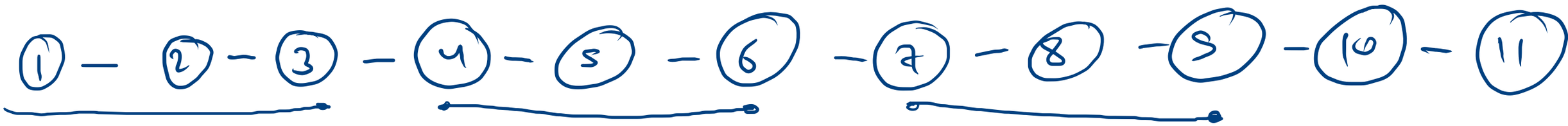
head - 1
tail - 11
size - 11

1 2 3 4 5 6 7 8 9 10 11

time O(n)
space O(1)

k = 3



head 3
tail 11
size 11

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11

k = 3

11

remove first

add first

remove last

add Last

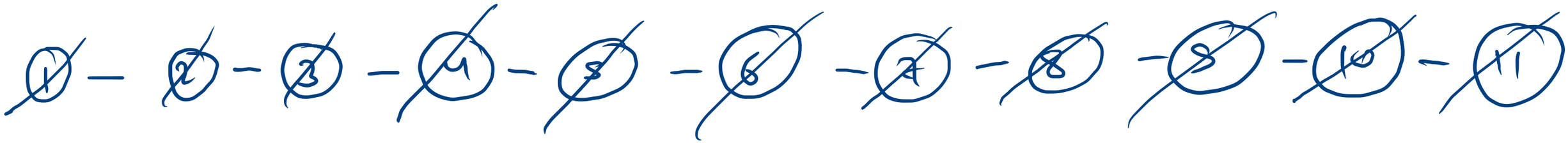
3 - 2 - 1

6 - 5 - 4

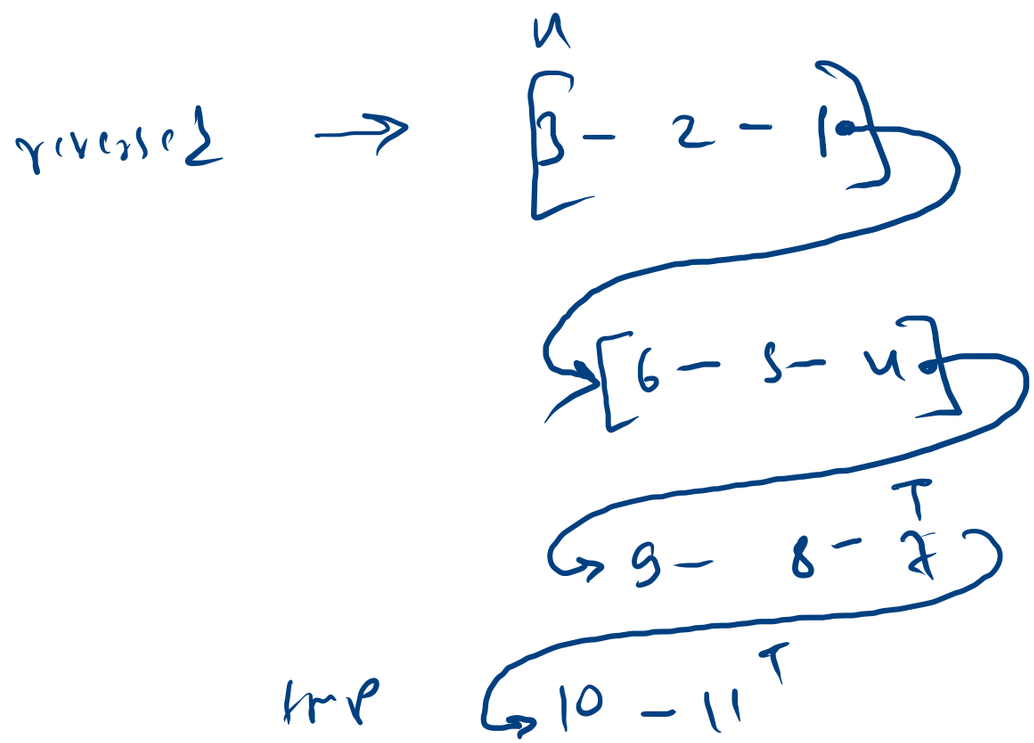
9 - 8 - 7

10 - 11

★	Input mutate
★	Input not change



k=3

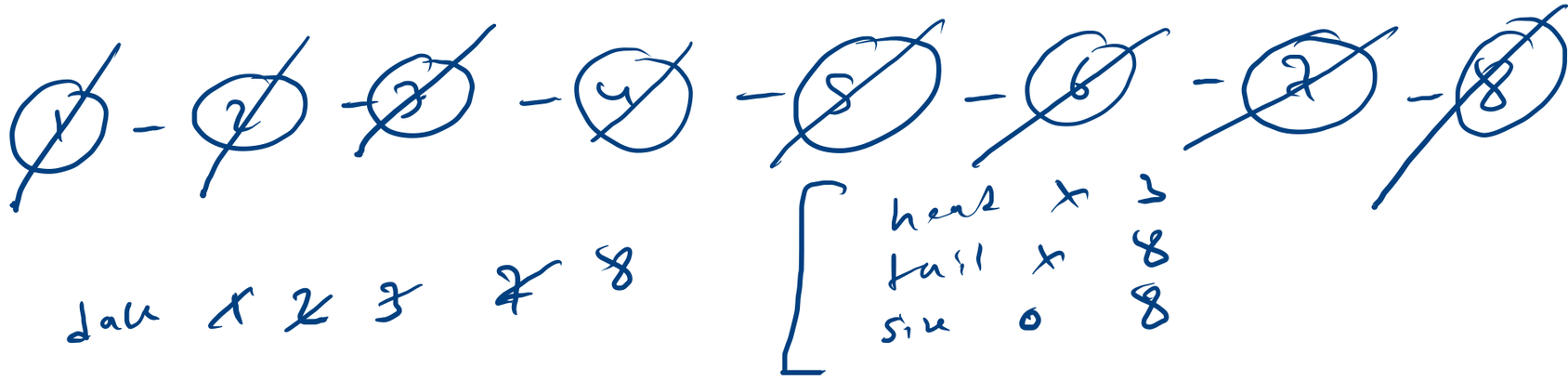


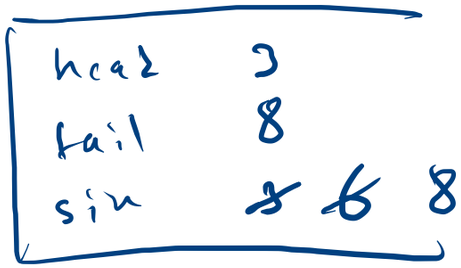

```
if ( reverse == null )
    reverse = tmp
else {
    R.tail.next = tmp.head;
    R.tail = tmp.tail
    R.size += 3;
}
```

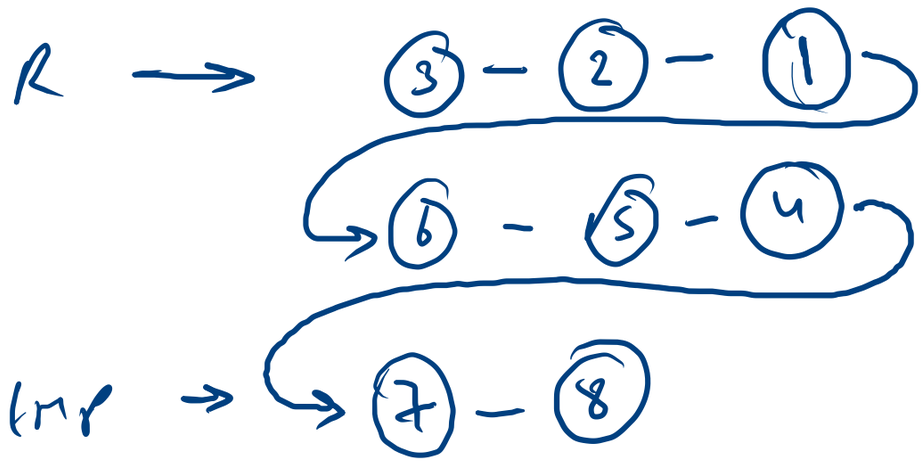
R → null

k = 3

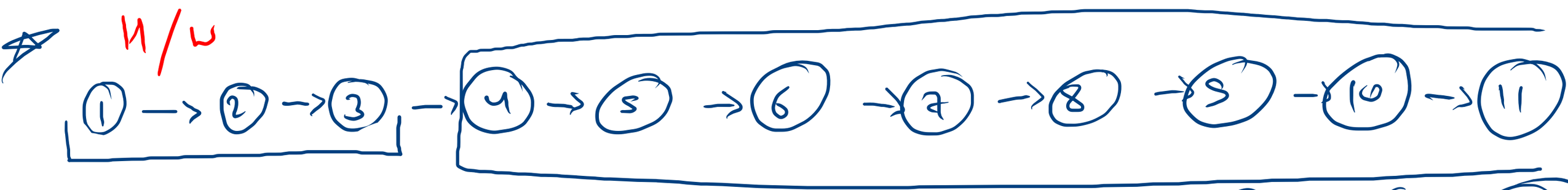
```
LinkedList R = null;

while(this.size() > 0){
    LinkedList tmp = new LinkedList();
    if(this.size() >= k){
        for(int i=0;i<k;i++){
            int data = this.getFirst();
            this.removeFirst();
            tmp.addFirst(data);
        }
    }else{
        while(this.size() > 0){
            int data = this.getFirst();
            this.removeFirst();
            tmp.addLast(data);
        }
    }

    if(R == null){
        R = tmp;
    }else{
        R.size += tmp.size();
        R.tail.next = tmp.head;
        R.tail = tmp.tail;
    }
}

this.head = R.head;
tail = R.tail;
size = R.size;
```

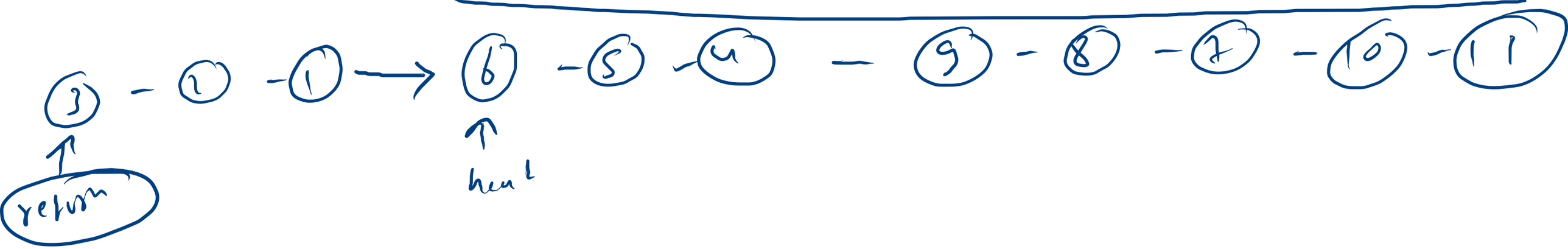






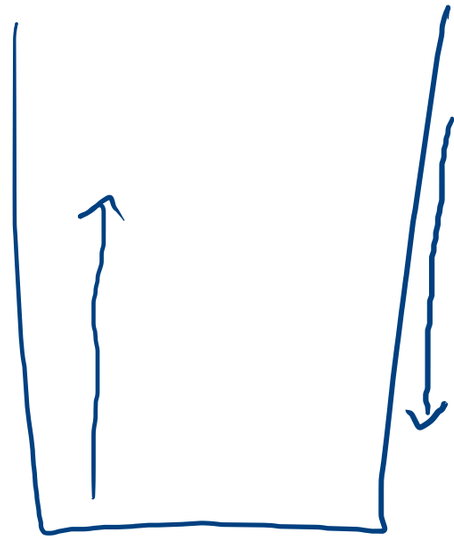
H/W





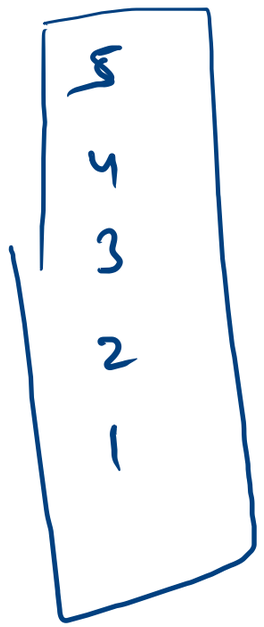
Node head = Path (k , y)

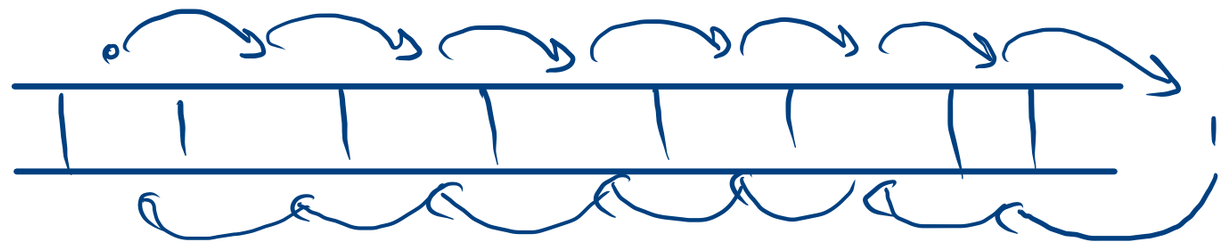
idea

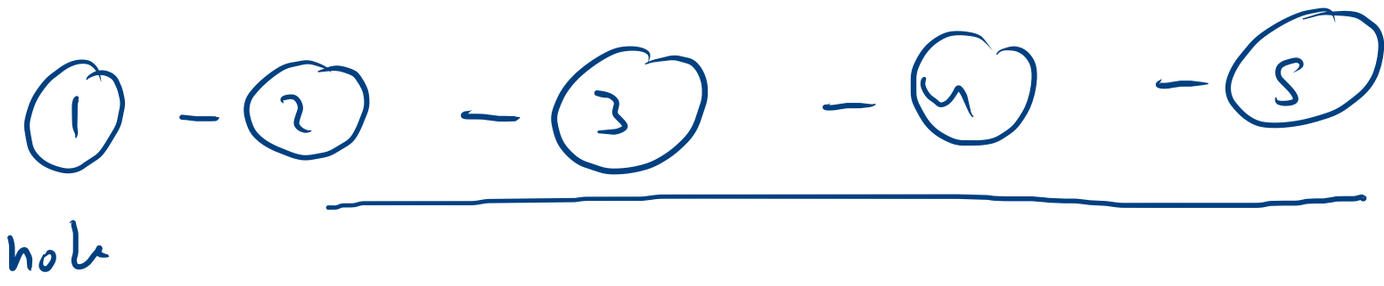


recursion

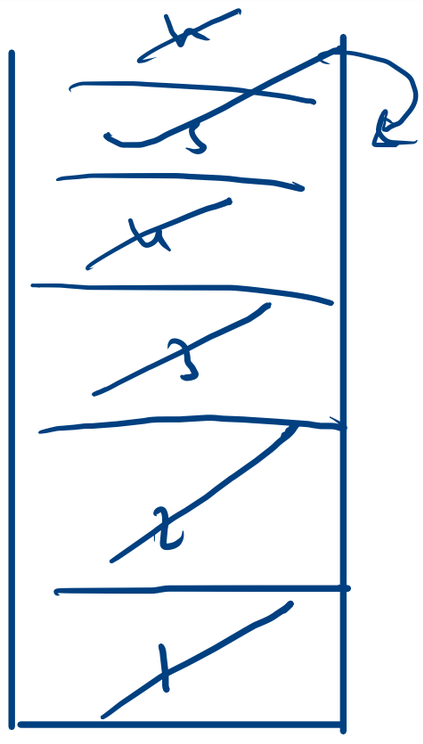
output →



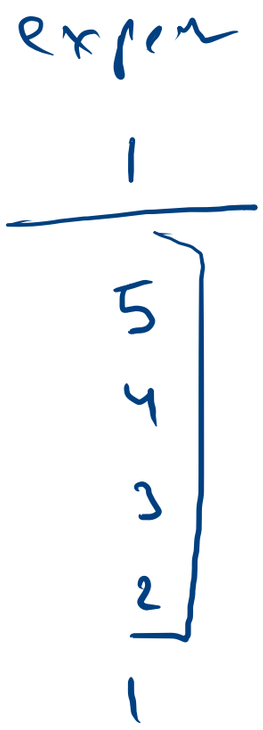


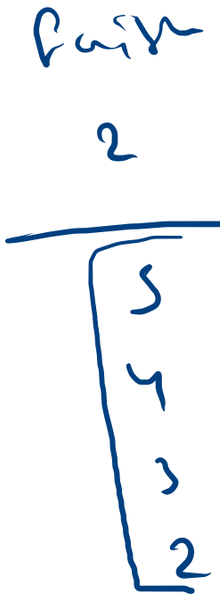


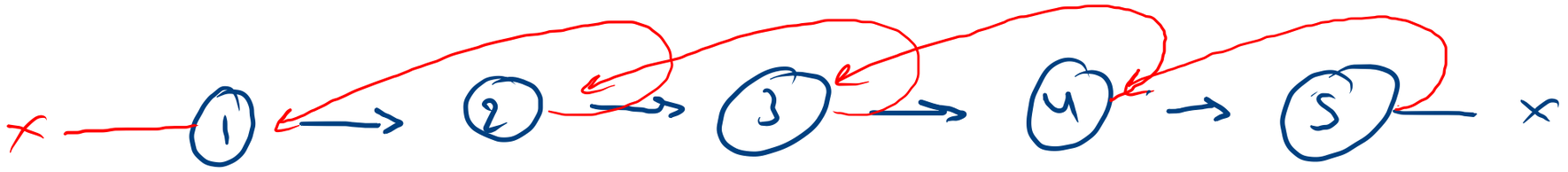

```java
private void displayReverseHelper(Node node){
    if(node == null)return;

    displayReverseHelper(node.next);
    print(node.data)
}
```



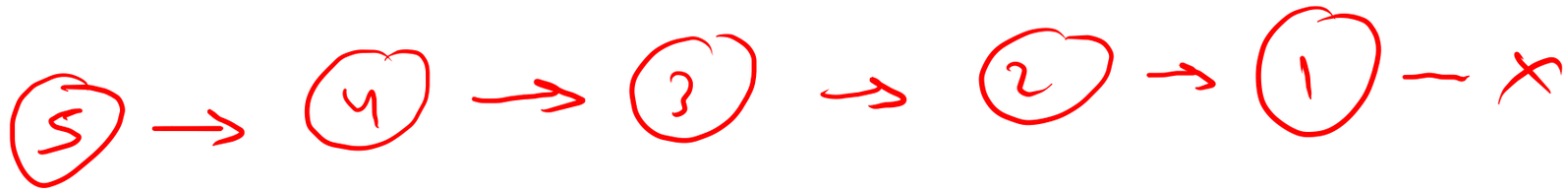
5
4
3
2
1





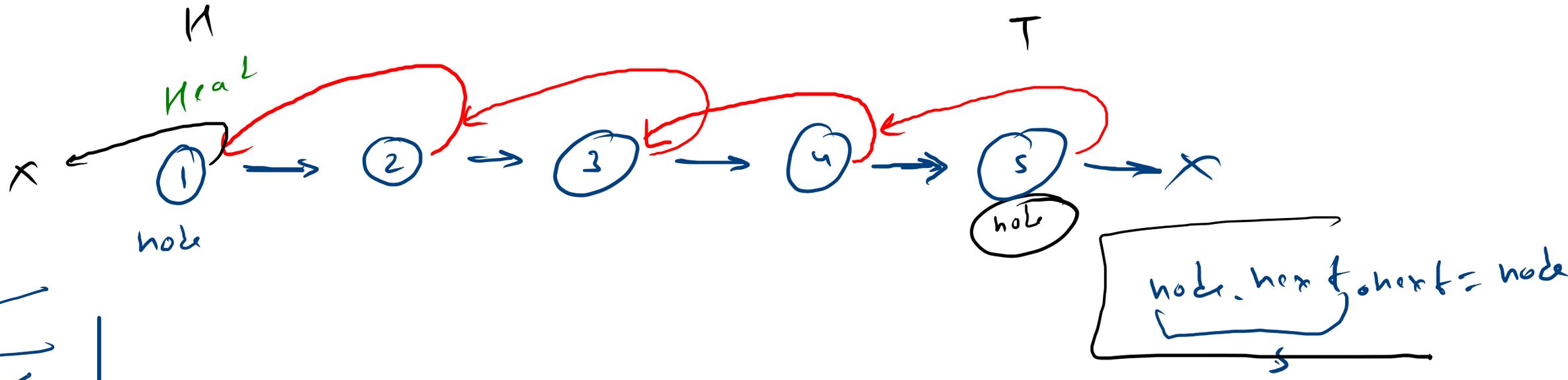


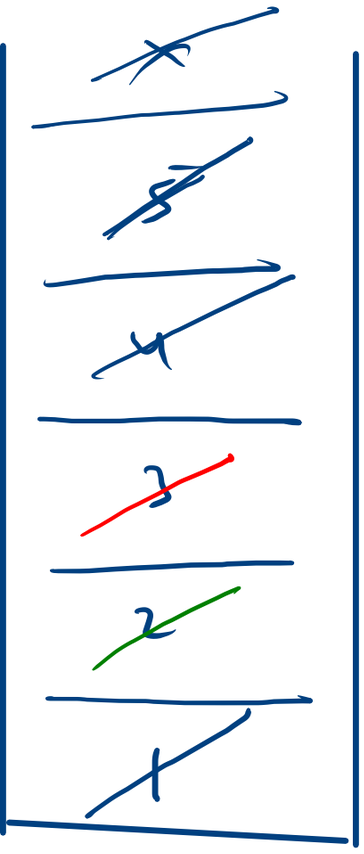
head 1
tail 5
size 5

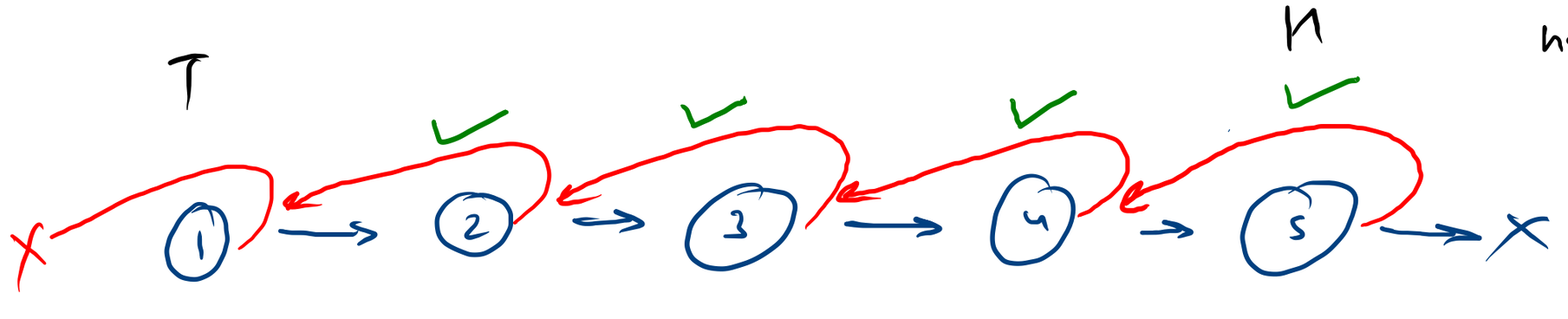


head 5
tail 1
size 5

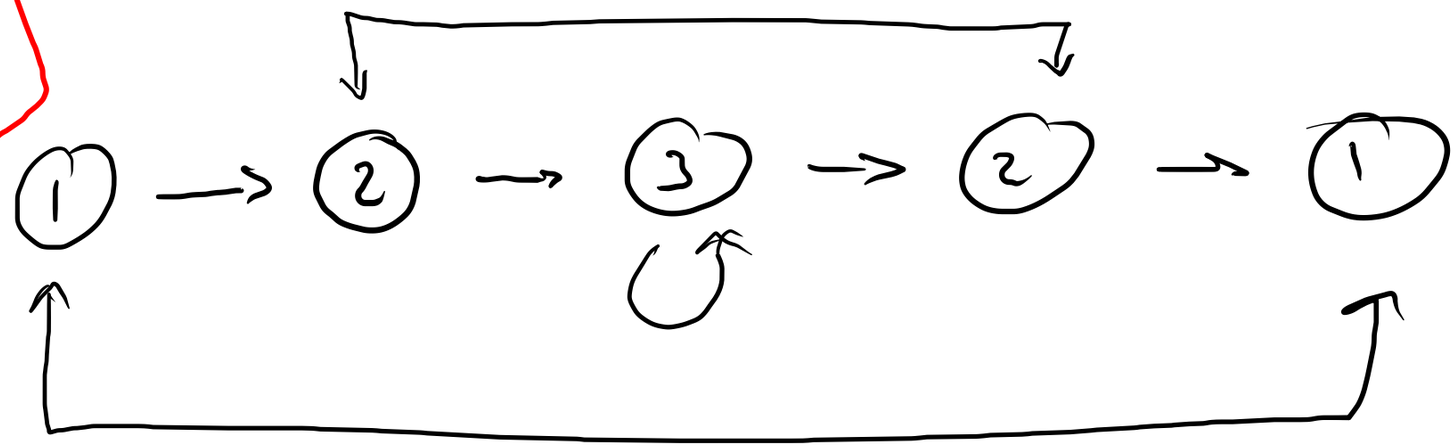
pointers
recursion


H
Head
T
X
1
2
3
4
5
X
node
node
node.next = node

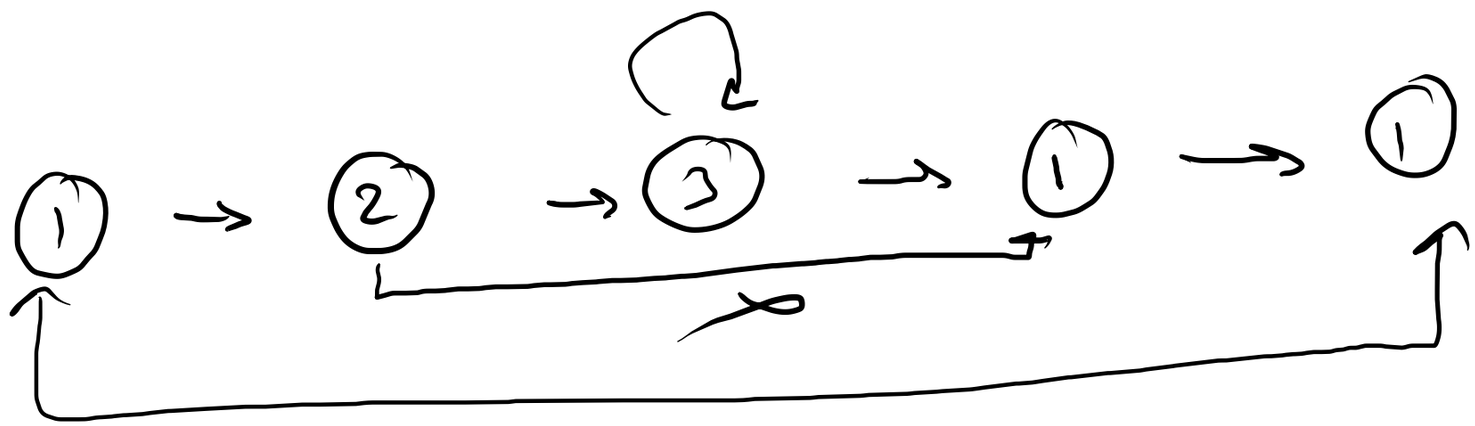



T
H
head.next=null
X
1
2
3
4
5
X

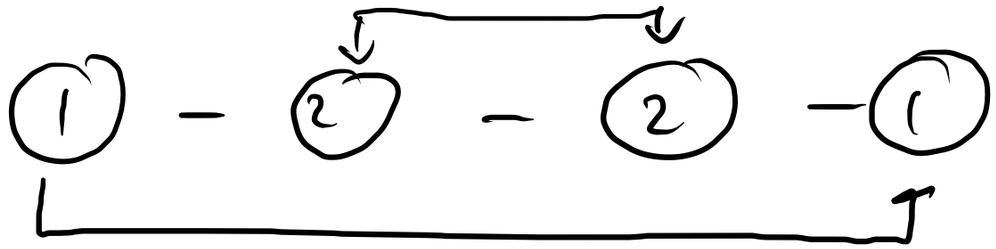
Time O(n)
space Recursion O(n)



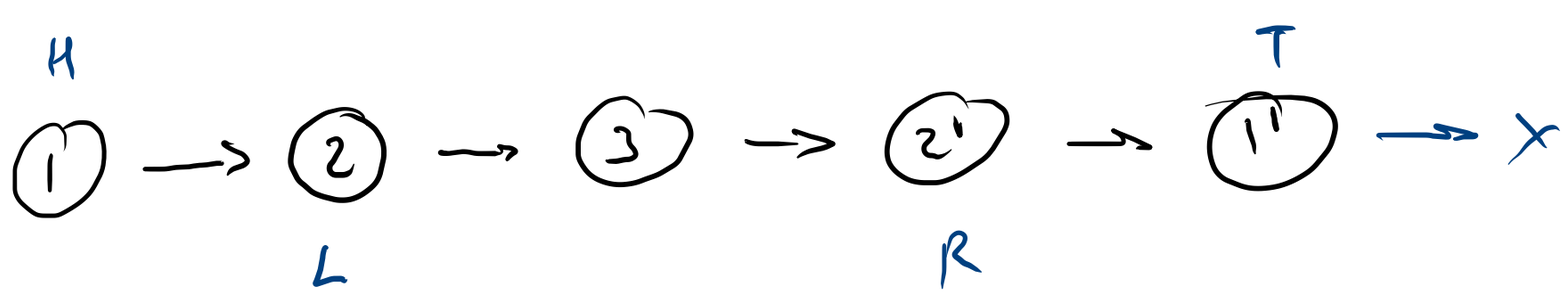
true



False

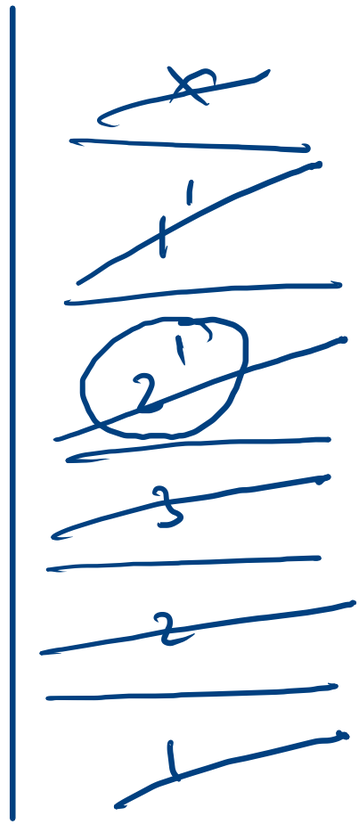


true

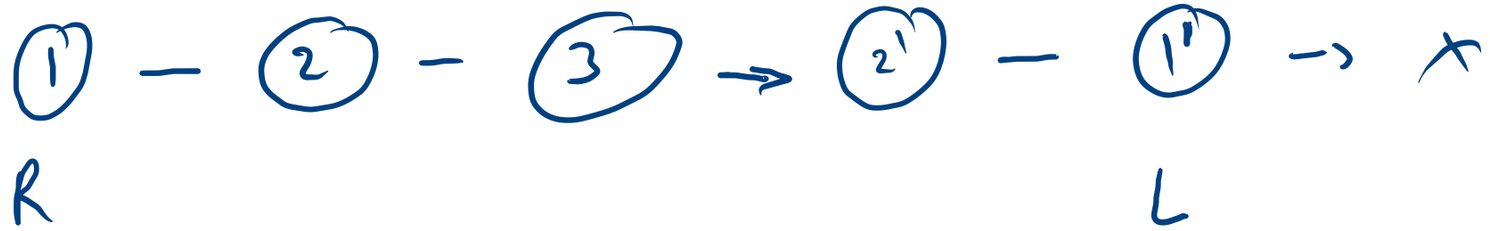



```java
public boolean IsPalindromeHelper(Node right) {
    if(right == null){
        left = head;
        return;
    }

    IsPalindromeHelper(right.next);
    // right , left

    left = left.next;
}

Node left=null;
```

head → 1
last → 1'
size → 5
left → ~~left~~ +2



O(n)



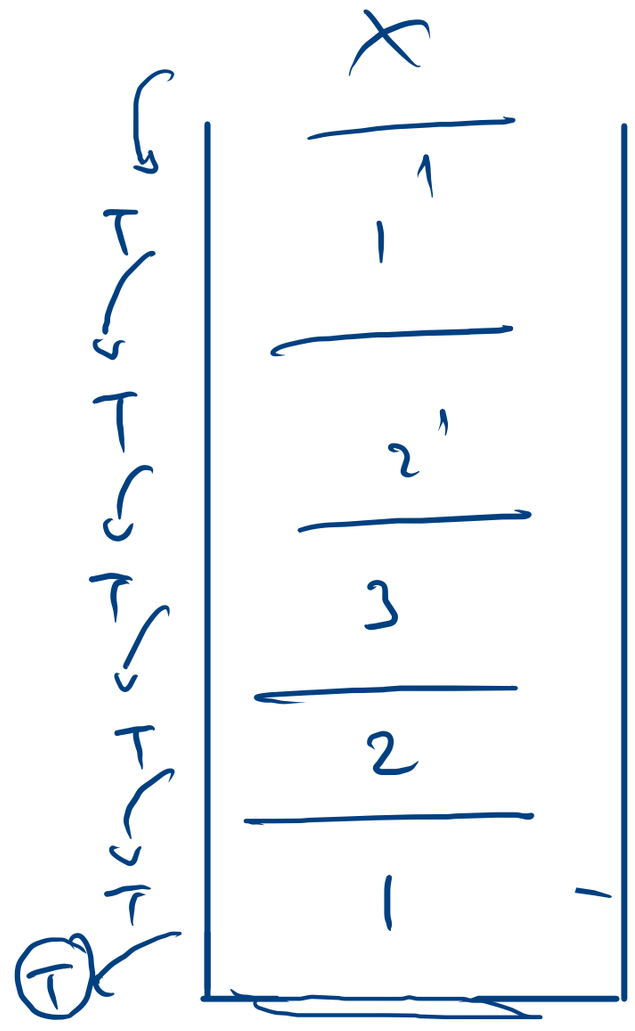

```
public boolean IsPalindromeHelper(Node right) {
    if(right == null){
        left = head;
        return true;
    }

    boolean faithP = IsPalindromeHelper(right.next);
    if(!faithP || right.data != left.data){
        return false;
    }

    left = left.next;
    return true;
}
```

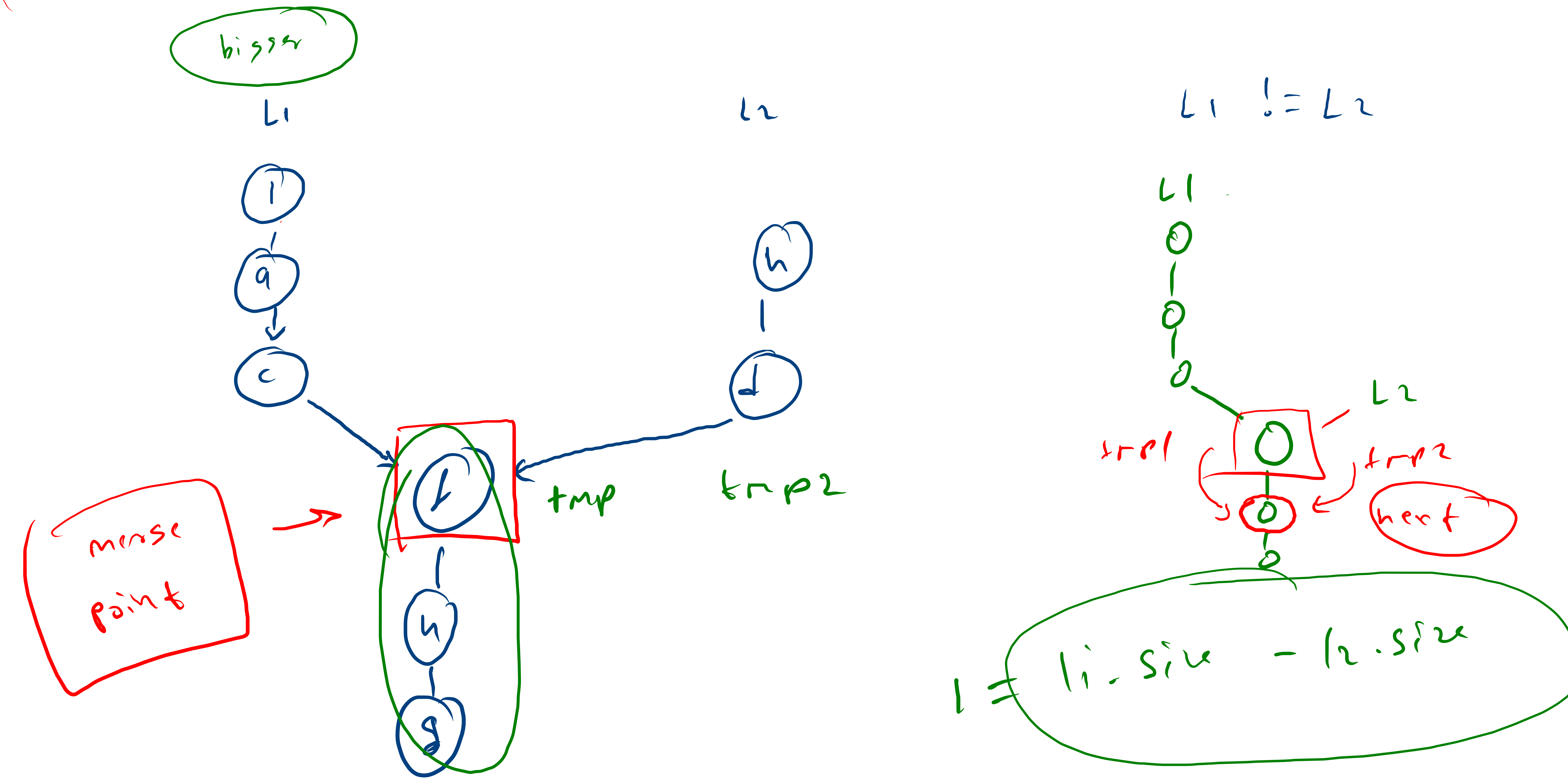
true

K

Loc x

```
Node left=null;
```

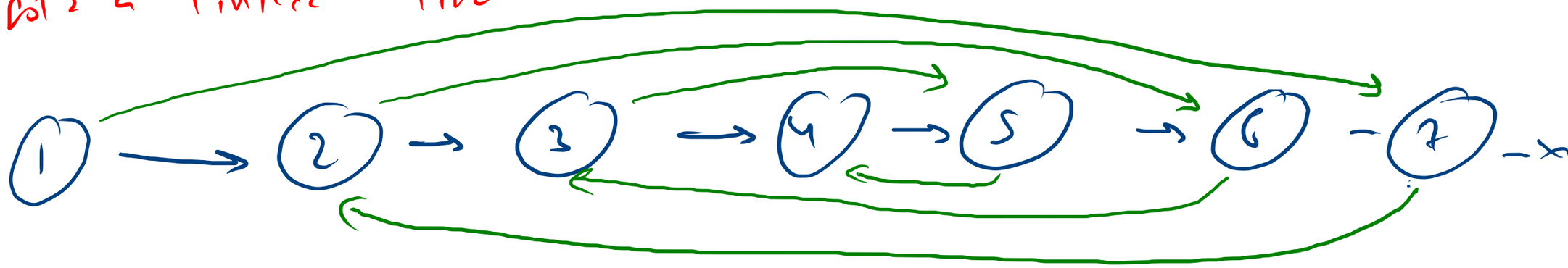




bigger
L1
L2
tmp
tmp2
merge point
L1 != L2
L1
L2
tmp1
tmp2
next
l = l1.size - l2.size

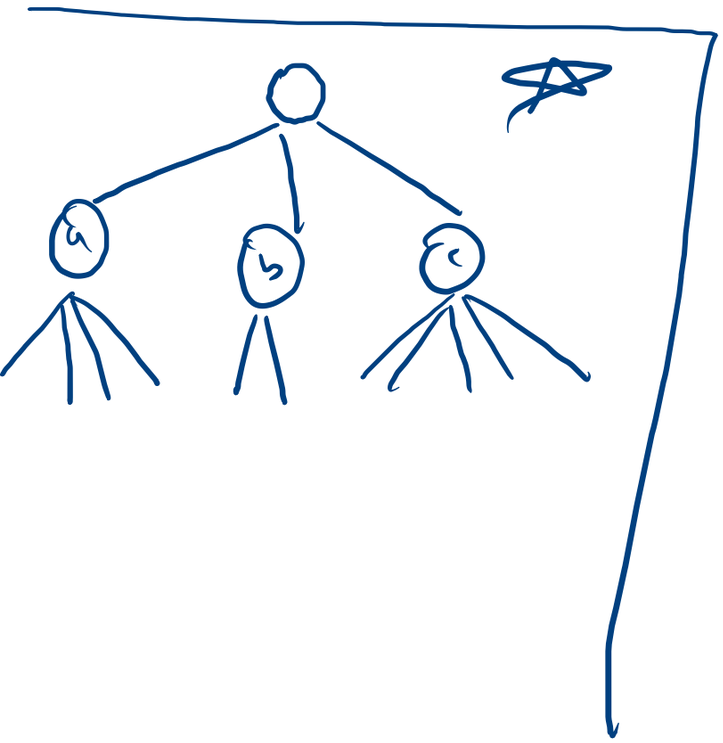
11.W

Doll a linked list



is palindrome / Node left

Tree



Array 

ArrayList [ ]

dynamic

stack 

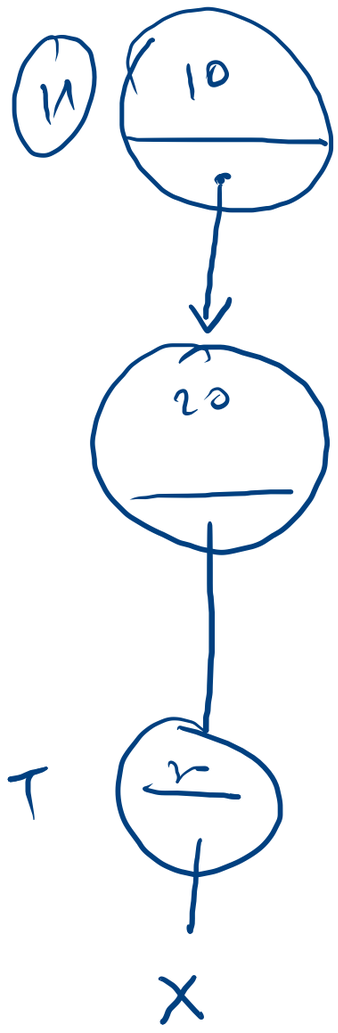
Queue 

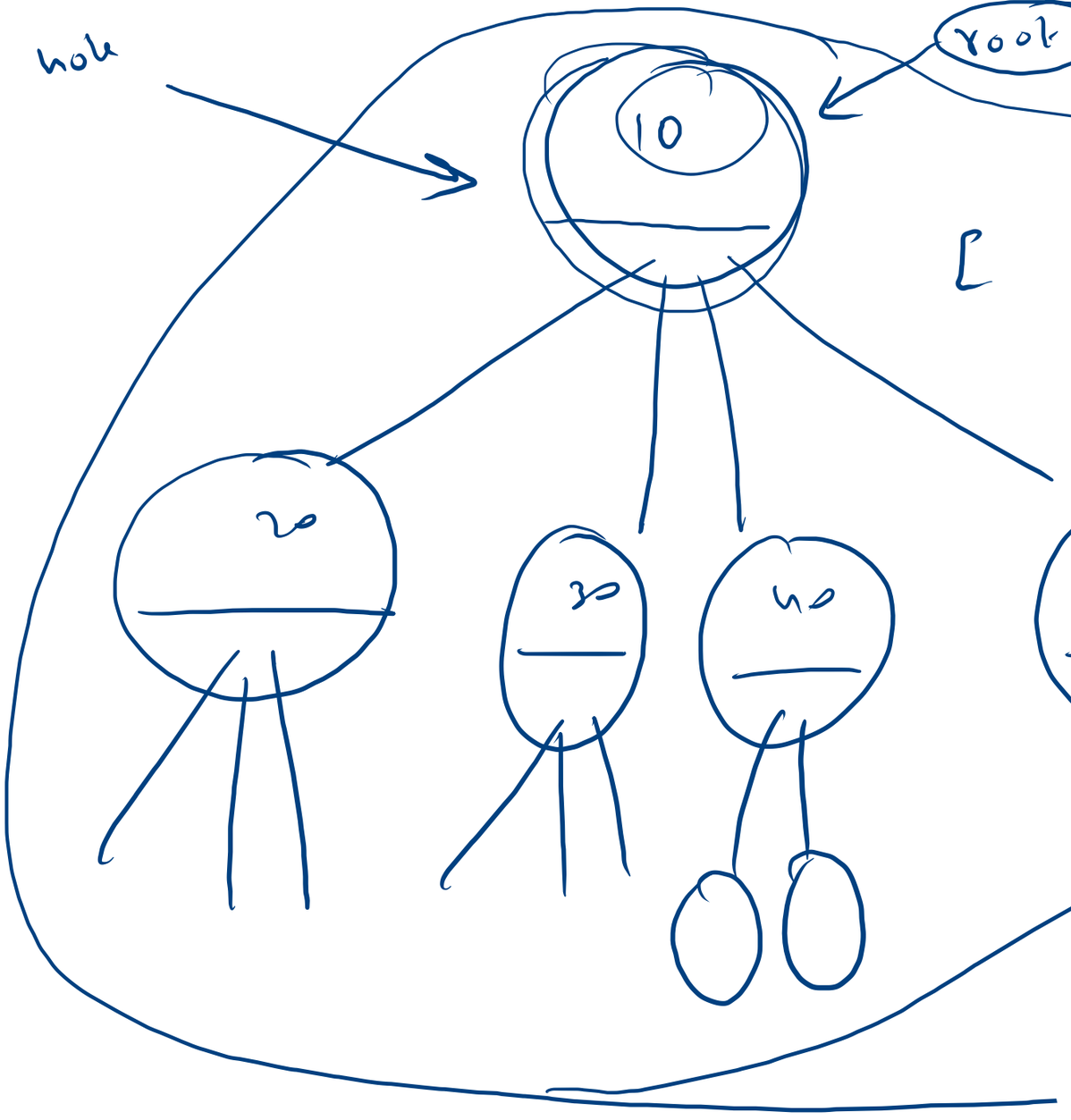
Linear

String 

LinkedList 

{
int data
Node next
}

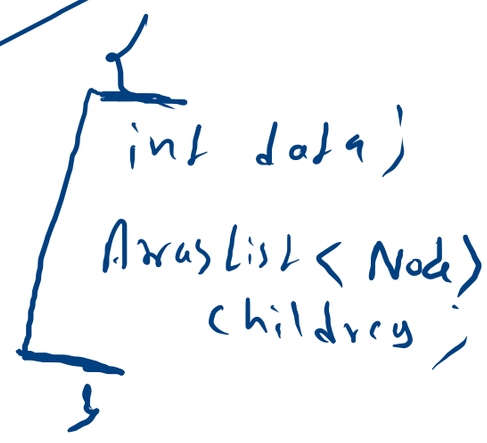




Requirement

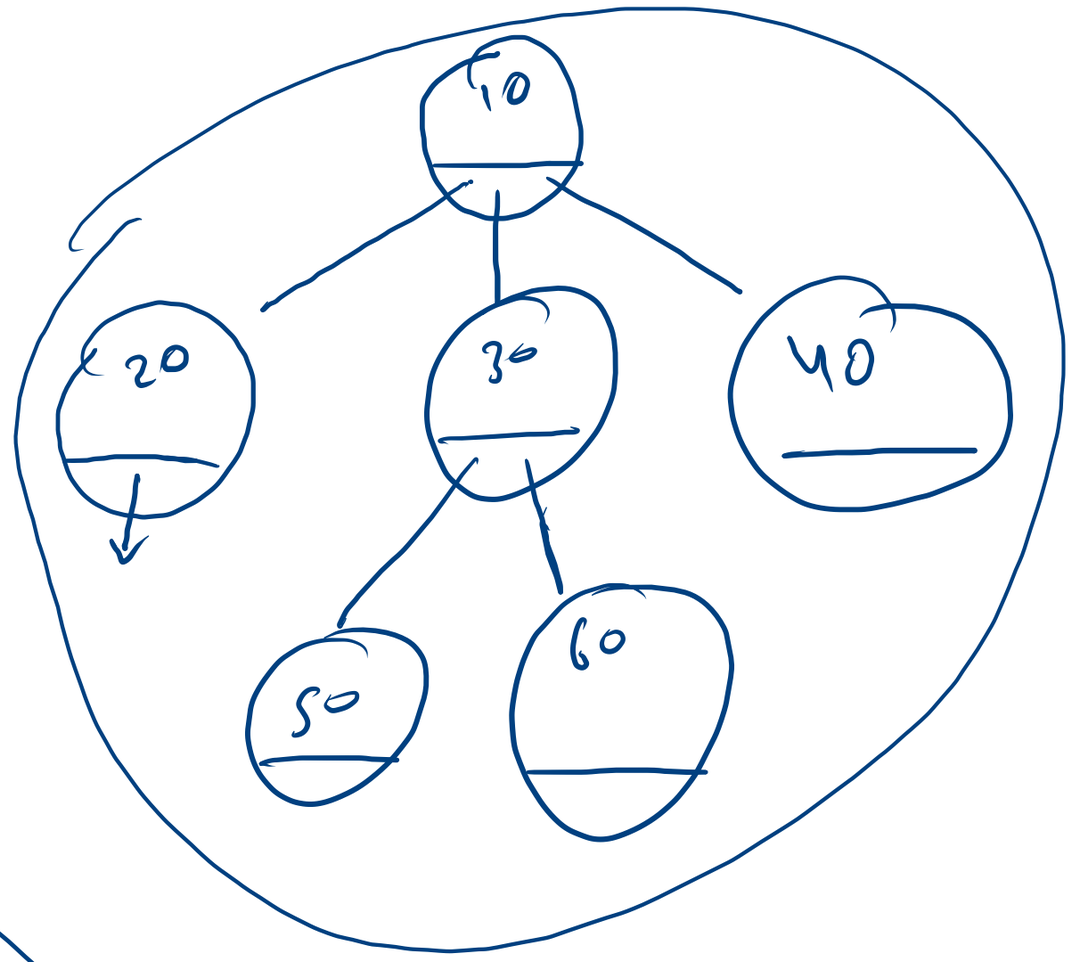
[ ]

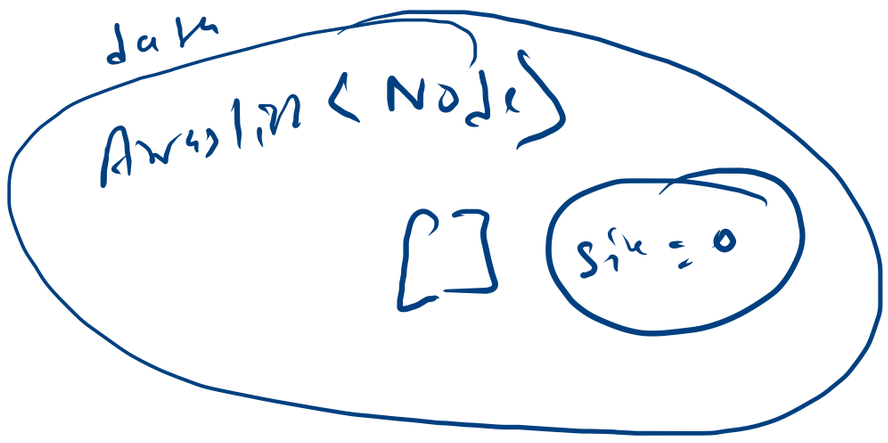
multiple dynamic



10, 20,
-1, 30,
50, -1,
60, -1,
-1, 40,
-1, -1

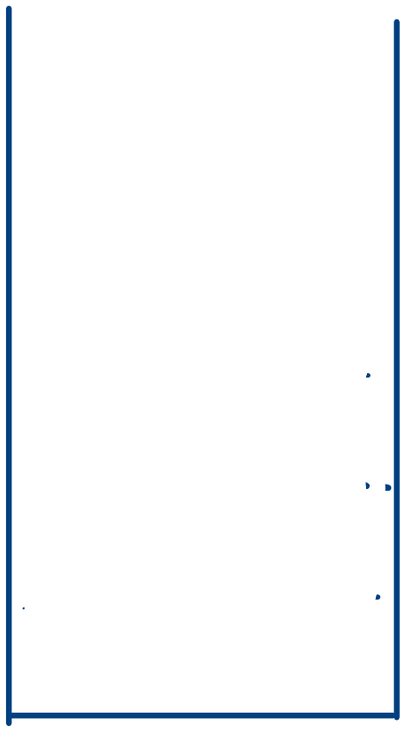
4

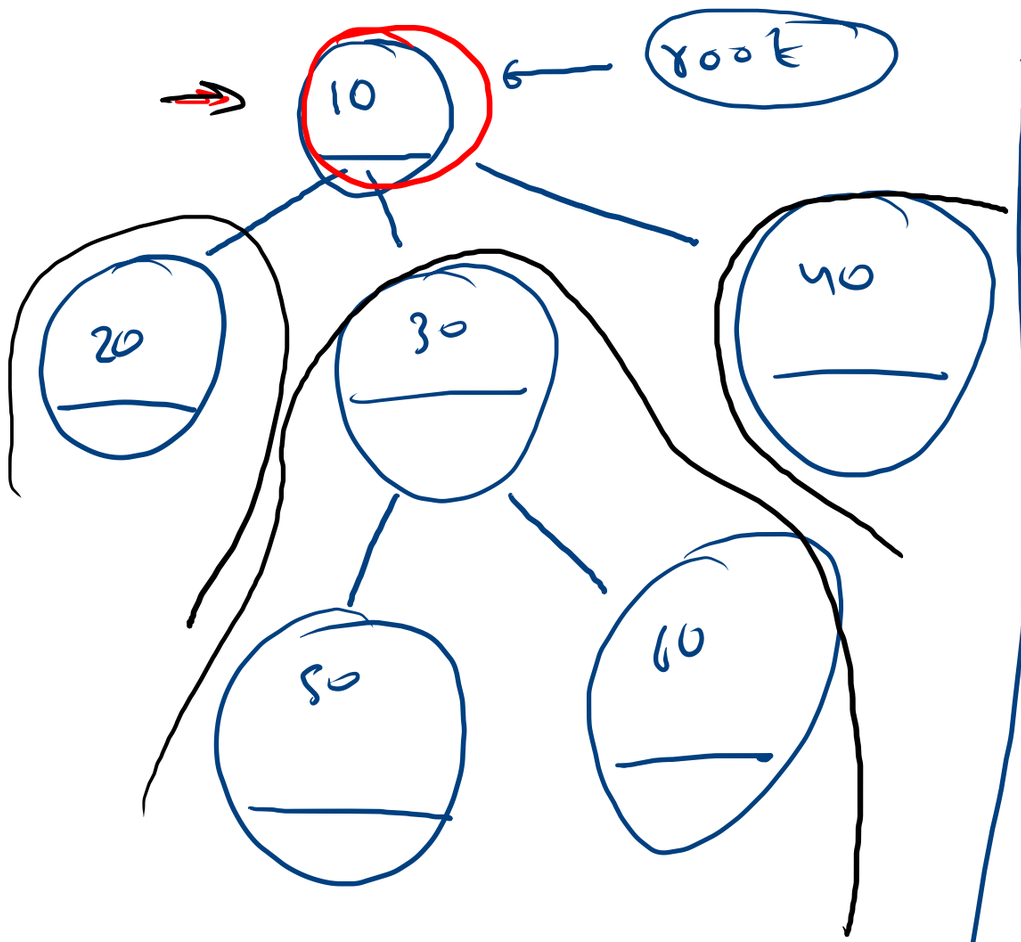




next = hu

10, 20,
-1, 30,
50, -1,
60, -1,
-1, 40,
-1, -1





★ 10 → 20, 30, 40, .
- 20 → .
- 30 → 50, 60, .
  - 50 → .
  - 60 → .
- 40 → .

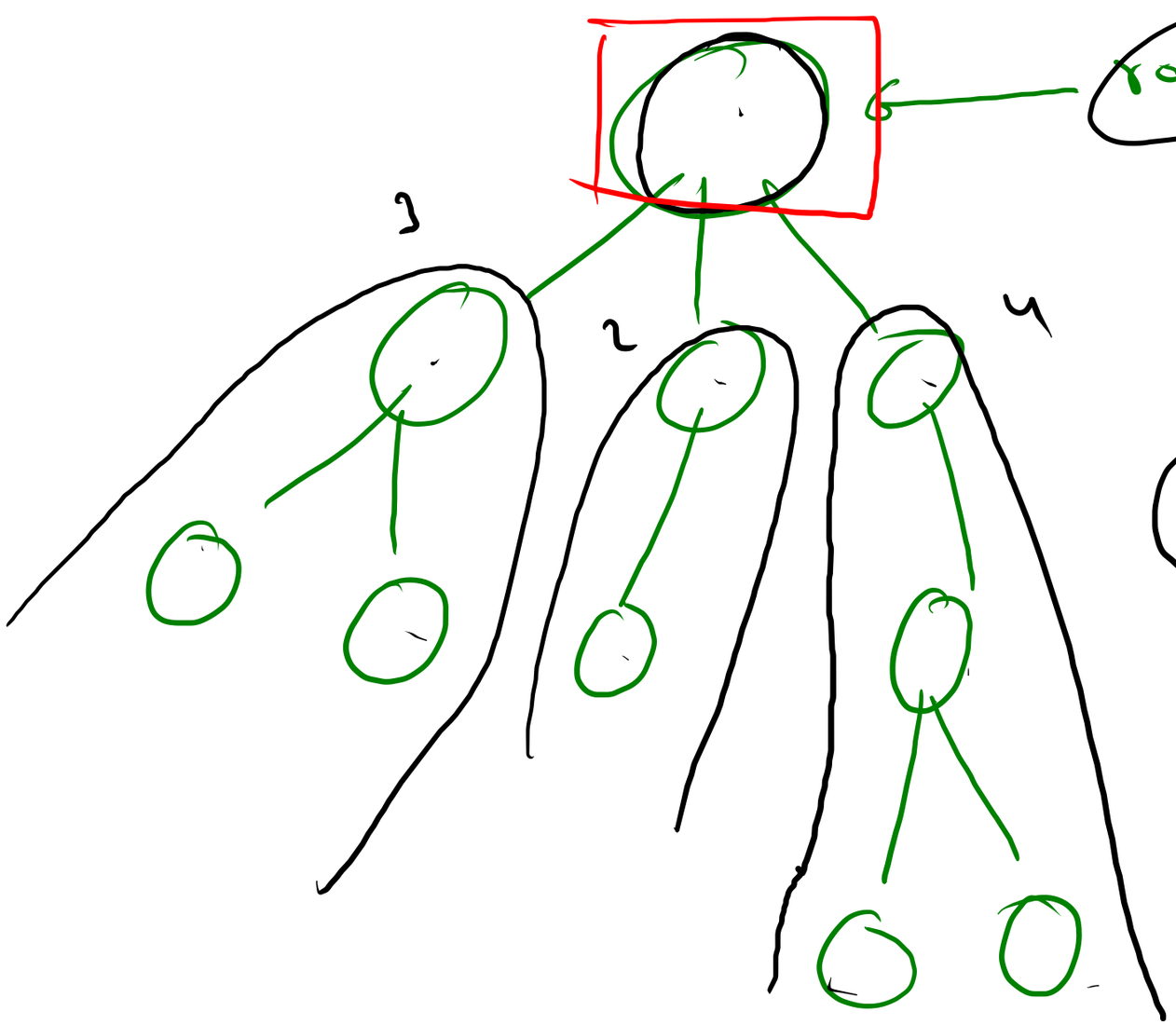
exp Lain

creak node

x-1 [peek / push .children.add(node)

-1 st.pop

expectation



```java
public static int size(Node node){
    int s = 0;

    for(Node child: node.children){
        s += size(child);
    }

    return s+1;
}
```

0+1