→ Students Exams

- → Marks : Total Marks
- → Rollno : —
- → Name : —

→ Say N Students

- → Marks [N]
- → Roll [N]
- → Name [N]

| Index : | 0th | 1nd | 2nd | 3rd |
|---|---|---|---|---|
| | 97 | 65 | 35 | 75 |
| | 14 | 24 | 31 | 33 |
| | Rashika | Saidutt | Viraj | Priyali |

→ 3 arrays

→ User defined datatype } → Syntax of Class Creation

```
class Student {
    int rollno ;
    String name ;
    int marks ;
    Student(int ro, int ma, String na){
        rollno = ro   marks = ma
        name = na
    }
}
```

// constructor

1) → Constructor name should be exactly same as Class name

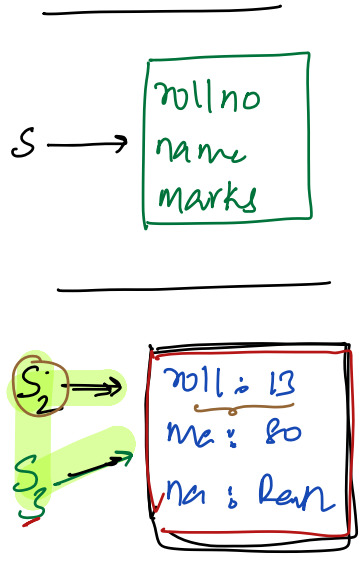
2) Constructor is exactly like a function but no return type.

main() {

Student s = new student()

| | In C++ | Python |
|---|---|---|
| s.rollno | | |
| s.name | → | • |
| s.marks | | |

In Java.




s.rollno = 75
s.name = "Sarduth"
s.marks = 80

Student $s_2$ = new Student(75, 80, "Rathika")

print($s_2$.marks) // Wether data initialized

Student $s_3$;

$s_3 = s_2$

print($s_3$.marks)

- Most commonly data structure

  → Arrays ?

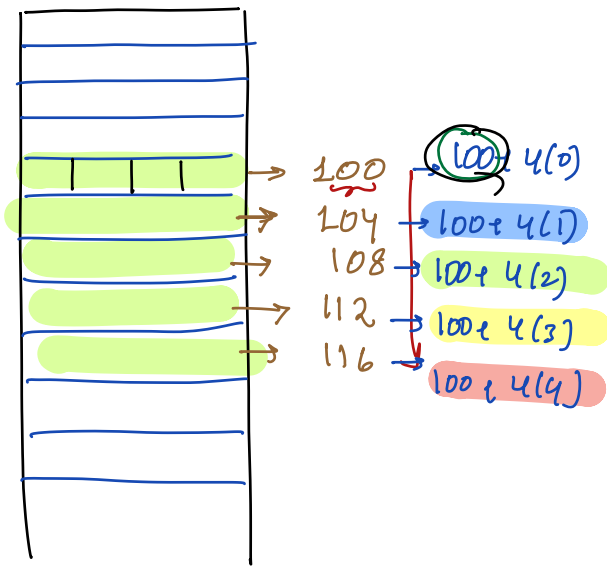
    ↳ $i^{th}$ index → ar[i] → O(1) ?

→ int ar[5] ;

↳

It will allocate continous

Blocks of memory



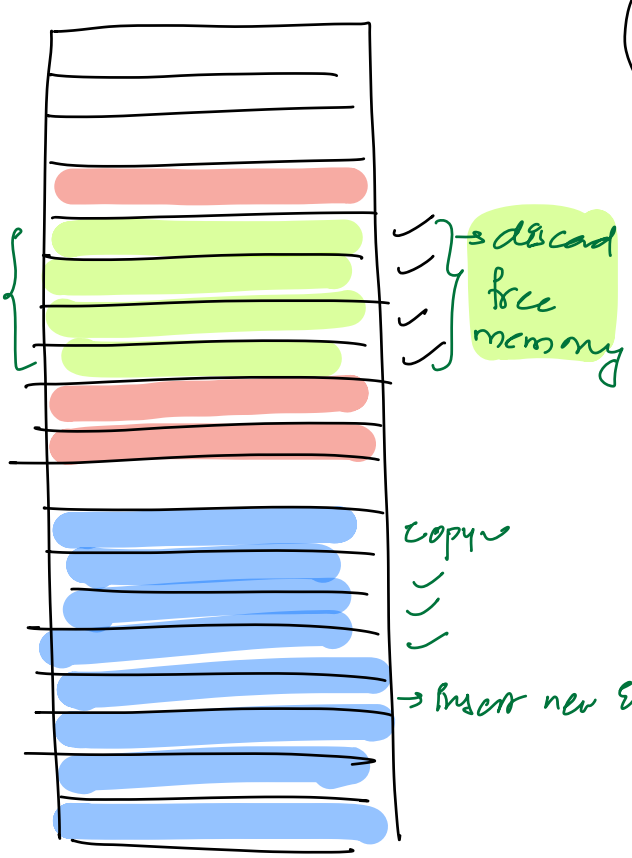

→ Insertion in a List → O(1)

↳ average

⇒ Dynamic arrays

⇒ { arrays }

int a[4] ⇒ } 4 Elements fetched




int $a_1[8]$ =

→ copy $a[4] \rightarrow a_1[8]$ ⇒ O(N)

→ Insert new element in $a_1[8]$

int a[N]

⇒ All N operations

int $a_1[2N]$

N copy → $a[] \rightarrow a_1[]$ : O(N)

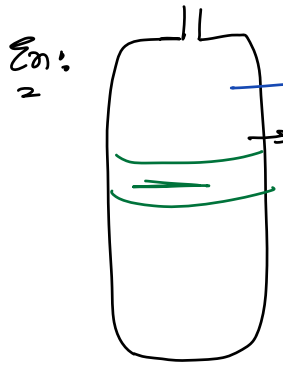
add new Elem $a_1[]$

Inbuilt Implementation

1 time investment

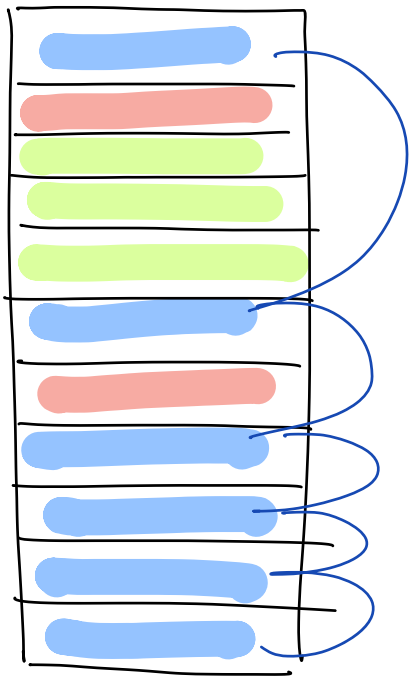
Aquaguard : 10,000

Eg: 2



→ 20 Litre : 100

→ weekly twice
n Thrice



Dynamic array

int ar[3]

3 blocks

=

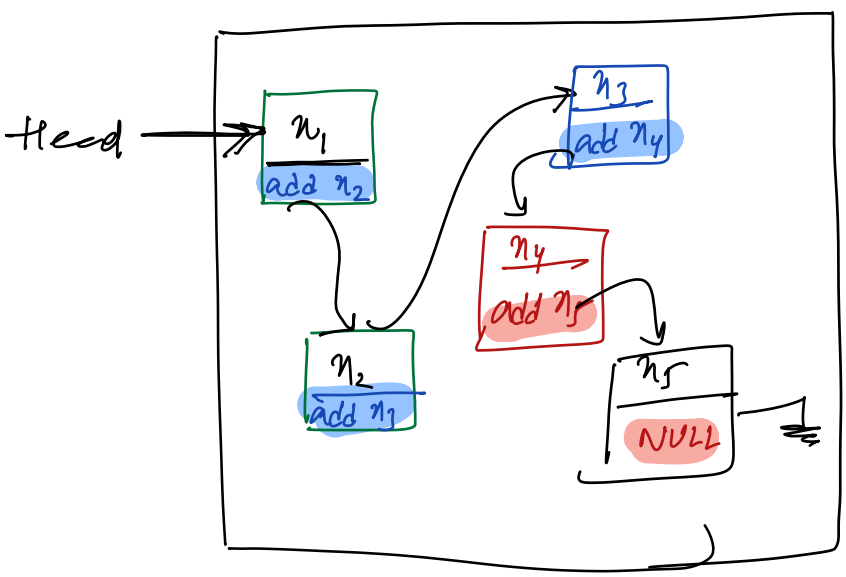
Put b[6] = 6 continous Blocks

⇒ you want to properly use memory,

& you comprise O(1) acc time ?

// & LinkedList

→ We can use random locations



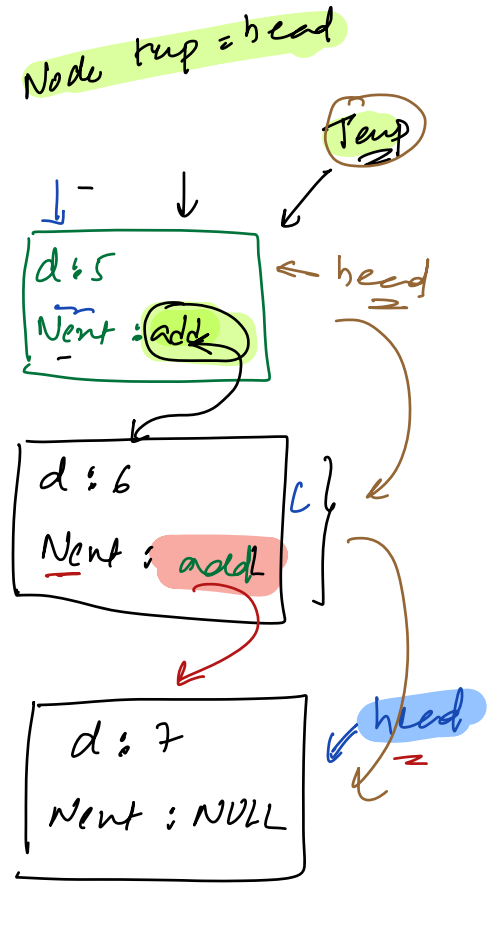

We should

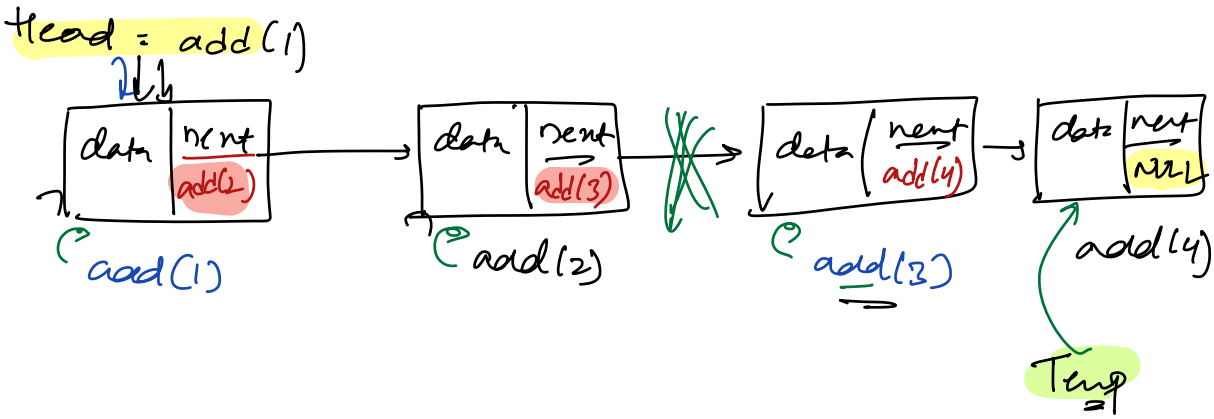
```
class Node {
    int data
    Node next // we hold address of next node
    Node(int n) { -> constructor
        data = n
        next = NULL
    }
}
```

Keyword

```
main() {
    Node head = new Node(5)
    head.next = new Node(6)
    head = head.next
    head.next = new Node(7)
    head = head.next
}
```

Note: Don't lose head node address






Temp = add(1)

Temp = Temp.next] (add(2))

Temp = add(2)

Temp = Temp.next] add(3)

Temp = add(3)

Temp = Temp.next] add(4)

Temp = add(4)

Temp = Temp.next) NULL

Temp = NULL

End game
no more nodes

class Node {

{ data can be anything
float / char / String / arr[] }

int data

Node next // we hold address of next node

Node(int n) { → constructor

data = n

next = NULL
↳ keyword

}

N>0

Node createLinkedList(N){ → first node address

N=3: 1 → 2 → 3 → NULL

N=5: 1 → 2 → 3 → 4 → 5

Just return head node

10:50 pm

memory allocated

Node head = new Node(1);

Node temp = head

for(int i=2; i<=N; i++){

head.next = new Node(i)

head = head.next

}

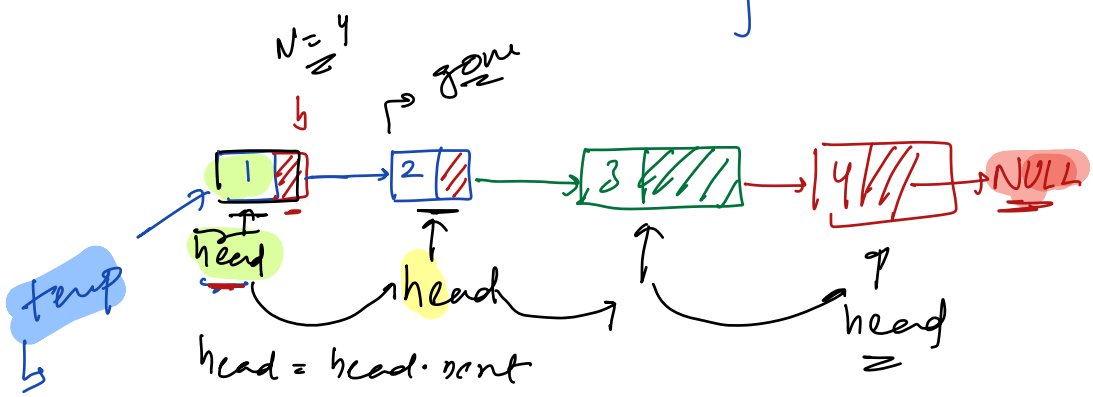
return temp;

}

N=4

i=2 3 4

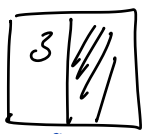
i) h.next = new Node



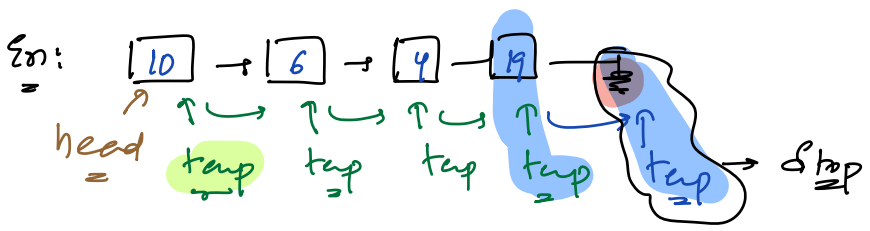

temp is pointing to first node

// Node temp;  → are used to hold address

temp = new Node(3)  → (memory)




// int sizeL(Node head)  → head node address




Node temp = head ; {good practice}

while( temp != NULL){  → if temp == NULL: break

temp = temp.next ; {update temp}

c++

}

return c;

}

temp = NULL → {Nothing}

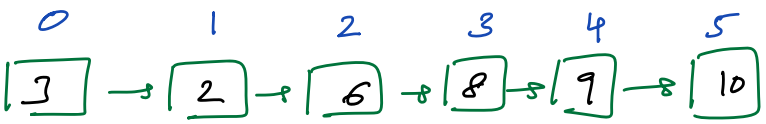
{temp.data, temp.next} → (NULL Pointer Exception) → Error

Edge Case

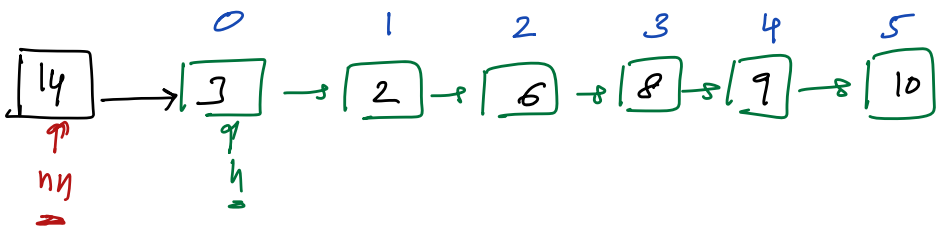
— Insert at k( Node head, Int n, Int k)

(k: 0 based indexing; n: at that index)

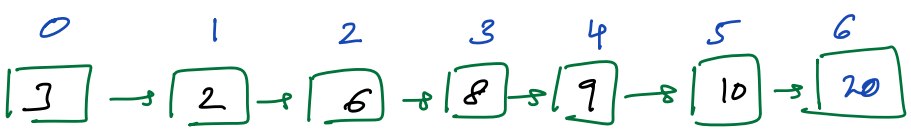
Eg:



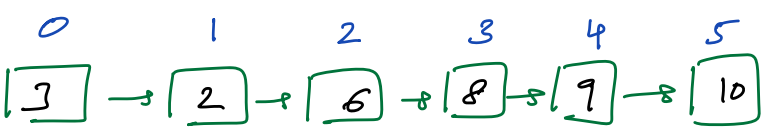

Case: k=0, n=14




Case 2: k=6, n=20



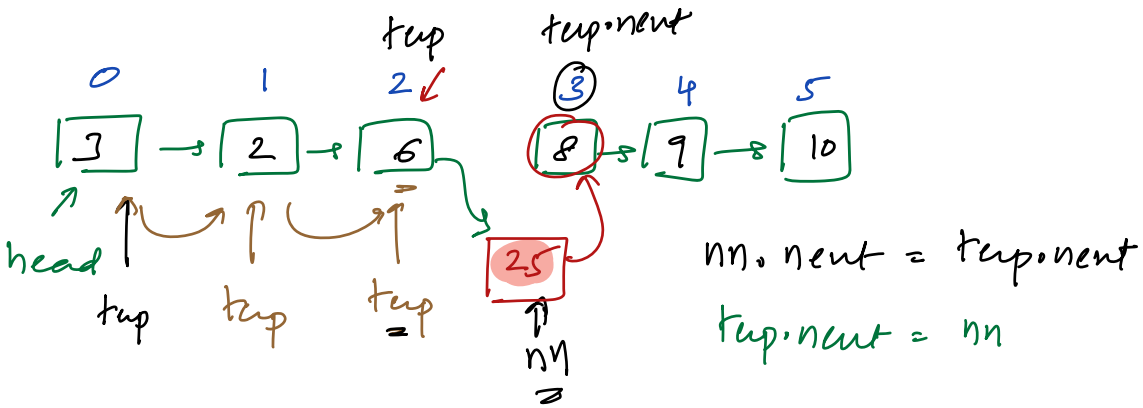

Case 3: k=7, n=14 : If k > (len of linked list) → cannot



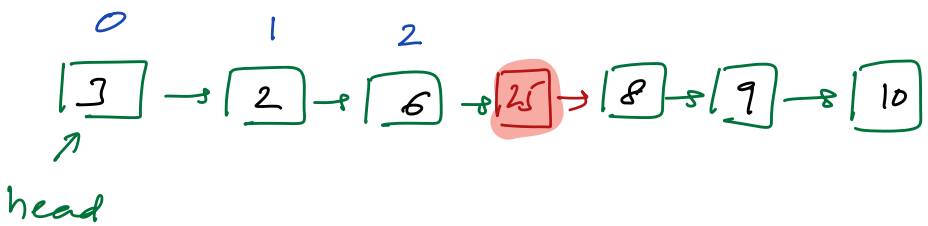

}

Case 4: k = 3, n = 25

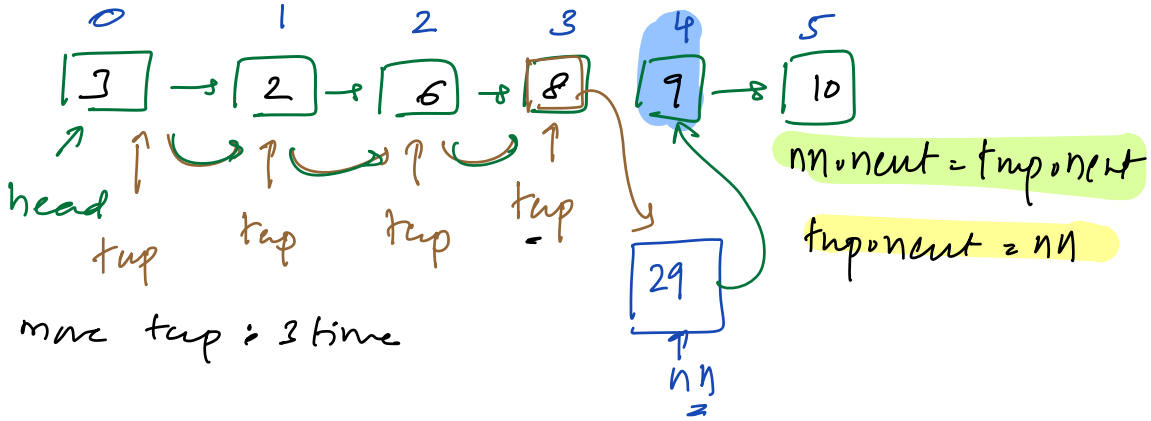
Node nn = new Node(25)




Note: Take temp till prev address

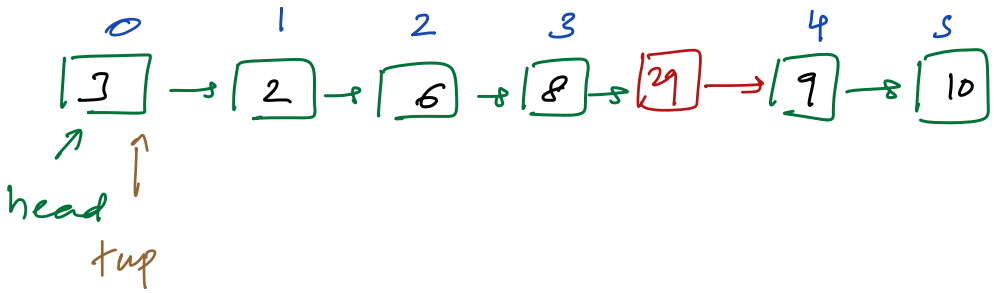


nn = new Node(29)

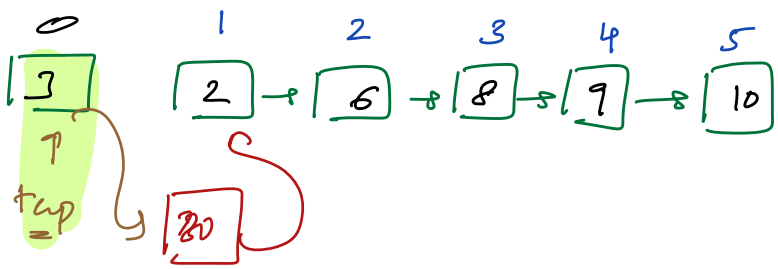



k = 4 → move tmp 3 time

x = 29

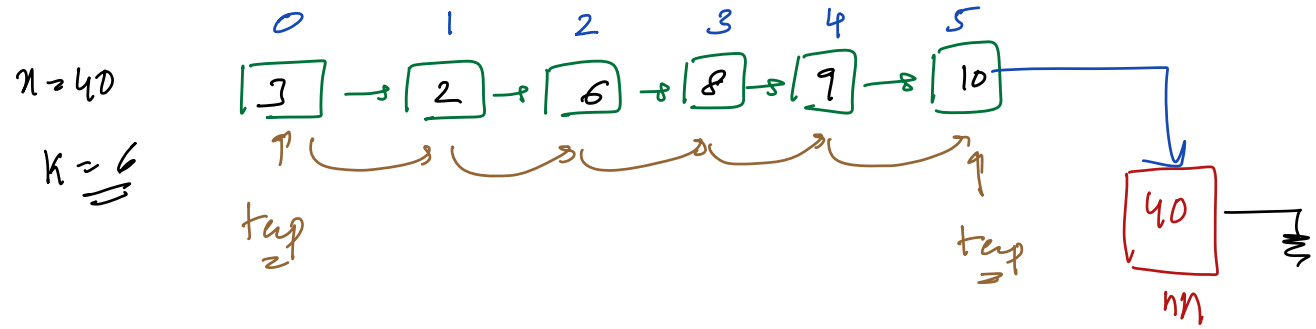


```
insertkpos( Node head, int dt, int k){

if( k > size(head)) {return head}

Node nn = new Node(dt);

if( k==0){
    nn.next = h; h = nn; return h
}

Node temp = head;
for(int i=1; i<=(k-1); i++){
    temp = temp.next
}

nn.next = tmp.next
tmp.next = nn
return head

}
```




n = 20, k = 1

Iterate (k-1)




nmoment = temp.next

nmoment = NULL

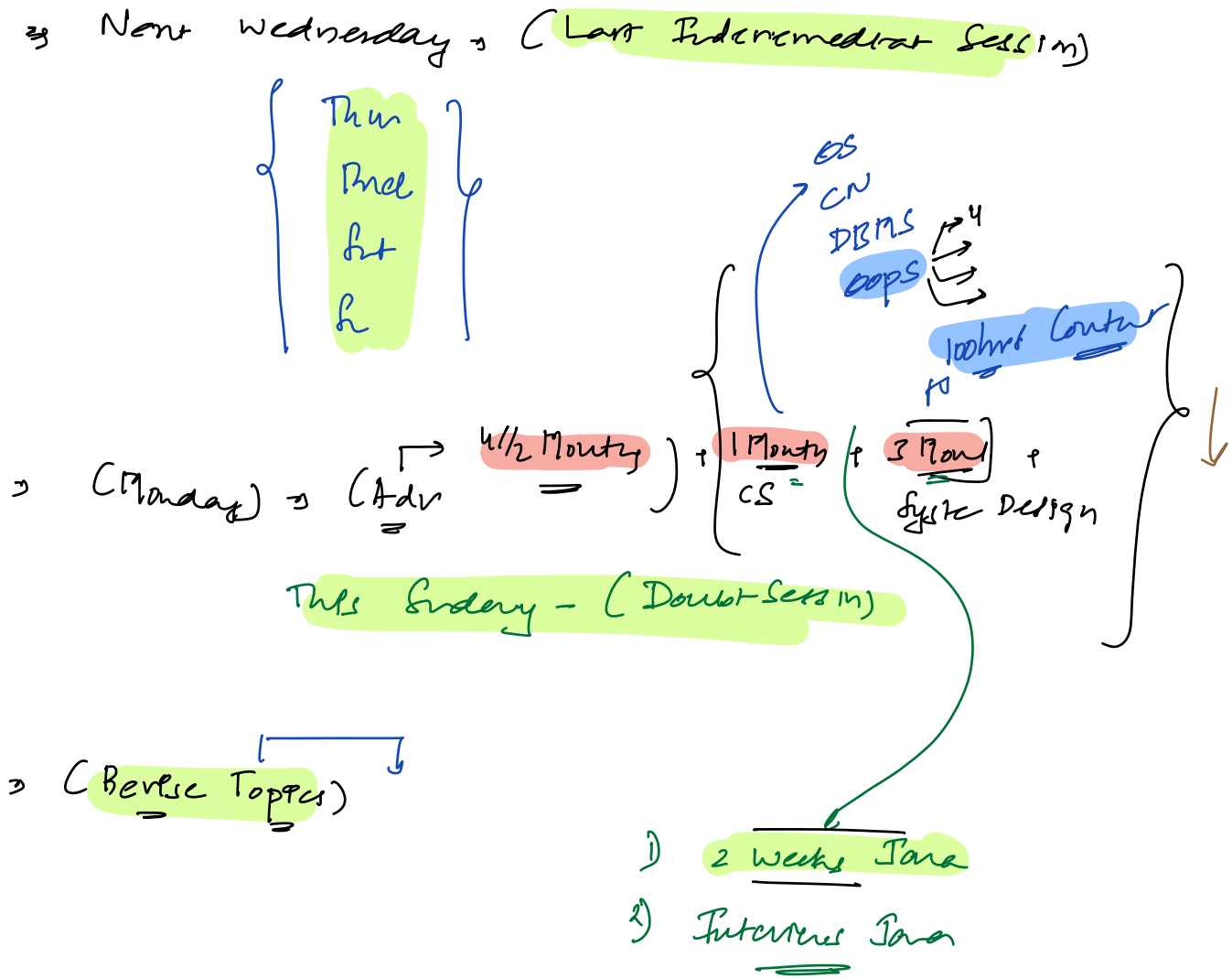
temp.next = nn

→ Delete k^th pos → (TODO)

Advanced Content → 4 Sessions

1) →
2) insert
3) del

Fridays → Stack → 3/4
Tre-1
Tre-2
→ Hrs


Next Wednesday → (Last Intermediate Session)
Thu
Fri
Sat
Sun
OS
CN
DBMS
OOPS
4
Monday → (Adv
4½ Months
1 Months
CS
3 Months
System Design
100hrs Content
10
This Sunday - (Doubt Session)
Basic Topics)
1) 2 weeks Java
2) Interview Java