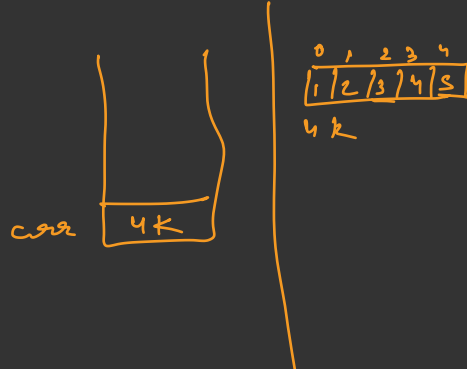


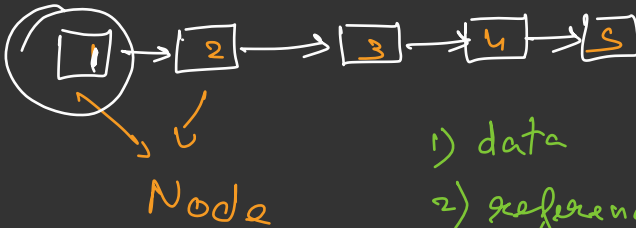
Linked List

int arr = new int[5]

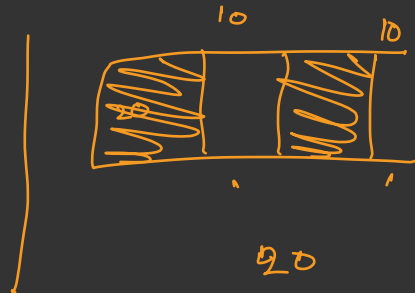
20 bytes
↗



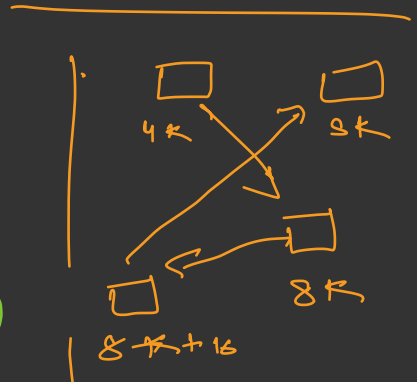
LL was introduced



- 1) data
- 2) reference to next



Node



LL

Node head;

Node tail;

int size

display

add

remove

→ addFirst (int val)

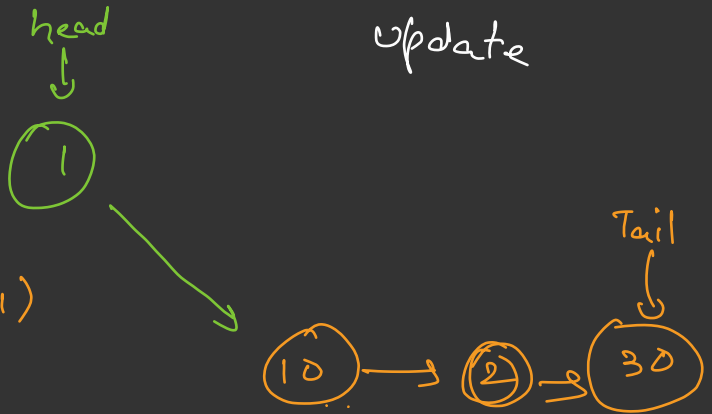
→ addLast (int val)

→ add (int val, int idx)

get

update

addFirst (1)



① Create new node with 1 as data;

② point the next to head

③ Point head to new node

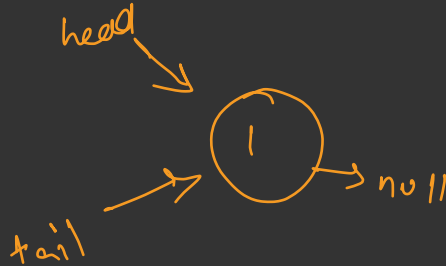
④ Increase size

if (size == 0)

Node n = new Node(1, null);

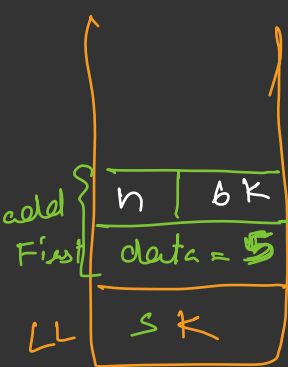
head = n;

tail = n

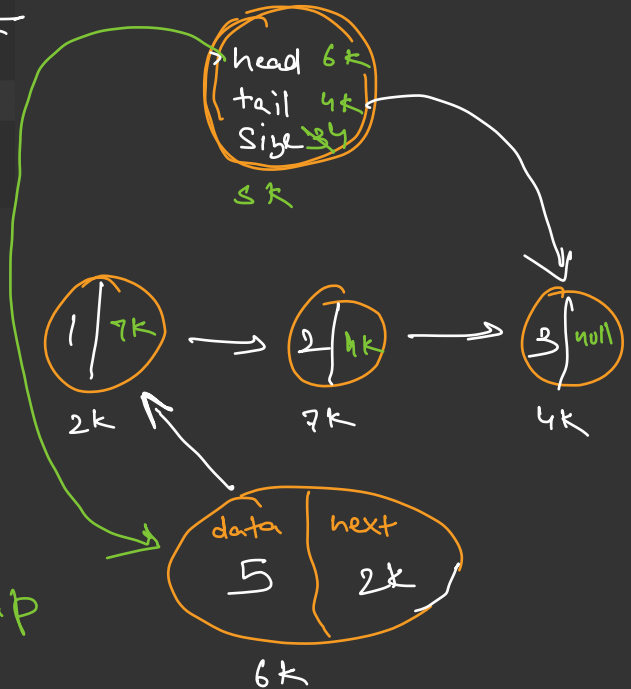


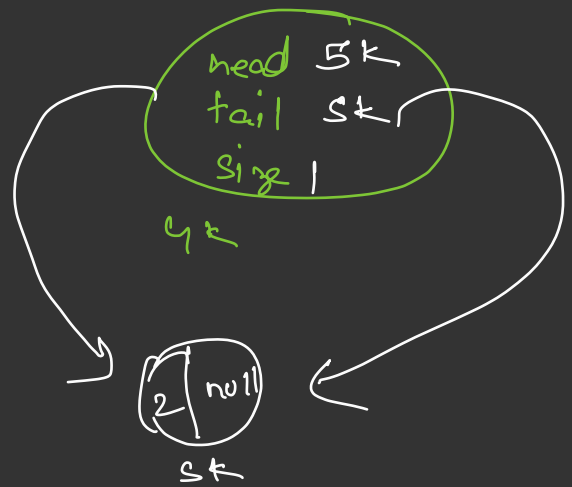
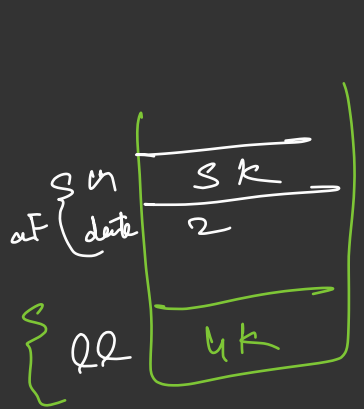
LL
size = 0
head = null
tail = null

```
Node n = new Node(); ←  
n.data = data; ←  
n.next = head; ←  
head = n; ←  
this.size++; ←
```

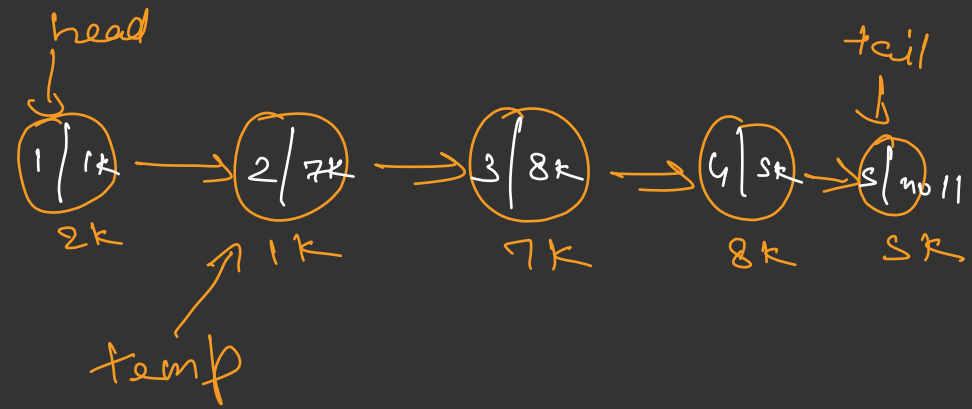


heap





ll = new L2()
addFirst(2)



temp = temp.next

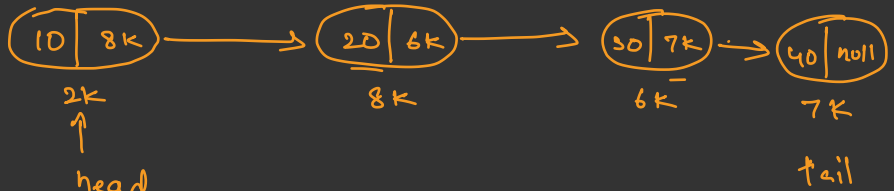


display
temp.data

```

public void display() {
    Node temp = head;
    while(temp != tail) {
        System.out.print(temp.data + " -> ");
        temp = temp.next;
    }
    System.out.println("END");
}

```



10, 20

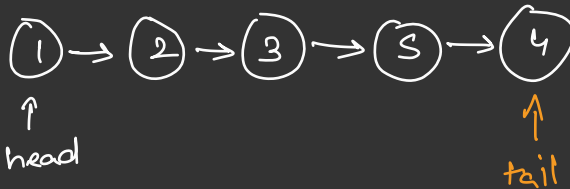
head = ~~2K~~ ~~8K~~ 6K

Node a = head
Node b = head

a = 2K

b = 2K

add Last (int val)



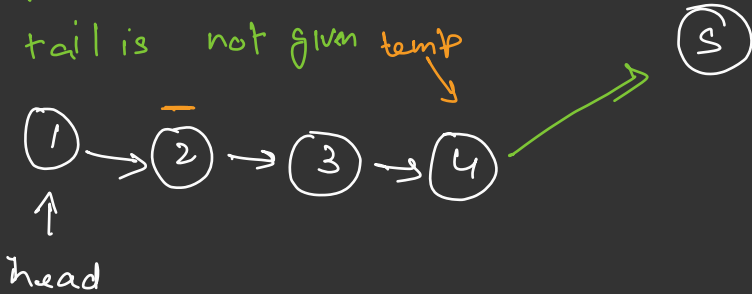
① create a node

② point tail.next to new node

(3) Point tail to new node

(4) Increase size:

Note Special Case
tail is not given temp



```
Node temp = head;
while (temp.next != null) {
    temp = temp.next;
}
Node n = new Node(data, null);
temp.next = n;
return n
```

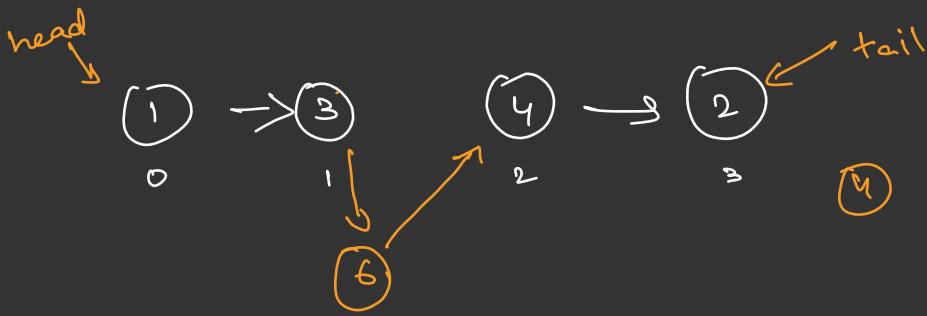
add(6, 2)

add (int data, int idx)



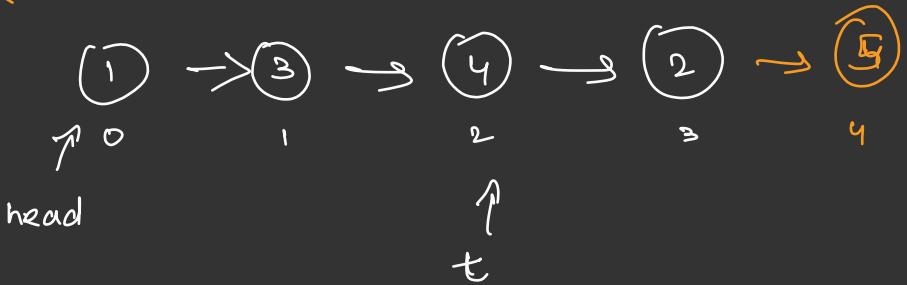
Output





- ① Traverse LL till idx - 1;
- ② Create new node and point t.next
- ③ point t to new node
- ④ Increase size

idx = 3. add(6, 3)



$i = 0 \neq 2$

while ($i < \text{idx} - 1$) {
 $t = t.\text{next};$
 $i++;$
}

get(int idx)

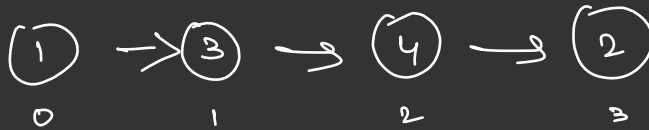
1) Traverse till idx

2) return node.data

get Node

1) return node.

Remove First



↑
head

↑
tail

if (size == 1)



size = 0

head = null

tail = null

