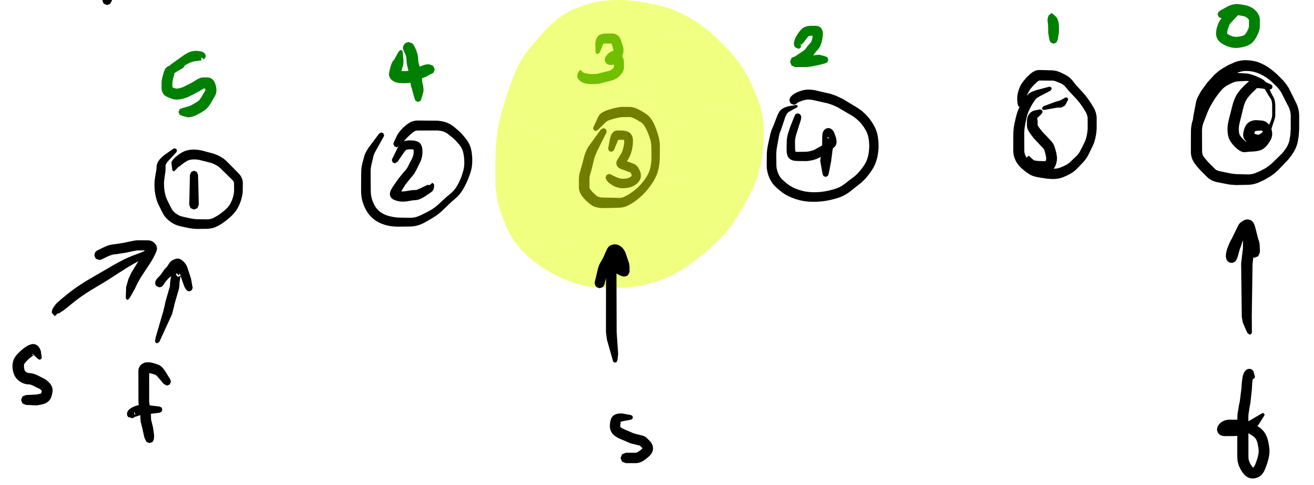
14th Jan, 2022

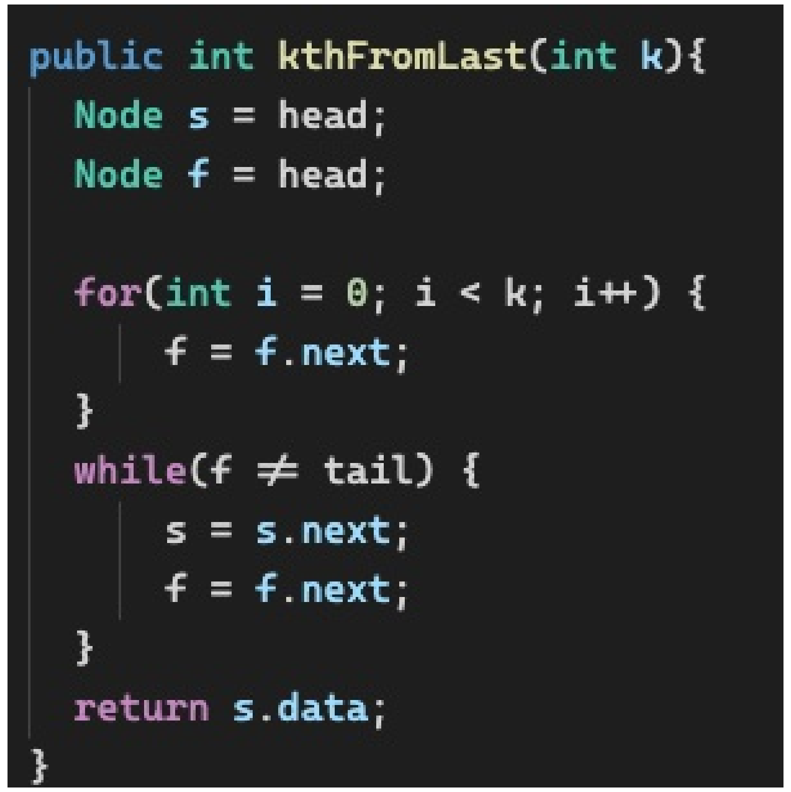
Lecture - 37

Strategy :-
* gap being maintained
* then processed. ✓

Problem: Kth node from the end of "Linked List".



(lets say k=3)



```
public int kthFromLast(int k){
  Node s = head;
  Node f = head;

  for(int i = 0; i < k; i++) {
      f = f.next;
  }
  while(f ≠ tail) {
      s = s.next;
      f = f.next;
  }
  return s.data;
}
```

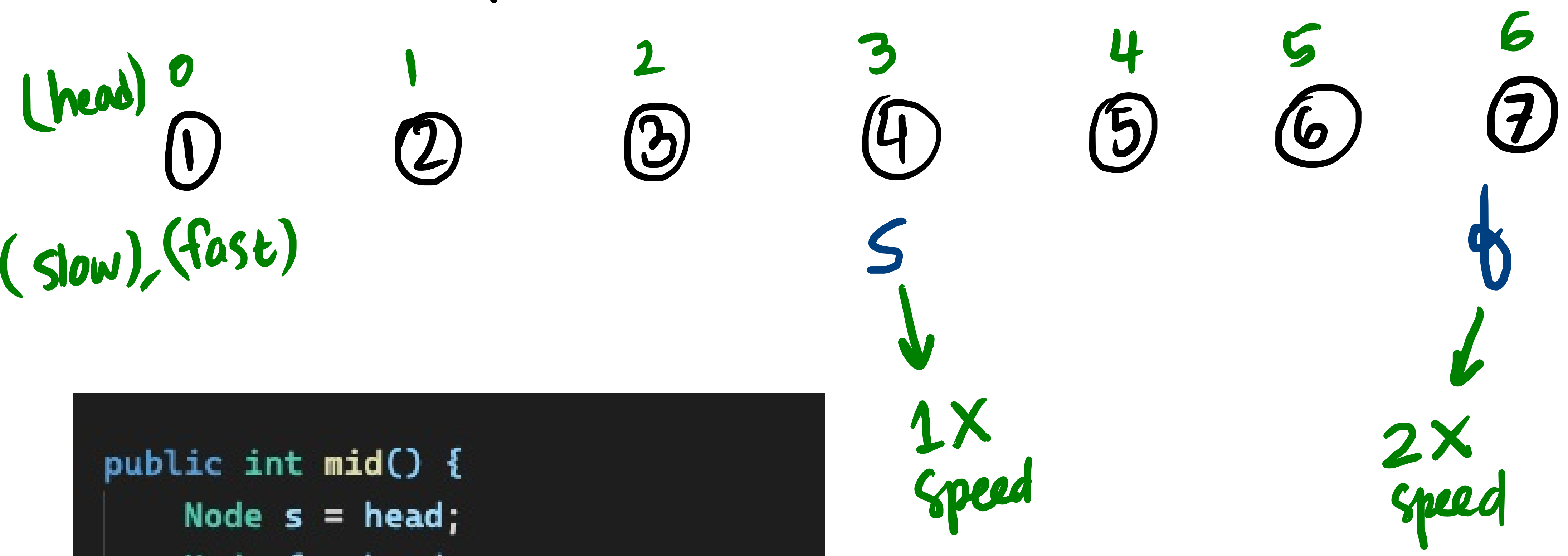
Start (slow, fast) from head

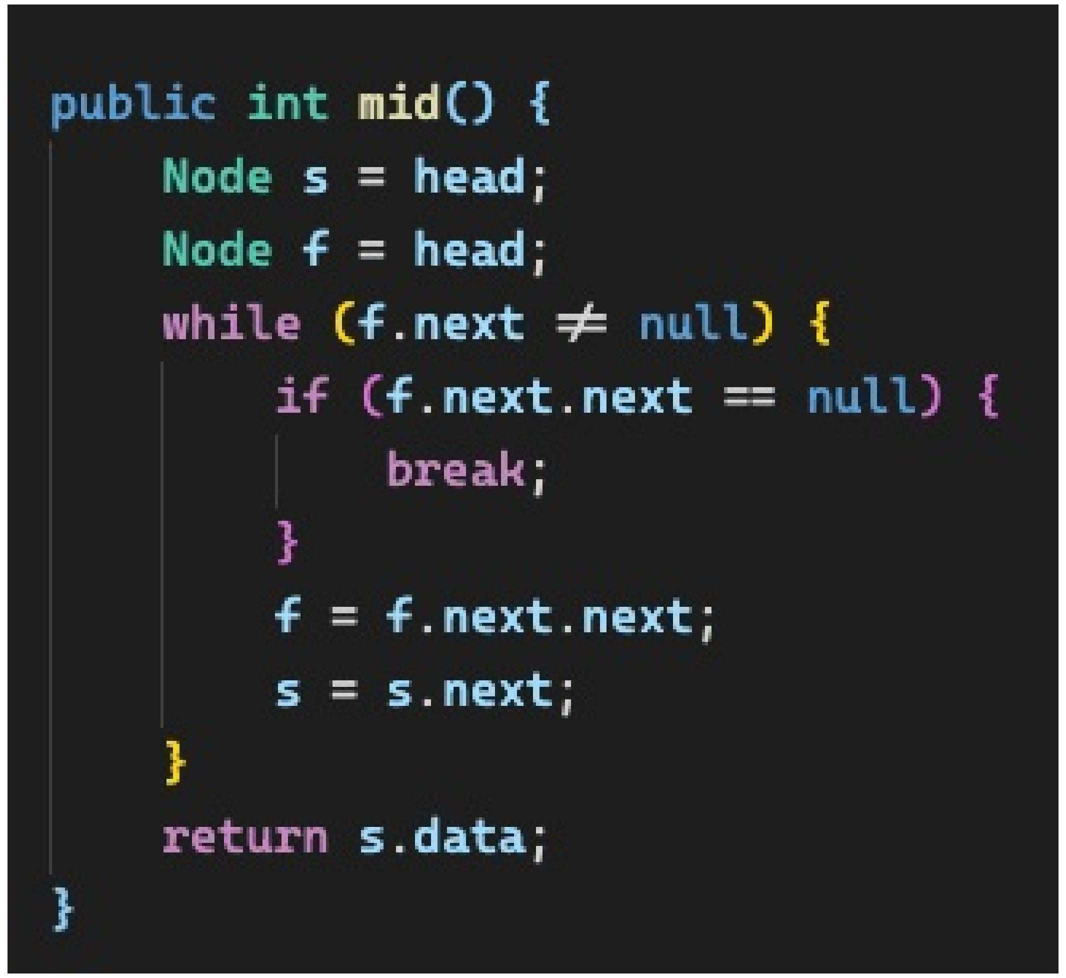
run fast k-times!

} inc. by 1 till fast.next == null !

// return s.data ← Ans!

Problem 2: Mid of a Linked List





```java
public int mid() {
    Node s = head;
    Node f = head;
    while (f.next ≠ null) {
        if (f.next.next == null) {
            break;
        }
        f = f.next.next;
        s = s.next;
    }
    return s.data;
}
```

Note:
if (f.next.next == null)
break;

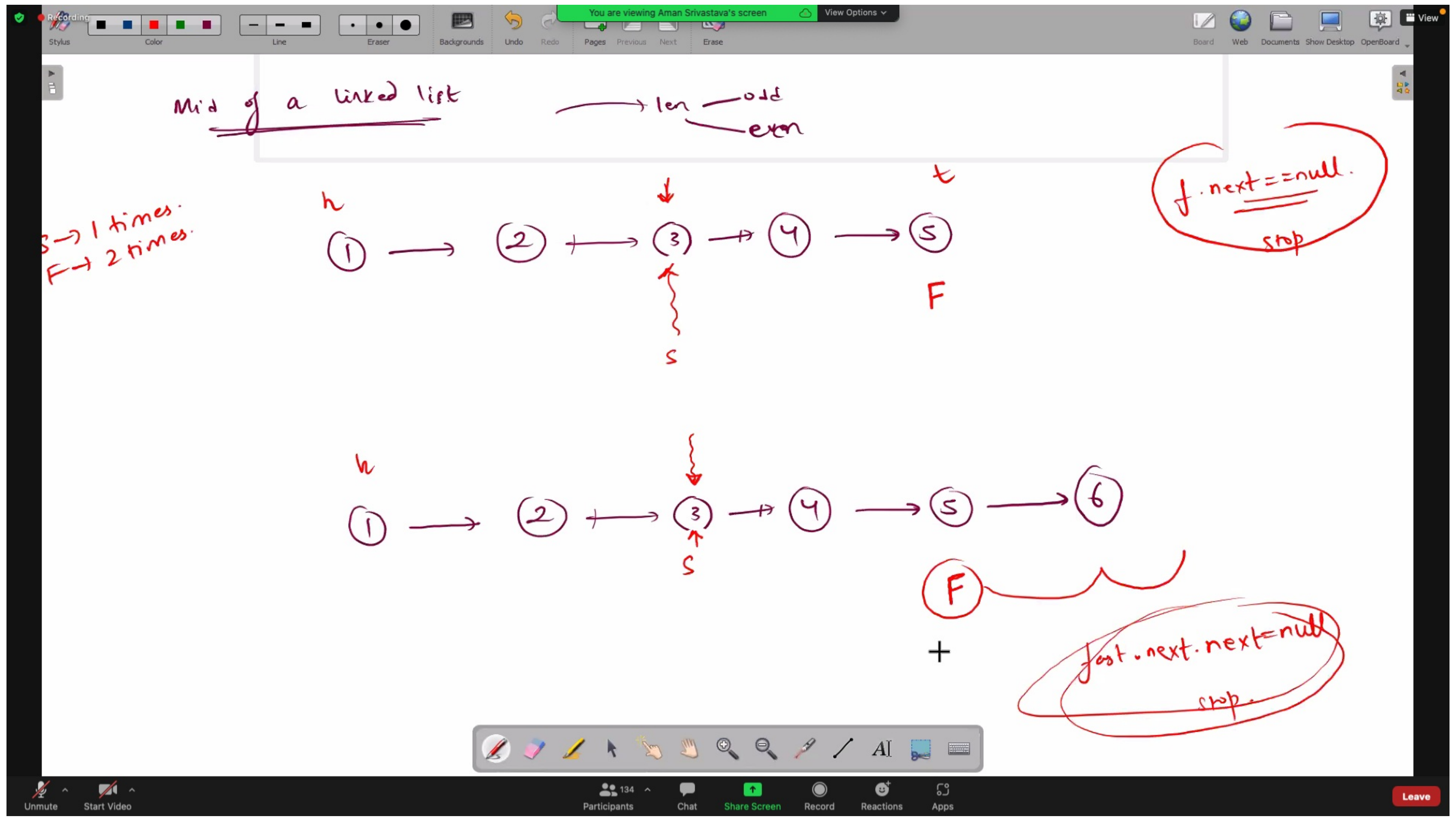
2 cases:
depends @ length

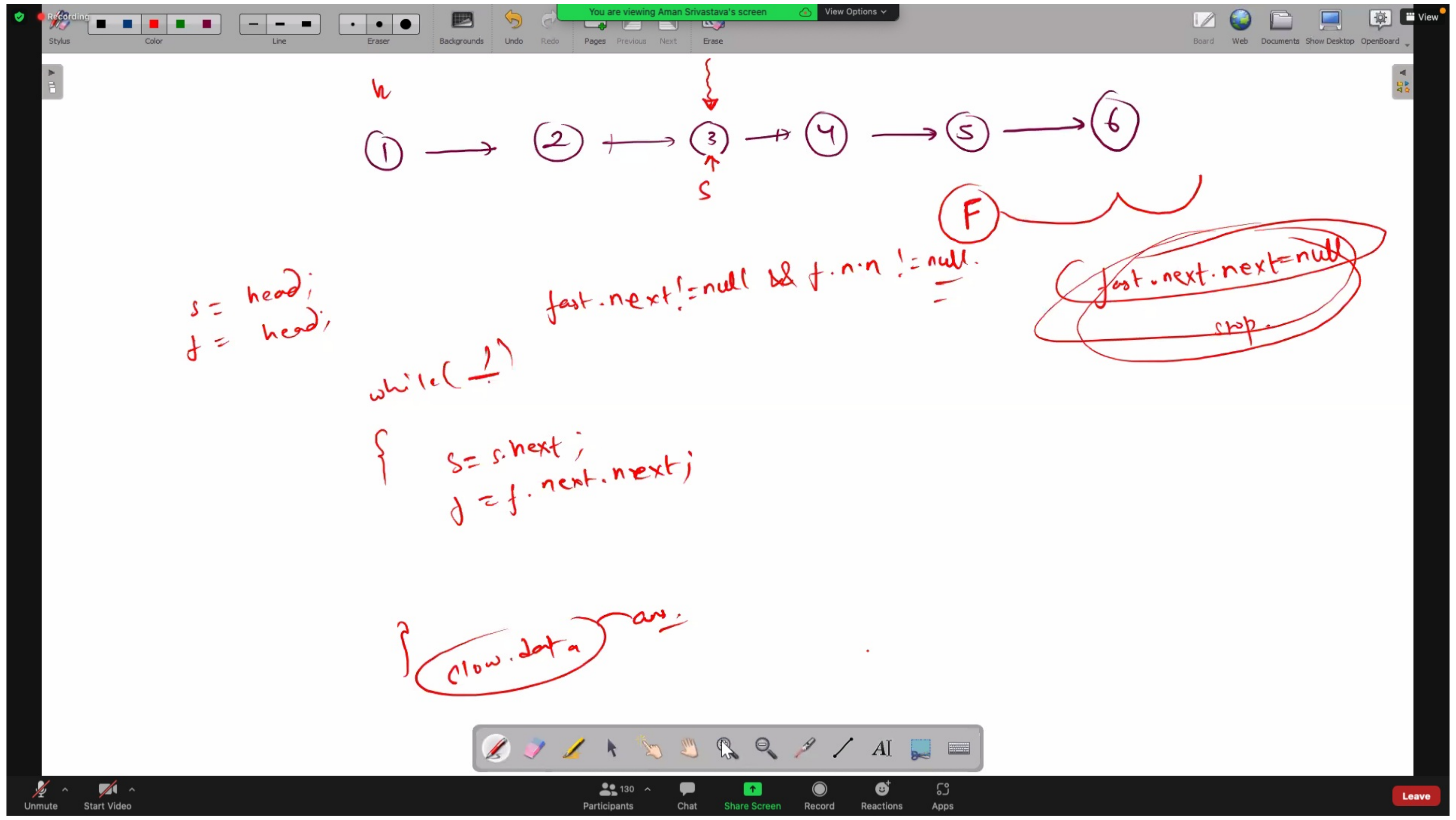
even
(consider - left end)

odd
(unambigious)

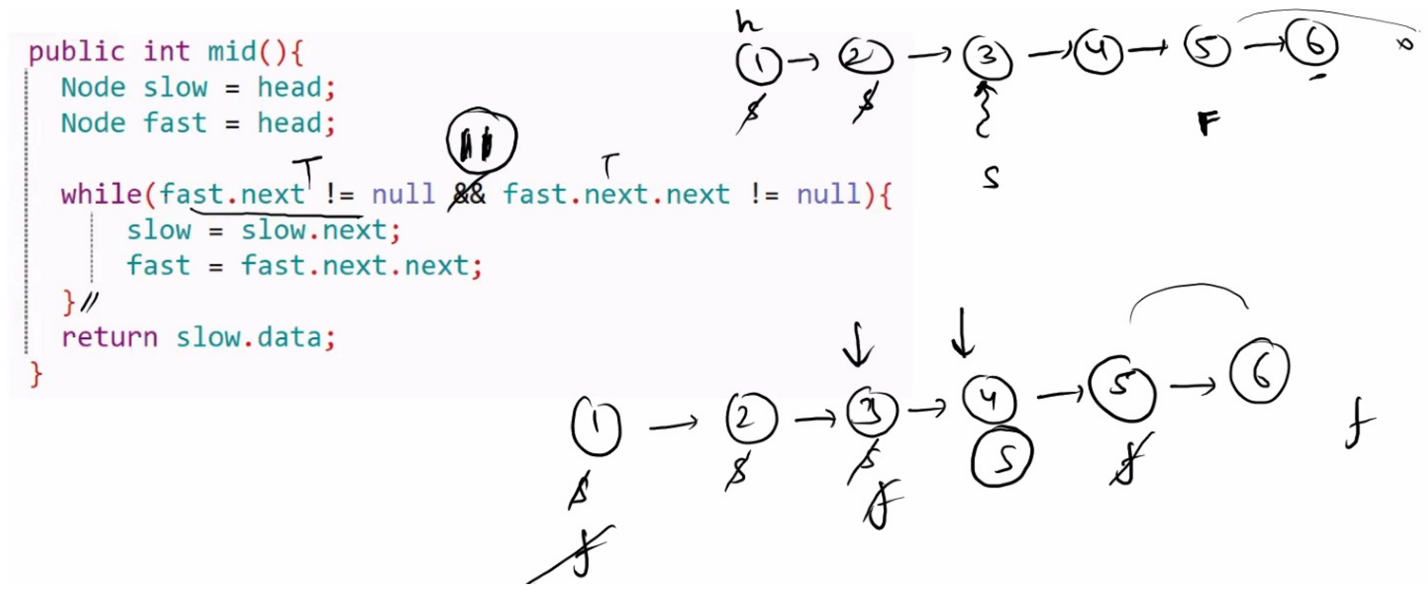
Strategy: -

move fast by 2
slow by 1
at the end slow reaches
midpoint.



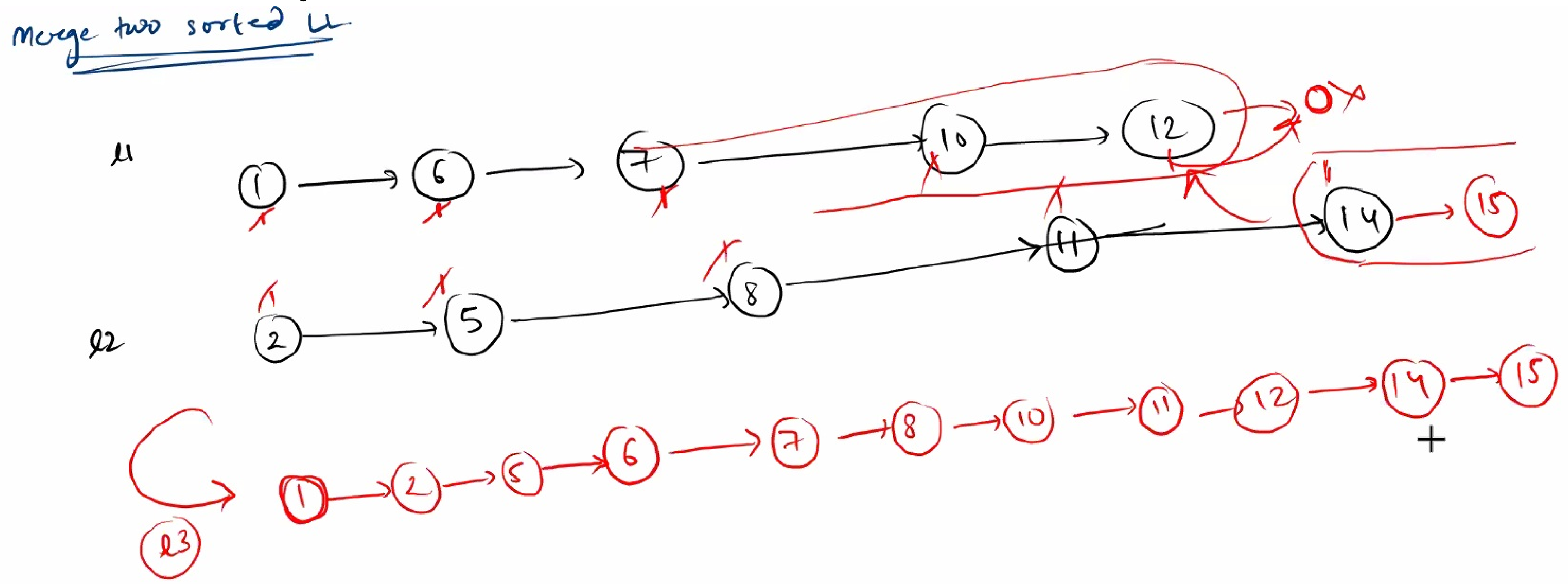


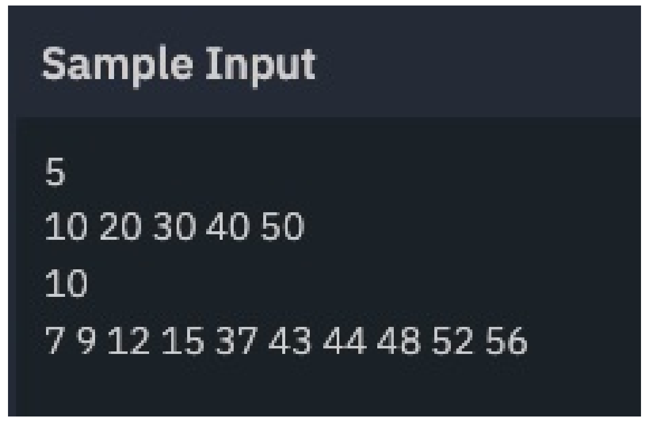
```
s = head;
f = head;
        fast.next!=null && f.n.n != null.
while( ? )
{
    s= s.next ;
    f = f.next.next;
}
  slow.data  ans.
```



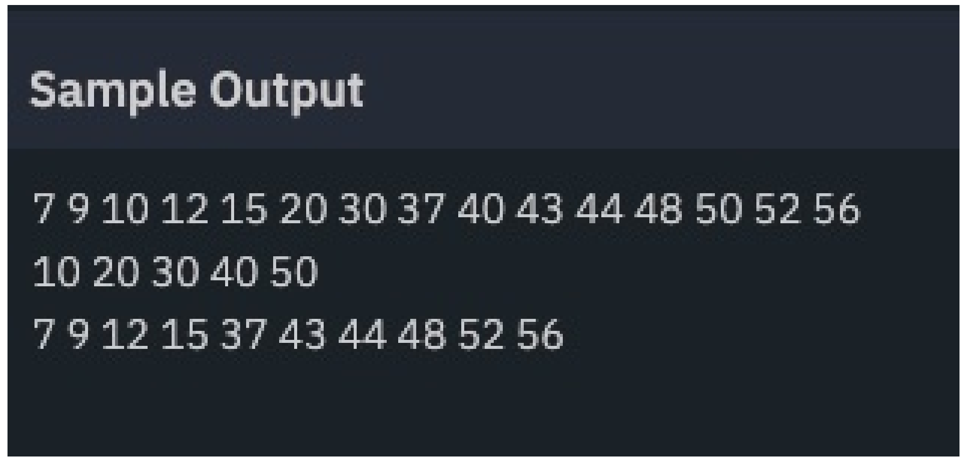
```
public int mid(){
  Node slow = head;
  Node fast = head;

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

Problem 3 : Merge two sorted Linked Lists

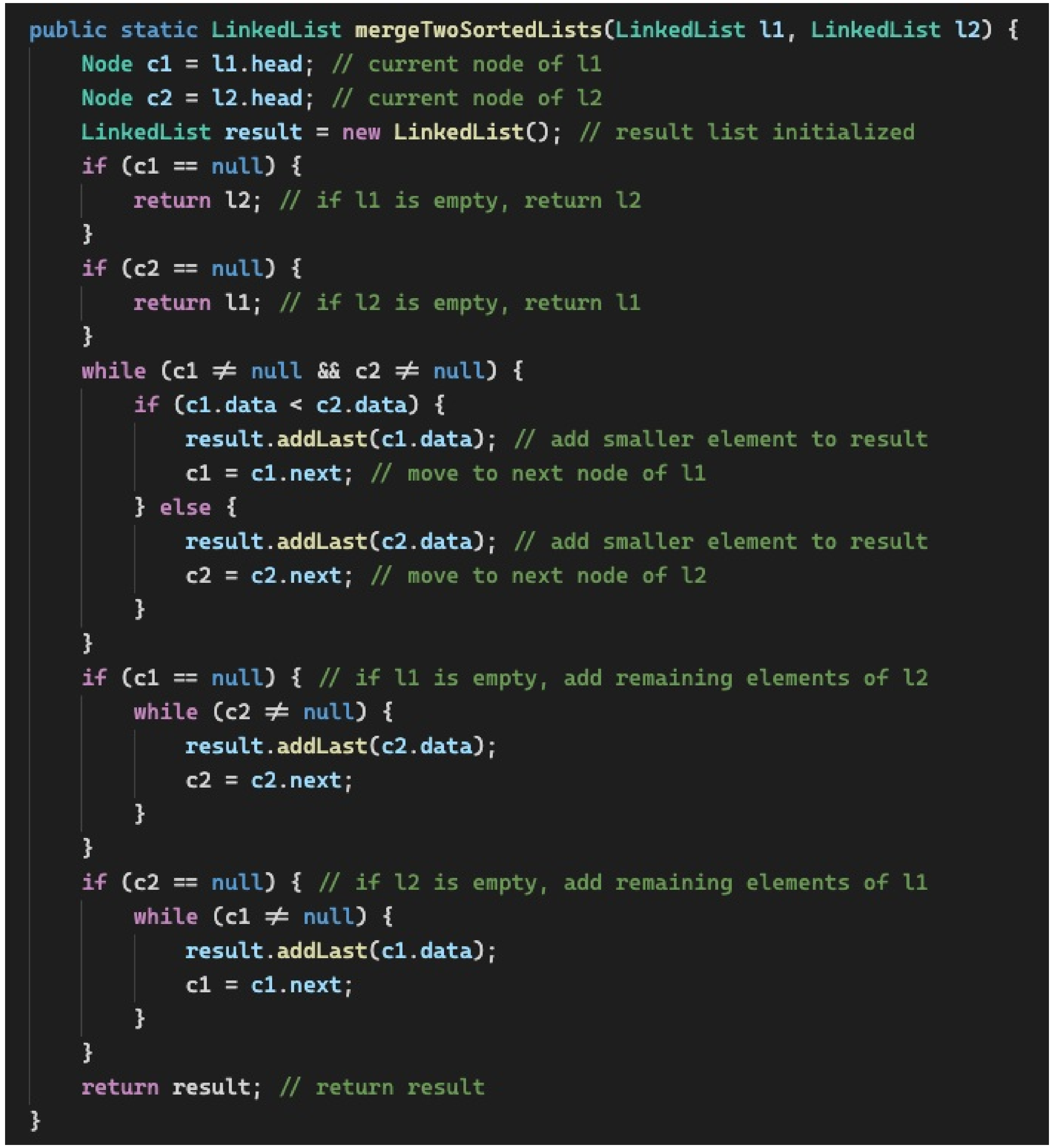



Sample Input

```
5
10 20 30 40 50
10
7 9 12 15 37 43 44 48 52 56
```


Sample Output

```
7 9 10 12 15 20 30 37 40 43 44 48 50 52 56
10 20 30 40 50
7 9 12 15 37 43 44 48 52 56
```



```
public static LinkedList mergeTwoSortedLists(LinkedList l1, LinkedList l2) {
    Node c1 = l1.head; // current node of l1
    Node c2 = l2.head; // current node of l2
    LinkedList result = new LinkedList(); // result list initialized
    if (c1 == null) {
        return l2; // if l1 is empty, return l2
    }
    if (c2 == null) {
        return l1; // if l2 is empty, return l1
    }
    while (c1 ≠ null && c2 ≠ null) {
        if (c1.data < c2.data) {
            result.addLast(c1.data); // add smaller element to result
            c1 = c1.next; // move to next node of l1
        } else {
            result.addLast(c2.data); // add smaller element to result
            c2 = c2.next; // move to next node of l2
        }
    }
    if (c1 == null) { // if l1 is empty, add remaining elements of l2
        while (c2 ≠ null) {
            result.addLast(c2.data);
            c2 = c2.next;
        }
    }
    if (c2 == null) { // if l2 is empty, add remaining elements of l1
        while (c1 ≠ null) {
            result.addLast(c1.data);
            c1 = c1.next;
        }
    }
    return result; // return result
}
```

Dry Run:

c1 . . . . . .

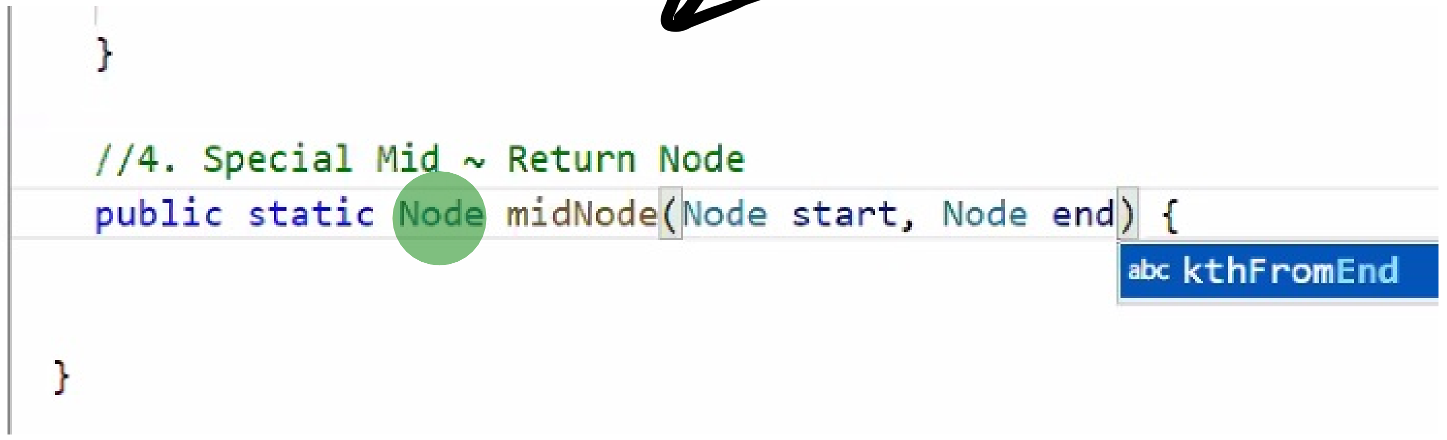
l1={1, 6, 7, 10, 12} c1

l2={2, 3, 4, 8, 11, 14, 15} c2

c2 . . . . . .

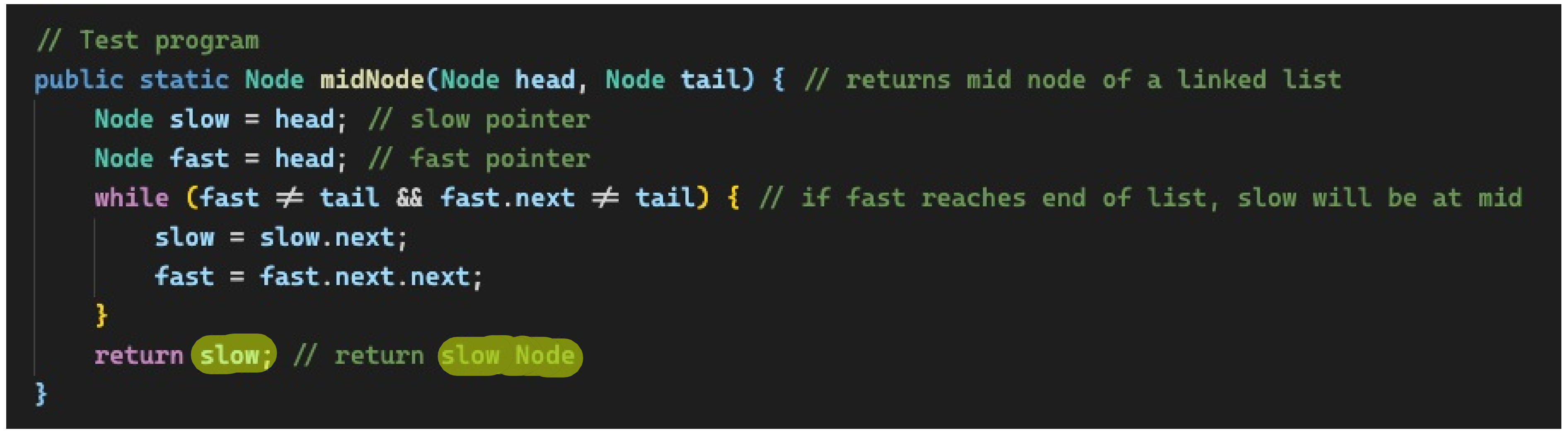
1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 15

Q. Task to complete the f^n^ of Node return type.



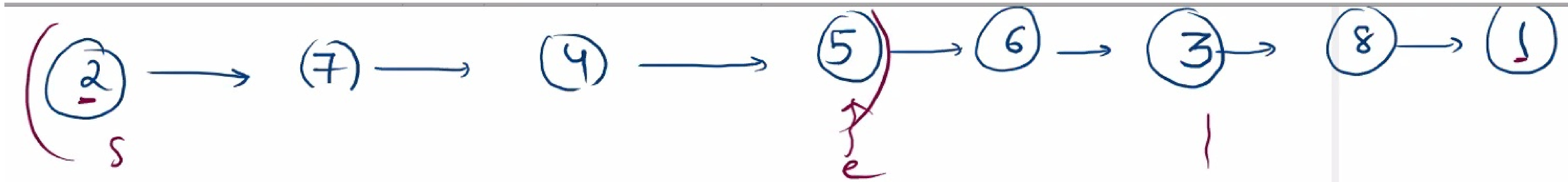
```
    }

    //4. Special Mid ~ Return Node
    public static Node midNode(Node start, Node end) {
                                                    abc kthFromEnd

}
```

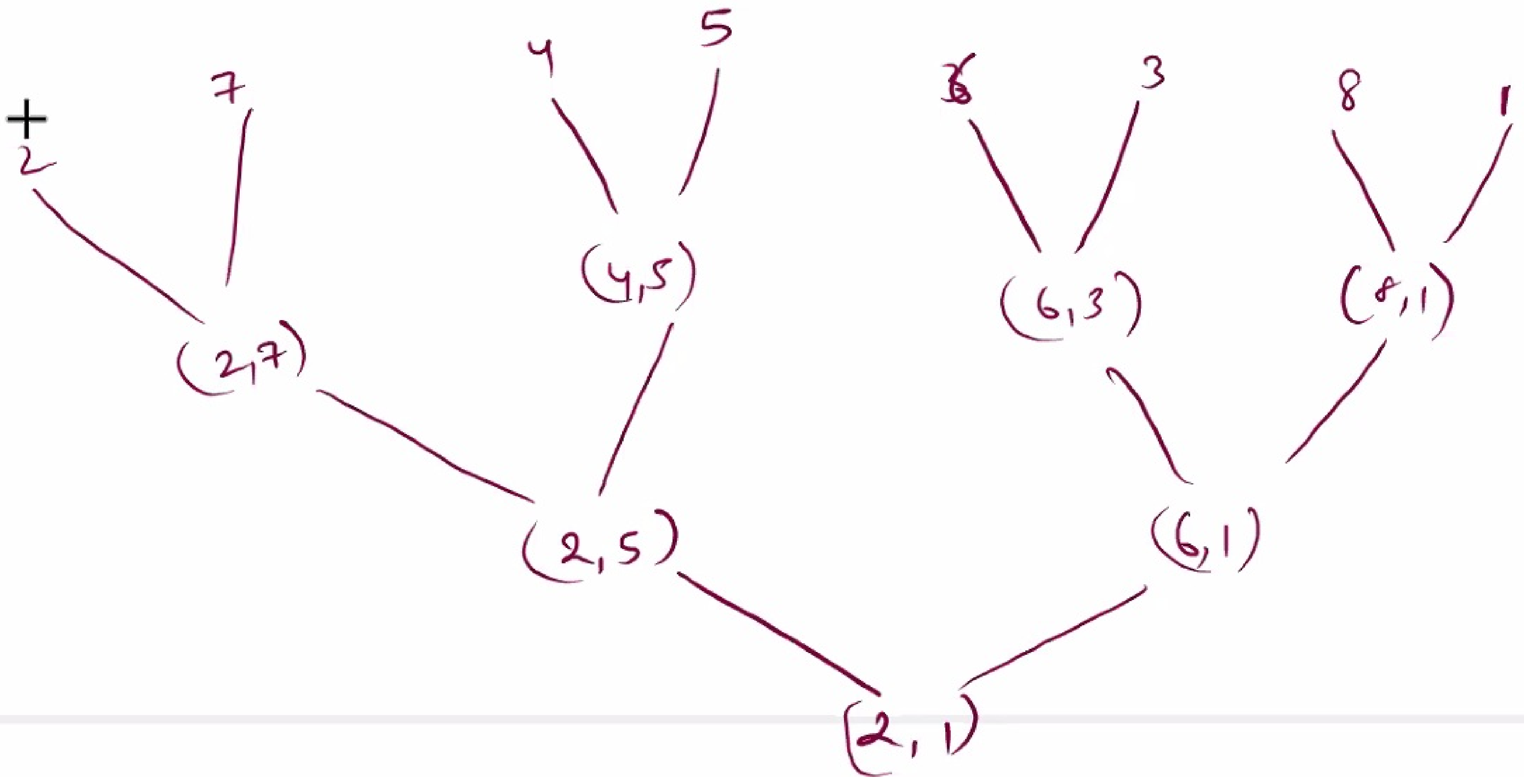
sol^n^:

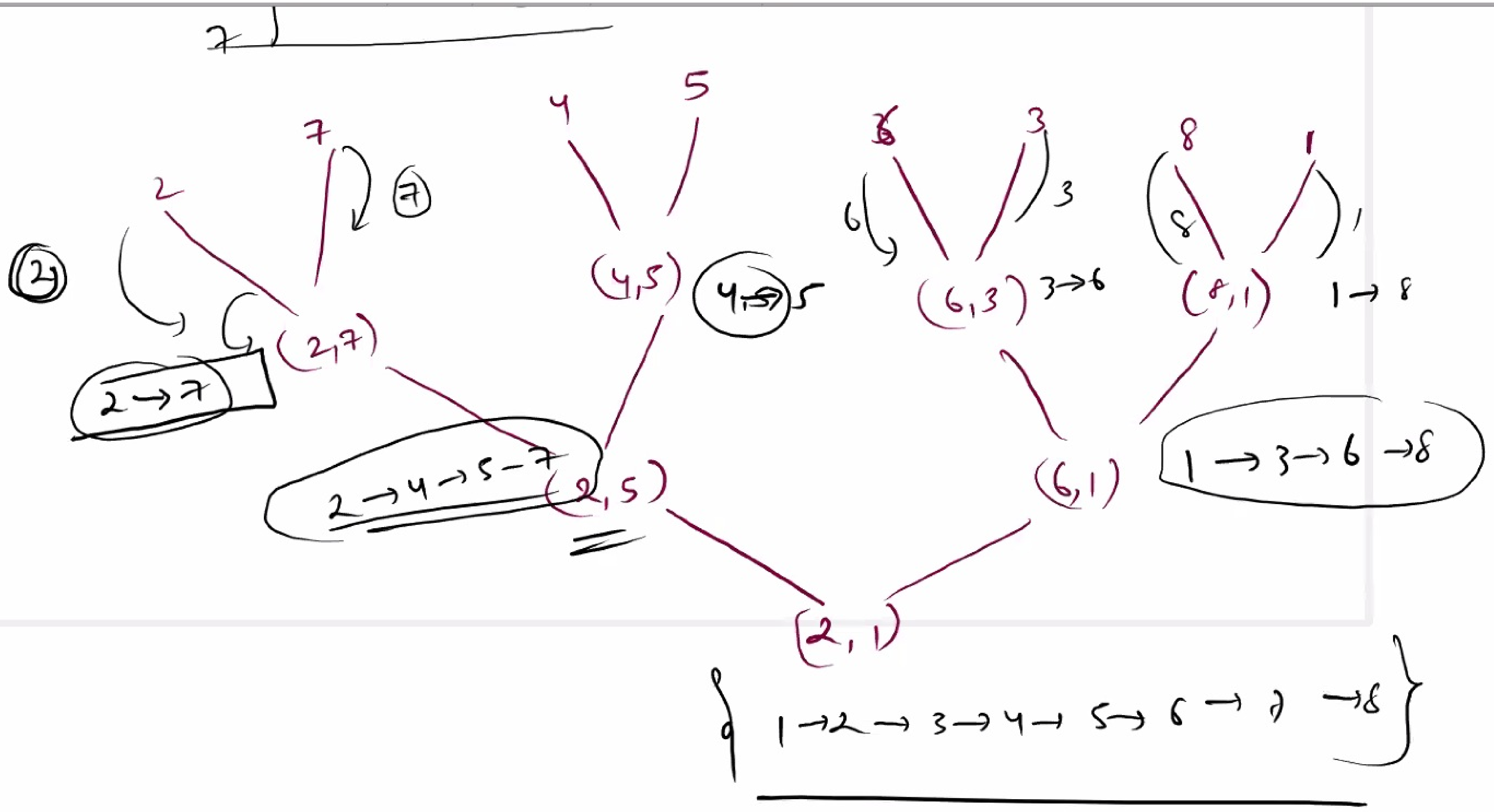


```
// Test program
public static Node midNode(Node head, Node tail) { // returns mid node of a linked list
    Node slow = head; // slow pointer
    Node fast = head; // fast pointer
    while (fast ≠ tail && fast.next ≠ tail) { // if fast reaches end of list, slow will be at mid
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow; // return slow Node
}
```

Problem 3: Merge Sort a Linked List.






7
2
7
2
7
(2,7)
2→7
4
5
(4,5)
4→5
6
3
6
3
(6,3)
3→6
8
1
8
1
(8,1)
1→8
2→4→5→7
(2,5)
(6,1)
1→3→6→8
(2,1)
{1→2→3→4→5→6→7→8}