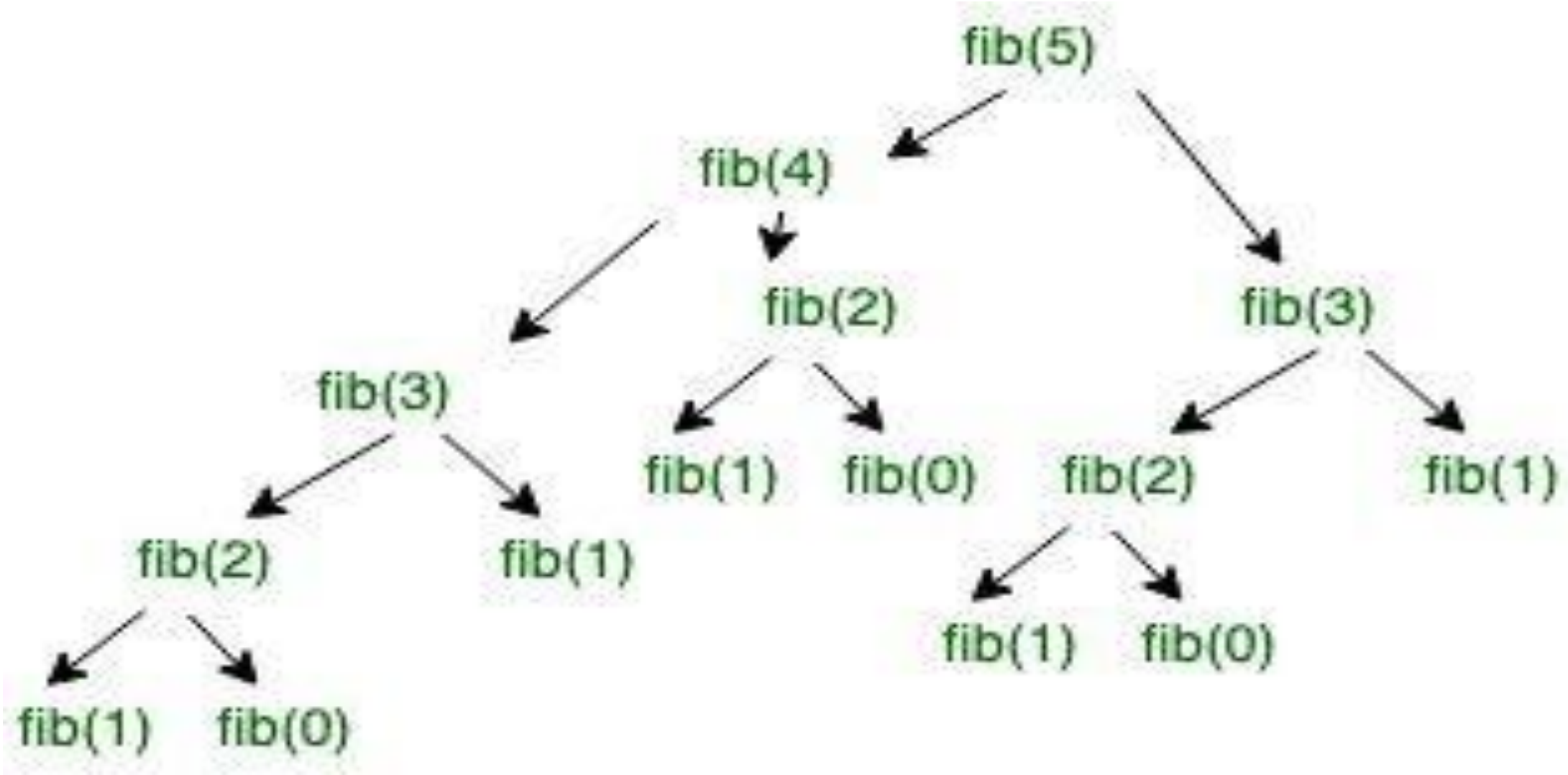
Algorithms & Data Structure
Day 3

Kiran Waghmare

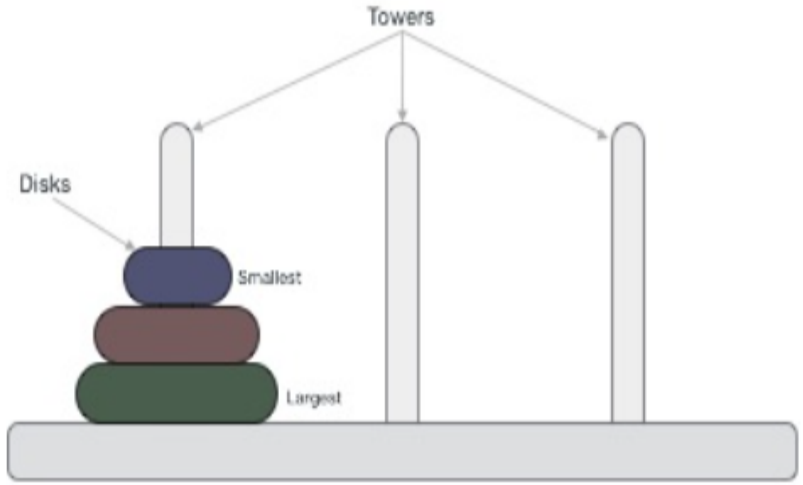

fib(5)
fib(4)
fib(3)
fib(2)
fib(3)
fib(2)
fib(1)
fib(1)
fib(0)
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)

What is Tower of Hanoi?

- A mathematical puzzle consisting of three towers and more than one ring is known as Tower of Hanoi.

- Tower of Hanoi

- The rings are of different sizes and are stacked in ascending order, i.e., the smaller one sits over the larger one. In some of the puzzles, the number of rings may increase, but the count of the tower remains the same.



What are the rules to be followed by Tower of Hanoi?

- **The Tower of Hanoi puzzle is solved by moving all the disks to another tower by not violating the sequence of the arrangements.**

The rules to be followed by the Tower of Hanoi are -

1. Only one disk can be moved among the towers at any given time.
2. Only the "top" disk can be removed.
3. No large disk can sit over a small disk.

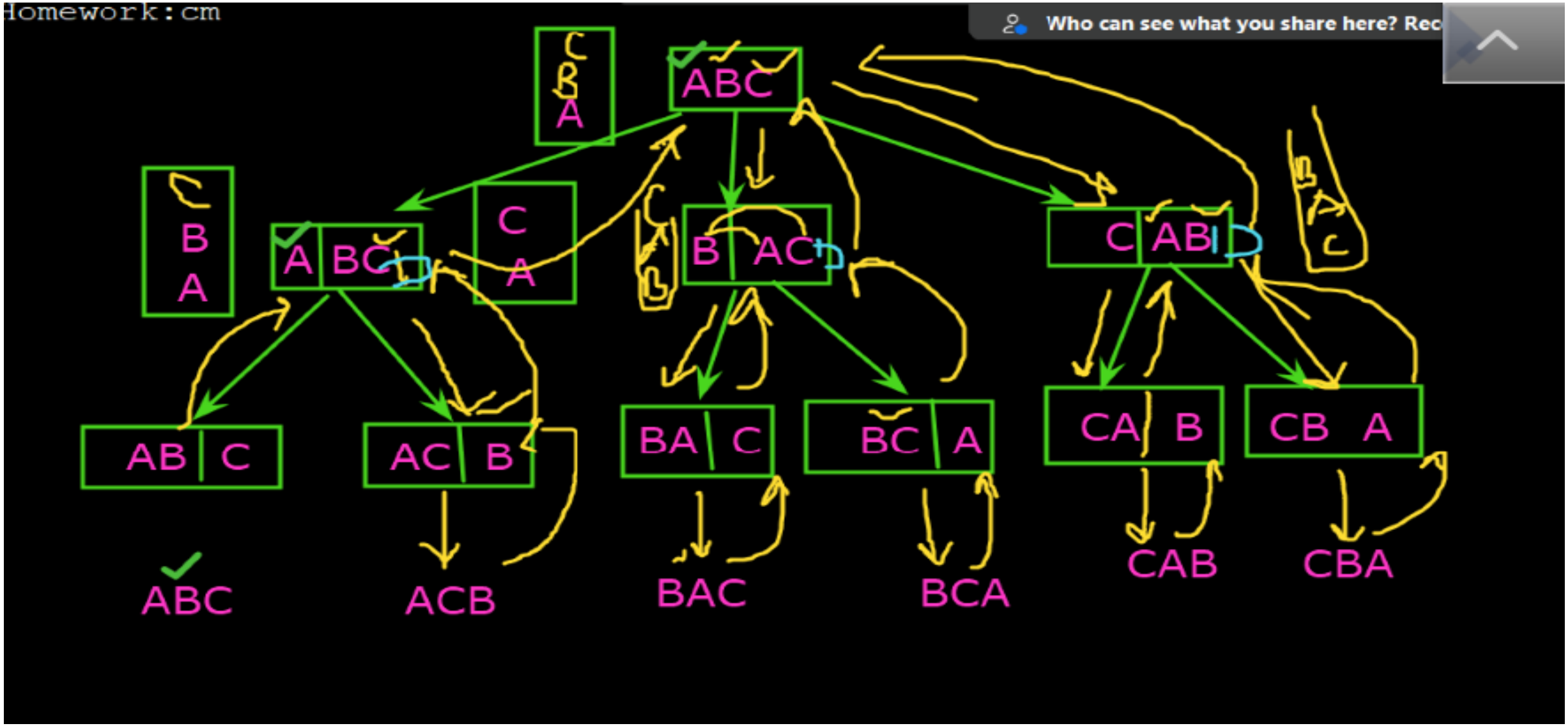
Algorithm 1: Recursive algorithm for solving Towers of Hanoi

```
1 function recursiveHanoi(n, s, a, d)
2     if n == 1 then
3         print(s + " to " + d);
4         return;
5     end
6     recursiveHanoi(n − 1, s, d, a);
7     print(s + " to " + d);
8     recursiveHanoi(n − 1, a, s, d);
9 end
```

Home Work

- Implement Tower of Hanoi Program
- No of Disk=3
- No of Disk=5
- No of Disk=n

Printing all permutation of a string



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?

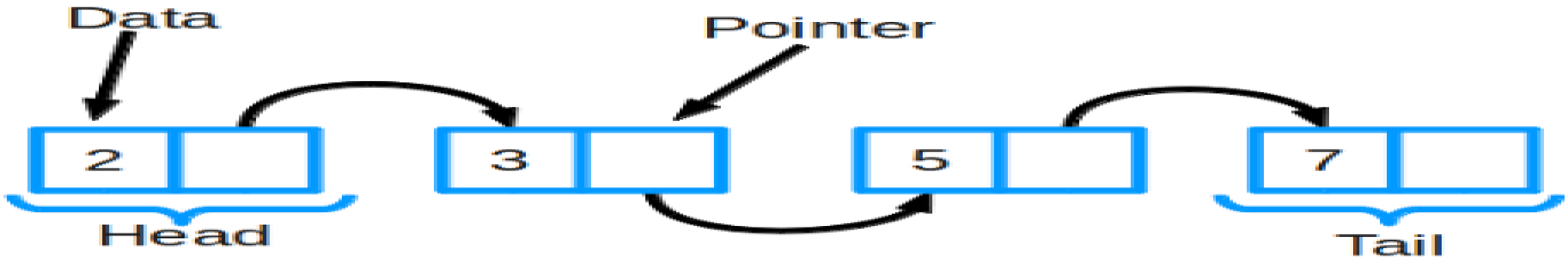
Assignment 1

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

Algorithms & Data Structure

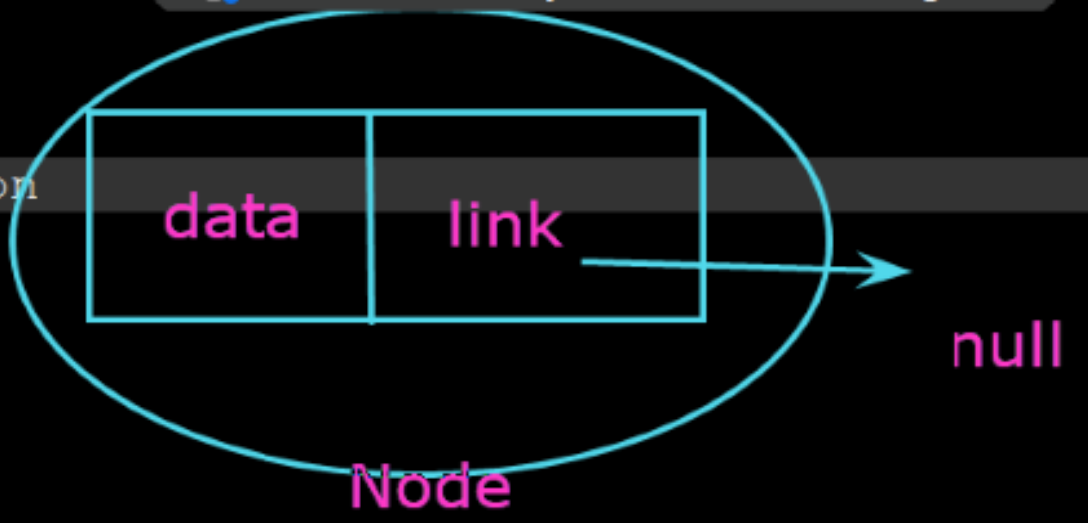
Kiran Waghmare

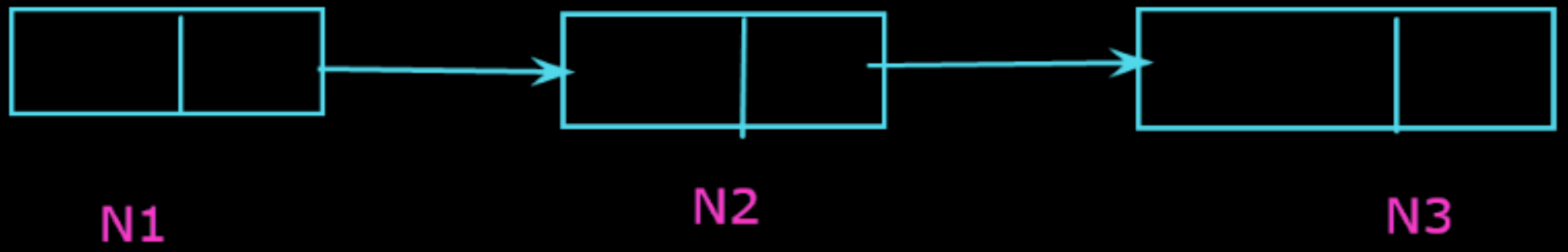
Linked list



Node:

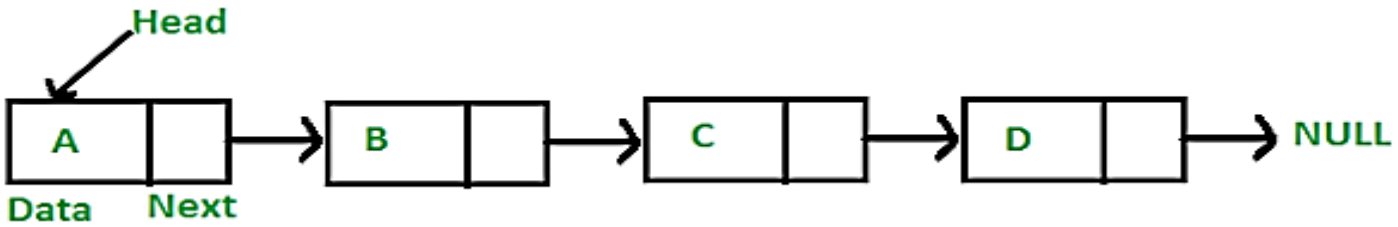
-data: value of node
-link: stores the address of next connection





Linked List Representation

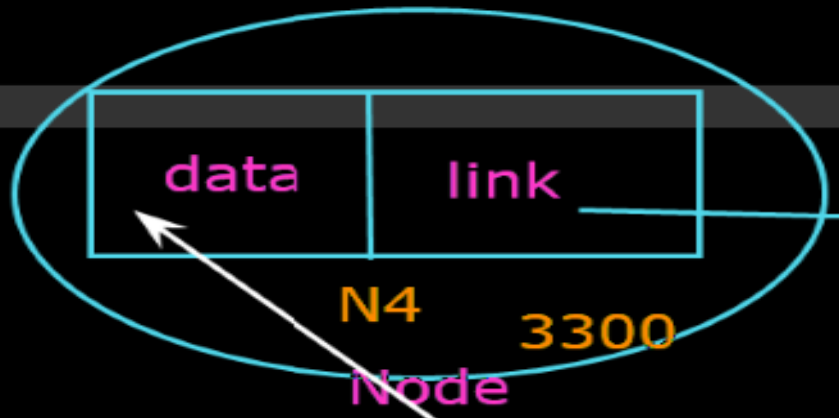
- **Linked list can be visualized as a chain of nodes, where every node points to the next node.**

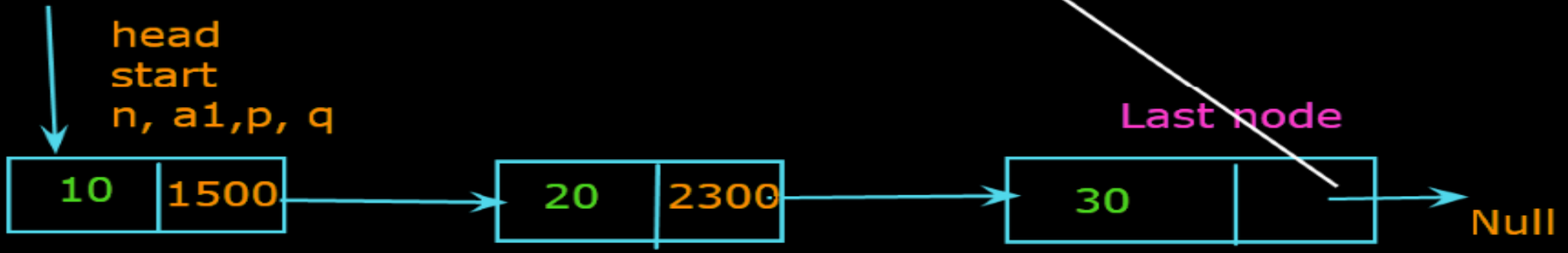


- **As per the above illustration, following are the important points to be considered.**

1. Linked List contains a **link element** called **first.**
2. Each link carries a **data field(s)** and a **link field** called **next.**
3. Each link is **linked with its next link** using its **next link.**
4. **Last link carries a link as null** to mark the end of the list.

```
class Abc
{
Abc a1;
a1 =new Abc();

}
```





N1
1000

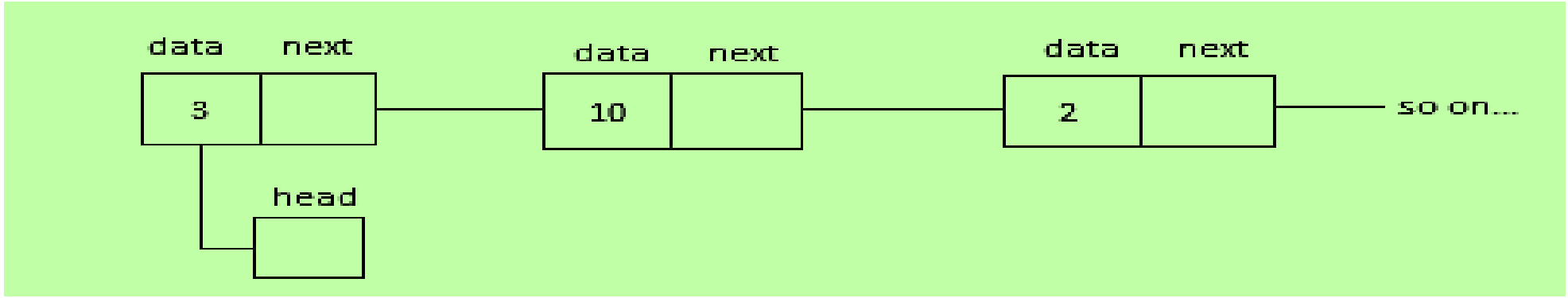
N2
1500

N3
2300

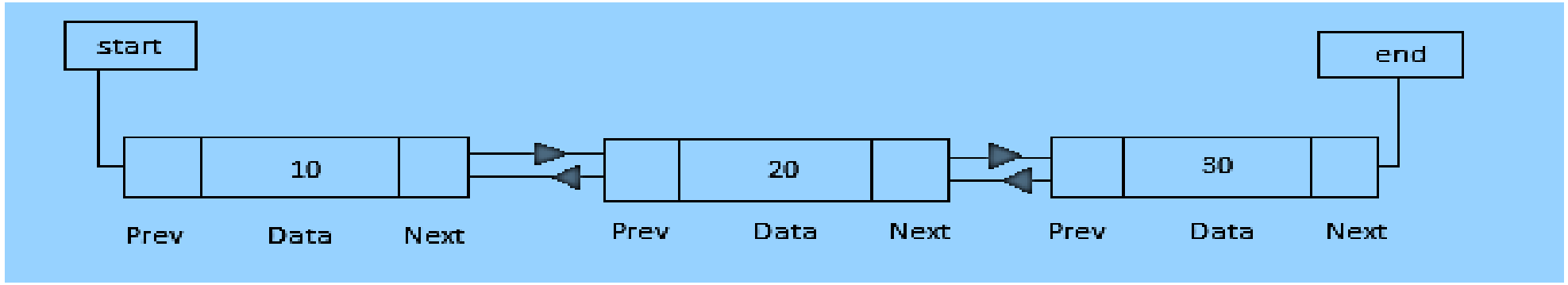
Types of Linked List

- **Following are the various types of linked list.**
  1. **Simple Linked List** – Item navigation is forward only.
  2. **Doubly Linked List** – Items can be navigated forward and backward.
  3. **Circular Linked List** – Last item contains link of the first element as next and the first element has a link to the last element as previous.

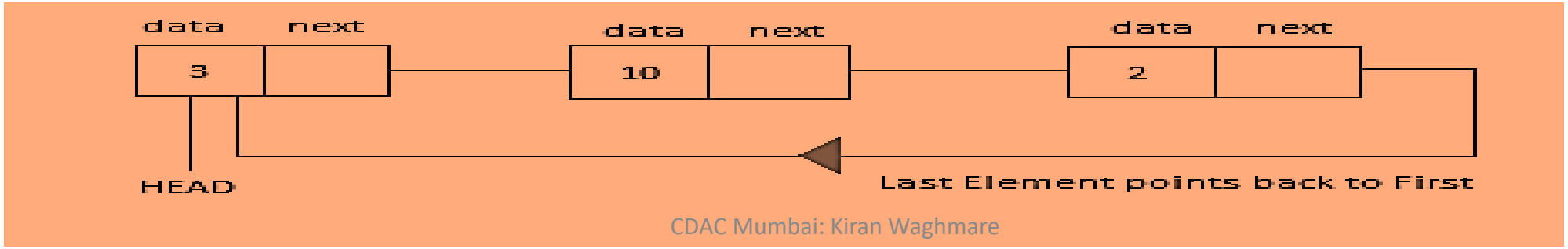
- **Simple Linked List**



- **Doubly Linked List**

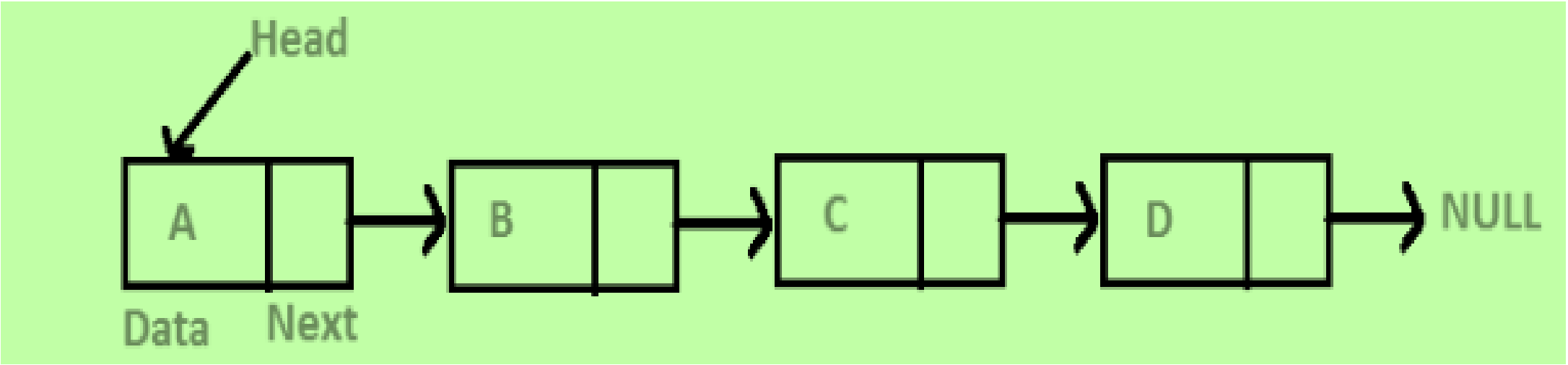


- **Circular Linked List**



Singly Linked List

- **Singly Linked Operations: Insert, Delete, Traverse, search, Sort, Merge**



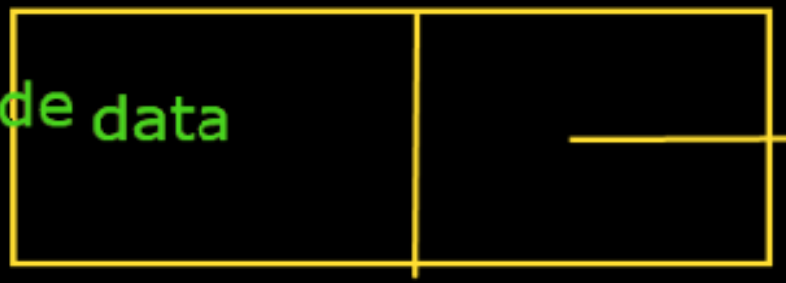
Node Implementation:


```
class Node
{
    int data;
    Node next;
    Node(int d)
    {
        data=d;
        next=null;
    }
}
```

structure of Node

variables for Node

values supplied to the Node



Node Implementation:

```
class Node
{
    int data;
    Node next;
    Node(int d)
    {
        data=d;
        next=null;
    }
}
```

create one single node

```
class LinkedList
{
    Node head;
```

Head pointer

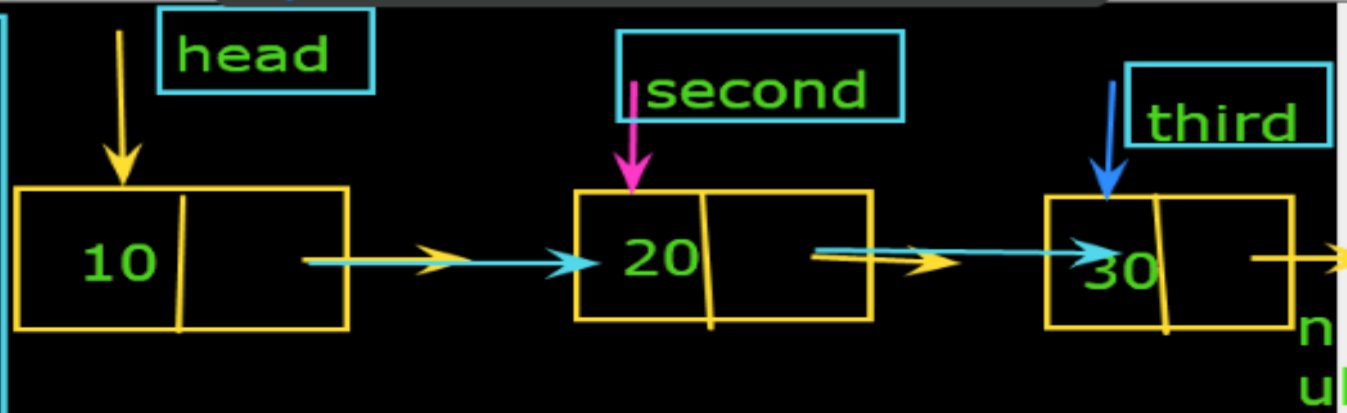
```
    class Node
    {
    int data;
    Node next;
    Node(int d)
    {
        data=d;
        next=null;
    }
    }
```

define Node

```
public static void main(String args[])
{
    LinkedList l1 = new LinkedList();
    l1.head = new Node(10);
    Node second = new Node(20);
    Node third = new Node(30);

    l1.head.next = second;
    second.next = third;

}

}
```

Node & Reference pointer connect

Link establish



Basic Operations

- **Following are the basic operations supported by a list.**
  1. **Insertion** − Adds an element at the beginning of the list.
  2. **Deletion** − Deletes an element at the beginning of the list.
  3. **Display** − Displays the complete list.
  4. **Search** − Searches an element using the given key.
  5. **Delete** − Deletes an element using the given key.