1. Recursion - Back
2. DP
3. Linked List
4. Graph

1. Hashmap, Heap
2. Stack, Queues

Structures

Level-1
Level 2, 2-3, 6-7 H.W.
Level 3

Direction

Linked List

- Implement
- Memory management

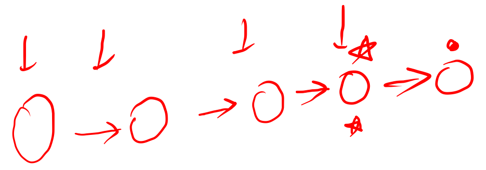
Operations

- add First , remove first
- add Last , remove last
- add At , remove at.

O(1)		O(1)
★✓ ① add First	—	remove first
② add last	—	remove last
O(1)		O(n)

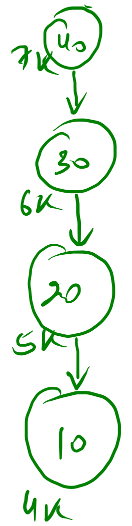
Stack ✓

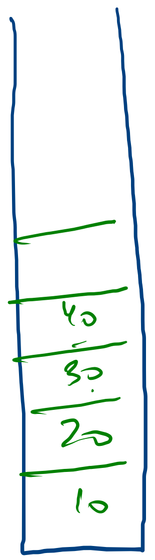
- Discipline → LIFO
- operations → push, pop, peek, size



✓

head :	7k
tail :	4k
size :	5



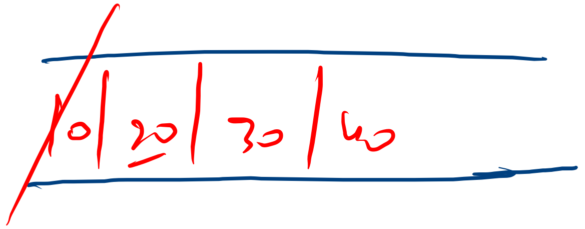


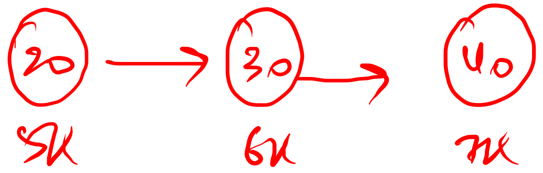
push → addFirst()
pop → removeFirst()
peek → getFirst()

push 10 ✓
push 20 ✓
push 30 ✓
peek ✓ → 30
push 40 ✓
push 50 ✓
pop → 50
peek ✓ → 40
size

Queue

- FIFO
- Operations
  - add
  - remove
  - peek





add → addLast

remove → removeFirst

peek → getFirst

head = 5K
tail = 7K
size = 3

✓ add 10

✓ add 20

✓ add 30

✓ add 40

→ Size

→ remove → 10

peek → 20

remove

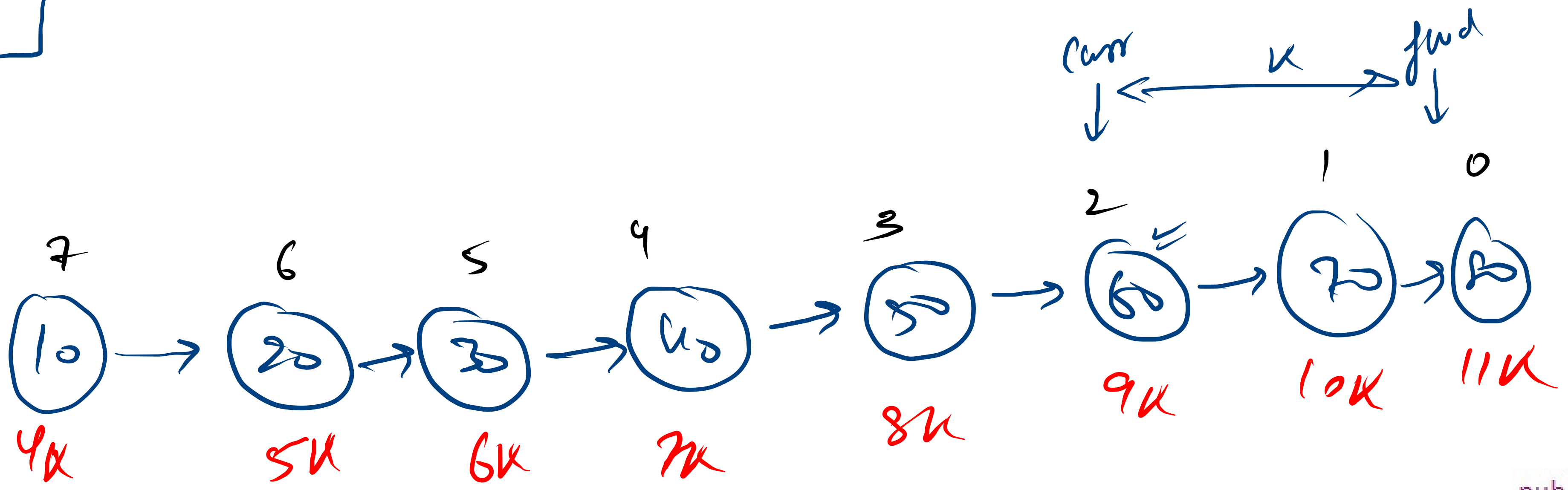
remove

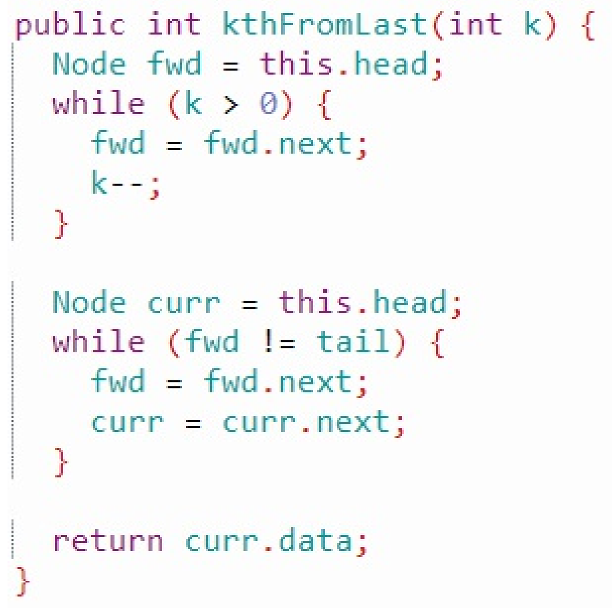
peek

head = 4k
tail = 11k
size = 8

~~th~~ Size ×

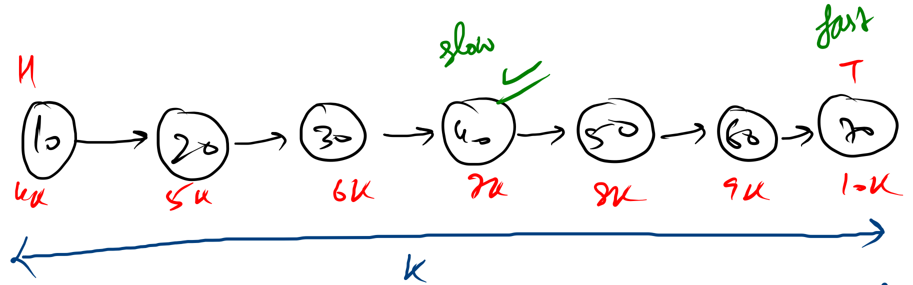
K => ~~2~~ 10 ✓




```
public int kthFromLast(int k) {
  Node fwd = this.head;
  while (k > 0) {
    fwd = fwd.next;
    k--;
  }

  Node curr = this.head;
  while (fwd != tail) {
    fwd = fwd.next;
    curr = curr.next;
  }

  return curr.data;
}
```



A ______________________ A T_A

B ____________ B T_B

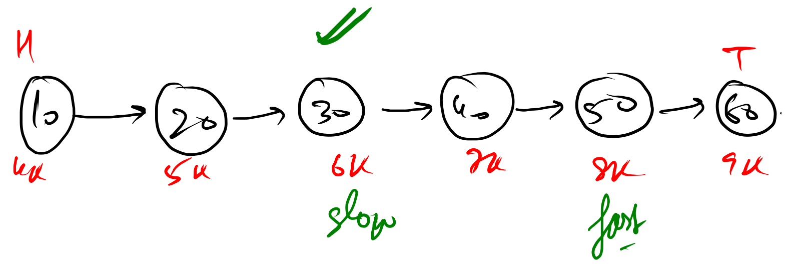
Speed $\Rightarrow \frac{\text{Distance}}{\text{time}}$,

$S_A = \frac{x}{T_A}$, $S_B = \frac{x}{2T_B}$

$T_A = \frac{x}{S_A}$, $T_B = \frac{x}{2 S_B}$

$\cancel{t}_A = \cancel{t}_B$

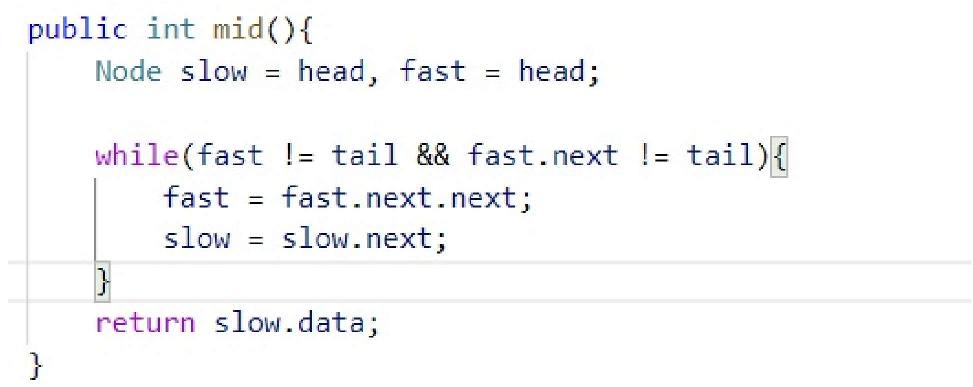
$\frac{\cancel{x}}{S_A} = \frac{\cancel{x}}{2S_B} \Rightarrow \boxed{S_A = 2S_B}$



fast == tail ↳ slow → mid ,

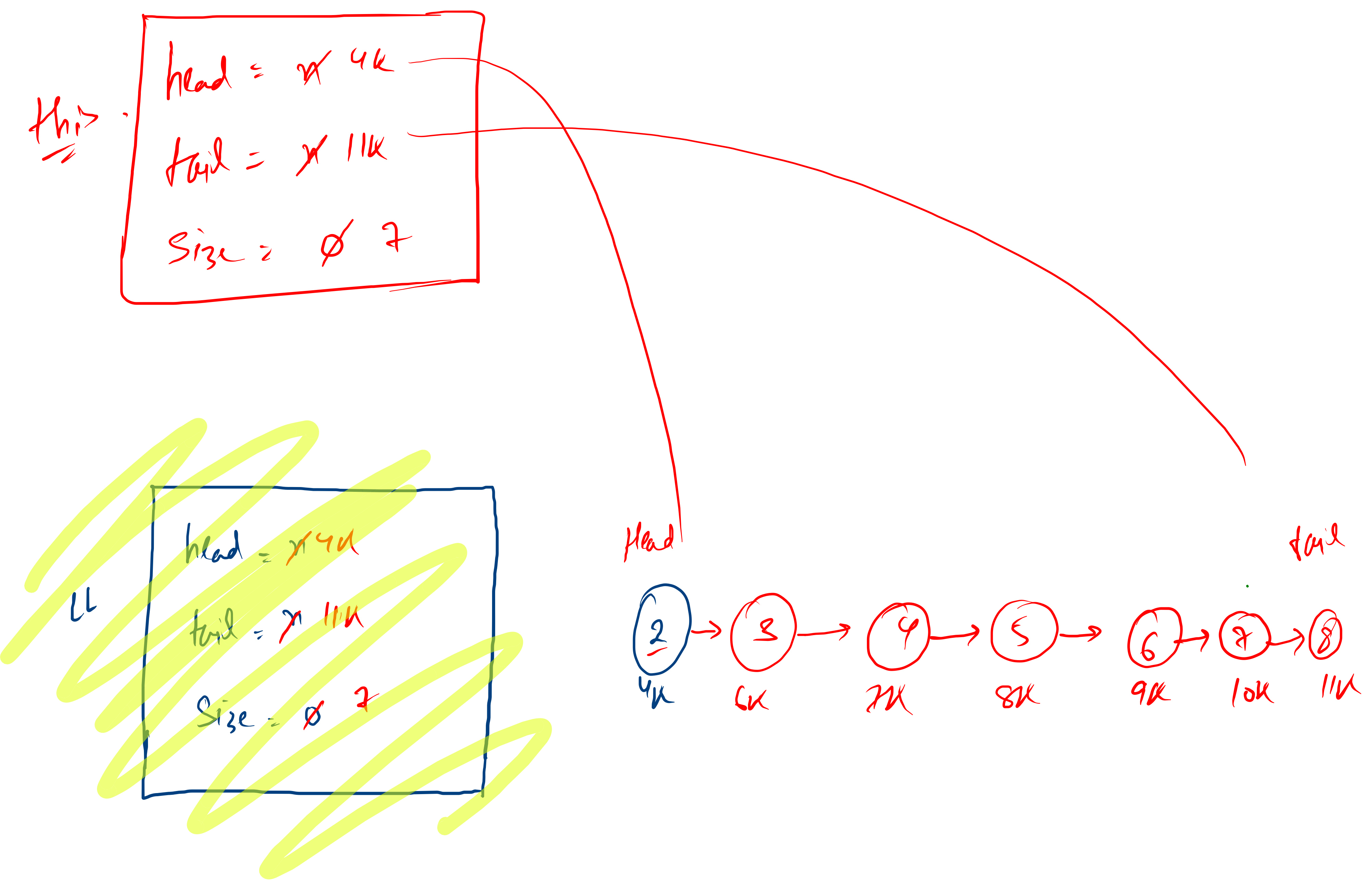
fast . next == null

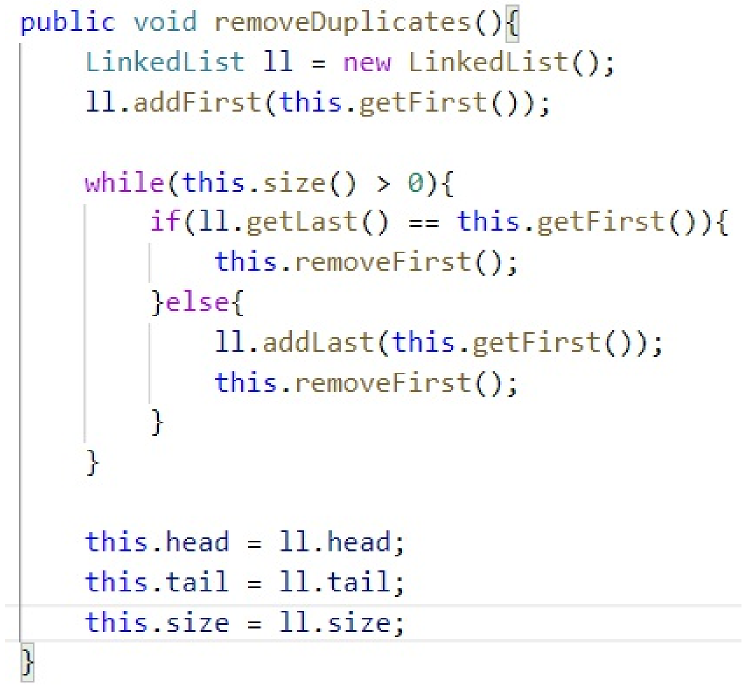
fast . next == tail ↳ slow → mid



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

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





```
public void removeDuplicates(){
    LinkedList ll = new LinkedList();
    ll.addFirst(this.getFirst());

    while(this.size() > 0){
        if(ll.getLast() == this.getFirst()){
            this.removeFirst();
        }else{
            ll.addLast(this.getFirst());
            this.removeFirst();
        }
    }

    this.head = ll.head;
    this.tail = ll.tail;
    this.size = ll.size;
}
```

this

head →	n
tail →	n
size →	0

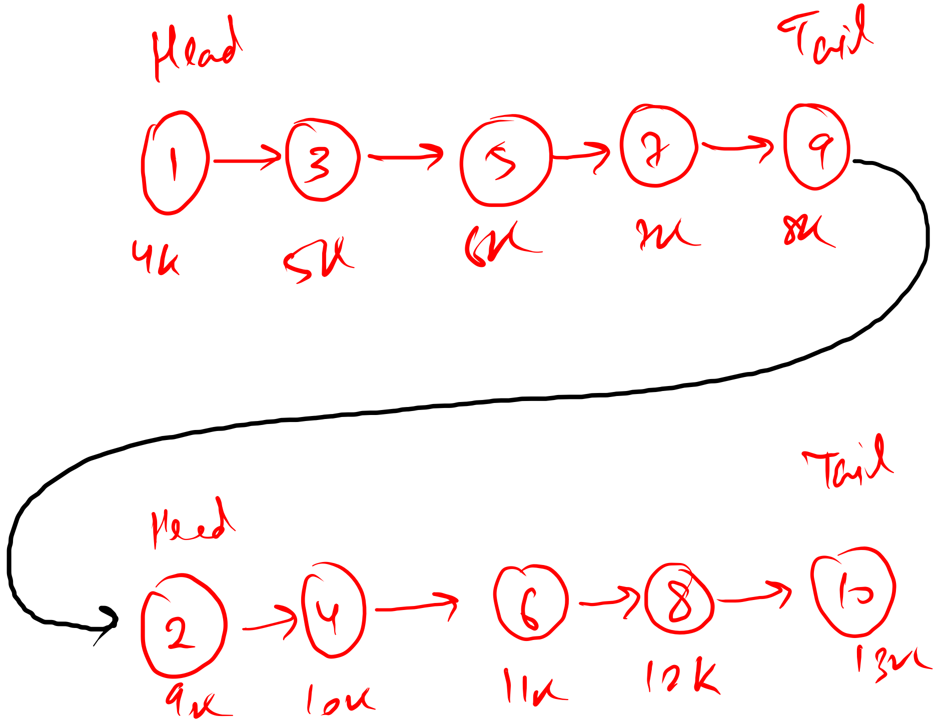
odd - Even → H.W.

odd

head →	4k
tail →	8k
size →	5

Even

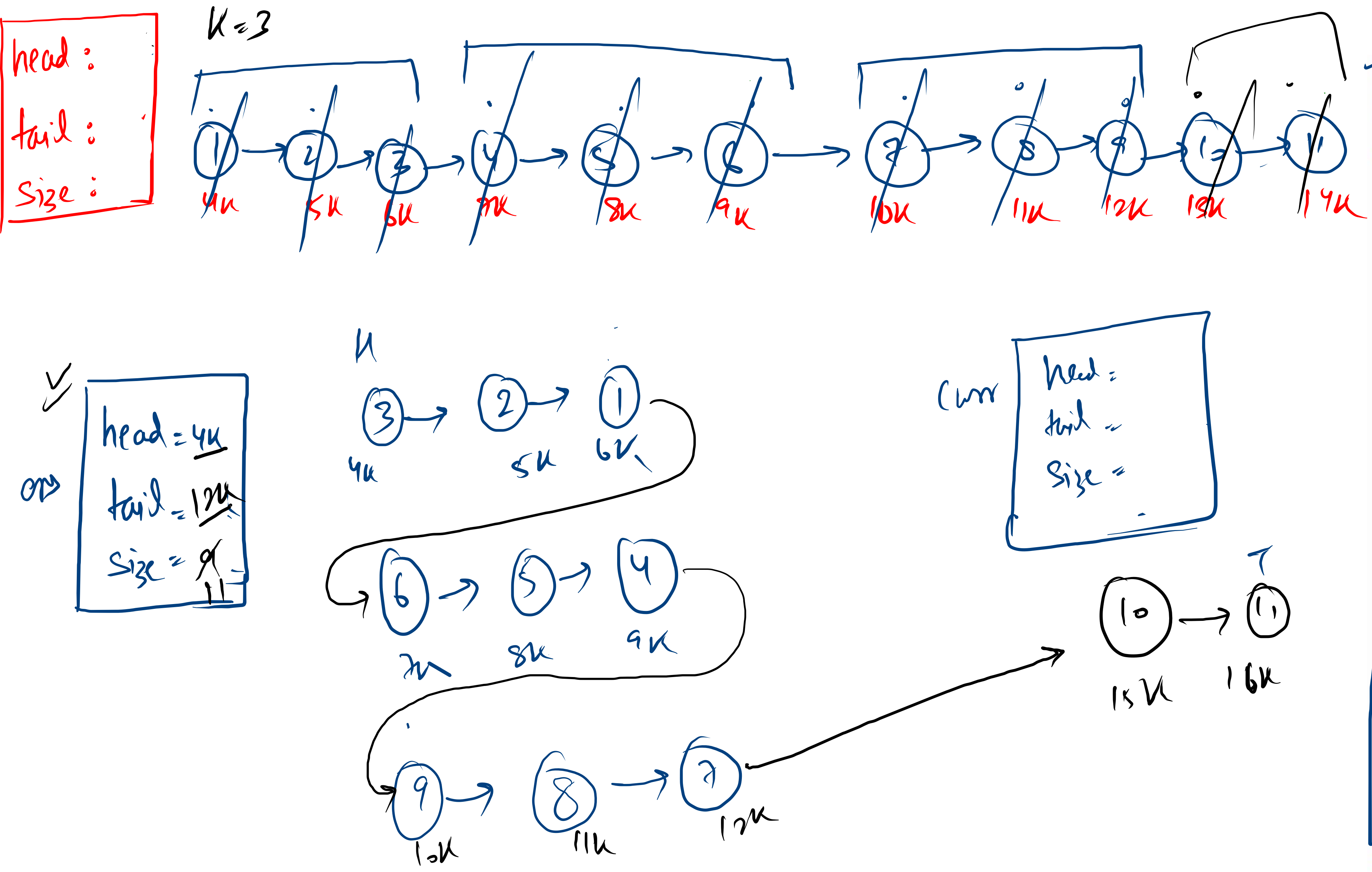
head →	9k
tail →	13k
size →	5

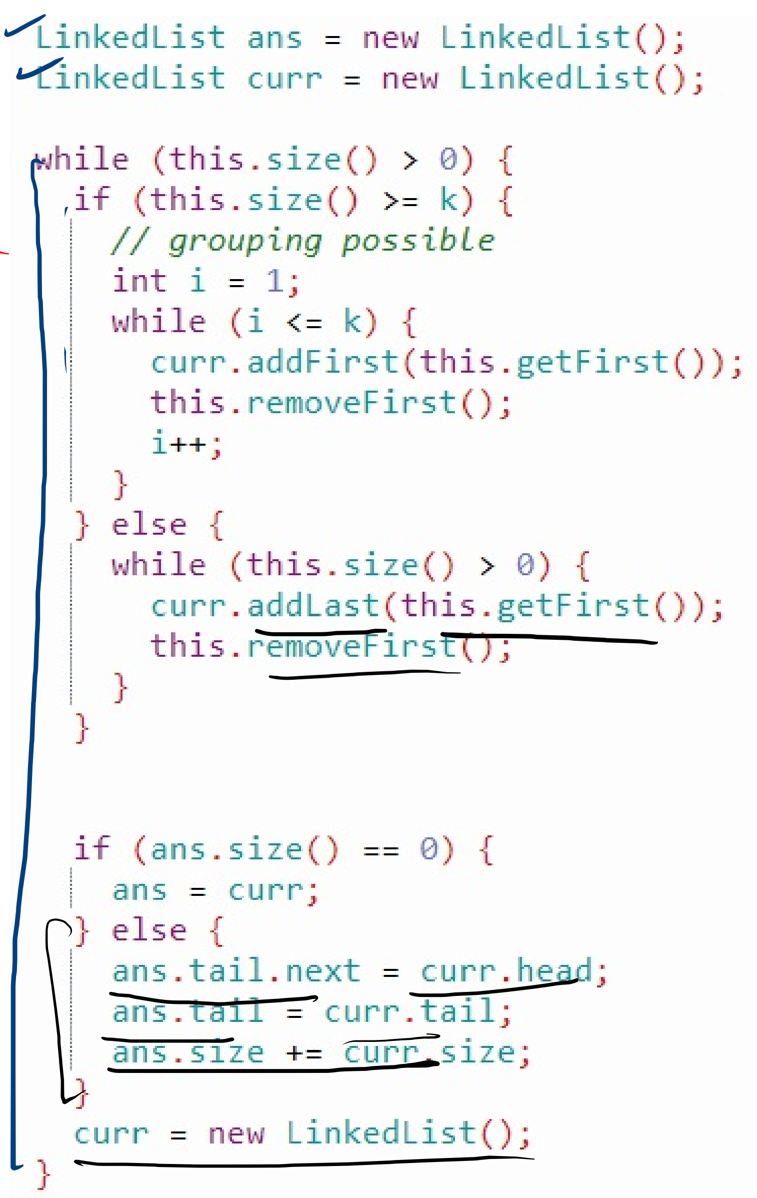


```
✓ odd.size() == 0
    this.head = Even.head
    this.tail = Even.tail
    this.size = Even.size

✓ even.size() == 0
    this.head = odd.head
    this.tail = odd.tail
    this.size = odd.size

odd.size != 0 && even.size != 0
    odd.tail.next = even.head
    this.head = odd.head
    this.tail = even.tail
    this.size => o.size + e.size
```



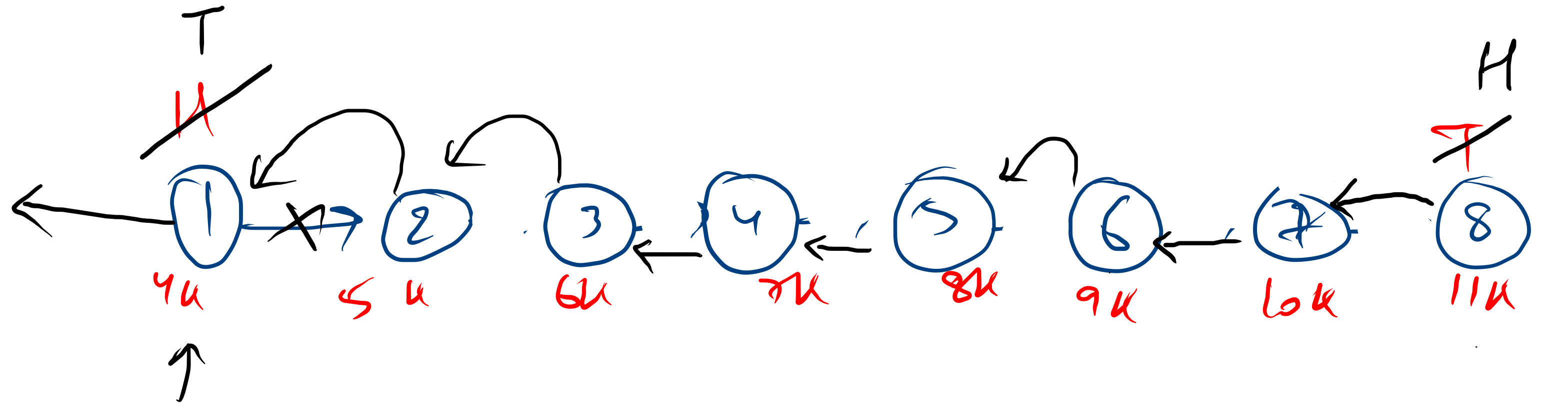


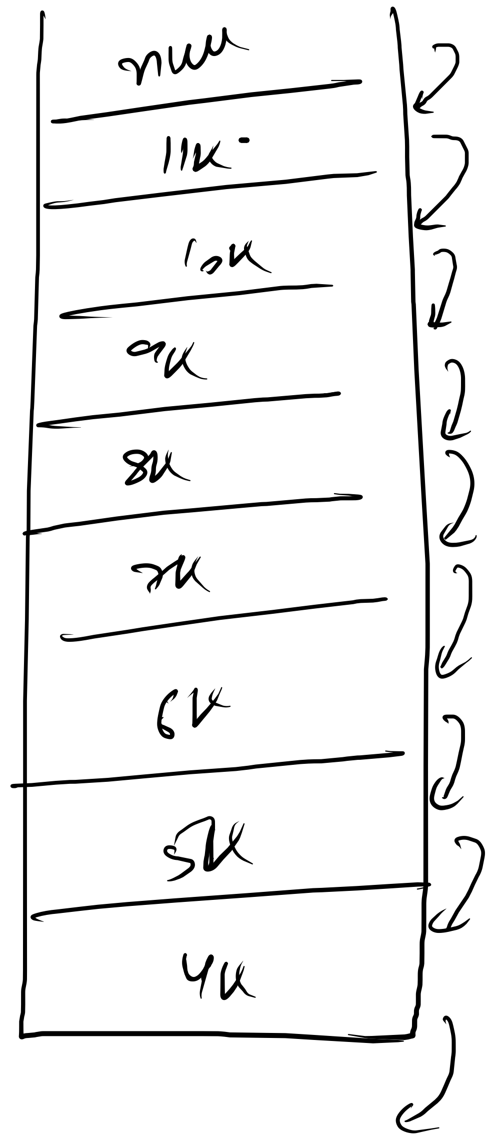
```java
LinkedList ans = new LinkedList();
LinkedList curr = new LinkedList();

while (this.size() > 0) {
  if (this.size() >= k) {
    // grouping possible
    int i = 1;
    while (i <= k) {
      curr.addFirst(this.getFirst());
      this.removeFirst();
      i++;
    }
  } else {
    while (this.size() > 0) {
      curr.addLast(this.getFirst());
      this.removeFirst();
    }
  }

  if (ans.size() == 0) {
    ans = curr;
  } else {
    ans.tail.next = curr.head;
    ans.tail = curr.tail;
    ans.size += curr.size;
  }
  curr = new LinkedList();
}
```

Display
reverse





node.next.next = node