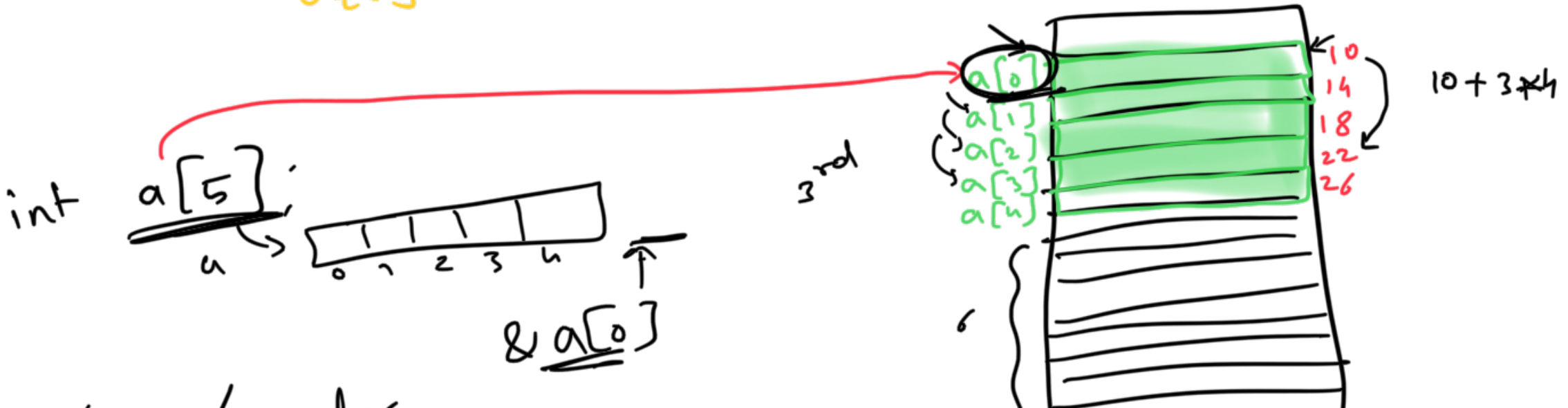
Linked List ..

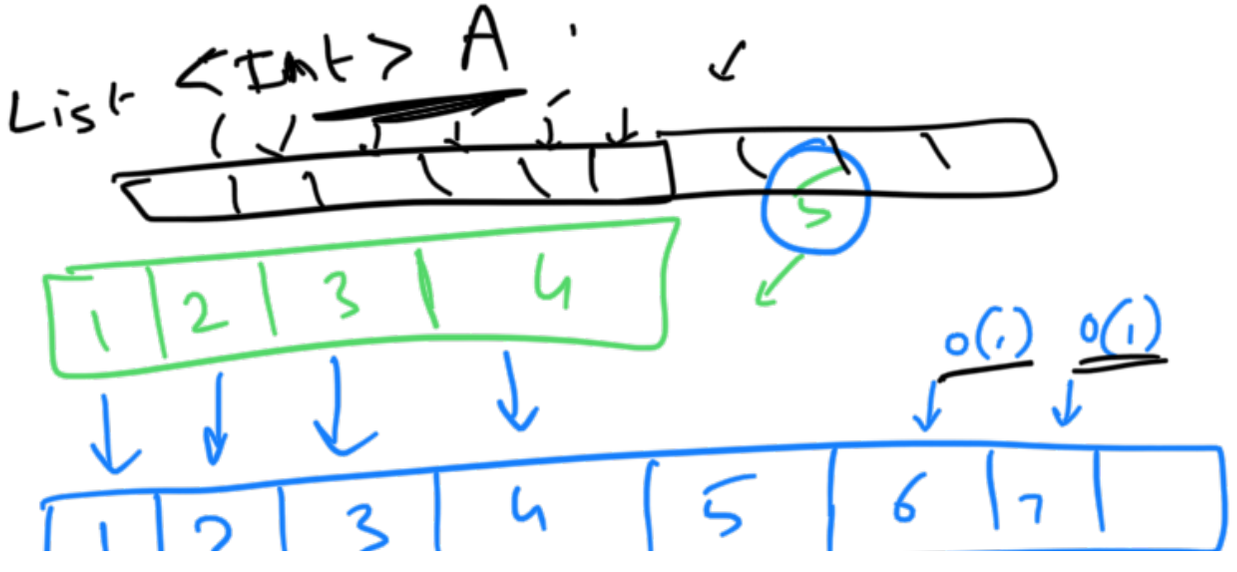
Arrays $\Longrightarrow$ O(1) random access

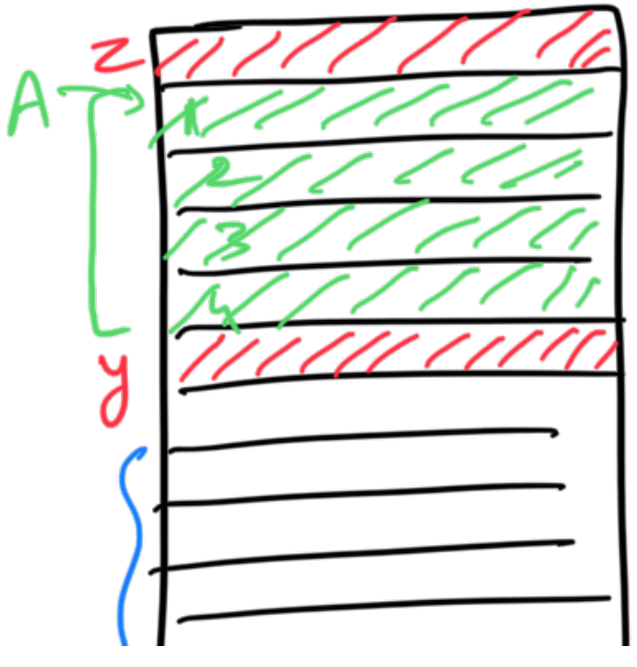
a[i] → i^{th} element in O(1)



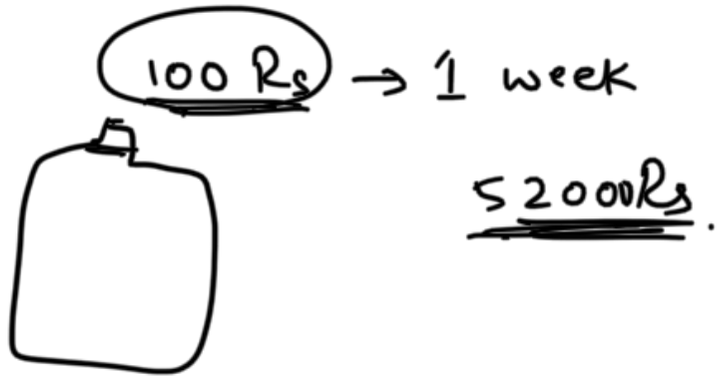
Arraylist / vector

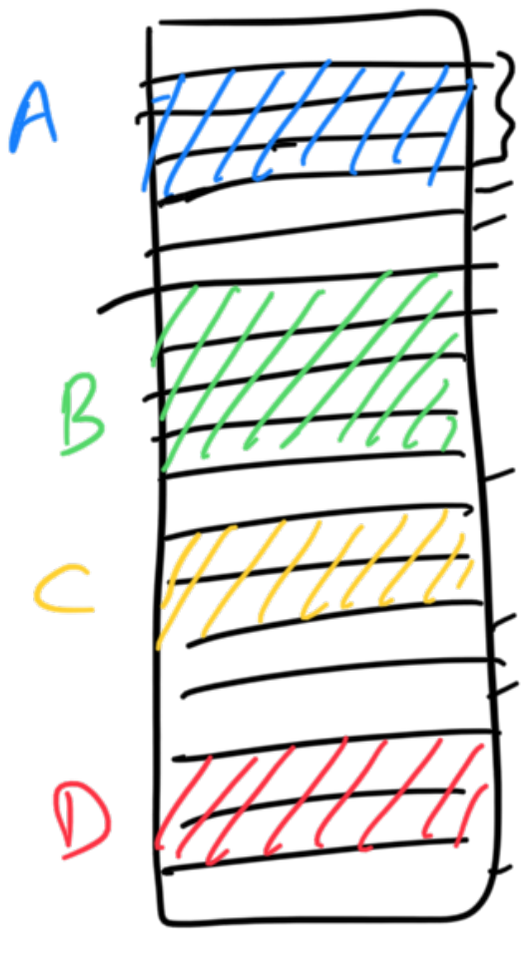
a[i] $\Longrightarrow$ O(1)





Insertion in dynamic array : O(1) [Amortized]



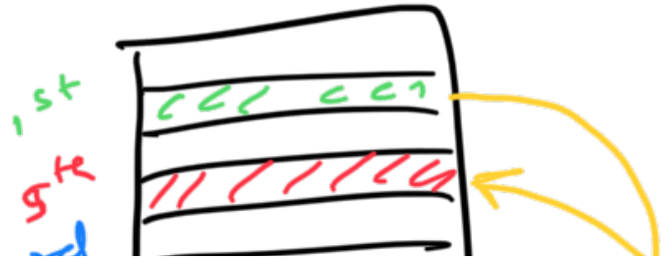


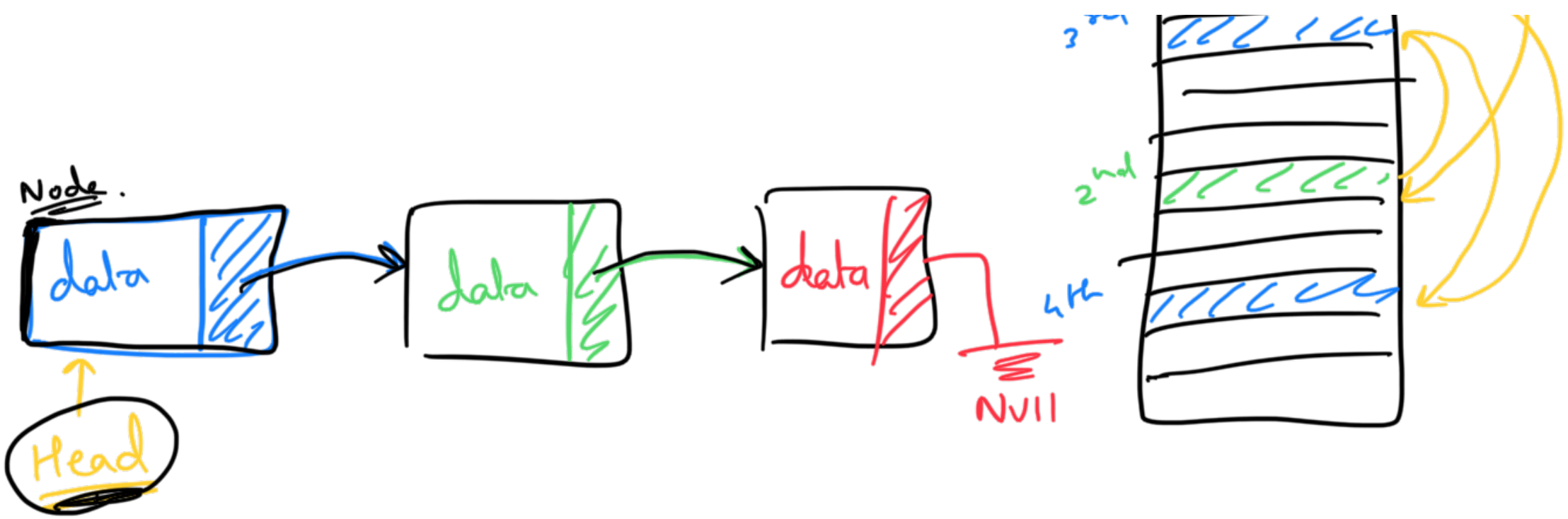
If we don't require O(1) Random access

A[3]

A.insert(10)

LinkedList



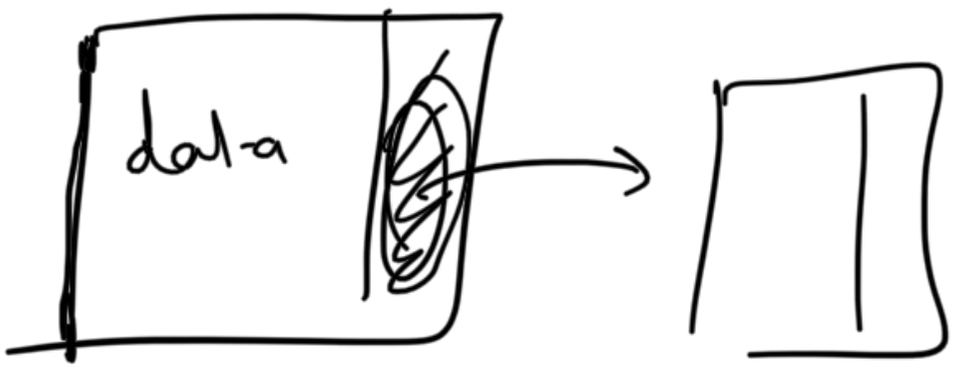


Class Node{

int data; (string, char, object)

Node next;

}



Constructor

Node x = new Node()

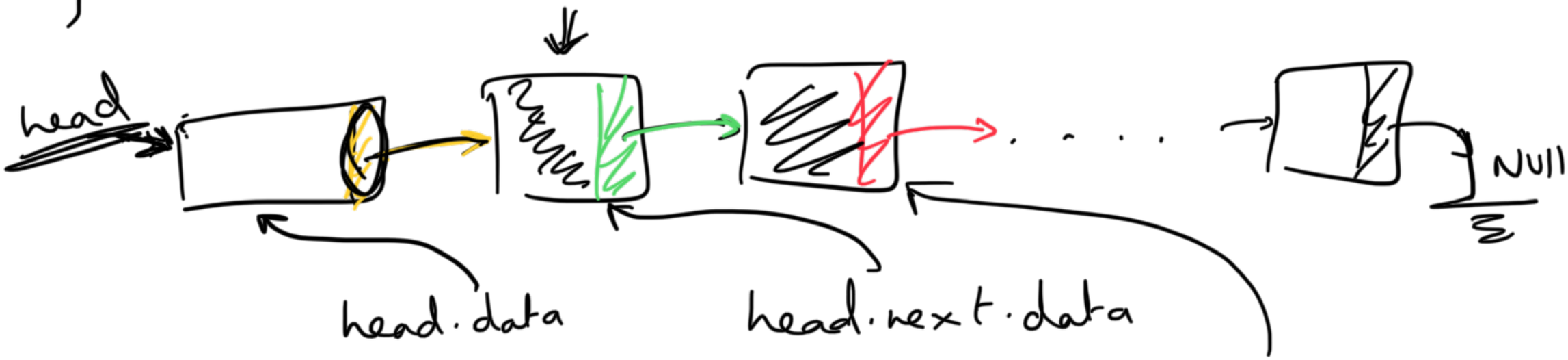
x.data = 0 / garbage

x.next = null / garbage

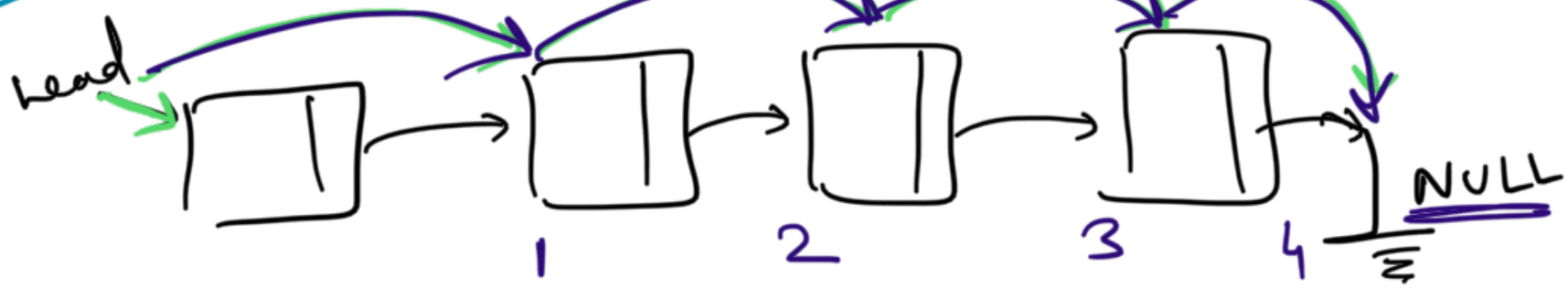
```
Class Node{
    int data;
    Node next;
    public Node(int a){
        this.data = a;
        this.next = null;
    }
}
```

Java

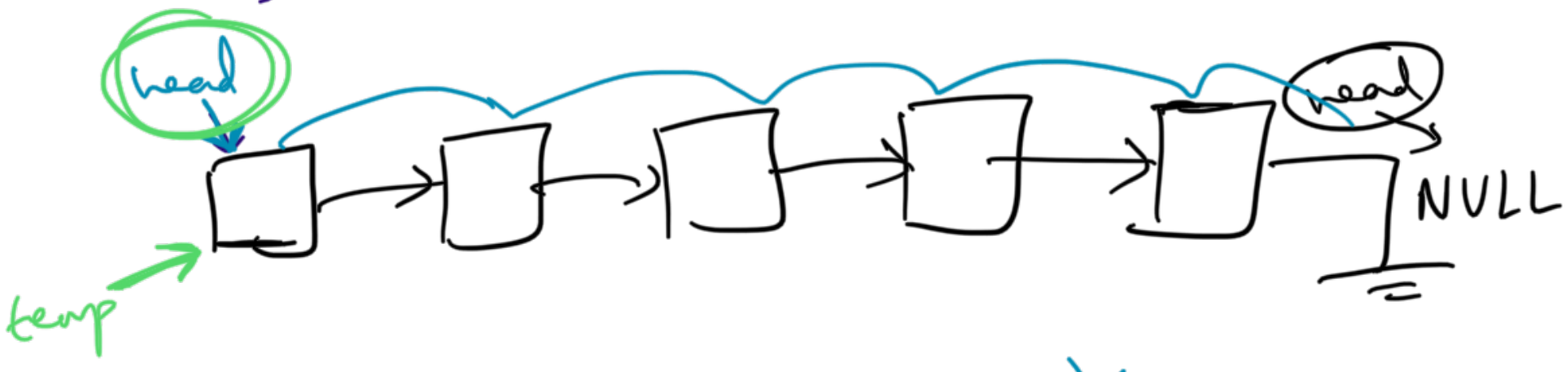
```
Node head = new Node(10)
head.data => 10;
head.next => null;
```



Q.1 Given a L.L, find its length.




```
int getLength (Node head){
    int len = 0
    While (head != null)
    {   len++;
        head = head.next
    }
    return len.
}
```




```
int getLength (Node head){
    int len = 0
```

(C++)

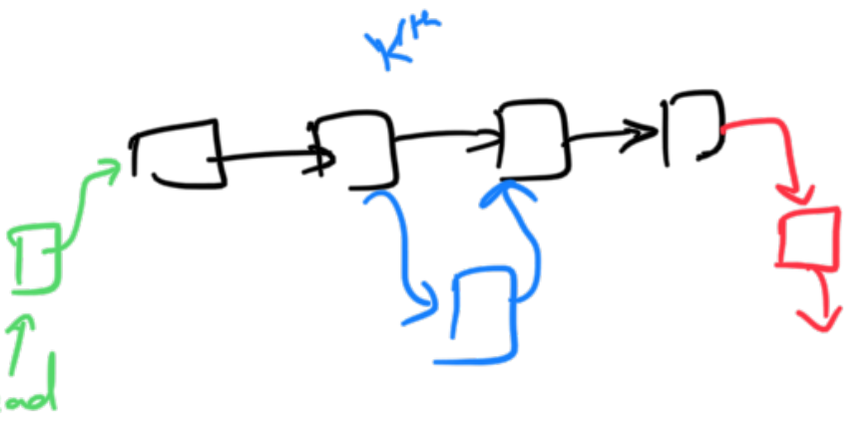
```
int getLength (Node* head)
    int len = 0
```

```
Node temp = head
while (temp != null)
{   len++;
    temp = temp.next
}
return len.
}
```

```
while (temp != NULL)
{   len++;
    temp = (*temp).next
}
return len;
}
```

Array< > x

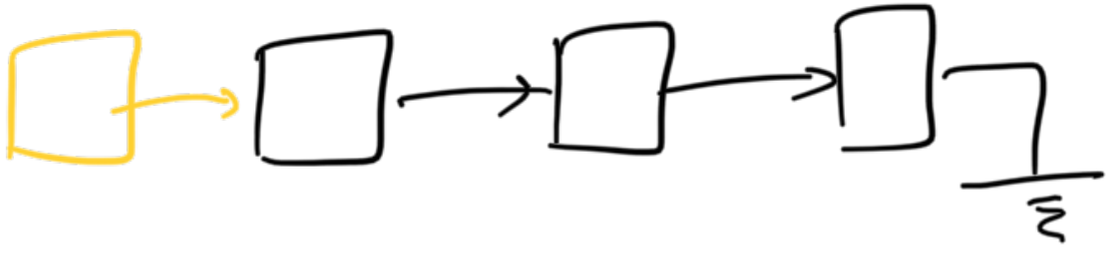
x.add(1)
x.add(2)



Insert-

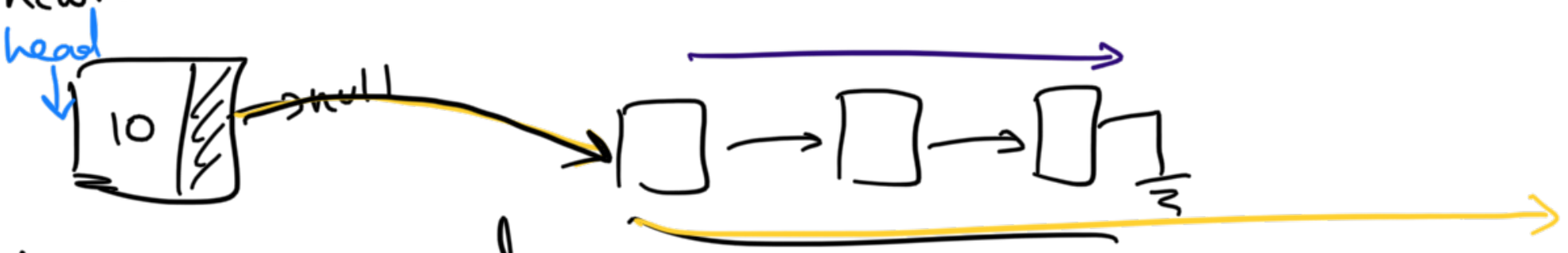
① At front

Given a LL,
Add data at front.



Add 10 :

Node newNode = new Node(10)



newNode.next = head

head = newNode;

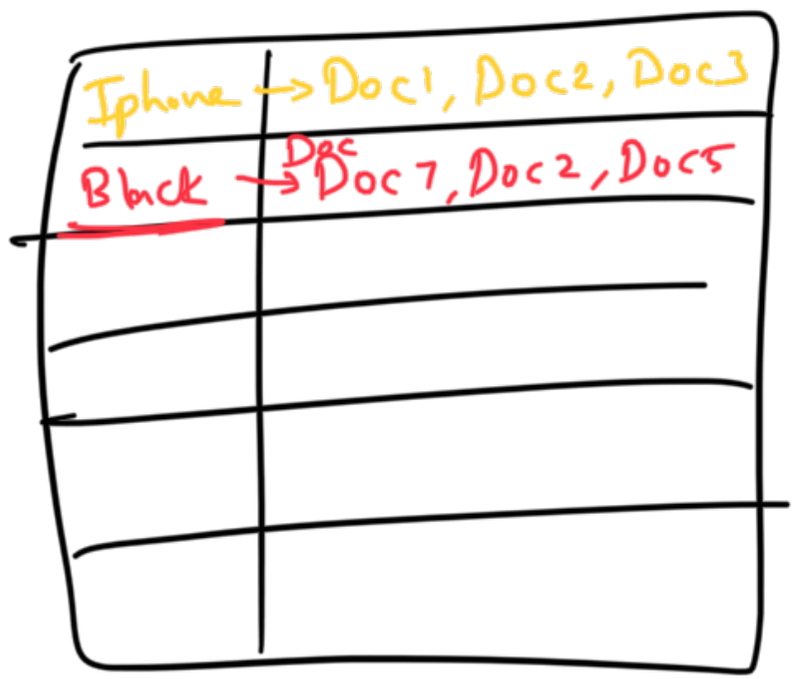
T.C : O(1)

[v.s. O(N) in arrays]

Elastic Search
Solar
Lucene

(Hashmap)

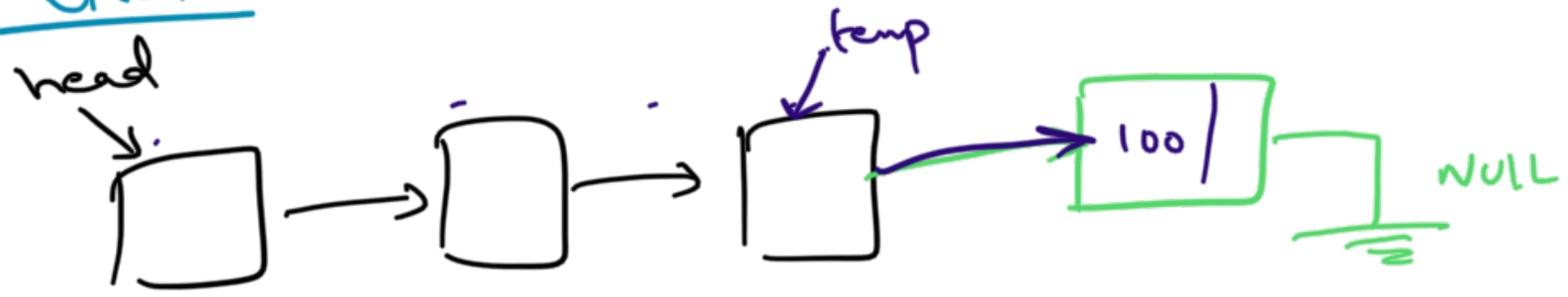
Key → Doc1, Doc7, Doc5



Inverted Index

Break ⇒ 10:14

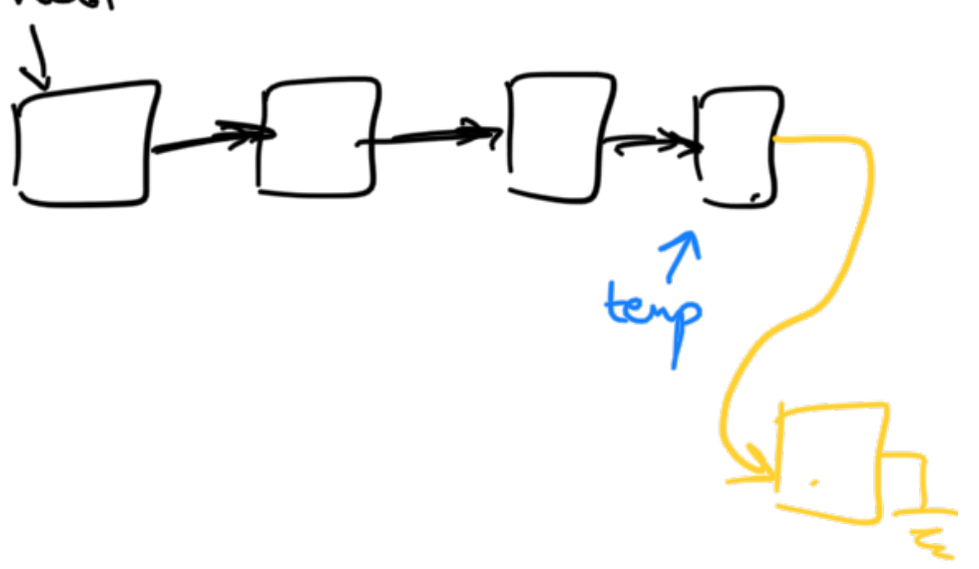
(2) Insert at end




```
Node newNode = new Node (100)
```

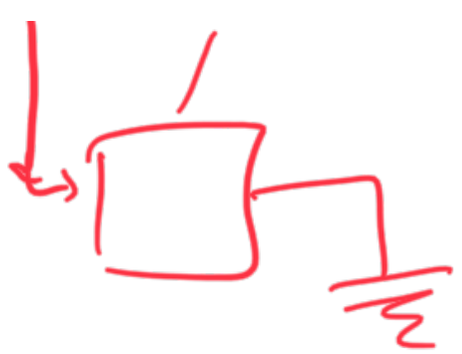
newNode.next = null

```
Node temp = head

While (temp.next != null)
{
    temp = temp.next;
}

temp.next = newNode;
```



T.C ⇒ O(N)

[V.S. O(N) in array]



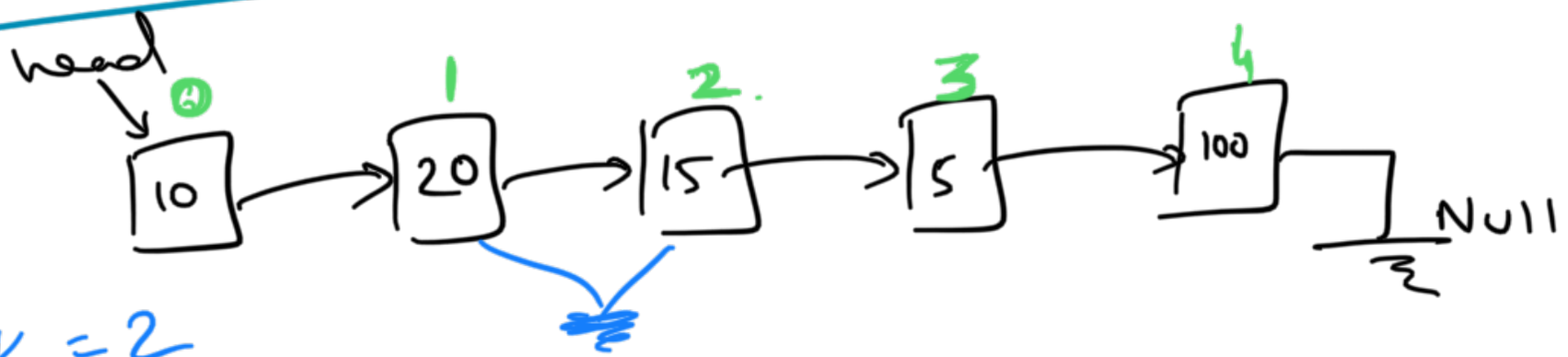
tail.next = newNode

tail = newNode

T.C ⇒ O(1)

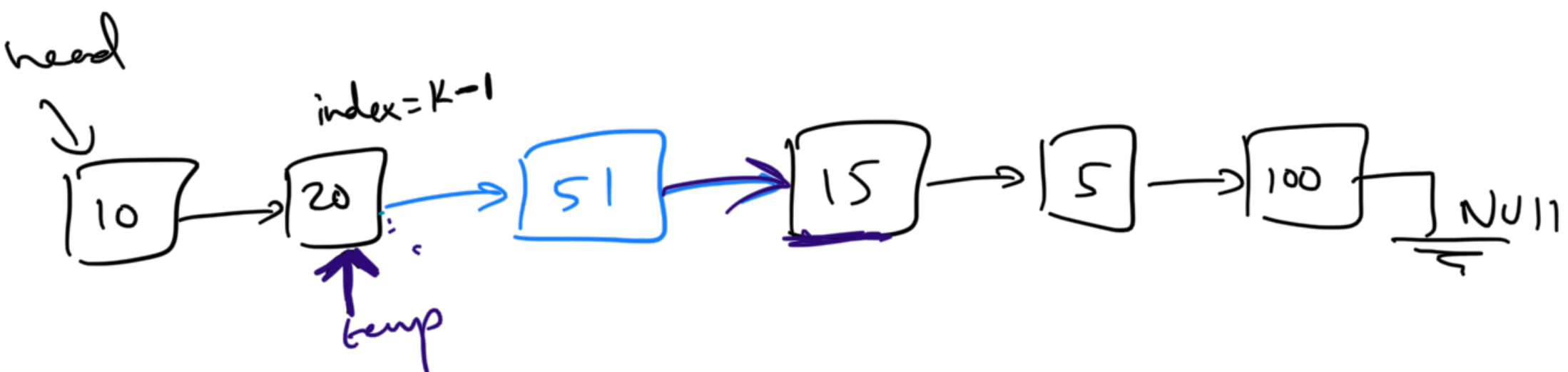
[V.S. O(N) in arr]

(3) Insert at K^{th} position



K = 2

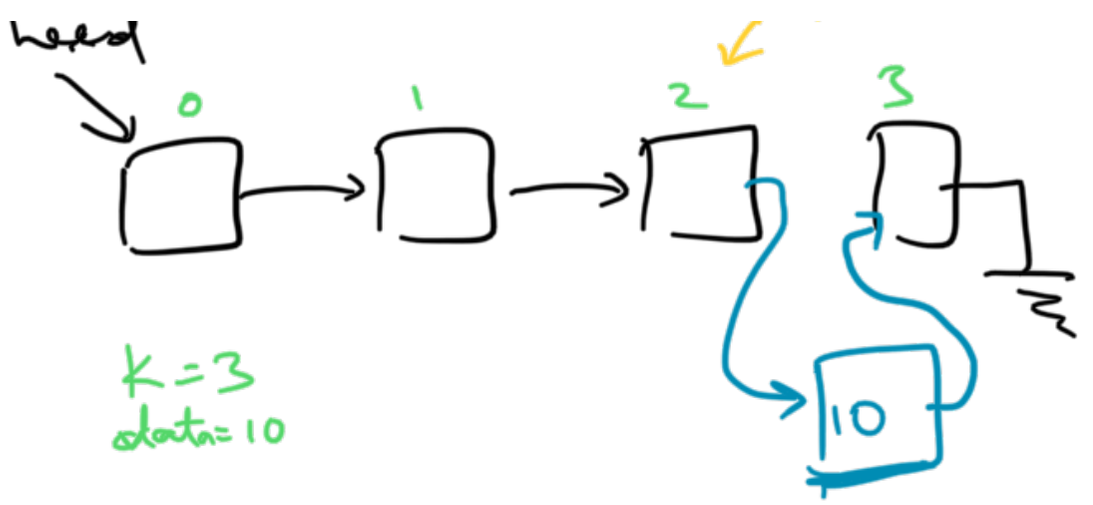
val = 51



c = 0

temp

```
temp = head
while (c <= k-1)
{
    temp = temp.next
    c++;
}
newNode = new Node (val)
newNode.next = temp.next;
temp.next = newNode
```



T.C ⇒ O(N)

[v.s. O(N) in array]

k=0
k=1
k>N

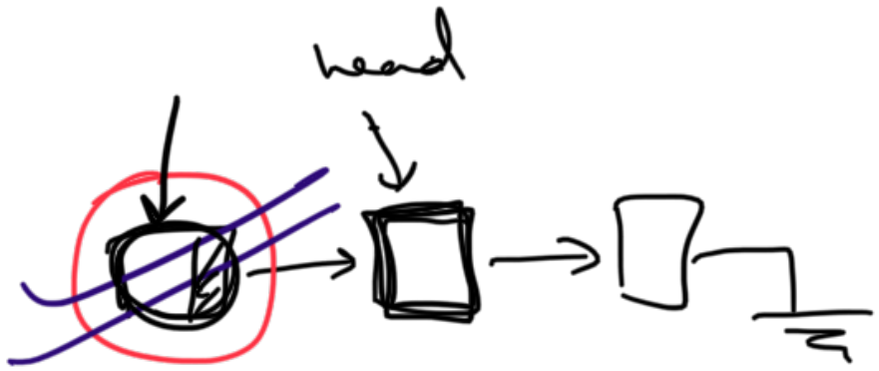
↳ write the code on paper

↳ Dry run

↳ Edge Case

- Null LL.
- size 1
- size 2/3/
- Problem specific

Deletion



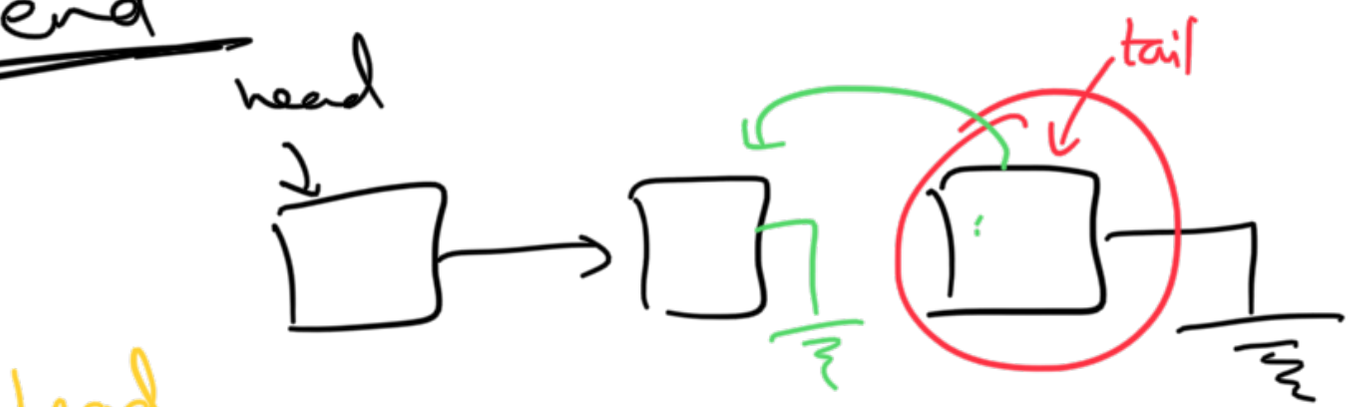
① From front

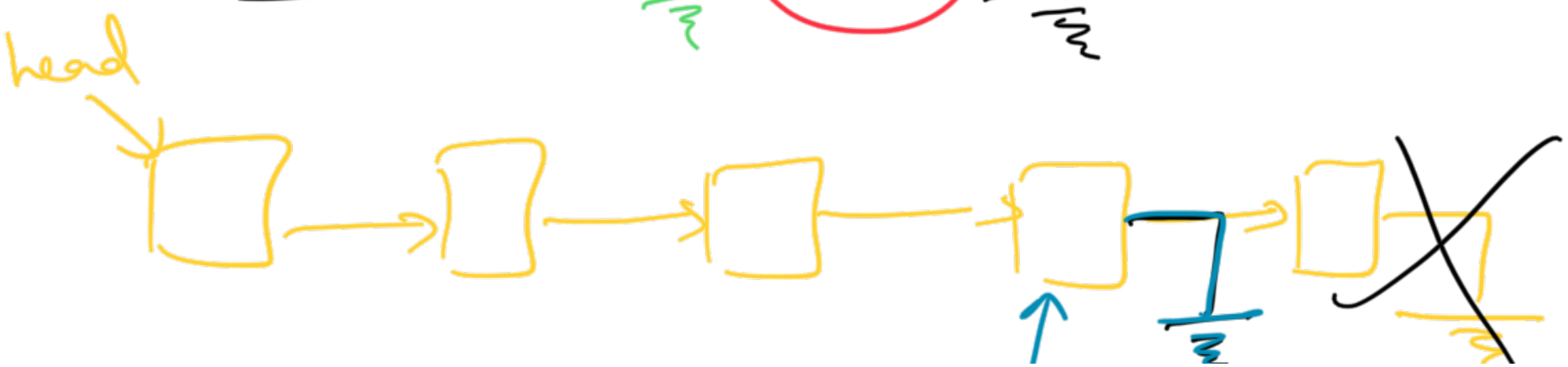
```
head = head.next
```

T.C ⇒ O(1)

[V.S. O(N) in array]

② From end





```
temp = head
While (temp.next.next != null)
{
    temp = temp.next
}
temp.next = null
tail = temp.
```

edge cases.

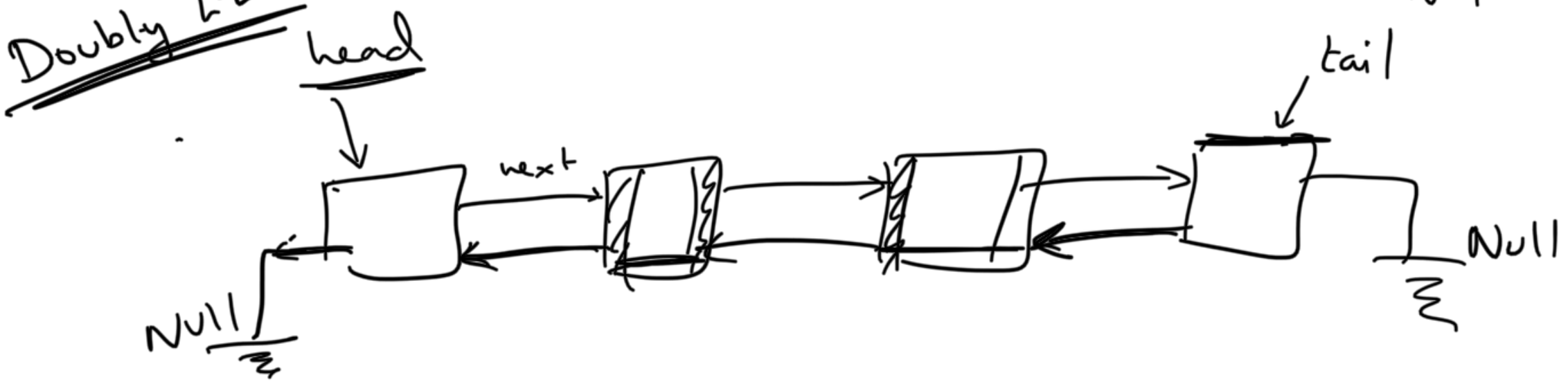
T.C ⇒ O(N)

[V.S. can be O(1) in array]

size ⇒ N

size ⇒ N-1

Doubly L.L.



O(N)

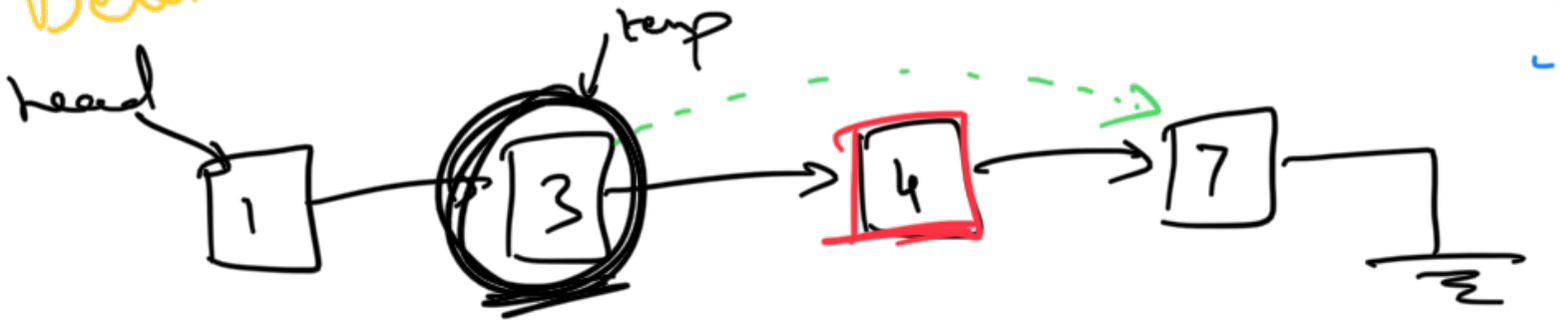
N int

N left

N steps

Google Q. Given a node of a L.L.
Delete this node from the L.L.

(There are no duplicate
This node will not
be the last node)




```
While (temp.next.data != node.data)
{   temp = temp.next
}
```

⇒ O(N).

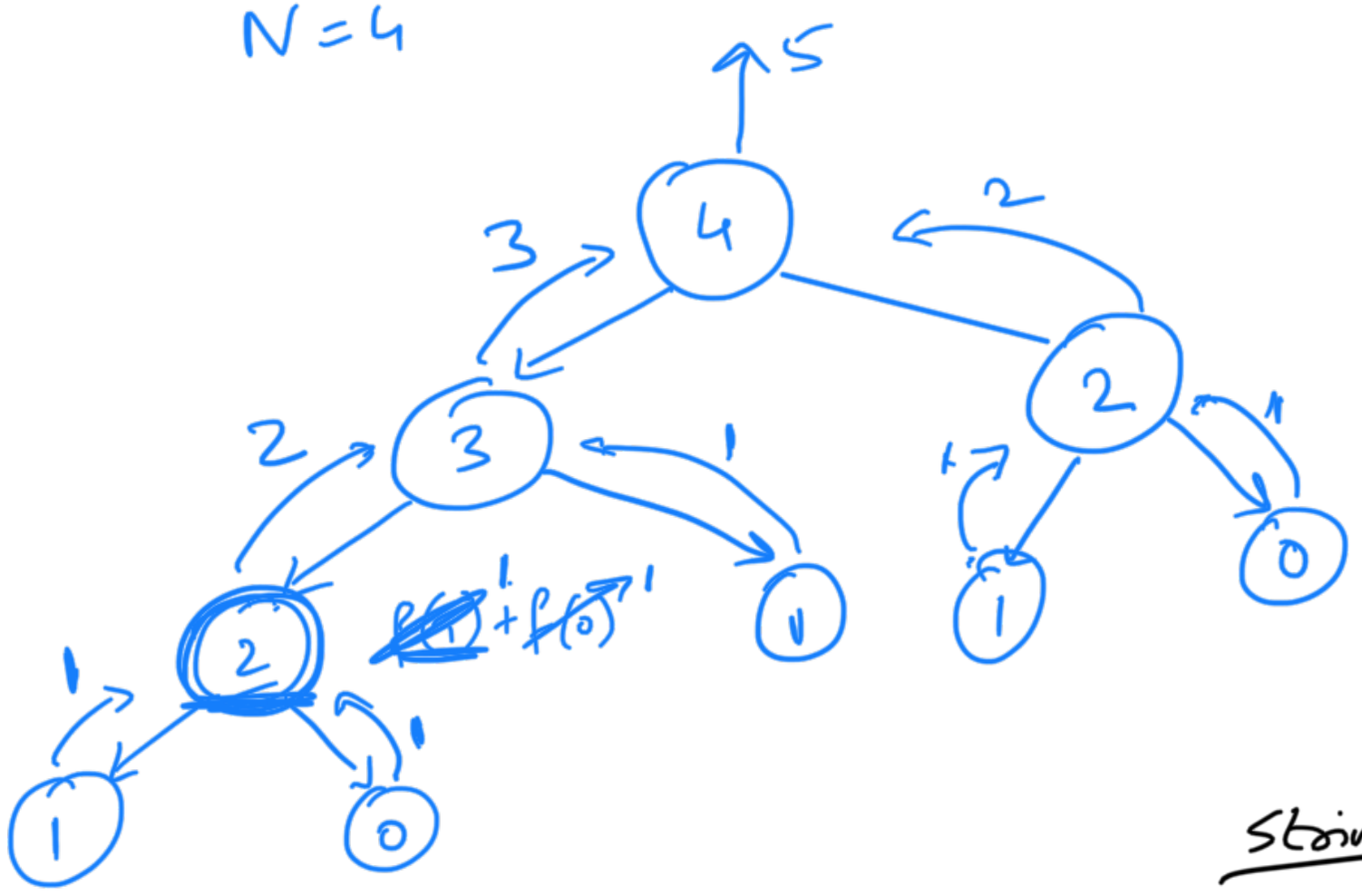
```
head
```

O(1)

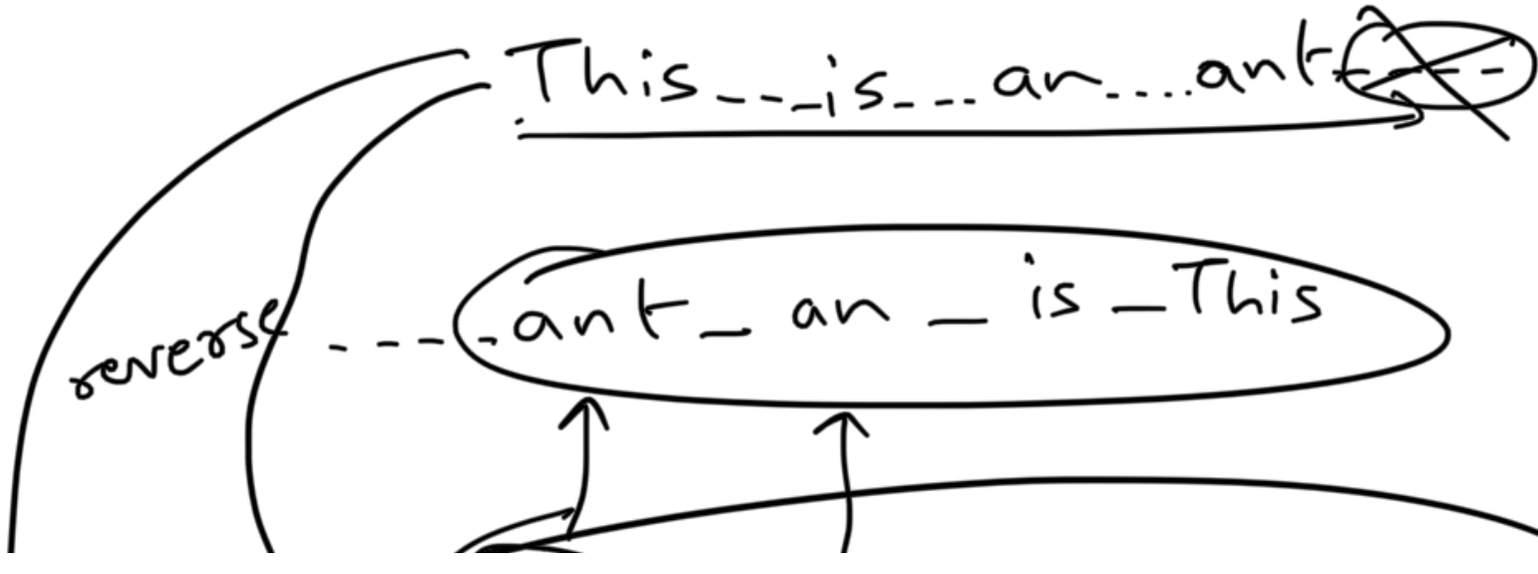
Doubts

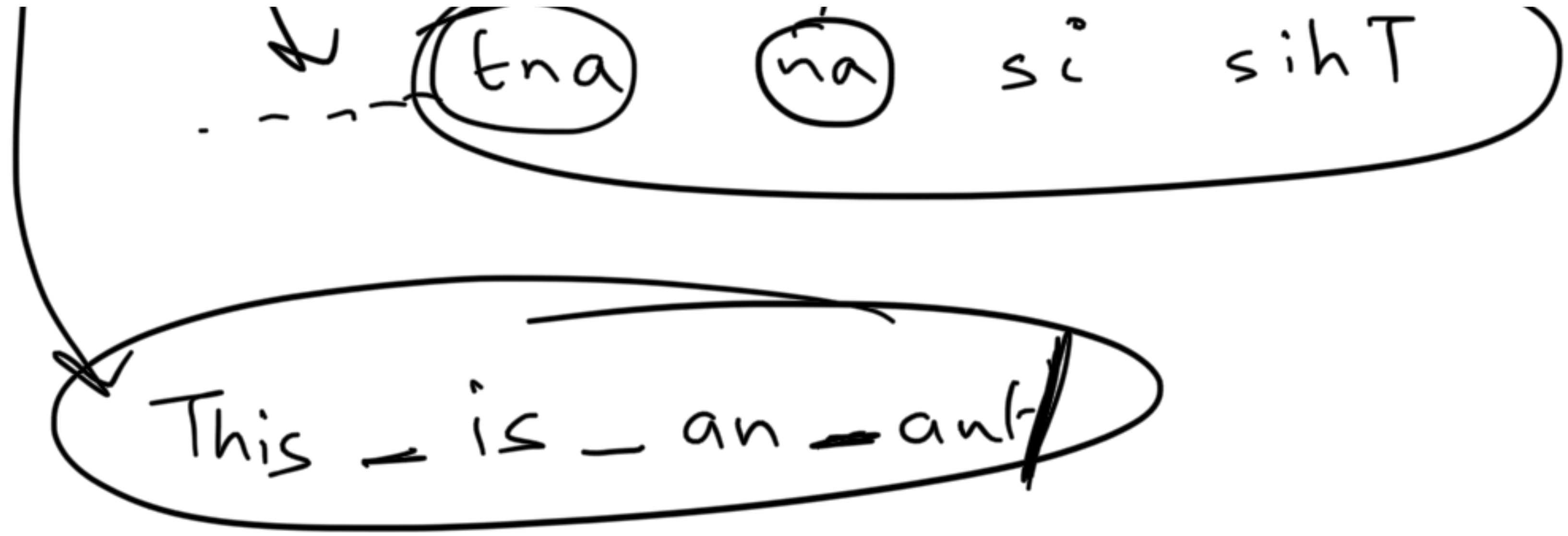
N = 4

f(0) = 1
f(1) = 1



Strings




tna
na
si
sihT
This _ is _ an _ ant