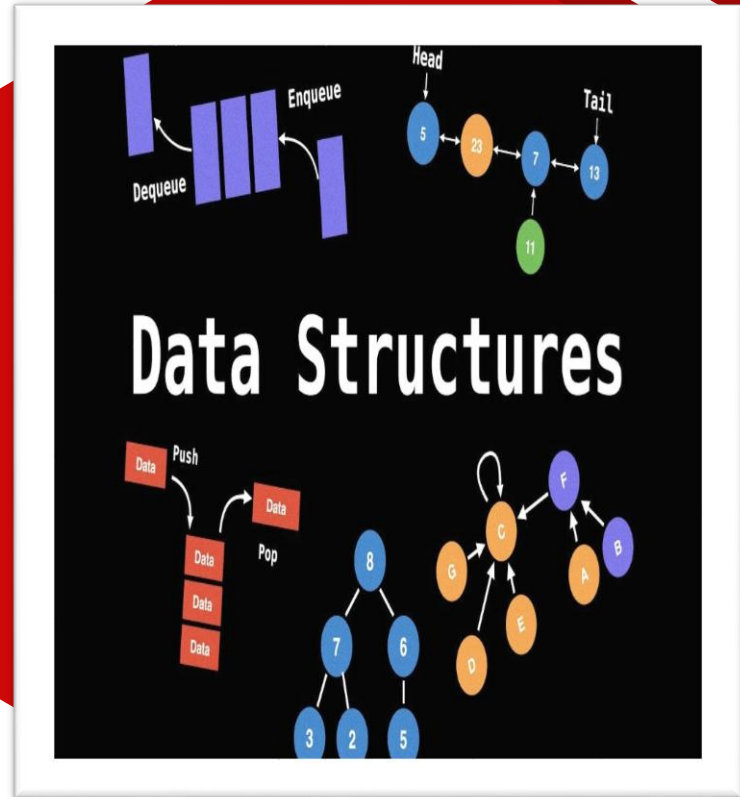


# Algorithms and Data Structures

## Introduction to Data Structures

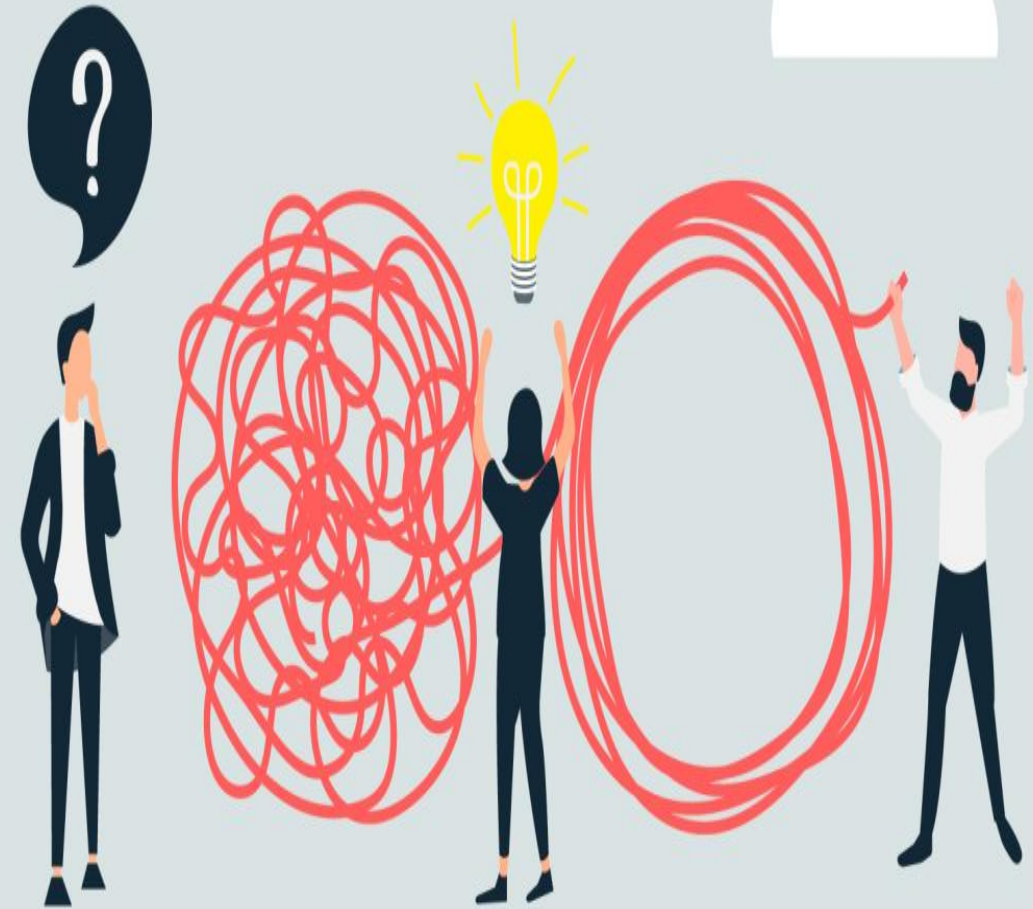
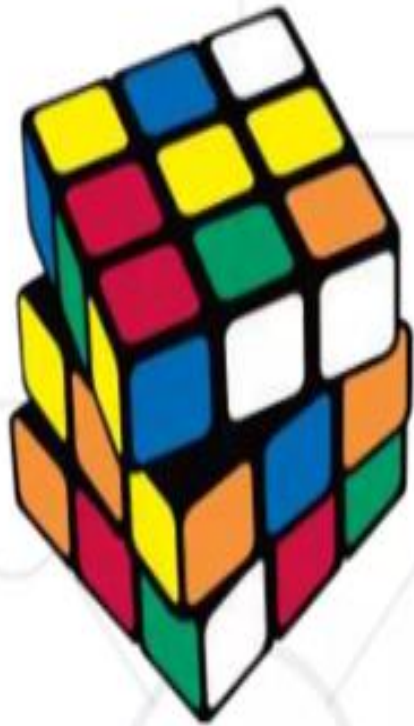
S e s s i o n : D a y 1

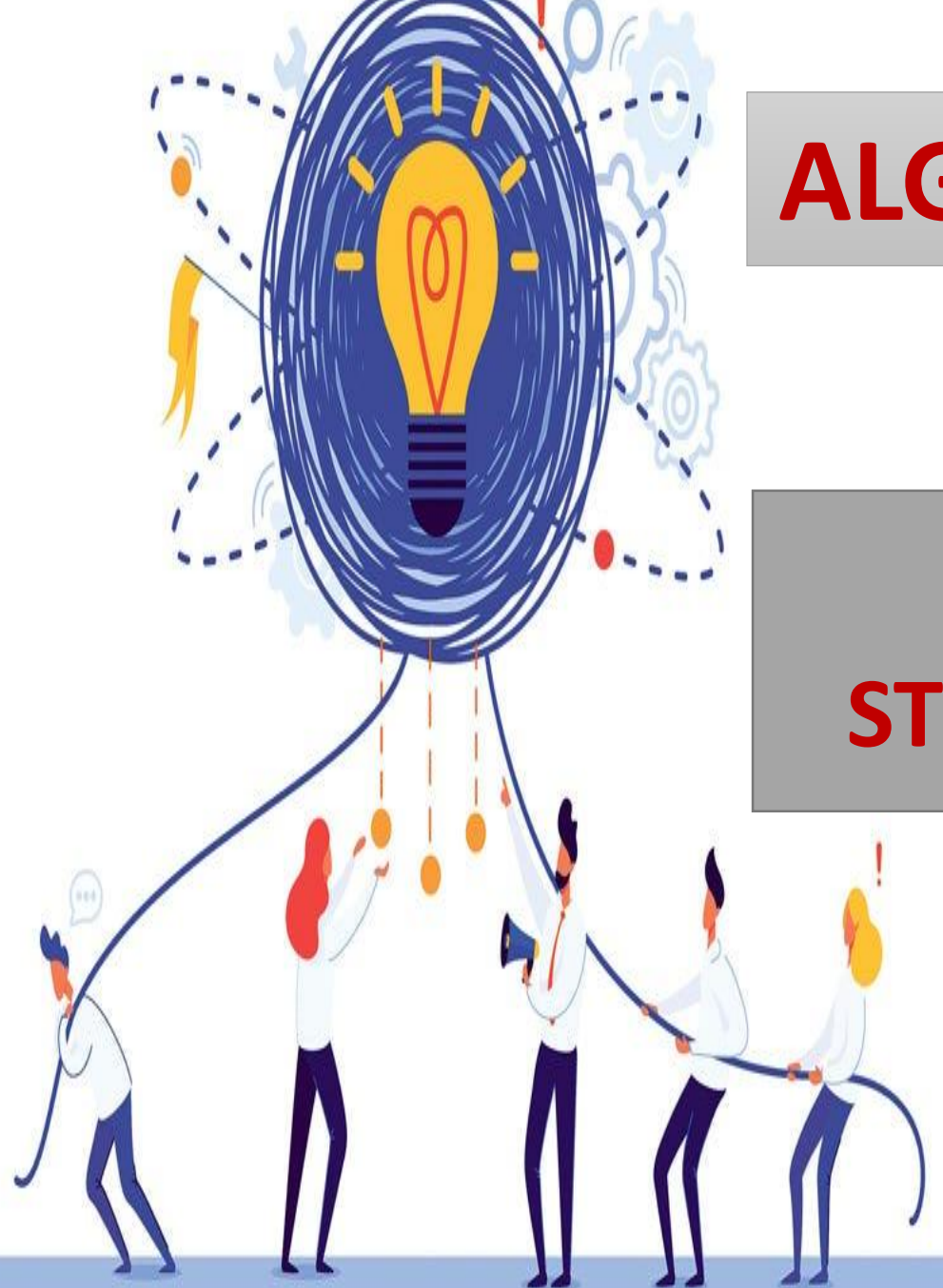
Dr Kiran Waghmare  
CDAC Mumbai



# Problem Solving

How to Solve Problems





**ALGORITHM**



**DATA  
STRUCTURE**



# 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

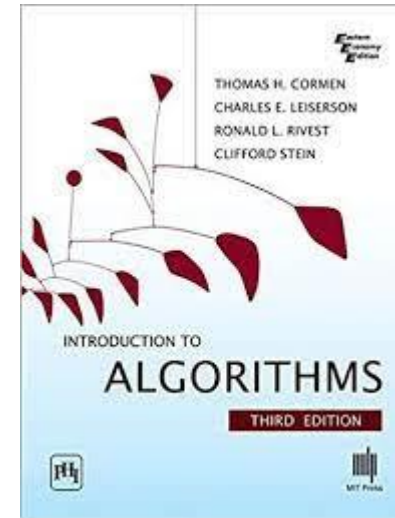
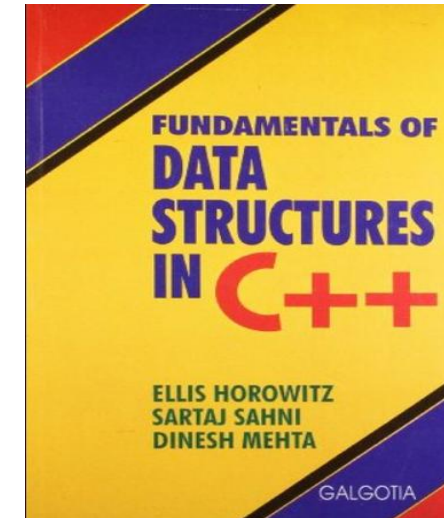
# Module 2: Algorithms and Data Structures

- **Text Book:**

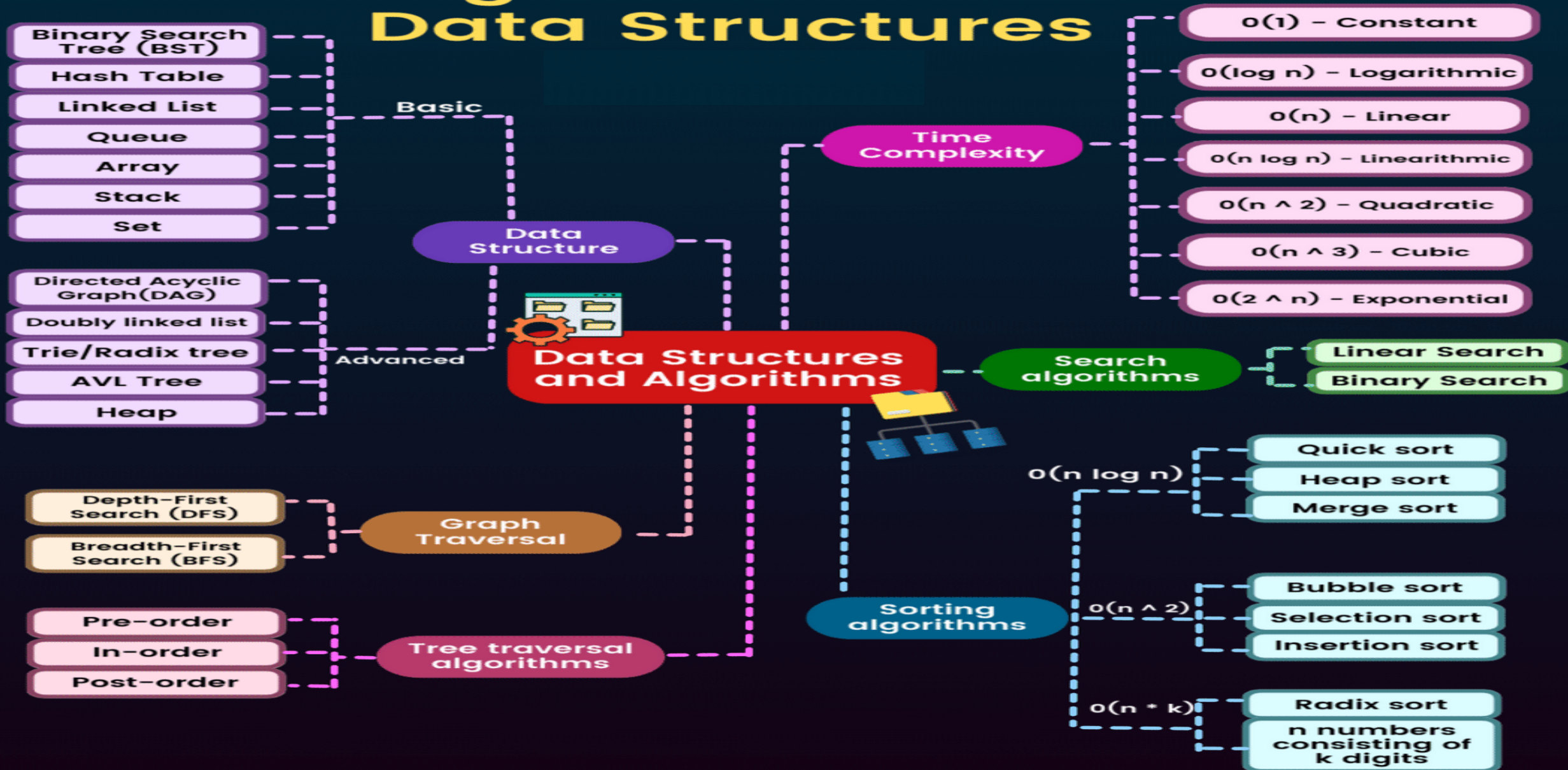
- Fundamentals of Data Structures in C++ by Horowitz, Sahani & Mehta

- **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



# Algorithms and Data Structures



# 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 Arie Warshel**

# Why Study Algorithms and Data Structures?

- World domination

For fun and profit.

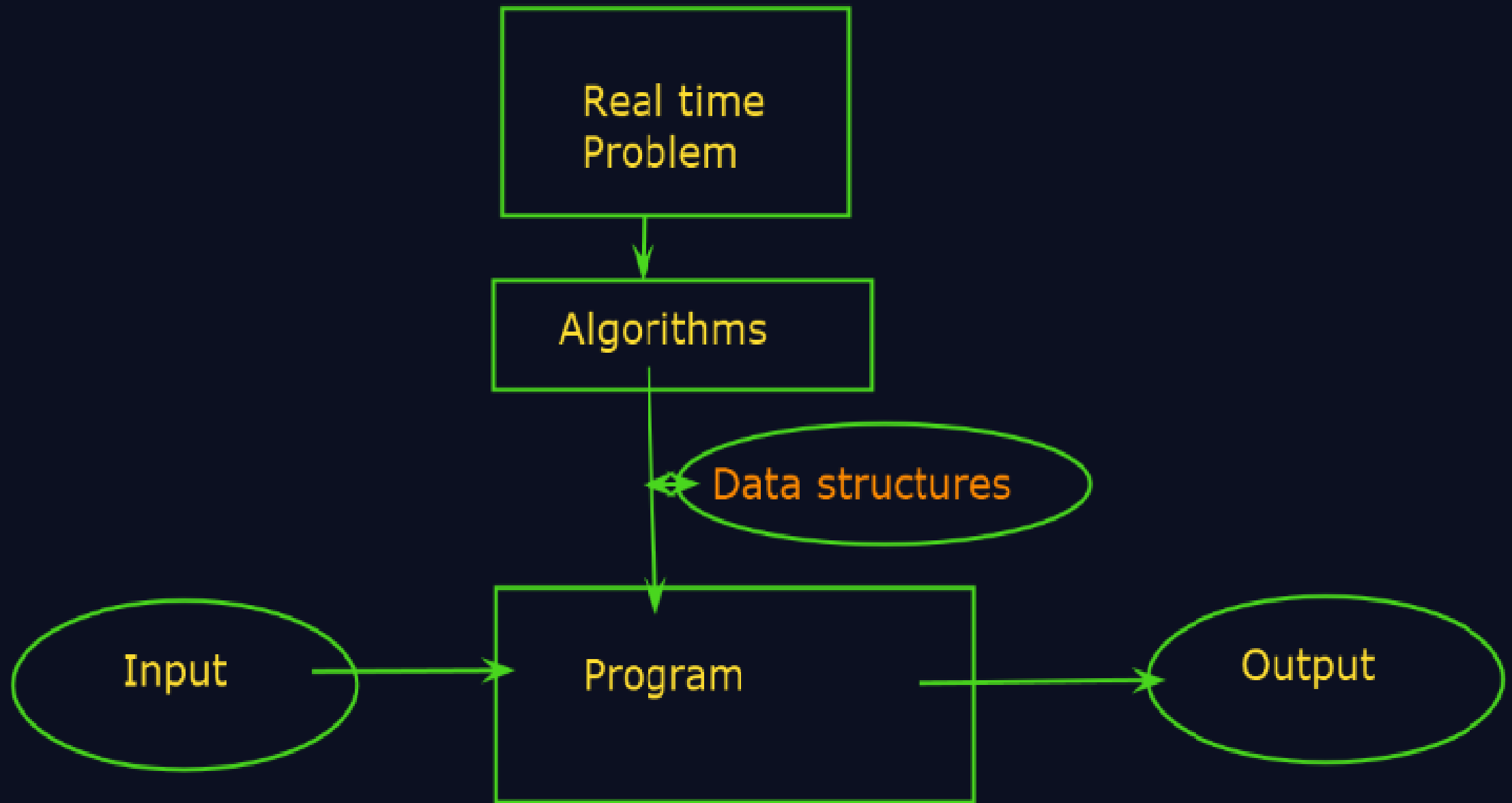


# Algorithms are Everywhere

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

# 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



# 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.

# 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

- **How good is the algorithm?**

- Correctness
- Time efficiency
- Space efficiency

- **Does there exist a better algorithm?**

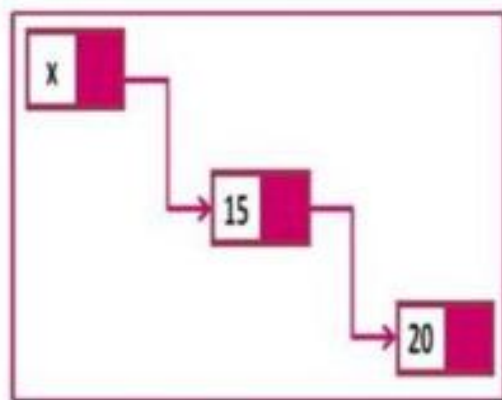
- Lower bounds
- Optimality

# Analysis of Algorithms

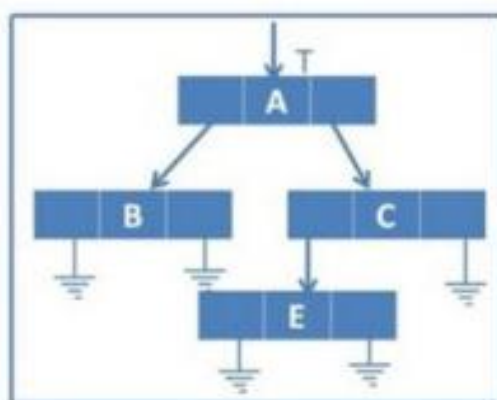
- An algorithm is said to be efficient and fast, if it **takes less time to execute and consumes less memory space.**
- The performance of an algorithm is measured on the basis of following properties :
  - 1.Time Complexity
  - 2.Space Complexity



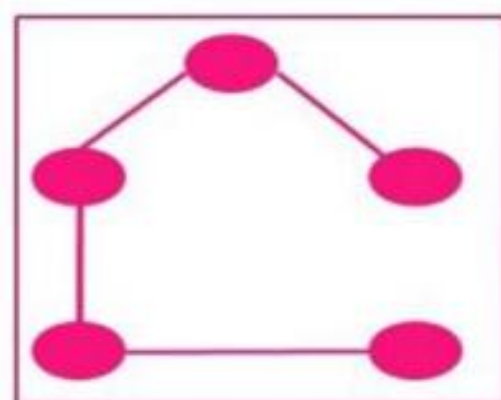
Sorting



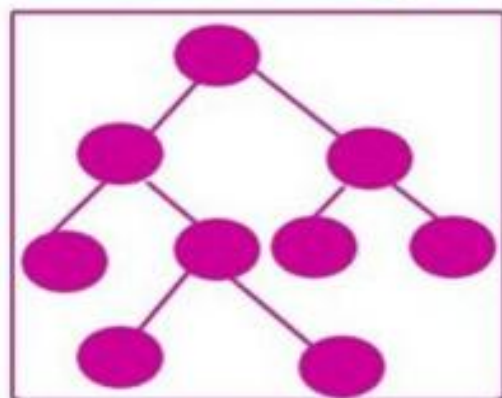
Link list



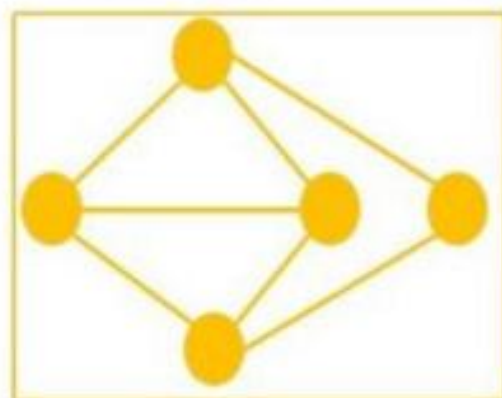
list



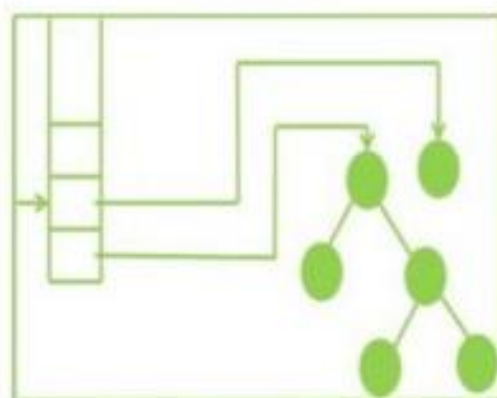
spanning tree



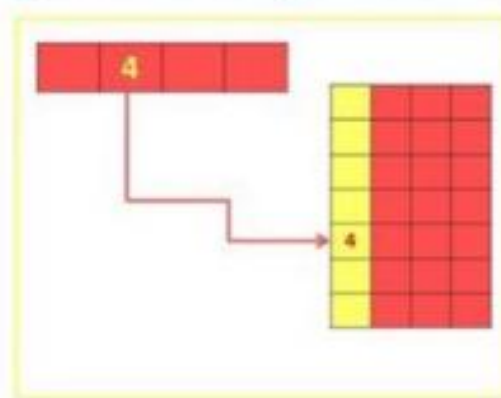
Tree



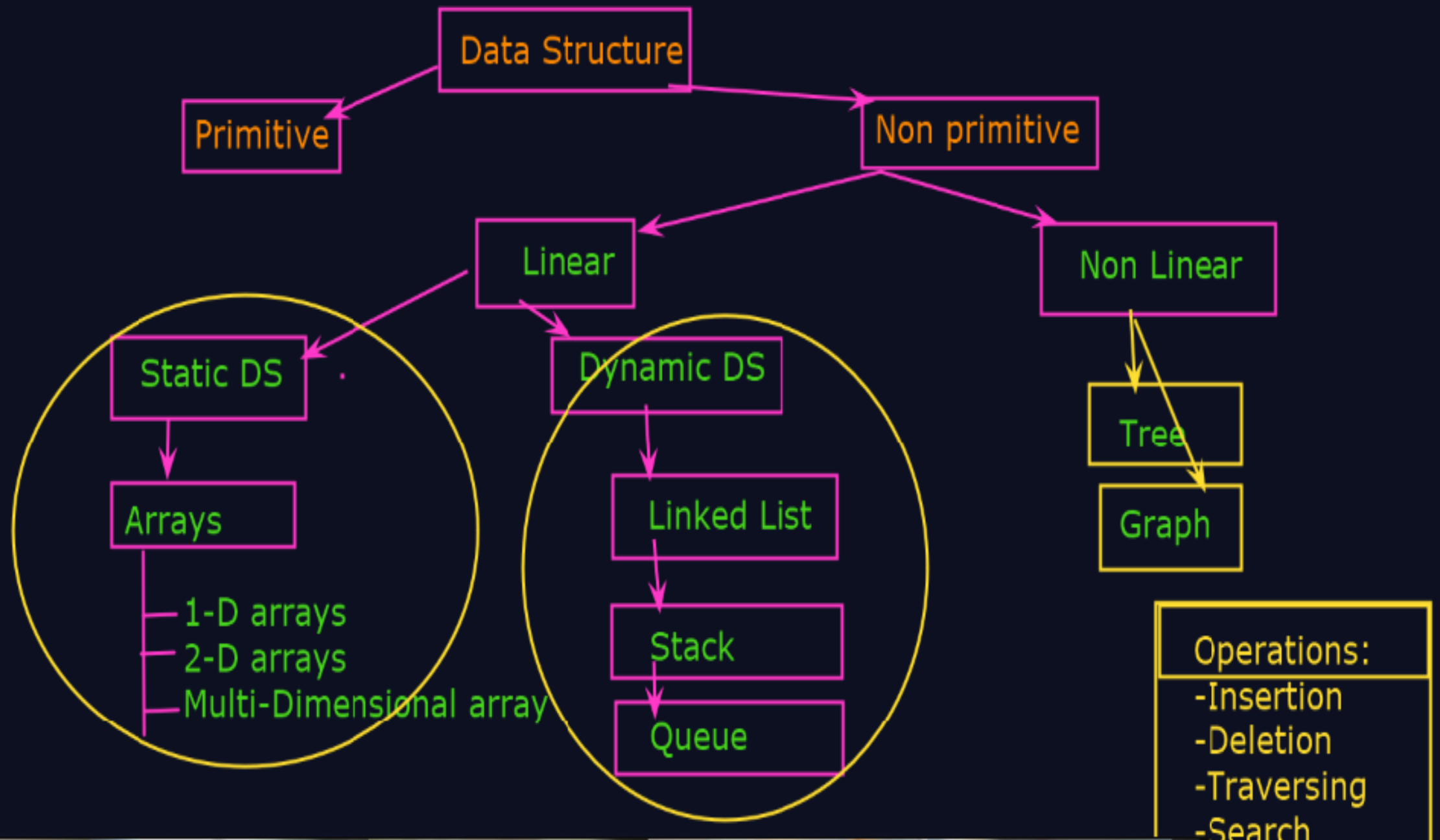
Graph



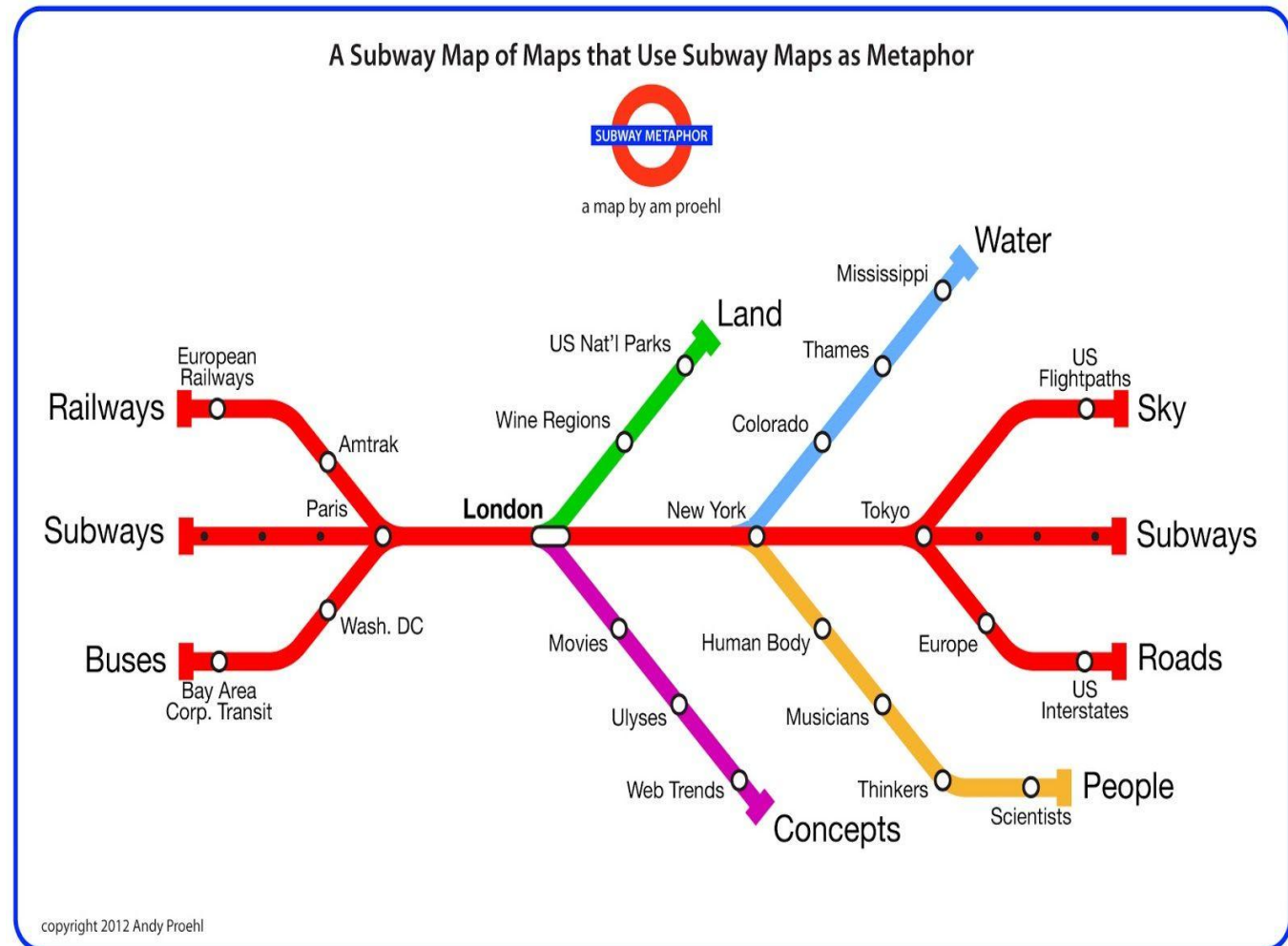
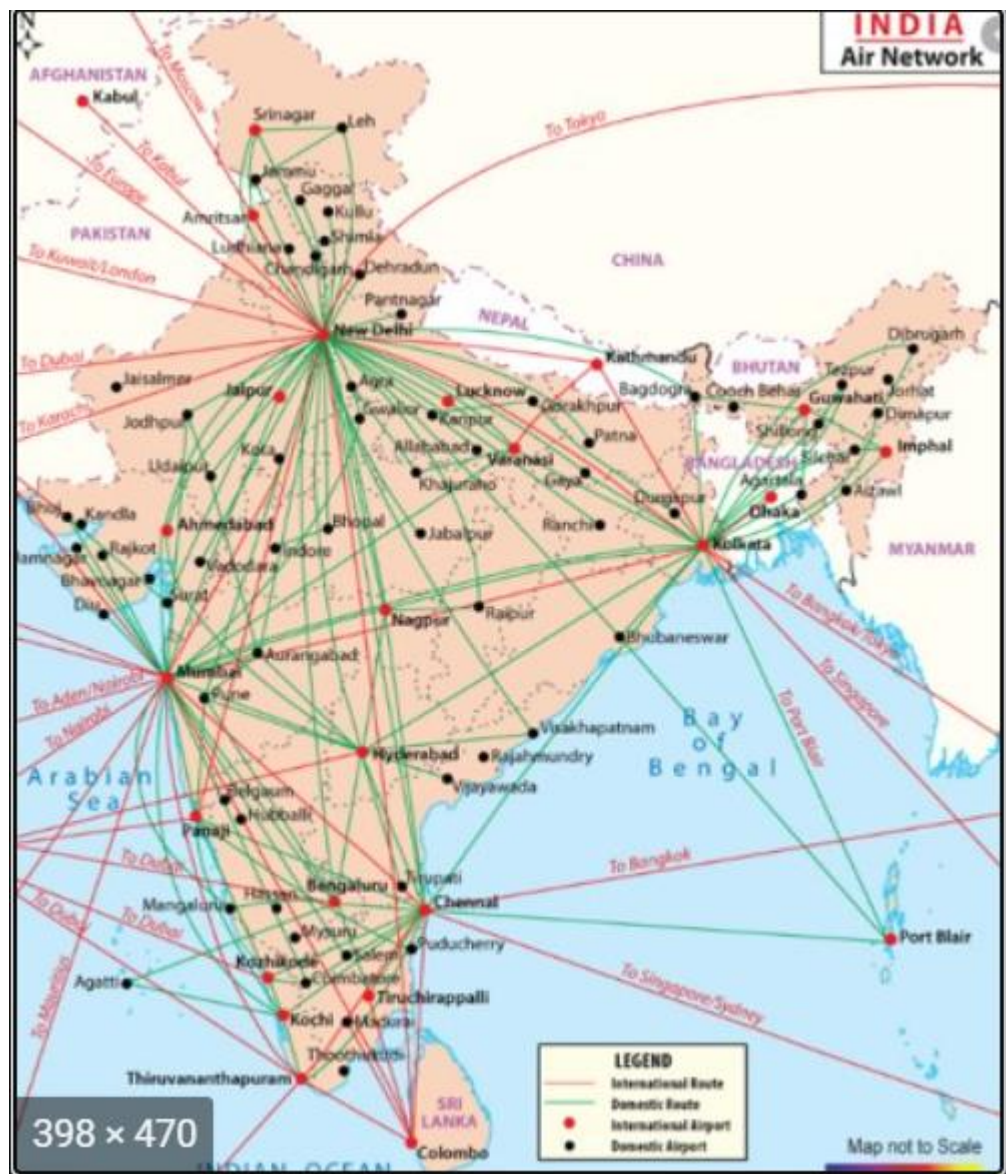
Stack

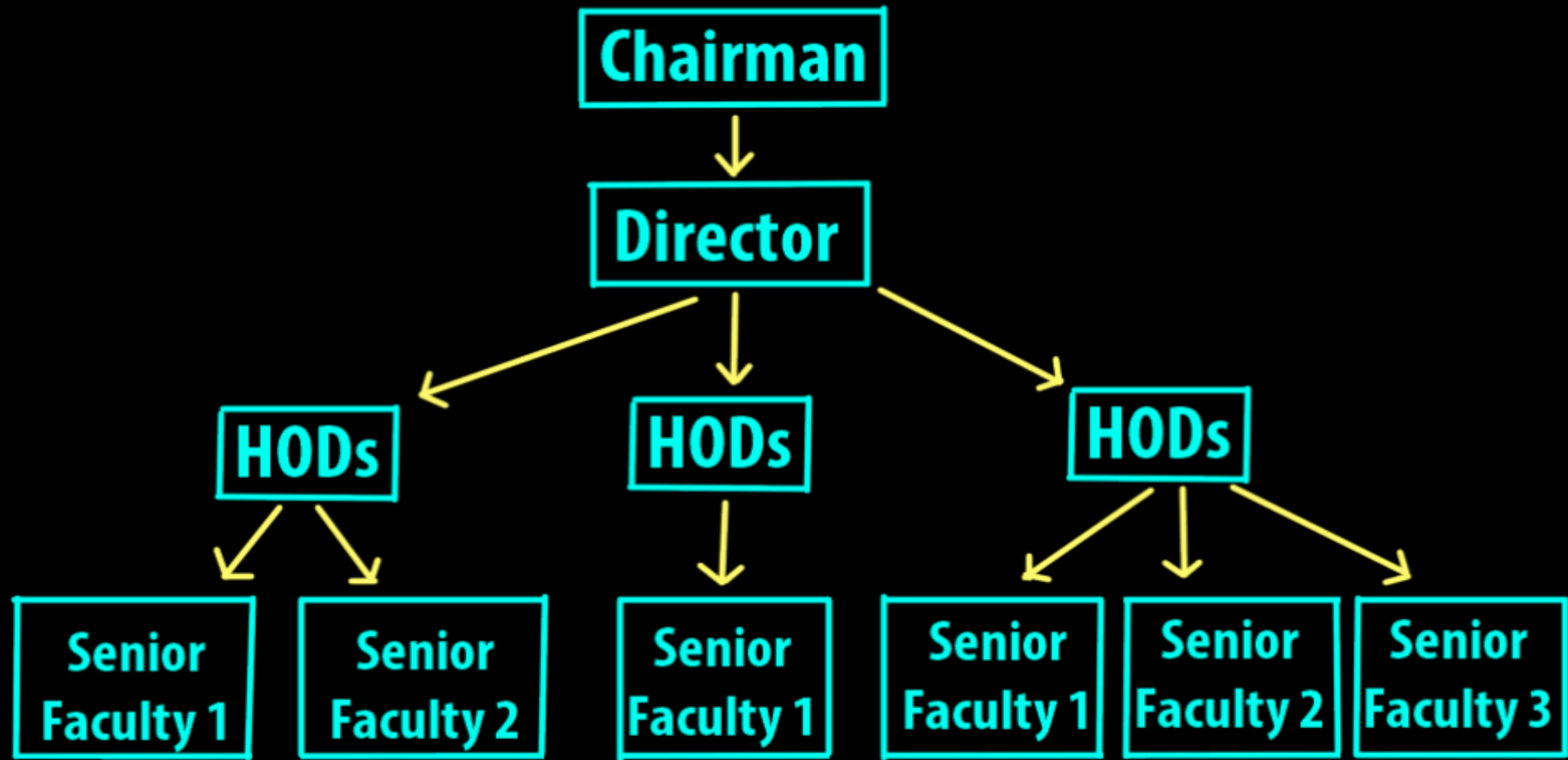


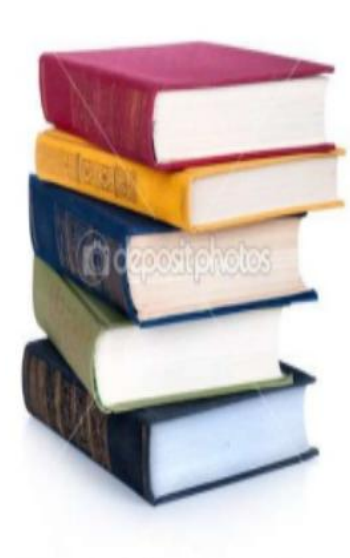
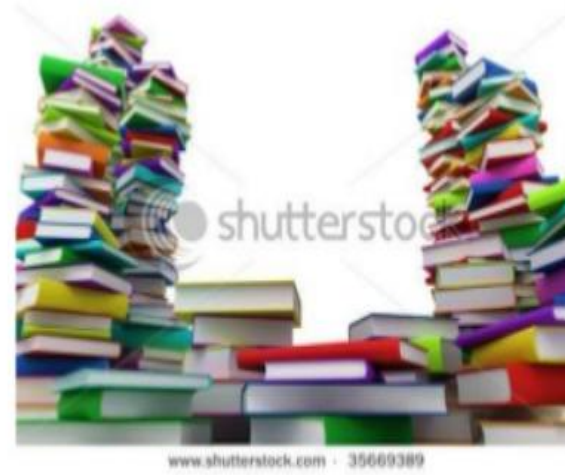
Hashing







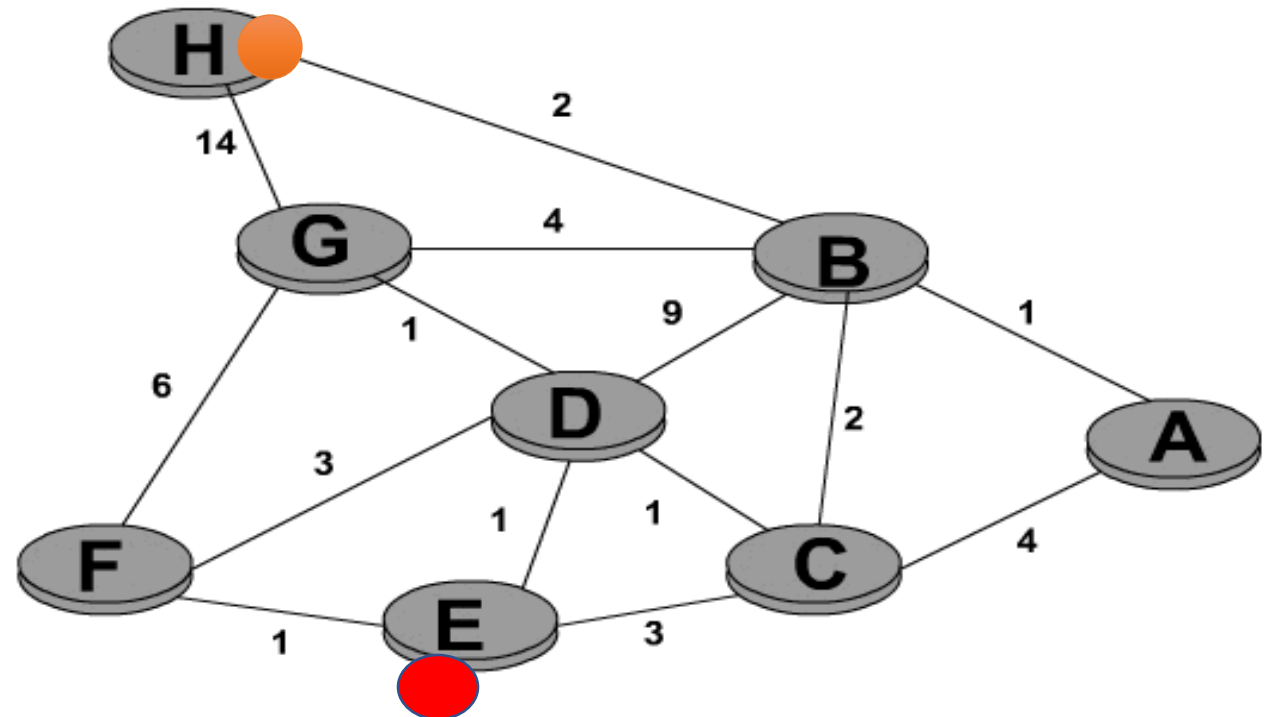
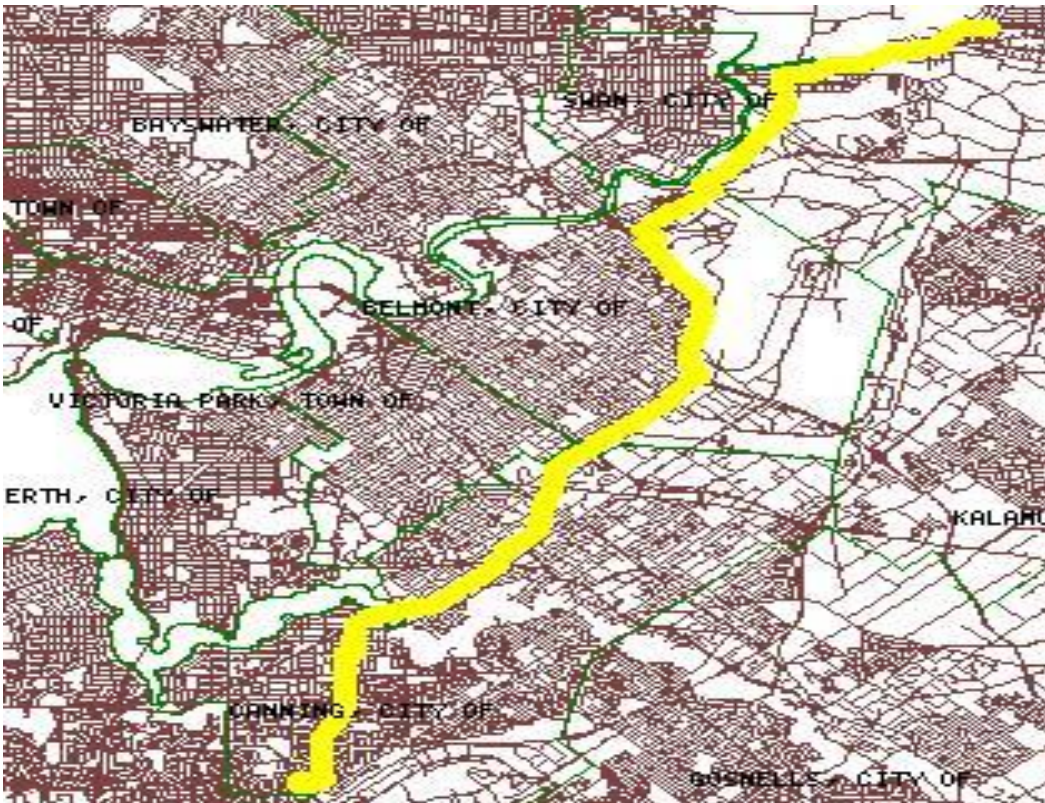






# Why you want to study Algorithms?

- Simply to be cool to invent something in computer science
- Example: Shortest Path Problem and Algorithm
- Used in GPS and Mapquest or Google Maps



# Abstract Data Type (ADT)

# 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**

## Abstraction

### Abstract view/ Logical view

- 8 GB Ram
- photos
- camera
- call()
- App inst()
- phot()
- video()

### Implementation view

```
class oppop{  
    private itnramsize;  
    strin processname  
    screensize  
    camerasize  
    androidver  
  
    public:  
    call(){}  
    photo(){}  
    video(){}  
}
```

arr →

10	20	30	40	50
0	1	2	3	

### Algorithms

### Abstract Data types

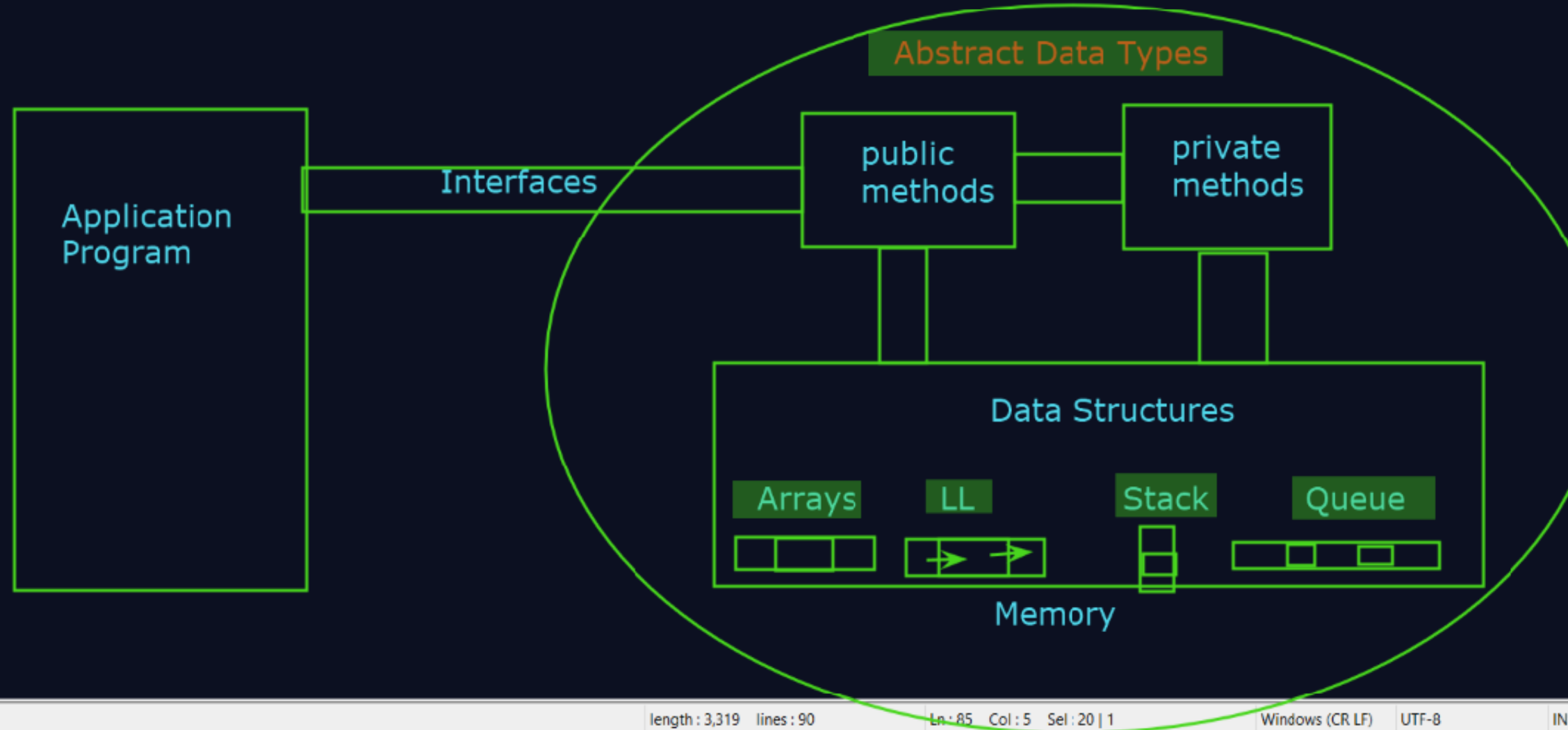
### Set of Rules

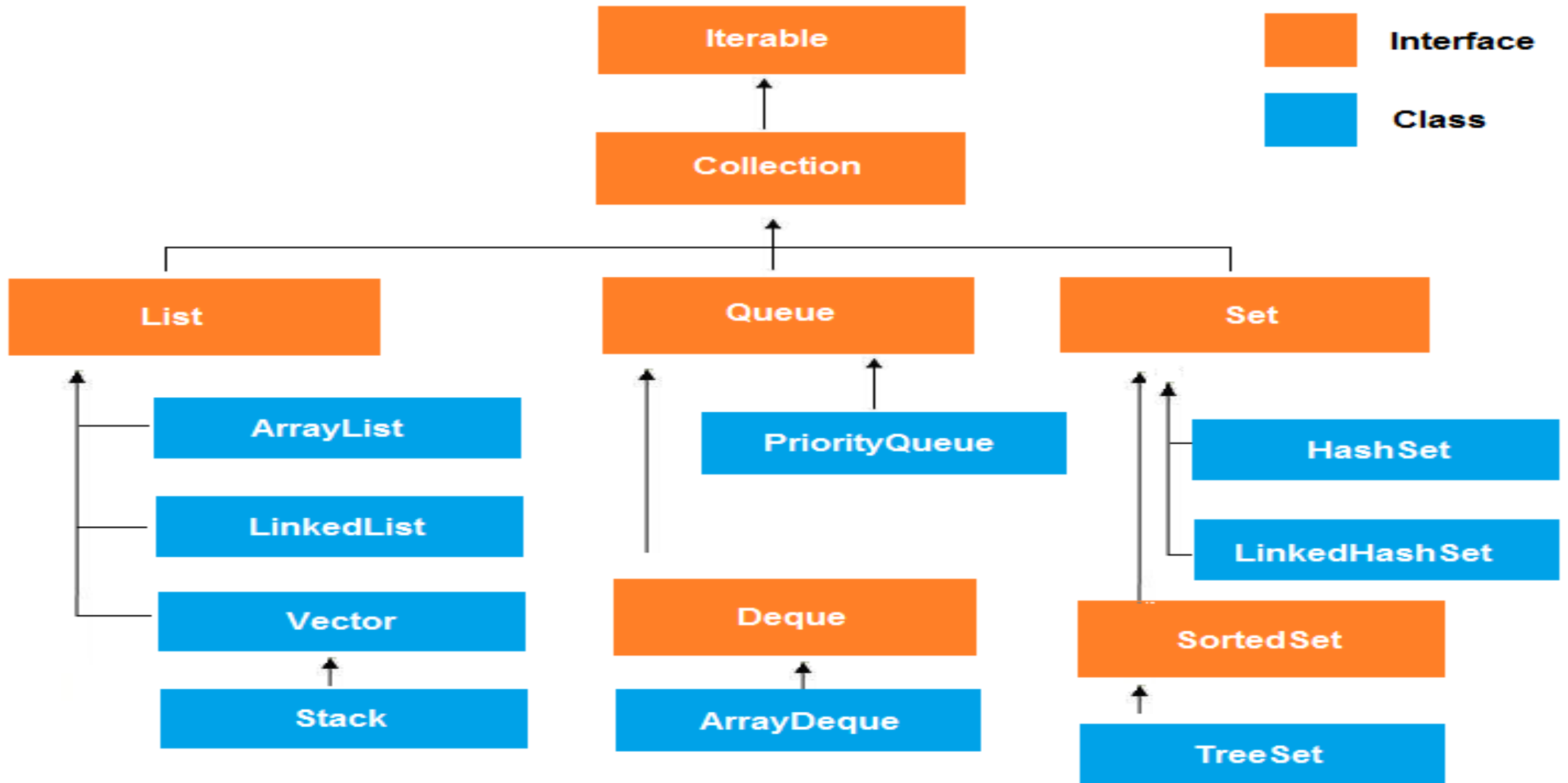
### Array

```
int arr[] = new int[5];
```

```
insert()  
delete()
```

- It is a type/class for objects whose behaviour is defined by value and a set of operations.
- It is called as 'abstract' because it gives an implementation-independent view.
- This process of providing only essentials for implementation and hiding the details is known as abstraction.





//To remove duplicates from ArrayList

```
import java.util.*;
```

```
public class TestDemo2 {  
    public static void main(String[] args) {
```

```
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1,2,2,3,1,4,4,5));
```

```
        //Preserve insertion order
```

```
        ArrayList<Integer> unique = new ArrayList<>(new LinkedHashSet<>(list));
```

```
        //Display
```

```
        System.out.println("Display original List="+list);
```

```
        System.out.println("Display Without duplicates List="+unique);
```

```
    }
```

```
}
```

1 2 3 4 5

//Reverse an ArrayList

```
import java.util.*;
```

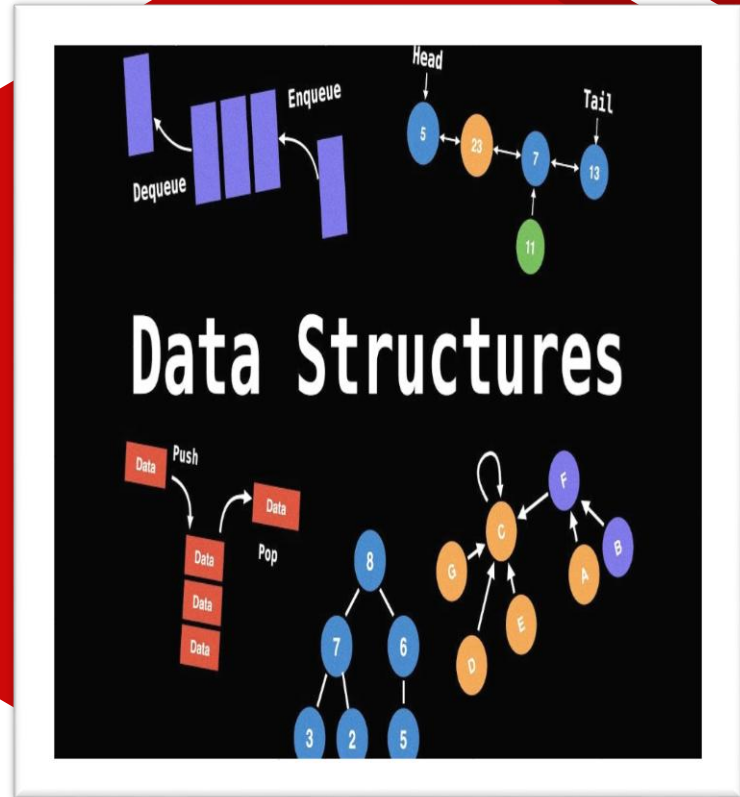
```
public class TestDemo3 {  
    public static void main(String[] args) {  
        ArrayList<String> list = new ArrayList<>(Arrays.asList("Apple", "Banana",  
            "Chickoo"));  
        System.out.println("Display original List="+list);  
        Collections.reverse(list);  
        //Display  
        System.out.println("Display Reverse List="+list);  
    }  
}
```

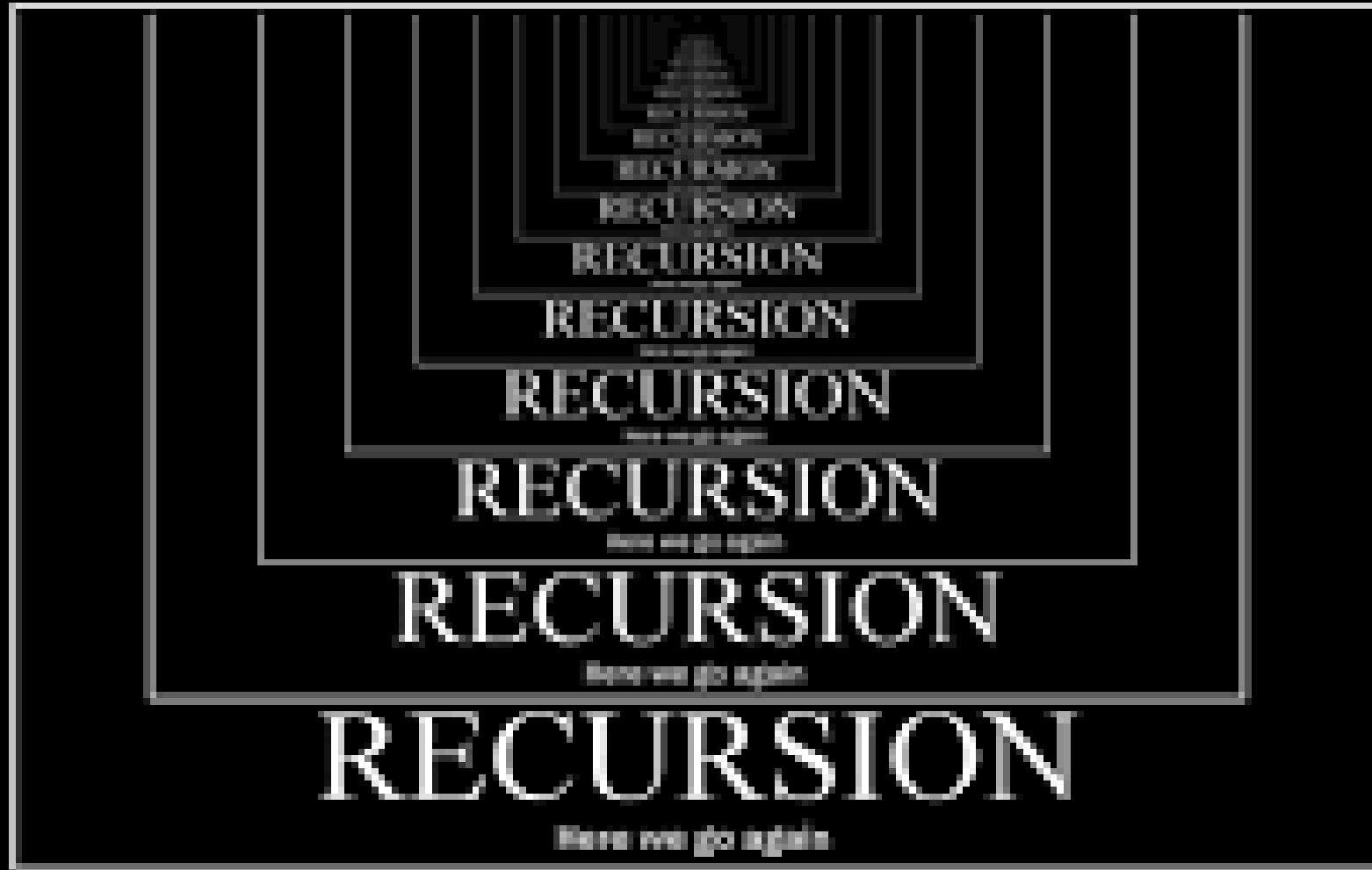
# Algorithms and Data Structures

## Recursion

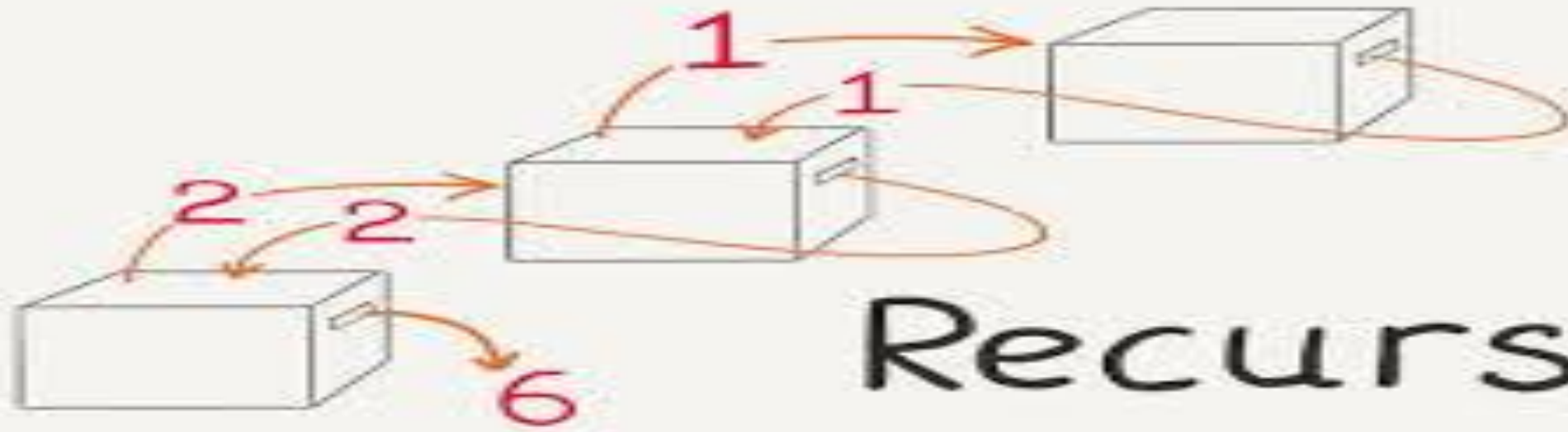
S e s s i o n : D a y 1

Dr Kiran Waghmare  
CDAC Mumbai





RECURSION  
Here we go again

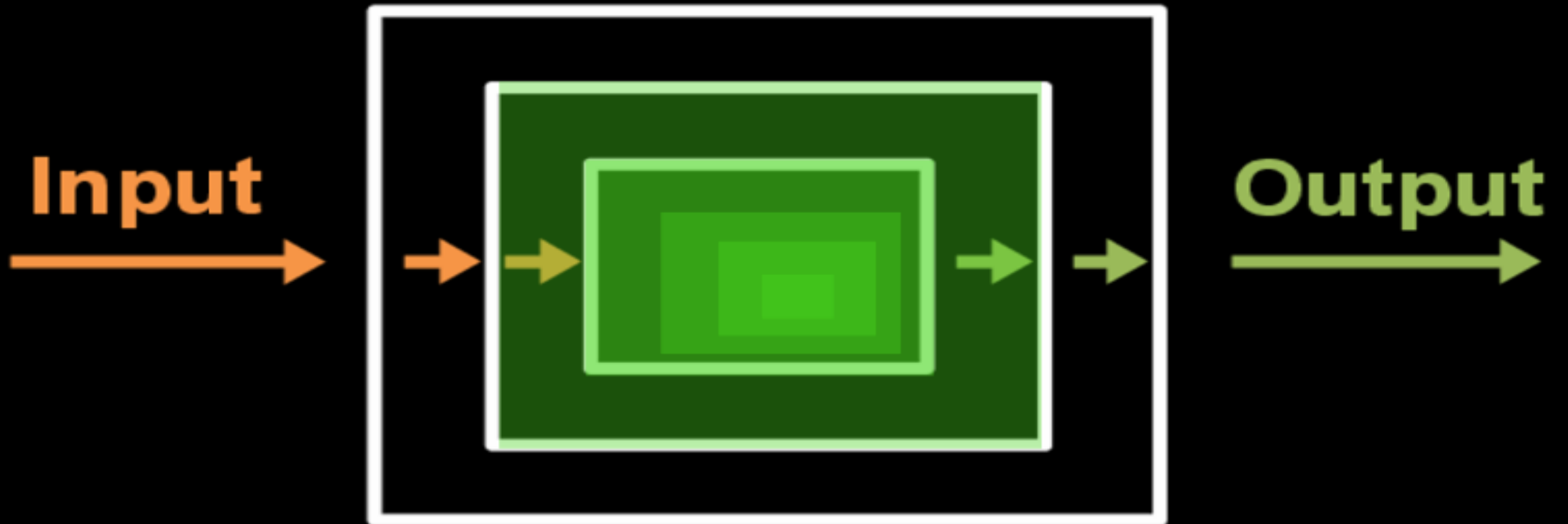


# Recursion

## Topics


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

# Recursion



# How does Recursion works?

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



The diagram illustrates the flow of recursive calls. An arrow labeled "recursive call" points from the `recurse();` line inside the `recurse()` function back to the start of the `recurse()` function. Another arrow points from the `recurse();` line inside the `main()` function to the start of the `recurse()` function.

```
Recursion:  
-----  
  
//recursion  
void rescue() {  
    ....  
    rescue(); //Recursive call  
    ....  
}  
  
int main()  
{  
    rescue();  
}
```

# 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.,

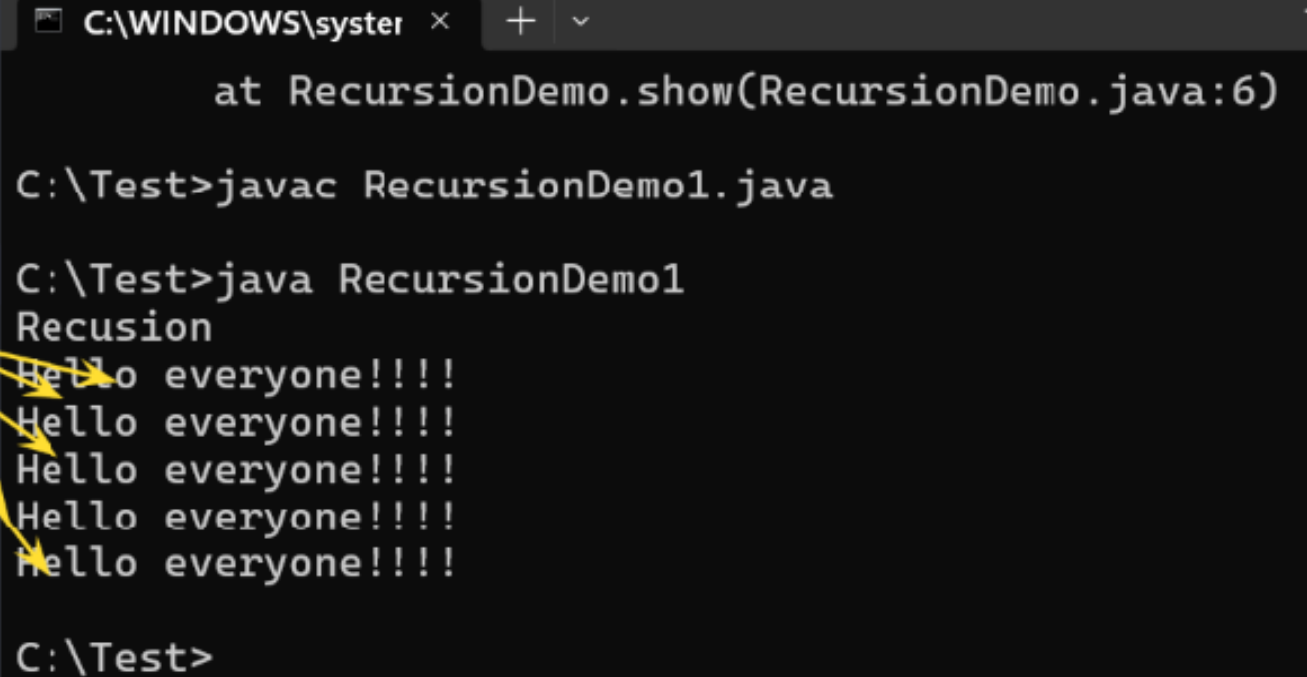
# What is base condition in recursion?

- 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  $n \leq 1$**  is defined and larger value of number can be solved by converting to smaller one till base case is reached.

```
class RecursionDemo1 {  
    static int i=0;  
  
    static void show() {  
        ++i;  
        if(i<=5) {  
            System.out.println("Hello e  
            show();  
        }  
    }  
  
    public static void main(String[] args) {  
        System.out.println("Recusion");  
        show();  
    }  
}
```



```
C:\WINDOWS\systemer x + v  
at RecursionDemo.show(RecursionDemo.java:6)  
  
C:\Test>javac RecursionDemo1.java  
  
C:\Test>java RecursionDemo1  
Recusion  
Hello everyone!!!!  
Hello everyone!!!!  
Hello everyone!!!!  
Hello everyone!!!!  
Hello everyone!!!!  
C:\Test>
```



```
class RecursionDemo2 {
```

```
    static void show(int 5n) {
```

```
        if (n==4)
```

```
            return n;
```

```
        else
```

```
            return 2*show(n+1);
```

```
    }
```

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

```
        System.out.println("Recursion");
```

```
        show(2);
```



16 show(2)  
2 \* show(3)  
8 = 2 \* show(4)  
4