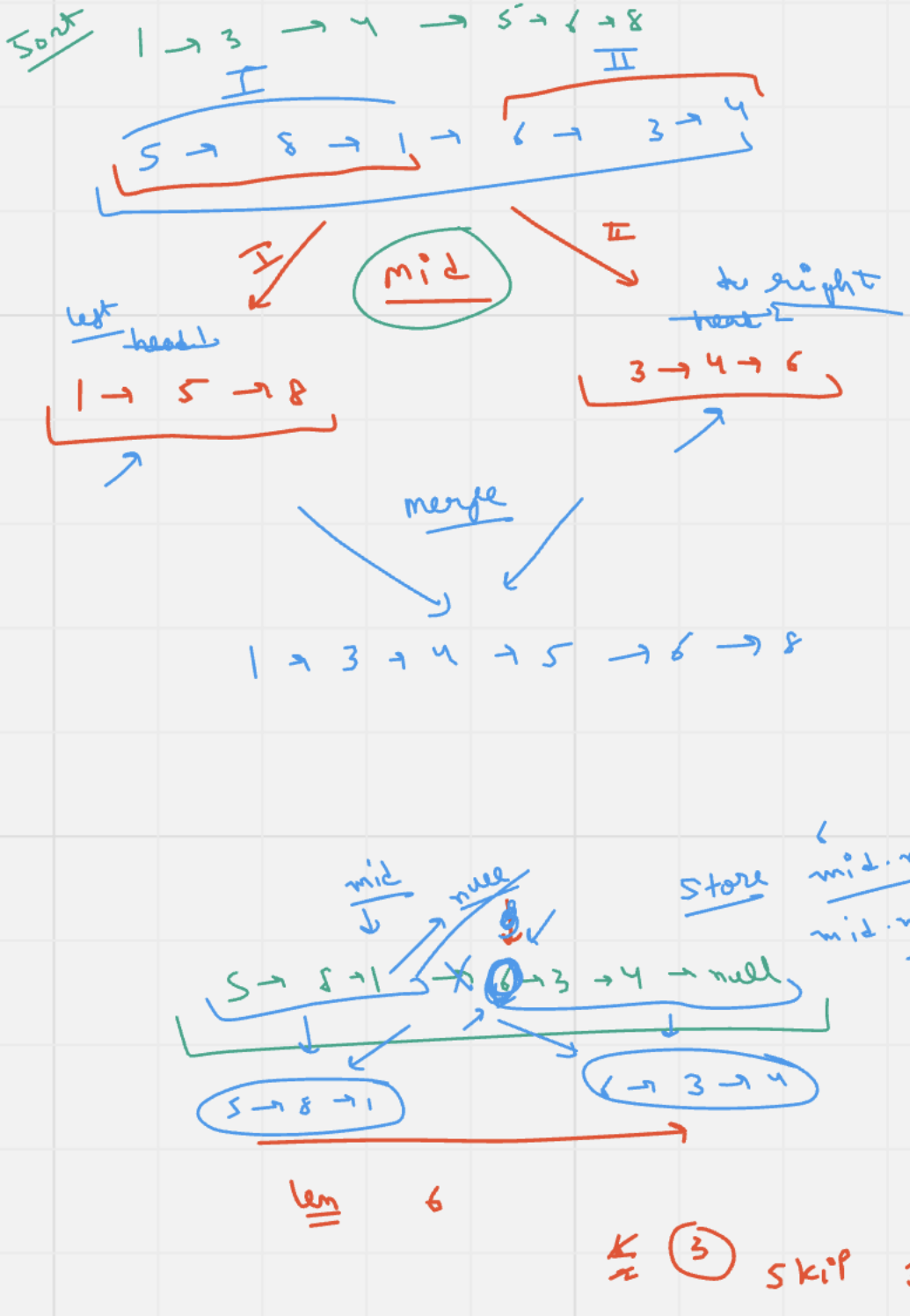
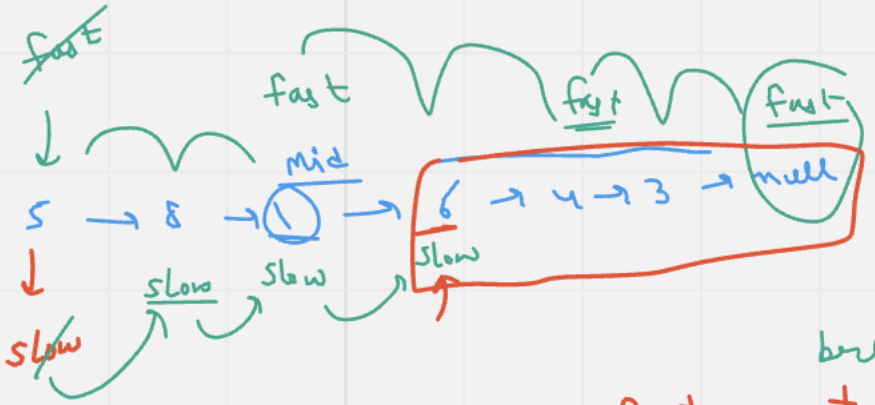

Sort
1 → 3 → 4 → 5 → 6 → 8
I
II
5 → 8 → 1
6 → 3 → 4
I
mid
II
left head
1 → 5 → 8
to right
3 → 4 → 6
merge
1 → 3 → 4 → 5 → 6 → 8
mid
null
Store
mid.next
mid.next
null
5 → 8 → 1 → 6 → 3 → 4 → null
5 → 8 → 1
6 → 3 → 4
len 6
k = 3
skip 3 nodes


start
x
2x
①
②
①
②
End



break

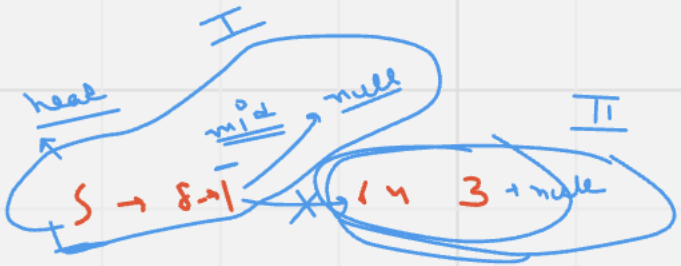
fast = fast.next.next

slow = slow.next

fast == null || fast.next == null

fast.next == null

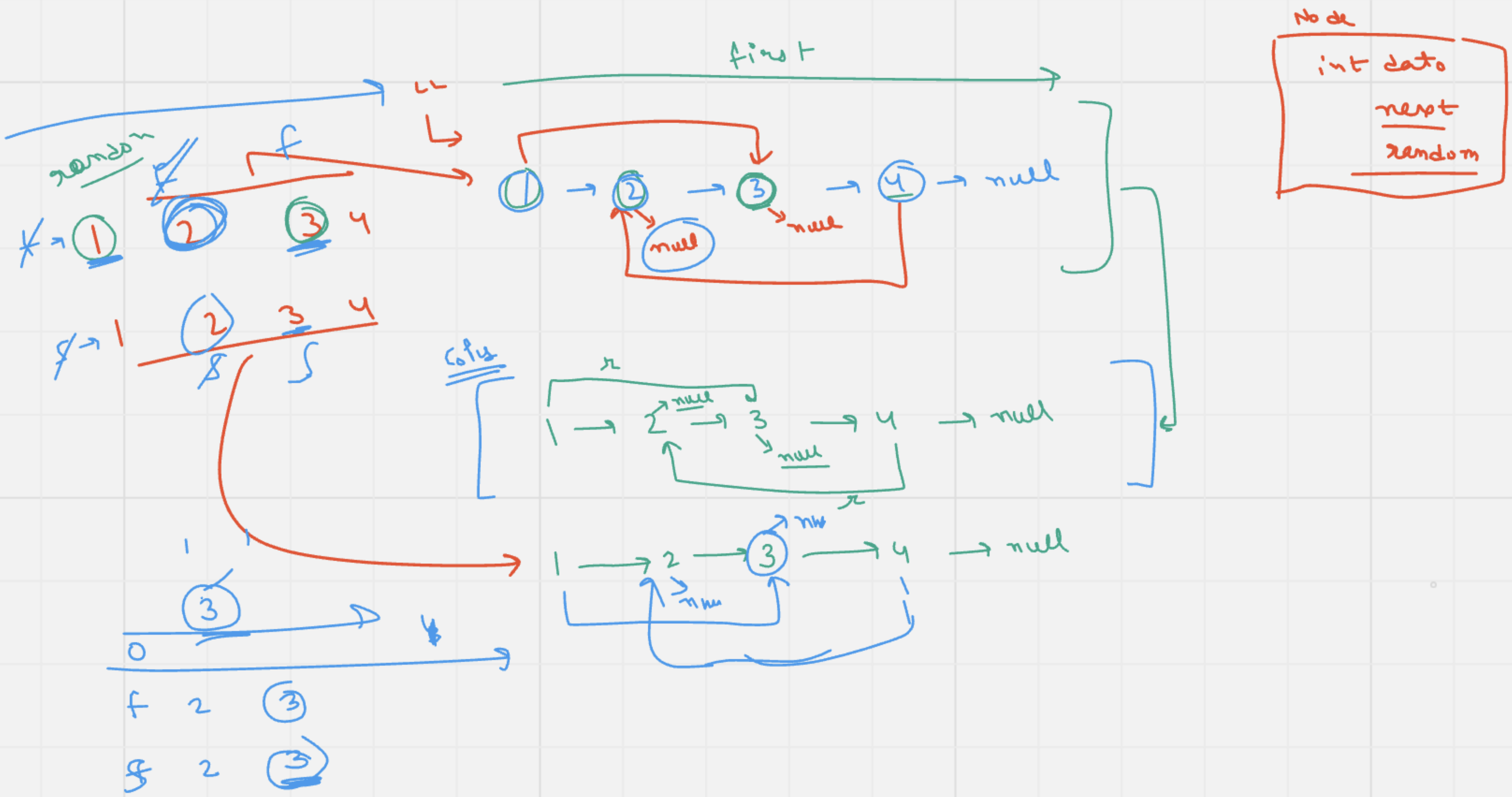
fast.next.next

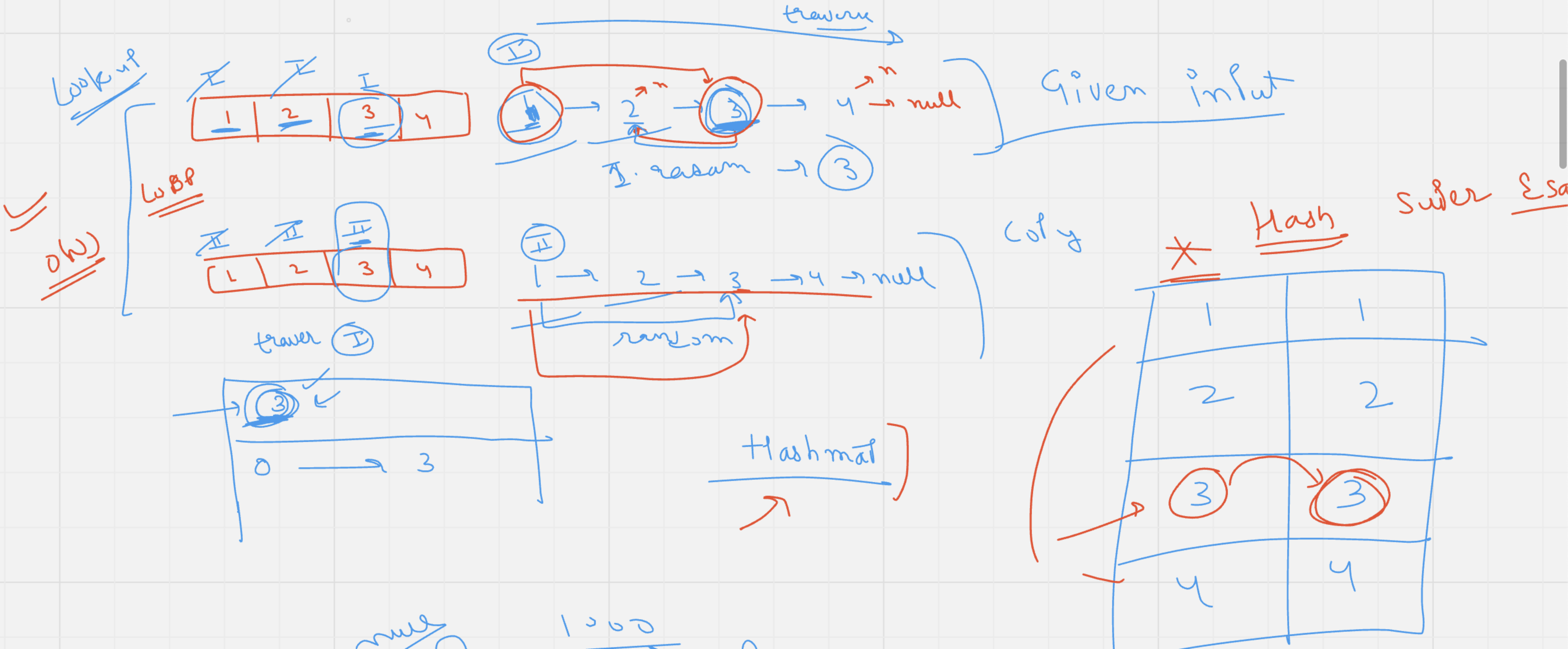


mergeSort (head)

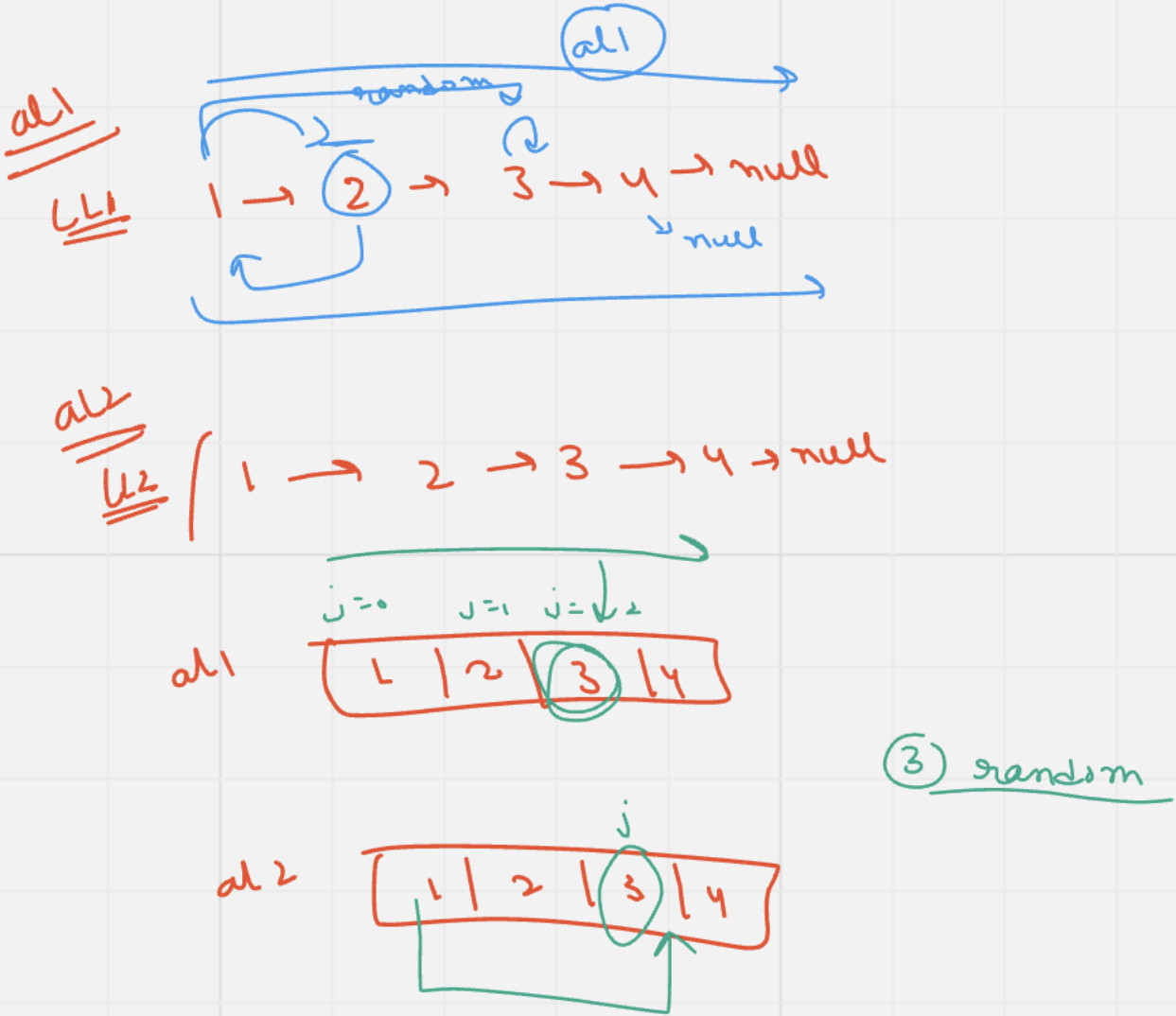
mid.next = null

mergeSort (mid → next)

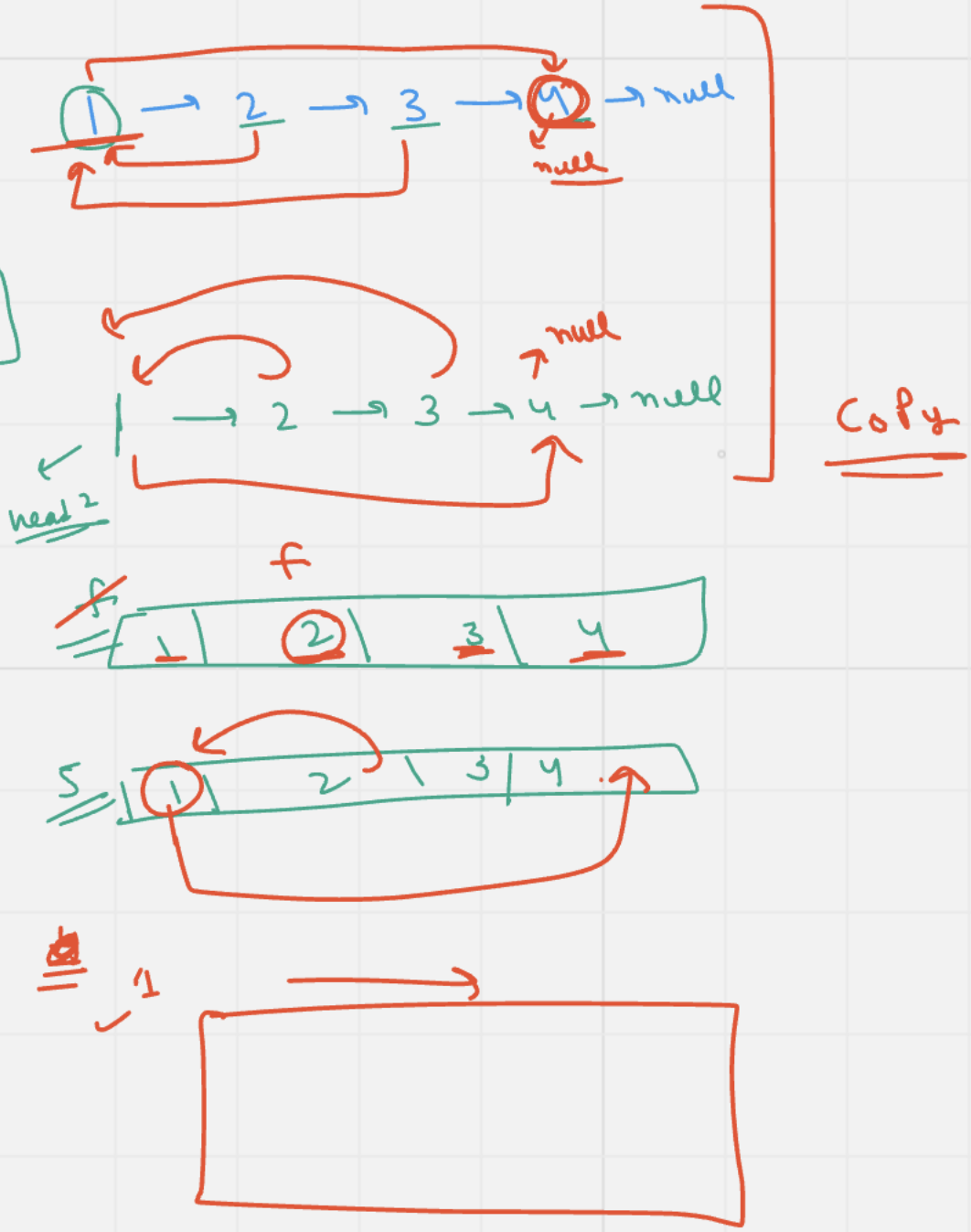

Node
int data
next
random
first
random
copy
null



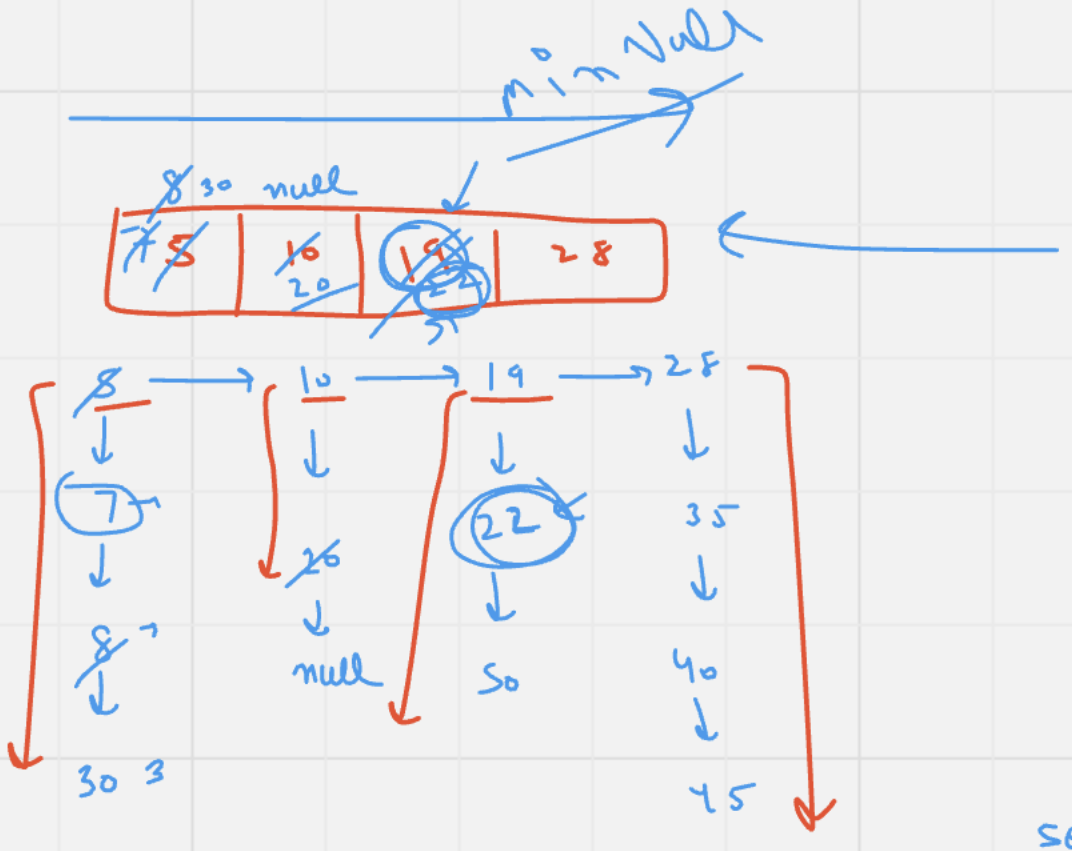
lookup
O(N)
traverse
Given input
null
random
copy
Hash
Hashmap


al1
LL1
al1
random
1 → 2 → 3 → 4 → null
null
al2
LL2
1 → 2 → 3 → 4 → null
j=0
j=1
j=2
al1
1 | 2 | 3 | 4
3) random
j
al2
1 | 2 | 3 | 4

```
Node copyList(Node head) {

    Node head2 = null;
    Node head1 = head;
    Node temp2 = null;

    ArrayList<Node> al1 = new ArrayList<>();
    ArrayList<Node> al2 = new ArrayList<>();

    while(head1 != null) {

        Node newNode = new Node(head1.data);
        if(head2 == null)
        {
            head2 = newNode;
            temp2 = head2;

        }else
        {
            temp2.next = newNode;
            temp2 = temp2.next;
        }

        al1.add(head1);
        al2.add(temp2);
        head1 = head1.next;
    }

    for(int i = 0; i < al1.size(); i++) {

        Node random = al1.get(i).random;
        if(random == null) continue;

        for(int j = 0; j < al1.size(); j++) {

            if(al1.get(j) == random) {
                al2.get(i).random = al2.get(j);
                break;
            }
        }

    }

    return head2;
```

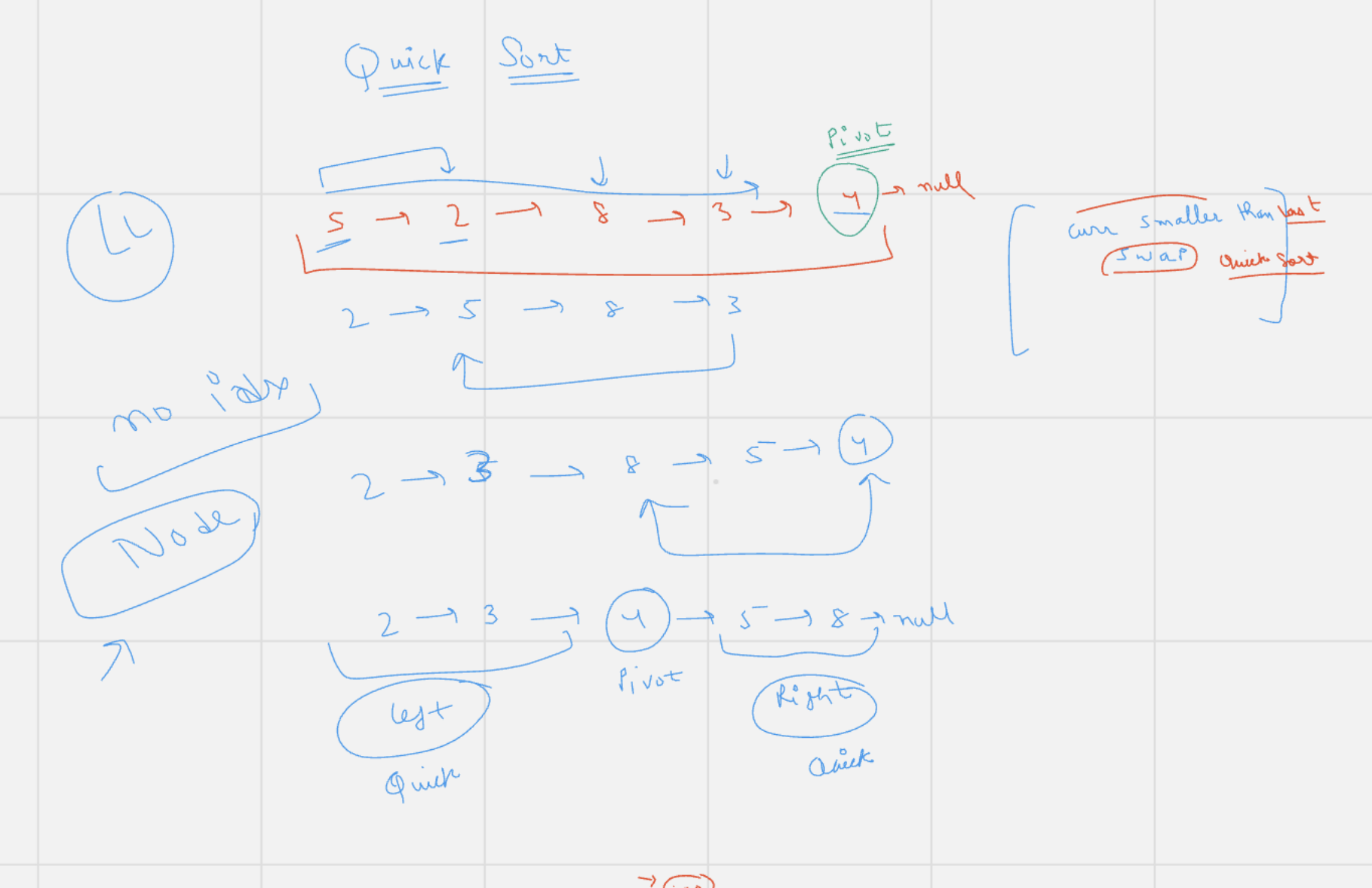


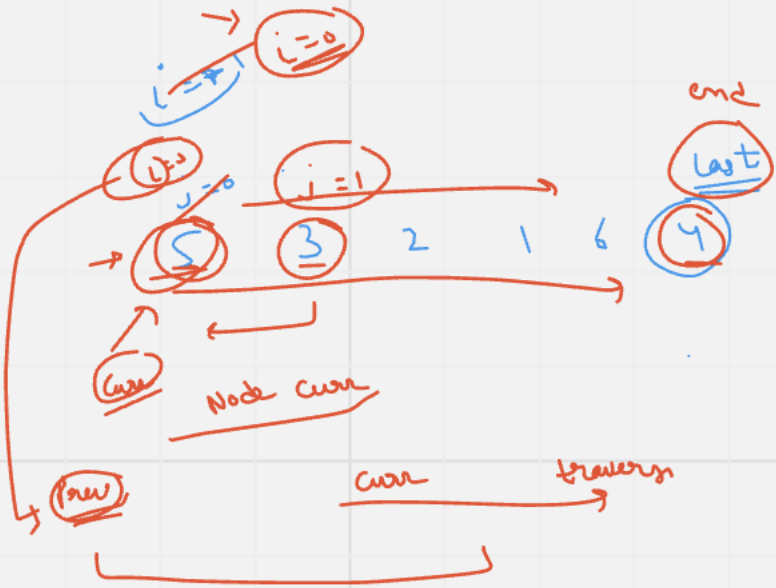
4
4
5 7 8 30
2
10 20
3
19 22 50
4
28 35 40 45

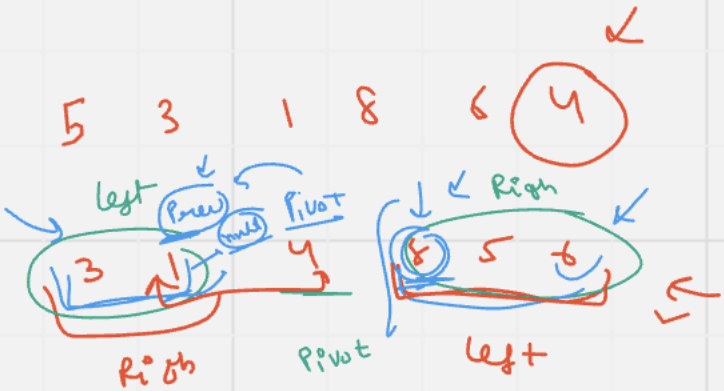


Sorted

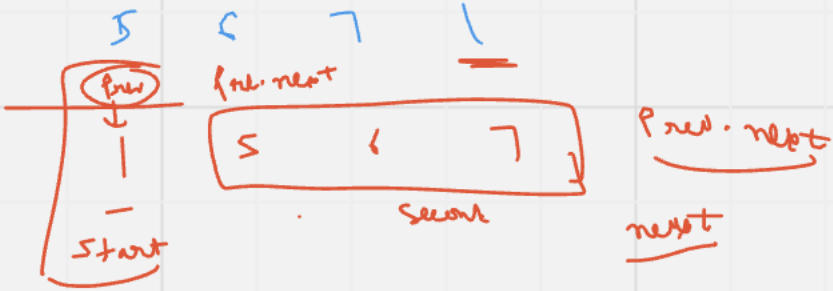
5 → 7 → 8 → 10 → 19 → 20


Quick Sort
Pivot
5 → 2 → 8 → 3 → 4 → null
LL
curr smaller than last
swap
Quick sort
2 → 5 → 8 → 3
no idea
Node
2 → 3 → 8 → 5 → 4
2 → 3 → 4 → 5 → 8 → null
left
Pivot
Right
Quick
Quick





= prev.next.next & second call



```
static Node partition(Node start, Node end) {

    int pivotVal = end.data;
    Node prev = start; // i
    Node curr = start; // j
    Node prevSwap = start;

    while(curr !=  end) {

        if(curr.data < end.data) {

            prevSwap = prev;
            int temp = prev.data;
            prev.data = curr.data;
            curr.data = temp;

            prev = prev.next;
        }

        curr = curr.next;

    }

    int temp = prev.data;
    prev.data = end.data;
    end.data = temp;

    return prevSwap;

}
public static void quickSort(Node start, Node last) {

    // base case
    if(start == null || start == last) {
        return ;
    }

    Node prev = partition(start, last);
    quickSort(start, prev);

    if(prev != null && prev == start) {
        quickSort(prev.next, last);
    }else if(prev != null && prev != start) {
        quickSort(prev.next.next, last);
    }

}
```