Linear Data Structures:→

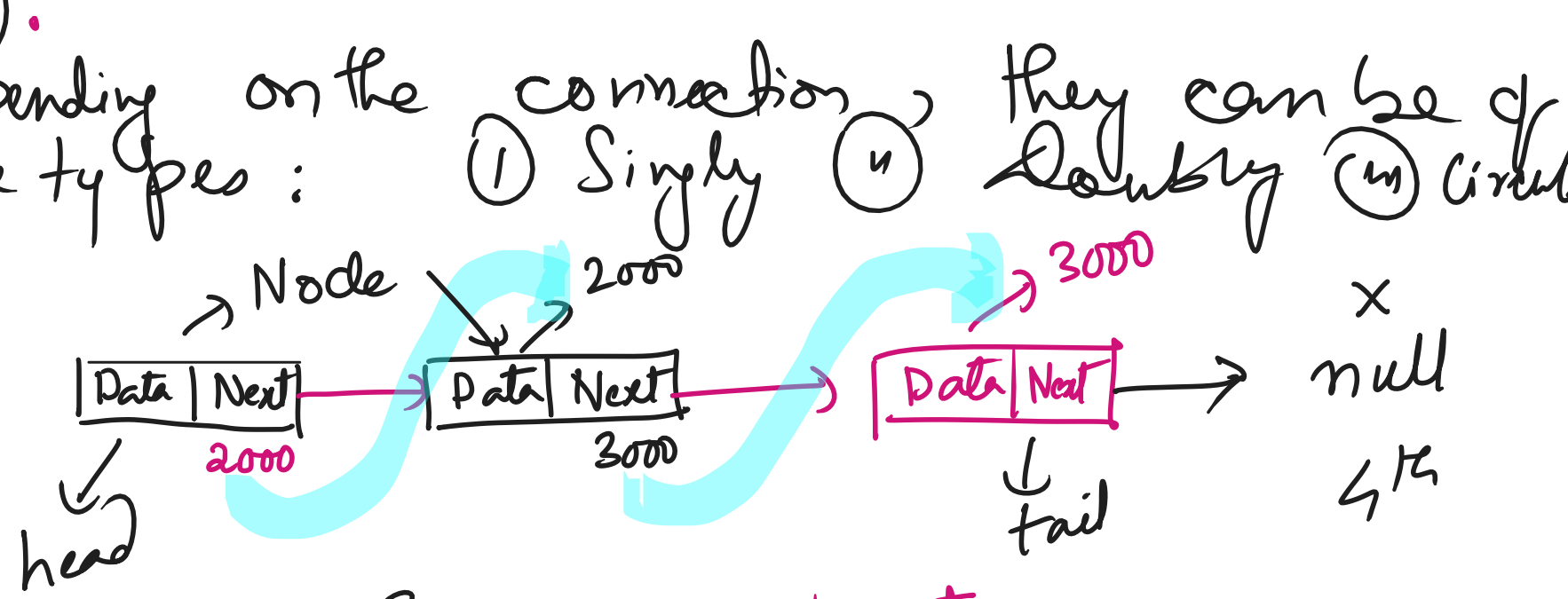
* Arrays ✓
* Stacks ✓
* Queues ✓
* Linked Lists → Singly
  → Doubly
  → Circular

Linked Lists:

① Singly Linked Lists:

* It is a data structure containing entities called nodes. Each node contains some data & pointers to it's neighbouring nodes (prev & next).
* Depending on the connection, they can be of three types: ① Singly ⑪ Doubly ⑪ Circular

B to E




```
class Node {
    int data;
    Node next;
}
```

→ Insert
→ Delete
→ Display

Array ✓

2	8	6	4	1
0	1	2	3	4

arr[2] → indexing 3rd element

TC → search is O(1)

arr[n-1]

O(n): | 1 | 2 | 4 | 5 | (0 1 2 3) insert 3 at 3rd pos, shifted to right → | | 3 | 4 | 5 |

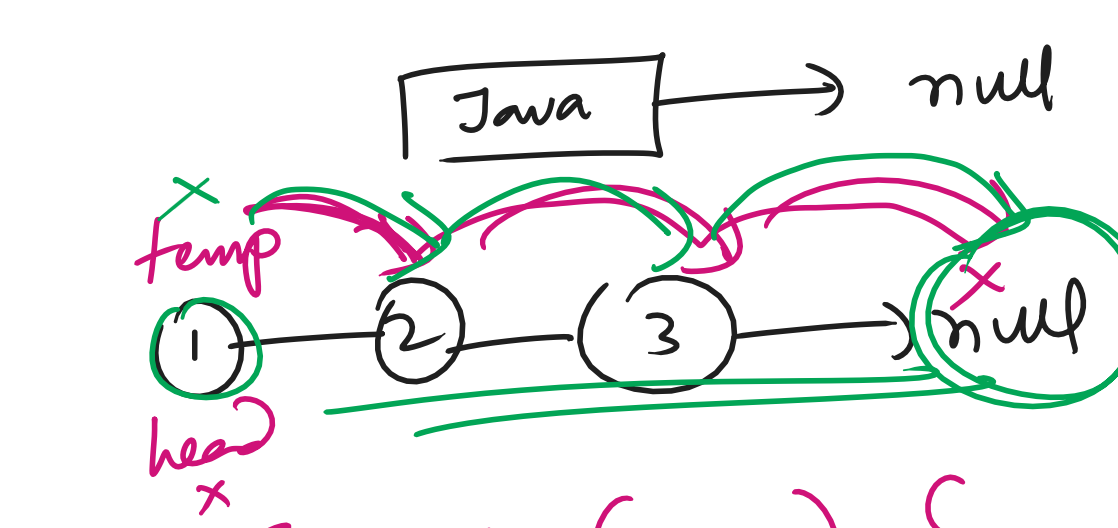
Time / Money — Linked List

① — ② — ③ → null (3rd) O(n)

finite steps O(1): ① → ② → ③ → null, Insert ⓪

Insert: Begin ✓, End ✓, After some node ✓

Delete: Begin, End, Target

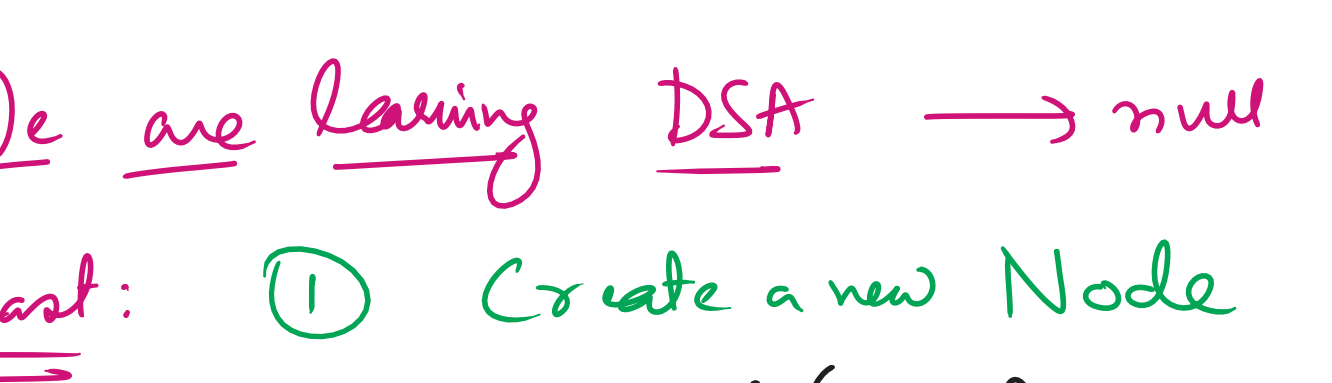


functions() ↓ * methods() ↓ behaviour

① → ② → ③ → null

```
void printListElements( ) {
    Node temp = head;
    While (temp != null) {
        Sout (temp.data + " → ");
        temp = temp.next;
    }
}
```

(null)



String "python"

① Create a new Node

Node newNode = new Node(data);

We are learning DSA → null

Add Last: ① Create a new Node — newNode

ⓐ Empty list

```
if(head == null){
    head = newNode;
    return;
}
```

Not Empty: Last Pos


```
Node last = head;
While (last.next != null) {
    last = last.next;
}
last.next = newNode;
```

head ↓ null — empty

```
① if (head == null) {
    Sout → "can't delete head, list is empty".
}
```

Single: ① → null, h → h = h.next; (this)

① → ② → ③ → ④ → ⑤ → null (h) Multiple elements

* Remove the last node/element:

① Empty we can't delete
  if (head == null)

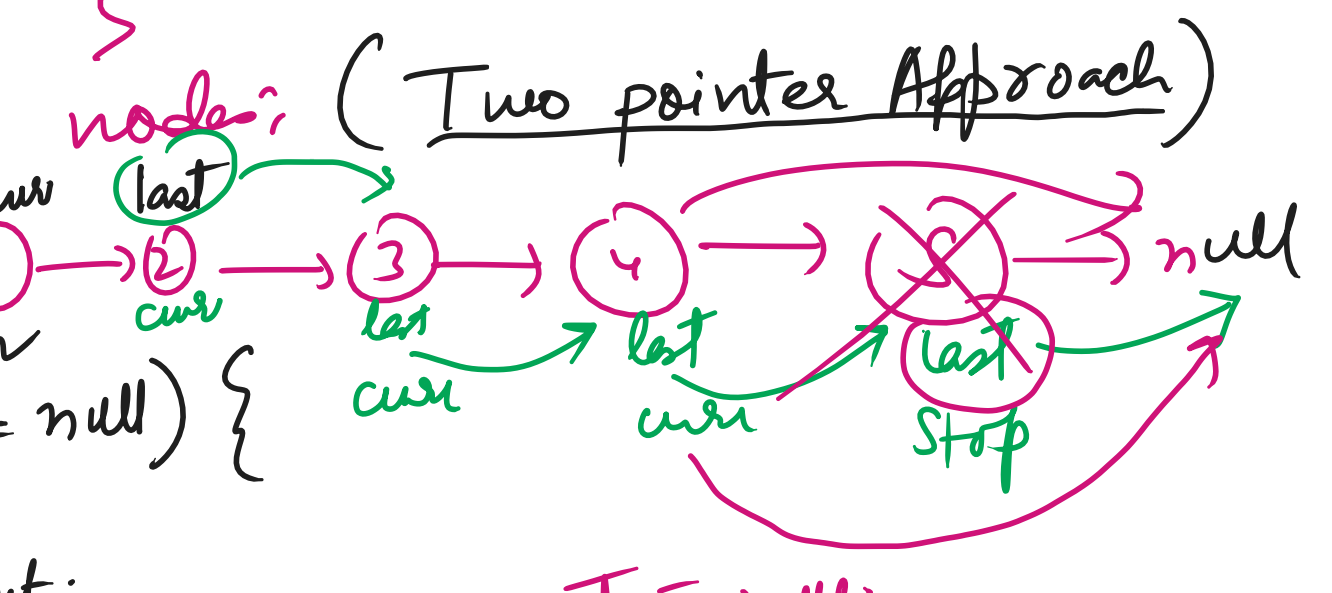
⑪ Single Element: head ① → null

```
if(head.next == null){
    h = null;
    return;
}
```

(JVM)

⑪ Multiple nodes: (Two pointer Approach)

```
Node curr = head;
Node last = head.next;
While (last.next != null) {
    curr = last;
    last = last.next;
}
```

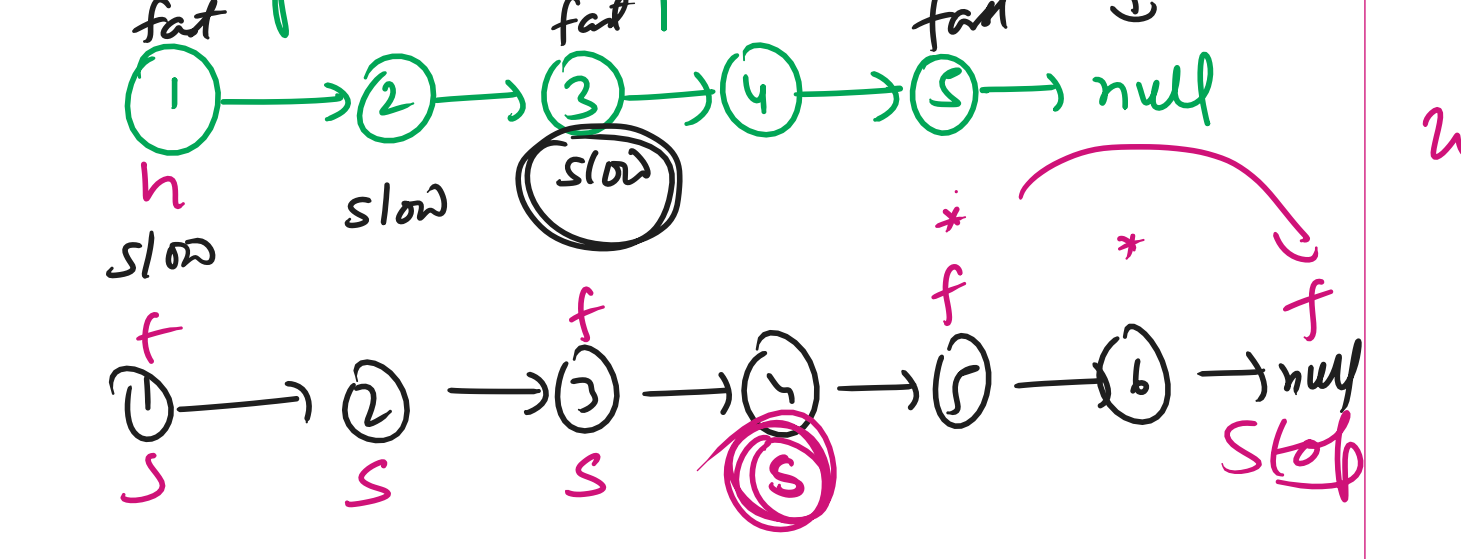


curr.next = null;

① Find the middle node of a linked list
⑪ Delete the middle node of a linked list
⑪ Detect cycle in a singly linked list
Ⓘⓥ Reverse a given linked list
ⓥ Reverse a linked list in groups of k
ⓥⓘ Intersection of two linked lists
ⓥⓘⓘ Sum of two numbers using linked lists
ⓥⓘⓘⓘ Merge two sorted linked lists

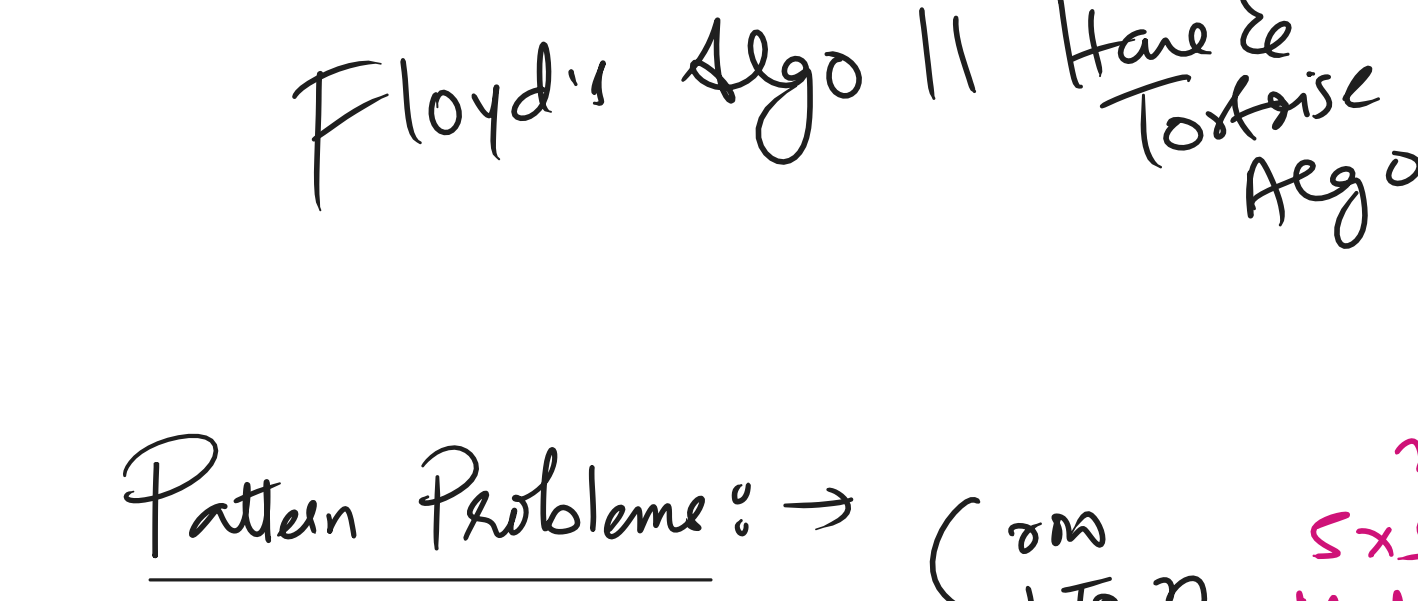
Algo: Floyd's Algo / Hare & Tortoise Algo

Floyd's Algo or (Hare & Tortoise Algorithm)
(Fast & Slow Pointers)




```
Node fast = head;
Node slow = head;
while (fast != null &&
       fast.next != null)
{
    slow = slow.next;
    fast = fast.next.next;
}
return slow;
```

* Cycle Detection in a Linked List:→

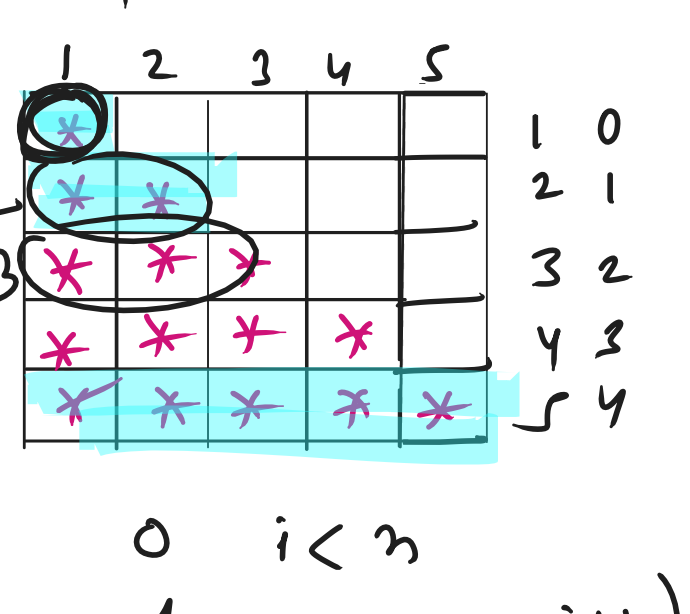


Floyd's Algo || Hare & Tortoise Algo

```
Node fast = head;
Node slow = head;
while (fast != null &&
       fast.next != null)
{
    slow = slow.next;
    fast = fast.next.next;
    if(slow == fast)
        return true;
} return false;
```

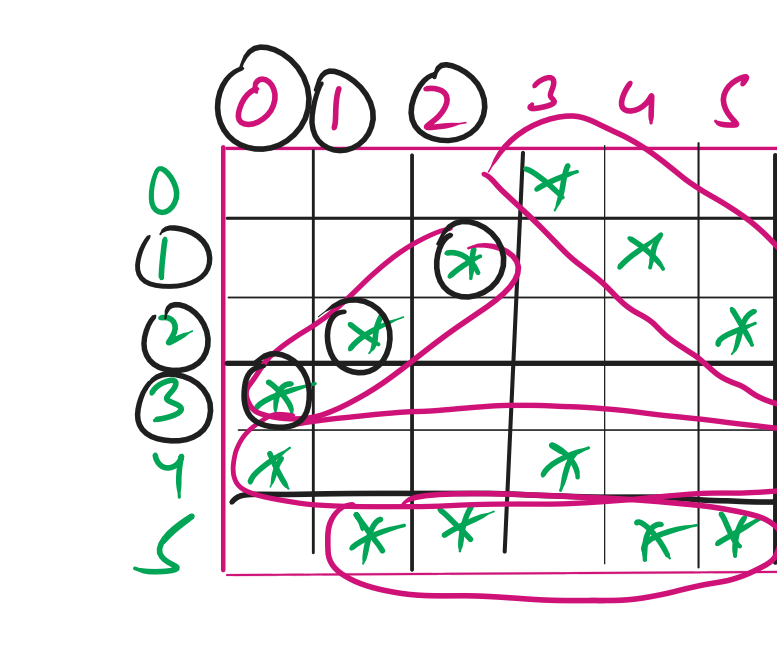
Pattern Problems:→

{ row 1 to n, col 1 to row no } — 5×5 Matrix, n=5, for (row), for (col)



```
for(int i = 1, i<=n; i++)
    for(int j = 1, j<=i; j++)
        sout.print ("*")
```

Heart Pattern



{Bulk Hiring} (Static Pattern) TCS → NQT Ninja, Basic; IBM; Cognizant; Capgemini

① Break into parts
⑪ The parts should be similar

Package	Time
5.5 LPA	5m
4.5 LPA	6m
3.5 LPA	7m
6.8 LPA	4m
7.0 LPA	3.5m
6.0 LPA	4.5m

① r == 0 — c 1, 2, 4, 5 — c % 3 != 0

⑪ r == 1 — c 0, 3, 6 — c % 3 == 0

⑪ r − c == 2
2 − 0
3 − 1
4 − 2
5 − 3

Coders Arcade

*, #, . " "

Ⓘⓥ r + c == 8
2 + 6
3 + 5
4 + 4

c − r == 3
r − c == −3
r + c == 3
r == 4 && c % 3 == 0
r == 5 && c % 3 != 0