** is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

	Q		
			Q
Q			
		Q	

Source			
			Dest.





RIZAL TECHNOLOGICAL UNIVERSITY  
COLLEGE OF ENGINEERING ARCHITECTURE AND TECHNOLOGY

Thank You 😊  
Keep safe  
and God bless!

