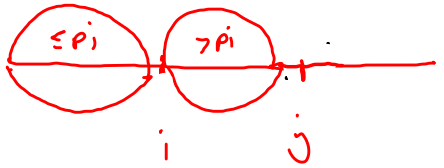
quick sort :

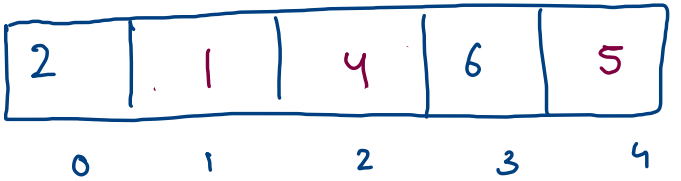


0 to i-1 → <= pivot

i to j-1 → > pivot

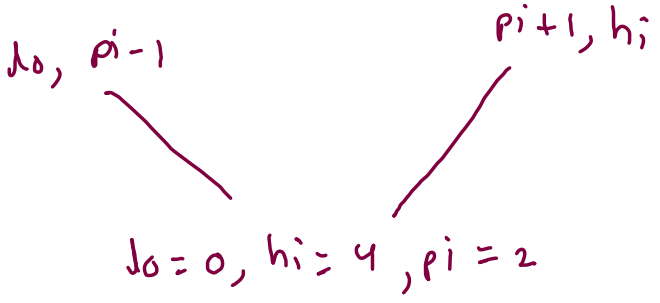
j to end → unknown

```
if (arr[j] > pivot) {
    j++;
}
else {
    swap(i, j);
    i++, j++;
}
```



pivot = 4

2 5 1 6 4



Segregate Node Of Linkedlist Over Last Index.

pivot -> last node

1. Given a singly linklist, Segregate Node of LinkedList over lastindex and return pivot node of linkedlist.
2. pivot is always be last index of linkedlist.
3. After segregation pivot Element should have to be present at correct position as in sorted linkedlist.

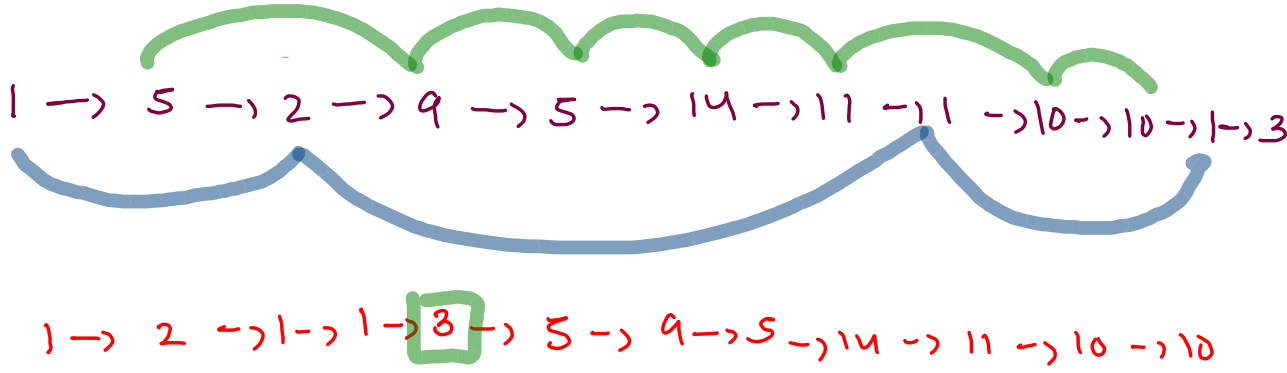
pivot = 3

Input Format

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

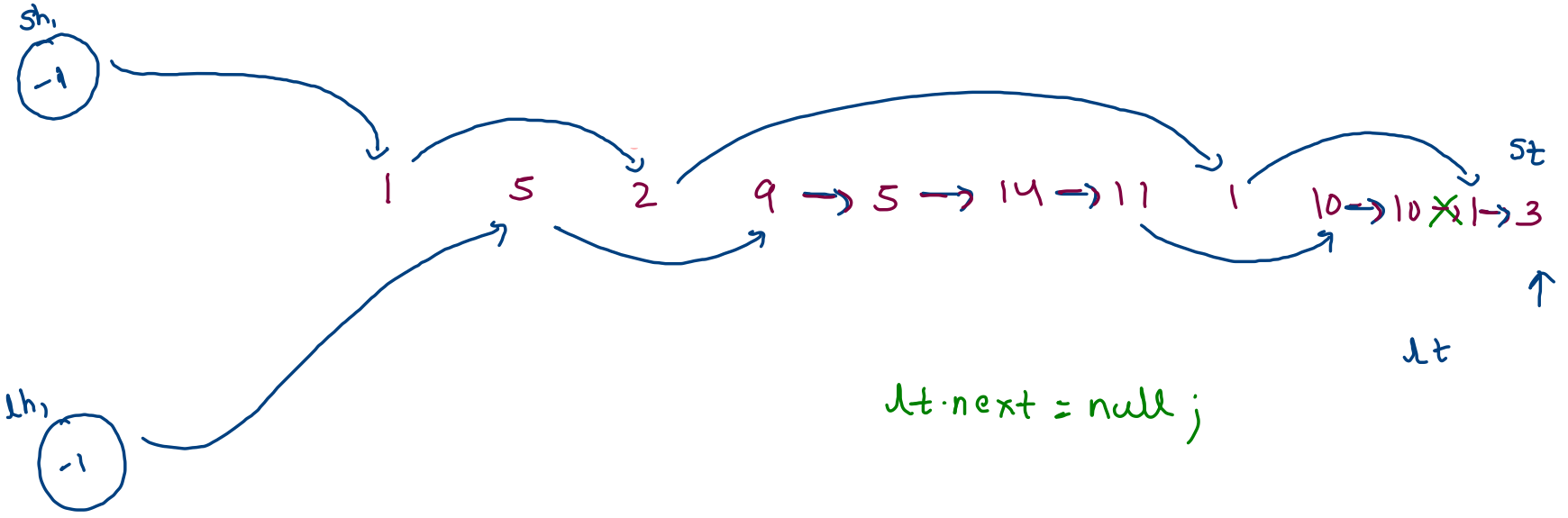
Output Format

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



Smaller : <= pivot

Larger : > pivot



lt.next = null;

st.next = lh.next

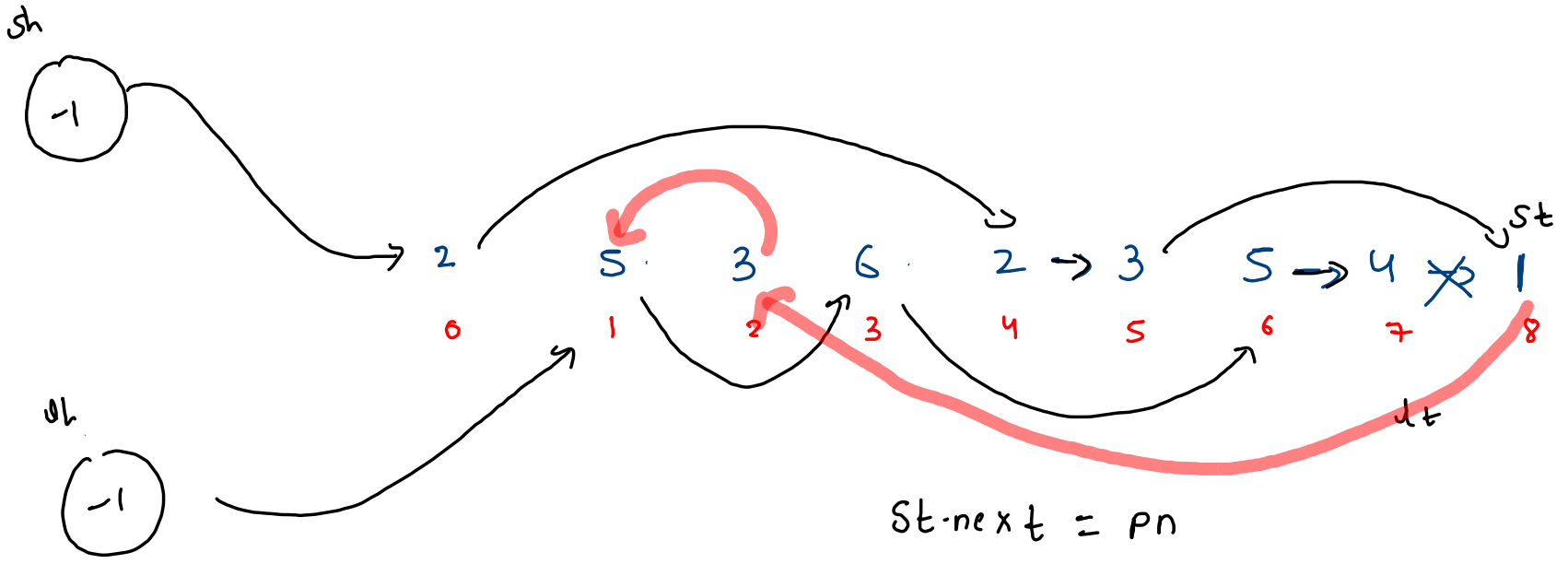
pivot = 3

return st;

Segregate Node Of Linkedlist Over Pivot Index

pi = 2

pivot = 3 (4k)



<= pi <

st.next = pn

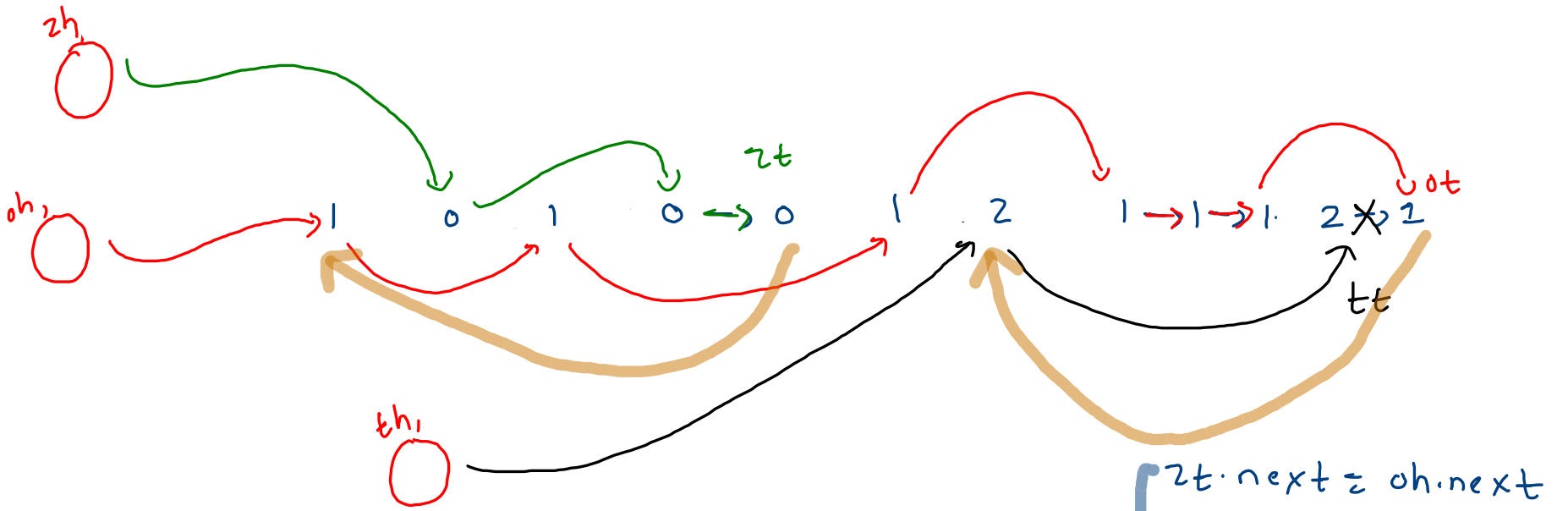
pn.next = dh.next

return sh.next;

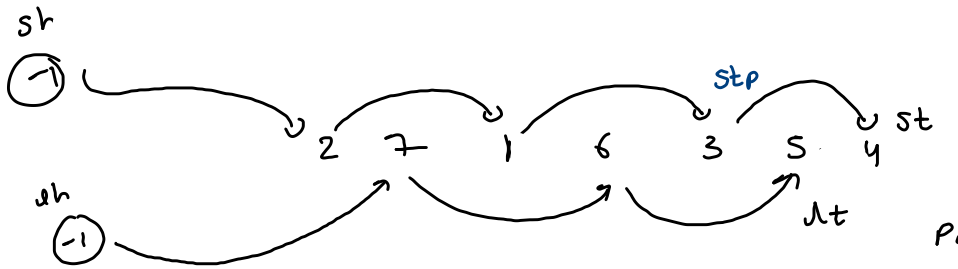
dt.next = null

Segregate 012 Node Of Linkedlist Over Swapping Nodes

1->0->1->0->0->1->2->1->1->1->2->1->1->null



Quicksort In Linkedlist



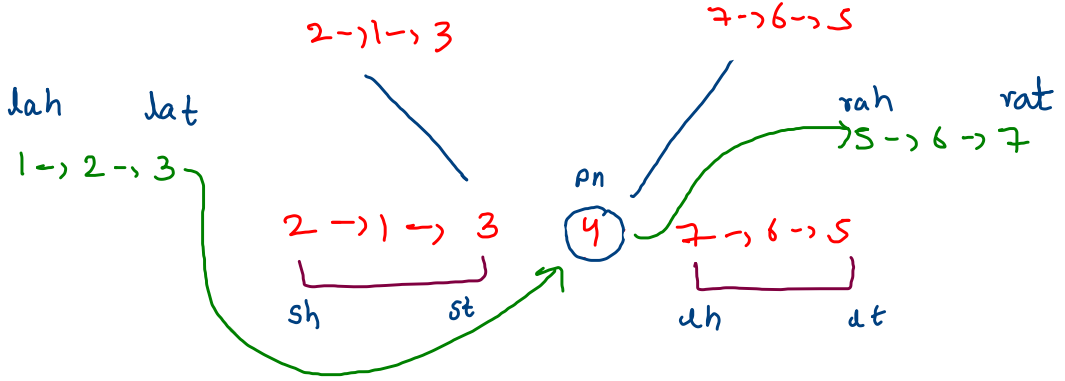
2 -> 7 -> 1 -> 6 -> 3 -> 5 -> 4

pd = 4

[sh.next to stp

[pn : st

[dh.next to dt



lat.next = pn;
pn.next = rah;
{ lah, rat }

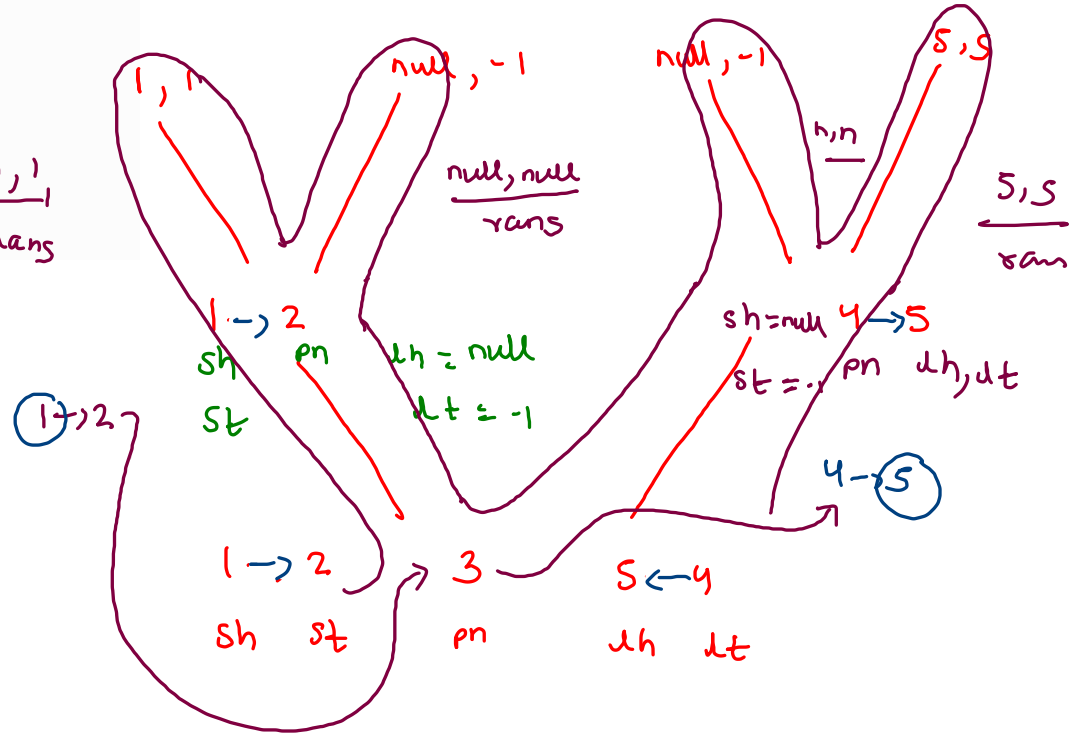
```java
public static QSPair quickSort(ListNode head,ListNode tail) {
    if(head.next == null || head == null) {
        return new QSPair(head,tail);
    }

    ParPair p = partition(head,tail.val);

    QSPair lans = quickSort(p.sh,p.st);
    QSPair rans = quickSort(p.lh,p.lt);

    QSPair ans = merge(lans,p.pn,rans);

    return ans;

}
```

```java
public static ParPair partition(ListNode head,int pivot) {

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

    ListNode lh = new ListNode(-1);
    ListNode lt = lh;

    ListNode temp = head;

    while(temp.next != null) {
        if(temp.val <= pivot) {
            st.next = temp;
            st = st.next;
        }
        else {
            lt.next = temp;
            lt = lt.next;
        }
        temp = temp.next;
    }

    ListNode pn = temp;

    st.next = null;
    lt.next = null;
    pn.next = null;

    return new ParPair(sh.next,st,pn,lh.next,lt);

}
```

```java
public static QSPair merge(QSPair lans,ListNode pn,QSPair rans) {

    if(lans.head != null && rans.head != null) {
        //both lans and rans are there
        lans.tail.next = pn;
        pn.next = rans.head;

        return new QSPair(lans.head,rans.tail);
    }
    else if(lans.head != null) {
        //only left ans is there
        lans.tail.next = pn;

        return new QSPair(lans.head,pn);
    }
    else {
        //only right ans is there
        pn.next = rans.head;

        return new QSPair(pn,rans.tail);

    }
}
```

5 → 1 → 2 → 4 → 3

head tail

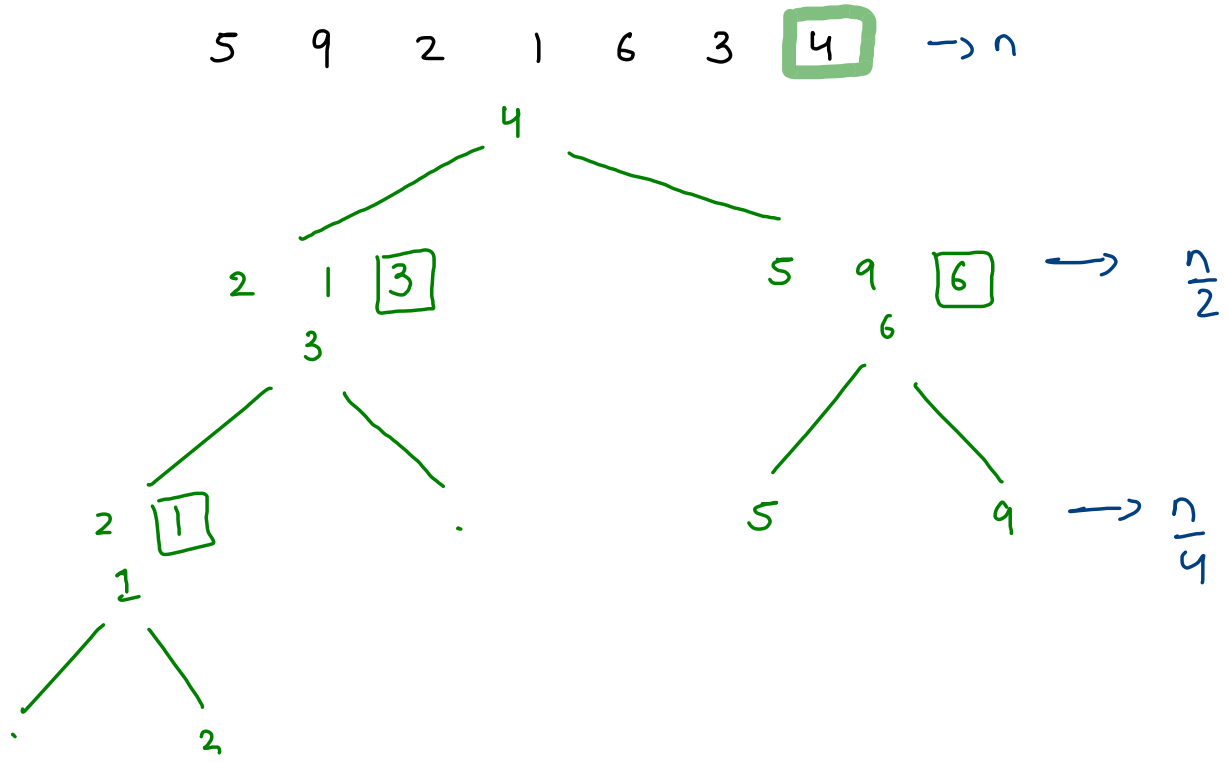


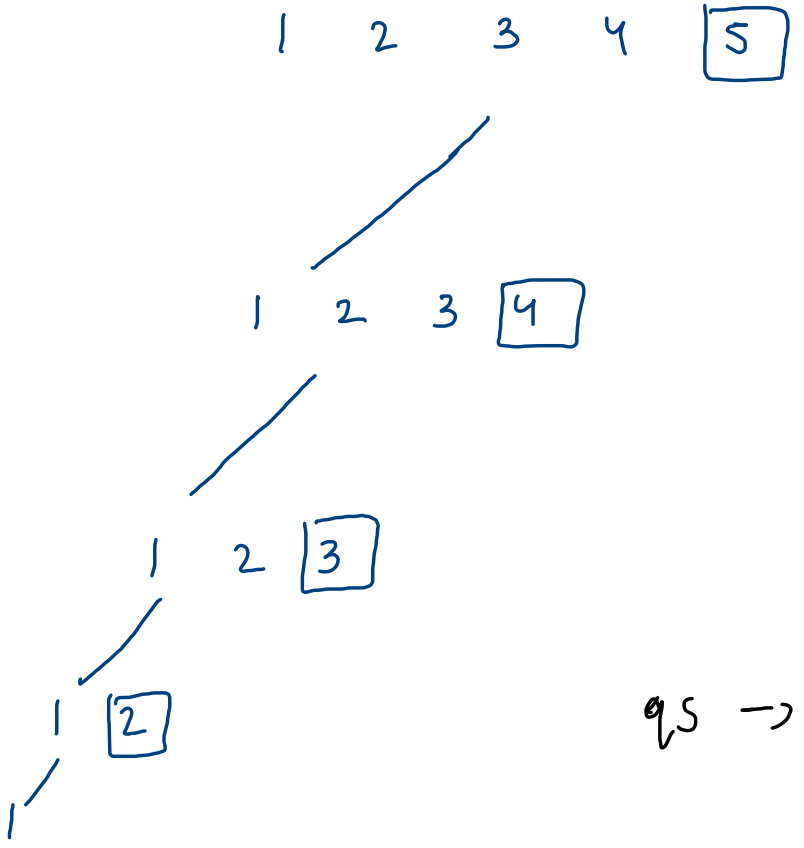
```java
public static QSPair quickSort(ListNode head,ListNode tail) {
    if(head == null || head.next == null) {
        return new QSPair(head,head);
    }

    ParPair p = partition(head,tail.val);

    QSPair lans = quickSort(p.sh,p.st);
    QSPair rans = quickSort(p.lh,p.lt);

    QSPair ans = merge(lans,p.pn,rans);

    return ans;
}
```

$$T(n) = n + 2T\left(\frac{n}{2}\right)$$

$n \log n$

(best case)





$T(n) = n + T(n-1)$

↓

$O(n^2)$

qs → nlogn to n^2