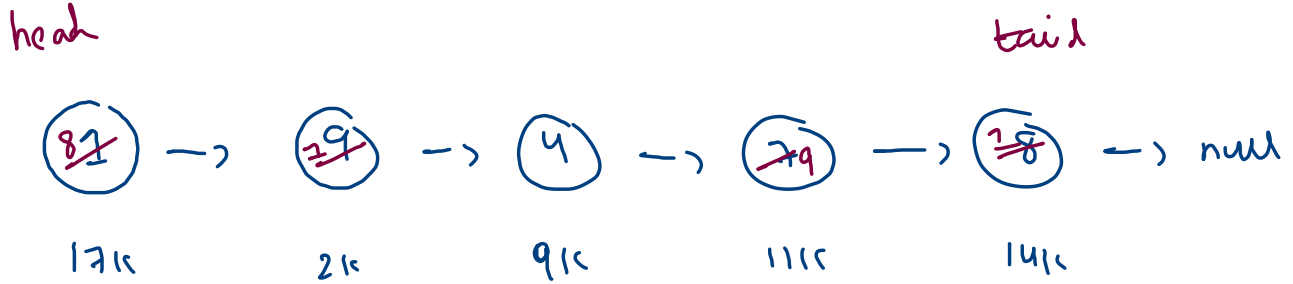
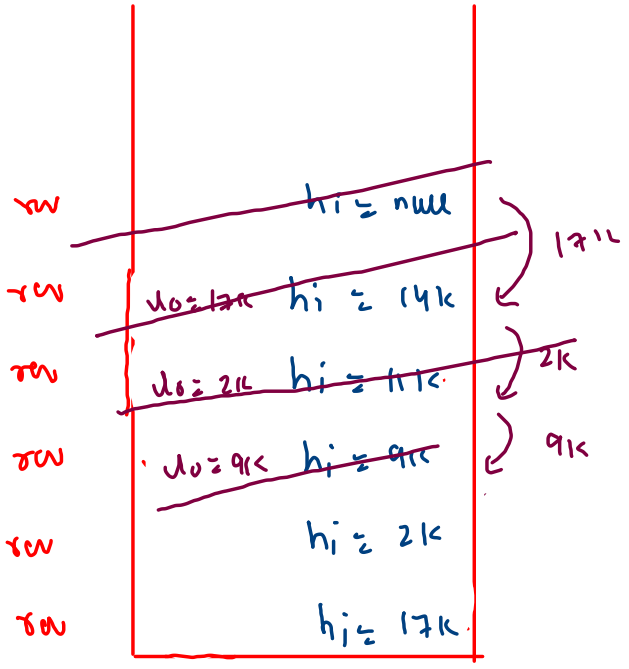
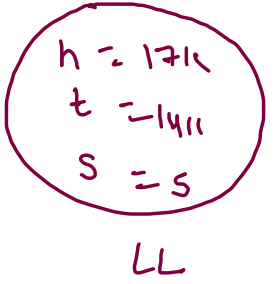
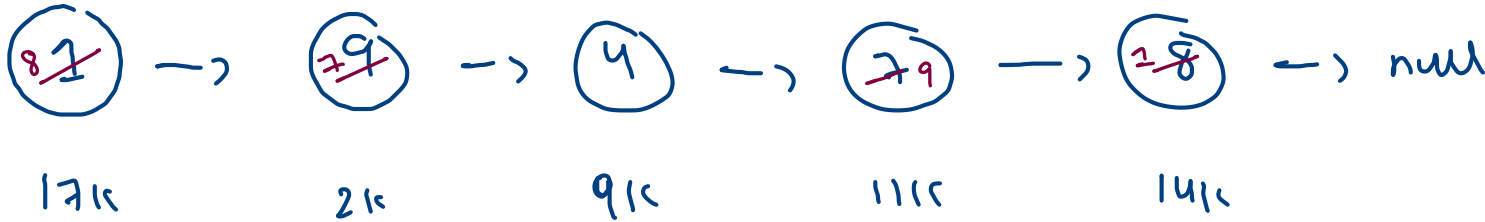
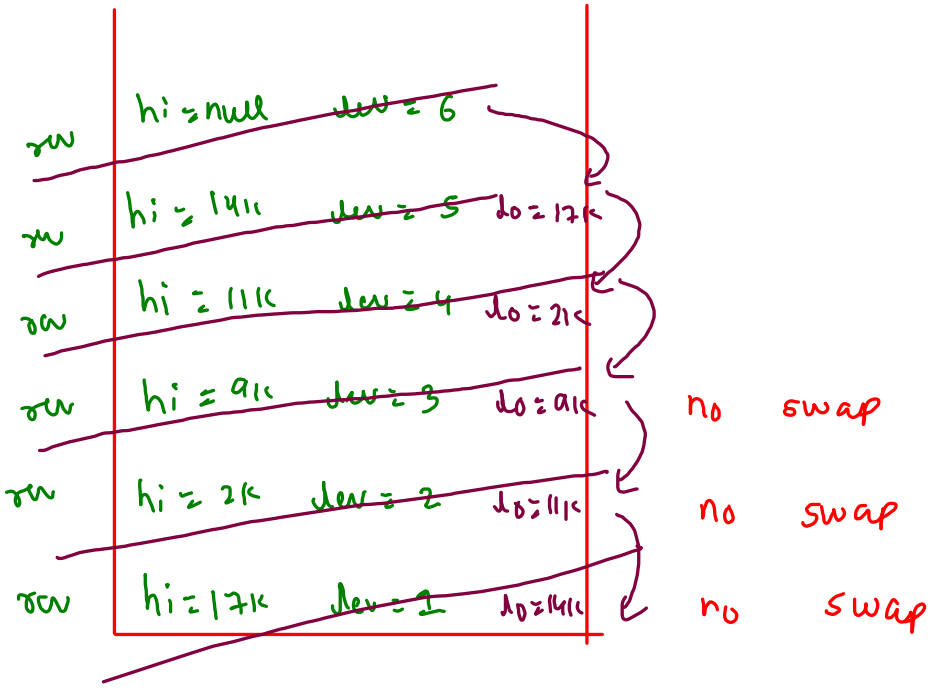

head
tail
1
9
4
9
8
null
17k
2k
9k
11k
14k


h1 = null
17k
h1 = 14k
2k
h1 = 11k
9k
h1 = 2k
h1 = 17k

O(n)


h = 17k
t = 14k
s = 5
LL





1 -> 9 -> 4 -> 7 -> 8

8 -> 7 -> 4 -> 9 -> 1

size -> 7

dew -> 7, 6, 5
1 2 3

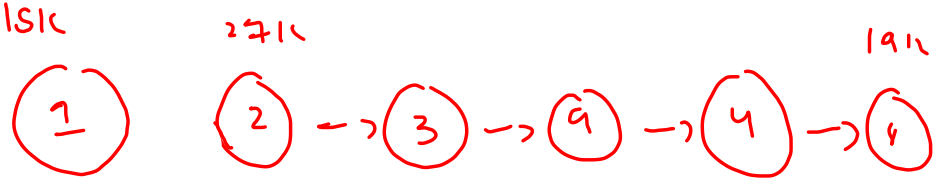
size -> 8

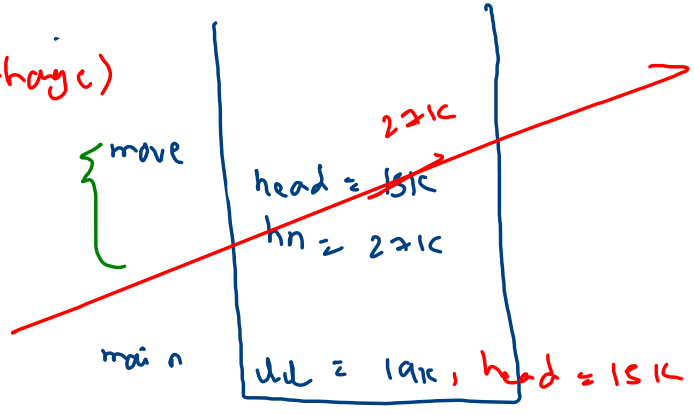
dew -> 8 7 6 5
1 2 3 4

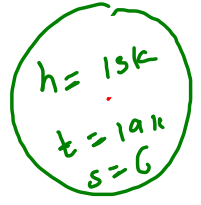
dew > (size / 2)

```
LL {
    Node head;
    Node tail;
    Node size;

move (Node head) {
    1. Node hn = head.next;
    2. head.next = null; (heap change)
    3. head = hn; (stack change)
}
}
```

```
main() {
    Node head = ll.head;
    move (head);
}
```

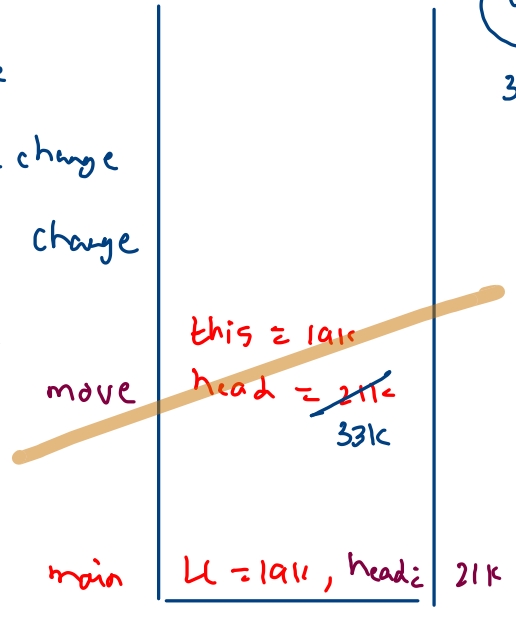


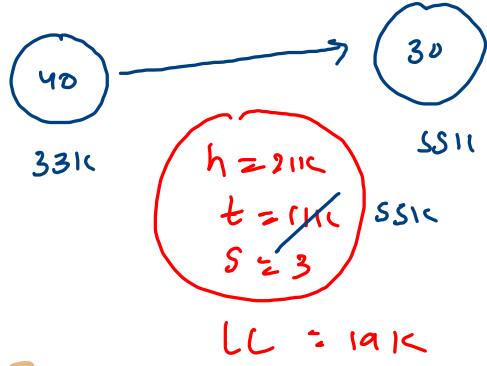


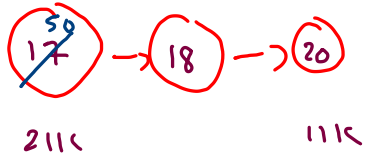


```
LL {
    Node head;
    Node tail;
    Node size;

    move (Node head) {
        {head.data = 50;      heap change
         head = new node(40); stack change
         tail = new Node(30); heap change
         head.next = tail;    heap change
        }
    }
}
```

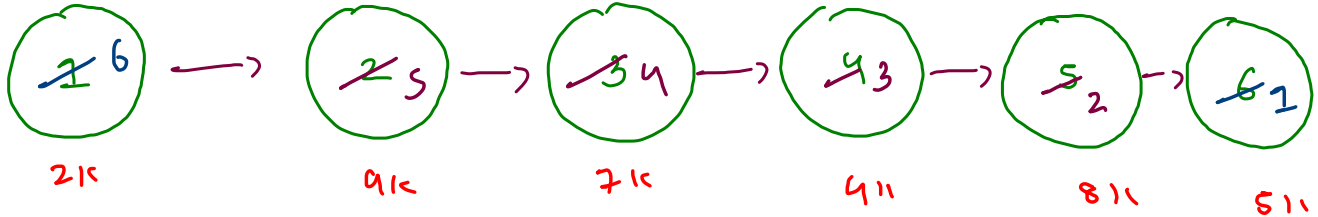
```
main {
    Node head = ll.head;
    ll.move(head);
}
```

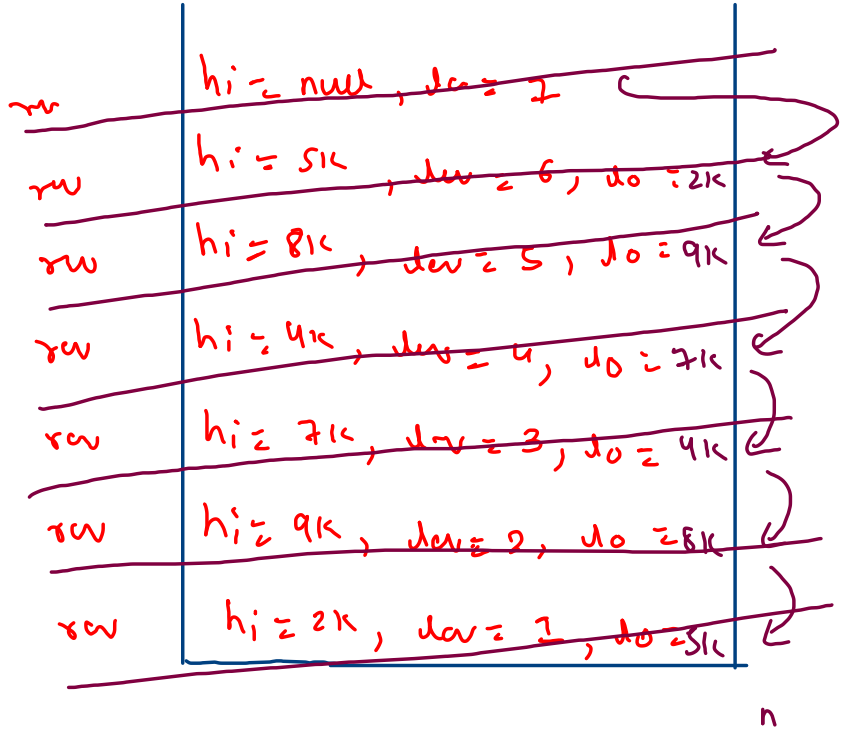






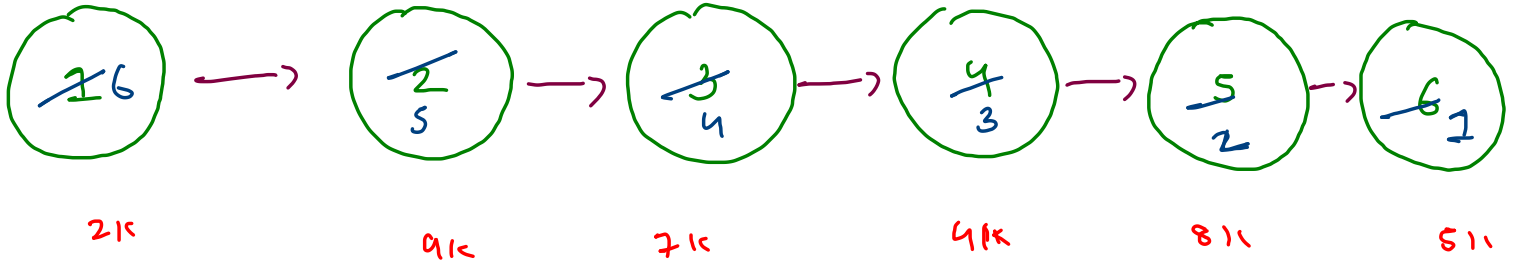
```java
private Node reverseDRhelper(Node hi,int lev) {
    if(hi == null) {
        return head;
    }

    Node lo = reverseDRhelper(hi.next,lev+1);

    if(lev > size/2) {
        //swap
        int temp = lo.data;
        lo.data = hi.data;
        hi.data = temp;
    }
    else {
        //no swapping
    }

    return lo.next;
}
```

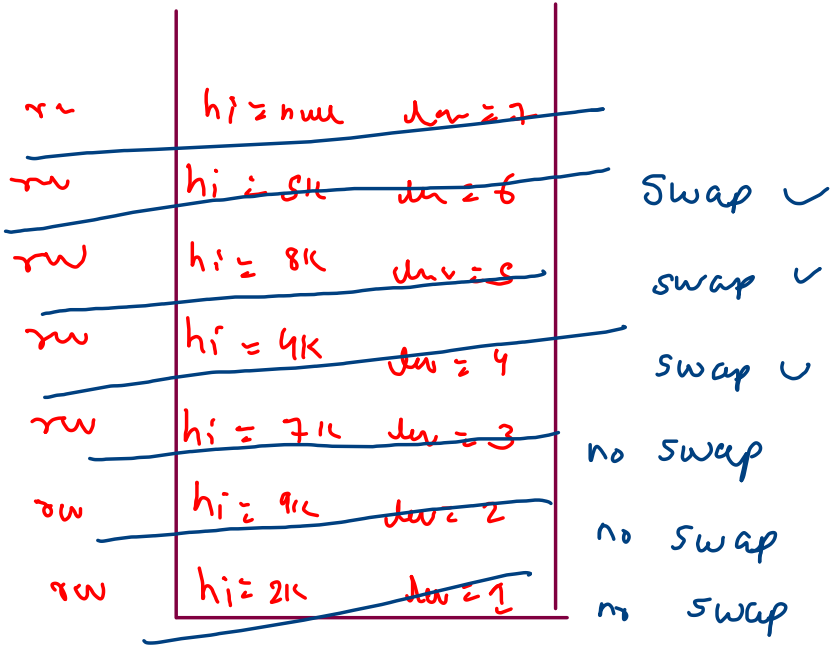




size = 6

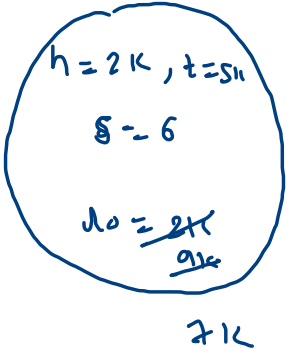
$\frac{size}{2} = 3$

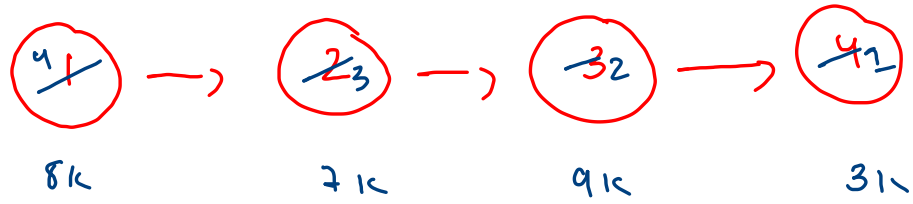




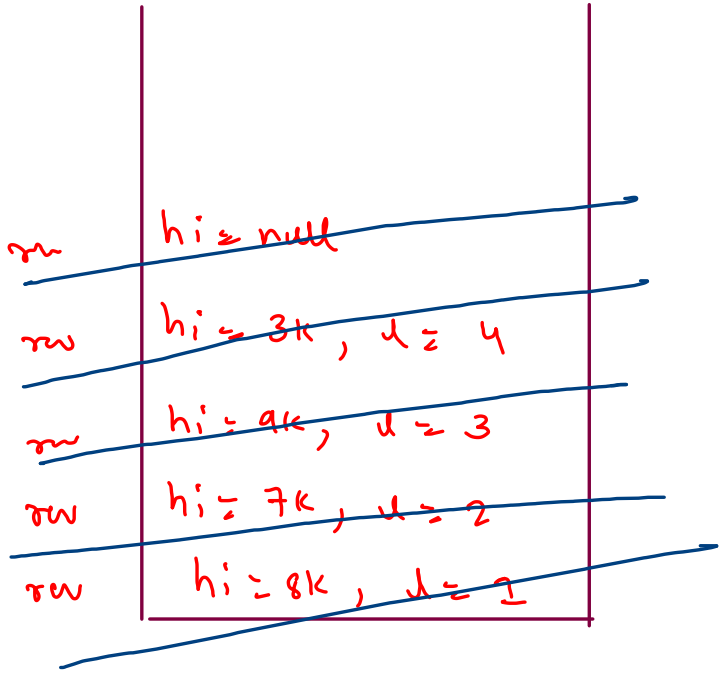
heap change

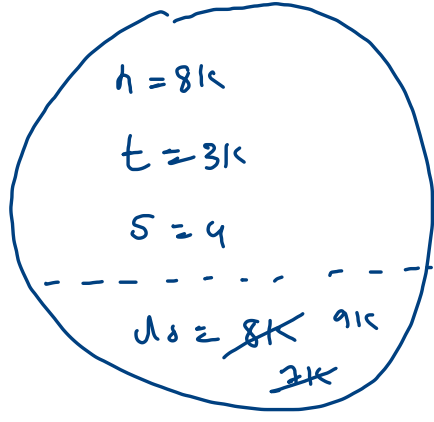
```
do = do.next
```



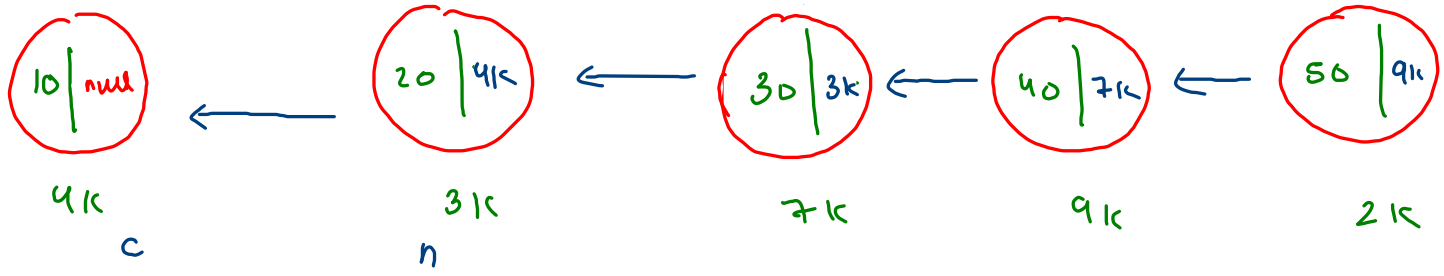


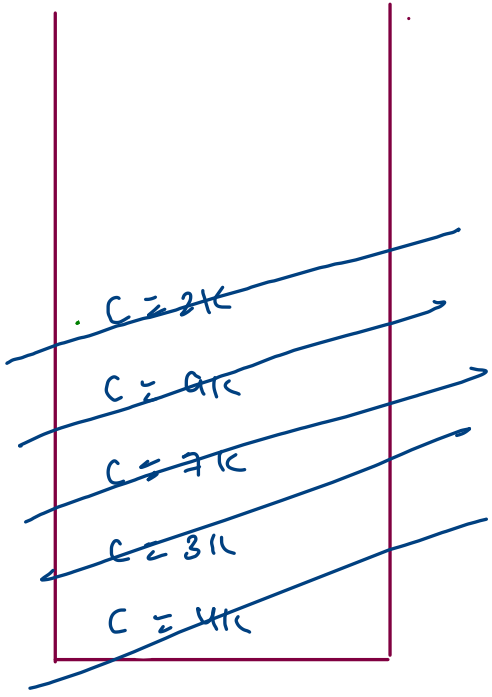

```java
private void reverseDRhelper(Node hi, int lev) {
    if (hi == null) {
        return;
    }

    reverseDRhelper(hi.next, lev + 1);

    if (lev > size / 2) {
        int temp = lo.data;
        lo.data = hi.data;
        hi.data = temp;

        lo = lo.next; //heap change
    } else {
        //no swapping
    }

}

Node lo;
public void reverseDR() {
    // write your code here
    lo = head;
    reverseDRhelper(head, 1);
}
```





$\frac{size}{2} = 4$



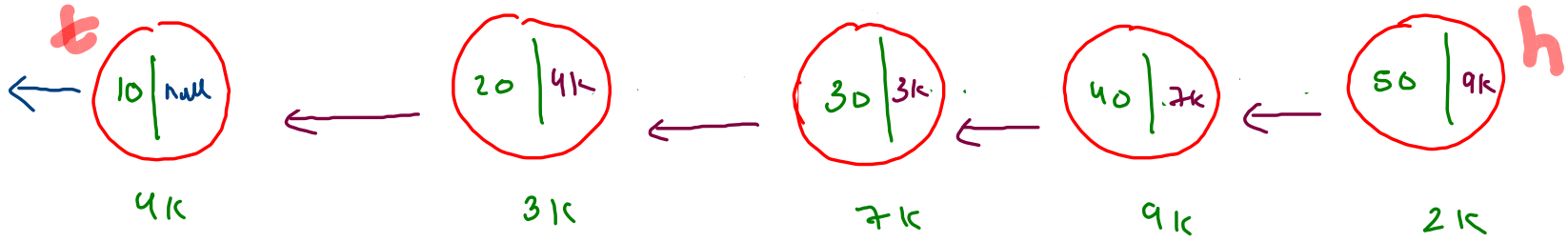


n.next = c

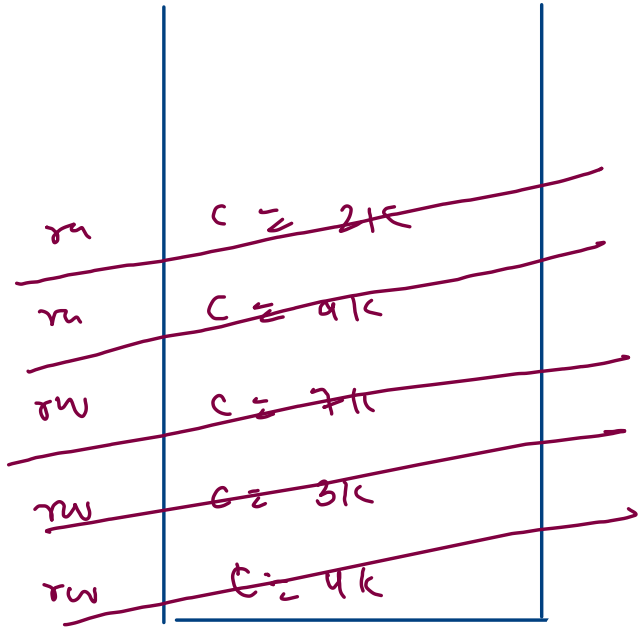
Final :

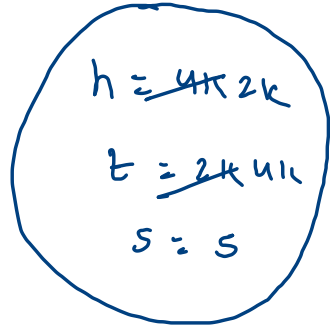
head.next = null

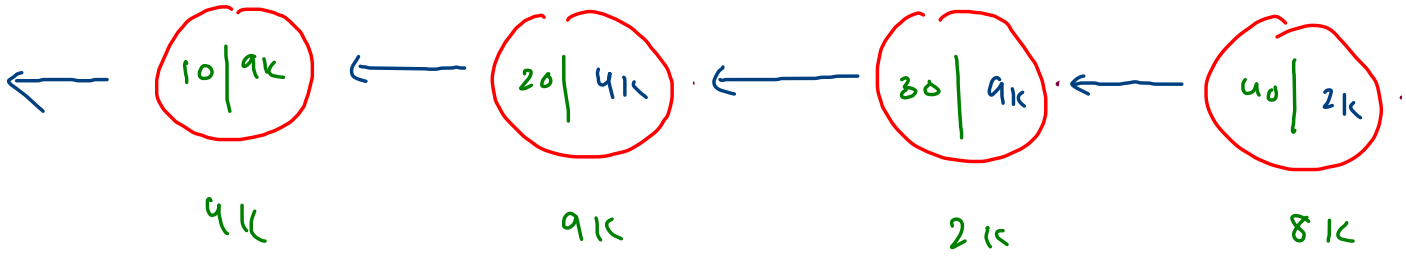
swap (head, tail)




```java
private void reversePRHelper(Node curr){
    if(curr == tail) {
        return;
    }

    reversePRHelper(curr.next);

    Node n = curr.next; //next node
    n.next = curr;

}

public void reversePR(){
  // write your code here
  reversePRHelper(head);

  //final steps
  head.next = null;

  Node temp = head;
  head = tail;
  tail = temp;
}
```







rw — c = null

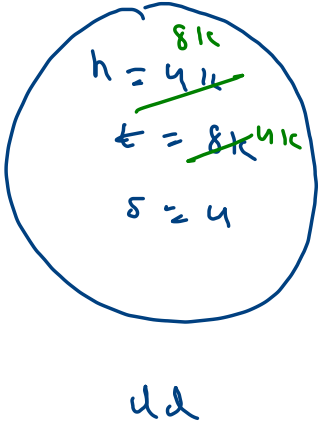
rw — c = 8k, p = 2k, n = null

rw — c = 2k, p = 9k, n = 8k

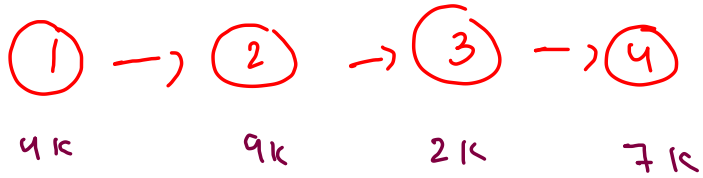
rev — c = 9k, p = 4k, n = 2k

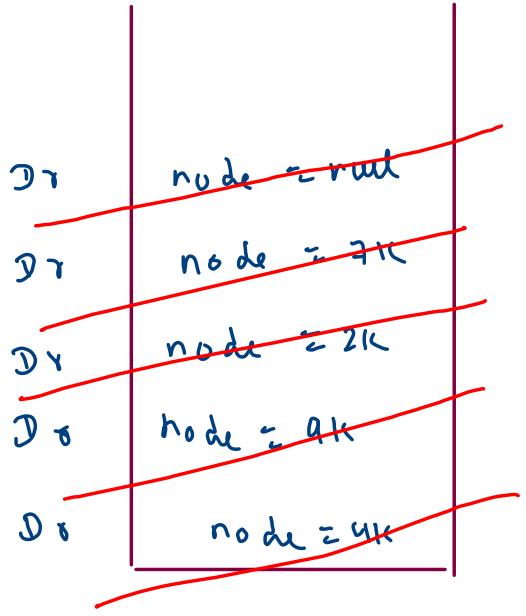
rw — c = 4k, p = null, n = 9k

c.next = p

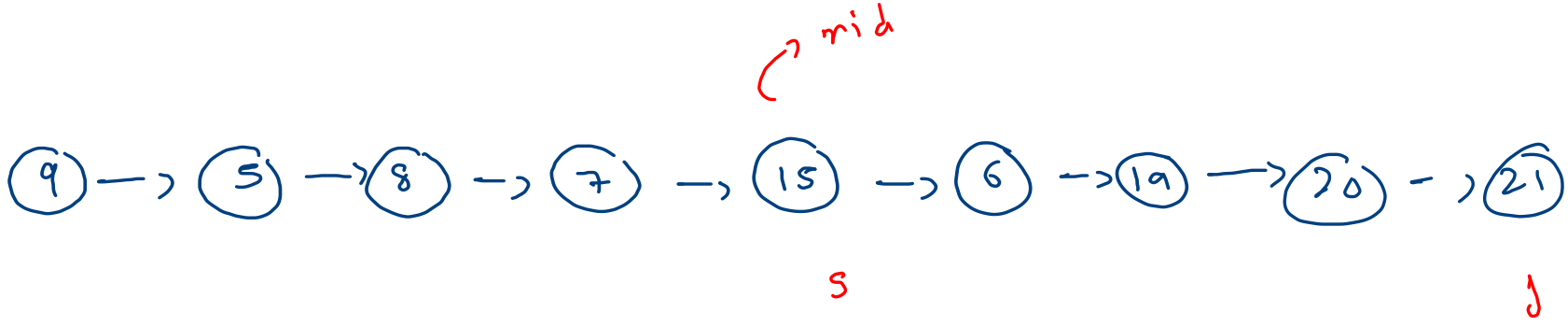


dd





4 3 2 1




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

    while(fast.next != null && fast.next.next != null) {
        //move slow by 1
        slow = slow.next;

        //move fast by 2
        fast = fast.next.next;
    }

    return slow.data;
}
```

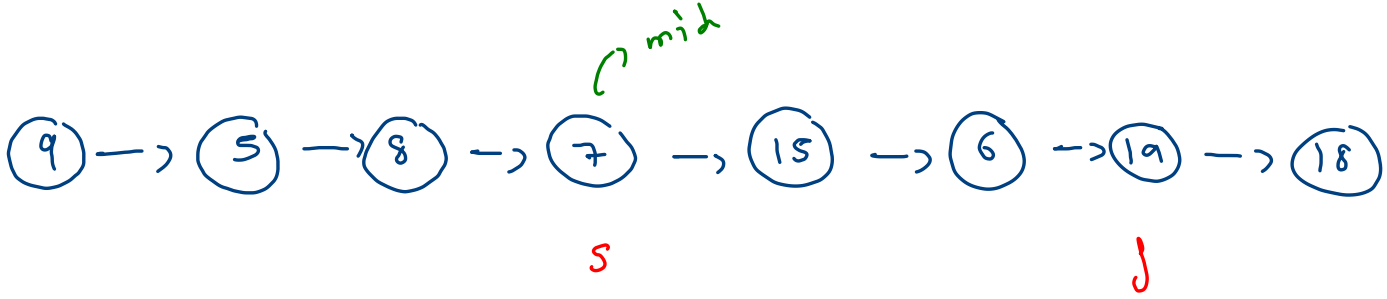
15

s = 1

f = 2

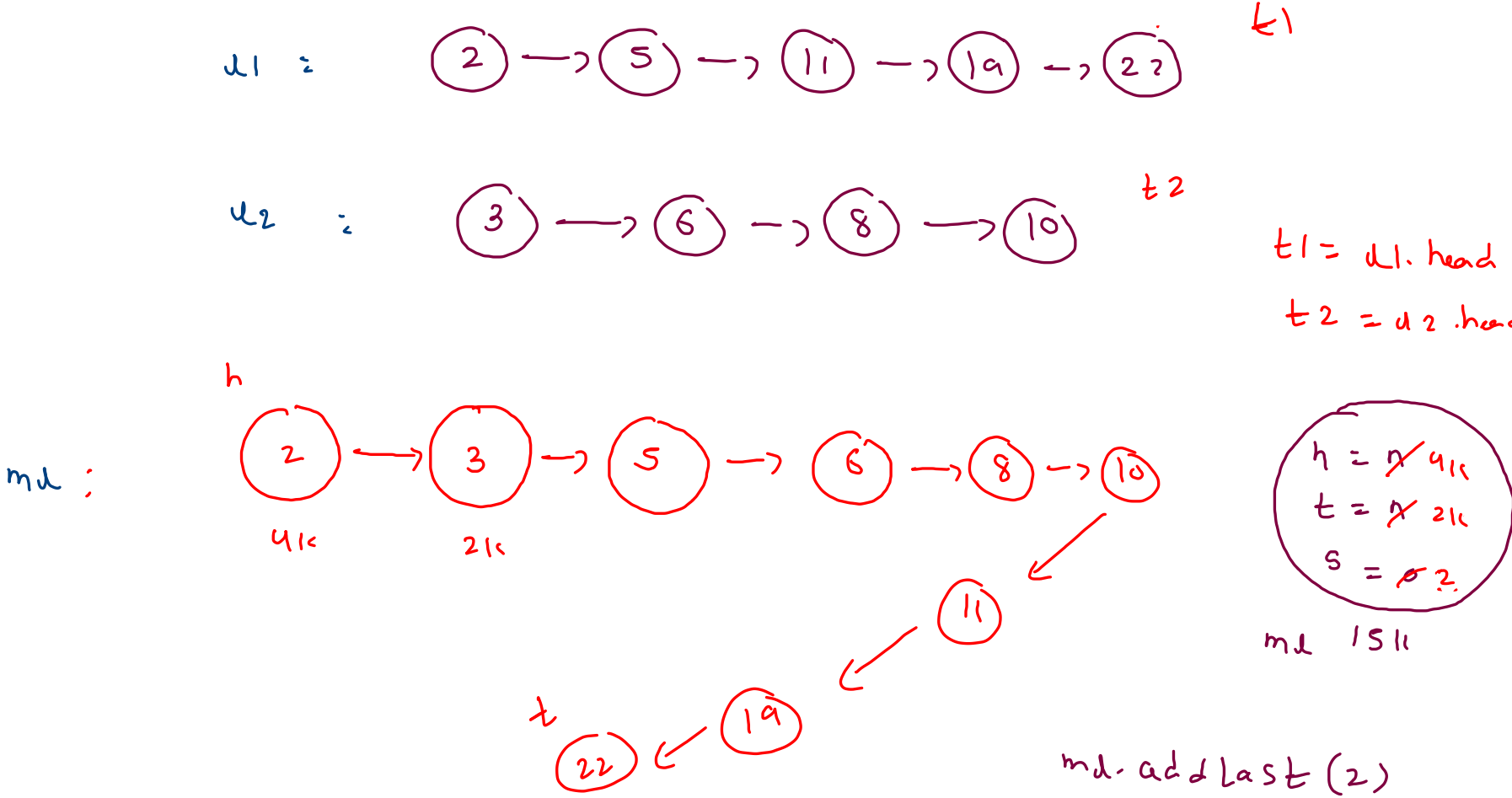
2 vs = vf

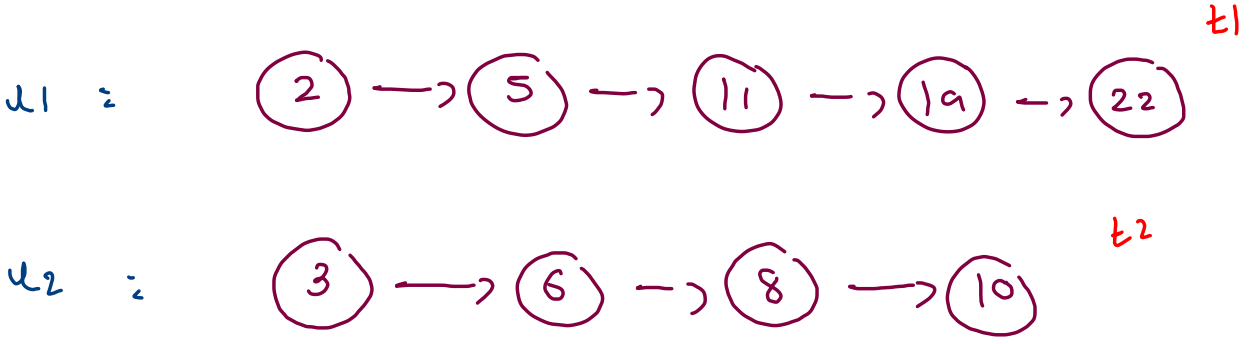
stop -> fast.next == null



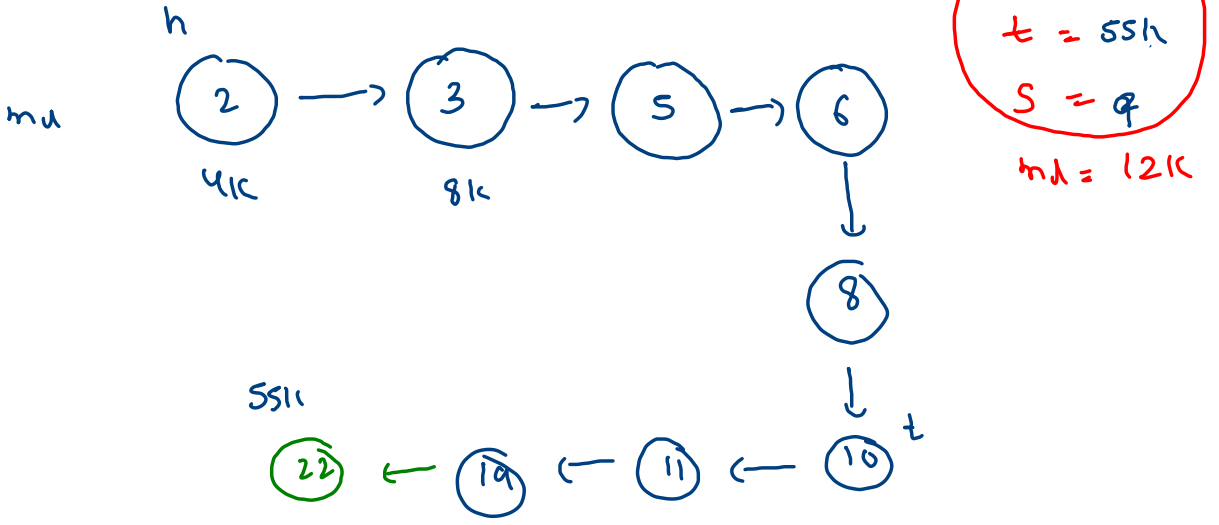

```
public int mid(){
    Node slow = head;
    Node fast = head;

    while(fast.next != null && fast.next.next != null) {
        //move slow by 1
        slow = slow.next;

        //move fast by 2
        fast = fast.next.next;
    }

    return slow.data;
}
```

stop : fast.next.next == null


l1 :
2
5
11
19
22
t1
l2 :
3
6
8
10
t2
t1 = l1.head
t2 = l2.head
h
ml :
2
3
5
6
8
10
11
19
22
t
h = ✗ 4lk
t = ✗ 2lk
s = ø 2
ml 1511
ml.addLast(2)
ml.addLast(3)




```java
public static LinkedList mergeTwoSortedLists(LinkedList l1, LinkedList l2) {
    // write your code here
    Node t1 = l1.head;
    Node t2 = l2.head;

    LinkedList ml = new LinkedList();

    while(t1 != null && t2 != null) {
        if(t1.data <= t2.data) {
            ml.addLast(t1.data);
            t1 = t1.next;
        }
        else {
            ml.addLast(t2.data);
            t2 = t2.next;
        }
    }

    while(t1 != null) {
        ml.addLast(t1.data);
        t1 = t1.next;
    }

    while(t2 != null) {
        ml.addLast(t2.data);
        t2 = t2.next;
    }

    return ml;
}
```



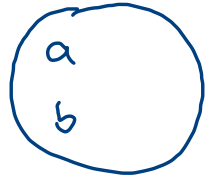
```
class A {
    int a;
    int b;

    static int c;

}
```

data member

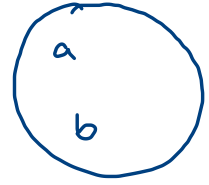
class variable

```
main {

    A o1 = new A();

    A o2 = new A();

}
```

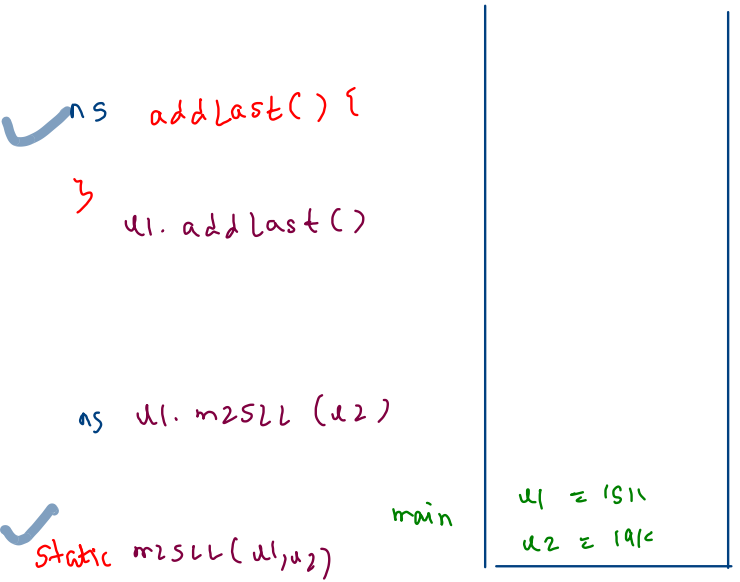
c

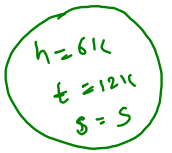


o1



o2

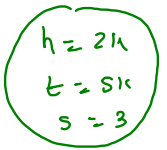




u2 = 191c

u1.addLast()

u2.addLast()



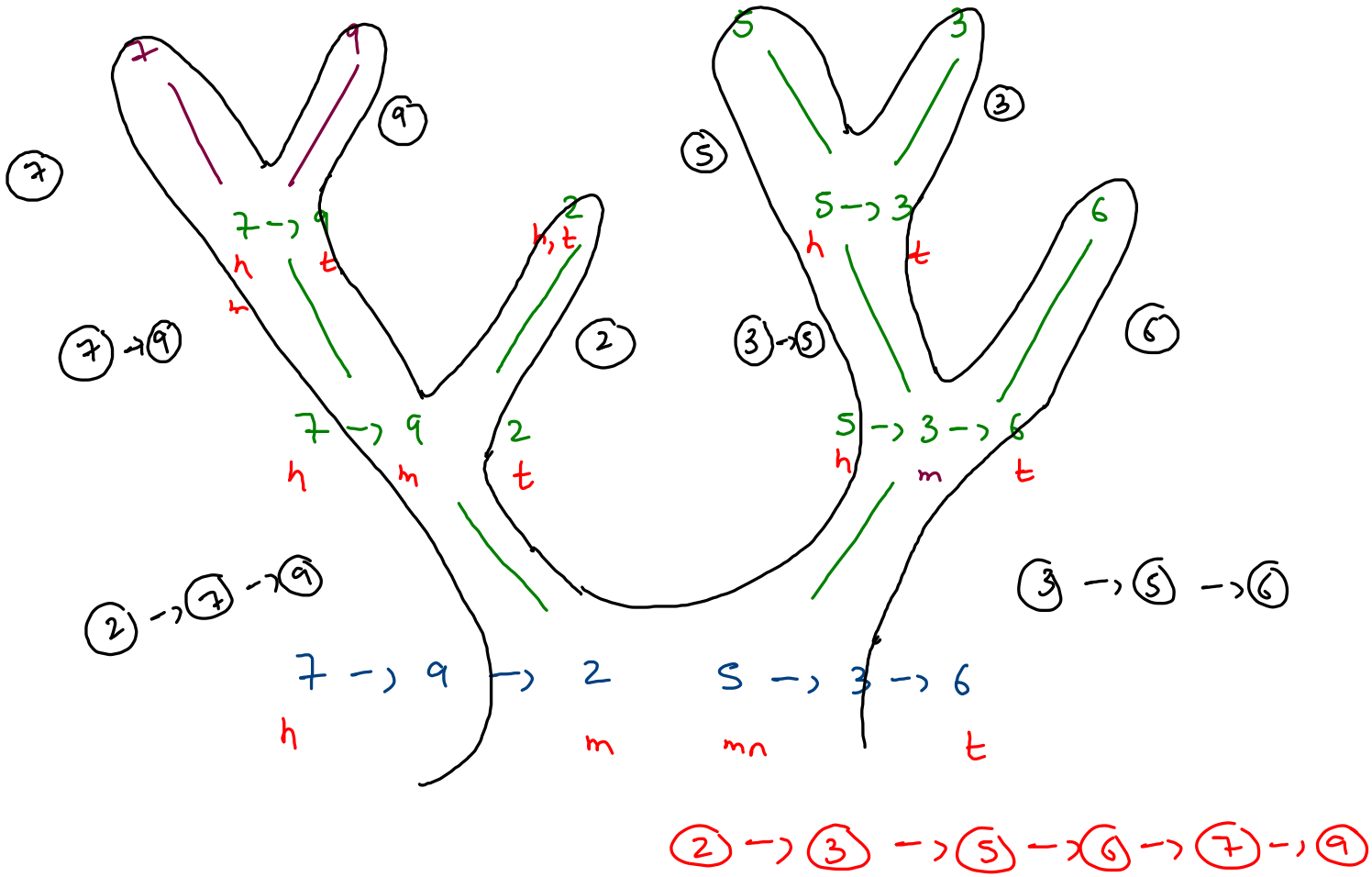
u1 = 151c

static -> class thing,
not bounded with obj

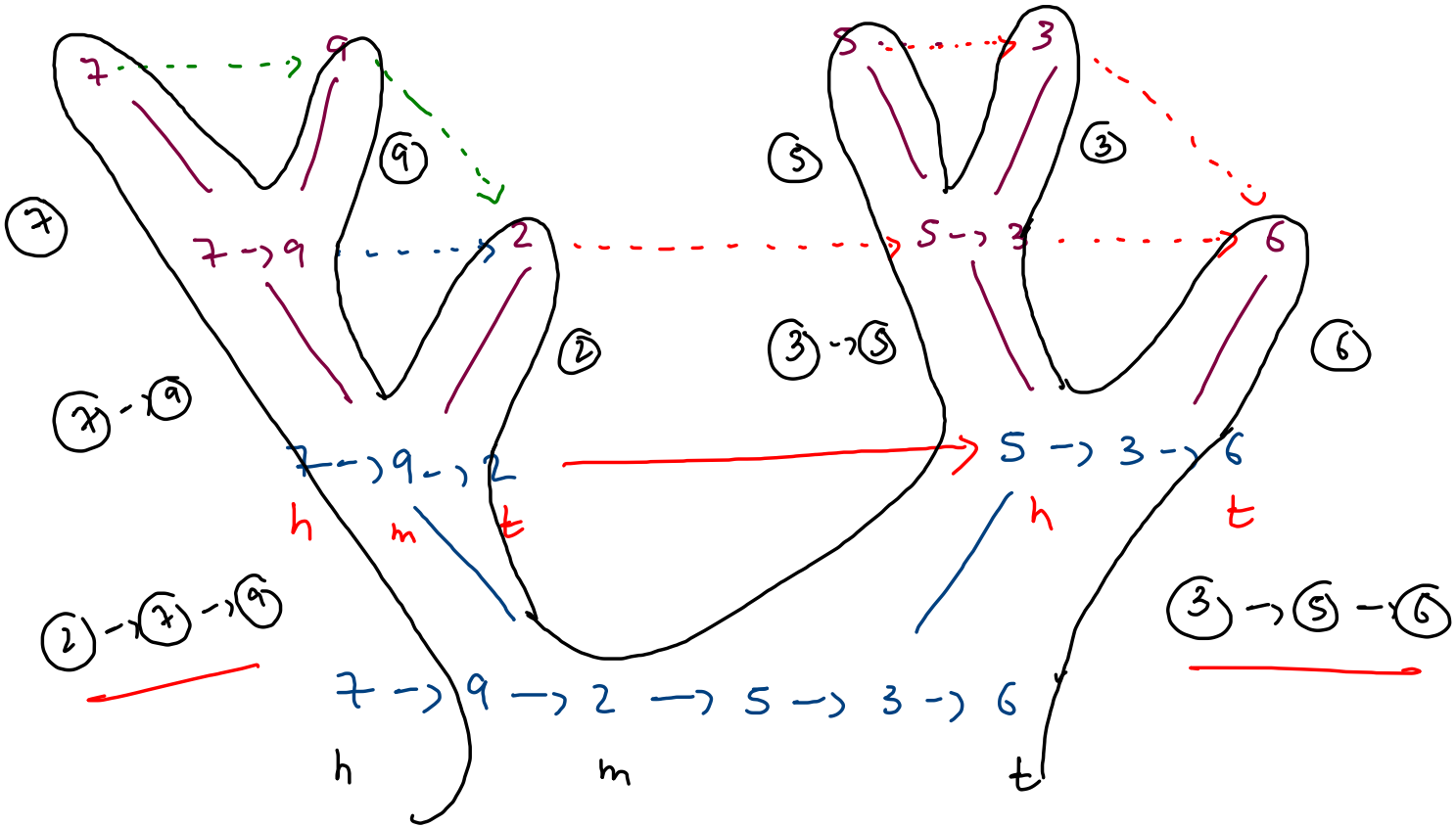
non-static -> bounded with obj

add2LL ⟶ static

updatehead (10) ⟶ non-static


7
9
7 -> 9
h
t
m
2
h, t
7 -> 9
2
h
m
t
5
3
5 -> 3
h
t
6
5 -> 3 -> 6
h
m
t
7 -> 9 -> 2
h
m
5 -> 3 -> 6
mn
t
m2SLL
2 -> 3 -> 5 -> 6 -> 7 -> 9

virtually smalling problem



2 -> 3 -> 5 -> 6 -> 7 -> 9