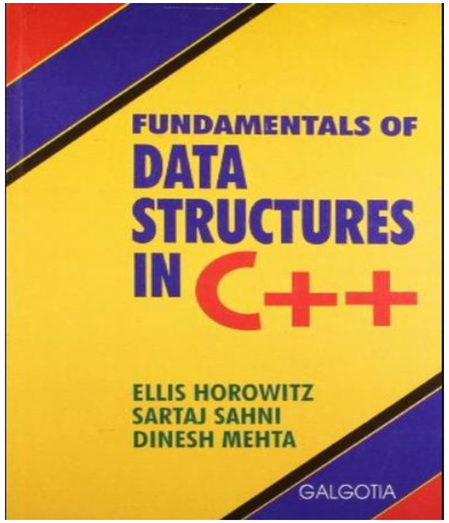
Mar24 : Day 1

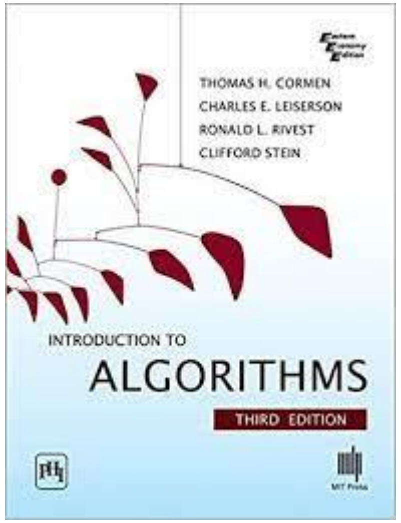
Kiran Waghmare

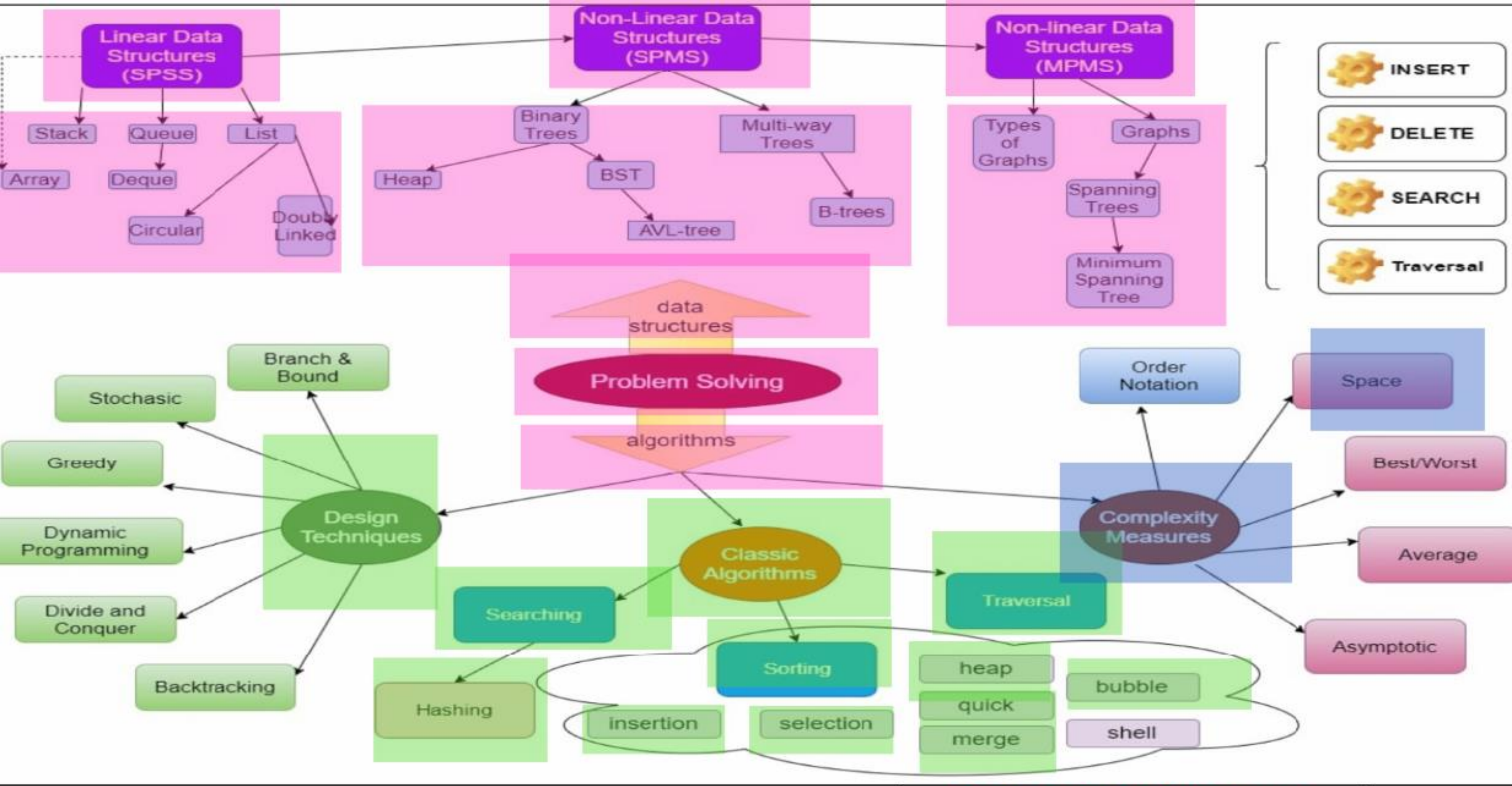
CDAC Mumbai

Module 2: Algorithms and Data Structures

- **Text Book:**
  - Fundamentals of Data Structures in C++ by Horowitz, Sahani & Mehta
- **Topics:**
  - 1.Problem Solving & Computational Thinking
  - 2.Introduction to Data Structures & Recursion
  - 3.Stacks
  - 4.Queues
  - 5.Linked List Data Structures
  - 6.Trees & Applications
  - 7.Introduction to Algorithms
  - 8.Searching and Sorting
  - 9.Hash Functions and Hash Tables
  - 10.Graph & Applications
  - 11.Algorithm Designs







Linear Data Structures (SPSS)

Stack

Queue

List

Array

Deque

Circular

Doubly Linked

Non-Linear Data Structures (SPMS)

Binary Trees

Multi-way Trees

Heap

BST

AVL-tree

B-trees

Non-linear Data Structures (MPMS)

Types of Graphs

Graphs

Spanning Trees

Minimum Spanning Tree

INSERT

DELETE

SEARCH

Traversal

data structures

Problem Solving

algorithms

Branch & Bound

Stochasic

Greedy

Dynamic Programming

Divide and Conquer

Backtracking

Design Techniques

Classic Algorithms

Searching

Hashing

Sorting

Traversal

insertion

selection

heap

quick

merge

bubble

shell

Order Notation

Space

Best/Worst

Average

Asymptotic

Complexity Measures

Mind Map by Dr. M Sasikumar, CDAC Mumbai

Agenda

- **Problem Solving & Computational Thinking**
- **Algorithm & Data Structure**

  OODesign: ADTs
- **Recursion**

  Base condition

  Direct & indirect recursion

  Memory allocation

  Pros and Cons

  Complexity analysis

Why Study Algorithms and Data Structures?

- World domination

For fun and profit.

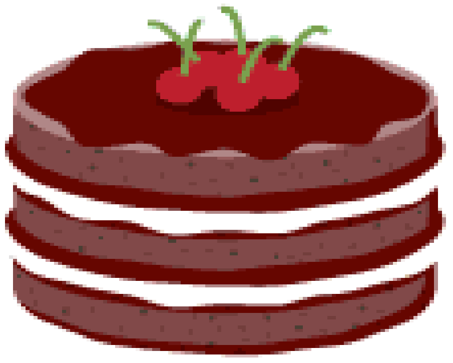
Algorithms are Everywhere

- Search Engines
- GPS navigation
- Self-Driving Cars
- E-commerce
- Banking
- Medical diagnosis
- Robotics
- Algorithmic trading
- and so on …

What is Computational Thinking?

- **Computational thinking is a problem solving process that includes:**

- **Decomposition:**
  - Breaking down data, processes, or problems into smaller, manageable parts.
- **Pattern Recognition:**
  - Observing patterns, trends, and regularities in data.
- **Abstraction:**
  - Identifying the general principles that generate these patterns.
  - This involves filtering out the details we do not need in order to solve a problem.
- **Algorithm Design:**
  - Developing the step by step instructions for solving this and similar problems.

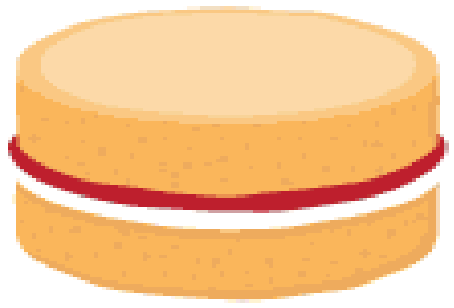


Preheat oven to 190°C

Blend butter, sugar & flour

Bake for 25 minutes

Whisk 300ml of cream



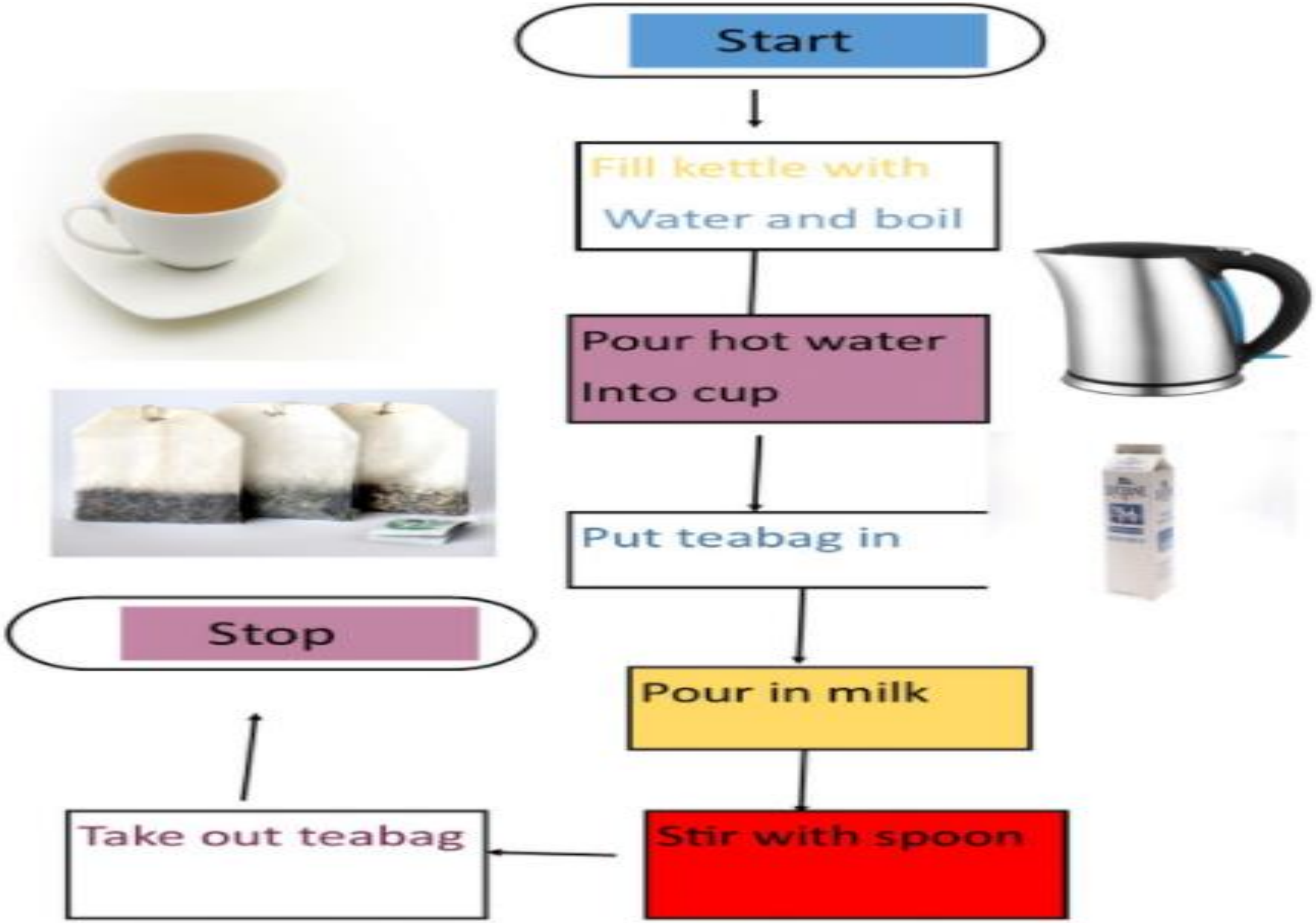
Preheat oven to 180°C

Whisk all butter and sugar

Mix in eggs

Bake for 30 minutes

Write Algorithm to prepare a Tea



Start

Fill kettle with

Water and boil

Pour hot water

Into cup

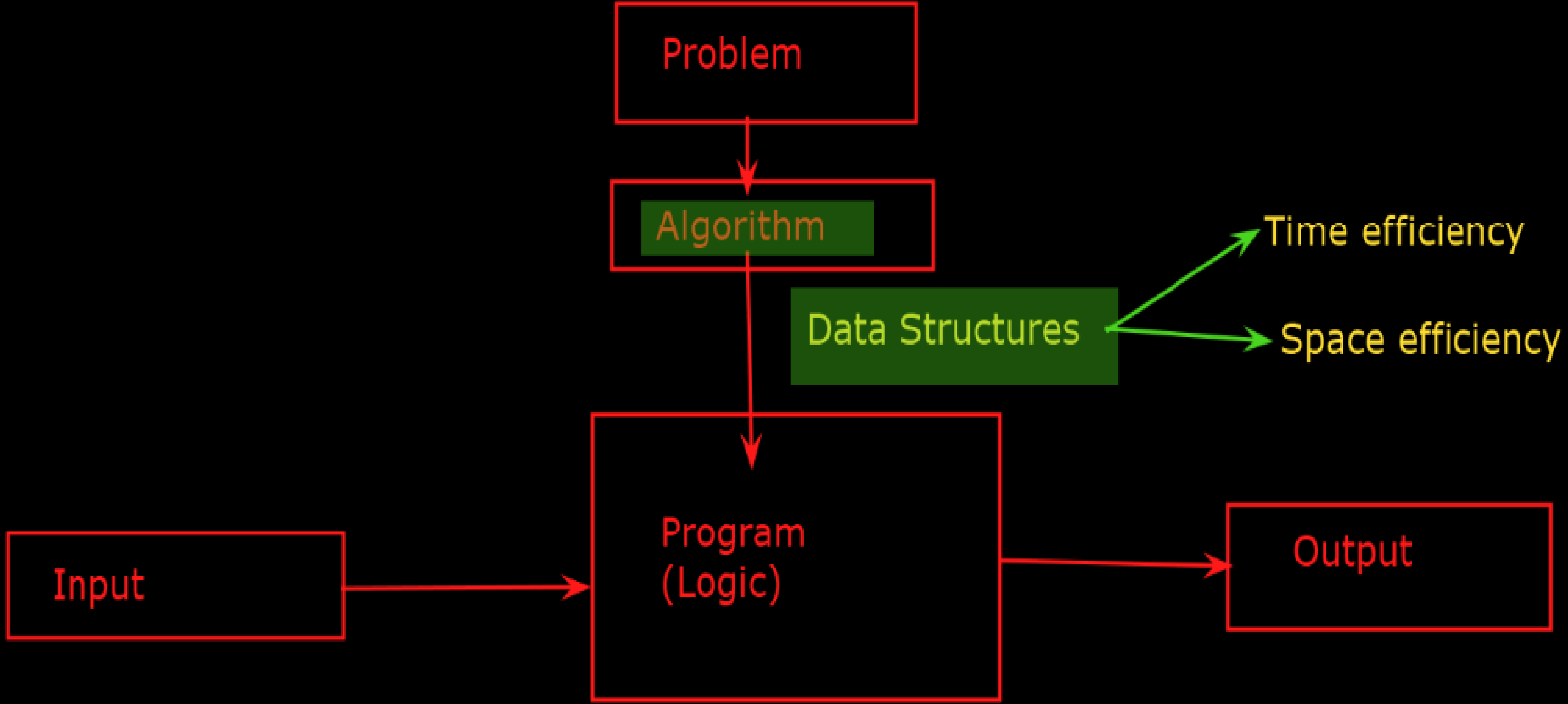
Put teabag in

Pour in milk

Stir with spoon

Take out teabag

Stop



Problem

Algorithm

Data Structures

Time efficiency

Space efficiency

Input

Program
(Logic)

Output

Algorithm:
-sequence of unambiguous instructions

Dataflow of an Algorithm

- **Problem:**
  - A problem can be a real-world problem or any instance from the real-world problem for which we need to create a program or the set of instructions. The set of instructions is known as an algorithm.
- **Algorithm:**
  - An algorithm will be designed for a problem which is a step by step procedure.
- **Input:**
  - After designing an algorithm, the required and the desired inputs are provided to the algorithm.
- **Processing unit:**
  - The input will be given to the processing unit, and the processing unit will produce the desired output.
- **Output:**
  - The output is the outcome or the result of the program.

Algorithm Design Strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Greedy approach
- Dynamic programming
- Backtracking and branch and bound
- Space and time tradeoffs

Invented or applied by many genius in CS

Some Well-known Computational Problems

- **Sorting**
  - e.g., school days…height wise, now rotation wise
- **Searching**
  - E.g. read books. Alexa, google
- **Shortest paths in a graph**
- **Minimum spanning tree**
- **Primality testing**
- **Traveling salesman problem**
- **Knapsack problem**
- **Chess**
- **Towers of Hanoi**

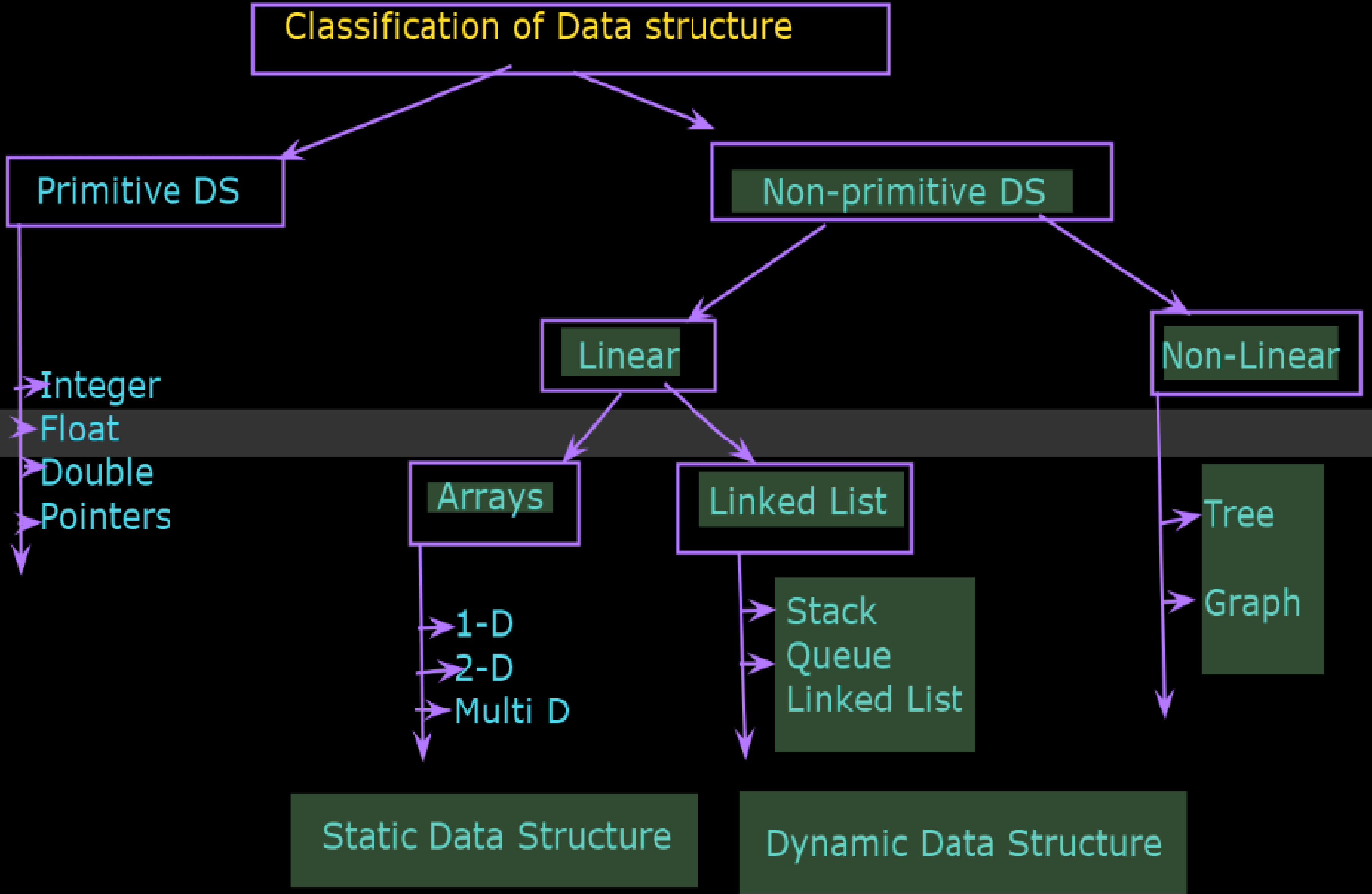
Data stucture

- **A data structure is a data organization, management and storage format that enables efficient access and modification.**
- **It is a way in which data is stored on a computer**
- **Need of Data Structure:**
  - Each data structure allows data to be stored in specific manner.
  - Data structure allows efficient data search and retrieval.
  - Specific Data structure are decided to work for specific problems.
  - It allows to manage large amount of data such as databases and indexing services such as hash table.

Operation:

- -Insertion
- -Deletion
- -Travesing
- -Search
- -Sorting

- -Merge
- -Combine



Classification of Data structure

Primitive DS

- Integer
- Float
- Double
- Pointers

Non-primitive DS

Linear

Arrays

- 1-D
- 2-D
- Multi D

Linked List

- Stack
- Queue
- Linked List

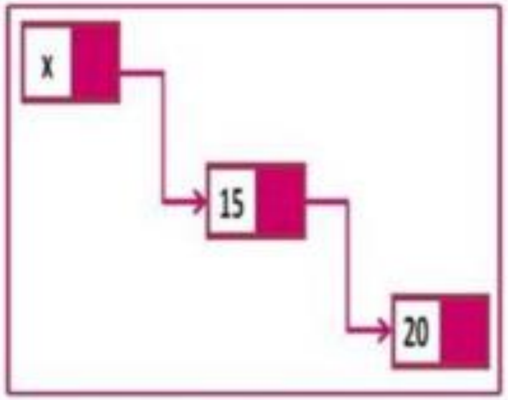
Non-Linear

- Tree
- Graph

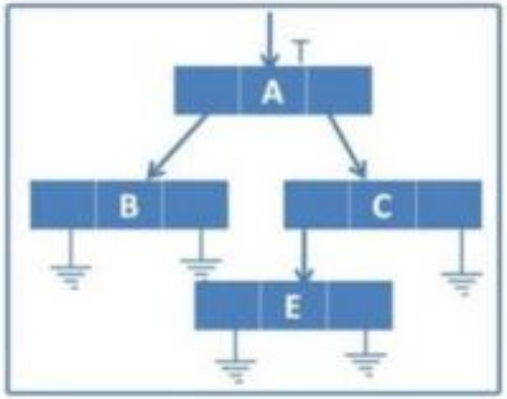
Static Data Structure

Dynamic Data Structure

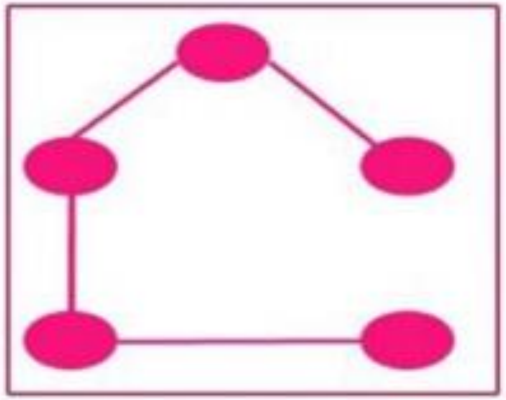
Sorting



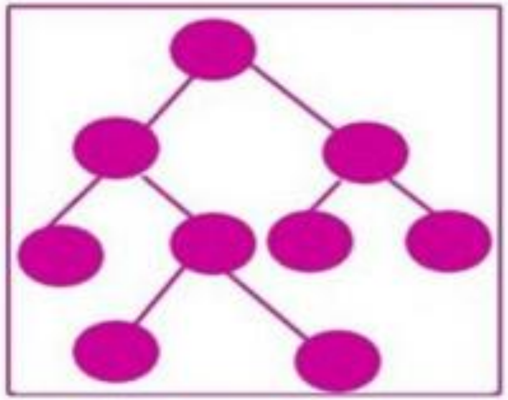
Link list



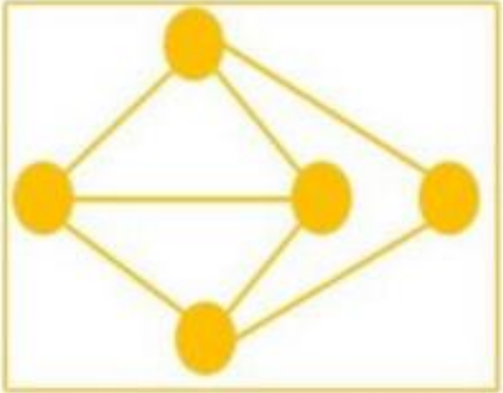
list



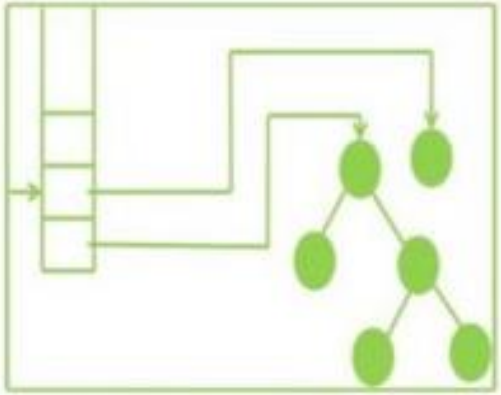
spanning tree



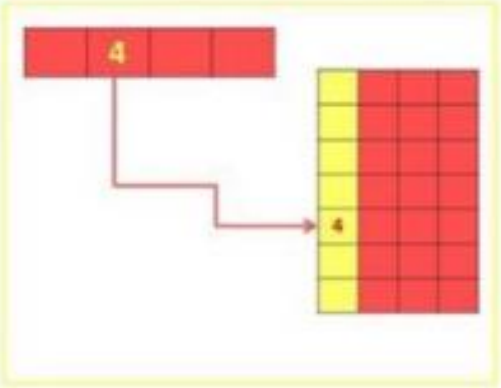
Tree



Graph



Stack



Hashing

By...navinkumardhoprephotography.com

-Linear :

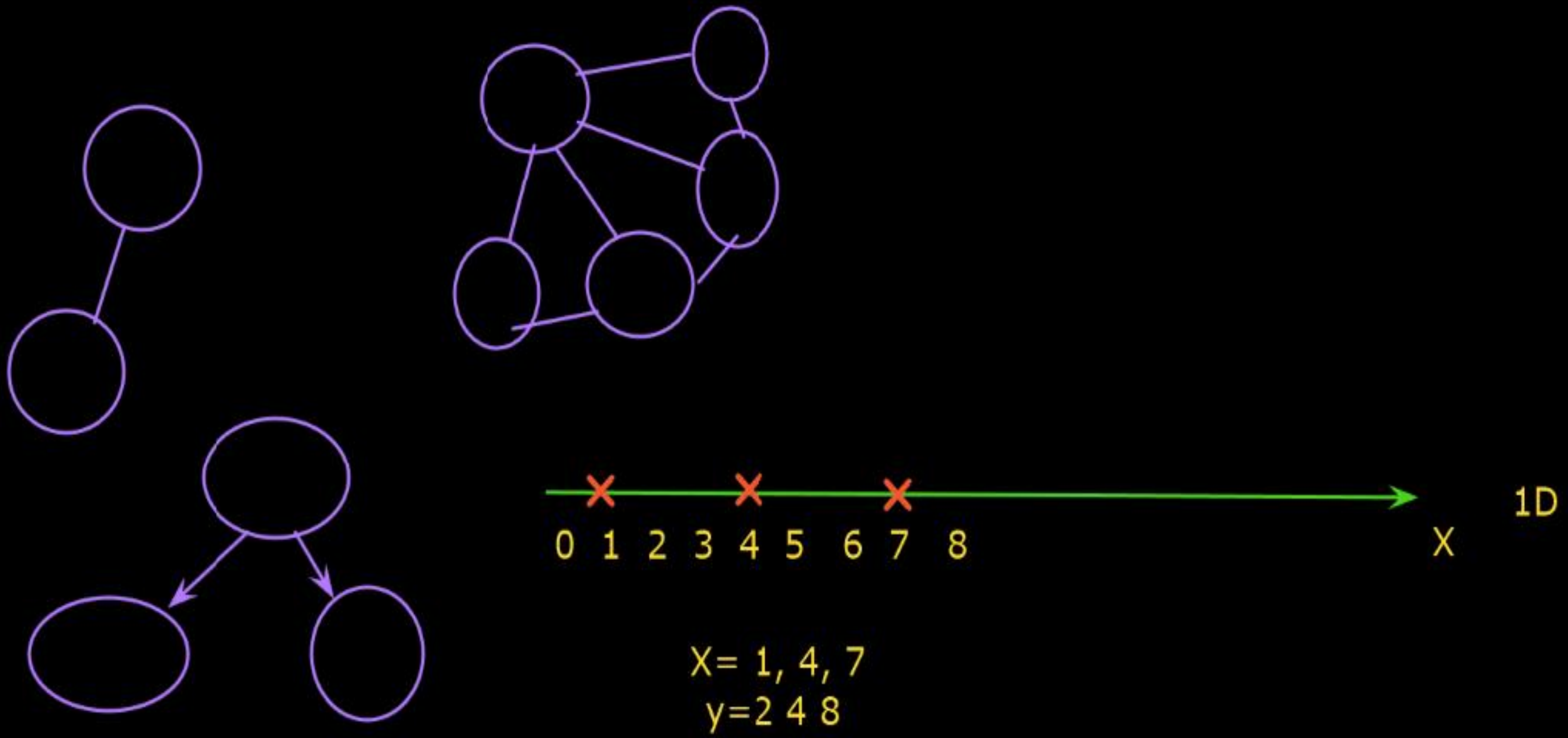
-Elements are arranged in one dimension, also known as linear dimension.

-Array, stack, list, queue etc ...

-Non Linear :

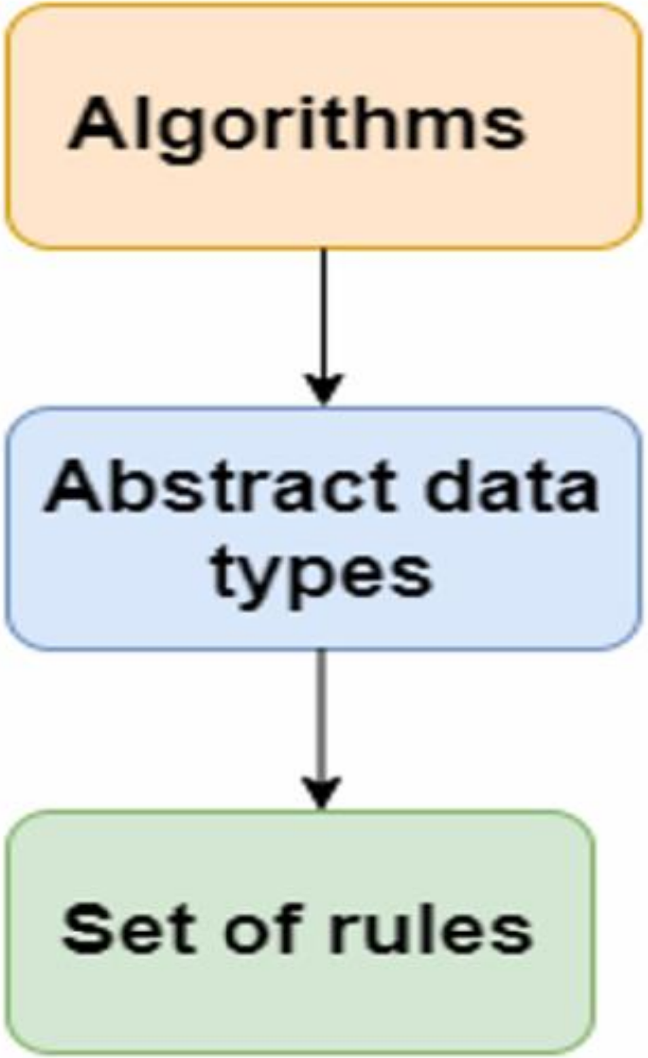
-Elements are arranged in one to many and many to many dimension.

-Tree, heap, graph,..



Abstract Data Type (ADT)

Abstract Data Type (ADT)



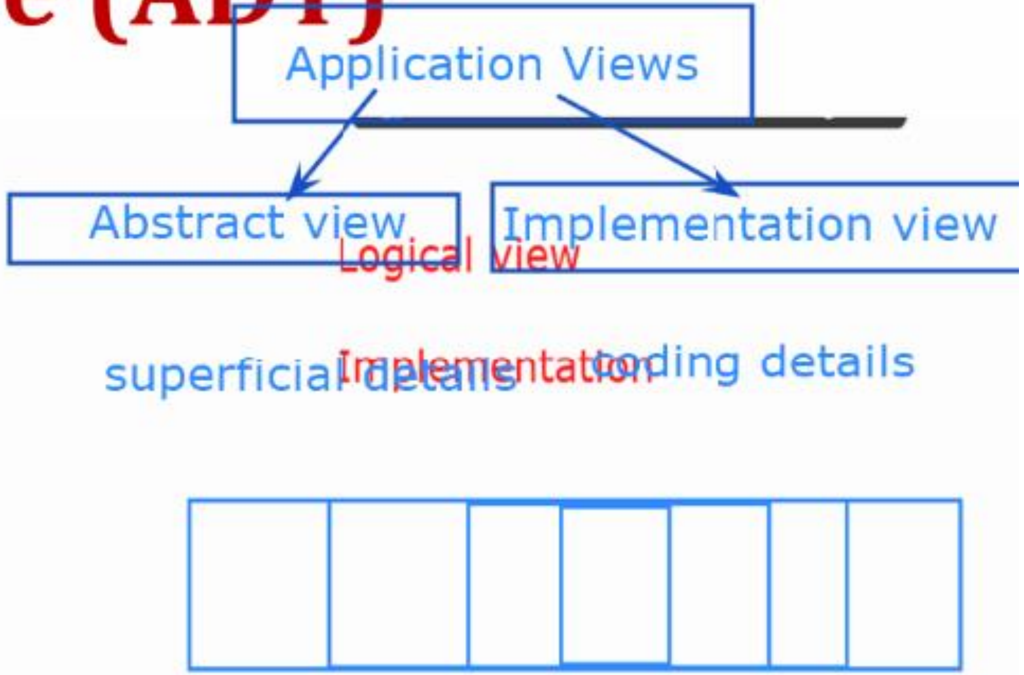
```
class Smartphone{
int ram;
Strin proces;

void call()
void text()
void photo()
}
```

OnePlus 9 5G
(Winter Mist, 12G...

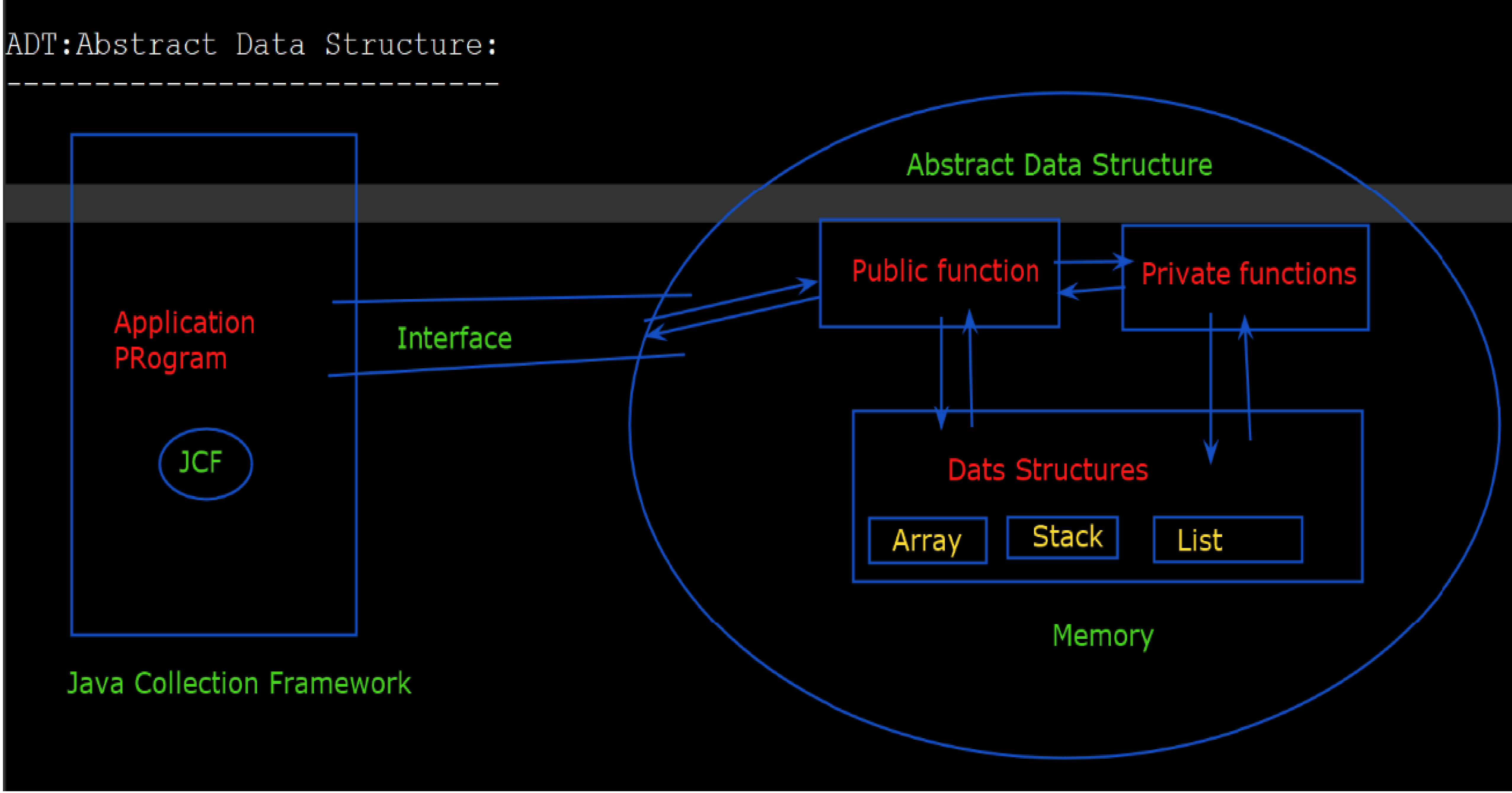
₹54,999

Amazon.in

Free shipping

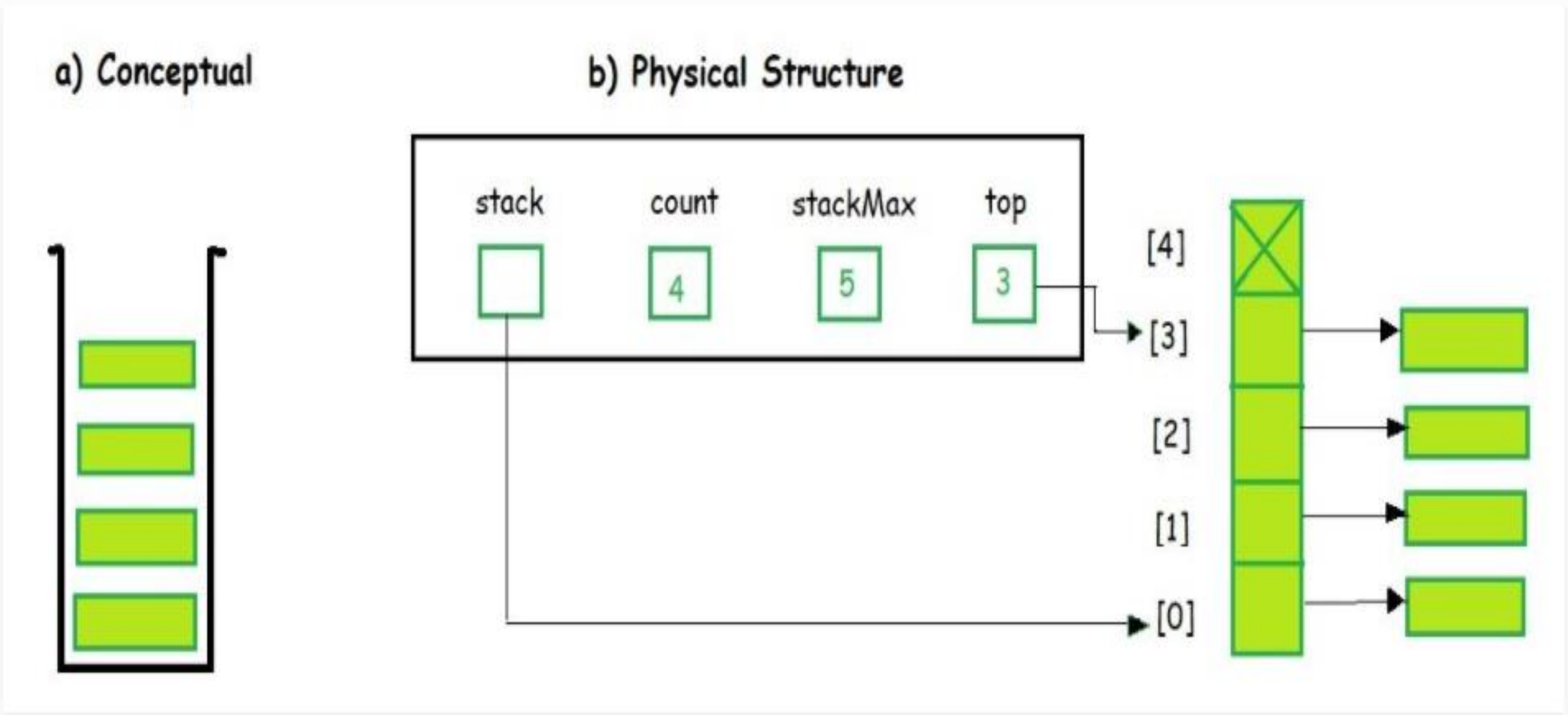


int a[]= (1,2,3,4};

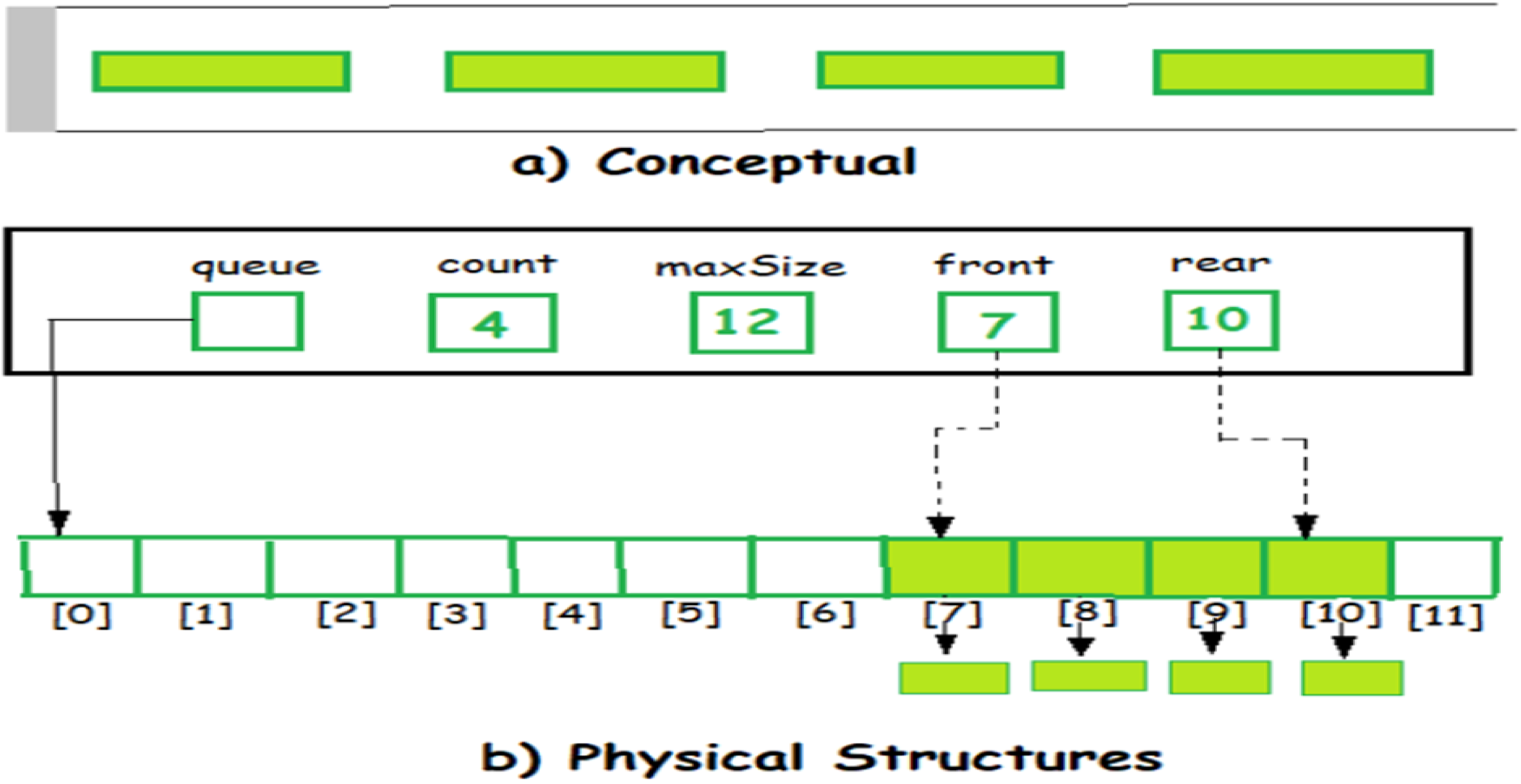


ADT:Abstract Data Structure:

Stack ADT


a) Conceptual
b) Physical Structure
stack
count
4
stackMax
5
top
3
[4]
[3]
[2]
[1]
[0]

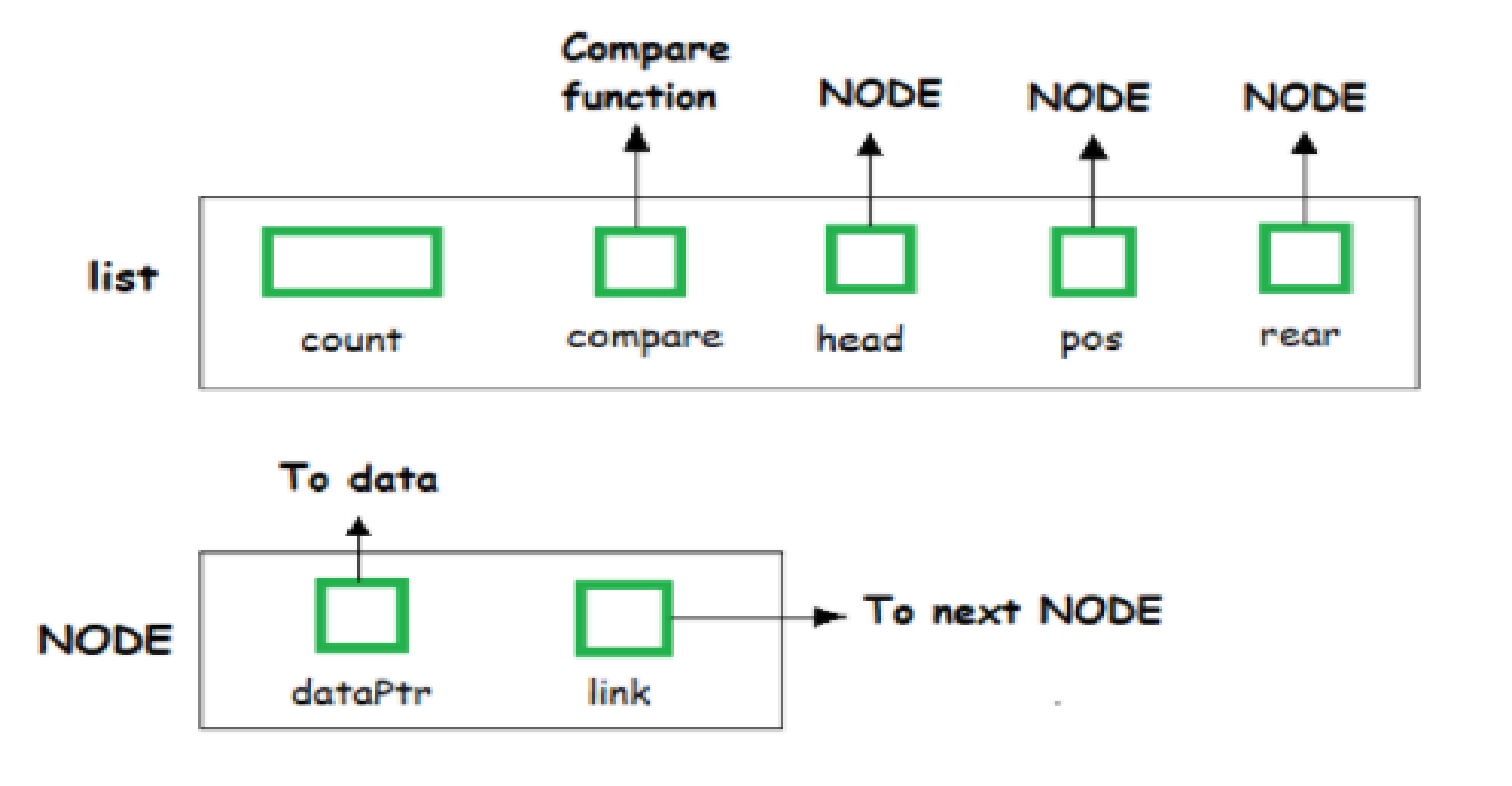
Queue ADT

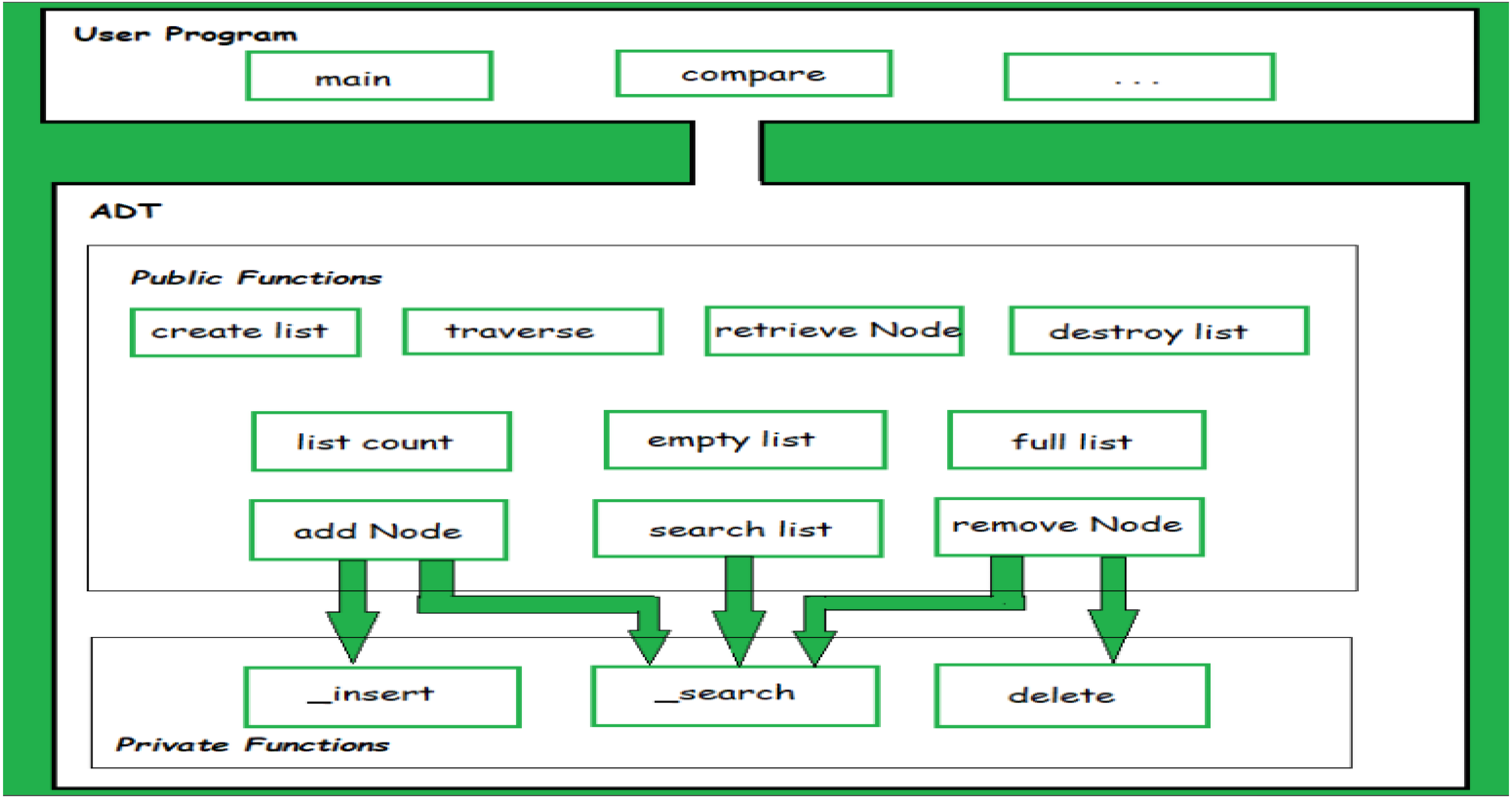


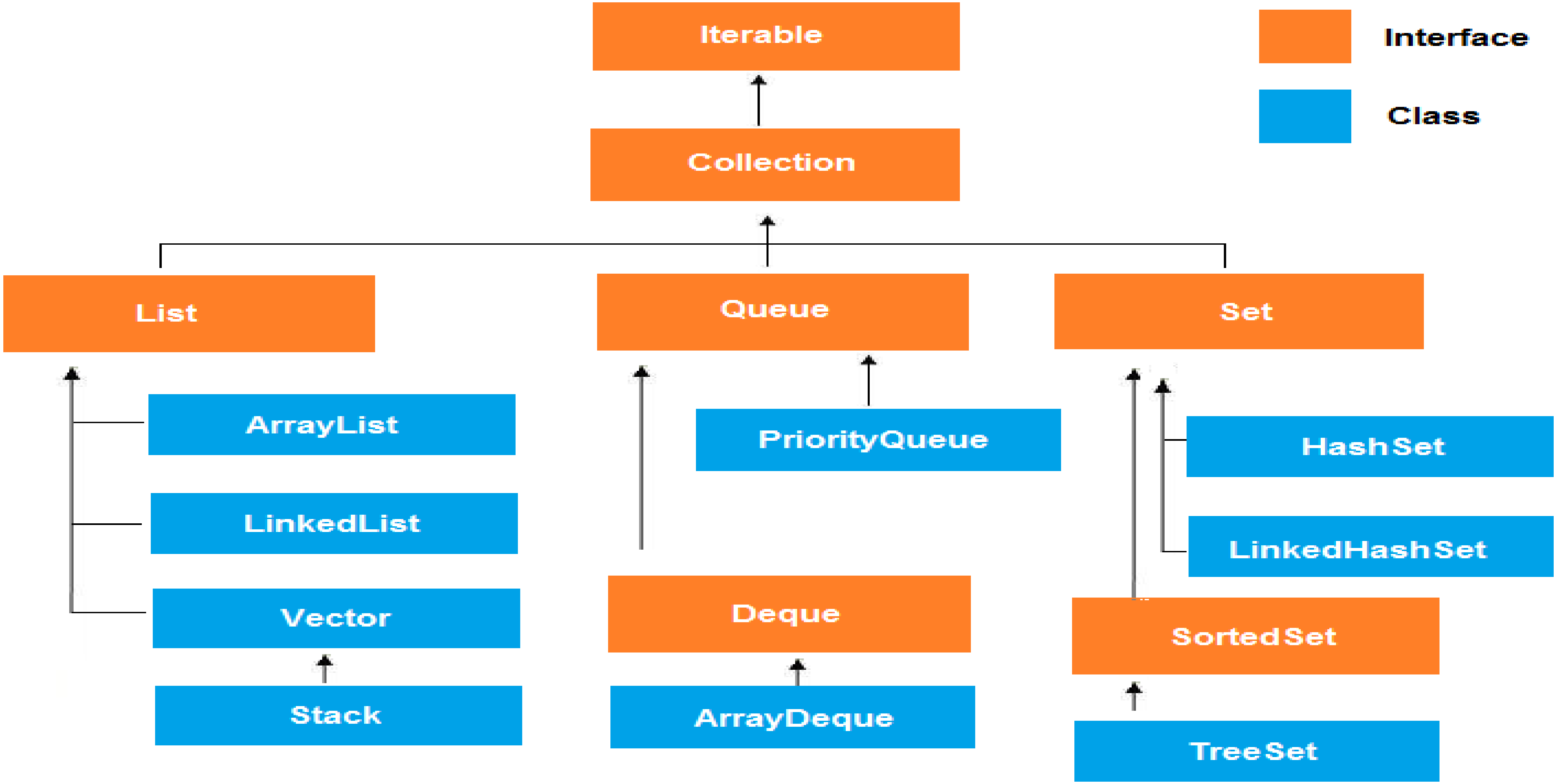
a) Conceptual

b) Physical Structures

List ADT


Compare function
NODE
NODE
NODE
list
count
compare
head
pos
rear
To data
NODE
dataPtr
link
To next NODE

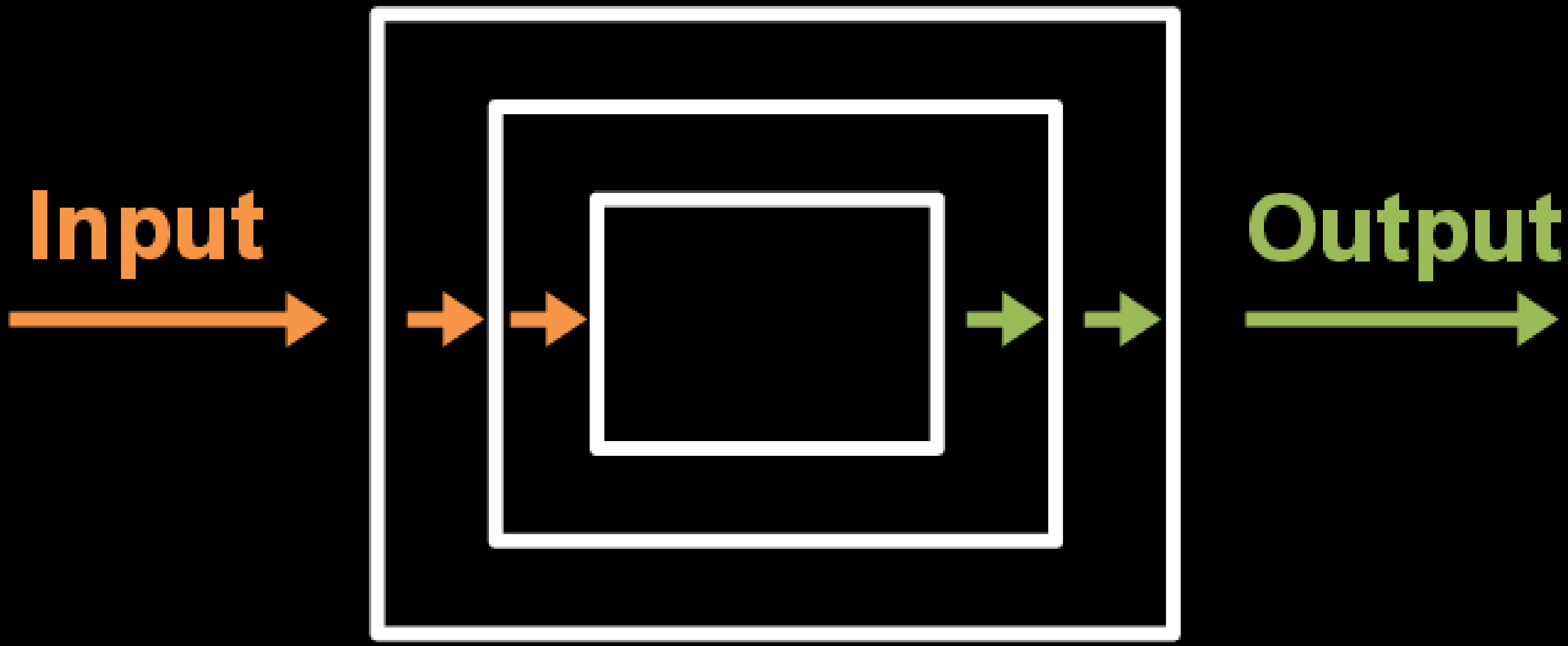

User Program
main
compare
. . .
ADT
Public Functions
create list
traverse
retrieve Node
destroy list
list count
empty list
full list
add Node
search list
remove Node
_insert
_search
delete
Private Functions


Iterable
Interface
Class
Collection
List
Queue
Set
ArrayList
PriorityQueue
HashSet
LinkedList
LinkedHashSet
Vector
Deque
SortedSet
Stack
ArrayDeque
TreeSet

Topics

1. Recursive definitions and Processes
2. Writing Recursive Programs
3. Efficiency in Recursion
4. Towers of Hanoi problem.

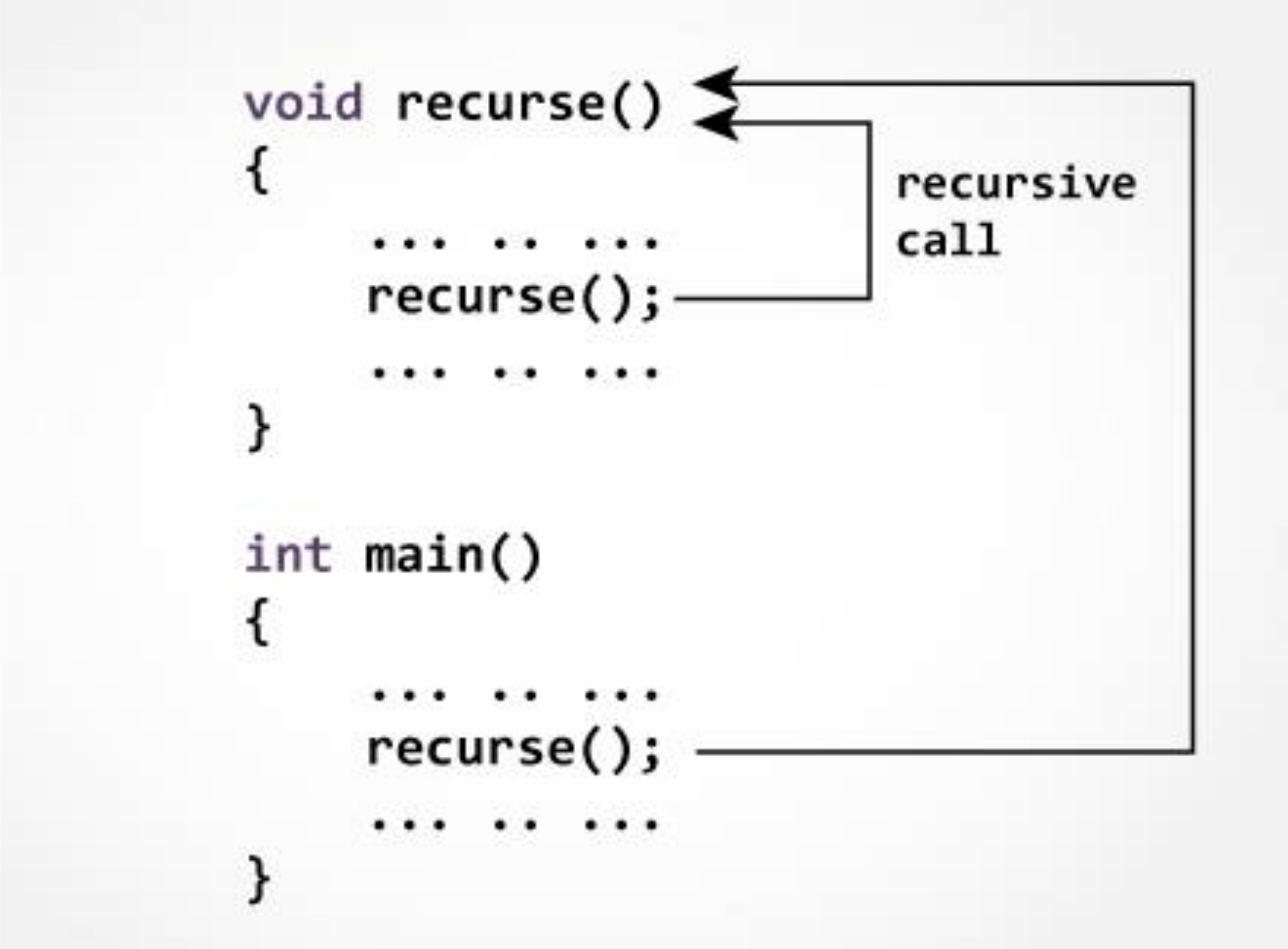
Recursion



Input

Output

How does Recursion works?



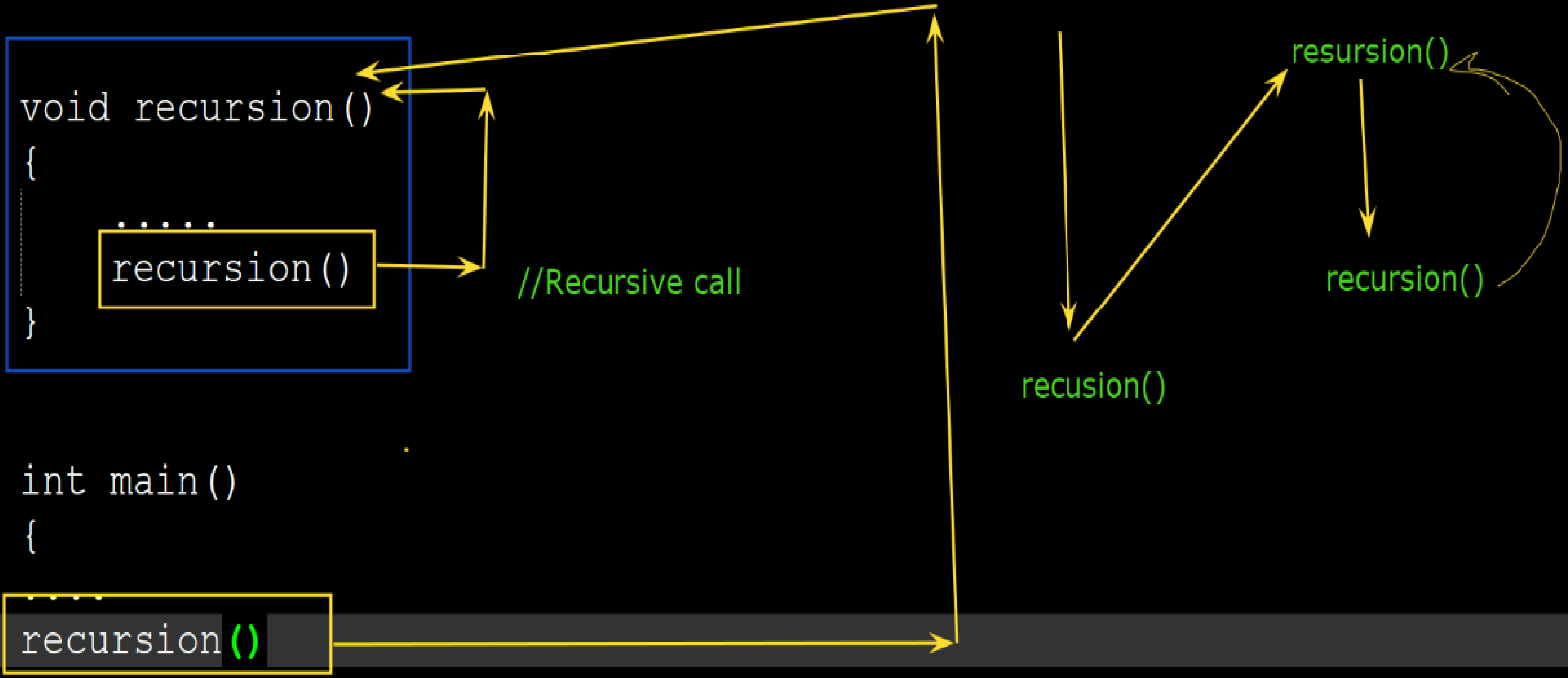
```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

Recursion

- Any function which calls itself directly or indirectly is called Recursion and the corresponding function is called as recursive function.
- A recursive method solves a problem by calling a copy of itself to work on a smaller problem.
- It is important to ensure that the recursion terminates.
- Each time the function call itself with a slightly simple version of the original problem.
- Using recursion, certain problems can be solved quite easily.
- E.g: Tower of Hanoi (TOH), Tree traversals, DFS of Graph etc.,

-Any function which calls itself directly or indirectly is called recursion.

Ex:



```
void recursion()
{
    .....
    recursion()          //Recursive call
}


int main()
{
....
recursion()
}
```

What is the difference between direct and indirect recursion?

A function fun is called **direct recursive** if it calls the same function fun.

A function fun is called **indirect recursive** if it calls another function say fun_new and fun_new calls fun directly or indirectly.

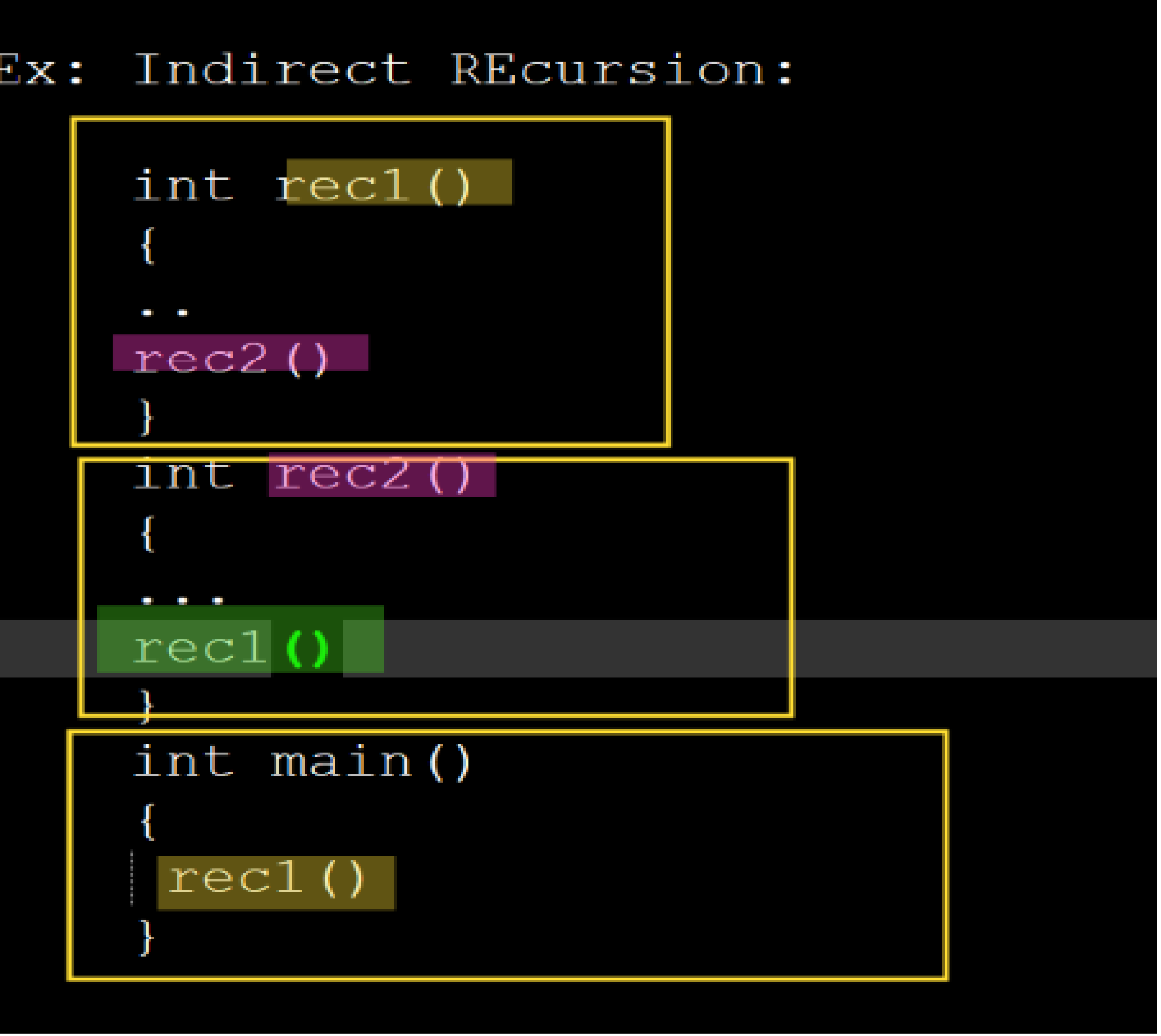
Difference between direct and indirect recursion has been illustrated in Table 1.

- **Direct recursion:**

```
void directRecFun()
{
    // Some code....
    directRecFun();
    // Some code...
}
```

- **Indirect recursion:**

```
void indirectRecFun1()
{
    // Some code...
    indirectRecFun2();
    // Some code...
}

void indirectRecFun2()
{
    // Some code...
    indirectRecFun1();
    // Some code...
}
```



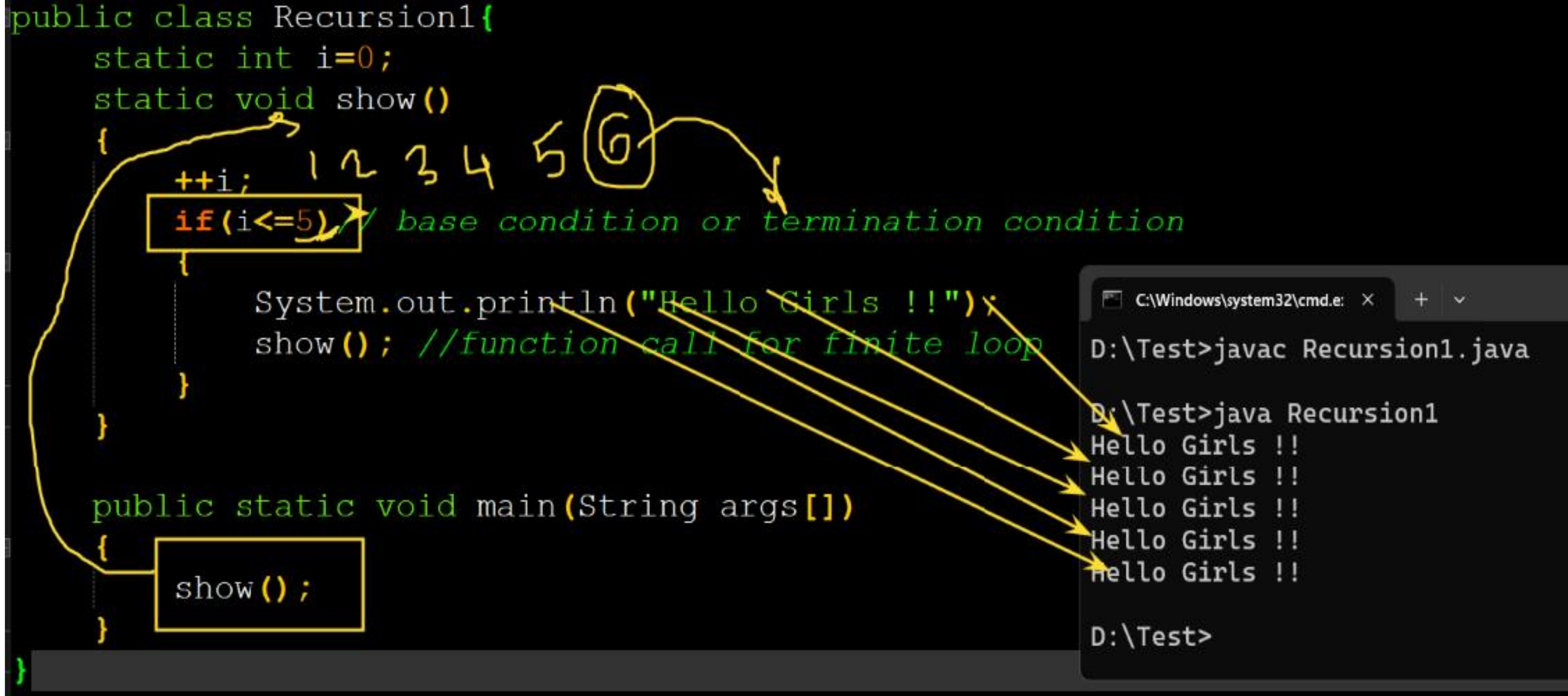
```
Ex: Indirect REcursion:

    int rec1()
    {
    ..
    rec2()
    }
    int rec2()
    {
    ...
    rec1()
    }
    int main()
    {
      rec1()
    }
```

What is base condition in recursion?

- In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

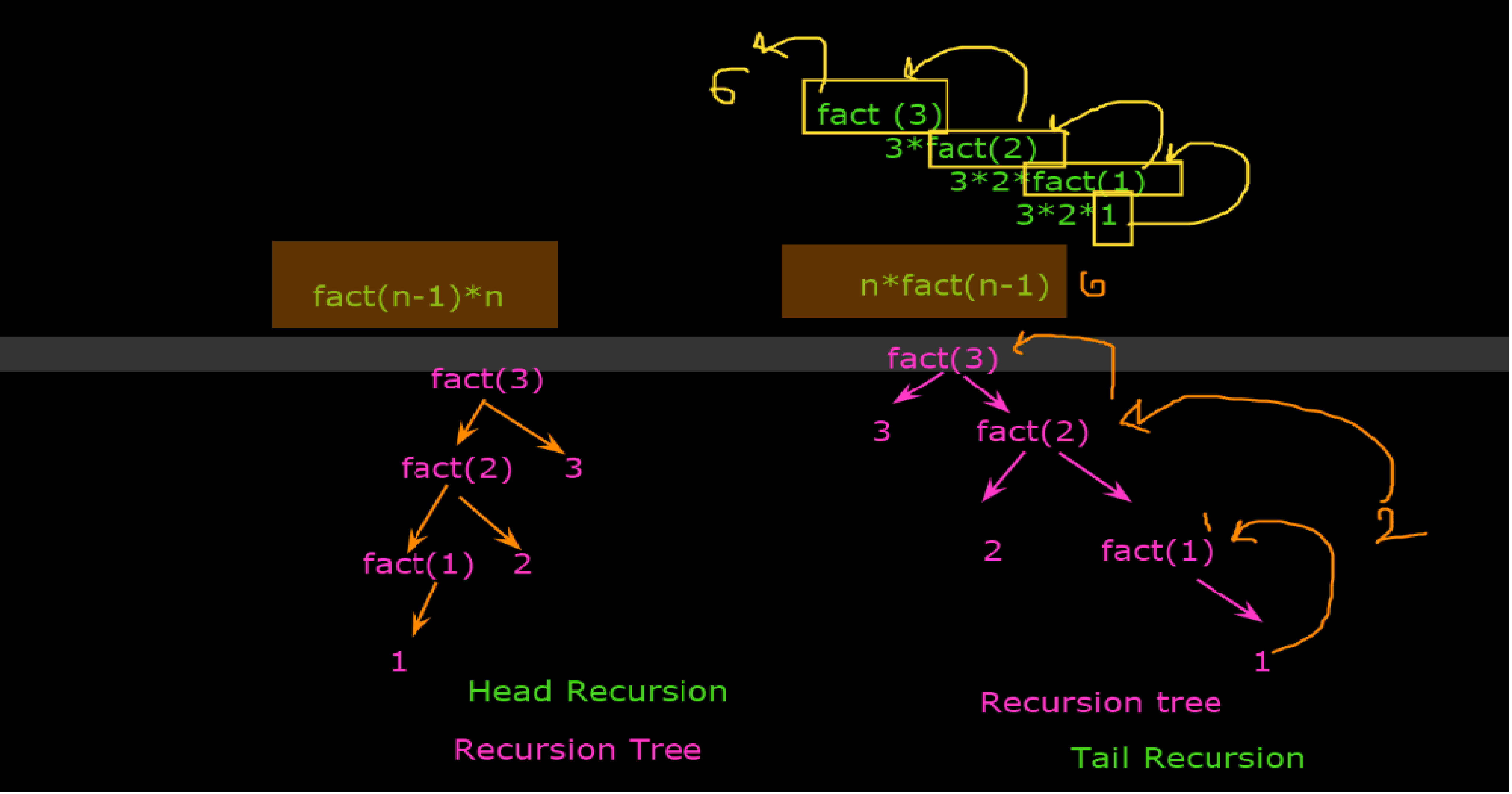
```
int fact(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

- In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.



```
public class Recursion1{
    static int i=0;
    static void show()
    {
        ++i;
        if(i<=5) // base condition or termination condition
        {
            System.out.println("Hello Girls !!");
            show(); //function call for finite loop
        }
    }

    public static void main(String args[])
    {
        show();
    }
}
```

```
D:\Test>javac Recursion1.java

D:\Test>java Recursion1
Hello Girls !!
Hello Girls !!
Hello Girls !!
Hello Girls !!
Hello Girls !!

D:\Test>
```



fact(n-1)*n

n*fact(n-1)

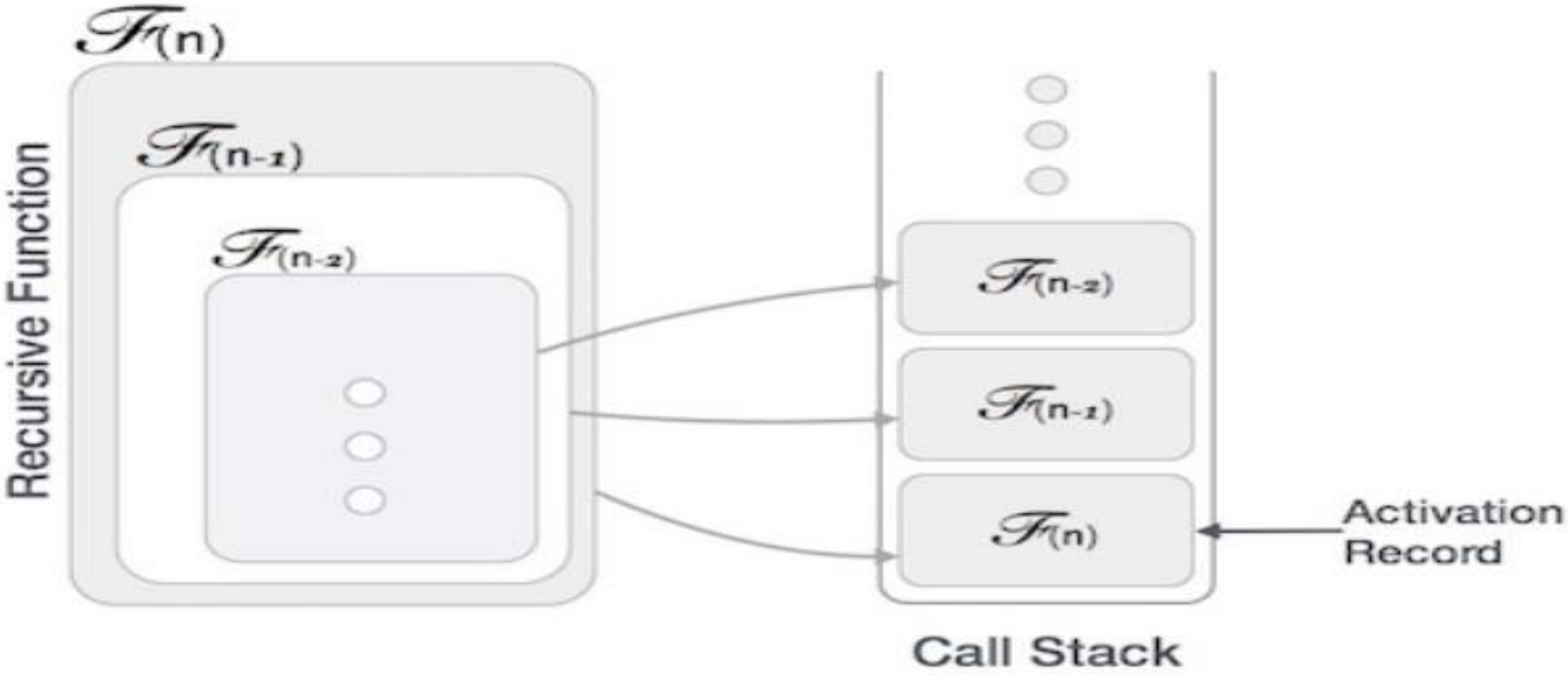
Head Recursion

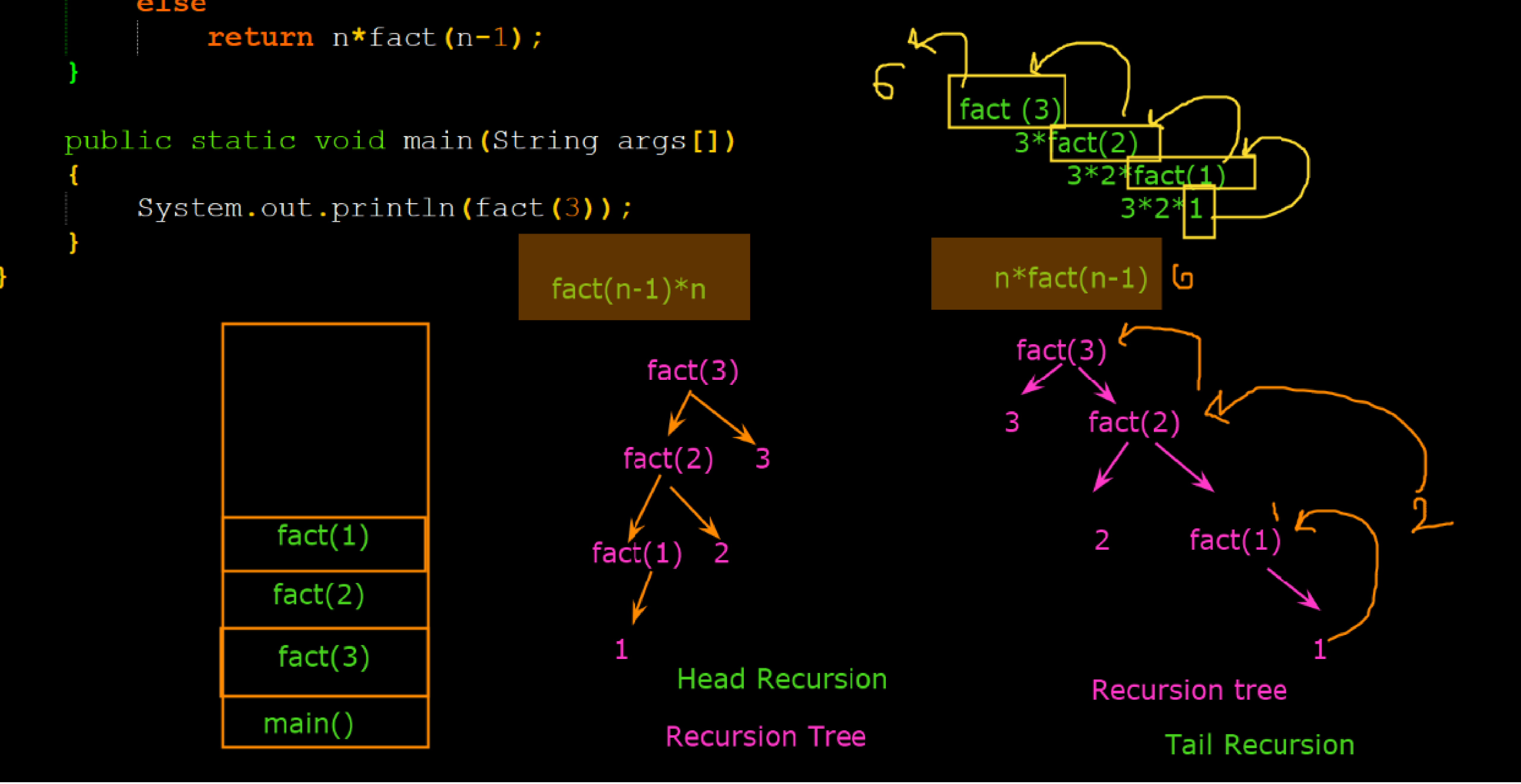
Recursion Tree

Recursion tree

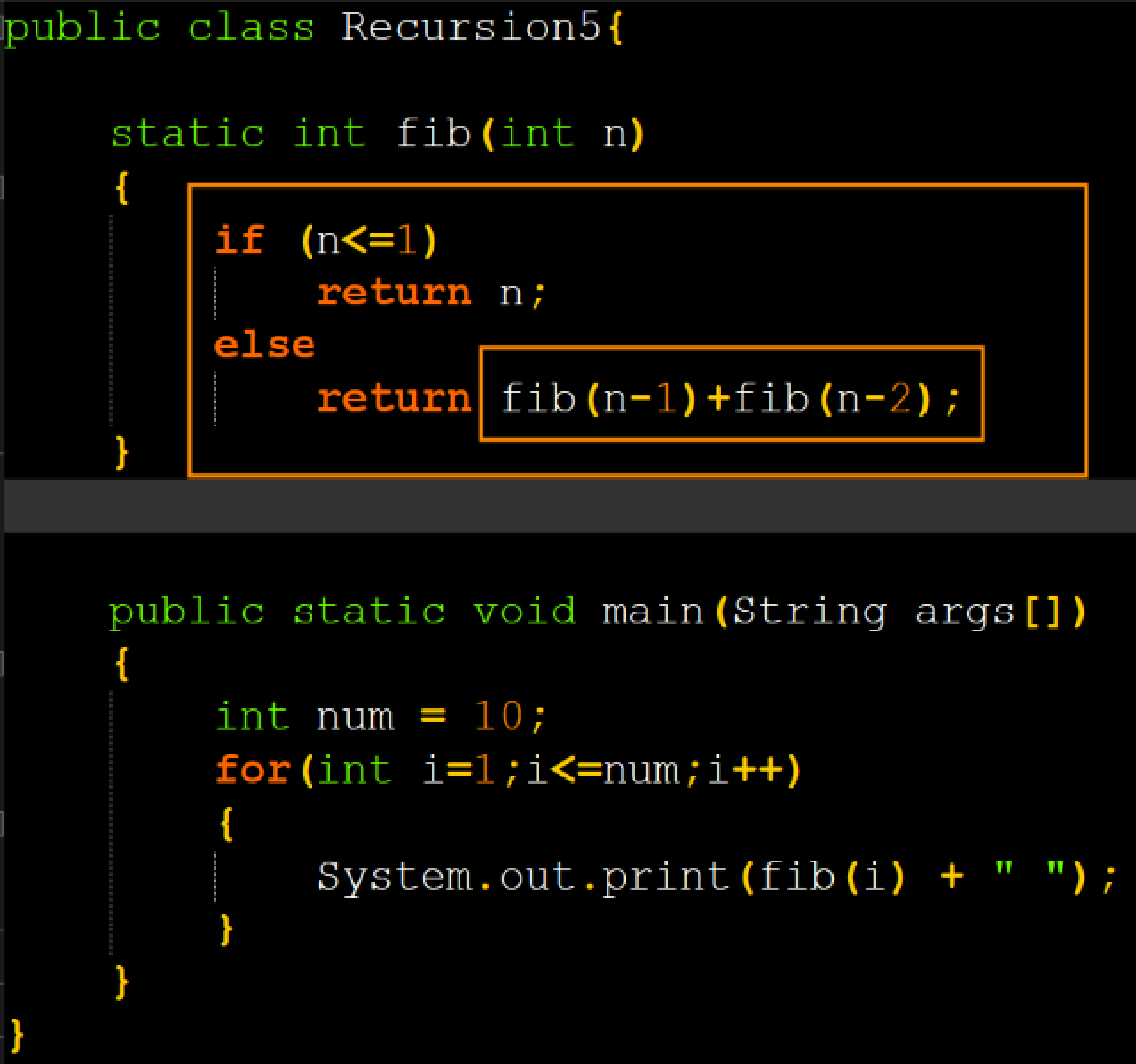
Tail Recursion

How Data Structure Recursive function is implemented?



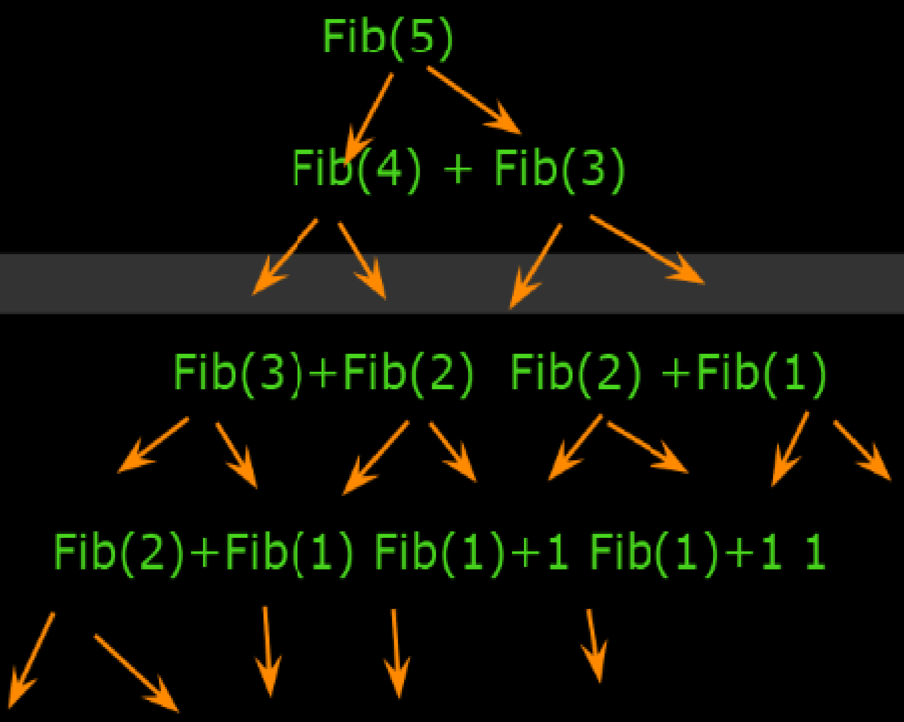


```
        else
            return n*fact(n-1);
    }

    public static void main(String args[])
    {
        System.out.println(fact(3));
    }
}
```



```java
public class Recursion5{

    static int fib(int n)
    {
        if  (n<=1)
            return n;
        else
            return fib(n-1)+fib(n-2);
    }

    public static void main(String args[])
    {
        int num = 10;
        for(int i=1;i<=num;i++)
        {
            System.out.print(fib(i) + " ");
        }
    }
}
```

Fib series: 0 1 1 2 3 5 8



Fib(5)
Fib(4) + Fib(3)
Fib(3)+Fib(2) Fib(2) +Fib(1)
Fib(2)+Fib(1) Fib(1)+1 Fib(1)+1 1

Why Algorithms?

- Fibonacci numbers
  - Compute first N Fibonacci numbers using iteration.
  - … using recursion.
- Write the code.
- Try for N=5, 10, 20, 50, 100
- What do you see? Why does this happen?

Assignment 1

1. Print a series of numbers with recursive Java methods
2. Sum a series of numbers with Java recursion
3. Calculate a factorial in Java with recursion
4. Print the Fibonacci series with Java and recursion
5. A recursive Java palindrome checker

Problem 1

Recursive program to find the Sum of the series 1 – 1/2 + 1/3 – 1/4 … 1/N
Given a positive integer N, the task is to find the sum of the series 1 – (1/2) + (1/3) – (1/4) +…. (1/N) using recursion.

Examples:

Input: N = 3
Output: 0.8333333333333333
Explanation:
1 – (1/2) + (1/3) = 0.8333333333333333

Input: N = 4
Output: 0.5833333333333333
Explanation:
1- (1/2) + (1/3) – (1/4) = 0.5833333333333333

Problem 2

Recursive Program to print multiplication table of a number
Given a number N, the task is to print its multiplication table using recursion.
Examples

Input: N = 5
Output:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Input: N = 8
Output:

```
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

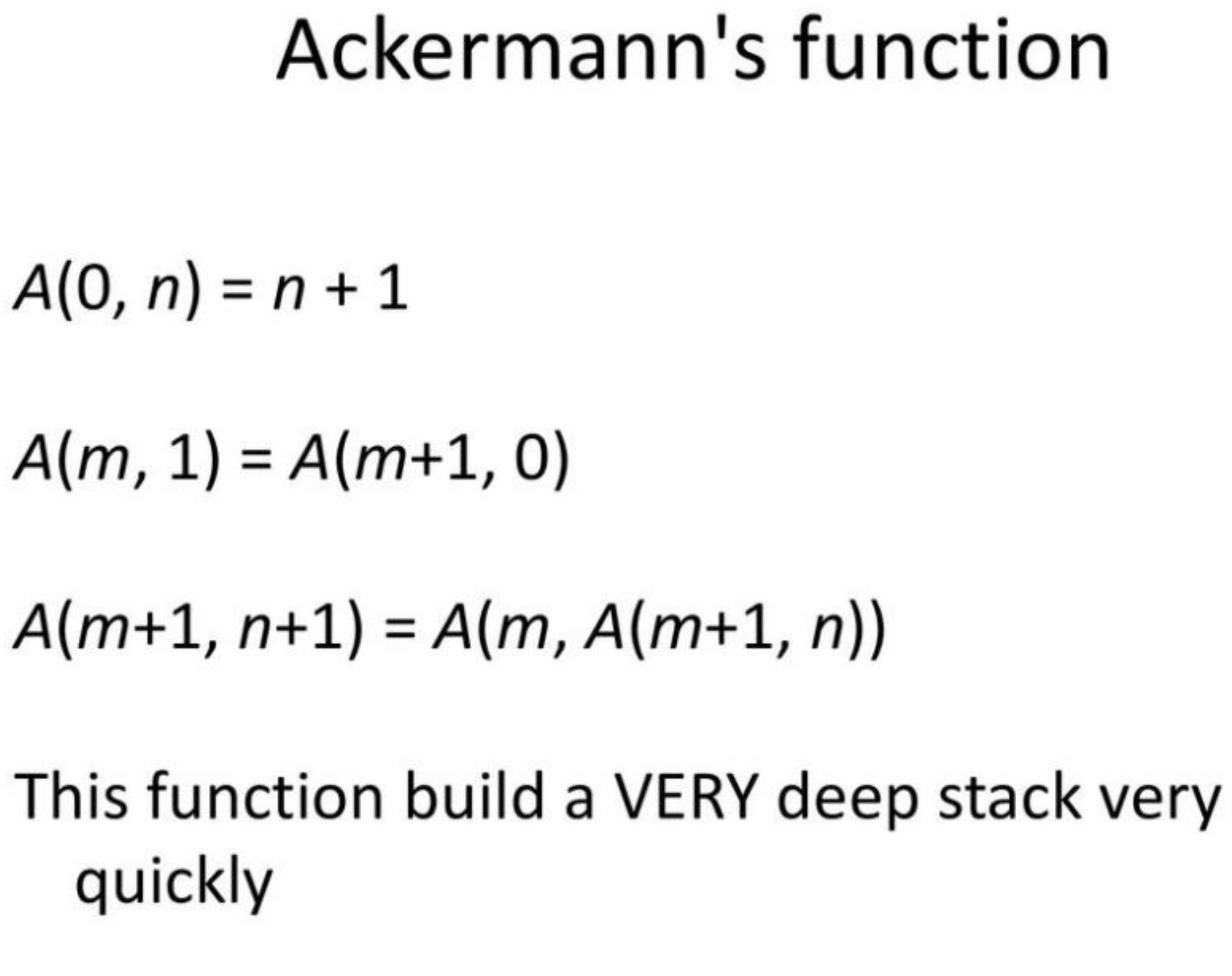
Problem 3

Recursive program to print formula for GCD of n integers

Given a function gcd(a, b) to find GCD (Greatest Common Divisor) of two number. It is also known that GCD of three elements can be found by gcd(a, gcd(b, c)), similarly for four element it can find the GCD by gcd(a, gcd(b, gcd(c, d))). Given a positive integer n. The task is to print the formula to find the GCD of n integer using given gcd() function.
Examples:

Input : n = 3
Output : gcd(int, gcd(int, int))

Input : n = 5
Output : gcd(int, gcd(int, gcd(int, gcd(int, int))))



Ackermann's function

$A(0, n) = n + 1$

$A(m, 1) = A(m+1, 0)$

$A(m+1, n+1) = A(m, A(m+1, n))$

This function build a VERY deep stack very quickly

Day 1 : Questions

----

1. WHAT IS AN ALGORITHM?
2. WHY WE NEED TO DO ALGORITHM ANALYSIS?
3. WHAT ARE THE CRITERIA OF ALGORITHM ANALYSIS?
4.WHAT ARE ASYMPTOTIC NOTATIONS?
5. BRIEFLY EXPLAIN THE APPROACHES TO DEVELOP ALGORITHMS.
6. GIVE SOME EXAMPLES GREEDY ALGORITHMS.
7. WHAT ARE SOME EXAMPLES OF DIVIDE AND CONQUER ALGORITHMS?
8. WHICH PROBLEMS CAN BE SOLVED USING RECURSION?
9. HOW DOES RECURSION WORK IN JAVA?
10. WHAT IS TOWER OF HANOI?
11.WHY IS RECURSION USED?
12.WHAT ARE THE ADVANTAGES O AND DISADVANTAGES OF RECURSION?
13.DIFFERENTIATE BETWEEN RECURSION AND ITERATION.
14.WHAT IS HEAD AND TAIL RECURSION?
15.DISCUSS APPLICATIONS OF RECURSION.