Linked List

Array	Linked List → Skip List
1) Continuous Memory Allocation	No continous memory allocation
2) Random access	No Random access
3) Searching	Insertion & Deletion Soequently

Linked List

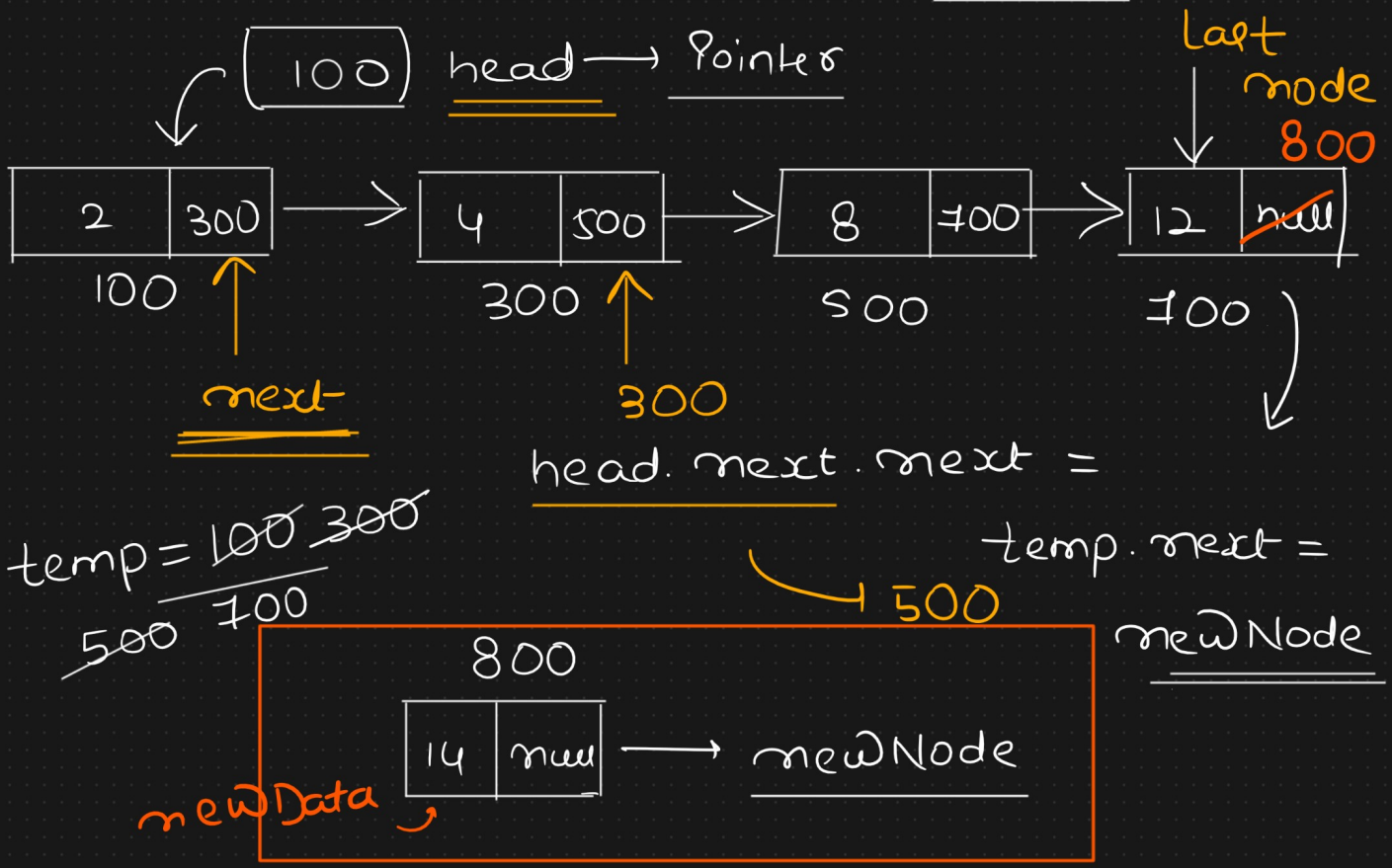
↳ Node

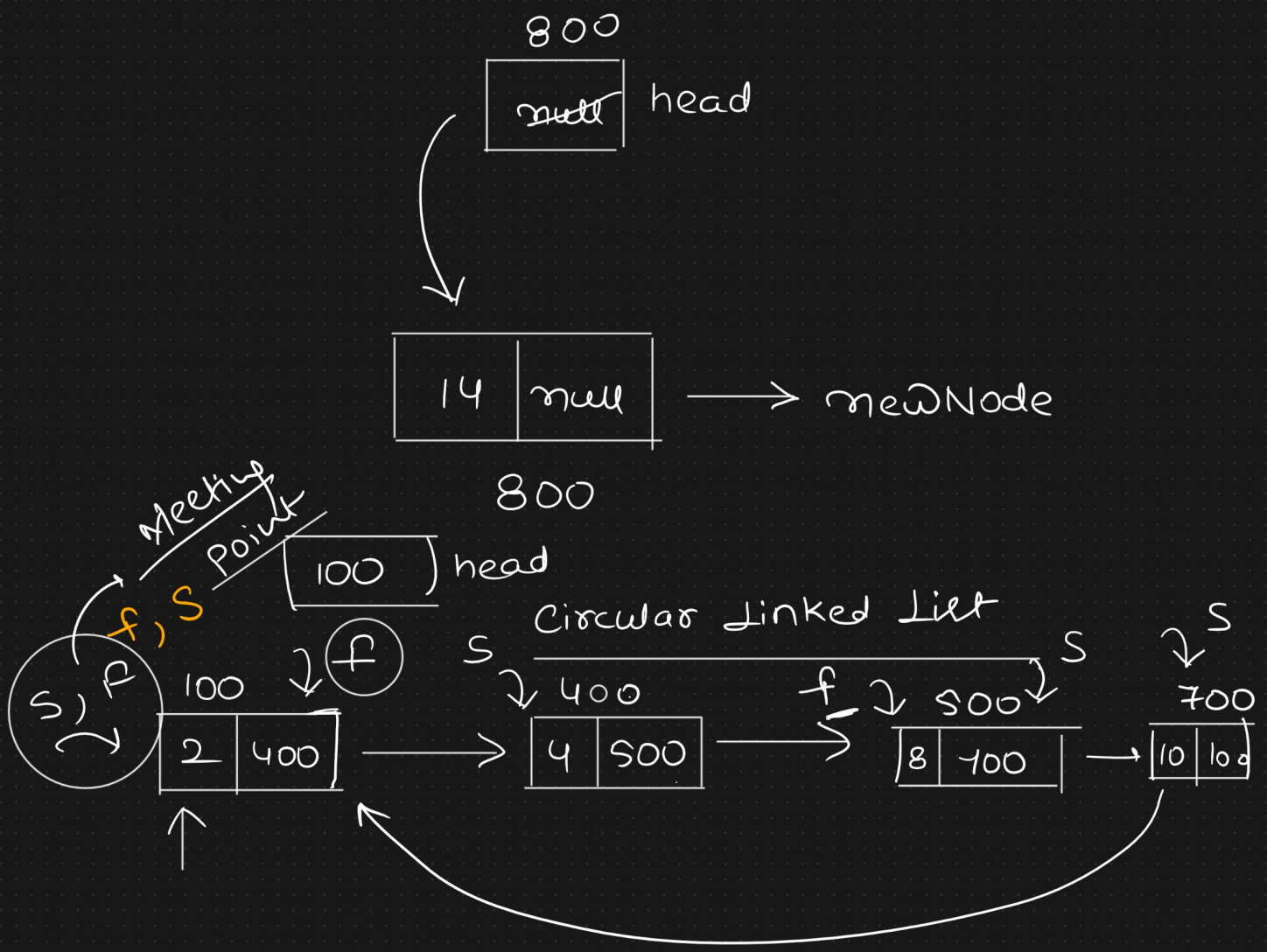
Operations

- Insertion
- Deletion

Interviews

- Reverse
  - Recursive
  - Iterative
- Middle Node of Linked List
- cycle Detection
  - Floyd's cycle Detection Algorithm





slow → 1 Node

fast → 2 Nodes

slow = slow.next

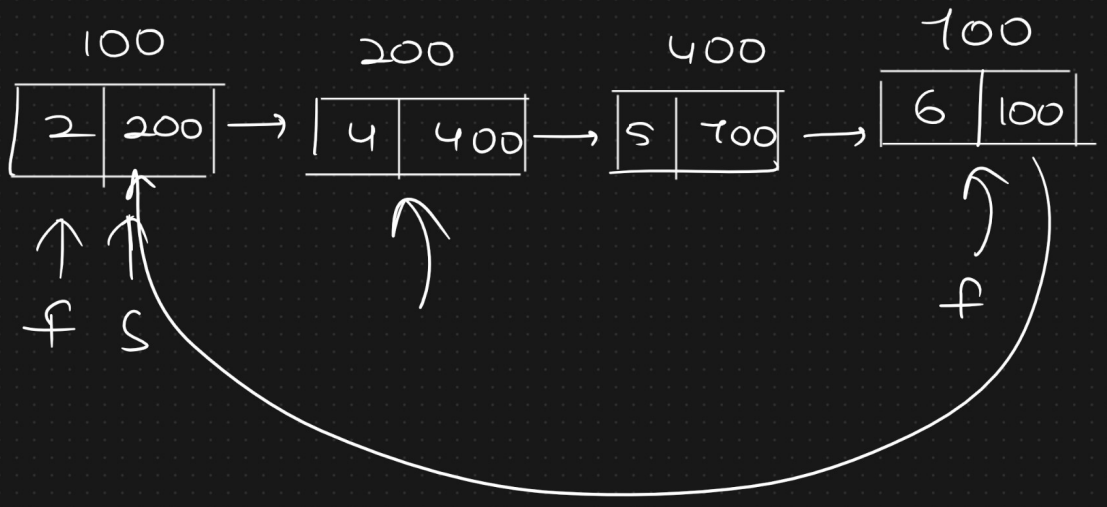
fast = fast.next.next

DSA → Maths

fast → 100

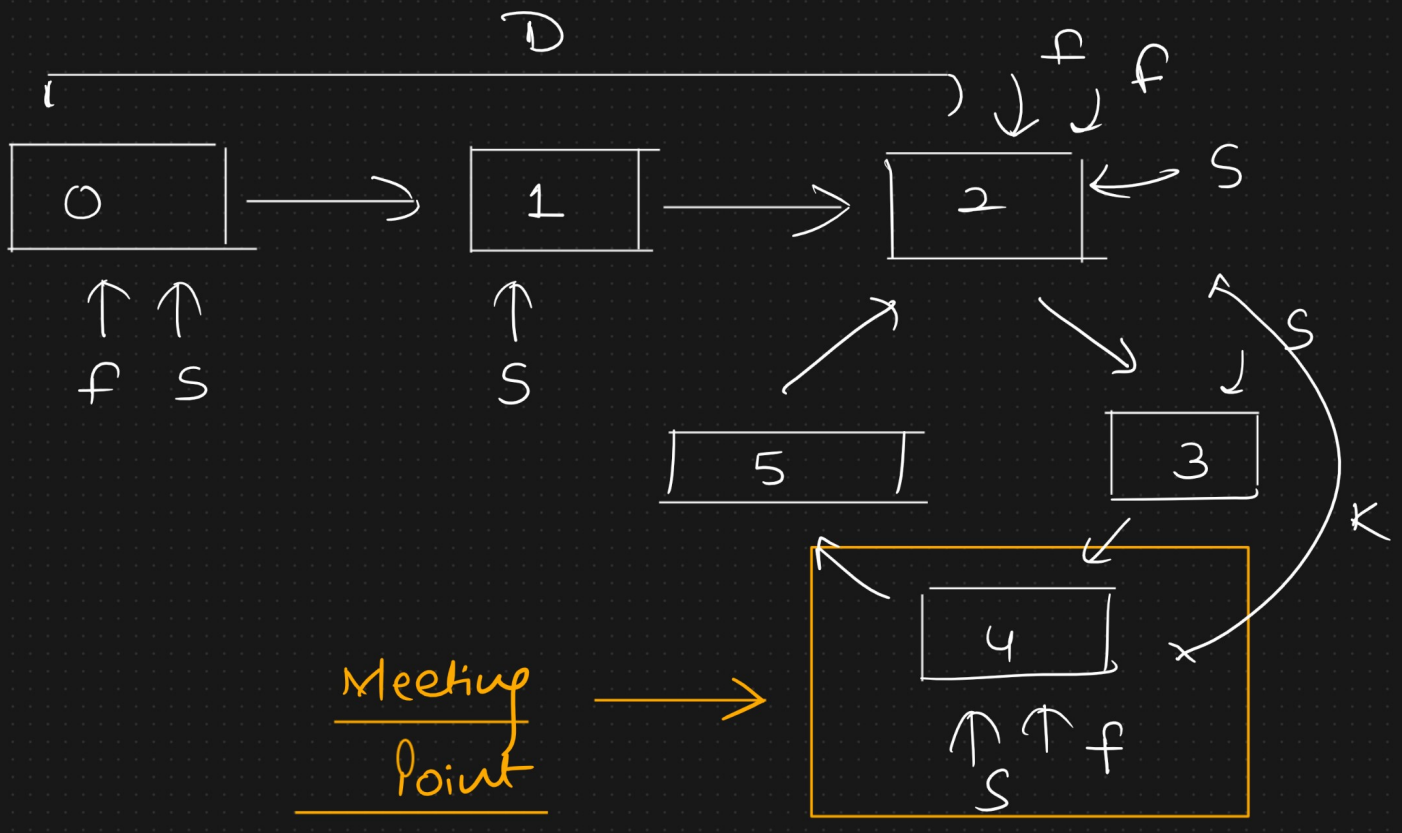
fast.next → 200

100 head



fast = 700

fast.next = 100

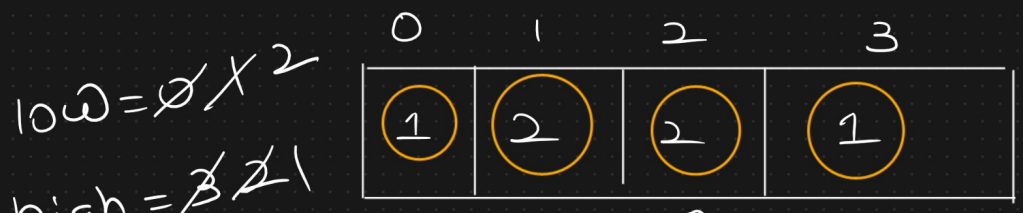


Maths Equation

① $N = D + k + C(i)$ —— slow

② $2N = D + k + C(j)$ —— fast

$N = C(j - i)$



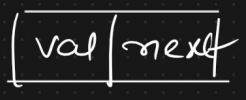
Time Complexity → O(n)

Space Complexity → O(n)

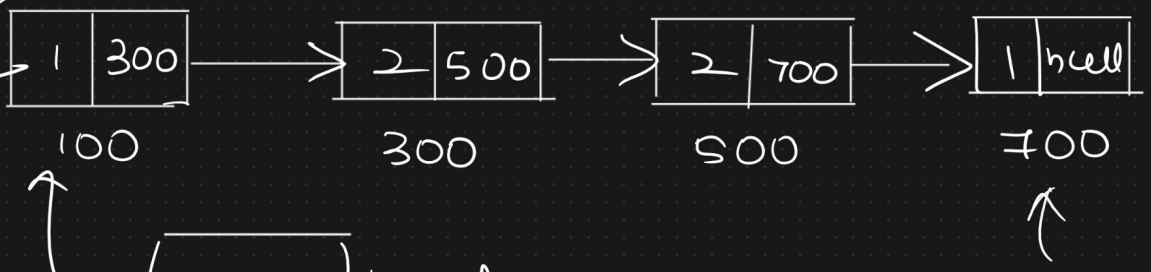
```
while (low < high) {
    if (arr[low] != arr[high]) {
        return false;
    }
    low = low + 1;
    high = high - 1;
}
return true;
}
```

Pseudocode

Static Array	Dynamic Array
↓	↓
Array	ArrayList



Linked List



100 head

null ~~700~~ ~~500~~

curr = ~~100~~ ~~300~~

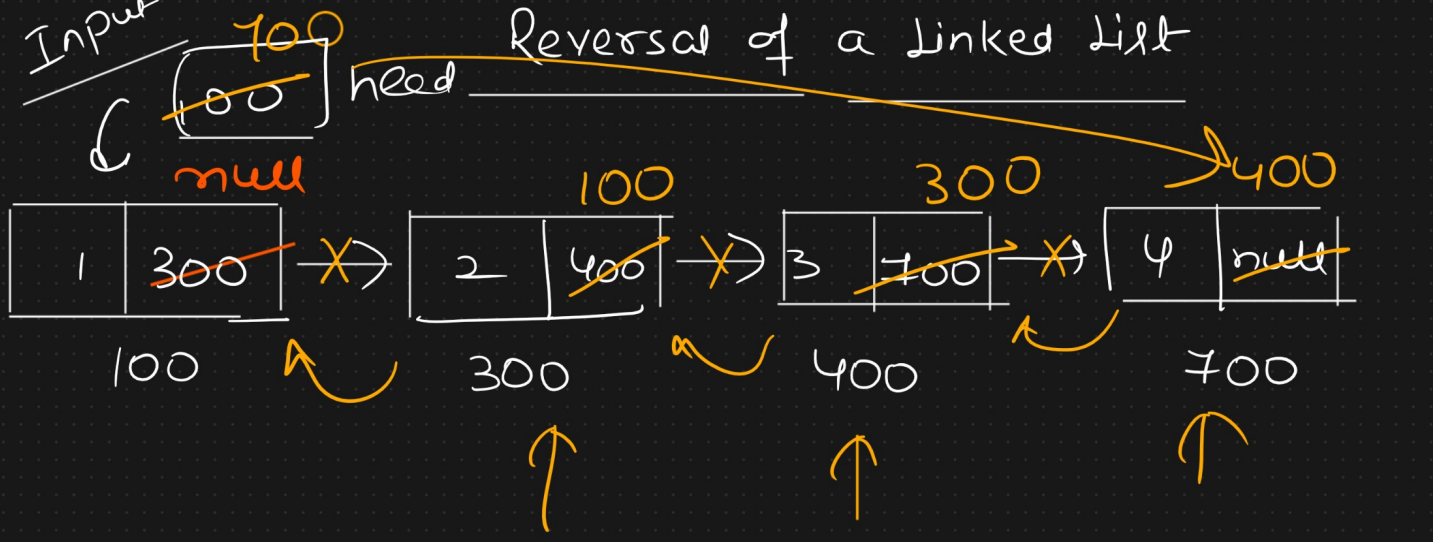
```
while (curr != null) {
    list1.add(curr.val)
    curr = curr.next;
}
```

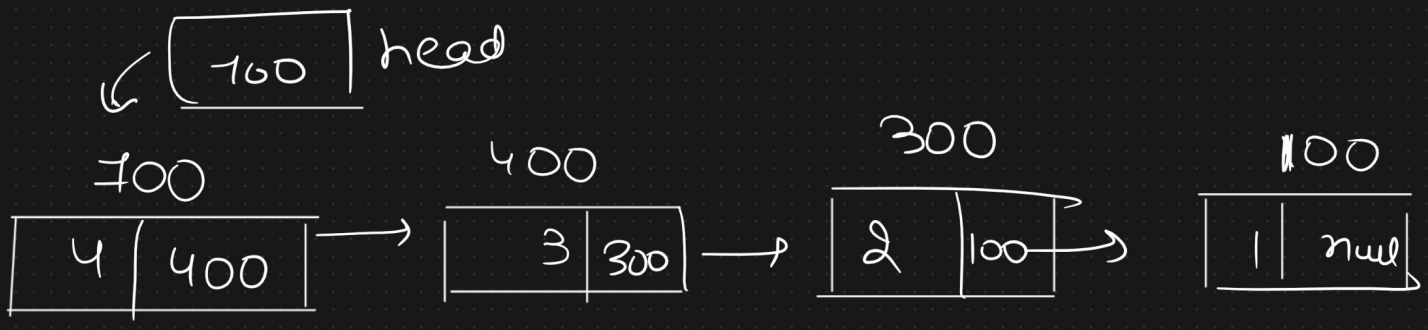
Dynamic Array

list-1

Reversal of a Linked List

Input





Iterative

curr = ~~100~~ ~~300~~ ~~400~~ ~~700~~

prev = ~~null~~ ~~100~~ ~~300~~ ~~400~~

nextPtr = ~~null~~ ~~300~~ ~~400~~ ~~700~~ ~~null~~

while(curr != null){

nextPtr = curr.next —— (1)

curr.next = prev

prev = curr

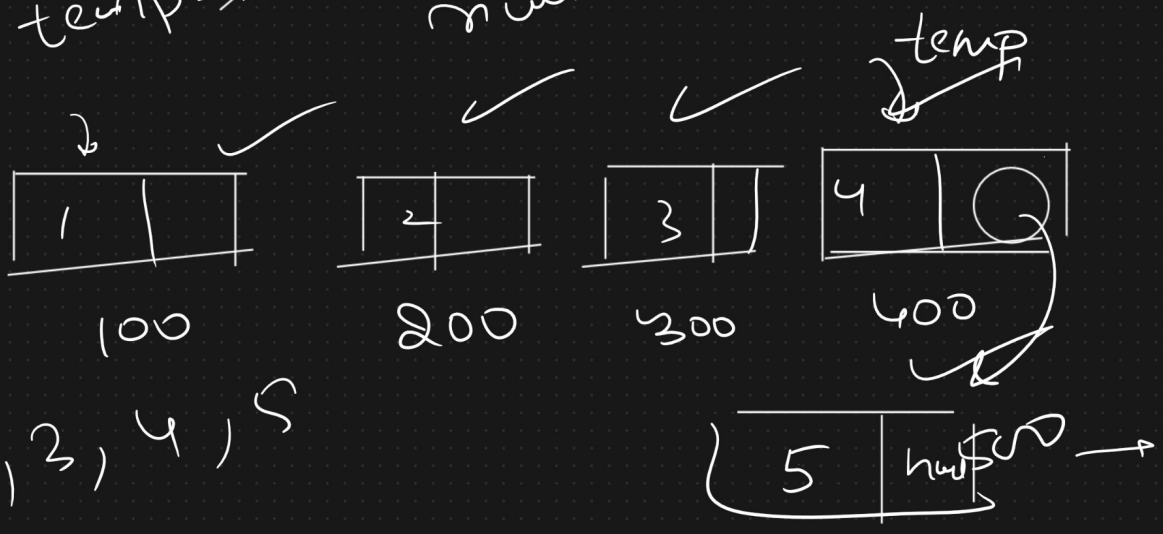
curr = nextPtr

}

head = prev;

1) Reverse Linked List II
2) Palindrome — Time: O(n)
                Space: O(n) → O(1)

temp = ~~100~~ ~~200~~ ~~300~~ ~~400~~ 500



1, 2, 3, 4, 5