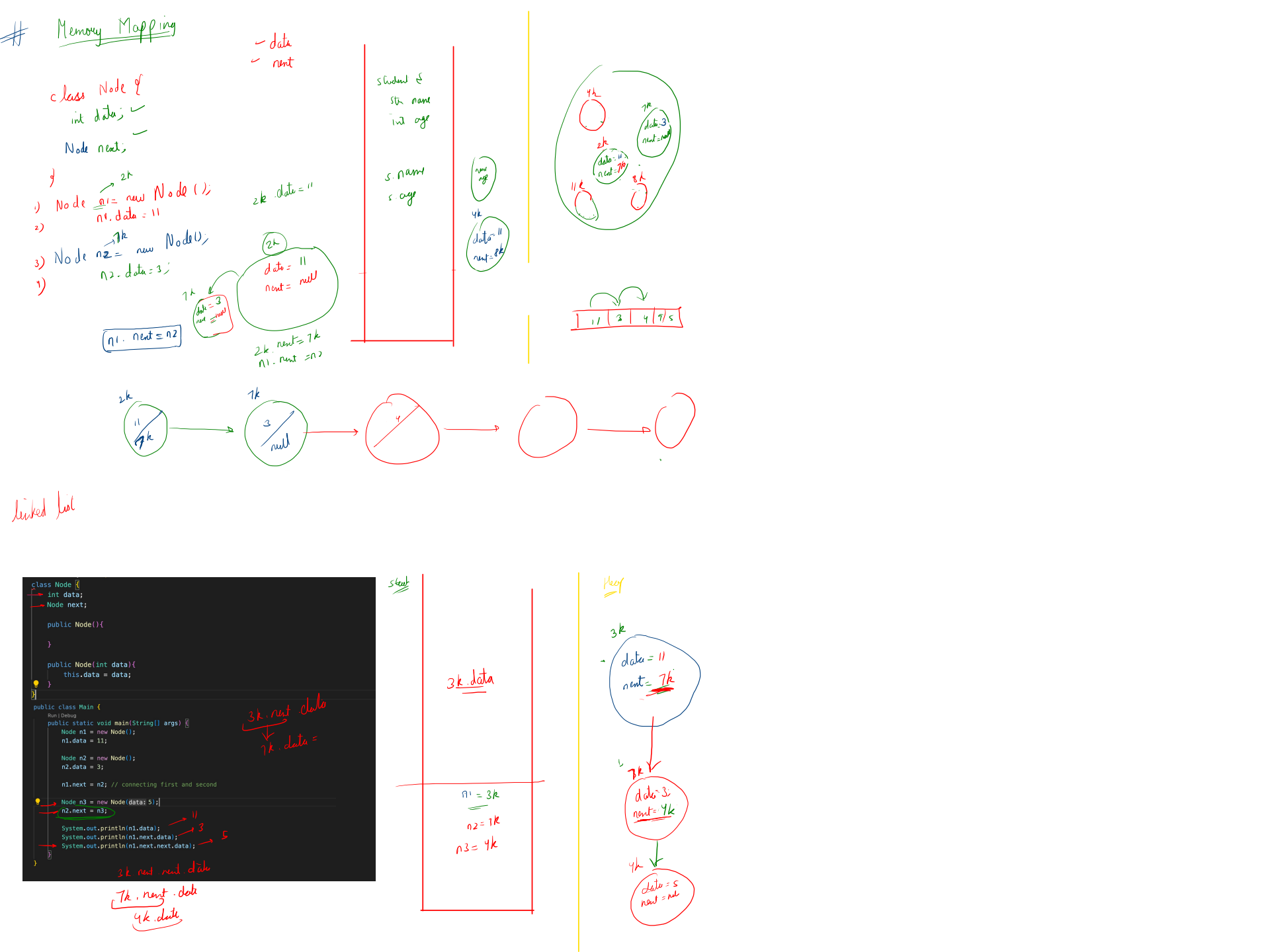
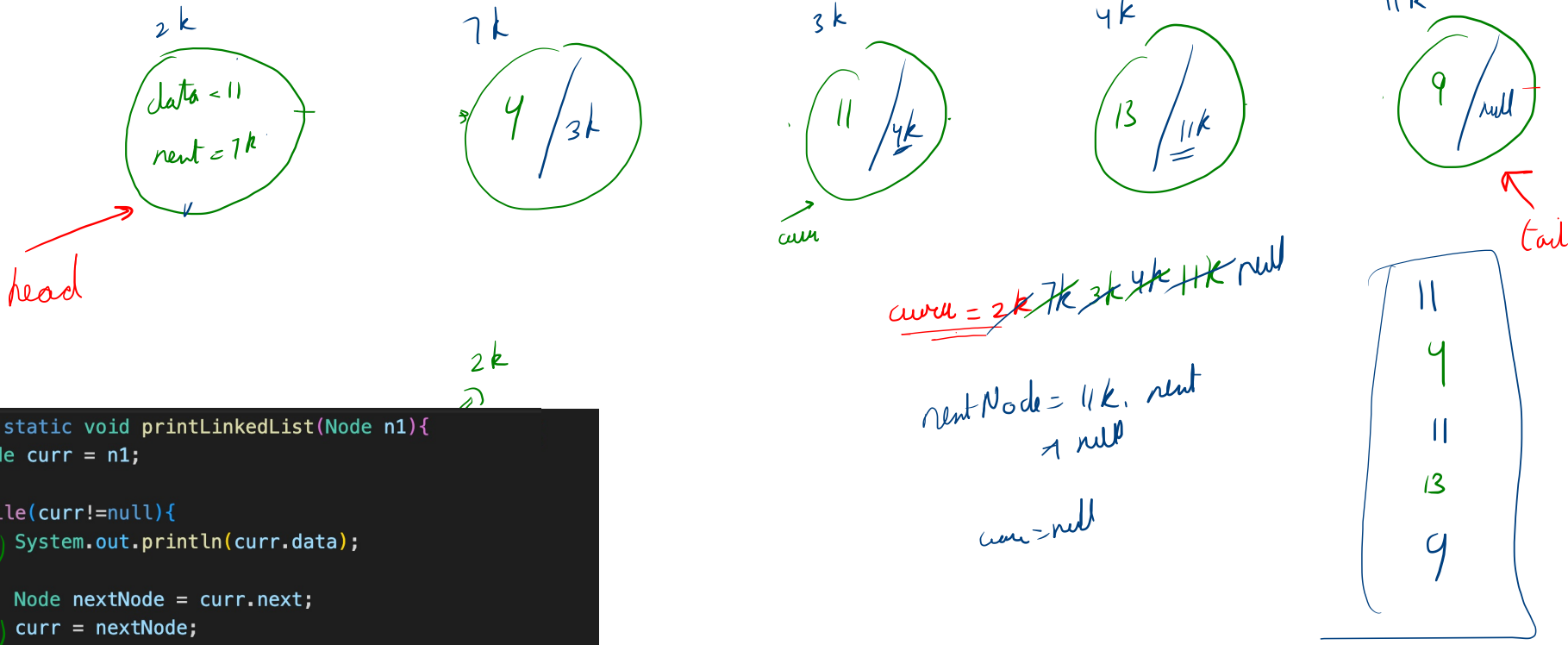
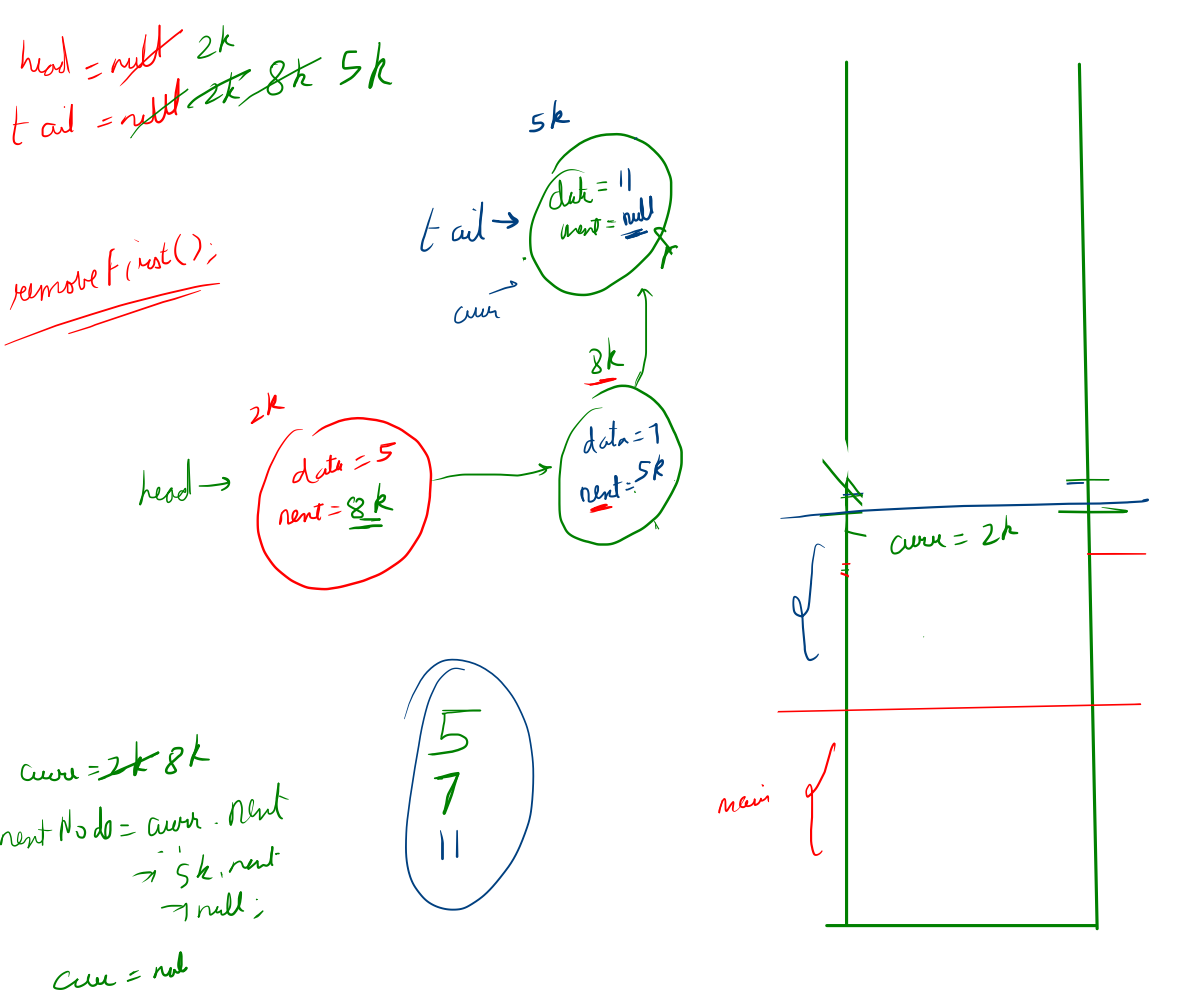
Memory Mapping




```java
class Node {
    int data;
    Node next;

    public Node(){

    }

    public Node(int data){
        this.data = data;
    }
}

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Node n1 = new Node();
        n1.data = 11;

        Node n2 = new Node();
        n2.data = 3;

        n1.next = n2; // connecting first and second

        Node n3 = new Node(data: 5);
        n2.next = n3;

        System.out.println(n1.data);
        System.out.println(n1.next.data);
        System.out.println(n1.next.next.data);
    }
}
```

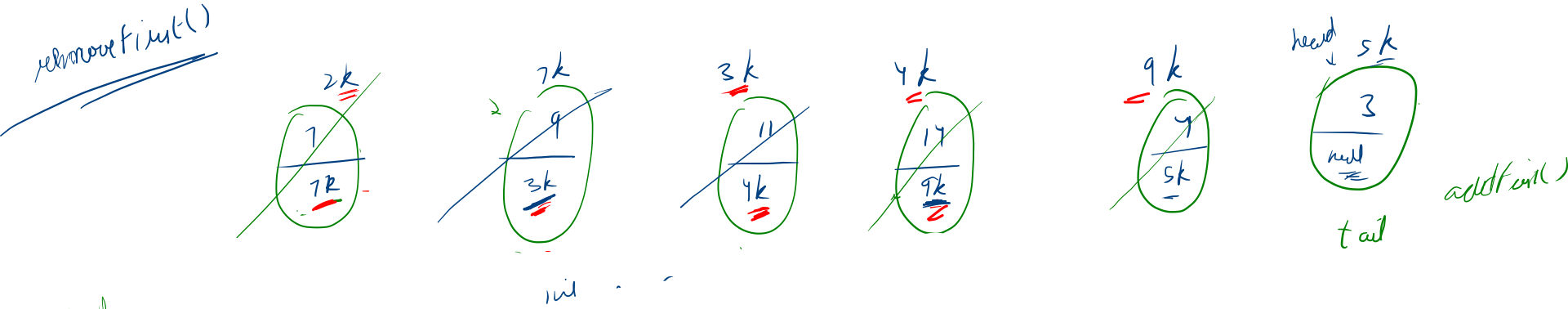



```java
public static void printLinkedList(Node n1){
    Node curr = n1;

    while(curr!=null){
        System.out.println(curr.data);

        Node nextNode = curr.next;
        curr = nextNode;
    }
}
```




```java
public class Main {
    static Node head;
    static Node tail;

    public static void addLast(int value){
        Node nn = new Node(value);

        if(head == null){
            head = nn;
            tail = nn;
        } else {
            tail.next = nn;
            tail = nn;
        }
    }

    public static void printLinkedList(Node n1){
        Node curr = n1;

        while(curr!=null){
            System.out.println(curr.data);

            Node nextNode = curr.next;
            curr = nextNode;
        }
    }
    Run | Debug
    public static void main(String[] args) {
        head = null;
        tail = null;

        addLast(value: 5);
        addLast(value: 7);
    }
}
```

(adding node at last)
(moving tail)
head
print (head)
addLast (11)




```java
public static void removeFirst(){
    if(head==null){ // 0 nodes in LL
        System.out.println(x: "LinkedList is empty, not possible to remove first");
    } else if(head == tail){ // only one node in linkedlist
        head = null;
        tail = null;
    } else {
        Node nextOfHead = head.next;
        head = nextOfHead;
    }
}
```

2. addFirst() → add in front

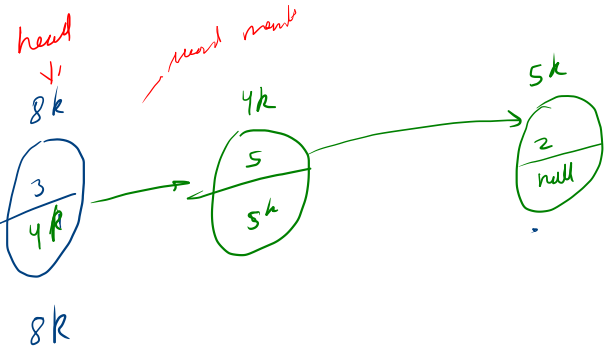
1. create linked list with user input

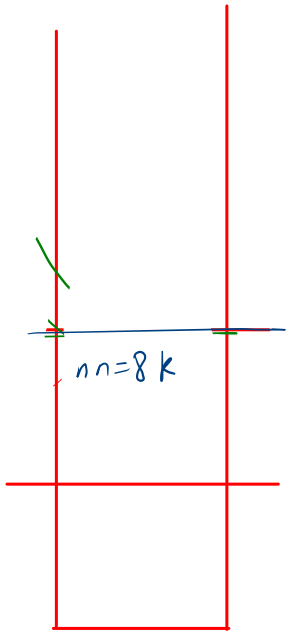
addFirst()

head = null 5k 4k 8k
tail = null 5k

addFirst(2)
addFirst(5)
addFirst(3)

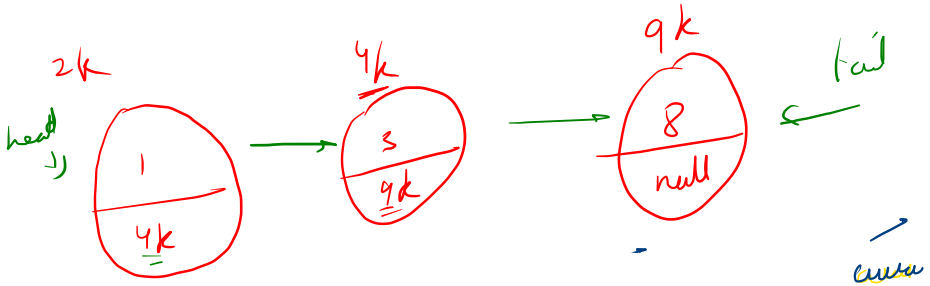
5 2






```java
public static void addFirst(int value){
    Node nn = new Node(value);

    if(head == null){
        head = nn;
        tail = nn;
    } else {
        nn.next = head;
        head = nn;
    }
}
```

size()



ans = 0 1 2 3

curr = 2k 4k 9k null

noC = 4k

curr = curr.next

noC = 9k null

```
public static int size(){
    int ans = 0;
    Node curr = head;

    while(curr!=null){
        ans++;
        Node nextOfCurr = curr.next;
        curr = nextOfCurr;
    }

    return ans;
}
```

length = 5

getAt (3)

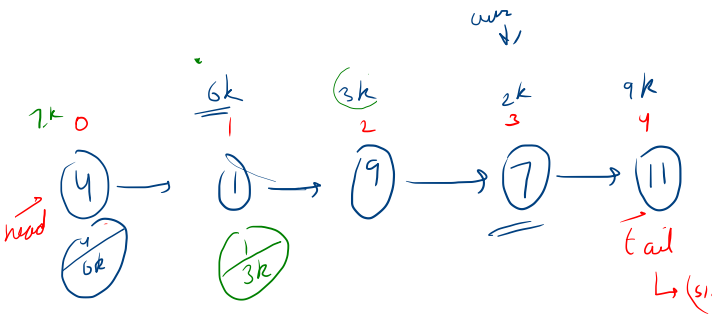
curr = 7k 6k 3k 2k

node = 6k 3k 2k

i = 0 1 2 3 i < 3

```
public static Node getAt(int idx){
    int length = size();
    if(idx < 0 || idx >= length){
        System.out.println(x: "No such index exists!!");
        return null;
    }

    Node curr = head;
    for(int i=0; i<idx; i++){
        Node nextOfCurr = curr.next;
        curr = nextOfCurr;
    }

    return curr;
}
```

getAt (idx):



Node

idx
↳ curr will move idx times

getAt (2) => 3k

getAt (0) => 7k

getAt (5) → no such element
return null;

getAt (-1) → return null;

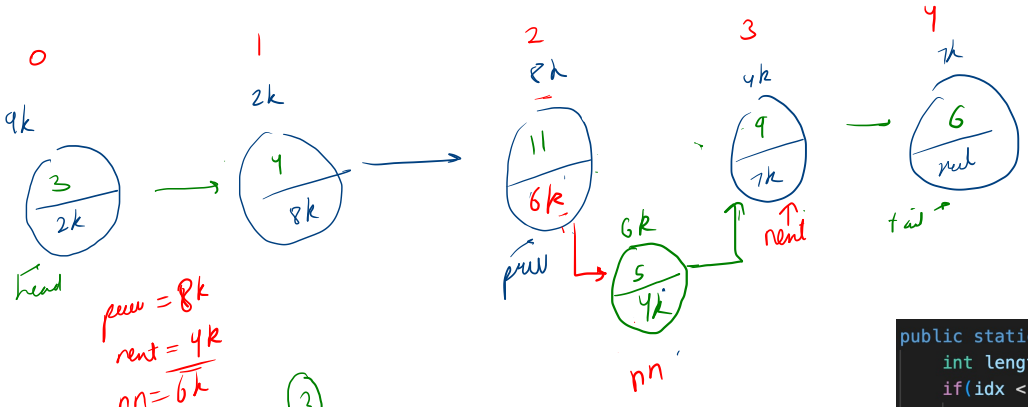
idx = 0 ↳ curr will move 0 times

idx = 1 ↳ curr will move 1 time

idx = 2 ↳ curr will move 2 time

idx = 4 ↳ curr will move 4 times

addAt (idx, value)



addAt (0, 7)
addAt (5, 8)
addAt (3, 5)
addAt (2, 7)
addAt (9, 11)

prev = 8k
next = 4k
nn = 6k

3

Ram to hold Noah
Noah needs to hold shyam

.next

prev Node at (2nd)

prev.next = nn

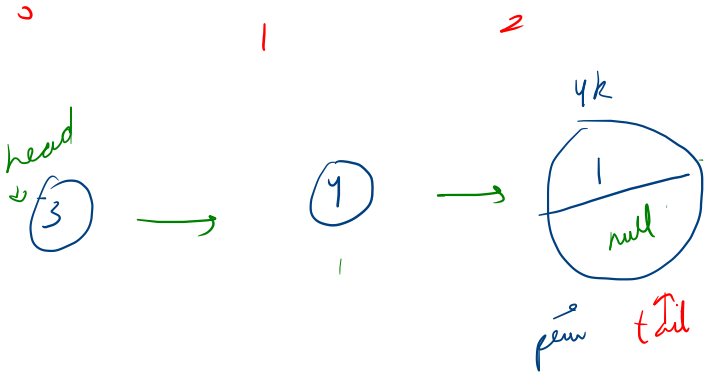
8k.next = 6k

6k.next = 4k

```java
public static void addAt(int idx, int value){
    int length = size();
    if(idx < 0 || idx > length){
        System.out.println(x: "No such index exists");
        return;
    }

    if(idx==0){
        addFirst(value);
    } else if(idx == length){
        addLast(value);
    } else {
        Node nn = new Node(value);

        Node prev = getAt(idx-1);
        Node next = prev.next;

        prev.next = nn;
        nn.next = next;
    }
}
```

prev attach new node
new node attach next

removeLast()



length = 4

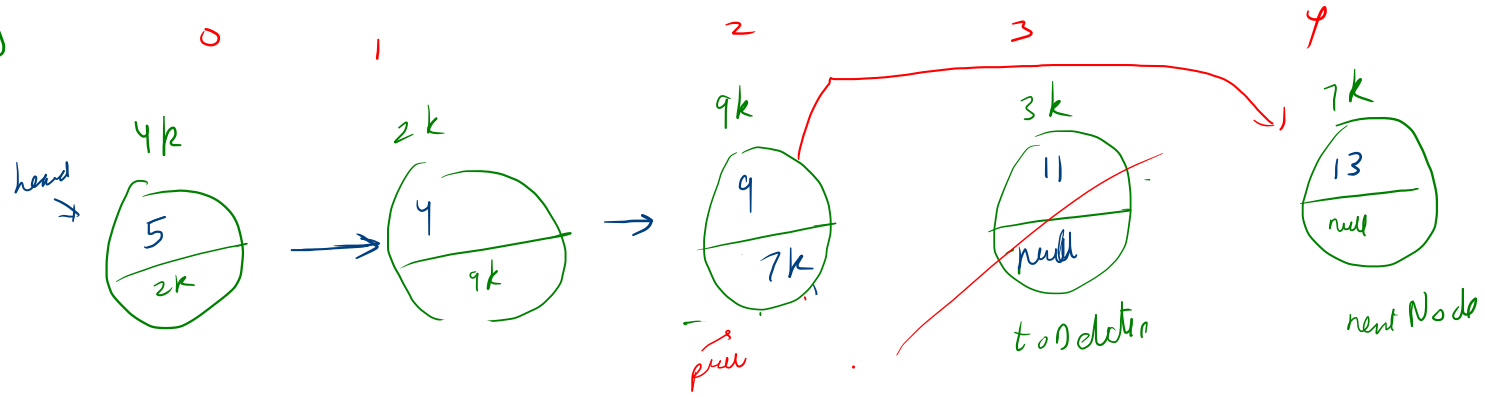
prev

prev = 4k

4k.next = null

```
public static void removeLast(){
    if(head==null){
        System.out.println(x: "LinkedList is empty, not possible to remove last");
    } else if(head == tail){
        head = null;
        tail = null;
    } else {
        int length = size();
        Node prev = getAt(length - 2);

        prev.next = null;
        tail = prev;
    }
}
```

removeAt(idx)



prev = 9k

removeAt(3)

prev = 9k =

toDelete = 3k

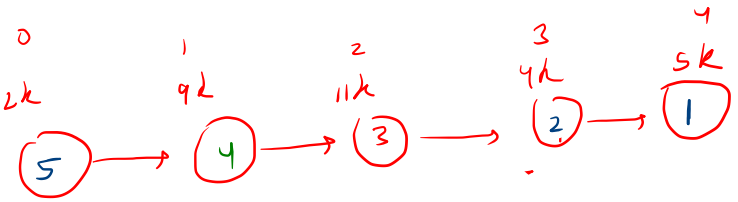
nextNode = 7k

-1

89

```java
public static void removeAt(int idx){
    int length = size();
    if(idx < 0 || idx>=length){
        System.out.println(x: "There is no such index");
        return;
    }

    if(idx == 0){
        removeFirst();
    } else if(idx == length - 1){
        removeLast();
    } else {
        Node prev = getAt(idx-1);
        // prev.next = prev.next.next;

        Node toDelete = prev.next;
        Node nextNode = toDelete.next;

        prev.next = null;
        toDelete.next = null;
        prev.next = nextNode;
    }
}
```

reverse Linked List() → how to solve in array



i = 0

j = 4

data

NodeAt i = 2k 9k

NodeAt j = 5k 4k

ValueAt i = 2

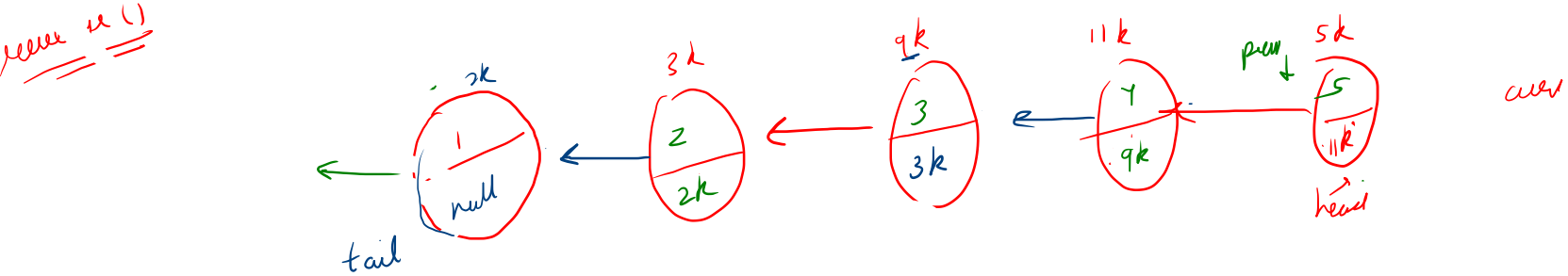
ValueAt j = 5 4

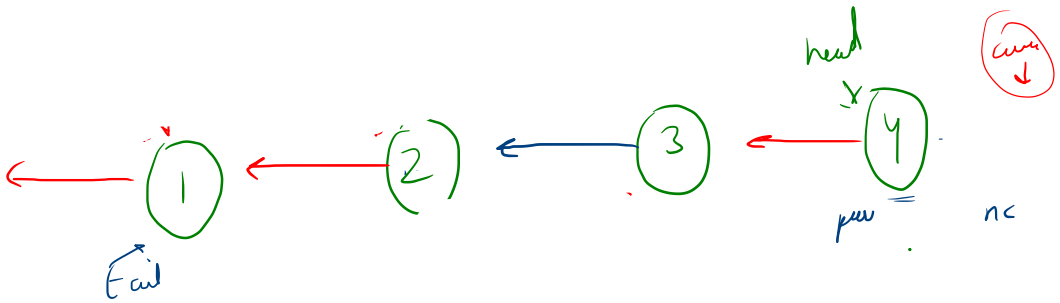
O(n²)

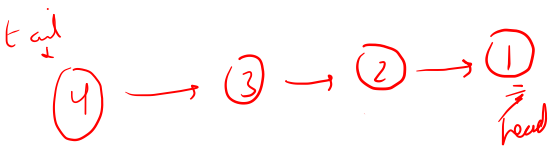
```
public static void reverse(){
    int length = size();

    int i = 0;
    int j = length - 1;

    while(i<j){
        Node nodeAti = getAt(i);
        Node nodeAtj = getAt(j);

        int valueAti = nodeAti.data;
        int valueAtj = nodeAtj.data;

        nodeAti.data = valueAtj;
        nodeAtj.data = valueAti;

        i++;
        j--;
    }
}
public static void removeAt(int idx){
```

O(n)

O(n)








```java
public static void reversePointerIteratively(){
    Node prev = null;
    Node curr = head;

    while(curr!=null){
        // save next
        Node nextOfCurr = curr.next;

        // reverse connection
        curr.next = prev;

        // move forward
        prev = curr;
        curr = nextOfCurr;
    }

    // shift head and tail
    tail = head;
    head = prev;
}
```