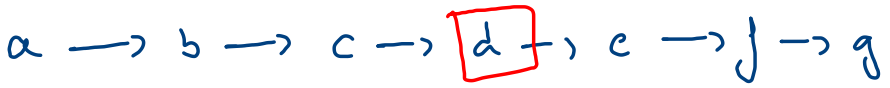
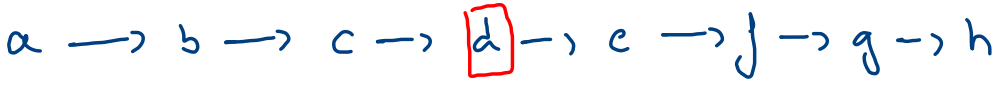
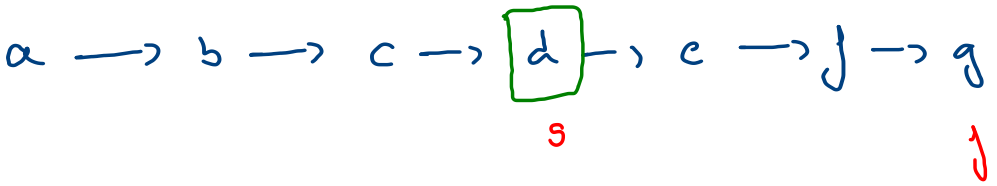
Mid Of Linked List





(i) size is not allowed to use.



$s = \frac{d}{t}$

$T_s = T_f$

$\frac{d_s}{s_s} = \frac{d_f}{s_f}$,

slow → x

fast → 2x

$d_s = s_s * \frac{d_f}{s_f}$

$= x * \frac{len}{2x} = len/2$

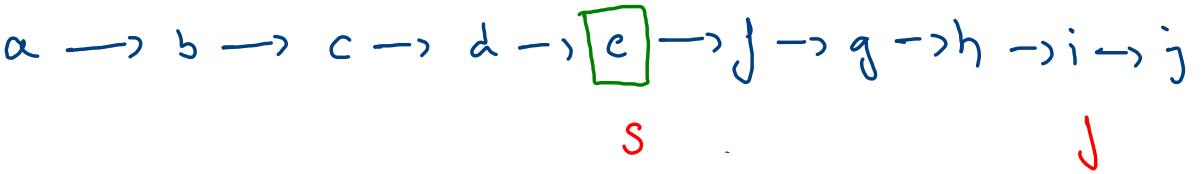
Stop : fast.next == null

```java
public int mid(){
    Node slow = head;
    Node fast = head;

    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    return slow.data;
}
```

T

F

even length



```java
public int mid(){
    Node slow = head;
    Node fast = head;

    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    return slow.data;
}
```

odd length

a → b → c → d → e → f → g → h → i → j → k

s

f

```java
public int mid(){
    Node slow = head;
    Node fast = head;

    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    return slow.data;
}
```

fast.next.next != null 이 fast.next != null

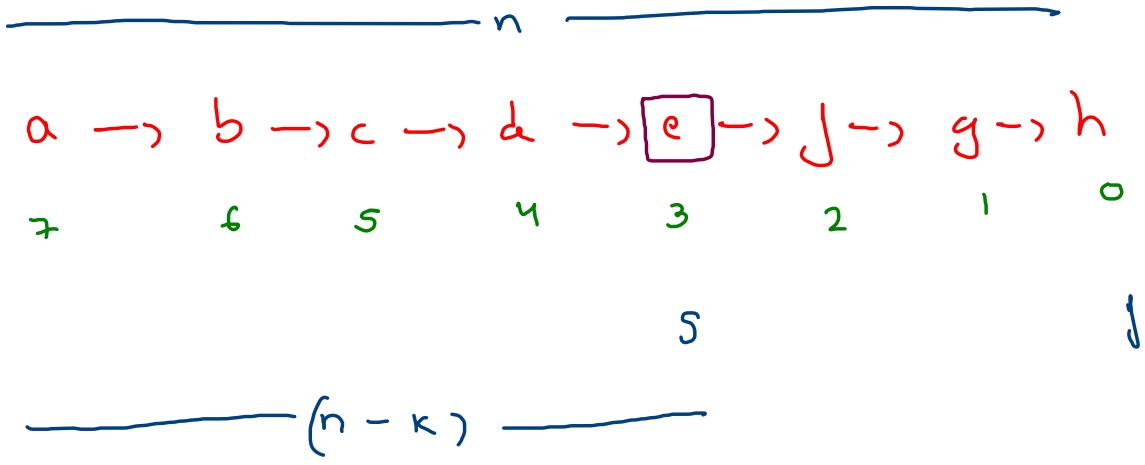
null

1 -> 2 -> 3 -> 4 -> 5

s f

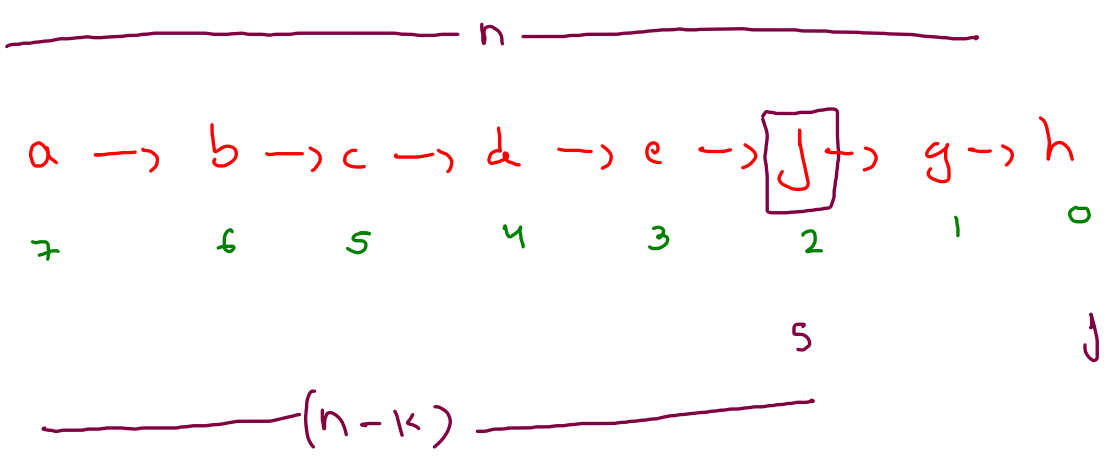
Kth Node From End Of Linked List

k = 3



(i) size not allowed.

$k = 2$

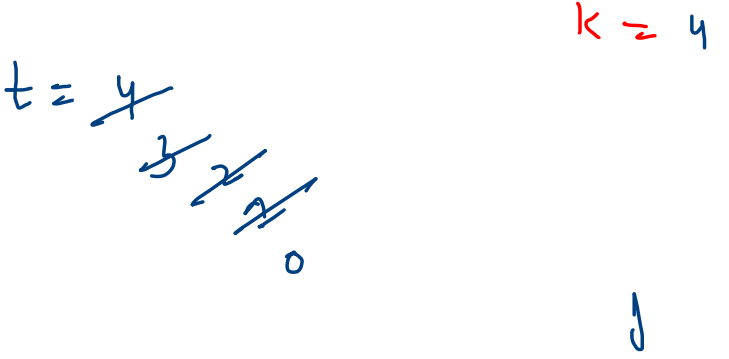


(i) maintain a gap(of k) blw slow and fast

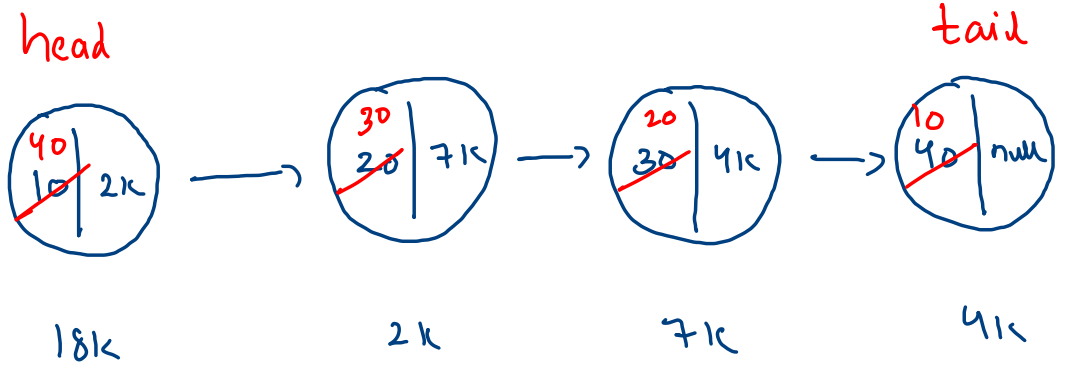
(ii) move slow and fast with same speed(1x)

(iii) when distance by fast is n (fast end of LL) distance by slow -> n-k

```java
public int kthFromLast(int k){
    Node slow = head;
    Node fast = head;

    //1. move fast by k steps
    int temp = k;
    while(fast != null && temp > 0) {
        fast = fast.next;
        temp--;
    }

    if(fast == null && temp > 0) {
        //k > size
        return -1;
    }

    //2. move slow and fast by same speed(1x)
    while(fast.next != null) {
        slow = slow.next;
        fast = fast.next;
    }

    return slow.data;
}
```



Reverse A Linked List (data Iterative)



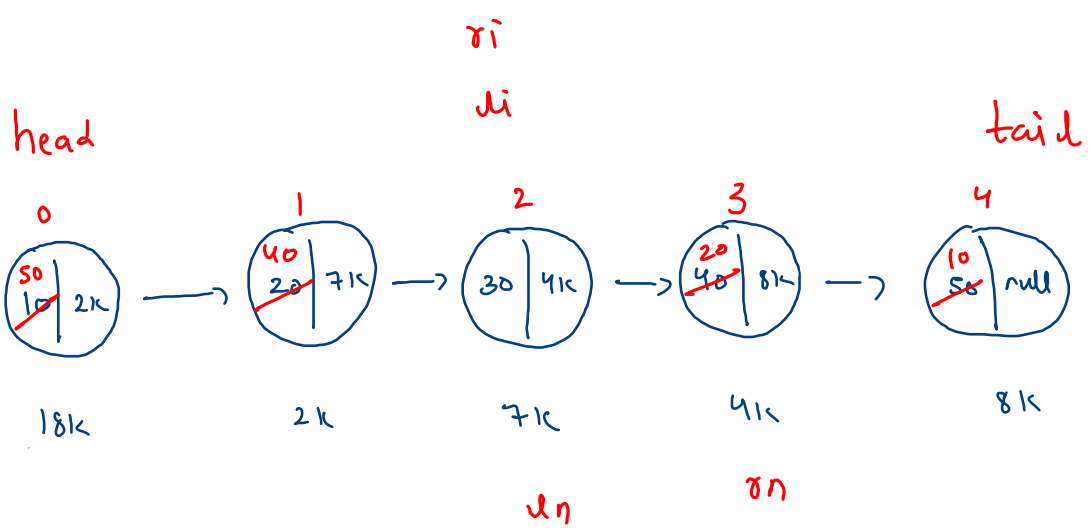
task

space X

time ✓

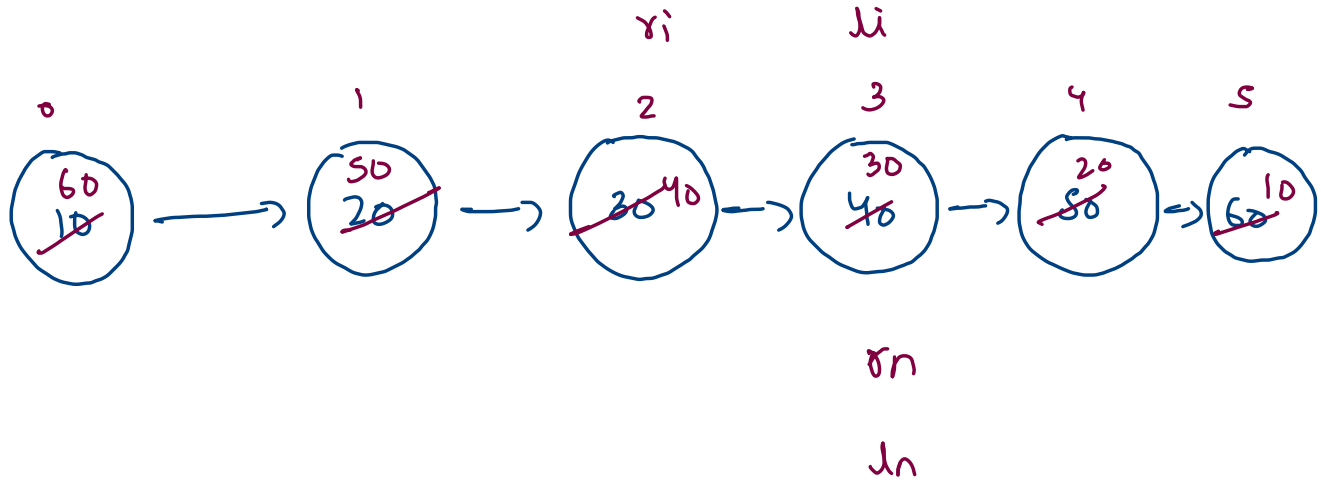
$O(n^2)$

```java
public void reverseDI() {
    Node ln = head;
    int li = 0;
    int ri = size-1;

    while(li < ri) {
        Node rn = getNodeAt(ri);

        //swap ln & rn data's
        int ld = ln.data;
        int rd = rn.data;
        ln.data = rd;
        rn.data = ld;

        ln = ln.next;
        li++;
        ri--;
    }
}
```

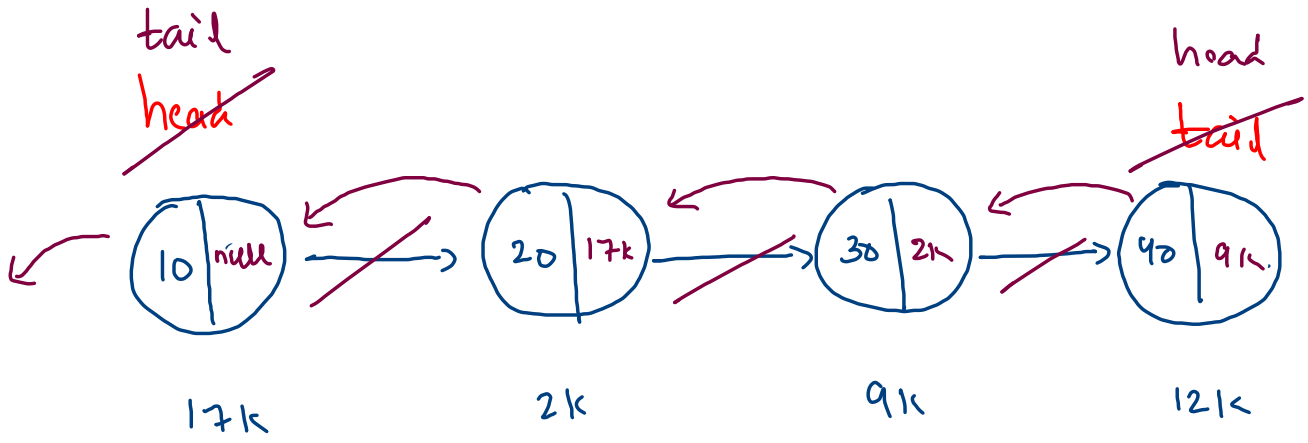


```java
public void reverseDI() {
    Node ln = head;
    int li = 0;
    int ri = size-1;

    while(li < ri) {
        Node rn = getNodeAt(ri);

        //swap ln & rn data's
        int ld = ln.data;
        int rd = rn.data;
        ln.data = rd;
        rn.data = ld;

        ln = ln.next;
        li++;
        ri--;
    }
}
```

$\frac{n}{2} * n \rightarrow O(n^2)$

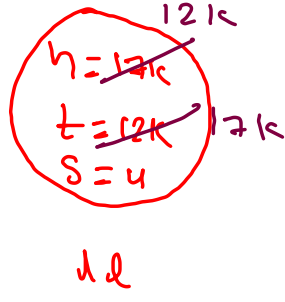


Reverse Linked List (pointer Iterative)



S X

T ✓

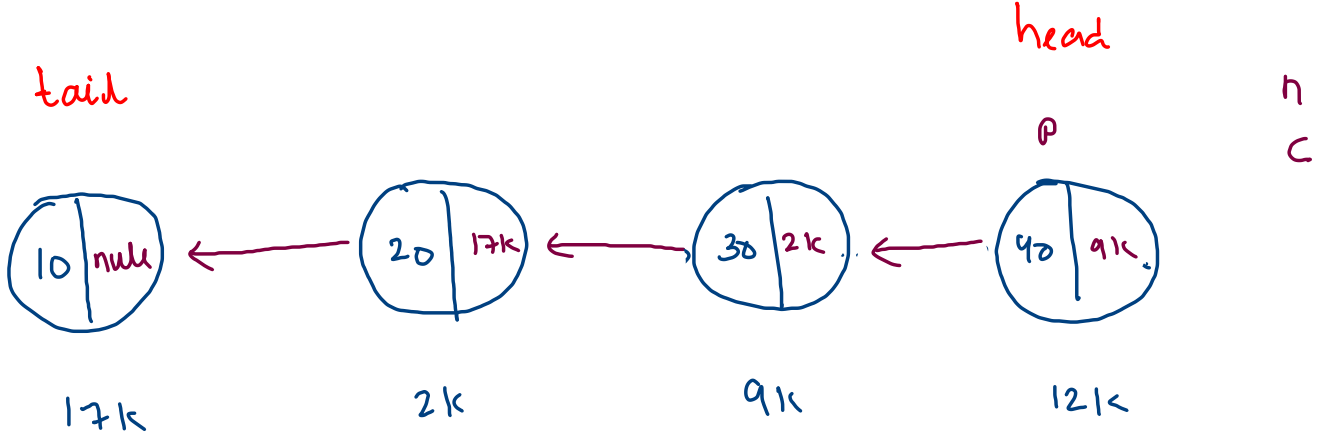


h = ~~17k~~ 12k

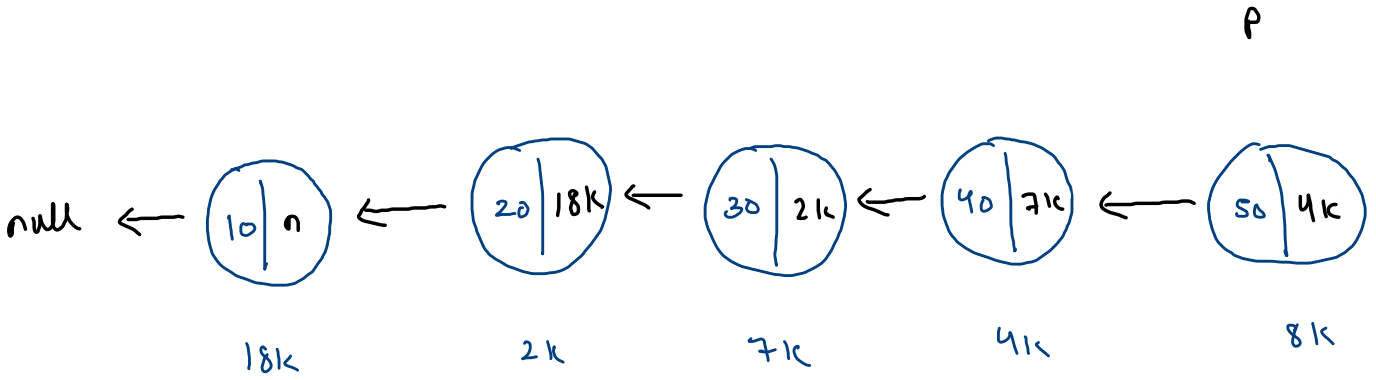
t = ~~12k~~ 17k

s = 4

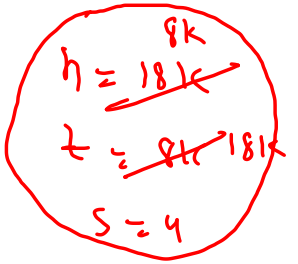
dl



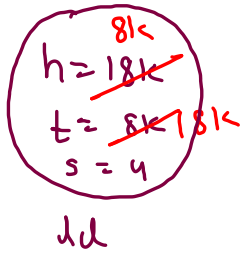

```
n = c.next // backup

c.next = prev // connection

p = c; c = n; // move
```




```
n = c->next  // backup
c->next = prev  // connection
p = c; c = n;  // move
```

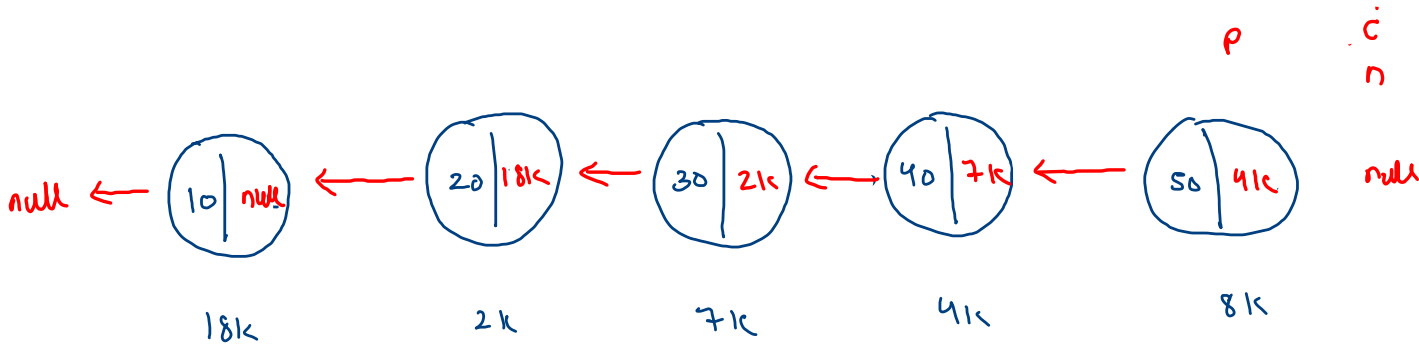


```java
public void reversePI(){
    Node prev = null;
    Node curr = head;

    while(curr != null) {
        //backup
        Node next = curr.next;

        //connection revert
        curr.next = prev;

        //move
        prev = curr;
        curr = next;
    }

    //swap head and tail
    Node temp = head;
    head = tail;
    tail = temp;
}
```



ll.reversePI()

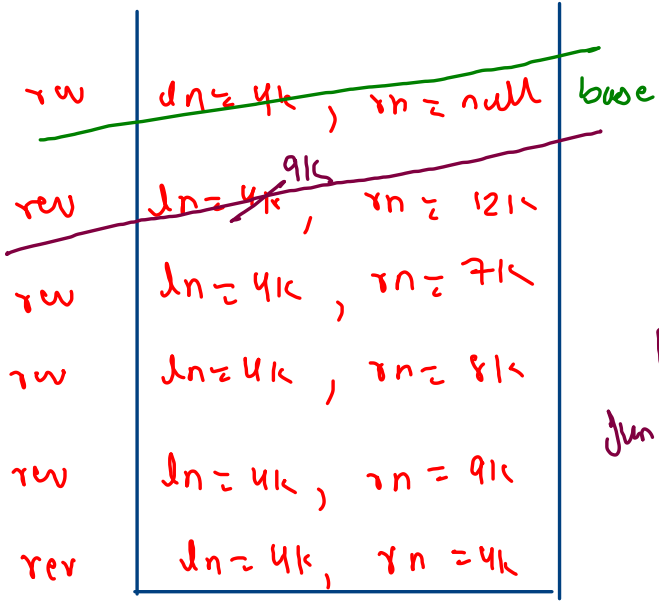
temp = 18k

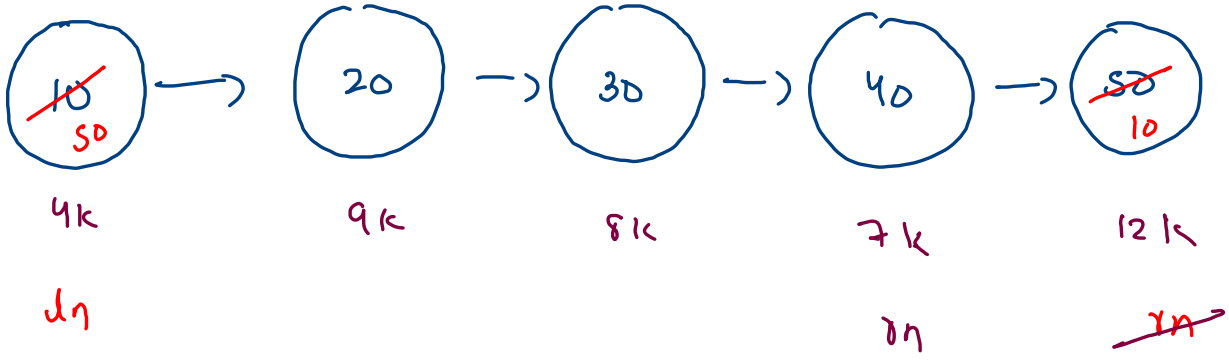


best reverse

S : O(1)
T : O(n)

Reverse A Linked List (data Recursive)





dn = dn.next

constraints
- (i) T : O(n)
- (ii) data property can be altered.
- (iii) recursion

```
LL {
Node head;
Node tail;
int size;
fun1 (Node head) {
    head.data = 100; -> heap change
    head = head.next -> stack change
}
}
```

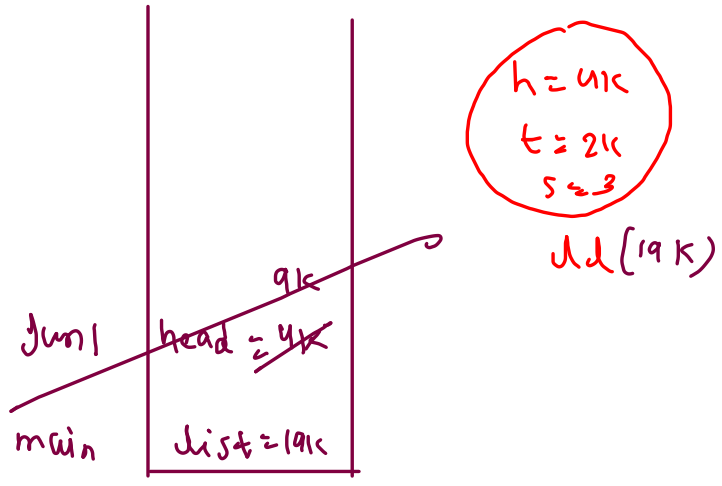
(a) 10 20 30

(b) 20 30

(c) 100 20 30

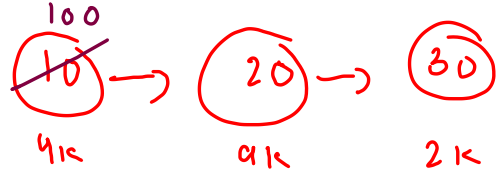
copy

```
main() {
    LL list = new LL();
    list.aL(10); list.aL(20);
    list.aL(30);
    fun1 (list.head);
    list.display() -> output
}
```



(i) java is always pass by value

(ii) heap change / stack changes



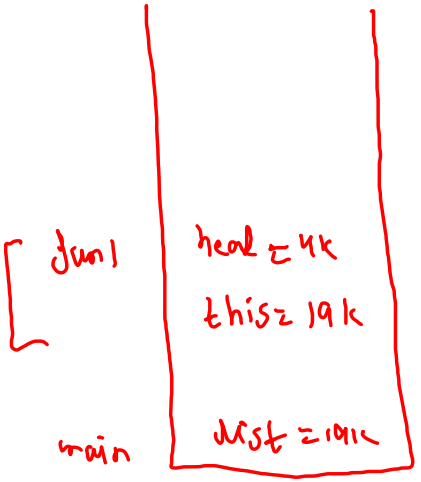
```
LL {
    Node head;
    Node tail;
    int size;
    fun1(Node head) {
        head.data = 100; --> heap change
        this.head = head.next; --> heap change
    }
}
```

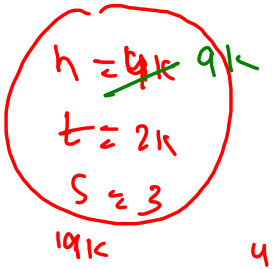
(a) 10 20 30

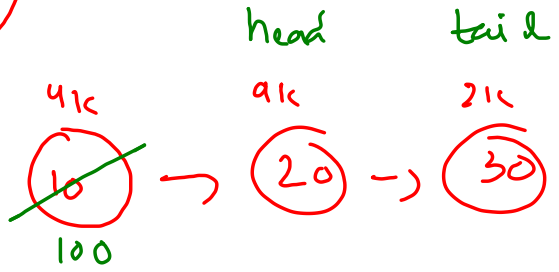
(b) 20 30

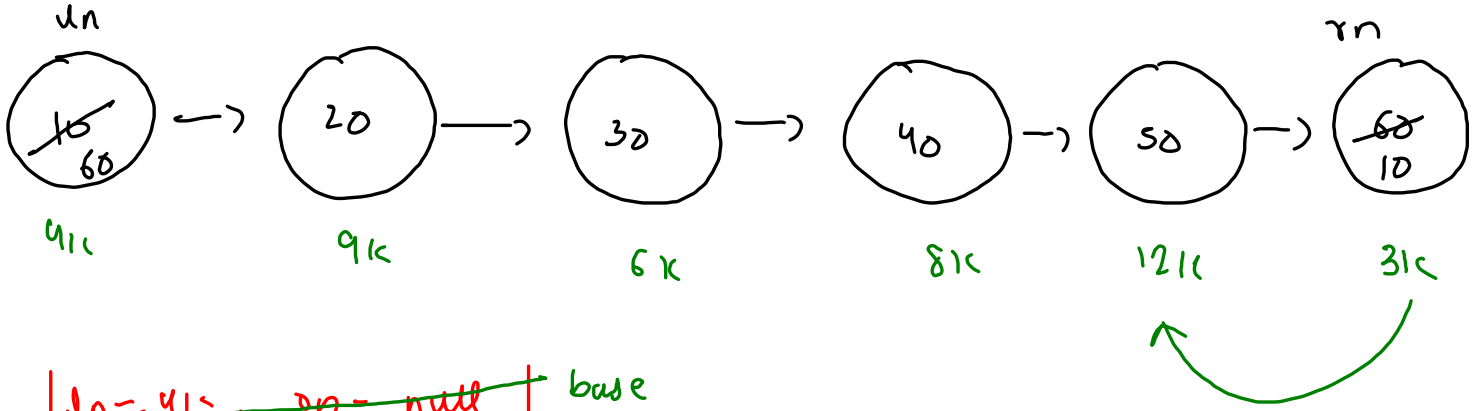
(c) 100 20 30

```
main() {
    LL list = new LL();
    list.aL(10); list.aL(20);
    list.aL(30);
    list.fun1(list.head);
    list.display() --> output
}
```









rev	ln = 4k, rn = null	base
rev	ln = 4k, rn = 3k	
rev	ln = 4k, rn = 12k	
rev	ln = 4k, rn = 8k	
rev	ln = 4k, rn = 6k	
rev	ln = 4k, rn = 9k	
rev	ln = 4k, rn = 4k	

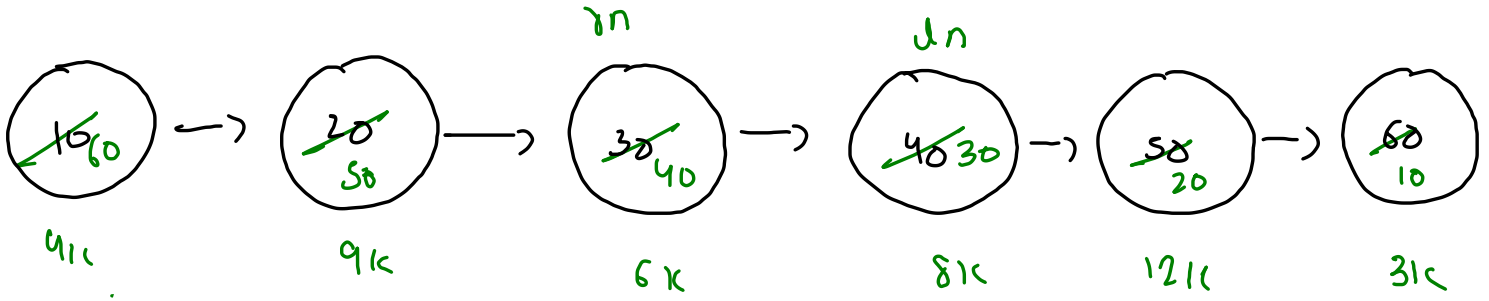
h = 4k
t = 3k
s = 6

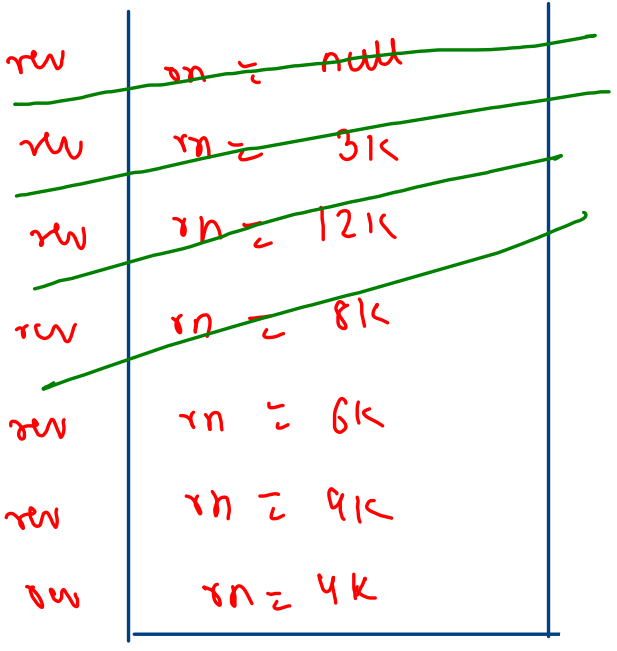
```
rev ( Node ln, Node rn) {
```

```
call -> [ rev ( ln, rn.next);
          swap(ln, rn);
          ln = ln.next -> Stack
```

change

3





h = 41k, t = 31k, size = 6

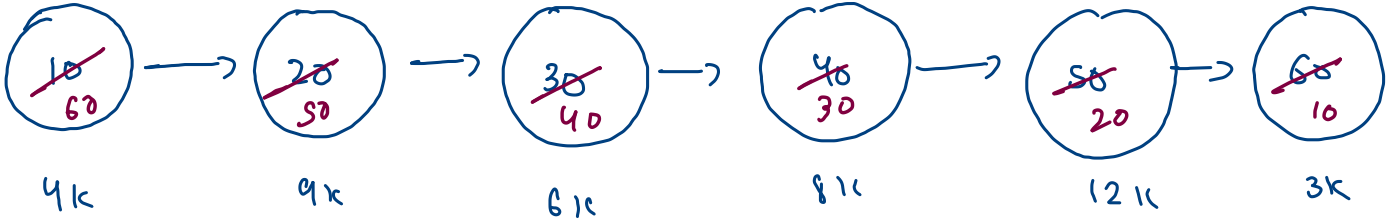
dn = ~~41k~~ ~~91k~~ ~~61k~~ 81k

191k (dd)

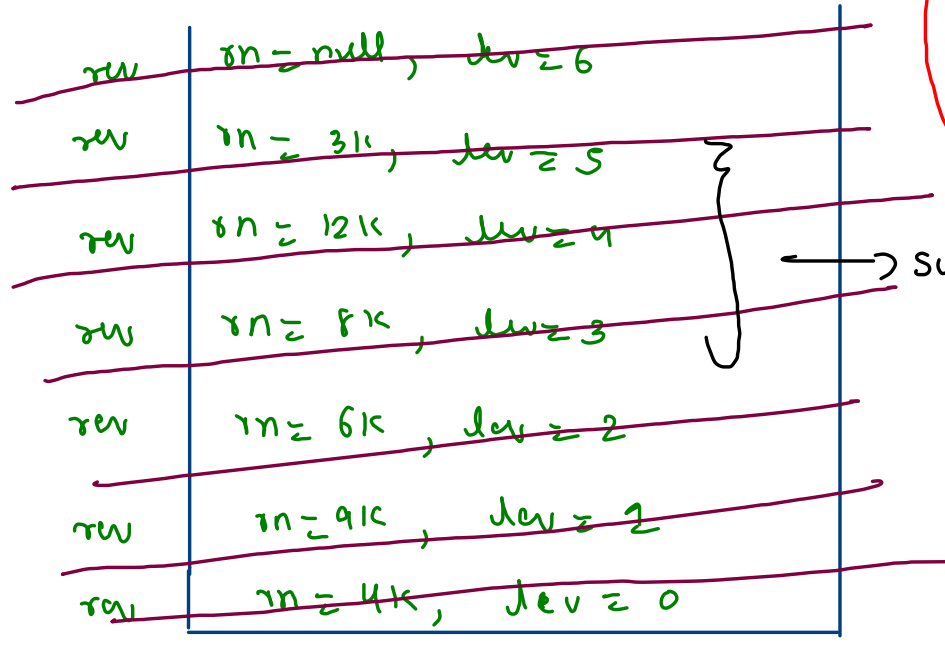
this

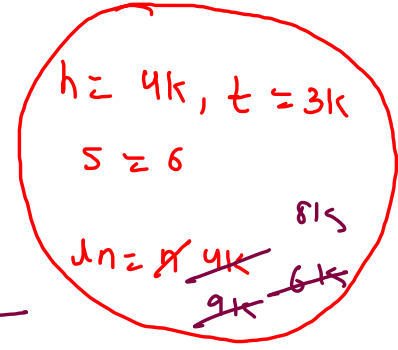
```
rev (Node rn) {
    rev(rn.next);
    swap(dn, rn);
    dn = dn.next;
}
```

this.dn = this.dn.next heap change




```java
Node ln;
public void reverseDR() {
  ln = head;
  reverseDRH(head,0);
}

public void reverseDRH(Node rn,int lev) {
    if(rn == null) {
        return;
    }

    reverseDRH(rn.next,lev + 1);

    if(lev >= (size/2)) {
        //swap
        int ld = ln.data;
        int rd = rn.data;
        ln.data = rd;
        rn.data = ld;

        ln = ln.next; //heap change (this.ln = this.ln.next)
    }
}
```





$\frac{size}{2} = 3$