Linked List → Node

Data Structure

1) Linear Based data structure
2) Dynamic Data Structure

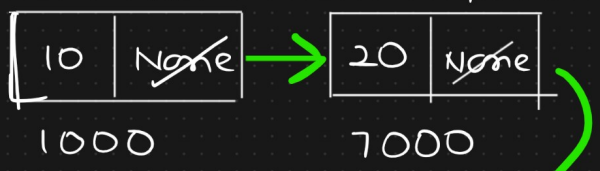
Linked List

1) No continous memory allocation

2) Node → data, Pointer

Pointer → head → 1000

Store the address of other element



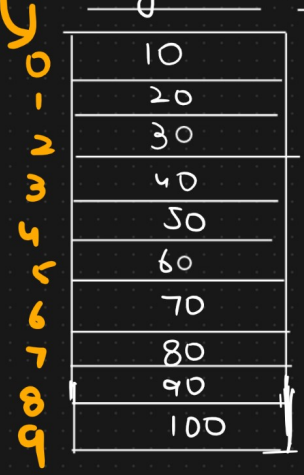
9000 (Single LL)

4 Last node pointer will always be None

Dynamic Array

1) contiguous memory allocation

int → 4 Bytes



n = 10

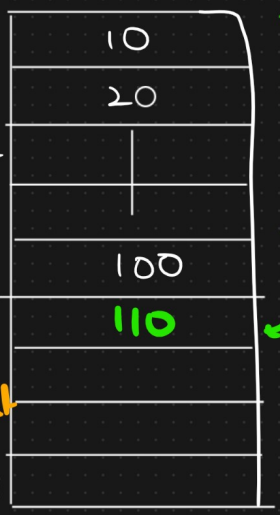
10 × 2 = 20

DA-1

DA-2

n = 20

New data element at 10th index



→ copy from DA-1 to DA-2

3) Random accell is not allowed.

4) Insertion, Deletion (frequent) ↓ Linked List

3) Random accell is allowed.

arr[12] → _____

↑ index number

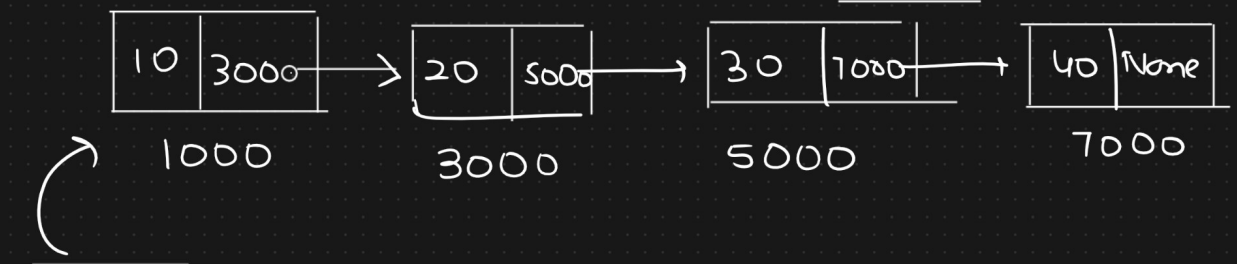
O(1)

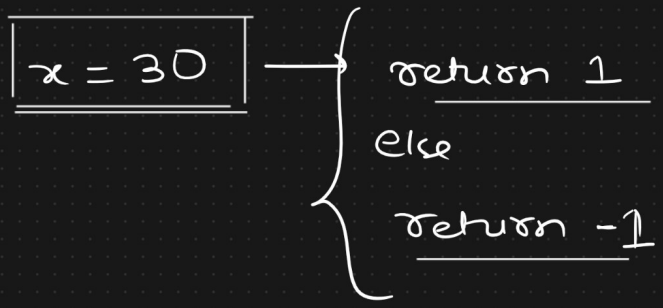
4) Searching operation (frequent) ↓ Array (Random accell)

Types of Linked List

- Singly Linked List
- Doubly Linked List
- Circular Linked List

LL is sorted in nature





Can we apply binary search??

mid = i + (j - i) // 2

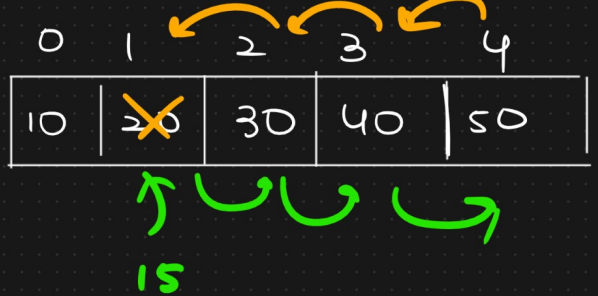
Random access → Not possible

Searching (Linked list) ⇒ $\Theta(n)$

↳ Traverse the array completely

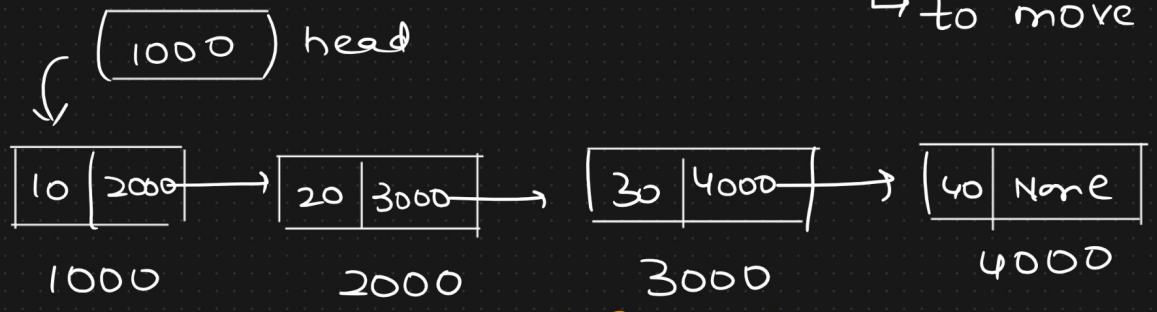
Advance Data Structure

↳ **Skip List**



Singly Linked List → One Pointer

↳ to move forward

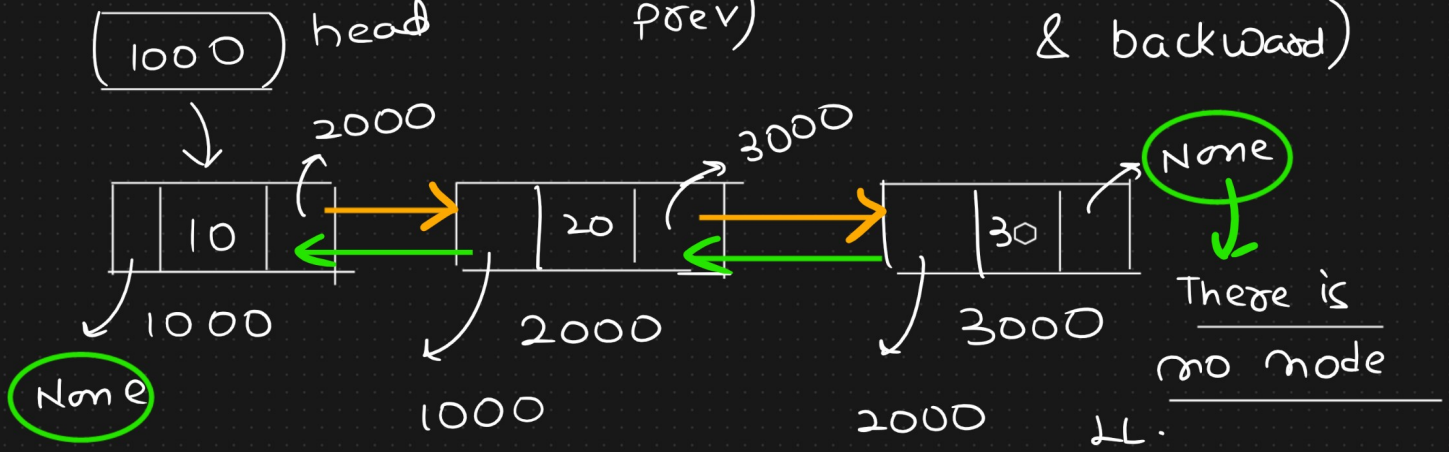


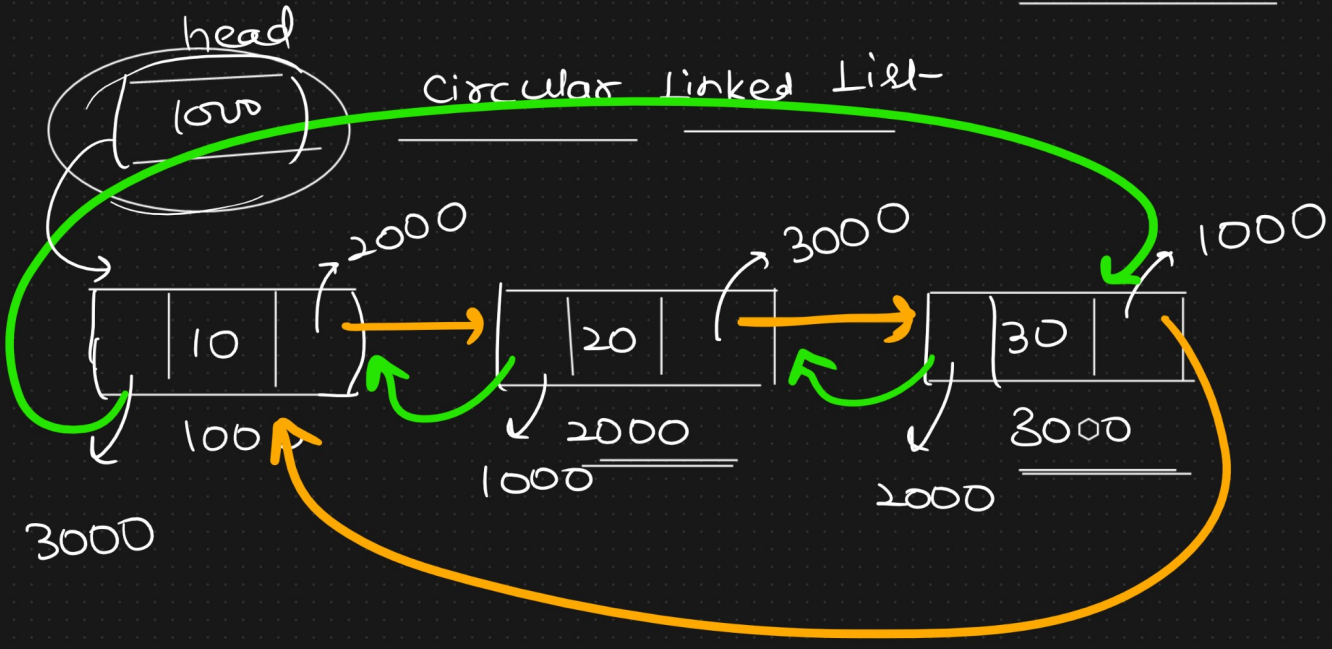
↳ Spotify

Doubly Linked List → 2 Pointers

(Next & prev)

(Move forward & backward)



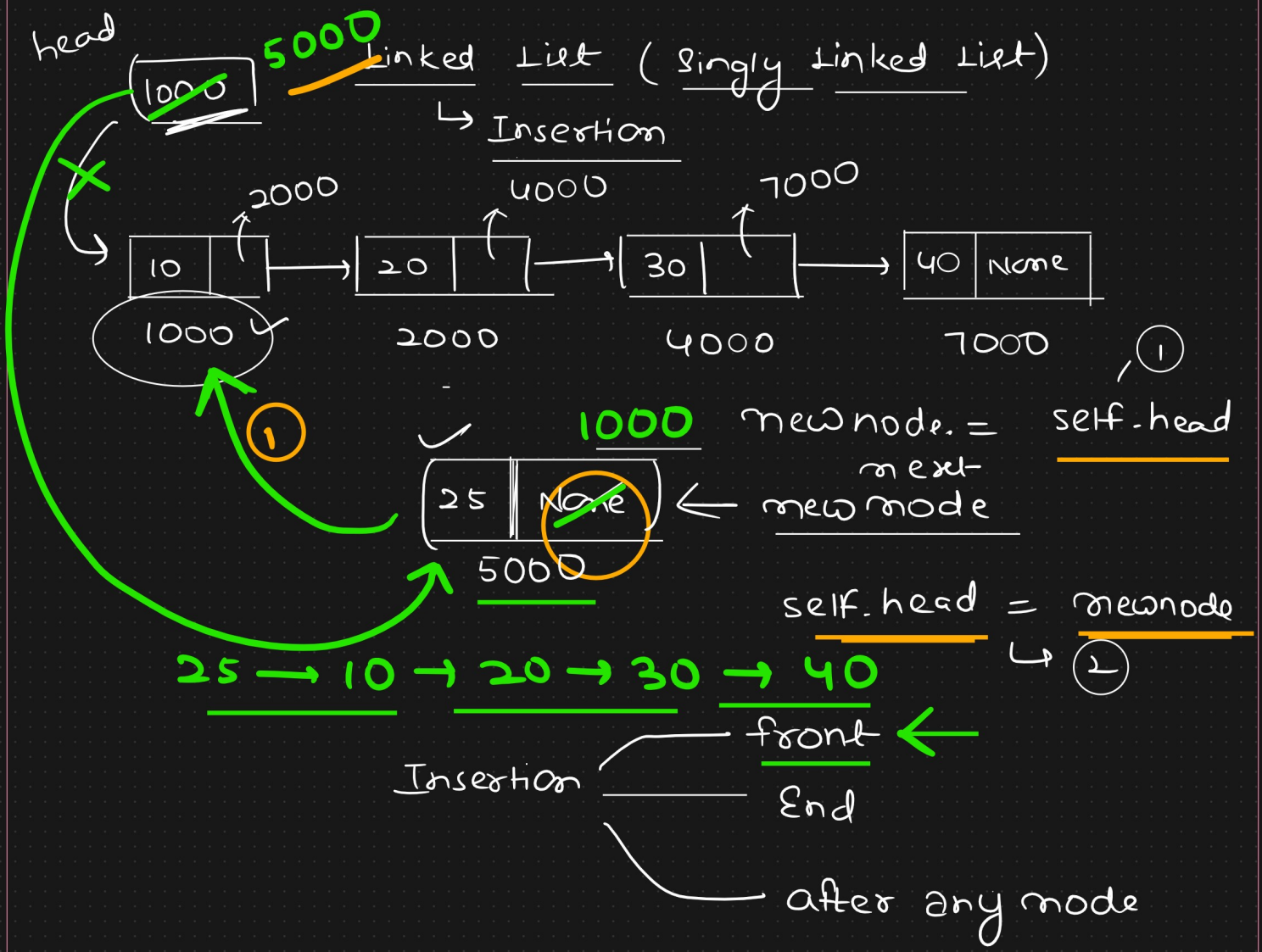


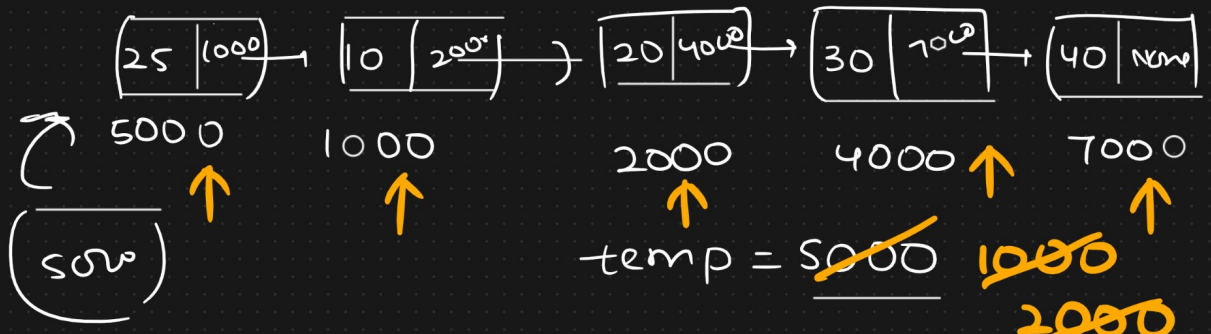
Pointer → address

next — prev

next → responsible to store addr of next node

prev → responsible to store addr of prev. node.



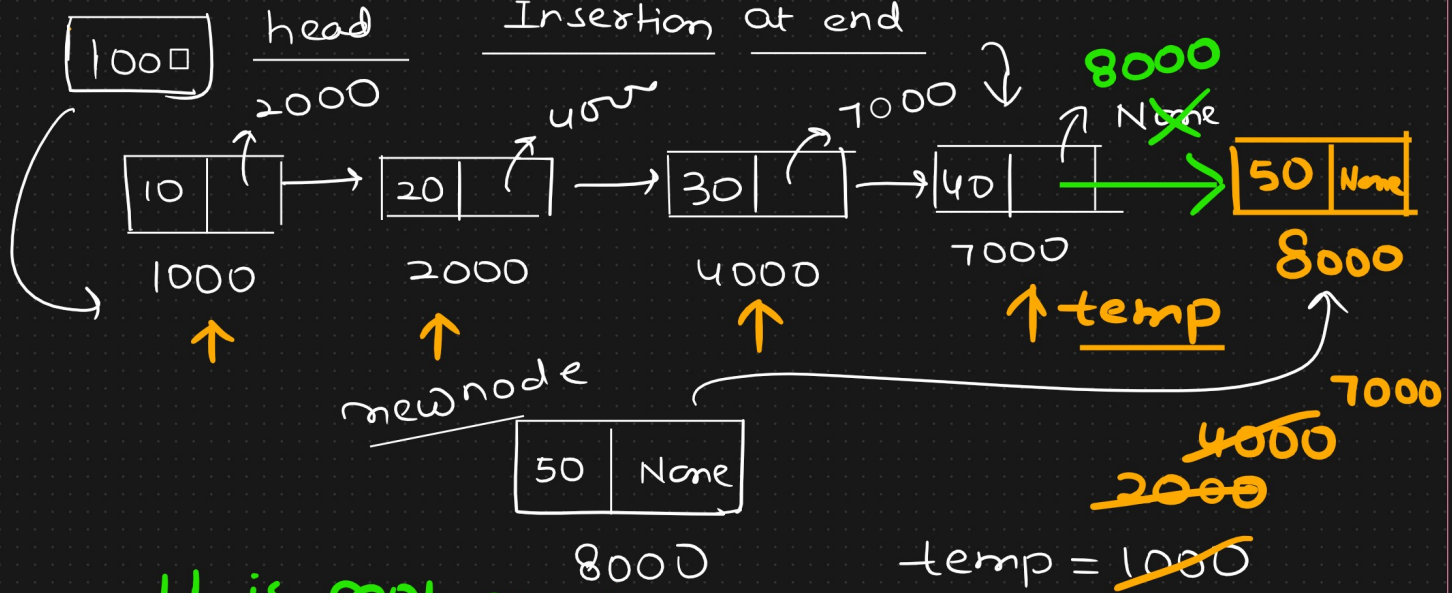


Traversal of LL

```
temp = self.head
while temp:
    print(temp.data)
    temp = temp.next
```

Not None

25, 10, 20, 30, 40

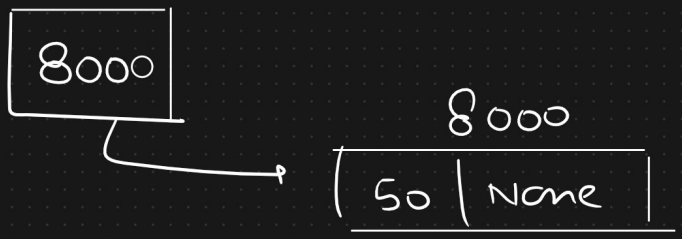


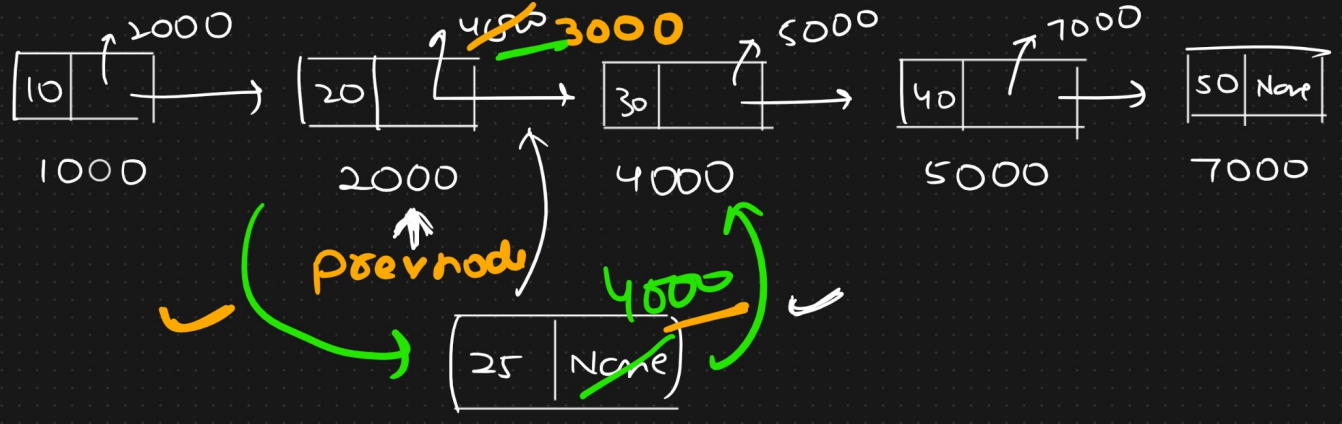
LL is not empty

```
temp = self.head
while temp.next is not None:
    temp = temp.next
temp.next = new_node
```

LL empty

```
if self.head is None:
    self.head = new node
    return
```





new node

prev_node

insertAfternode(llist . head . next , 25)

newnode.next = prev_node.next

prevnode.next = newnode