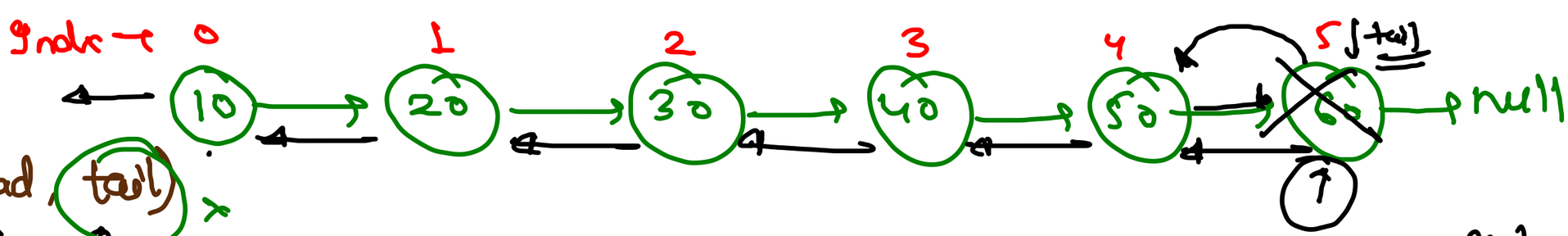
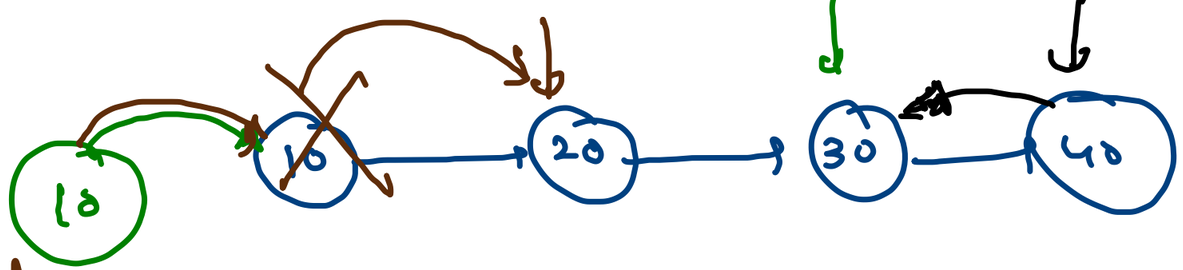


get → ① getFirst()

n = size of linked list

n log(n) max value.

② getLast()



Complexity of getLast()

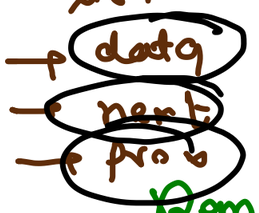
→ tail ✗

✓ O(n)

tail ⇒ O(1)

STL → Doubly linked list

Assumption → Head & tail



head = head.next

pointer - Arithmatic } O(1)

removeFirst() O(1)

removeLast() O(n)

removeAt() O(n)

addFirst O(1)

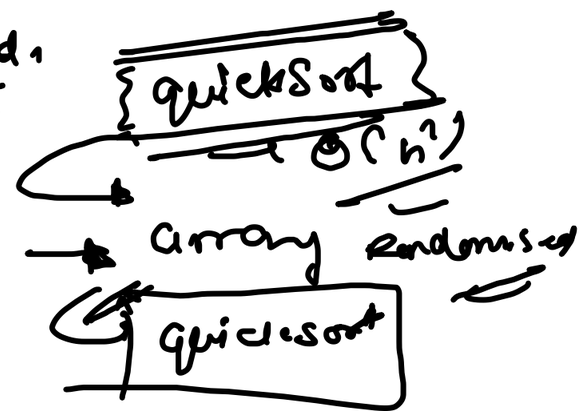
addLast O(1)

addAt → O(n)

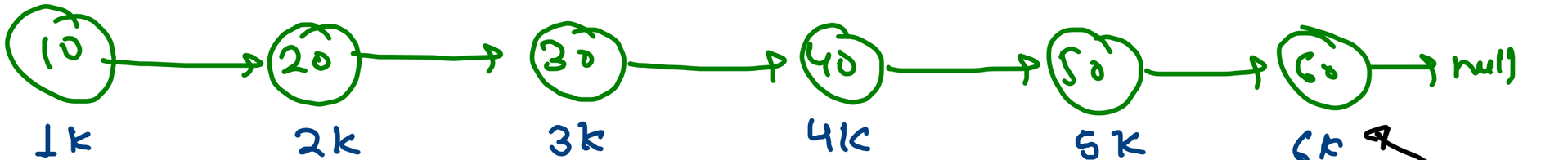
Remove Last

Second Last → (n-1) traverse → O(n)

upper bound



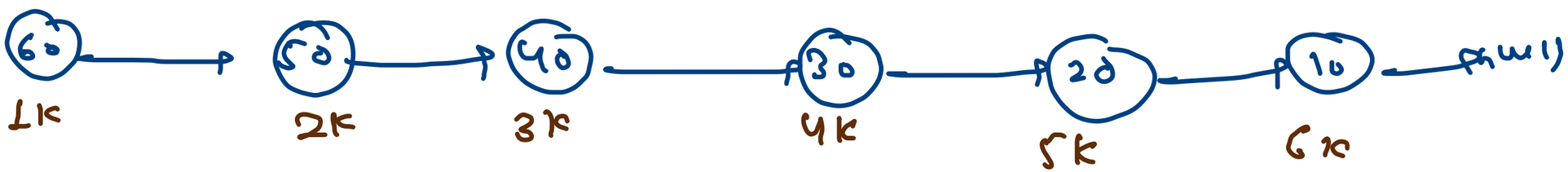
Reverse Linked List (Data Iterative) Complexity - $O(n^2)$ ≈



Address of head & tail → 1k & 2k

↓ Reverse data || Don't interface in connection.

Result/ op



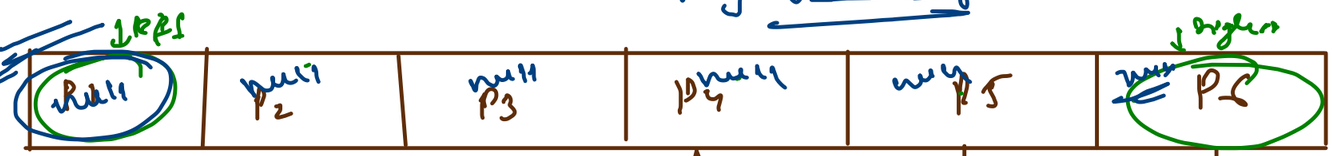
Address of head & tail → 1k & 2k

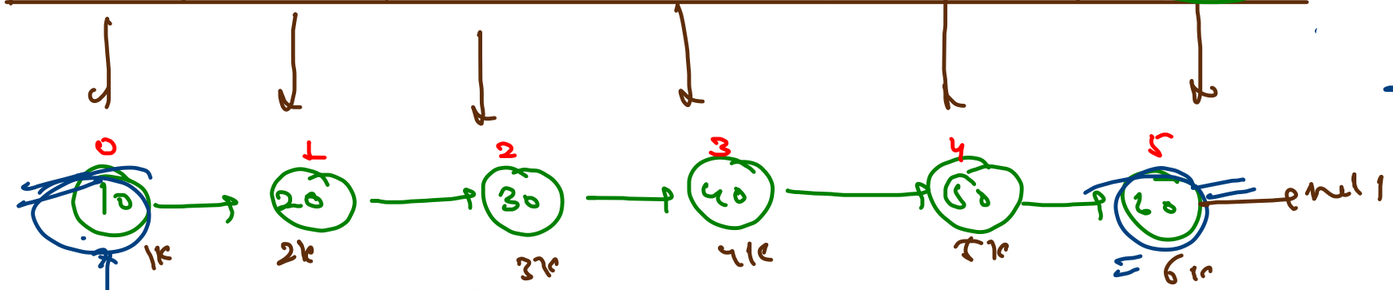
Node [ ] nodeArr = new Node (6) ;

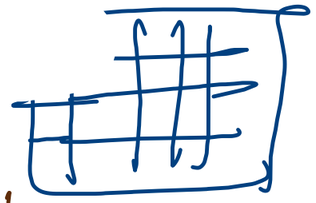
(Underflow) | objectArray

O(n)

O(n)







temp2

Array Reverse

left = 0

right = size - 1 = 5

Ammuthu ✓

Devender ✓

Rohan ✓

B.R. ✓

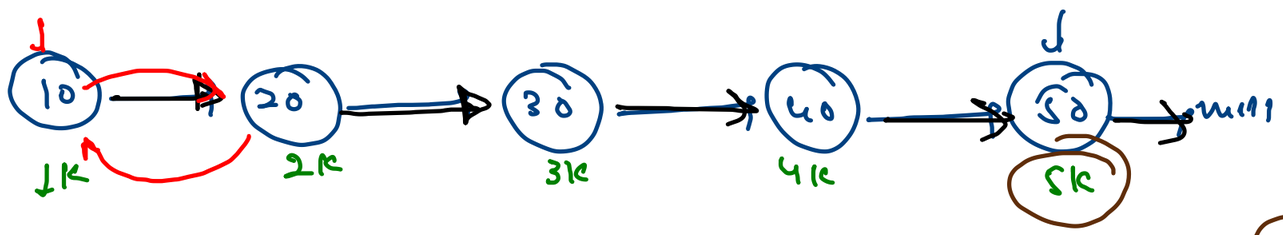
Alekhya ✓

Darsan ✓

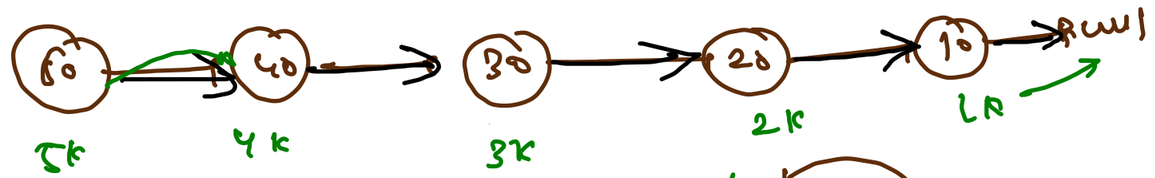
Nadeem ✓

```
while ( left < right ) {                    O(n)
    Node leftNode = temp2;
    Node rightNode = getNodeAt(right);      Abstract
                                            O(n)
    int temp1 = leftNode.data;
    leftNode = rightNode.data;
    rightNode.data = temp1;

    left++;
    right--;
    temp2 = temp2.next;
}
```



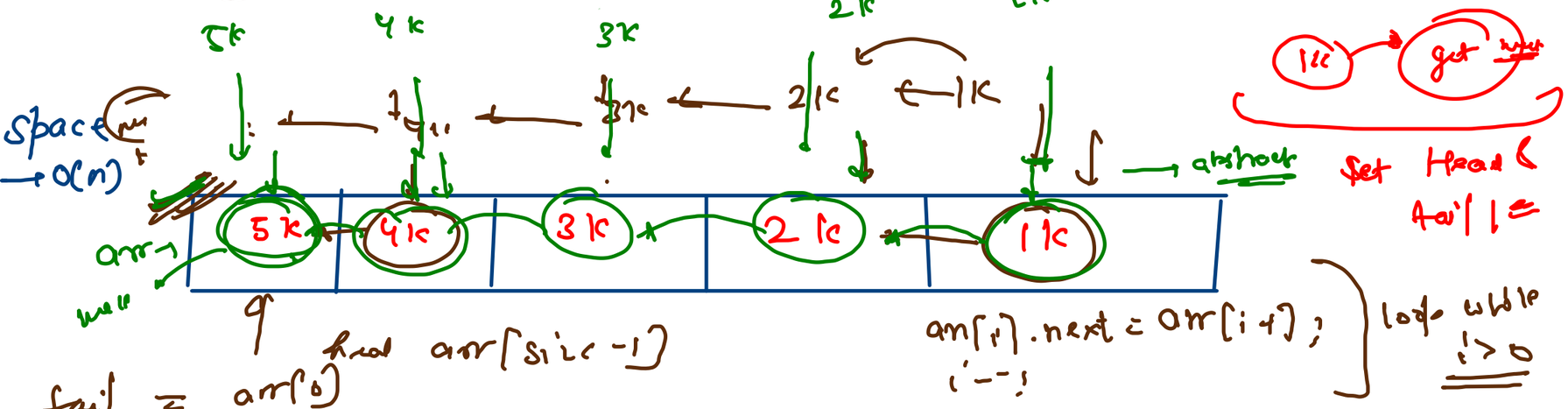
Reverse pointer iteratively. →



complexity - O(n)

5K → get 4K
set 5K → 4K
move 4K

4K → get 3K
set 4K → 3K
move 3K



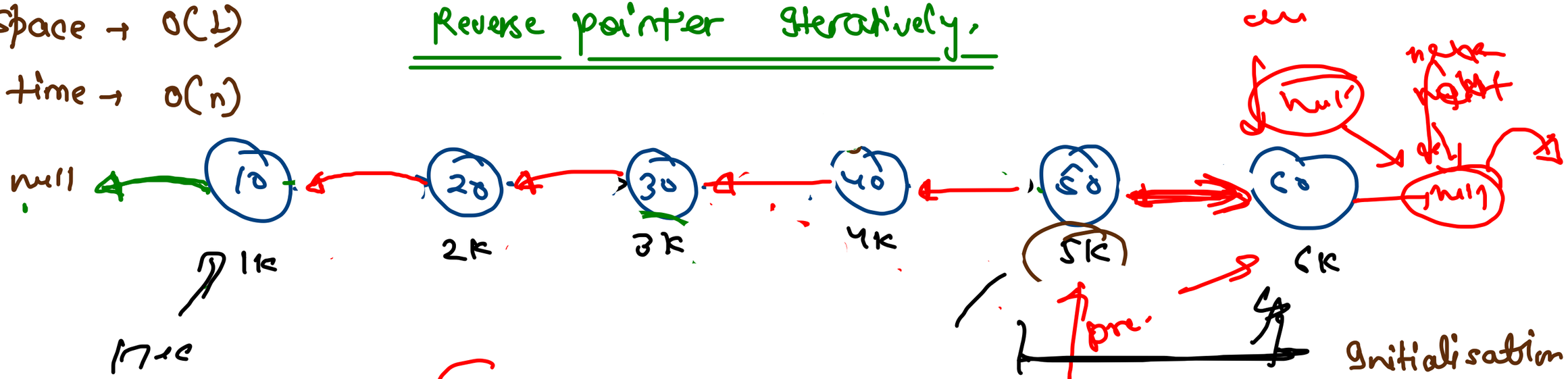
Pseudo Code

```
for(int i = arr.length-1; i > 0 ; i--){
        arr[i].next = arr[i-1];
}
arr[0].next = null,
Head = arr[arr.length-1];
tail = arr[0];
```

Space → O(1)

time → O(n)

Reverse pointer iteratively.



while next != null

curr.next = prev;

head =

1. curr.next = prev
2. prev = curr
3. curr = next
4. next = next.next

```
curr = head;
prev = null;
next = head.next
```

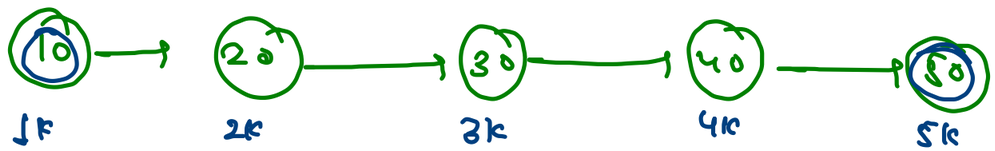
desires →

→ I want to set next of curr on prev node

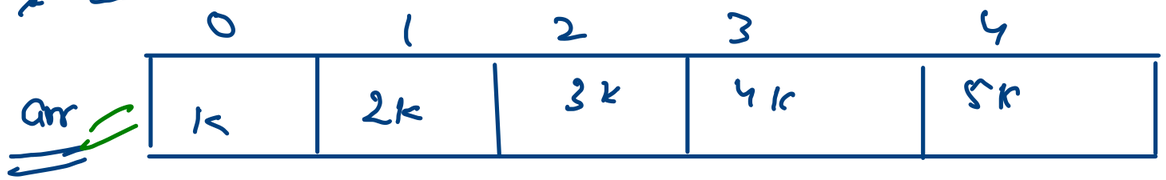
→ I will hold a next pointer before set of current

→ Because I want to set current node's next on prev node so I hold prev. node too.

if space allowed Data Iterative



i:0 to 2



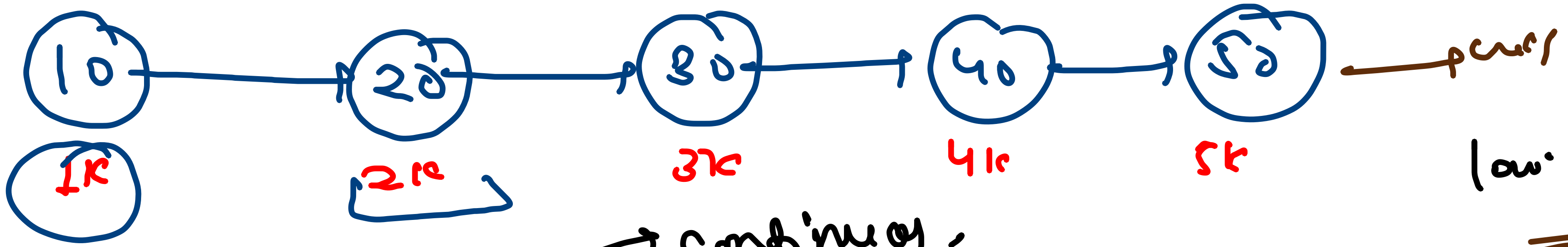

```
left=0
right: 4;

while(left < right){
    Node leftNode = arr[left];
    Node rightNode = arr[right];

      int temp=leftNode.data;
      leftNod.data = rightNode.data;
       rightNode.data=temp;

    left++
    right:--
}
```

print linked List Recursively.



1. Expectation
2. faith
3. merging

] High level

I. low level Approach → Base case

continuous array - size x

linked list

array

if (temp == null) return;

Recursive call

10 20 30 40 50

stack

Java (10,000)

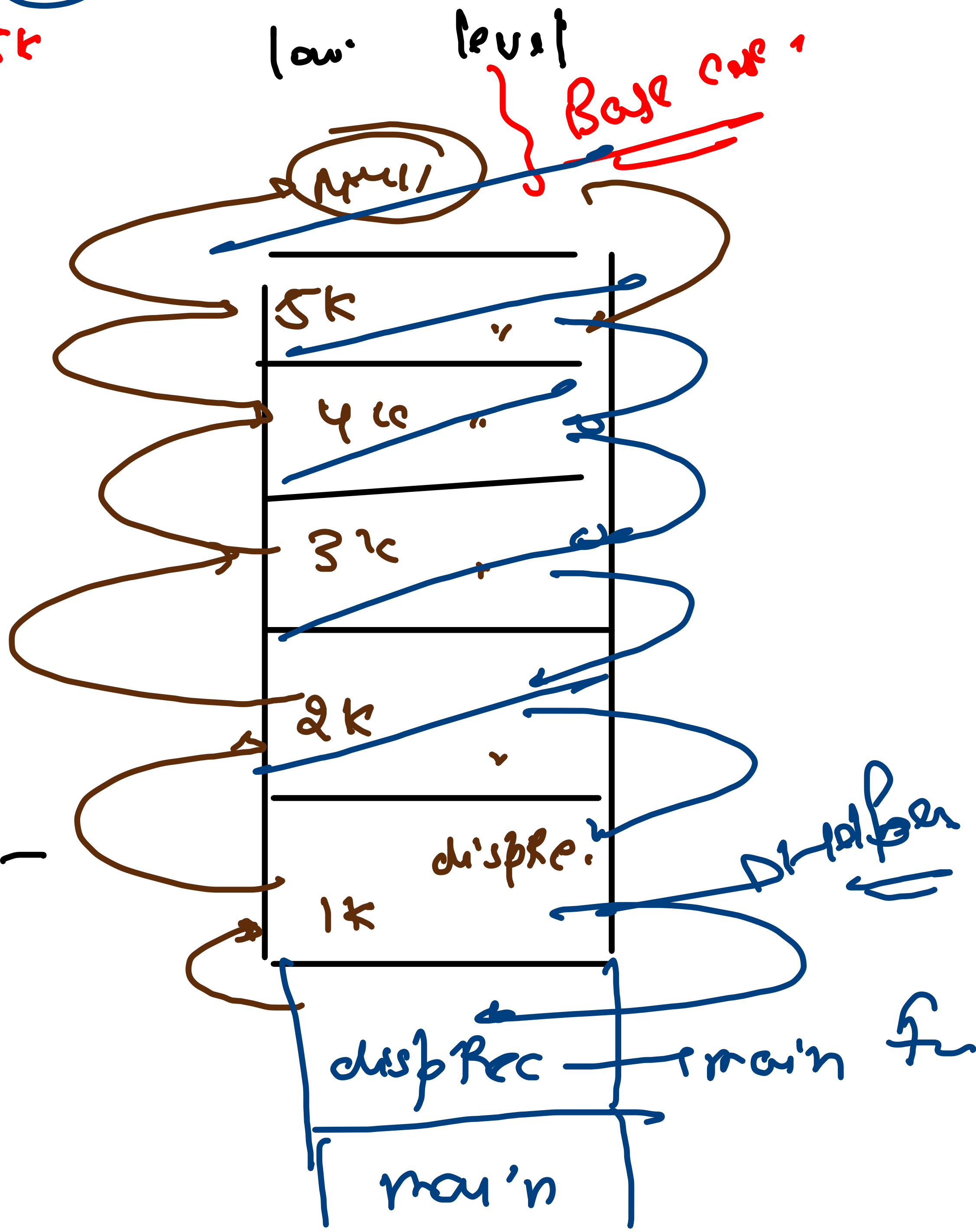
High level

fait → call (temp.next);

→ print (temp.data);

call (temp.next);

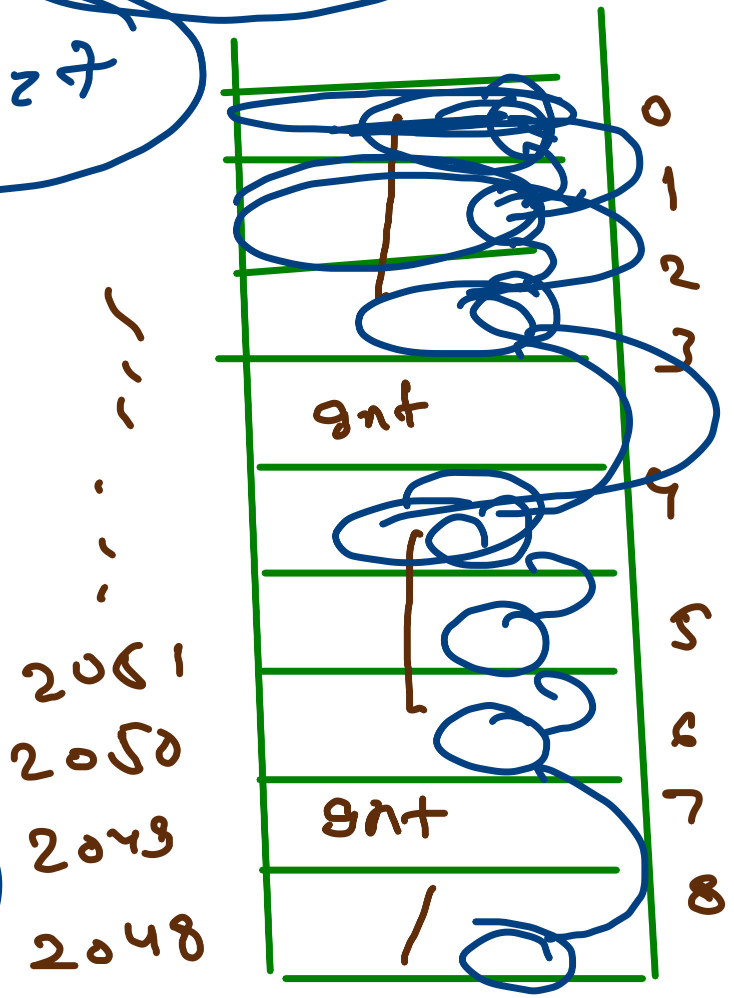
Search



	Max Size	Reason
array	10^9	Continus Memory
Linked	More than array	No continuous memory allocation

Size = 3 array

Size of linked list = 7



Reverse using array

Size of linked list

max size of array

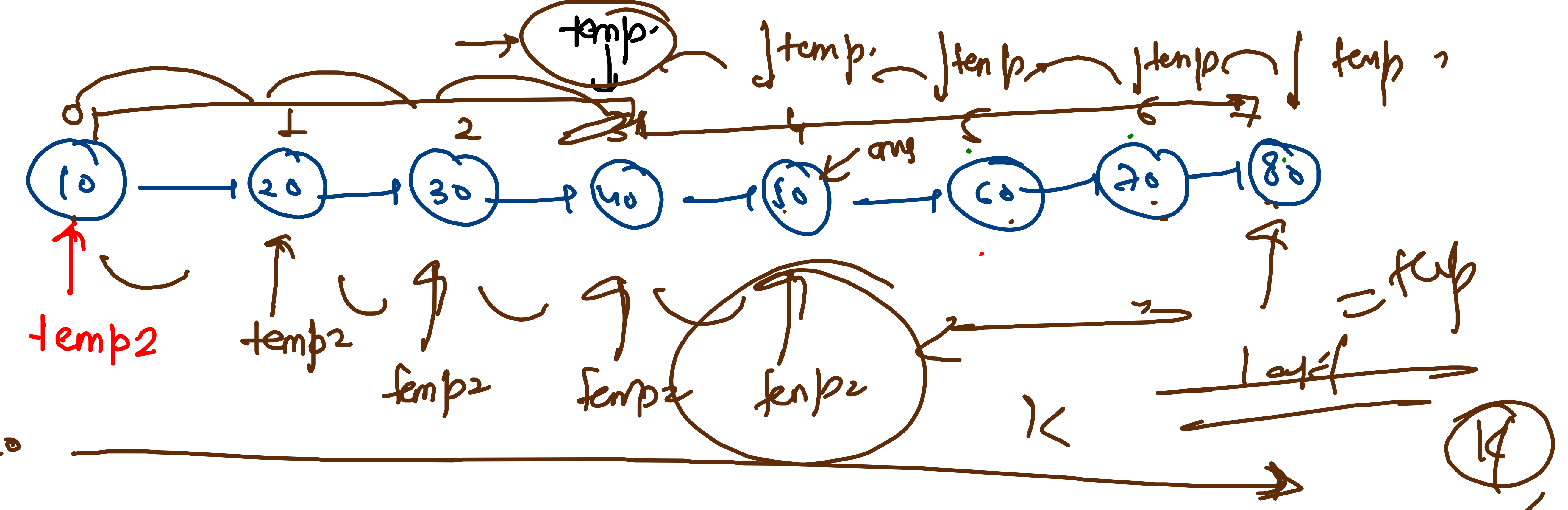
10^8

Recursion 10,000 nth 10^4

Stack size

kth Last

space → O(1)



k = 0 → 80

k = 1 → 70

k = 2 → 60

k = 3 → 50

k = 4 → 40

k = 3

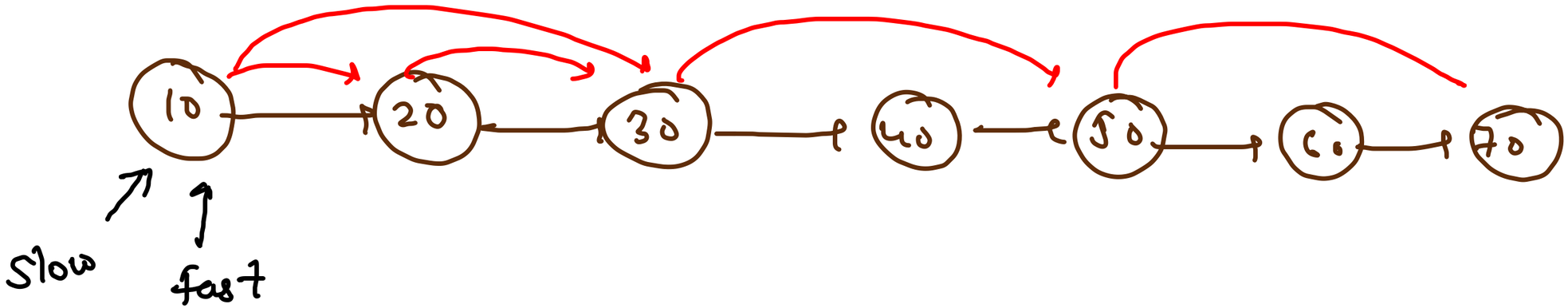
temp index – temp2 index = k

1. temp = head & move it k time.
2. temp2 = head // temp & temp2 have k gap.
3. simultaneously move temp & temp2

temp2 → curr

single traversal → loop ×

n → 2n



slow = slow.next } x

Speed of slow pointer : x.

Speed of fast pointer = 2x.

fast = fast.next.next } 2x

time = t.

d = sxt

d = 2x xt

distance traved by fast
= 2xt = d,

distance travelled by slow.

d = sxt = xxt = xt

slow = d/2

odd:

Even:

return S.data

= S0

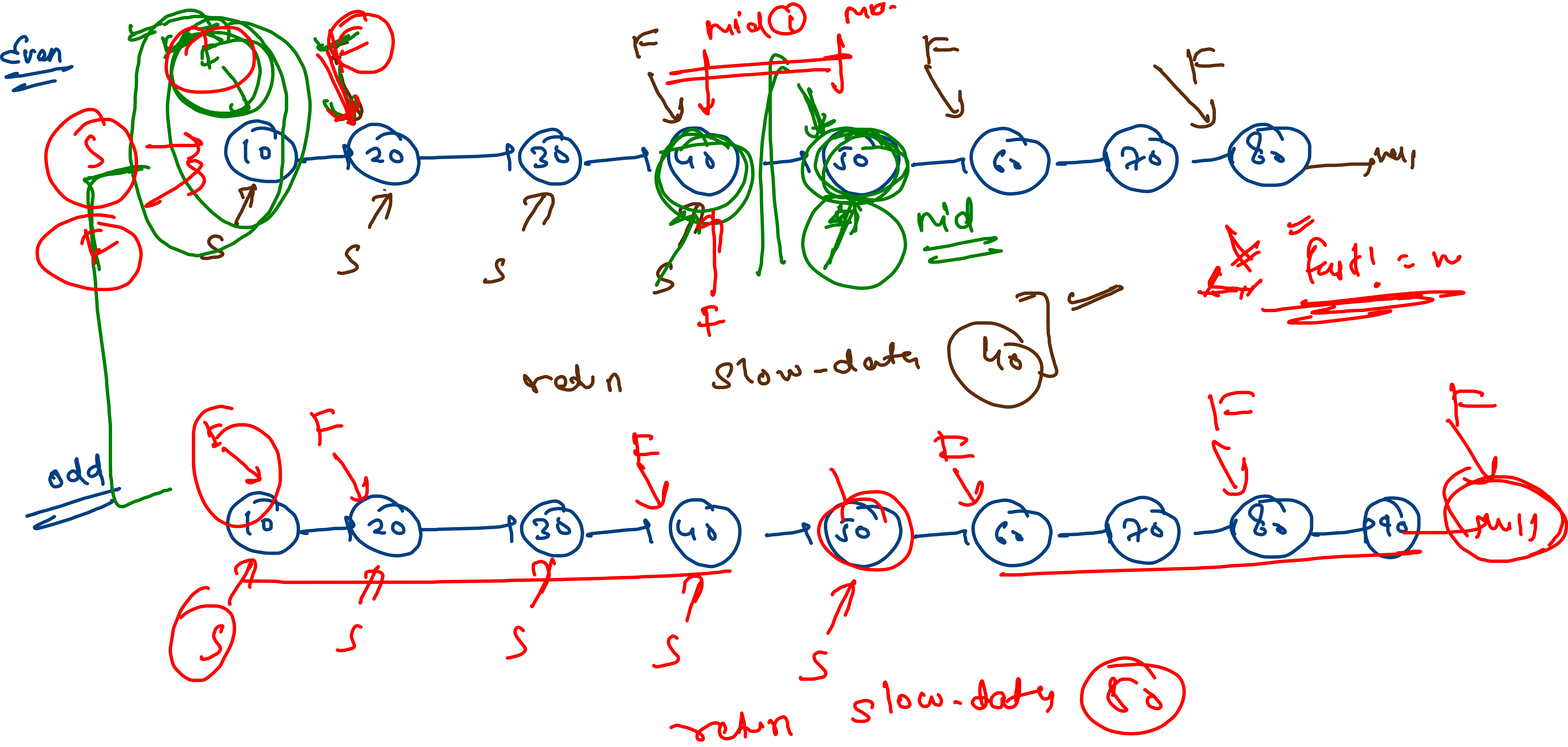
fast == null

fast.next == null

break

End of first half

Second half first.


Even
S
F
10
20
30
40
50
60
70
80
null
mid
fast != n
return slow-data 40
odd
10
20
30
40
50
60
70
80
90
null
return slow-data 50