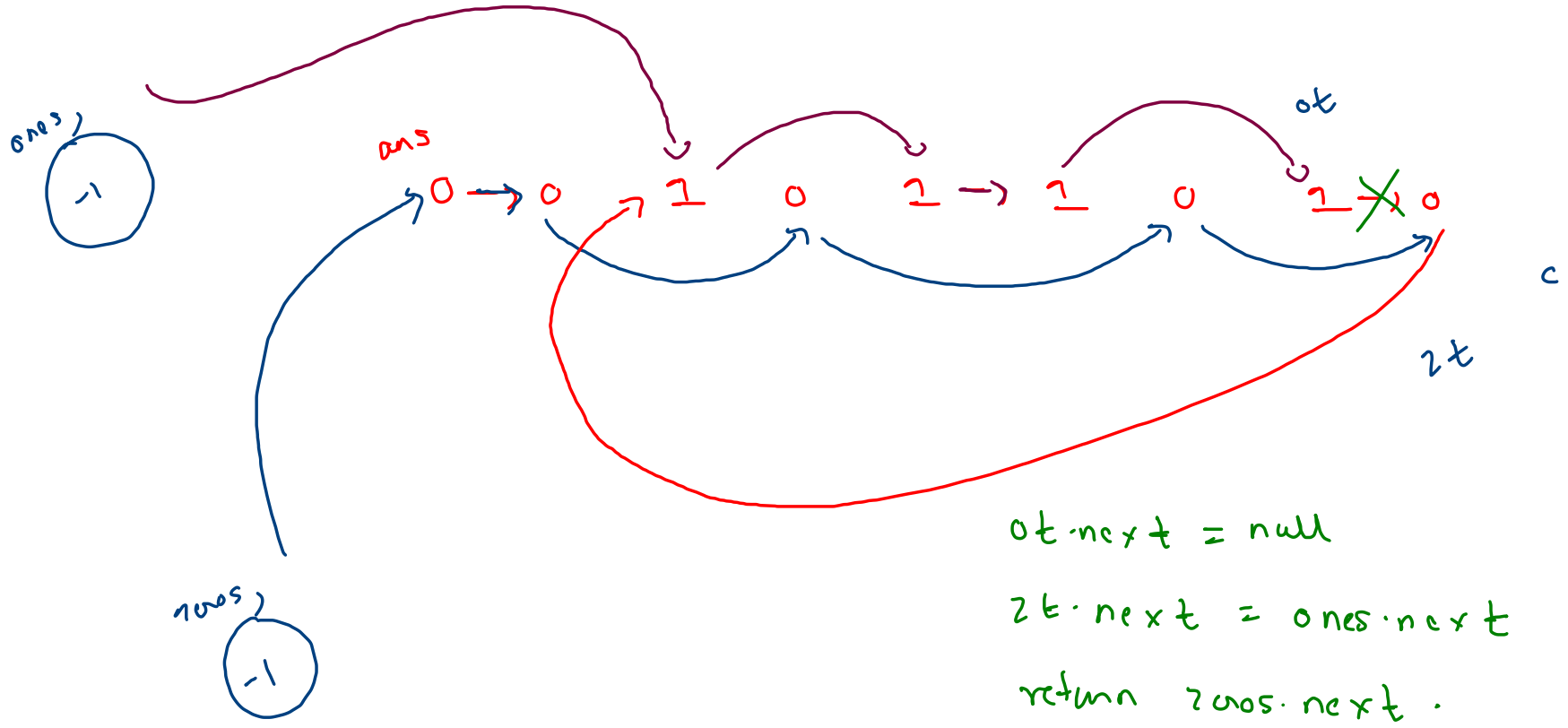
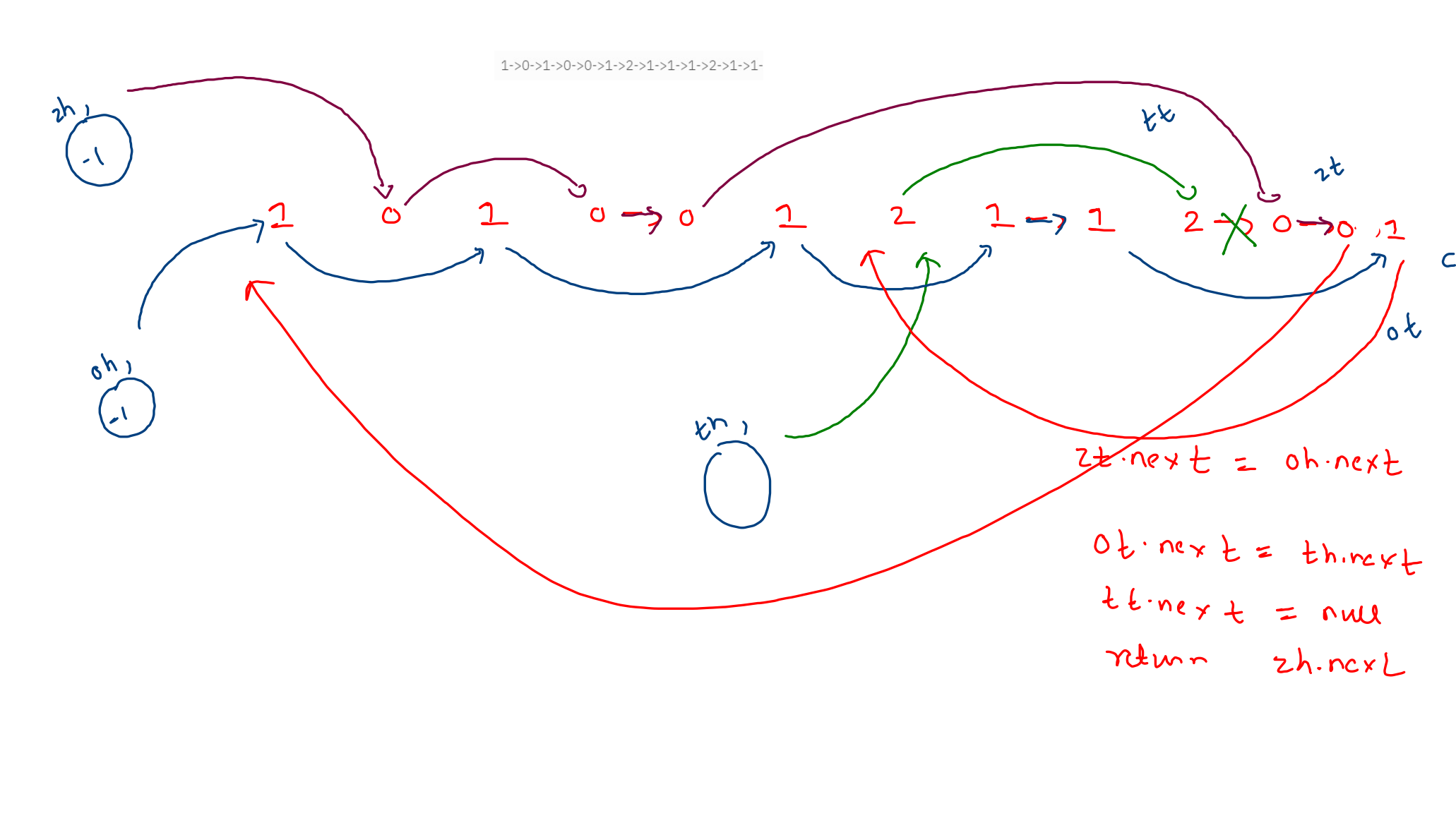


ot.next = null

2t.next = ones.next

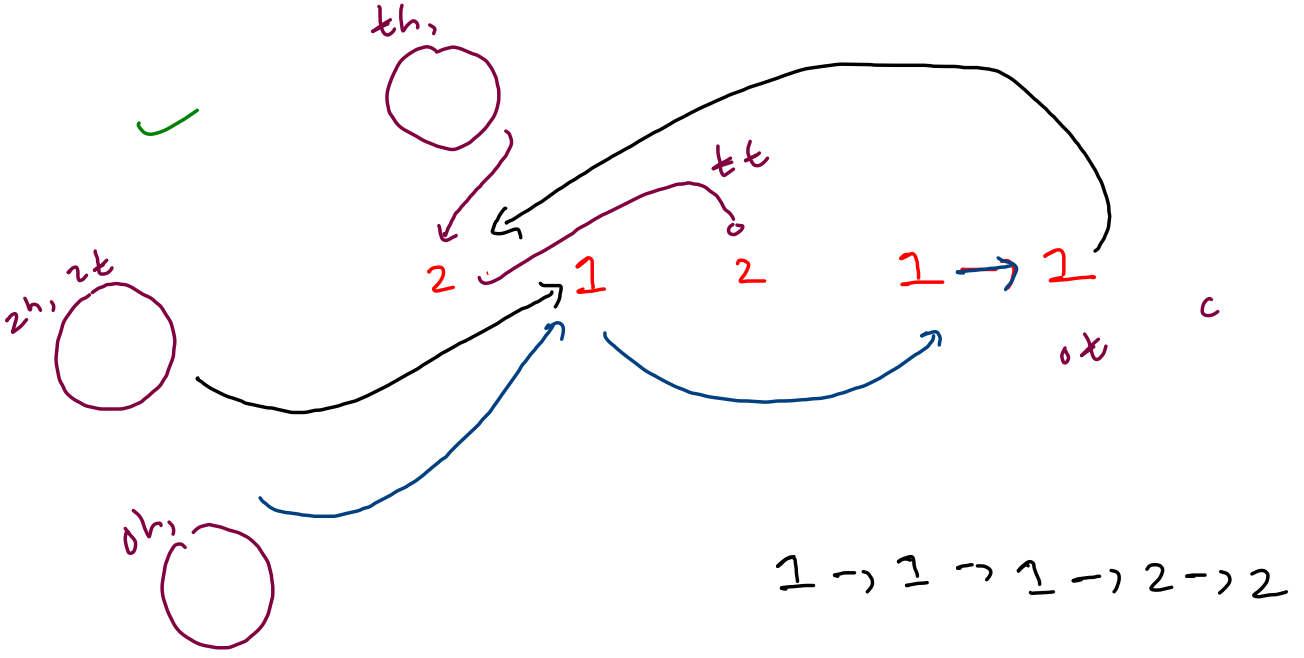
return zeros.next.


1->0->1->0->0->1->2->1->1->1->2->1->1-
zh
-1
tt
zt
1
0
1
0
0
1
2
1
1
2
0
0
1
oh
-1
th
ot
zt.next = oh.next
ot.next = th.next
tt.next = null
return zh.next

(i) 0 x ✓

(ii) 1 x ✗

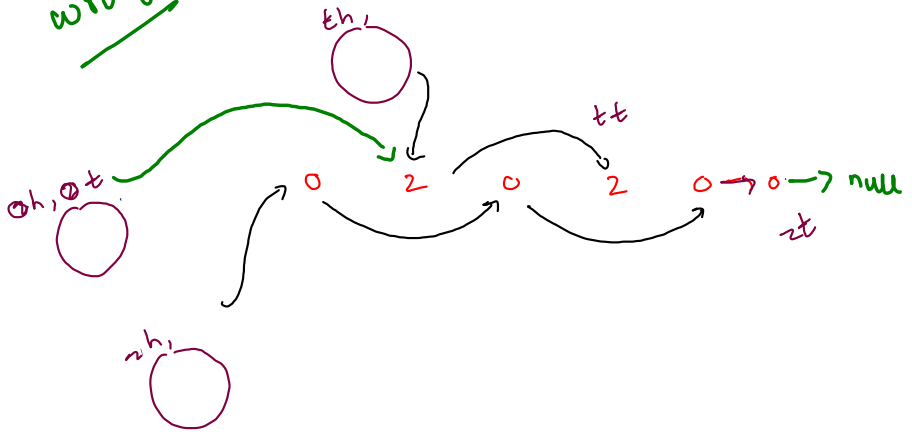
(iii) 2 x ✓




```
while(curr != null) {
    if(curr.val == 0) {
        zt.next = curr;
        zt = zt.next;
    }
    else if(curr.val == 1){
        ot.next = curr;
        ot = ot.next;
    }
    else {
        tt.next = curr;
        tt = tt.next;
    }
    curr = curr.next;
}

zt.next = oh.next;
ot.next = th.next;
tt.next = null;

return zh.next;
```

(ii) IX code X

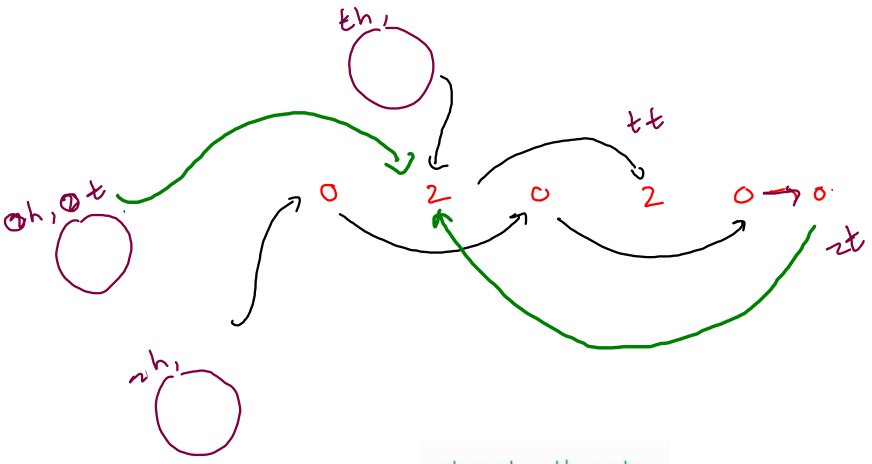
wrong




```
zt.next = oh.next;
ot.next = th.next;
tt.next = null;
```

0 -> 0 -> 0 -> 0 X

correct

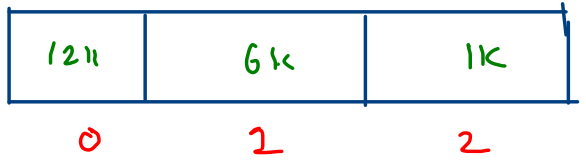


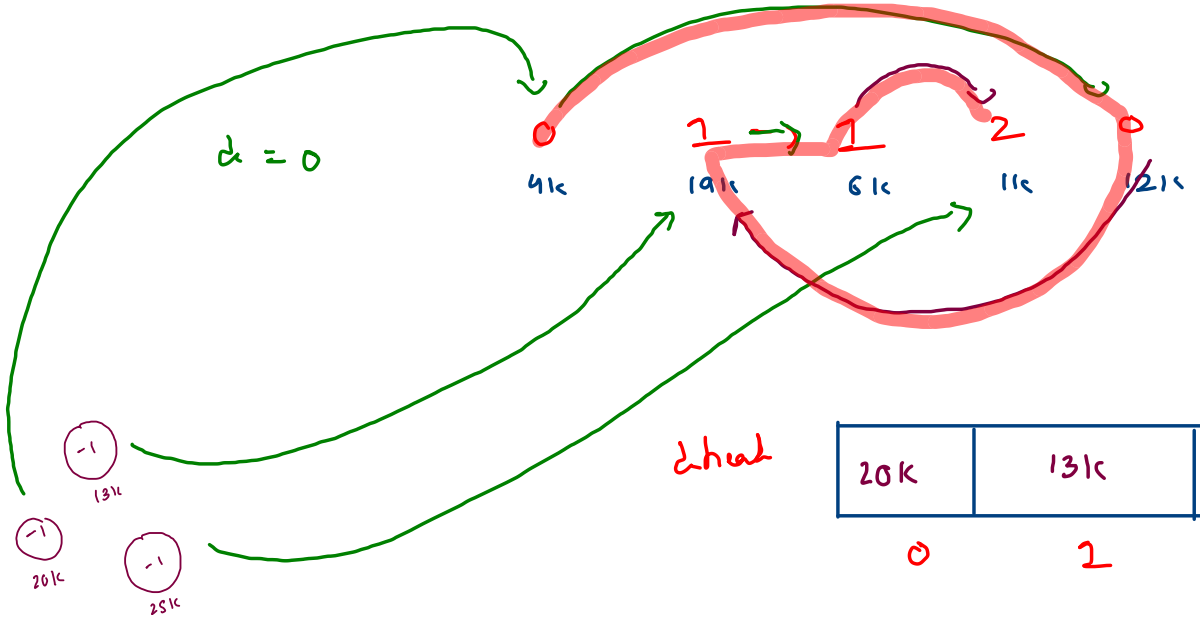

```
ot.next = th.next;
zt.next = oh.next;
tt.next = null;
```

0 -> 0 -> 0 -> 0 -> 2 -> 2

```
ListNode[]dh = new ListNode[k]; //array of dummy heads
ListNode[]dt = new ListNode[k]; //array of dummy tails

for(int i=0; i < k;i++) {
    dh[i] = new ListNode(-1);
    dt[i] = dh[i];
}

ListNode curr = head;

while(curr != null) {
    int d = curr.val;

    dt[d].next = curr;
    dt[d] = dt[d].next;
    curr = curr.next;
}

dt[1].next = dh[2].next;
dt[0].next = dh[1].next;
dt[2].next = null;

return dh[0].next;
```

dtail



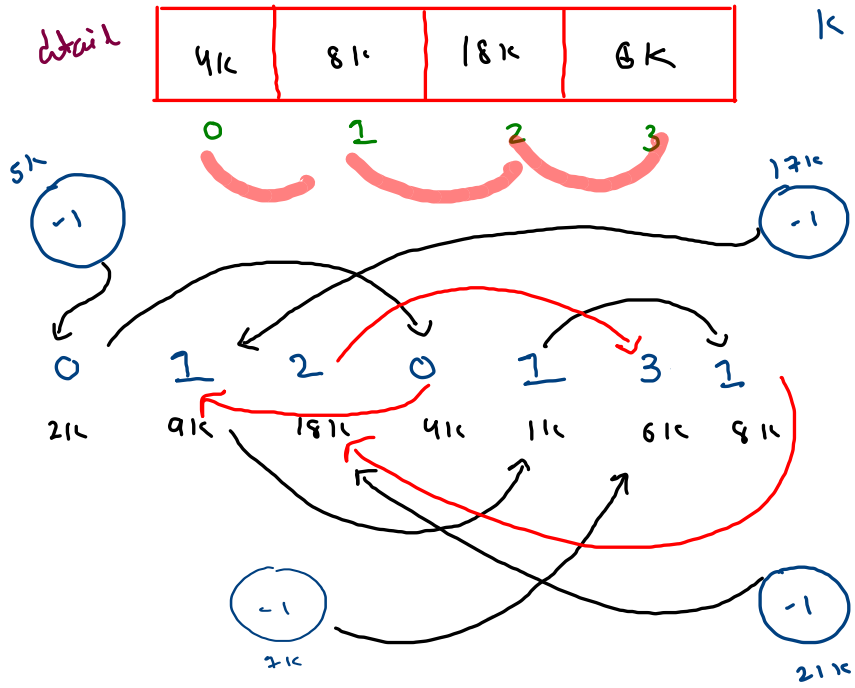


dhead

dt.next = th.next

2t.next = dh.next

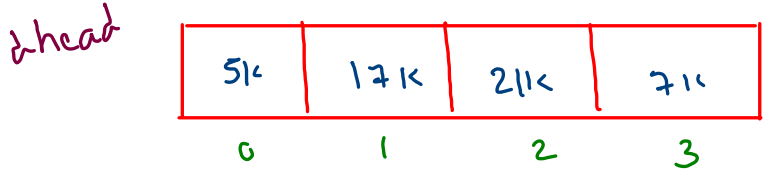
```
ListNode[]dh = new ListNode[k]; //array of dummy heads
ListNode[]dt = new ListNode[k]; //array of dummy tails

for(int i=0; i < k;i++) {
    dh[i] = new ListNode(-1);
    dt[i] = dh[i];
}

ListNode curr = head;

while(curr != null) {
    int d = curr.val;

    dt[d].next = curr;
    dt[d] = dt[d].next;
    curr = curr.next;
}

dt[1].next = dh[2].next;
dt[0].next = dh[1].next;
dt[2].next = null;

return dh[0].next;
```



0, 1, 2, 3

ans -> 5k.next

6k.next = null

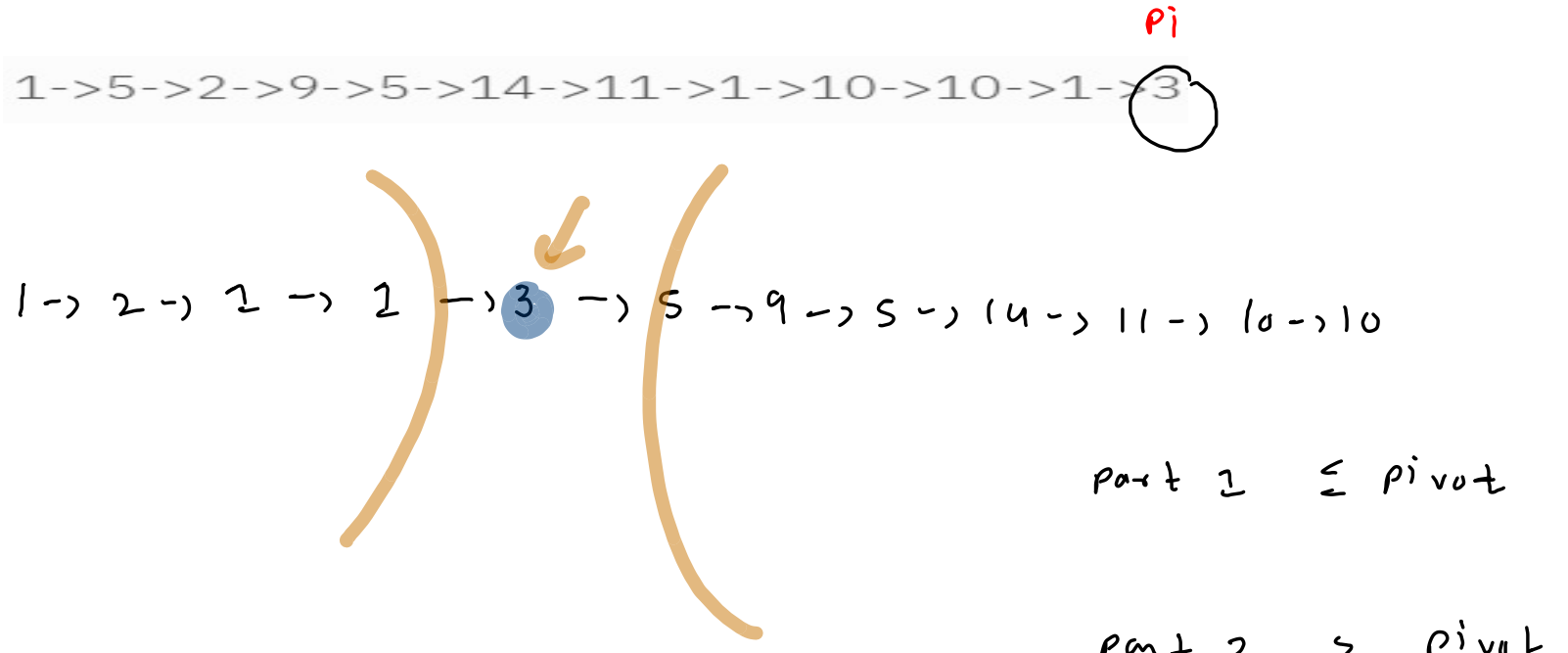


i = 3

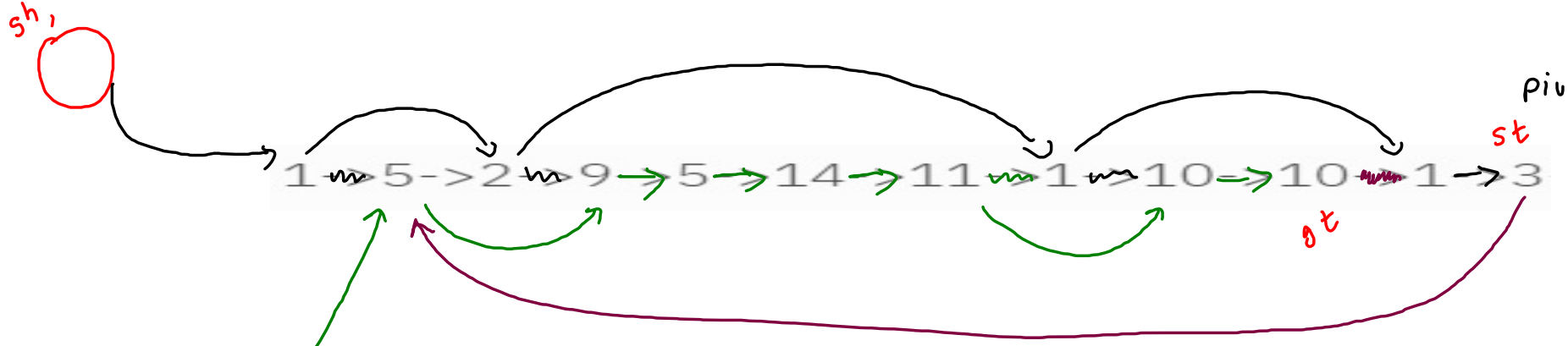
imit ih

imit.next = ih.next

3->5->9->5->14->11->10->10->null



data = 3



st.next = gh.next

gt.next = null

return st.

curr.val <= data {

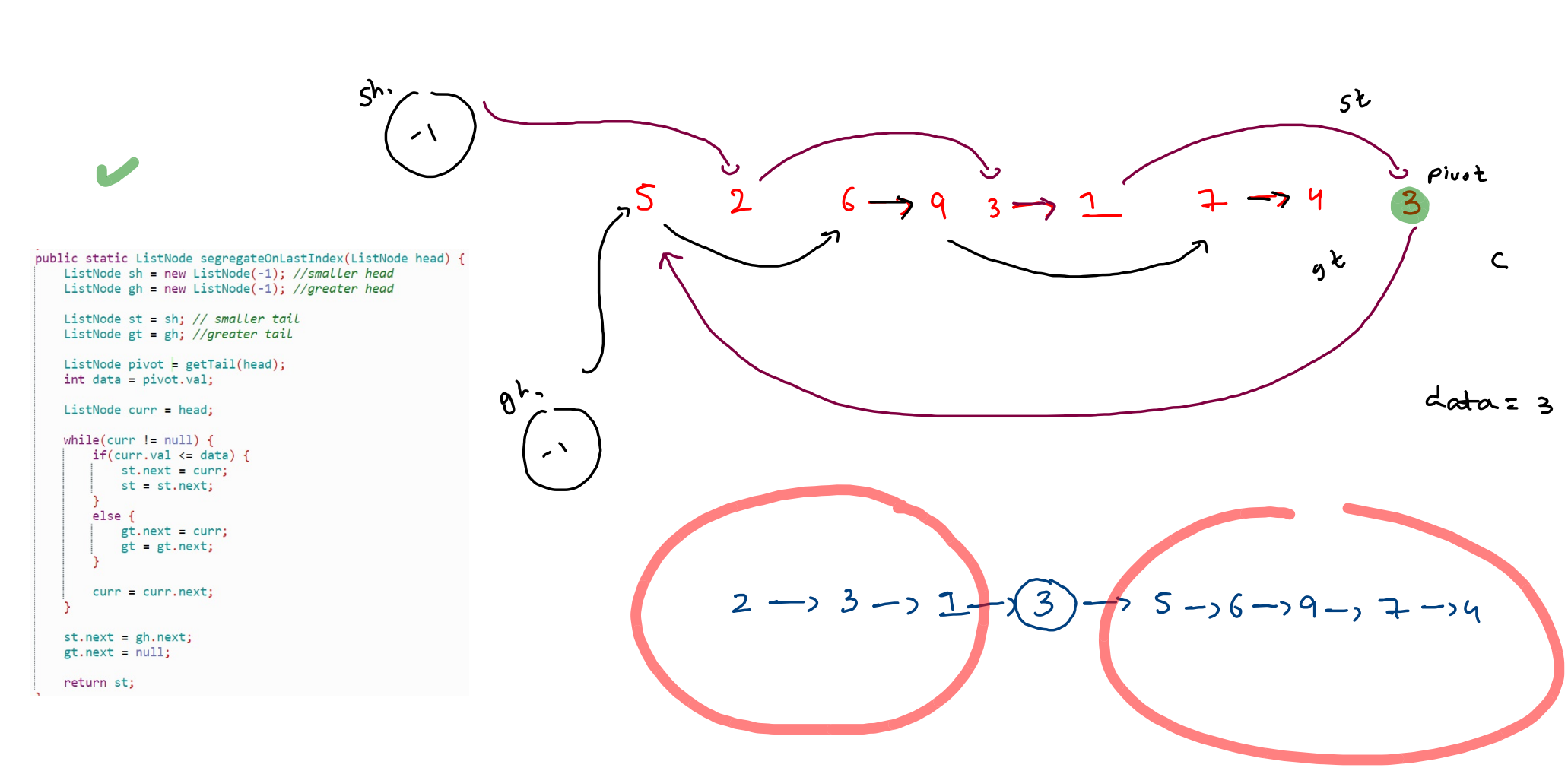
smaller region

}

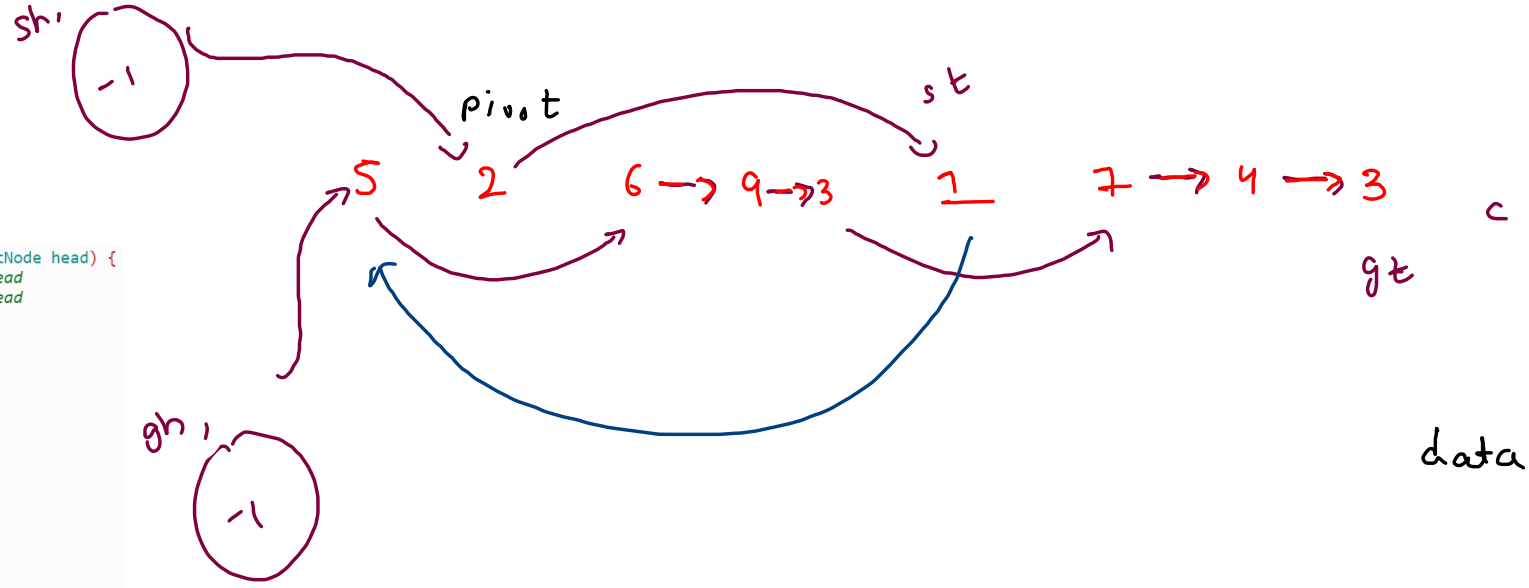
else {

greater region

}



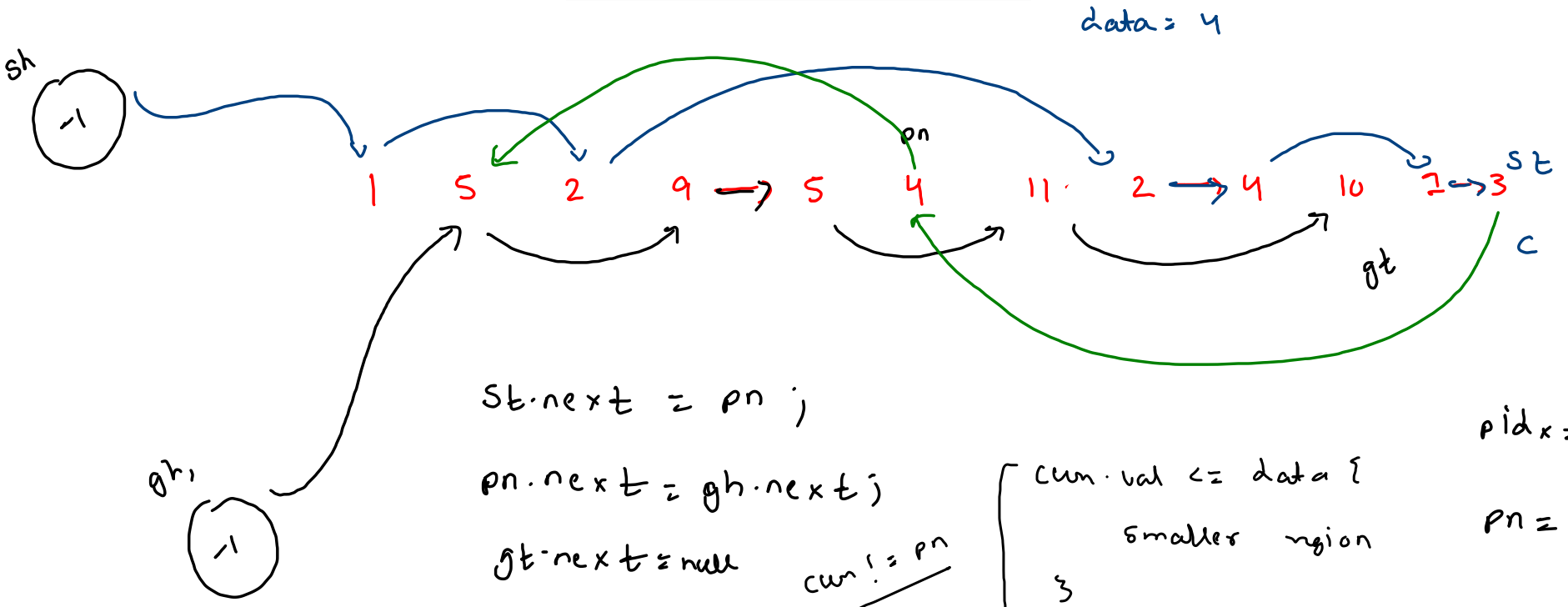

```java
public static ListNode segregateOnLastIndex(ListNode head) {
    ListNode sh = new ListNode(-1); //smaller head
    ListNode gh = new ListNode(-1); //greater head

    ListNode st = sh; // smaller tail
    ListNode gt = gh; //greater tail

    ListNode pivot = getTail(head);
    int data = pivot.val;

    ListNode curr = head;

    while(curr != null) {
        if(curr.val <= data) {
            st.next = curr;
            st = st.next;
        }
        else {
            gt.next = curr;
            gt = gt.next;
        }

        curr = curr.next;
    }

    st.next = gh.next;
    gt.next = null;

    return st;
```

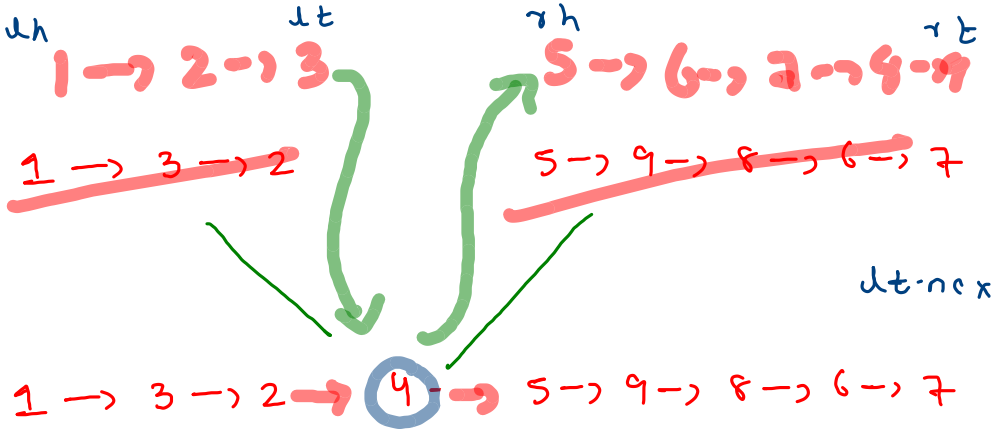



```java
public static ListNode segregateOnLastIndex(ListNode head) {
    ListNode sh = new ListNode(-1); //smaller head
    ListNode gh = new ListNode(-1); //greater head

    ListNode st = sh; // smaller tail
    ListNode gt = gh; //greater tail

    ListNode pivot = getTail(head);
    int data = pivot.val;

    ListNode curr = head;

    while(curr != null) {
        if(curr.val <= data) {
            st.next = curr;
            st = st.next;
        }
        else {
            gt.next = curr;
            gt = gt.next;
        }

        curr = curr.next;
    }

    st.next = gh.next;
    gt.next = null;

    return st;
```

2 -> 1 -> 5 -> 6 -> 9 -> 3 -> 7 -> 4 -> 3

1->5->2->9->5->14->11->1->10->10->1->3-

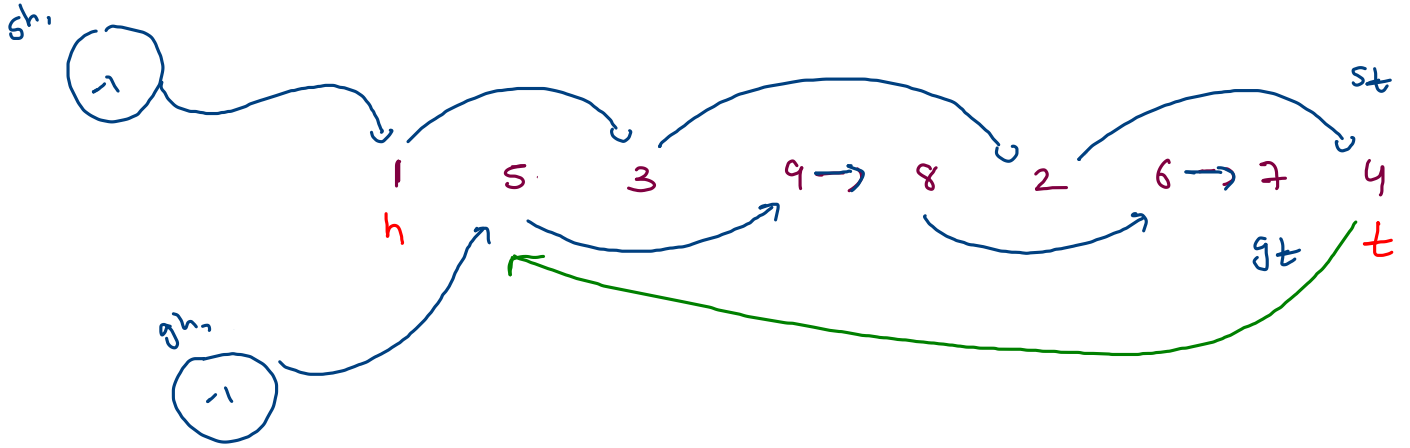




lt.next = pn

pn.next = rh

return {lh, rt}

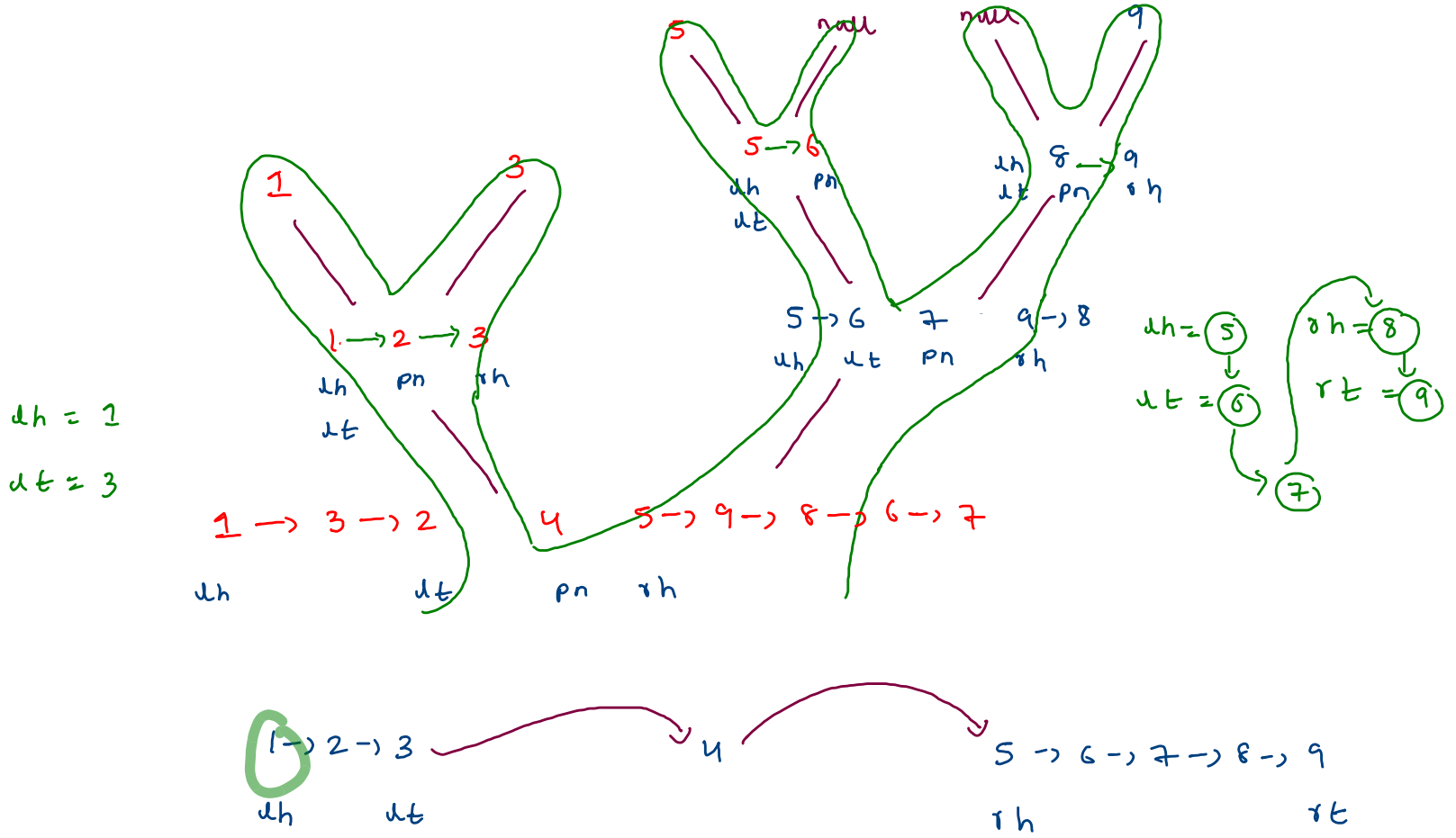


st.next = gh.next

gt.next = null

sh.next

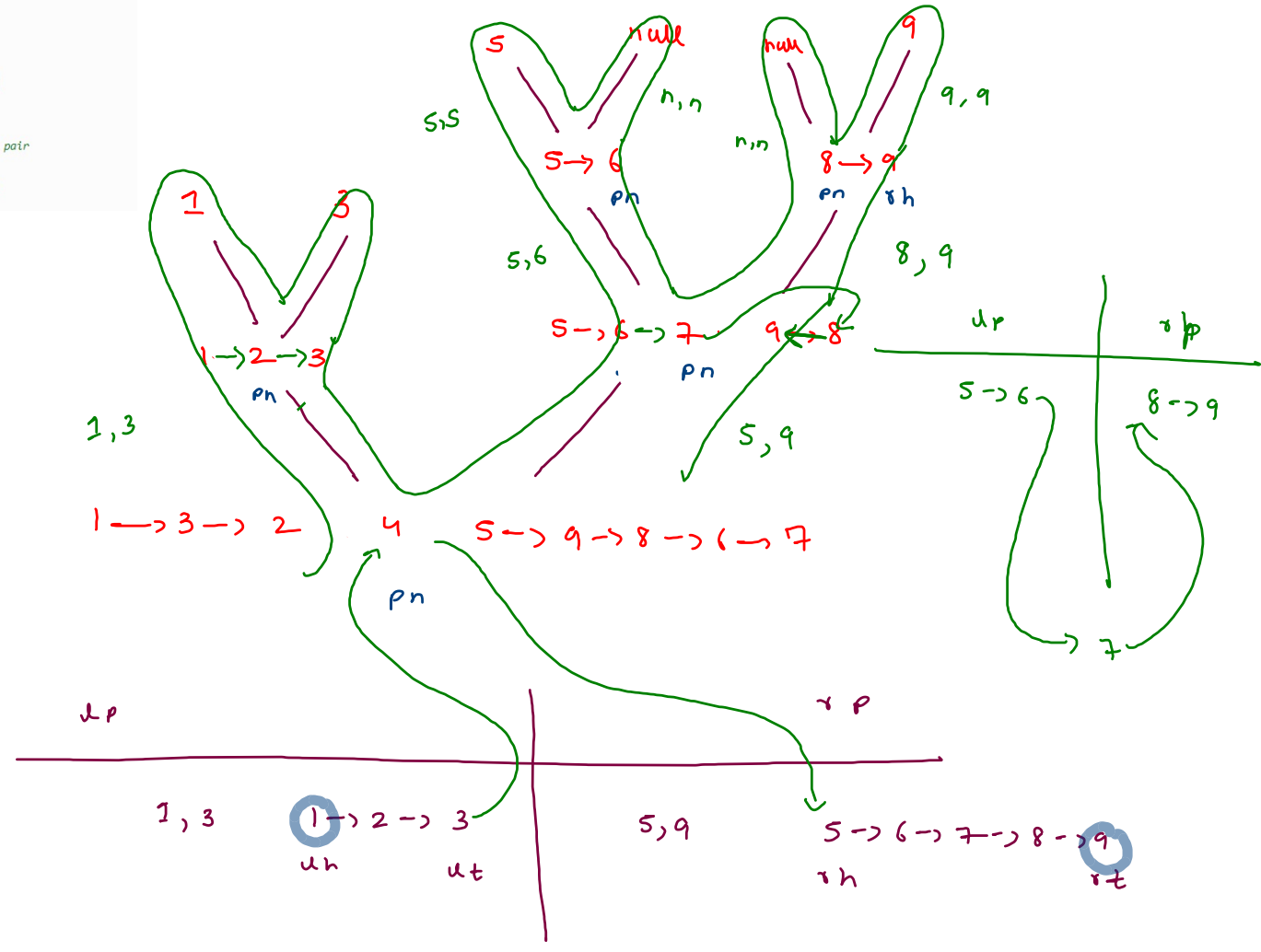
sh.next , st.prev, pn


lh = 1
lt = 3
1 → 3 → 2
4
5 → 9 → 8 → 6 → 7
lh
lt
pn
rh
1 → 2 → 3
5 → 6
7
9 → 8
null
null
5
9
lh = 5
lt = 6
rh = 8
rt = 9
7
1 → 2 → 3
4
5 → 6 → 7 → 8 → 9
[lh, rt] = 1, 9

```java
public static QSPair quickSortH(ListNode head) {
    if(head == null || head.next == null) {
        return new QSPair(head,head);
    }

    ParPair p = partition(head);


    if(p.lt != null) {
        p.lt.next = null;
    }
    else {
        p.lh = null;
    }

    ListNode rh = p.pn.next;

    p.pn.next = null;

    QSPair lp = quickSortH(p.lh);
    QSPair rp = quickSortH(rh);

    QSPair ap = merge(lp,p.pn,rp);
    return ap;
}
```

```java
public static QSPair merge(QSPair lp,ListNode pn,QSPair rp) {

    if(lp.head == null && rp.head == null) {
        return new QSPair(null,null);
    }
    else if(lp.head == null) {
        //use only pivot and right pair
        pn.next = rp.head;
        return new QSPair(pn,rp.tail);
    }
    else if(rp.head == null) {
        //use only left pair and pivot node
        lp.tail.next = pn;
        return new QSPair(lp.head,pn);
    }
    else {
        //use left pair , pivot node, right pair
        lp.tail.next = pn;
        pn.next = rp.head;
        return new QSPair(lp.head,rp.tail);
    }
```

```java
public static ParPair partition(ListNode head) {

    ListNode tail = getTail(head);
    int data = tail.val;

    ListNode sh = new ListNode(-1);
    ListNode gh = new ListNode(-1);


    ListNode st = sh;
    ListNode gt = gh;
    ListNode stp = null; //smaller tail prev

    ListNode curr = head;

    while(curr != null) {
        if(curr.val <= data) {
            st.next = curr;
            stp = st;
            st = st.next;
        }
        else {
            gt.next = curr;
            gt = gt.next;
        }
        curr = curr.next;
    }

    st.next = gh.next;
    gt.next = null;

    ParPair p = new ParPair(sh.next,stp,st);
    return p;
}
```



quicksort

nlogn best case

n^2 worst case