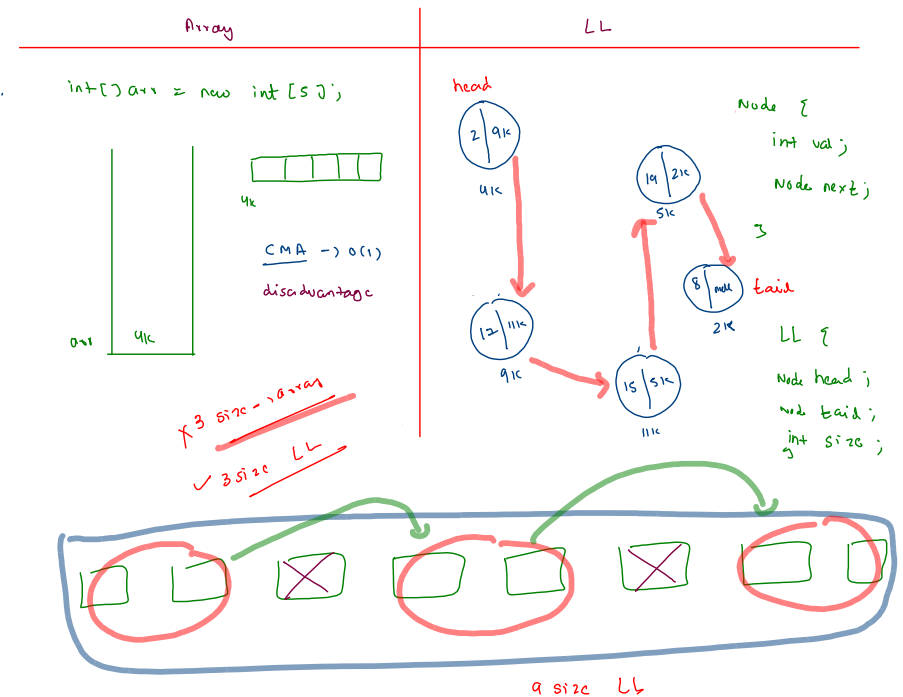

Array
LL
int[] arr = new int[5];
4k
CMA -> O(1)
disadvantage
arr
4k
head
2 | 9k
4k
17 | 11k
9k
15 | 5k
11k
19 | 2k
5k
8 | null
2k
tail
Node {
int val;
Node next;
}
LL {
Node head;
Node tail;
int size;
}
X 3 size - array
✓ 3 size LL
9 size LL

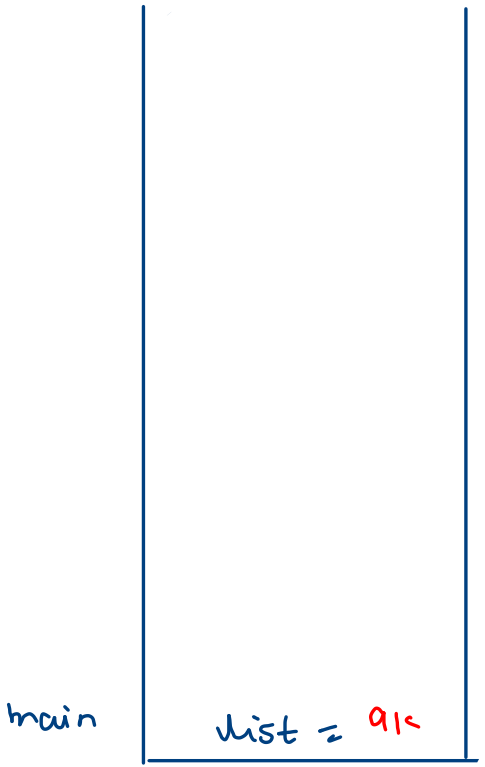
```java
public static class Node {
  int data;
  Node next;
}

public static class LinkedList {
  Node head;
  Node tail;
  int size;

  void addLast(int val) {
    // Write your code here
  }
}
```

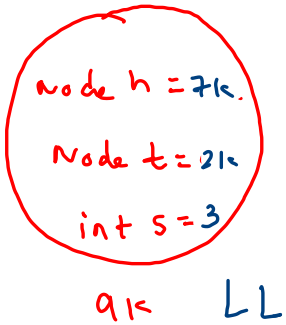
```java
LinkedList list = new LinkedList();
```

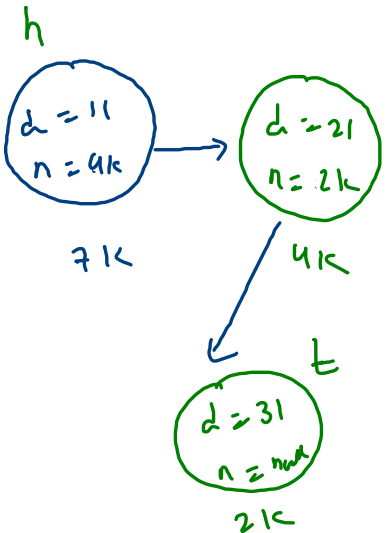
list.addLast(11);

list.addLast(21);

list.addLast(31);







```java
public static class LinkedList {
  Node head;
  Node tail;
  int size;

  void addLast(int val) {
    Node nn = new Node();
    nn.data = val;
    nn.next = null;

    if(size == 0) {
        //linked list is empty
        head = nn;
        tail = nn;
    }
    else {
        tail.next = nn;
        tail = nn;
    }
    size++;
  }
}
```

```java
LinkedList list = new LinkedList();
```

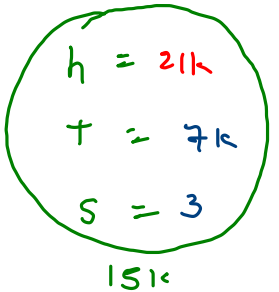
list.addLast(10);

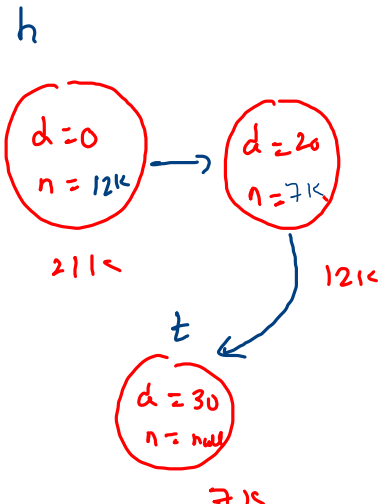
list.addLast(20);

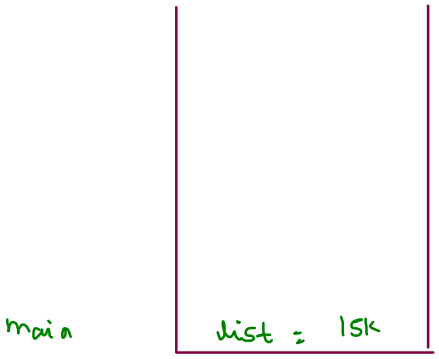
list.al(30);

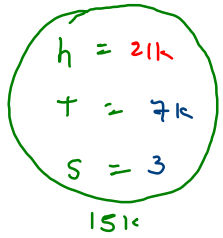
main

list = 15k

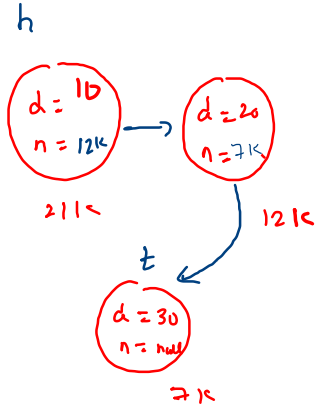








list = 15k

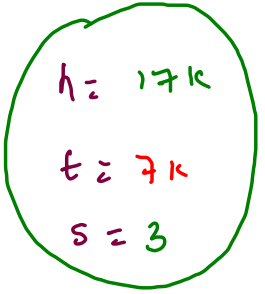


tmp = null

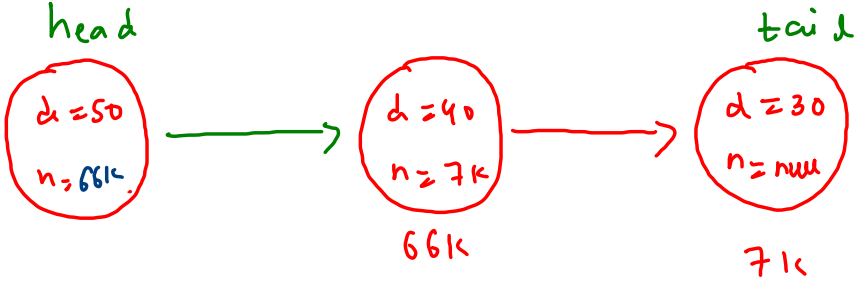
```
display ( LinkedList list ) {
    Node temp = list.head;
    while ( temp != null ) {
        syso ( temp.data );
        temp = temp.next;
    }
}
```

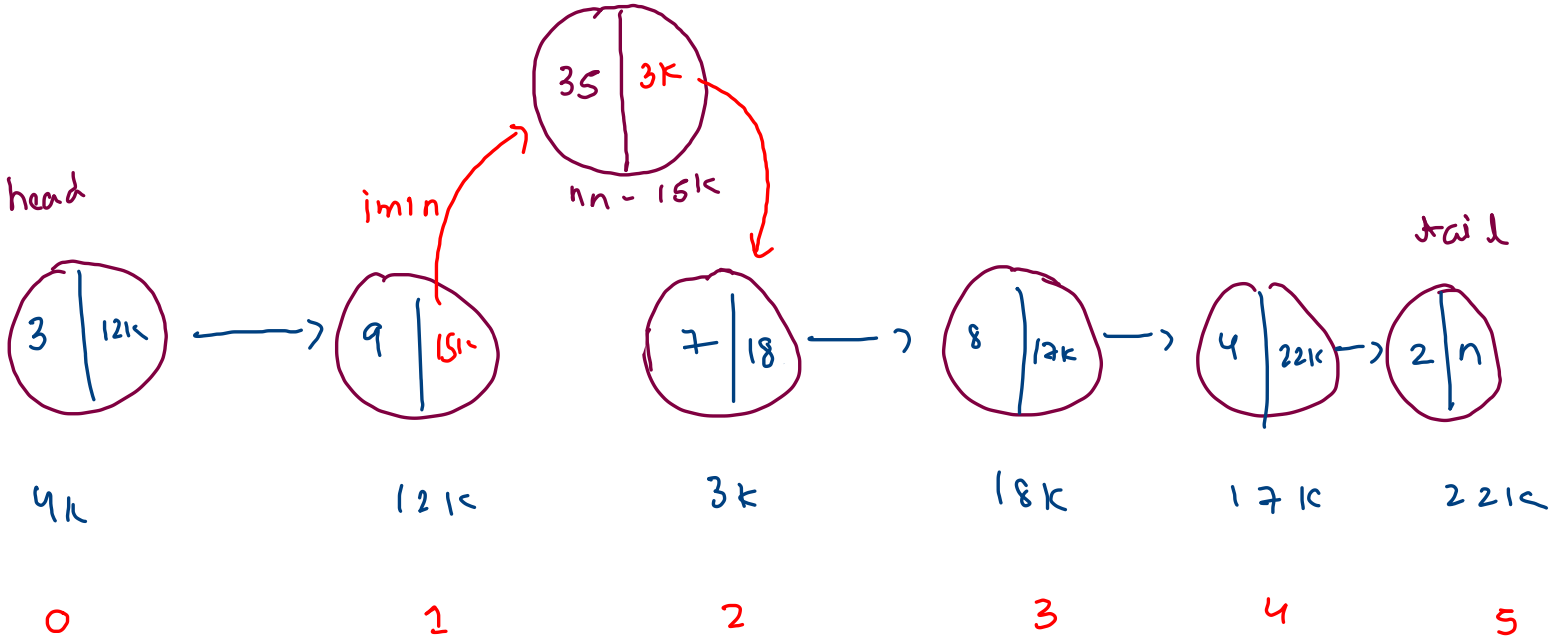
10 20 30 →

```
list.addF(30);
list.addF(40);
list.addF(50);
```



list = 21K





idx == 0 (AF)

idx == size (AL)

idx > 0 && idx < size (SW)

idx < 0 || idx > size (Invalid arg

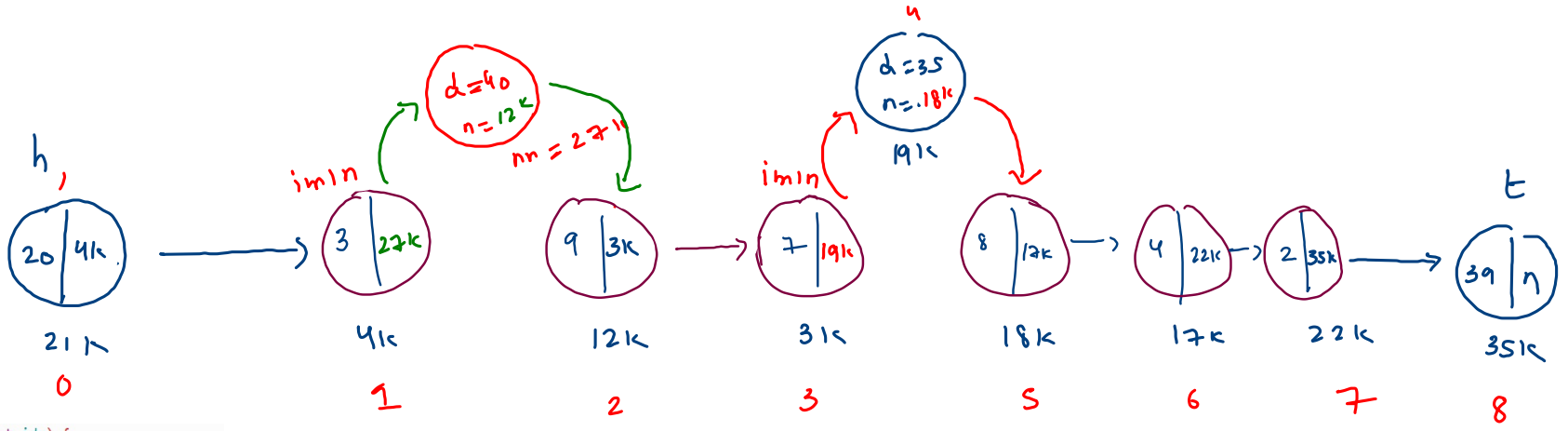
list.addAt(2, 35);

idx, val

head = 4K

tail = 22K

size = 6

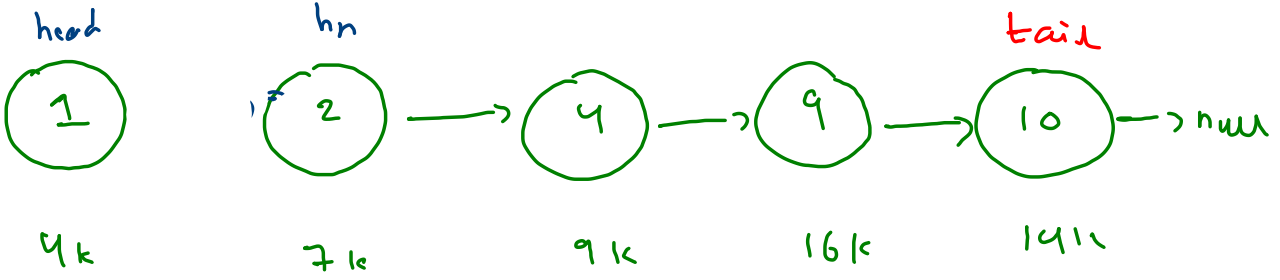



```
private Node getNodeAt(int idx) {
    Node temp = this.head;

    for(int i=0; i < idx;i++) {
        temp  = temp.next;
    }

    return temp;

}

public void addAt(int idx, int val){
  if(idx == 0) {
      addFirst(val);
  }
  else if(idx == size) {
      addLast(val);
  }
  else if(idx < 0 || idx > size) {
      System.out.println("Invalid arguments");
  }
  else {
      //idx -> 1 to size-1
      Node nn = new Node();
      nn.data = val;
      nn.next = null;

      Node im1n = getNodeAt(idx-1); //idx-1 node
      nn.next = im1n.next;
      im1n.next = nn;

      size++;
  }
}
```

1. ll.addAt(3,35);
2. ll.addAt(0,20);
3. ll.addAt(8,39);
4. ll.addAt(2,40);

idx = 2

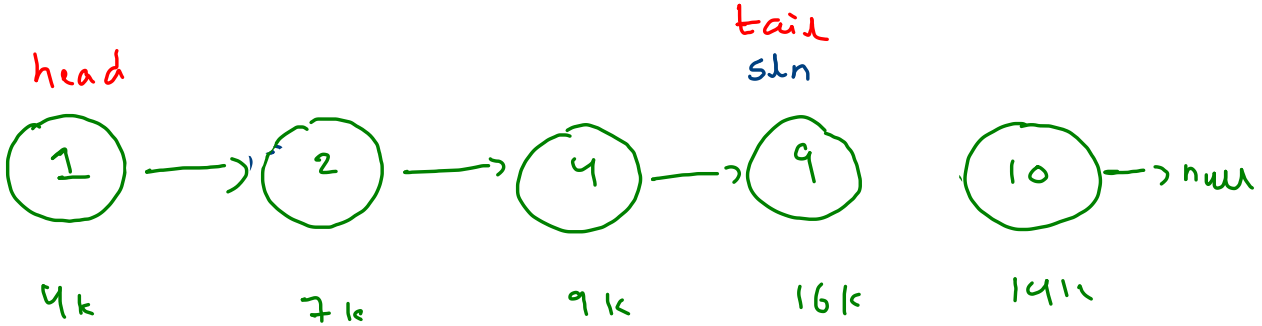
h = 7k
t = 14k
s = 5




```
if(size == 0) {
    // no work;
}
else if( size == 1 ) {
    head = tail = null;
}
else
```

```
hn = head.next
head.next = null
head = hn
```

h = 4k
t = 16k
s = 5



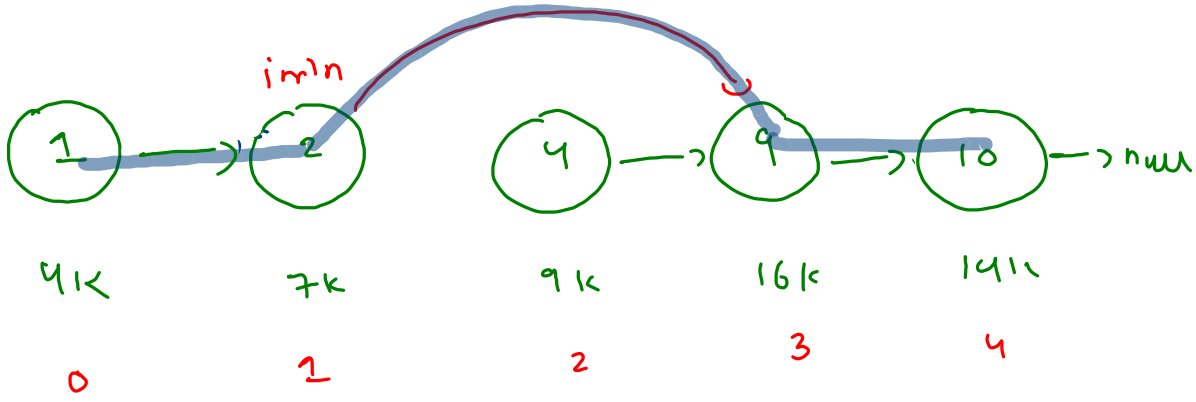

```
if(size == 0) {
    // no work
}
else if(size == 1) {
    head = tail = null;
}
else
```

last node -> size-1

second last node

posn -> size-2

```
sln.next = null
tail = sln
```



idx == 0 (8F)

idx == (size-1) 8L

idx = 2

idx >= 1 && idx <= size-2

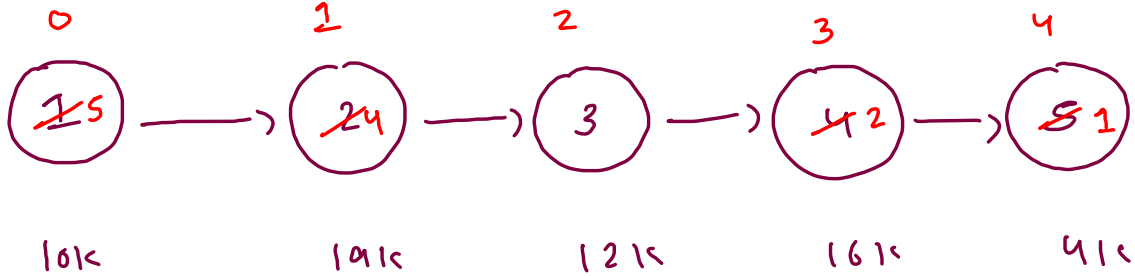
```
imin.next = imin.next.next
```

reverse data

iterative

h = 101
t = 41
s = 5

41 (151)



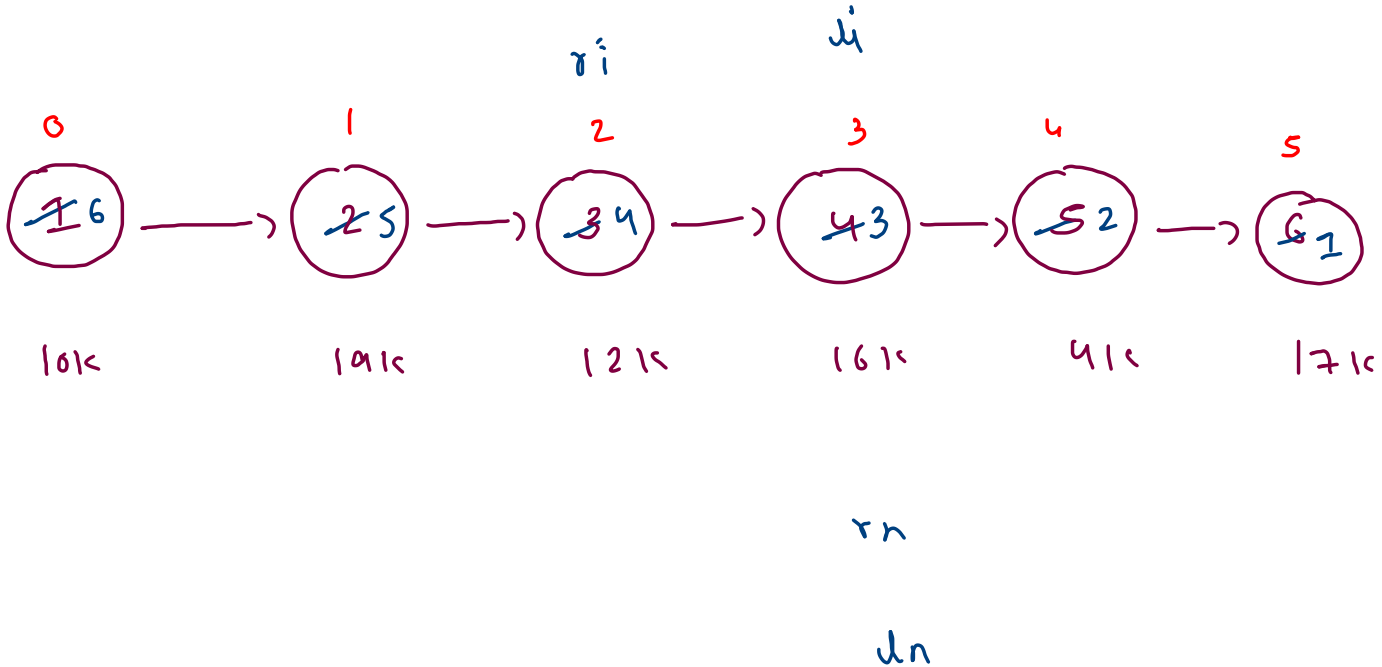
idx = 2

Space → O(1)

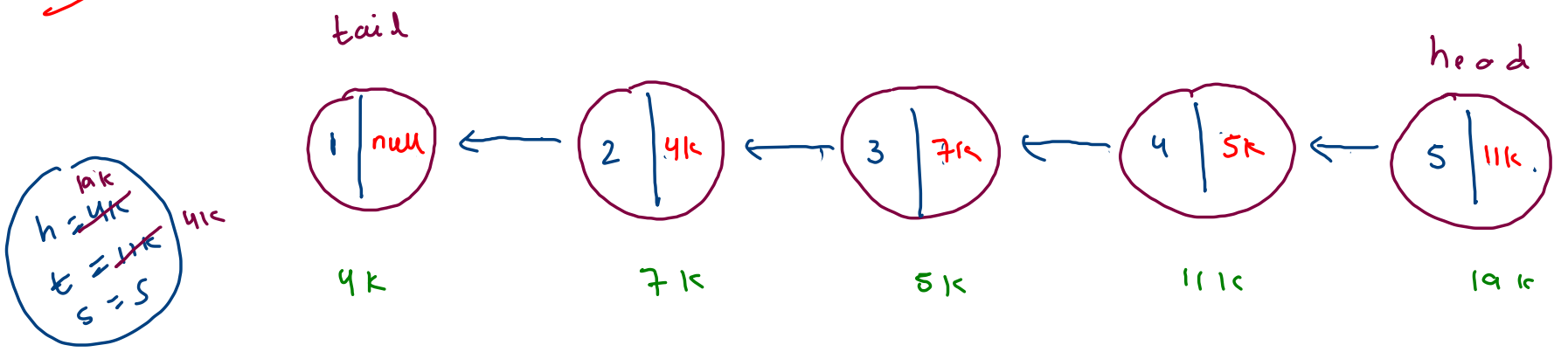
do

hi

```java
public void reverseDI() {
    int li = 0;
    int ri = size-1;

    Node ln = head;

    while(li < ri) {
        Node rn = getNodeAt(ri);

        //swap data of left node and right node
        int temp = ln.data;
        ln.data = rn.data;
        rn.data = temp;

        ln = ln.next;
        li++;
        ri--;
    }
}
```



~~reverse~~ pointer iterative

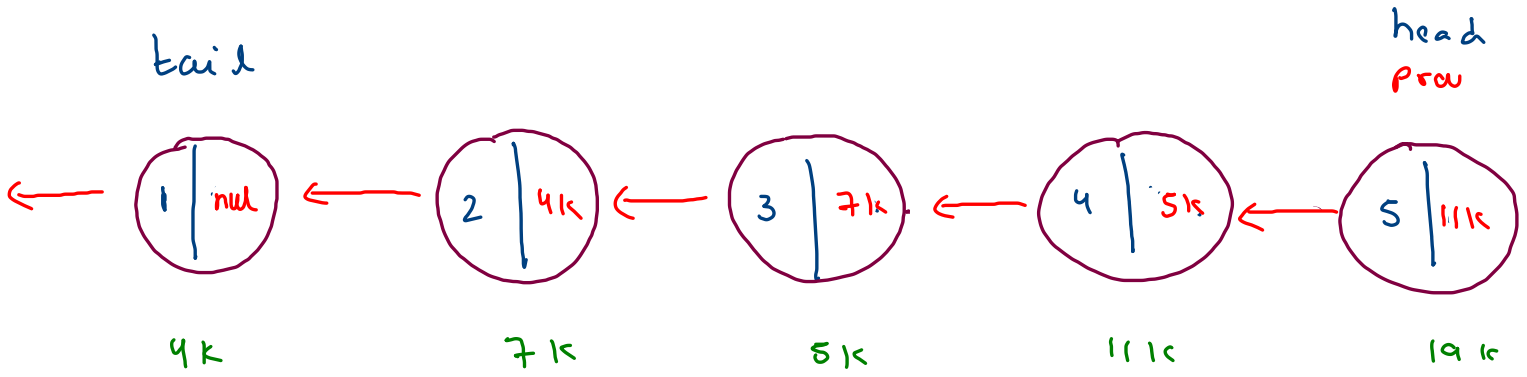


n = c.next

c.next = p

space - O(1)

time - O(n)




```java
public void reversePI(){
  // write your code here
  Node prev = null;
  Node curr = head;

  while(curr != null) {
      //backup
      Node next = curr.next;
      //reverse link
      curr.next = prev;
      //move
      prev = curr;
      curr = next;
  }

  //swap head,tail
  Node temp = head;
  head = tail;
  tail = temp;
}
```

space -> O(1)

time -> O(n)

h = ~~4k~~ 19k

t = ~~19k~~ 4k

5 = 5