Linked List

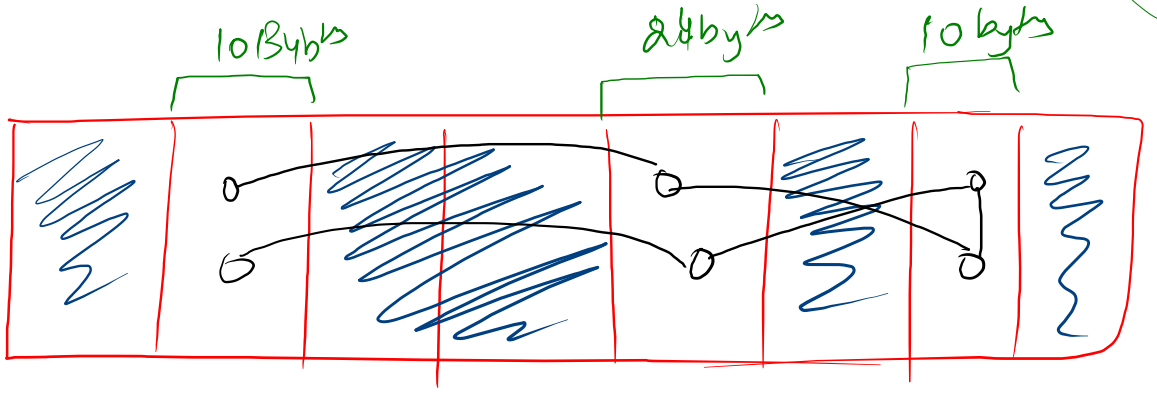
Level-1

Head ⇒ 1st ele

Tail ⇒ last ele

Size ⇒ no. of elements in a list

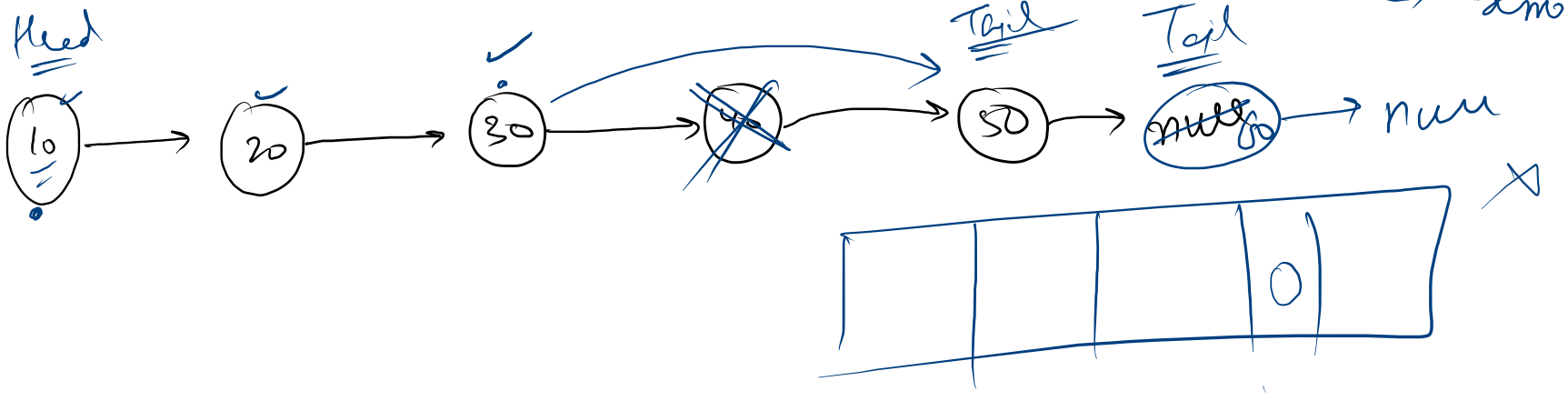
Array → Contiguous mem



→ (direct Access) ✗

→ (size → prequisite) ✗

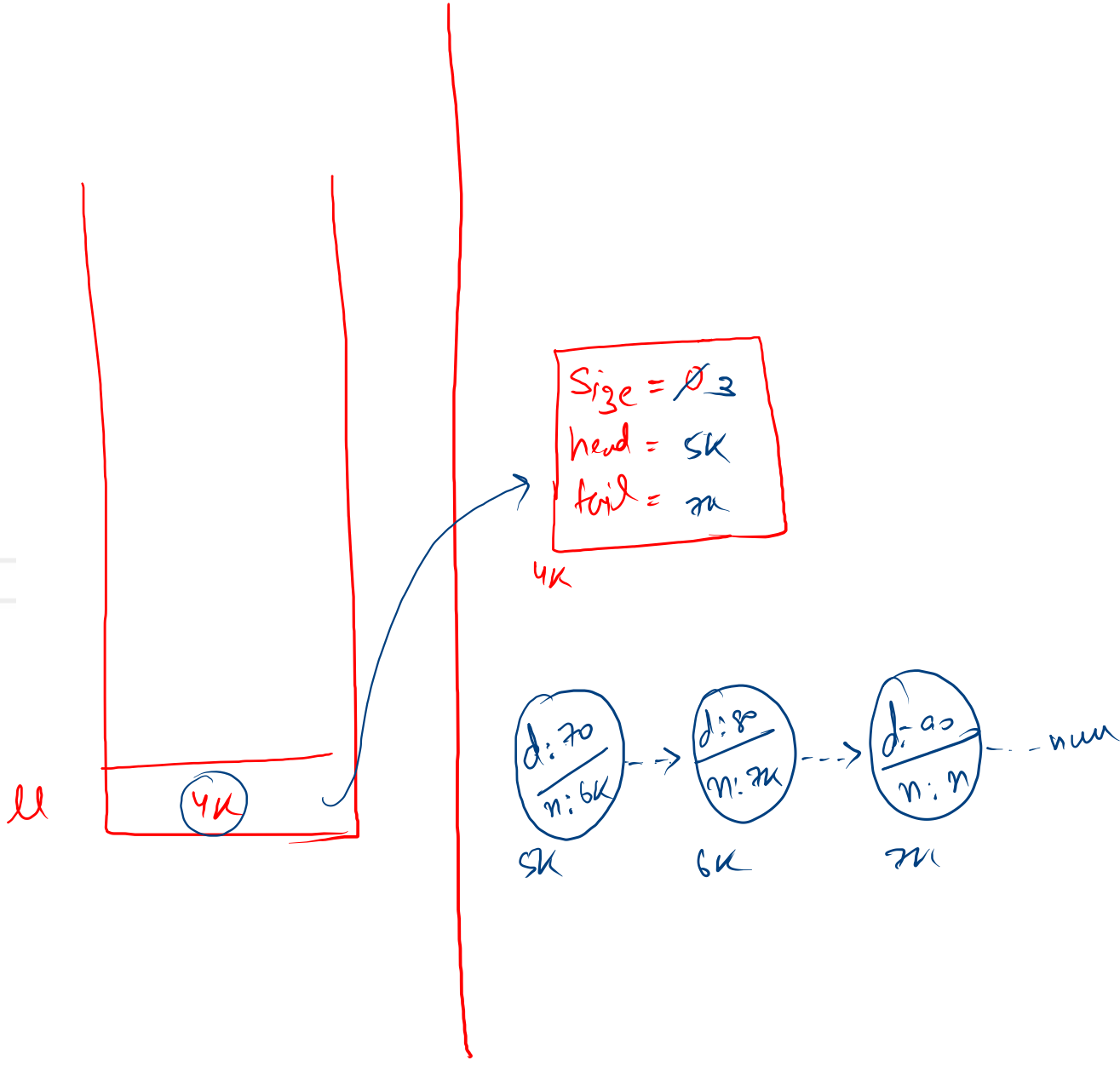
→ remove / delete operation ✓



Class Node

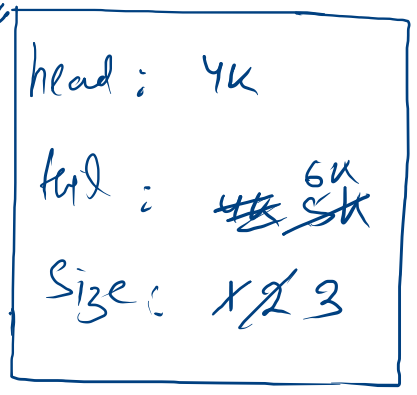
- data (int)
- next (Node)

```java
public static class Node{
    int data;
    Node node;
}

public static class LinkedList{
    int size;
    Node head,tail;
}

Run | Debug
public static void main(String args[]){
    LinkedList ll = new LinkedList();
}
```



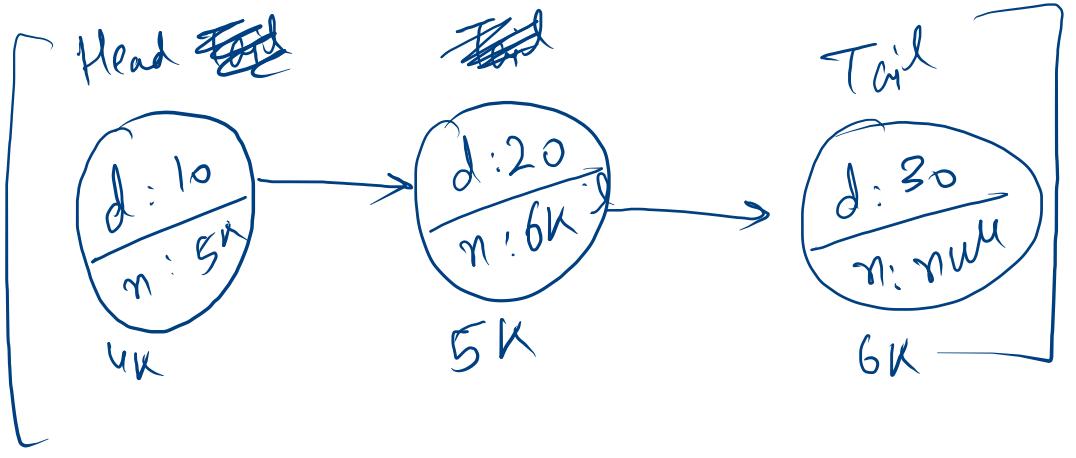
① Create a new Node

② assign val



- ll.addLast(10)
- ll.addLast(20)
- ll.addLast(30)

```
public static class LinkedList{
    int size;
    Node head,tail;

    void addLast(int val) {
        Node node = new Node();
        node.data = val;

        if(this.size == 0){
            this.head = this.tail = node;
        }else{
            this.tail.next = node;
            this.tail = node;
        }

        this.size++;
    }
}
```

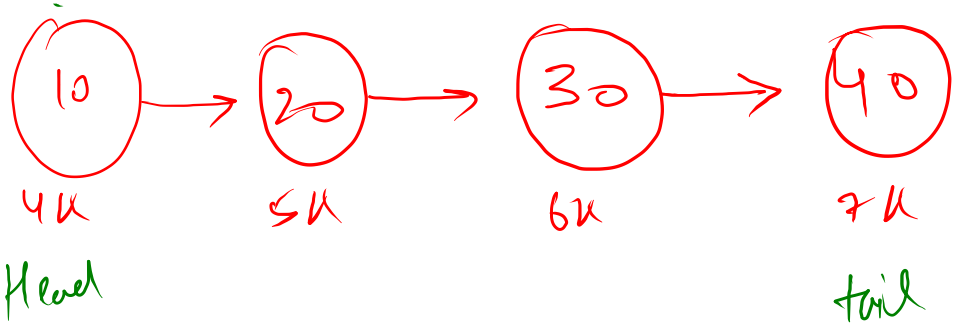


Head = 4K

Tail = 7K

Size = 4

Display → 10 20 30 40



tmp

```
public void removeFirst(){
  Node nbr = this.head.next;
  this.head.next = null;
  this.head = nbr;
  this.size--;
}
```

Head = ~~8K~~ null

Tail = 8K

Size = ~~1~~ 0

removeFirst();

removeFirst();

=

=

=

head = 4K

tail = 9K

size = 6

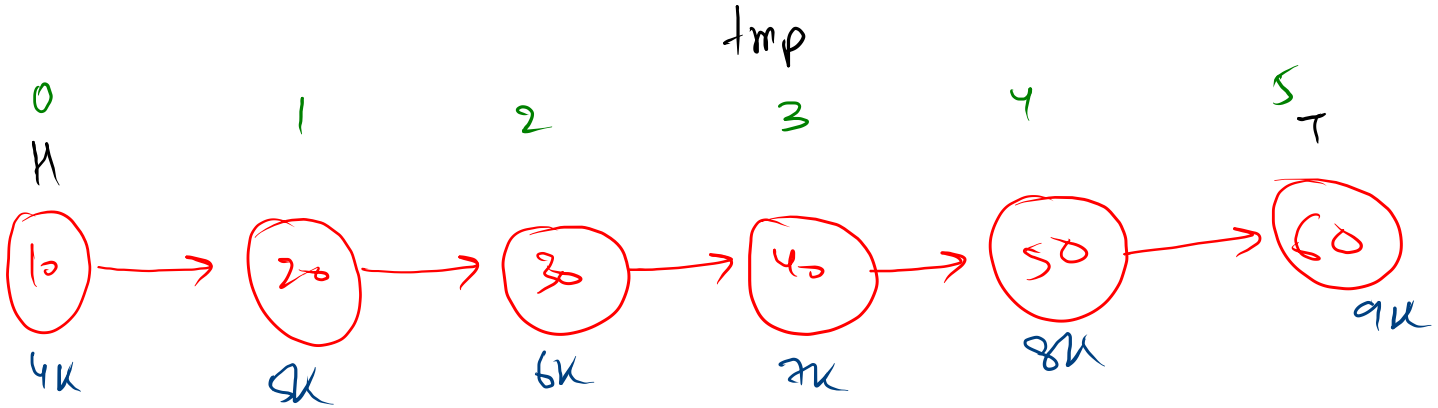
getFirst()

↳ head.data

getLast()

↳ tail.data

getAt(3)



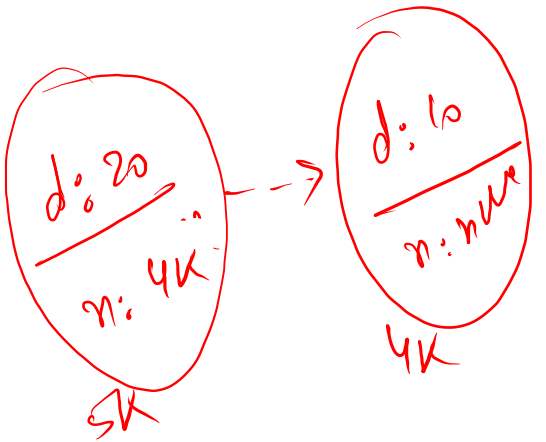
```java
public void addFirst(int val) {
    Node node = new Node();
    node.data = val;

    if(this.size == 0){
        this.head = this.tail = node;
    }else{
        node.next = this.head;
        this.head = node;
    }
    this.size++;
}
```

head = 4K ~~5K~~
tail = 4K
size = ~~1~~ 2

✓ addFirst (10)

✓ addFirst (20)

addFirst (30)

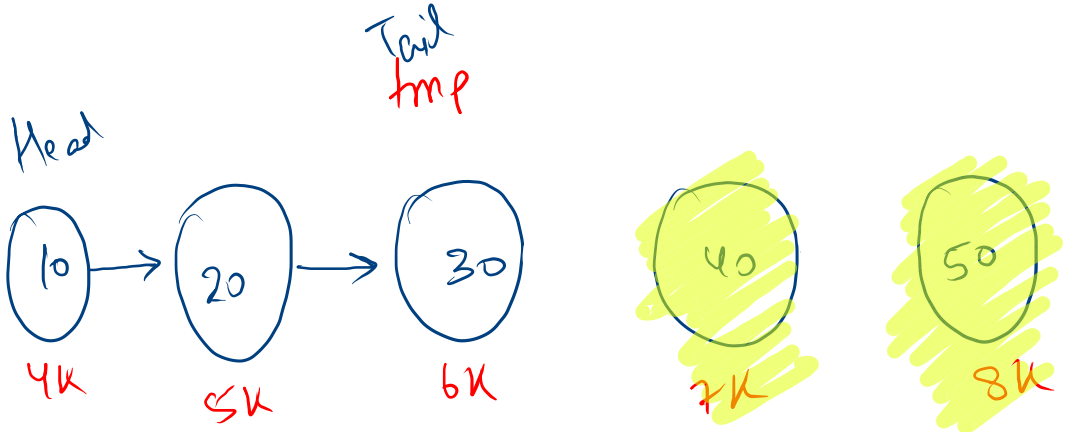


Head = 4K

Tail = ~~8K~~ ~~7K~~ 6K

Size = ~~5~~ X 3

removeLast()




```java
public void removeLast(){
    if(this.size == 0){
        System.out.println("List is empty");
        return;
    }

    if(this.size == 1){
        this.head = this.tail = null;
    }else{
        Node tmp = head;
        while(tmp.next != tail){
            tmp = tmp.next;
        }

        tmp.next = null;
        tail = tmp;
    }
    this.size--;
}
```

addAt(idx, val)

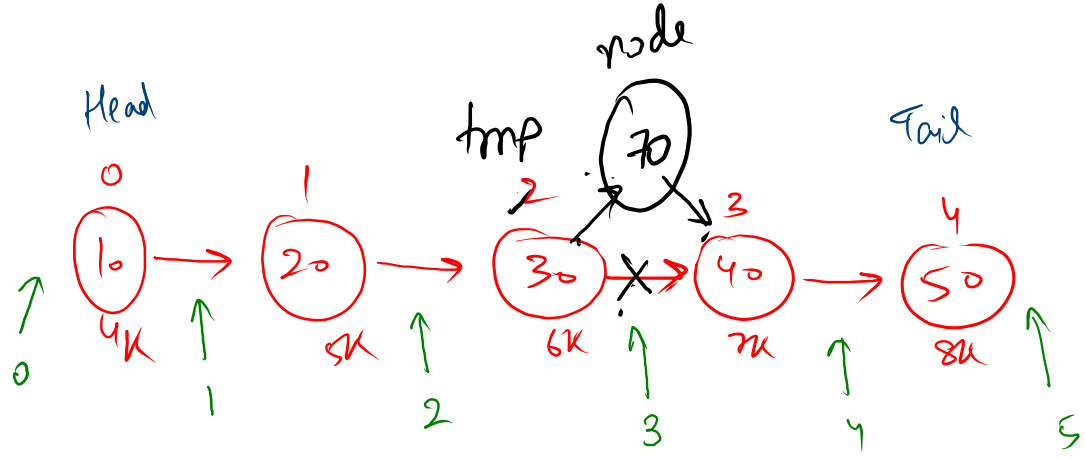
↳ [0, size]

head = 4K

tail = 8K

size = ~~5~~ 6

add(3, 70)



idx = ~~3~~ 2 1

```
Node node = new Node();
node.data = val;

Node tmp = head;

while(idx > 1){
    tmp = tmp.next;
    idx--;
}

node.next = tmp.next;
tmp.next = node;

this.size++;
```

```
Node tmp = head;

while(idx > 1){
    tmp = tmp.next;
    idx--;
}

Node nbr = tmp.next;
tmp.next = nbr.next;
nbr.next = null;

this.size--;
```

0 - 5

(idx < 0 || idx >= size)

↳ Invalid arguments

head = 4K

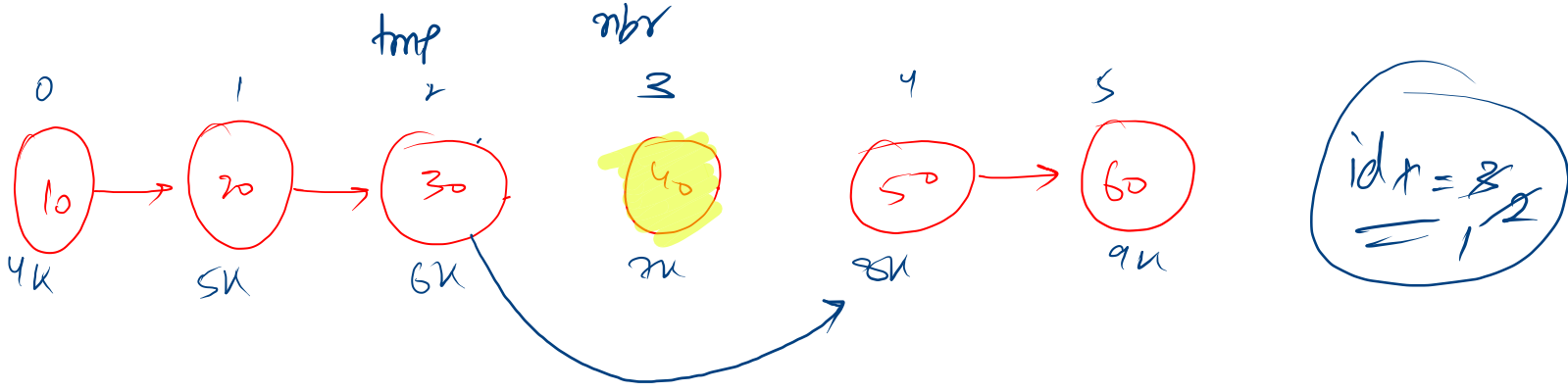
tail = 9K

Size = ~~6~~ 5

removeAt(0) → removeFirst

removeAt(5) → removeLast

removeAt(3) →



LinkedList
return

data - (Iterative)

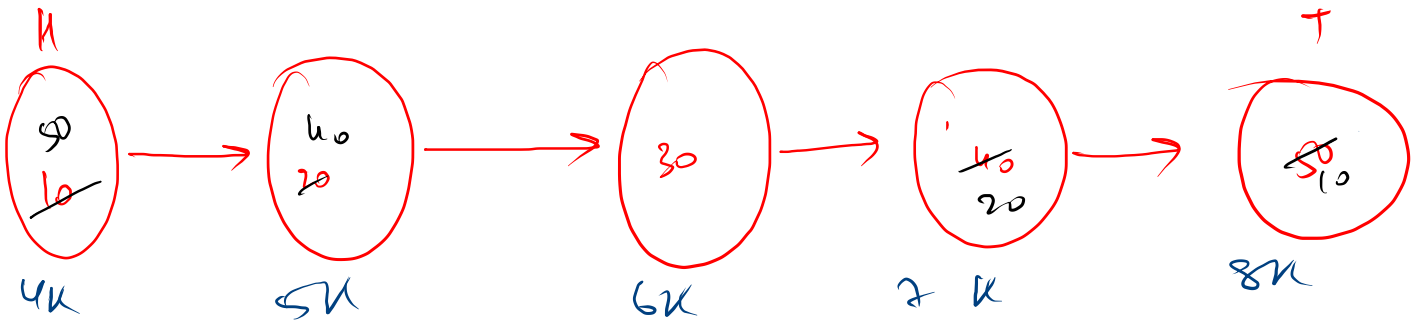
head = 4K
tail = 8K
Size = 5

lp rp

0 1 2 3

```
public Node getNodeAt(int idx){ }
public void reverseDI() {
    int lp = 0 , rp = this.size-1;

    while(lp < rp){
        Node left = getNodeAt(lp);
        Node right = getNodeAt(rp);

        int temp = left.data;
        left.data = right.data;
        right.data = temp;

        lp++;
        rp--;
    }
}
```

1 + 2 + 3 + 4 + 5

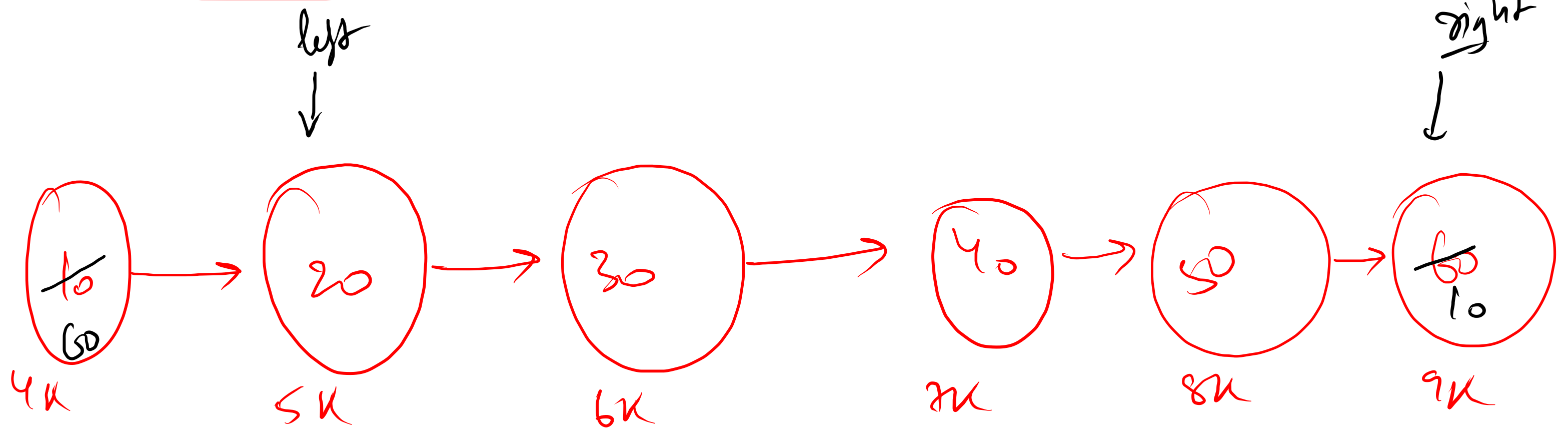


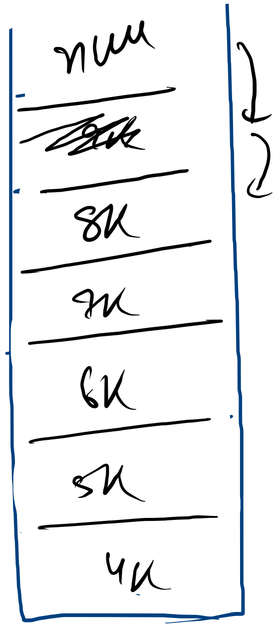
T.C. -> 1+2+3+4+....+n

T.C. -> n(n+1)/2

T.C. -> O(n²)

head = 4K
tail = 9K
Size = 6





1. Swap
2. left →
3. right

$T(n) = T(n-1) + 1$ → $O(n)$

$2n \rightarrow O(n)$

```
static Node leftRev;
public void reverseDR() {
  leftRev = head;
  reverseDRHelper(head,0);
}

public void reverseDRHelper(Node node,int idx){
    if(node == null){
        return;
    }
    reverseDRHelper(node.next,idx+1);

    if(idx >= this.size/2){
        Node right = node;

        int temp = right.data;
        right.data = leftRev.data;
        leftRev.data = temp;

        leftRev = leftRev.next;
    }
}
```

O(1)

O(1)

DRH

DRH()

DRH()

DRH()

DRH()

DRH

revDR()

null, 5

8K, 4

7K, 3

6K, 2

5K, 1

4K, 0

3K

Y → 2

n → 2n

LL

leftRev = 4K → 5K → 8K → 7K

head = 4K
tail = 8K
size = 5

3K