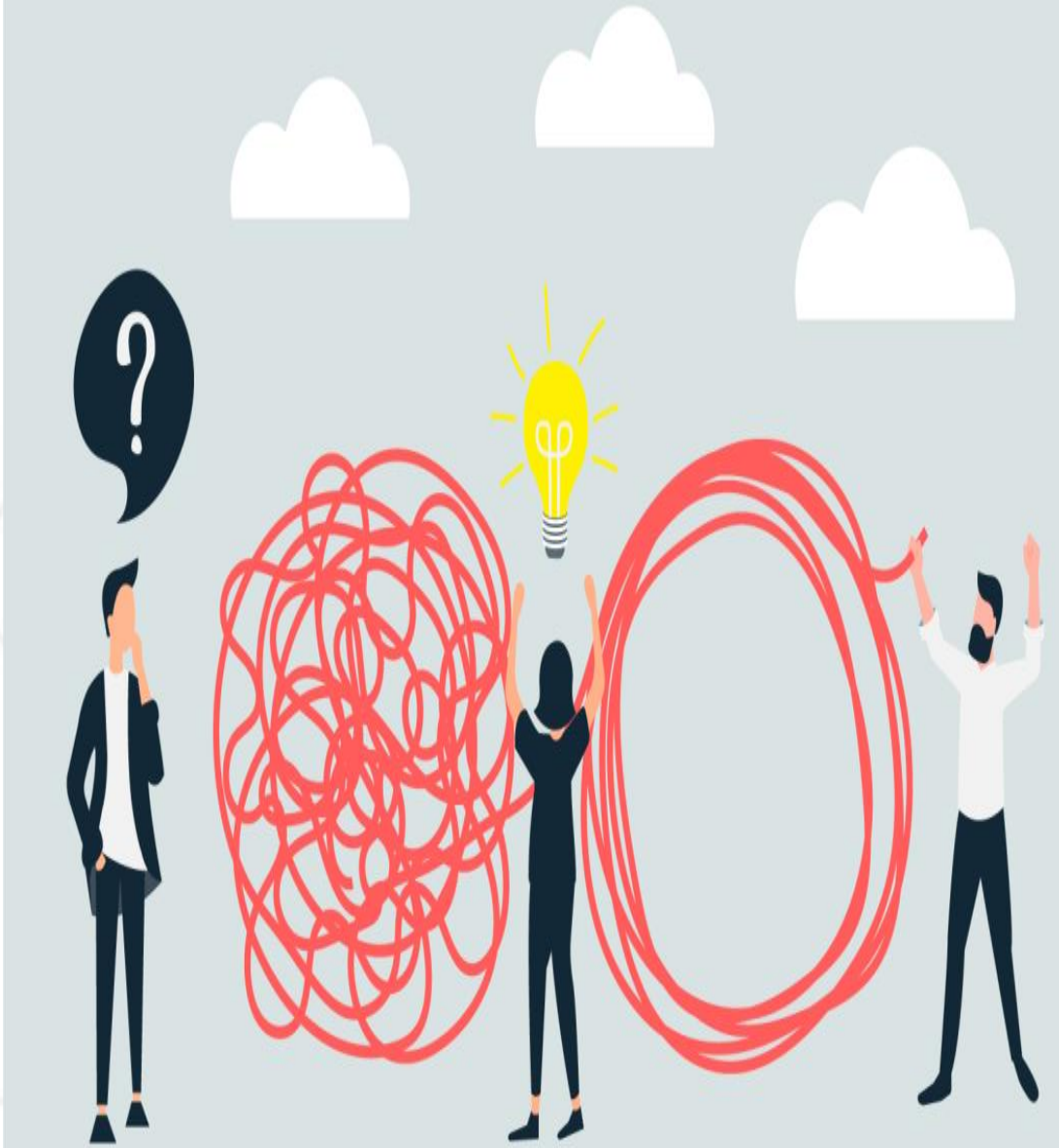
PG-DAC AUGUST 2024

Algorithms and Data Structures

Kiran Waghmare
CDAC Mumbai

Problem Solving

How to Solve Problems

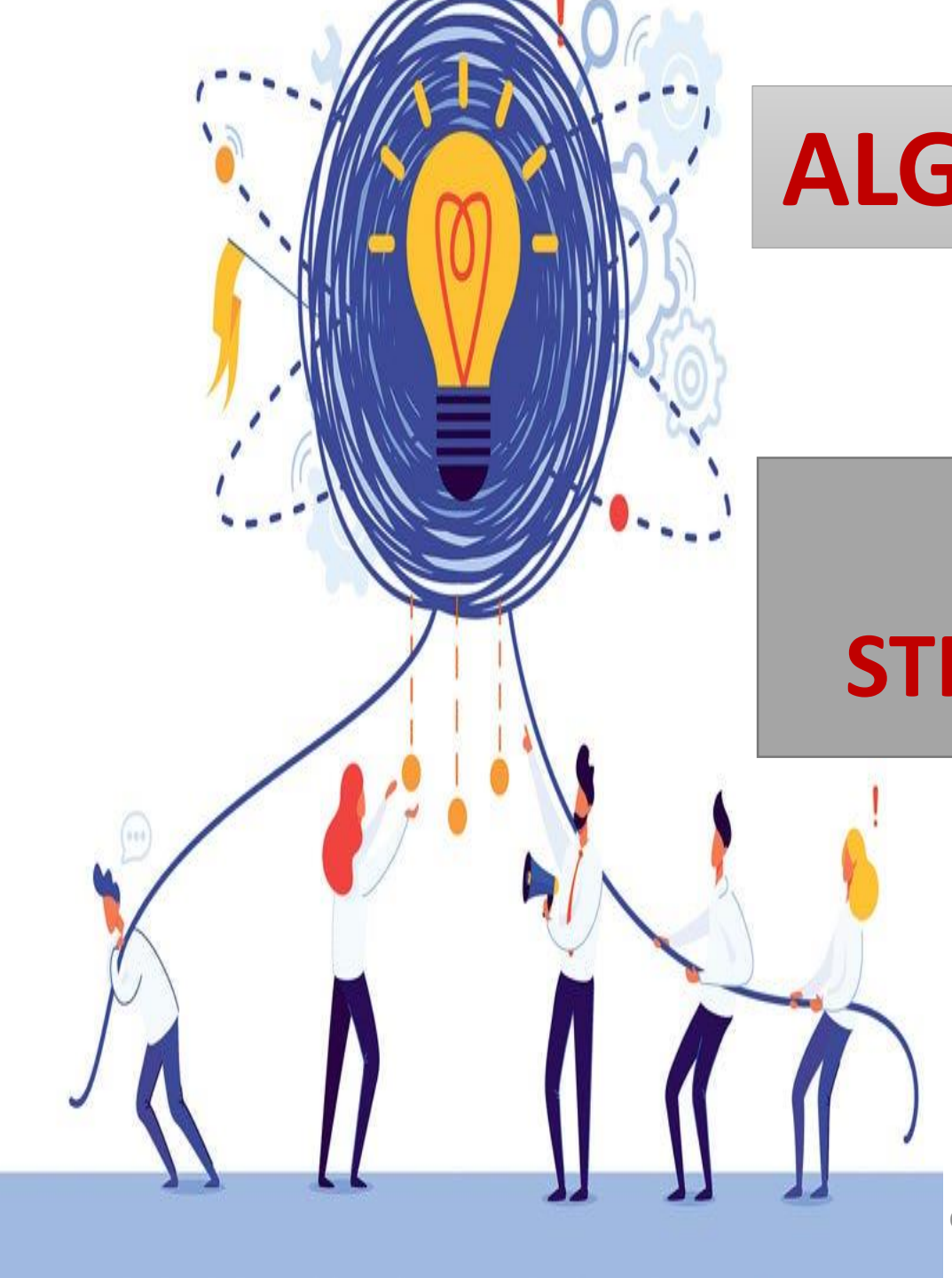


Problem Solving

- Problem solving involves taking on challenges and finding ways to resolve them. It's a process that typically includes both convergent (analytical, logical thinking) and divergent (creative thinking) approaches. Problem solving requires creativity when traditional approaches fail or when new solutions need to be developed.
- **Steps in problem solving:**
  - Identify the Problem:
    - Define the problem clearly. This can include gathering data or asking why a problem exists.
  - Generate Possible Solutions:
    - Use creative thinking to generate many potential solutions without immediately judging them.
  - Evaluate Solutions:
    - Weigh the pros and cons of each potential solution. This is where critical thinking becomes important.
  - Choose the Best Solution:
    - Based on the evaluation, select the most viable solution.
  - Implement the Solution:
    - Put the solution into action, monitoring it to see if it works.
  - Review:
    - Reflect on the effectiveness of the solution and adapt if necessary.

Logical Real life Problem

1. Travelling from Mumbai to Goa
2. Criteria for Marriage
3. ATM money withdrawal
4. Online Money transfer
5. Online shopping

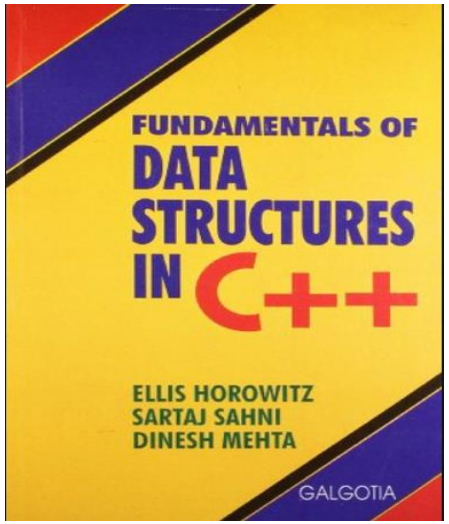


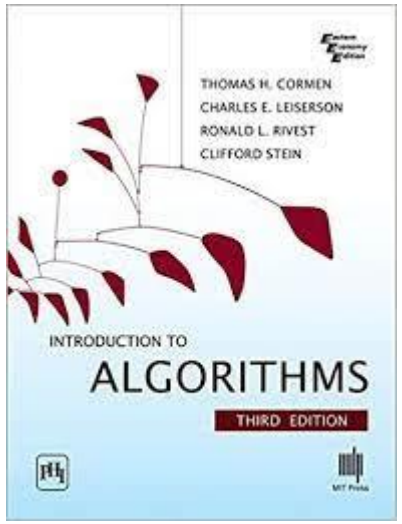
ALGORITHM

DATA STRUCTURE

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





Agenda

- **Problem Solving & Computational Thinking**
- **Algorithm & Data Structure**
  - OODesign: ADTs
- **Recursion**
  - Base condition
  - Direct & indirect recursion
  - Memory allocation
  - Pros and Cons
  - Complexity analysis

Computational Thinking : Researcher

- Niklaus Wirth

Linus Torvalds

Martin Karplus, Michael Levitt, and Arieh Warshel

Algorithms are Everywhere

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

Intelligent Computational Systems

"Big data" will allow us to put the "smarts" into everything …

- Smart homes
- Smart cars
- Smart health
- Smart robots
- Smart crowds and human-computer systems
- Smart interaction (virtual and augmented reality)
- Smart discovery (exploiting the data deluge)

Why Data Structures?

- Data is just the raw material for information, analytics, business intelligence, advertising, etc
- Computational efficient ways of analyzing, storing, searching, modeling data
- For the purpose of this course, need for efficient data structures comes down to:
  - Linear search does not scale for querying large databases
  - N^2 processing or N^2 storage infeasible
  - Smart data structures offer an intelligent tradeoff:
  - Perform near-linear preprocessing so that queries can be answered in much better than linear time

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.

Definition

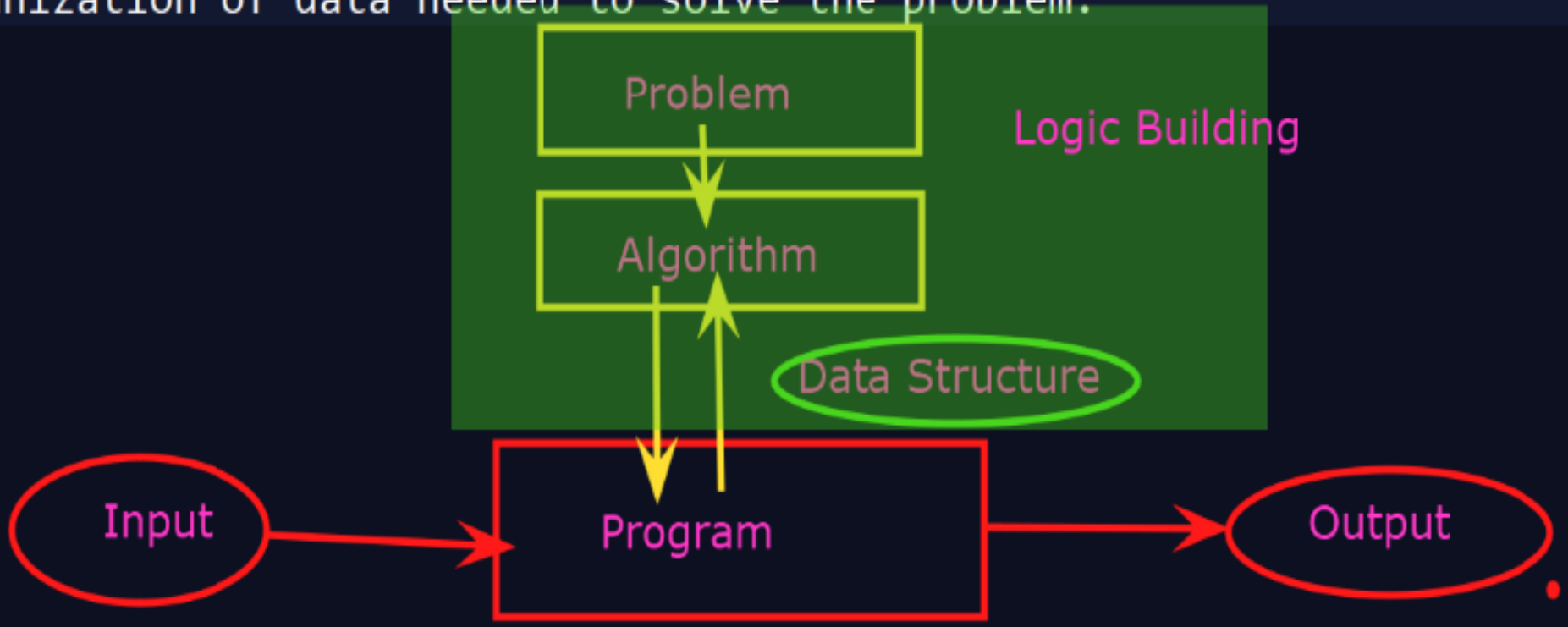
- **Data:**
  - Collection of Raw facts.
- **Algorithm:**
  - Outline, the essence of a computational procedure, step-by-step instructions.
- **Program:**
  - An implementation of an algorithm in some programming language
- **Data Structure:**
  - Organization of data needed to solve the problem.
  - The programmatic way of storing data so that data can be used efficiently

Data: Collection of raw facts.

Algorithms:The essence of a computational procedure, in step by step manure.

Program:An implementation of an algorithm in some programming language.

Data structure: Organization of data needed to solve the problem.



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

- An ***algorithm*** is a sequence of unambiguous instructions/operations for solving a problem, for obtaining a required output for any legitimate input in a finite amount of time.

What is an Algorithm?

- An algorithm is a process or a **set of rules required to perform calculations** or some other problem-solving operations especially by a computer.
- The formal definition of an algorithm is that it contains the **finite set of instructions** which are being carried in a specific order to perform the specific task.
- It is not the complete program or code; it is just a **solution (logic) of a problem,** which can be represented either as an informal description using a Flowchart or Pseudocode.

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

Analysis of Algorithms

- An algorithm is said to be efficient and fast, if **it takes less time to execute and consumes less memory space**.
- The performance of an algorithm is measured on the basis of following properties :

1.Time Complexity

2.Space Complexity

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**

Algorithm Analysis:

1.Time Complexity:

-Measures the time of an algorithm takes as a function of the input size.

2.Space complexity:

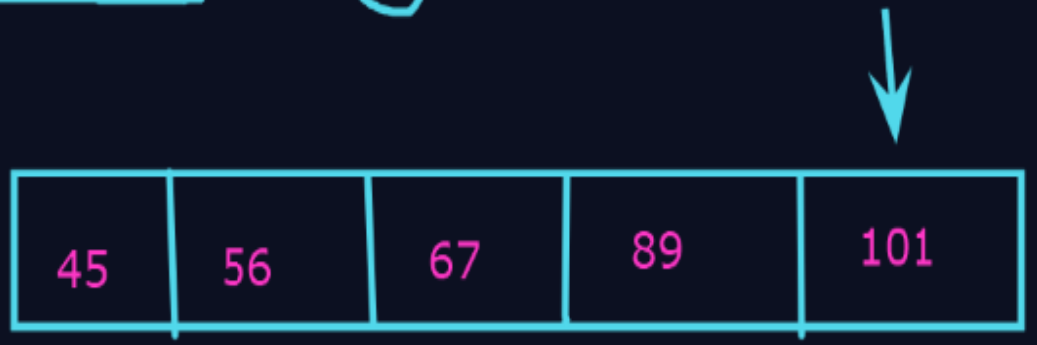
-Measures of the amount of memory consume by an algorithm uses as a function of the input size.

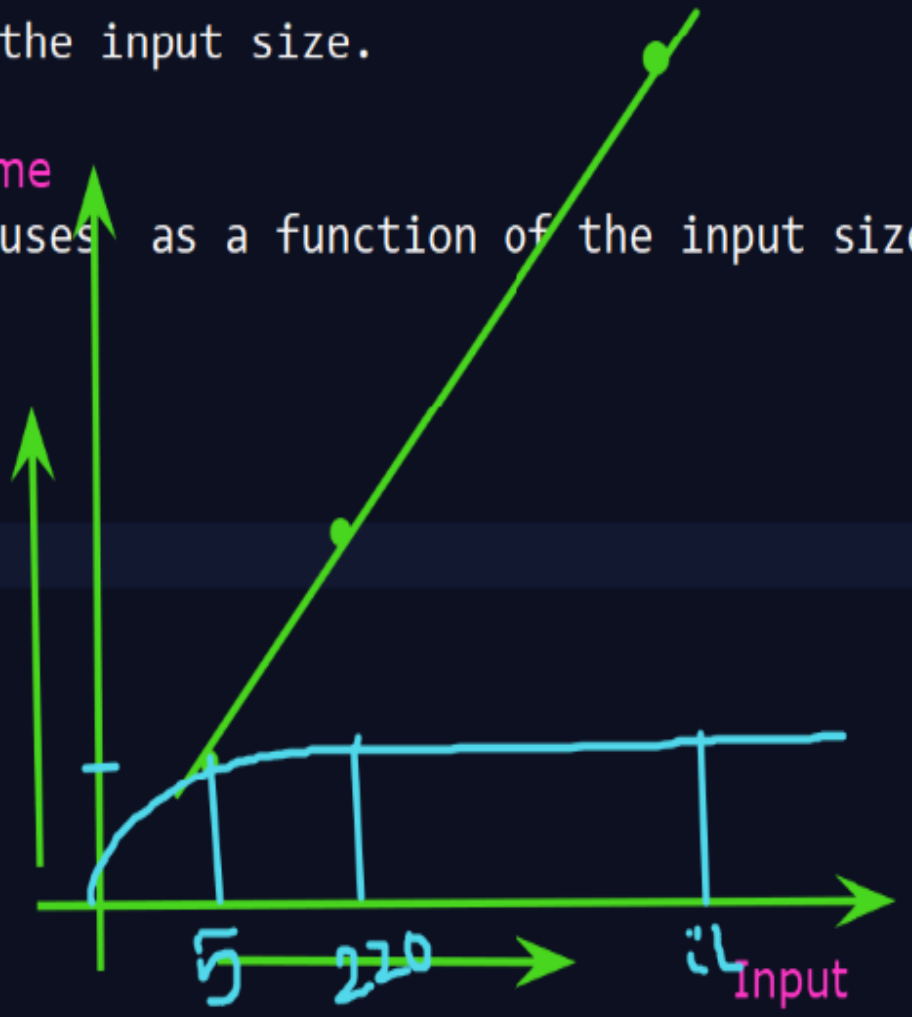
3. Big O Notation:

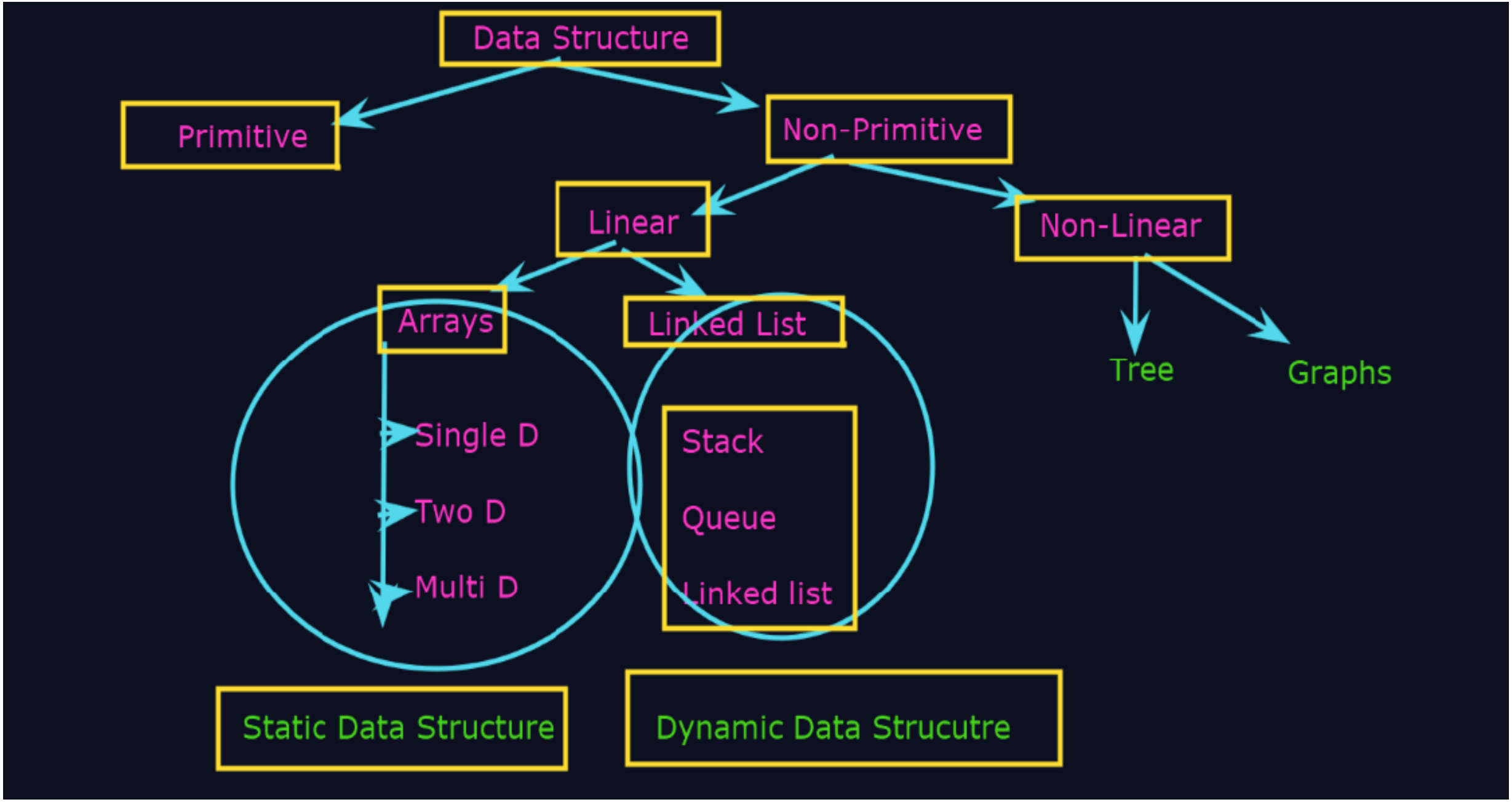
-Best case

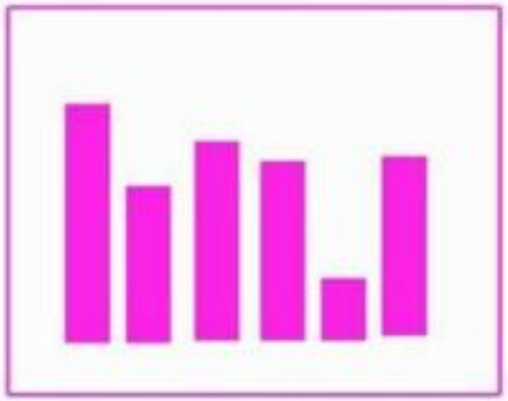
-Average case

-Worst case

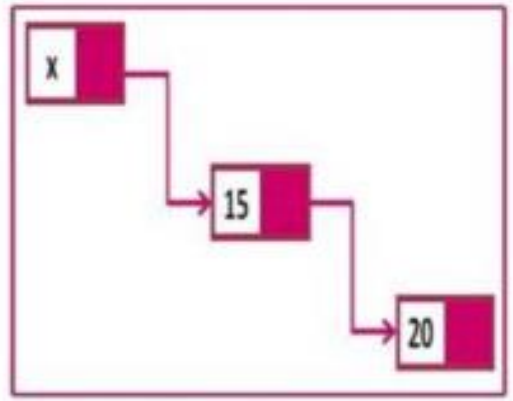




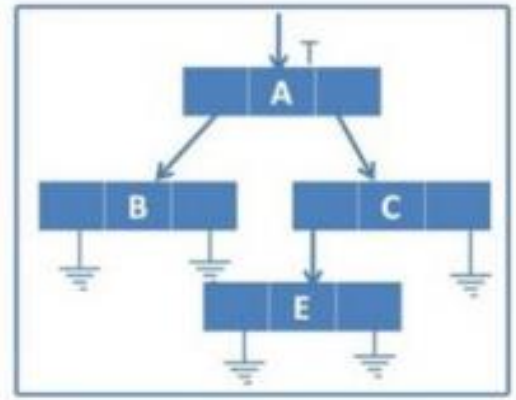

Data Structure
Primitive
Non-Primitive
Linear
Non-Linear
Arrays
Linked List
Tree
Graphs
Single D
Two D
Multi D
Stack
Queue
Linked list
Static Data Structure
Dynamic Data Strucutre



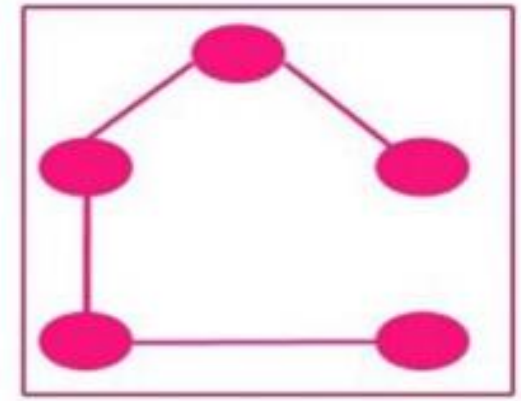
Sorting



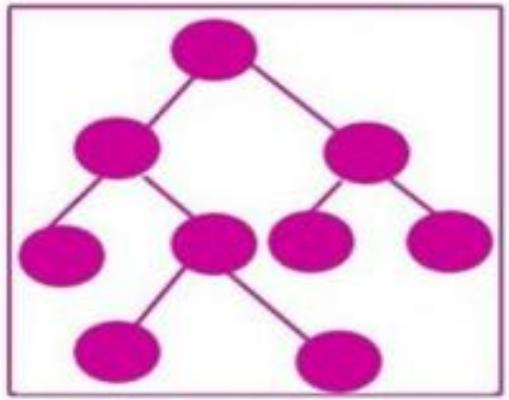
Link list



list

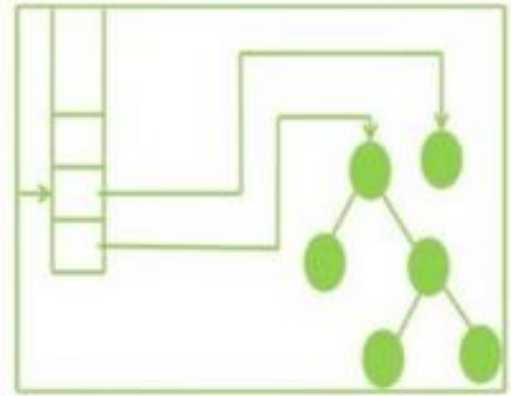


spanning tree

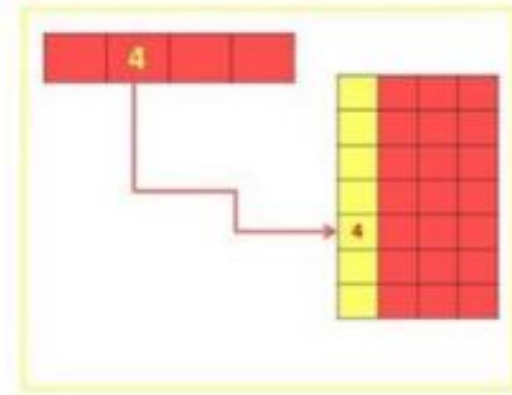


Tree

Graph



Stack

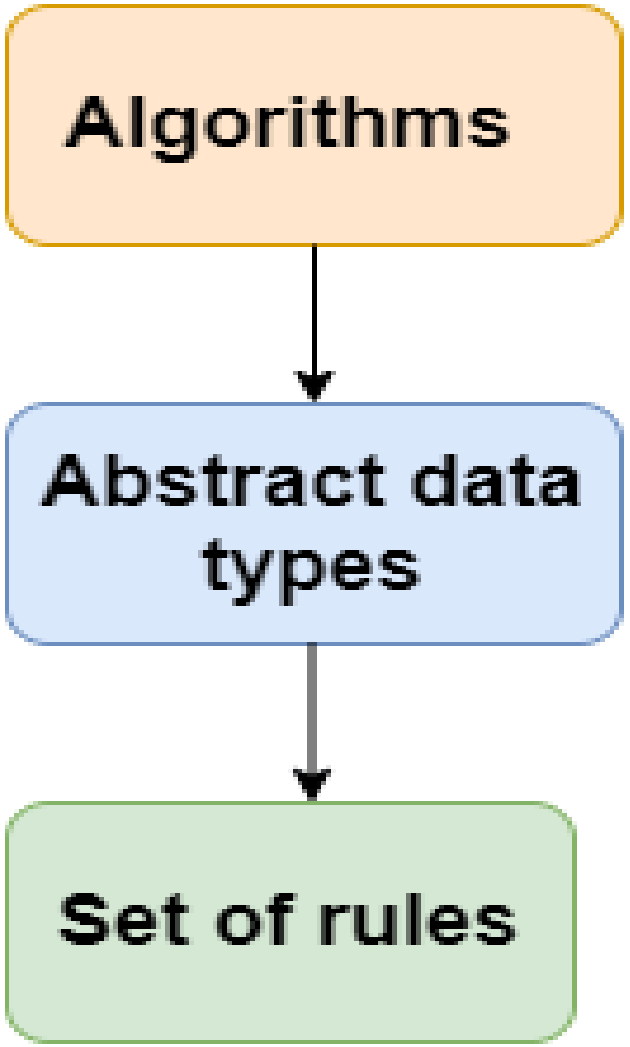


Hashing

By...navinkumardhoprephotography.com

Abstract Data Type (ADT)

Abstract Data Type (ADT)



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

₹54,999

Amazon.in

Free shipping

Logical view

Implementation

Audio Video Participants Chat Share Pause Annotate Remote control Show meeting More

You are screen sharing EN Stop share

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

₹54,999

Amazon.in

Logical view/
Abstract view

Implementation view:
class smartphone{

int ramsize;
string processorname
int camerapixelsize

void call();
void exit();
void photo();
}

Algorithms

↓

Abstract Data types

↓

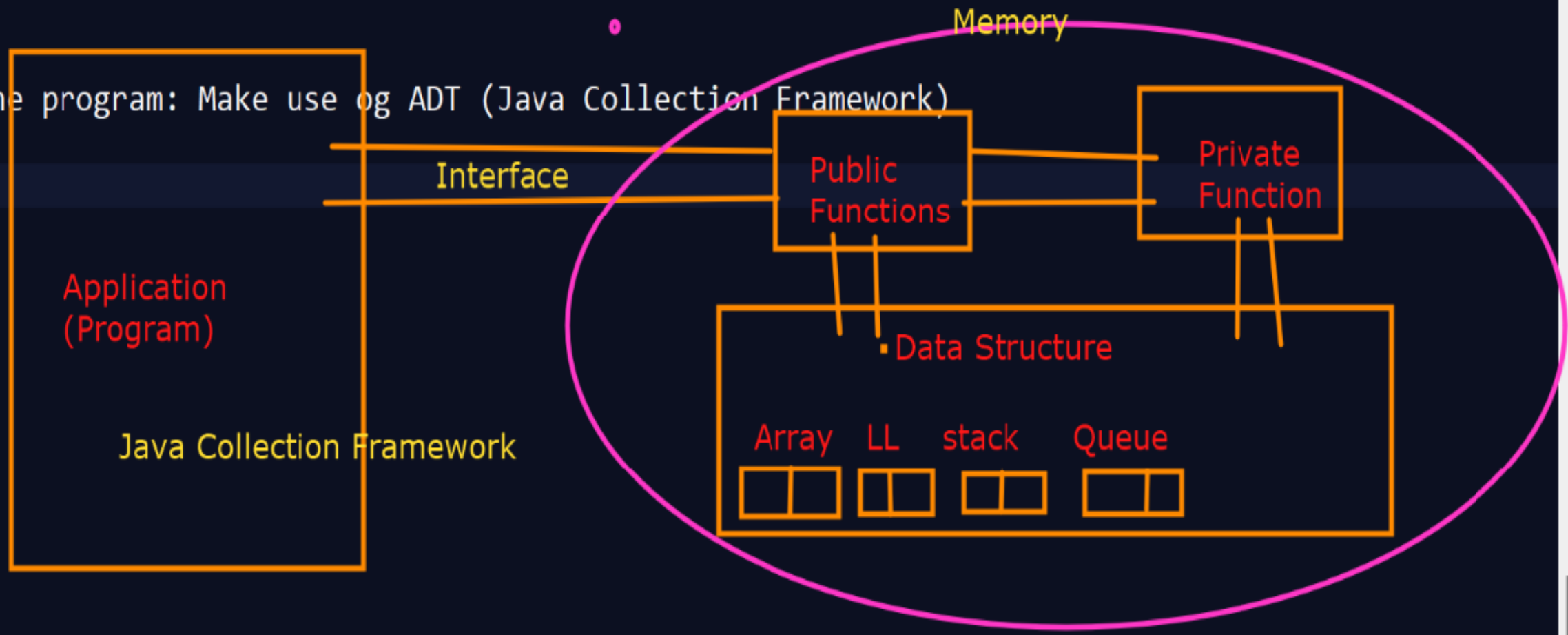
Set of rules

ADT

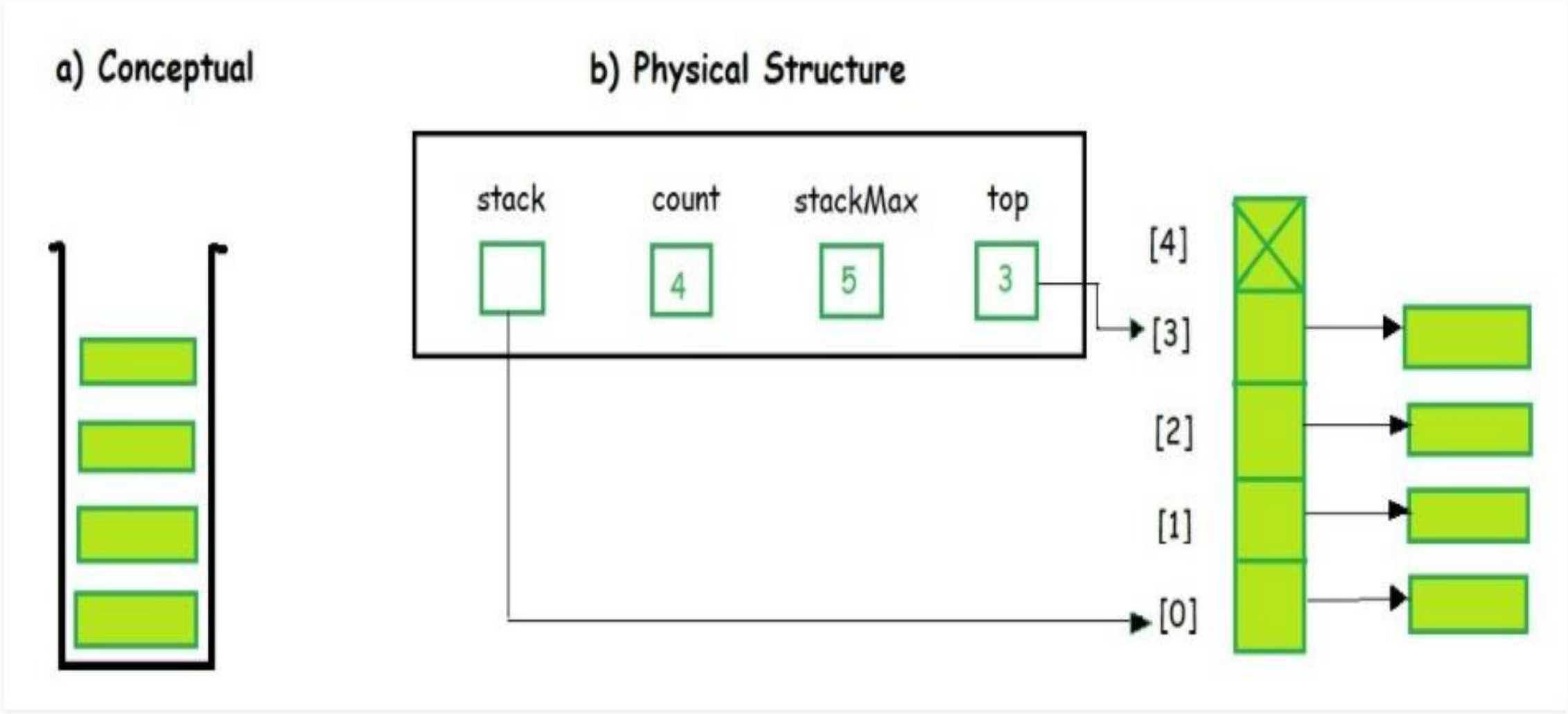
- Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- It is called “abstract” because it gives an implementation-independent view.
- The process of providing only the essentials and hiding the details is known as abstraction.
- All primitive data types support basic operations,+,-,*,/ etc

-ADT is a type or class for objects whose behaviour is defined by the set of rules.
-The definition of ADT only mentiones what operations are to be performed but not how to perform or how to implement.
-It does not specify how data will be organized in memory and what algorithms are used for implementation. It is called 'abstract' because it give an implementation-independent view.

Optimize the program: Make use og ADT (Java Collection Framework)

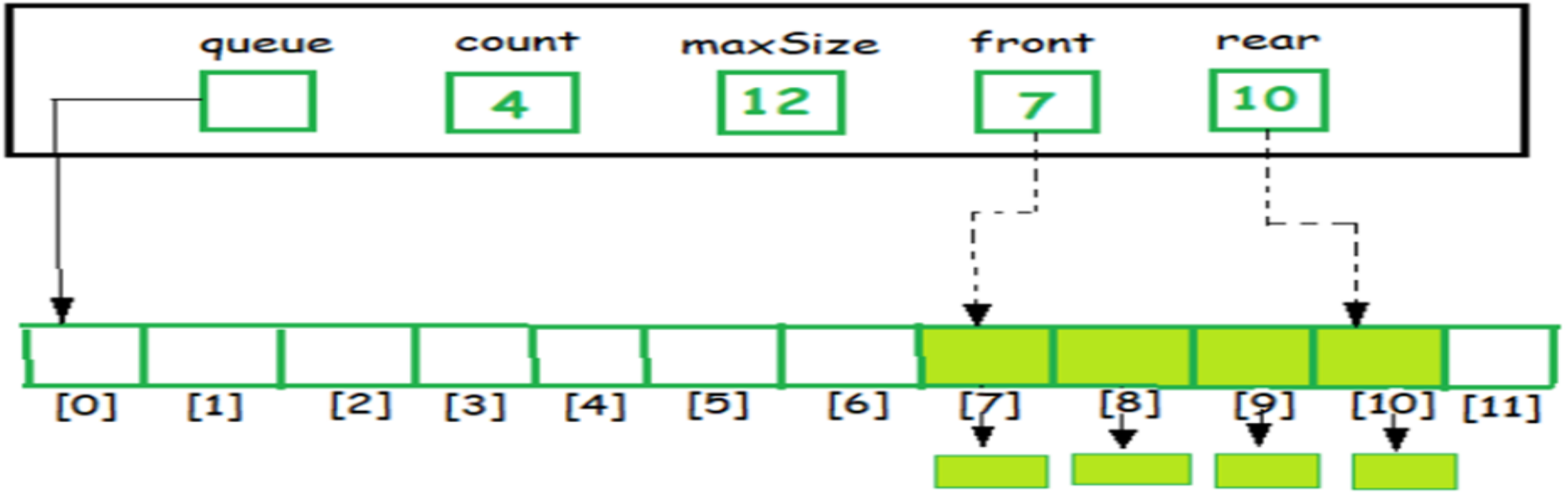


Stack ADT


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

Queue ADT

a) Conceptual

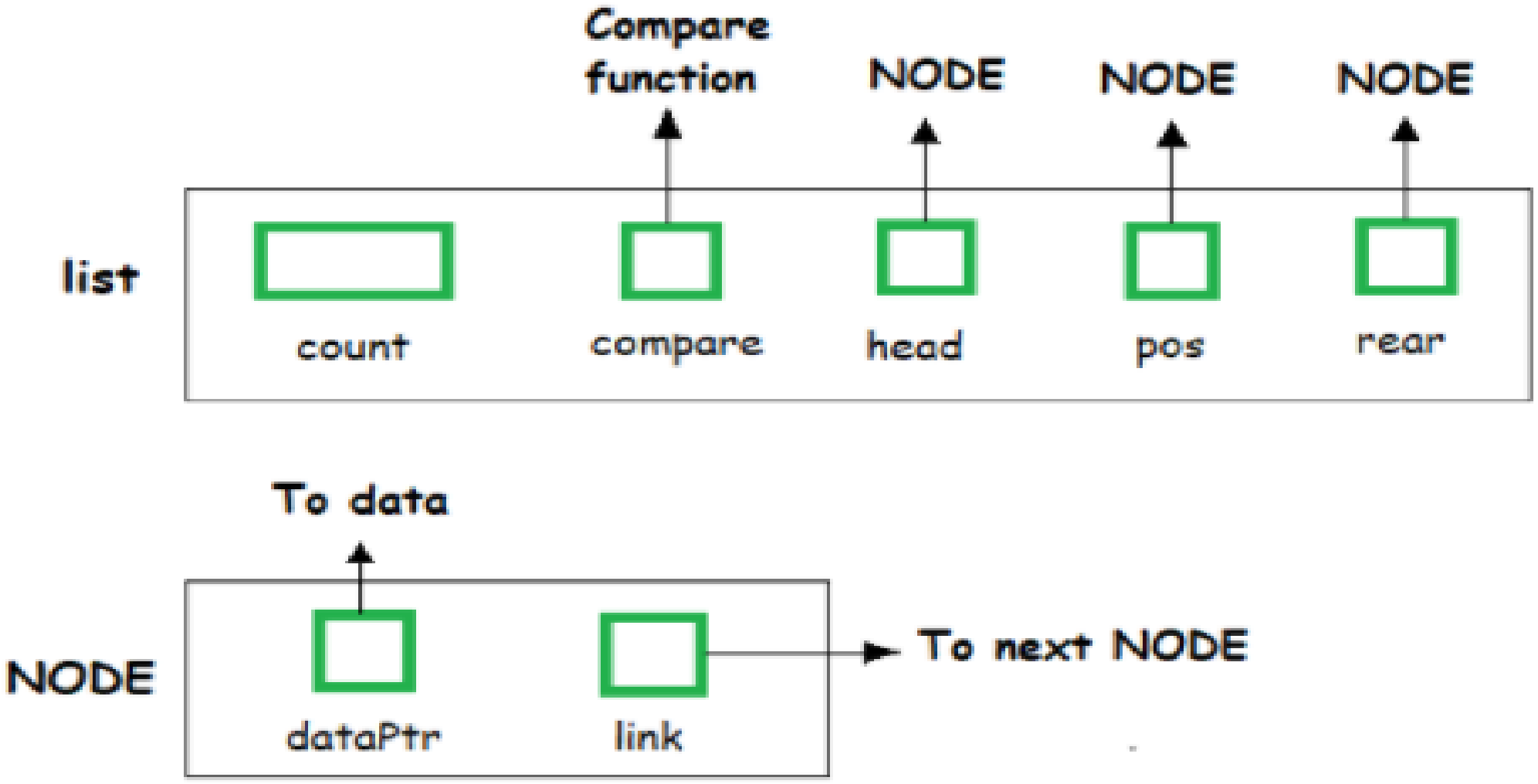


b) Physical Structures

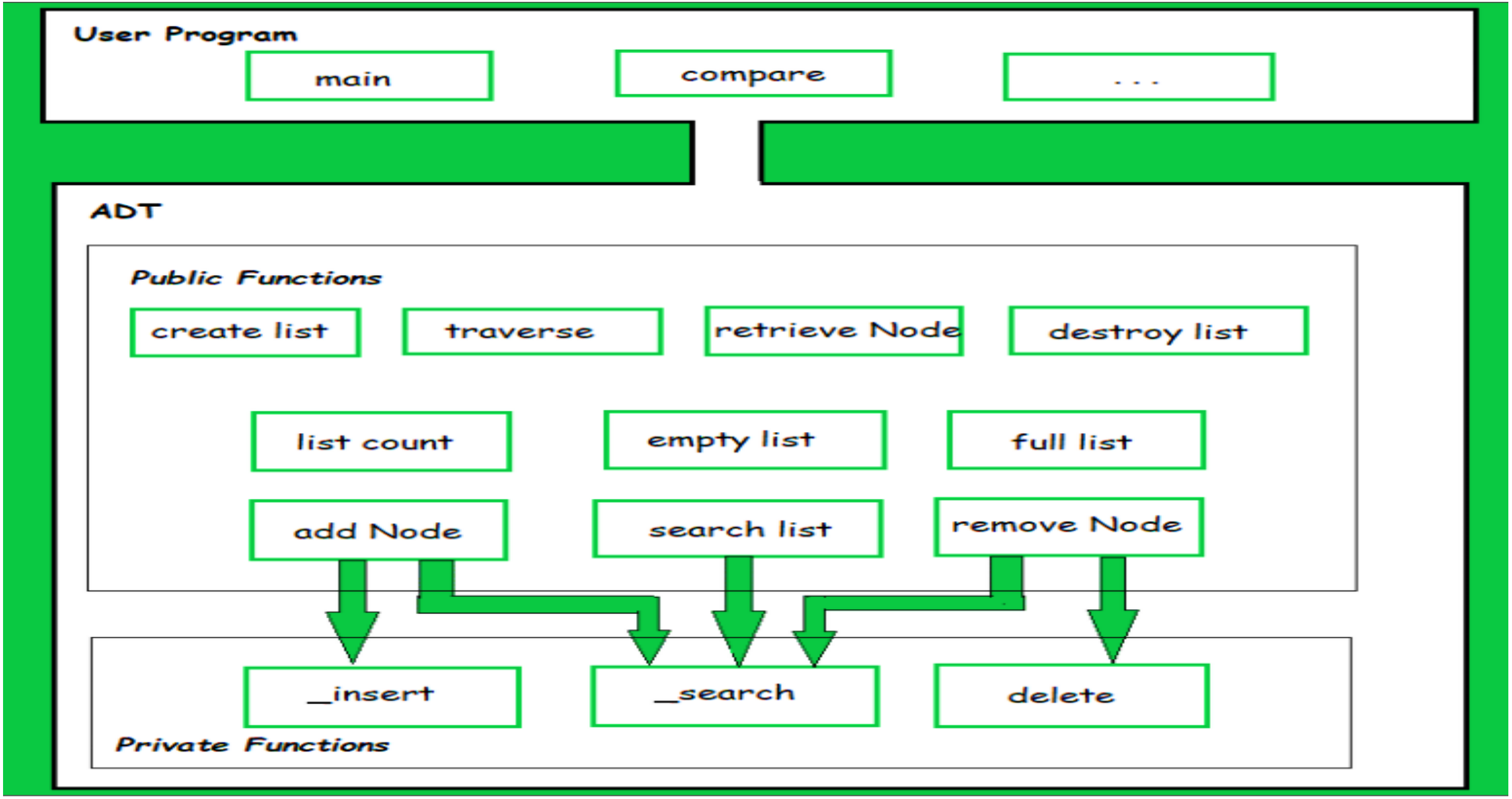
Let us see some operations of those mentioned ADT −

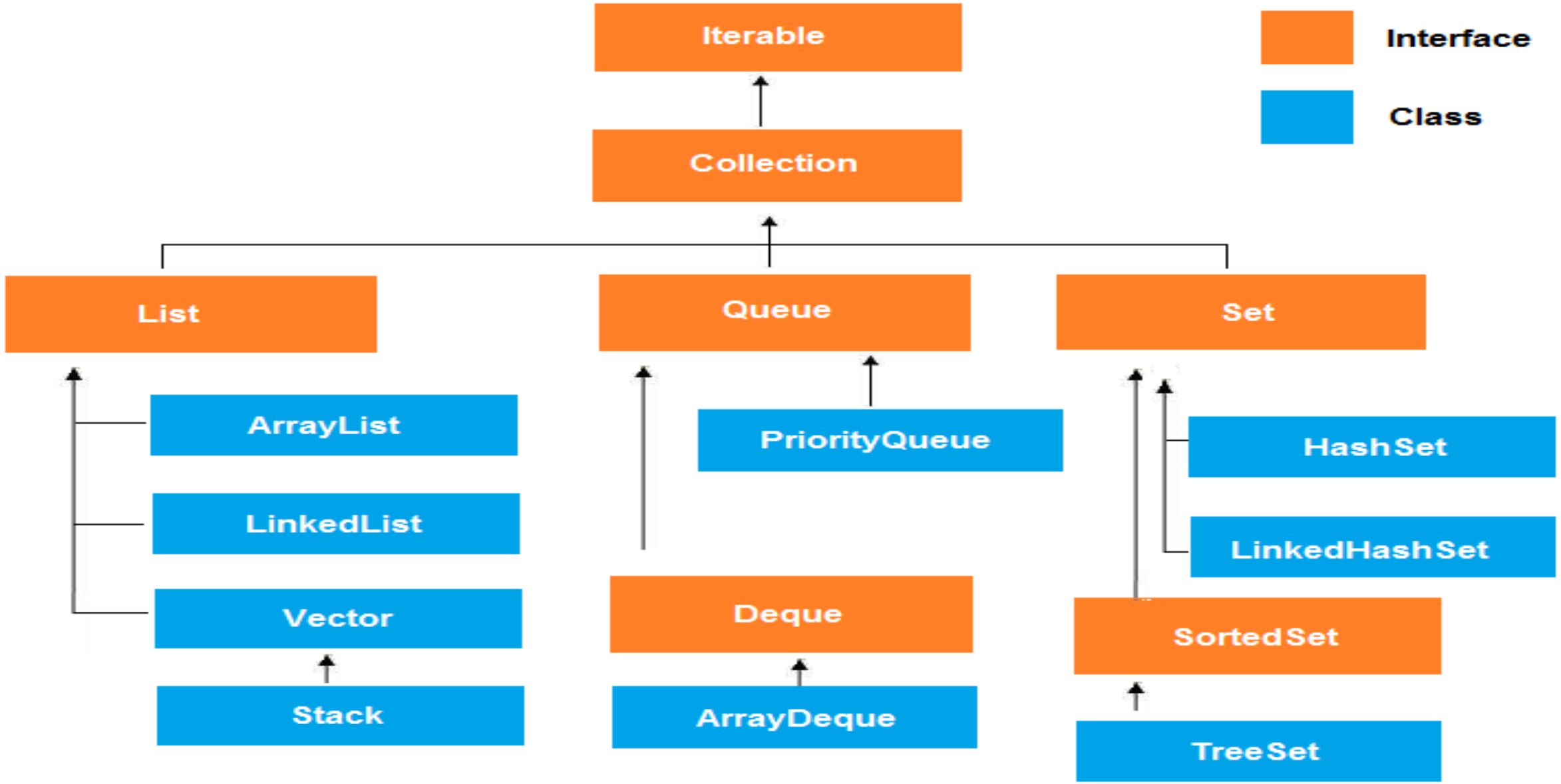
- **Stack −**
  - isFull(), This is used to check whether stack is full or not
  - isEmpry(), This is used to check whether stack is empty or not
  - push(x), This is used to push x into the stack
  - pop(), This is used to delete one element from top of the stack
  - peek(), This is used to get the top most element of the stack
  - size(), this function is used to get number of elements present into the stack
- **Queue −**
  - isFull(), This is used to check whether queue is full or not
  - isEmpry(), This is used to check whether queue is empty or not
  - insert(x), This is used to add x into the queue at the rear end
  - delete(), This is used to delete one element from the front end of the queue
  - size(), this function is used to get number of elements present into the queue

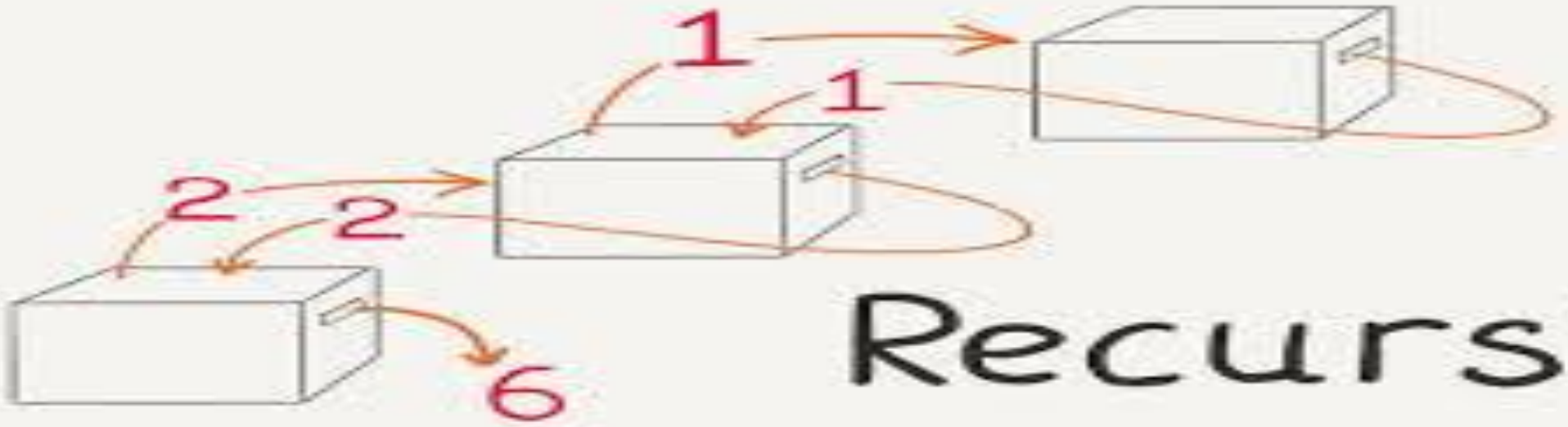
List ADT


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

- **List −**
  - size(), this function is used to get number of elements present into the list
  - insert(x), this function is used to insert one element into the list
  - remove(x), this function is used to remove given element from the list
  - get(i), this function is used to get element at position i
  - replace(x, y), this function is used to replace x with y value


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

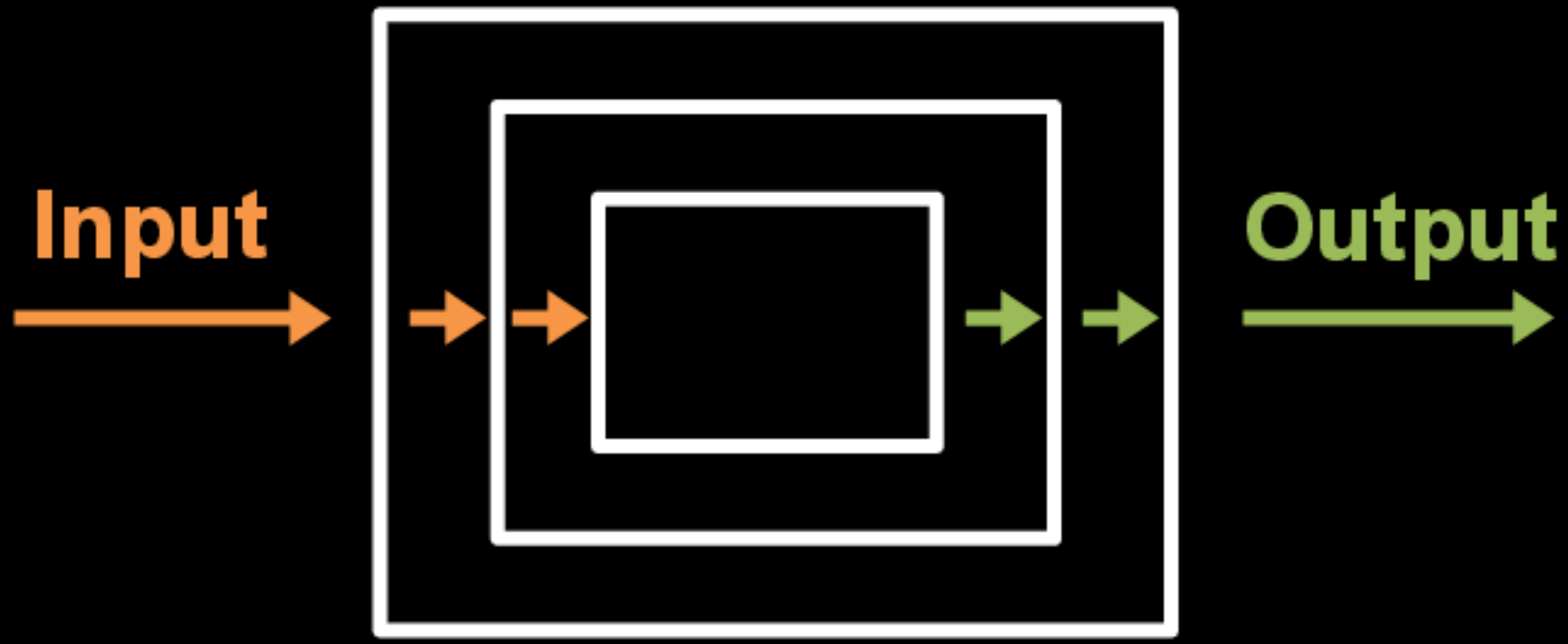

Iterable
Collection
List
Queue
Set
ArrayList
LinkedList
Vector
Stack
PriorityQueue
Deque
ArrayDeque
HashSet
LinkedHashSet
SortedSet
TreeSet
Interface
Class



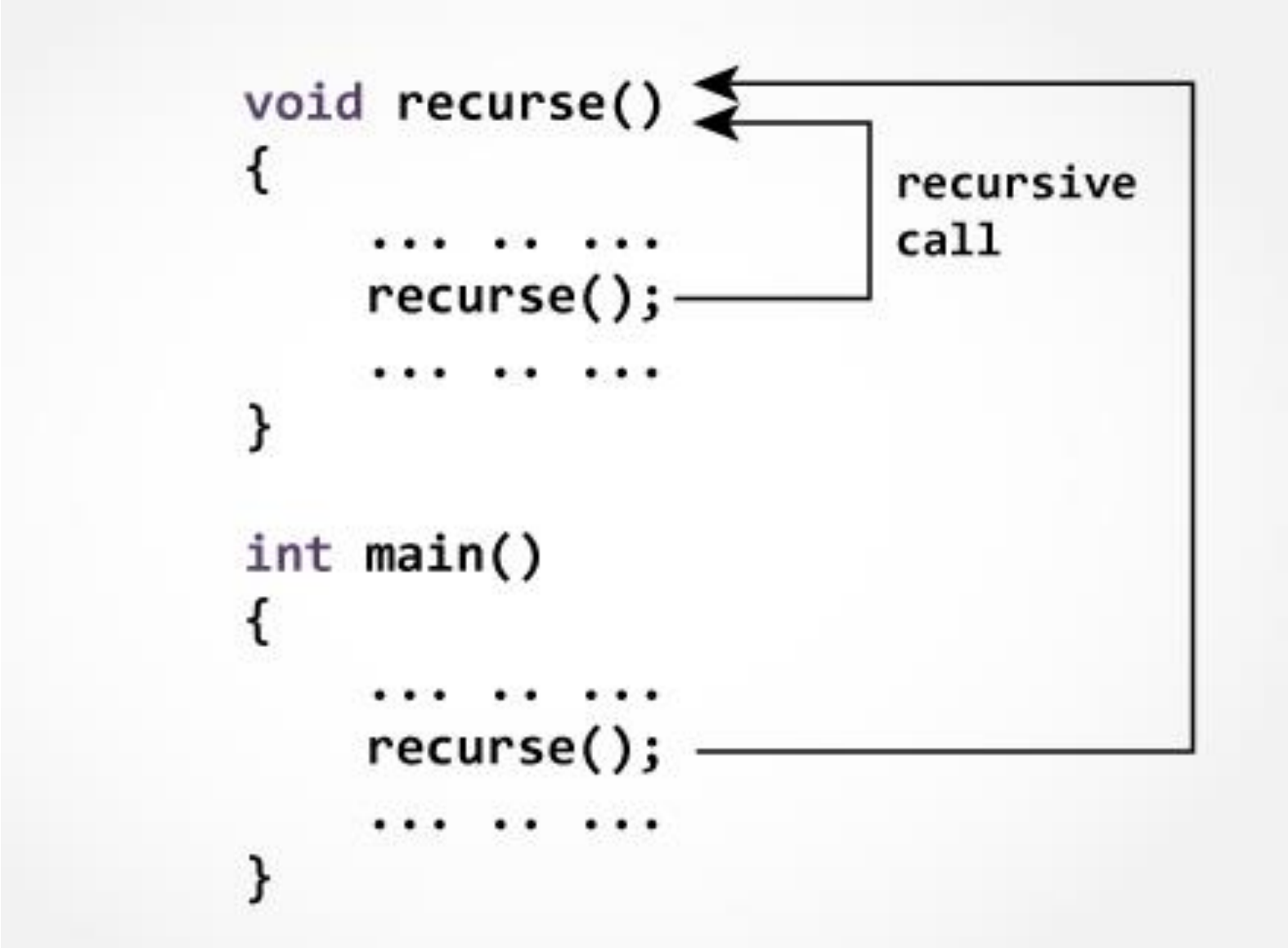
Topics

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

Recursion



How does Recursion works?



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.,

Why Recursion is used Java ?

- Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.
- It makes the code compact but complex to understand.
- Recursive code is generally shorter and easier to write than iterative code.
- It terminates when a base case is reached.
- Each recursive call requires extra space on the stack frame (i.e., Memory)
- Solution to some problem are easier to formulate recursively.

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...
}
```

```java
//Infinite loop
class Recursion2{
    static int i=0;
    static void show()// recusion method
    {
        ++i;1 2 3 4 5 6
        if(i<=5)// termination condition or base
        //to stop execution if condition is true.
        {
        System.out.println("Hi Girls !!!");
        show();// recursive call
        }
    }
    public static void main(String args[])
    {
        show();// call for method
    }
```

```
C:\Windows\system32\c

D:\Test>javac Recursion2.java

D:\Test>java Recursion2
Hi Girls !!!
Hi Girls !!!
Hi Girls !!!
Hi Girls !!!
Hi Girls !!!

D:\Test>
```

```java
//Infinite loop
class Recursion4{

    static int fact(int n)
    {
        if(n<=1)
            return 1;
        else
            return n*fact(n-1);
    }
    public static void main(String args[])
    {
        System.out.println(fact(5));// call for method
    }
}
```

fact(5)= 5*fact(4)
=5*4*fact(3)
=5*4*3*fact(2)
=5*4*3*2 fact(1)
=5*4*3*2*1