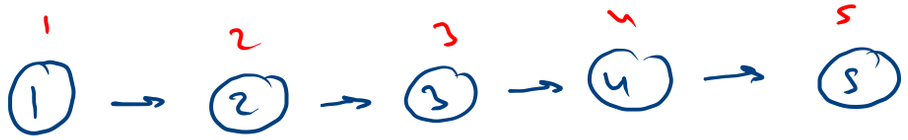
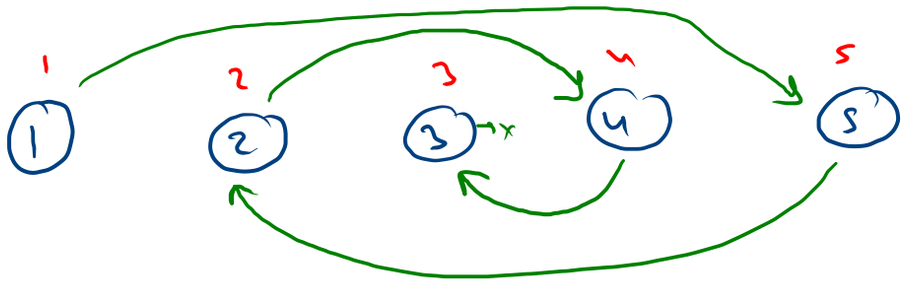
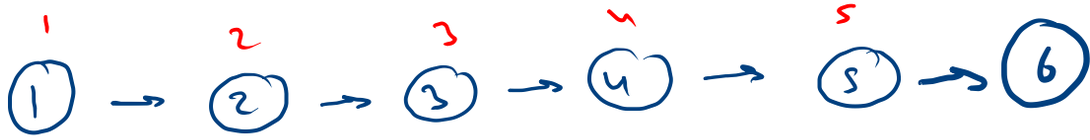
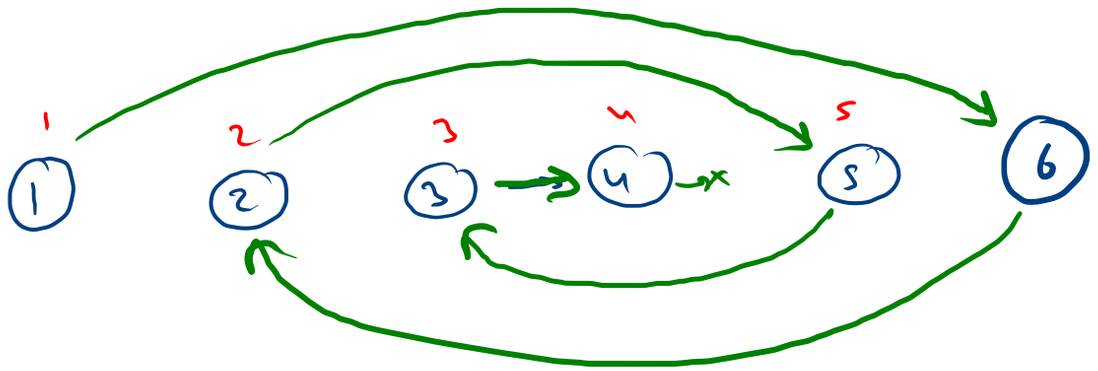
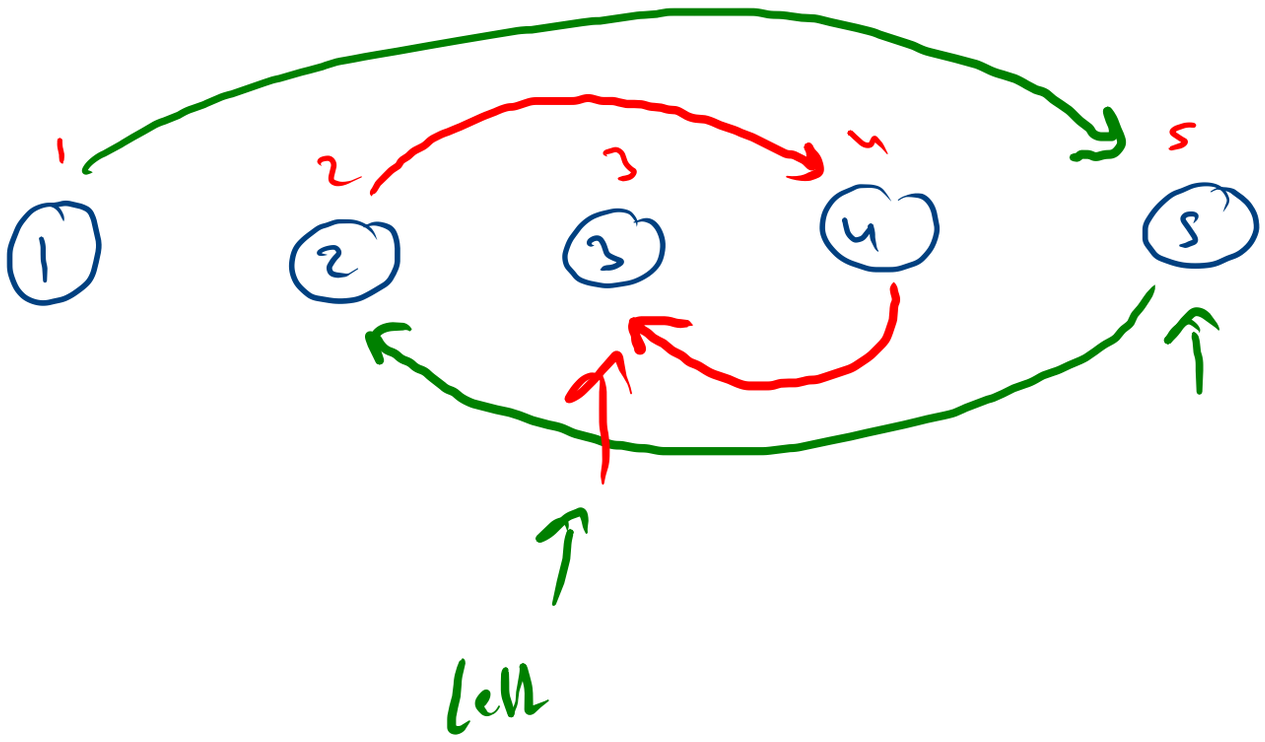


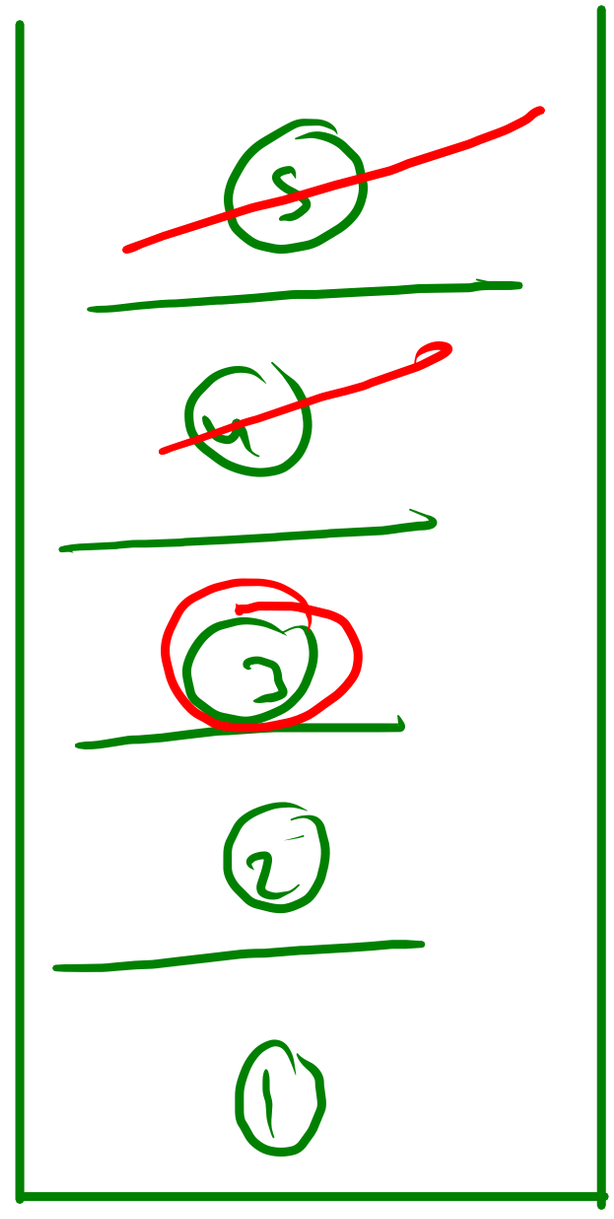


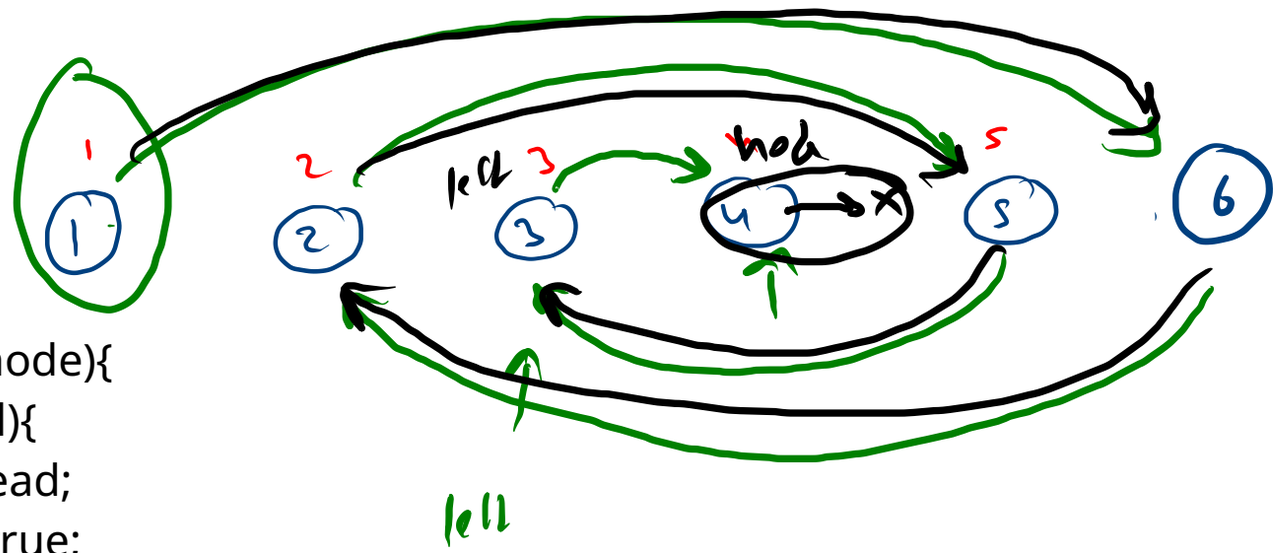
head 1
tail 5

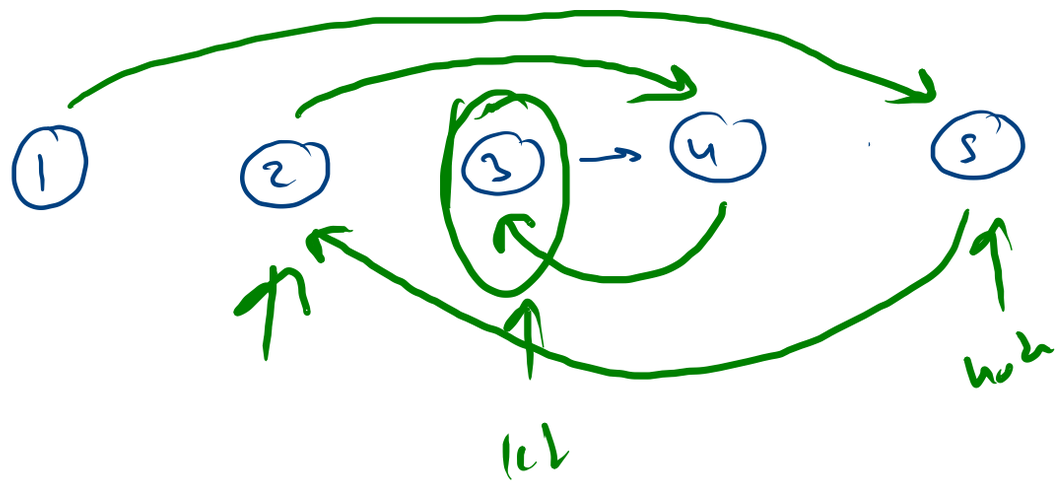


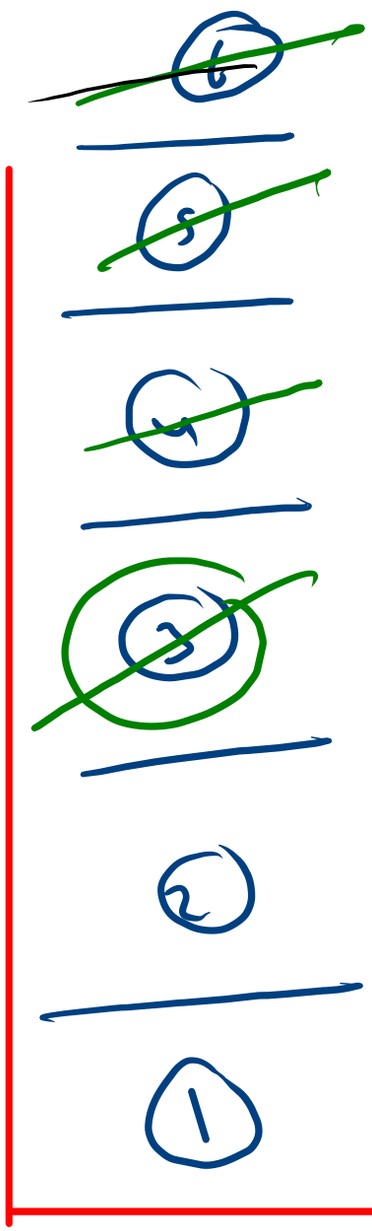



1
2
3
4
5
1
2
3
4
5
Left


5
4
3
2
1



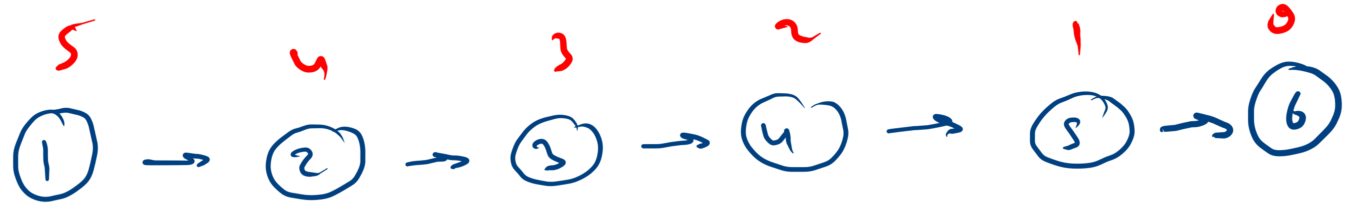





```
booelan fold(Node node){
        if(node == null){
                left = head;
                return true;
        }

        if(false==fold(node.next))return false;

        if(left == node){
                left.next = null;
                tail = left;
                return false;
        }
        if(left.next == node){
                node.next = null;
                tail = node;return false;
        }

        node.next = left.next;
        left.next = node;
        left = node.next;
}
```

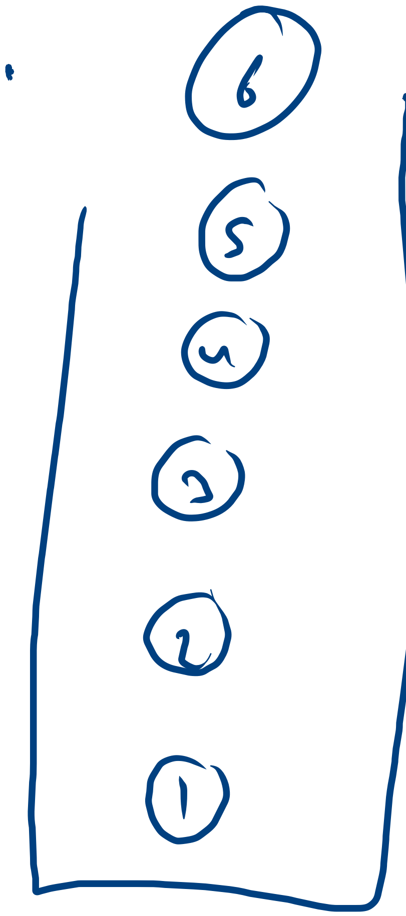
kth node

i = 2



Stack

space O(1)

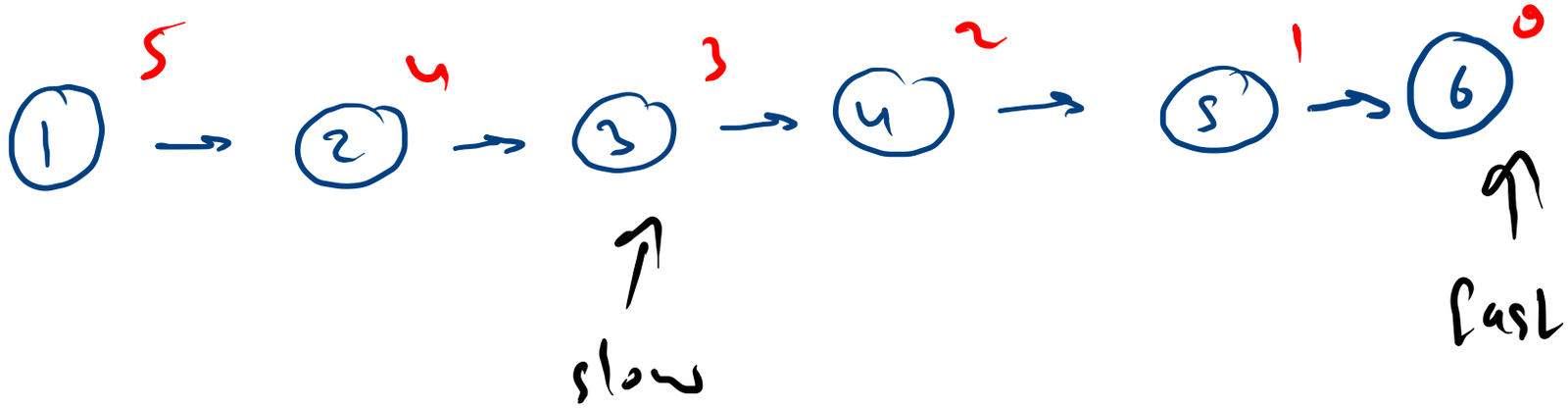


i

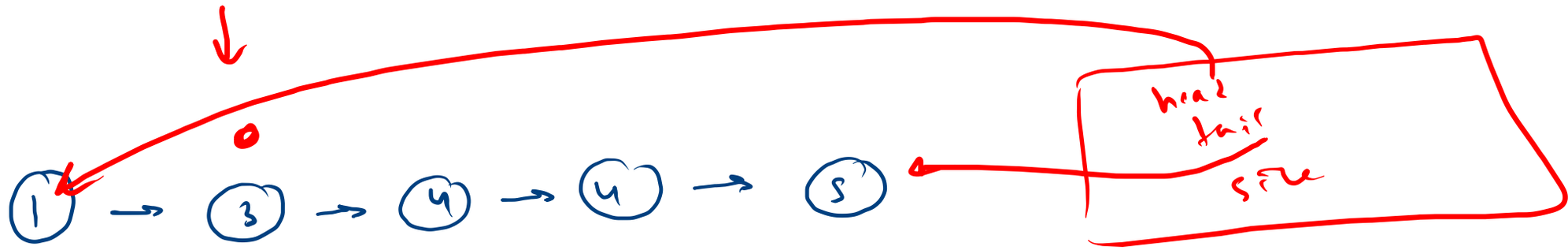
2

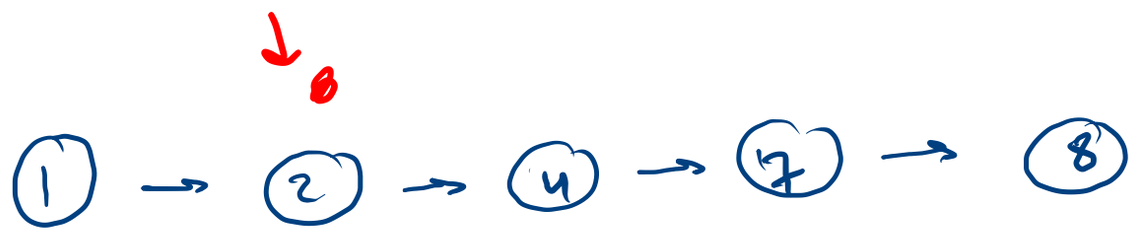
1

i = 3



Merge 2 sorted linked list

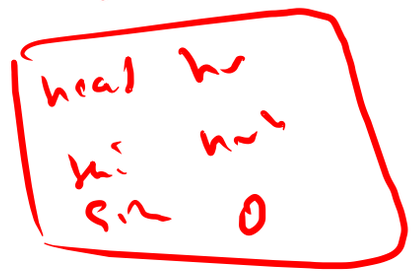


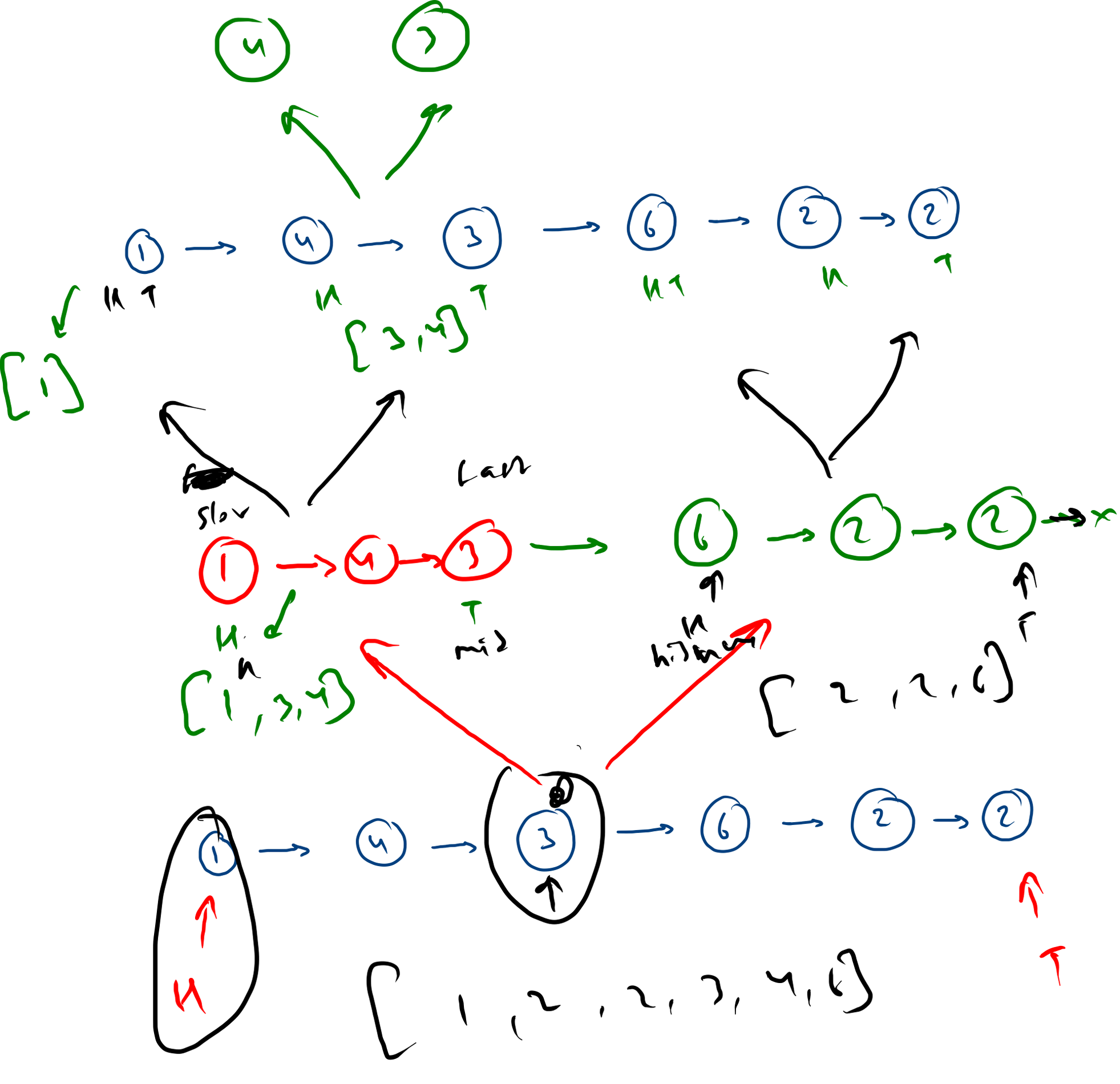


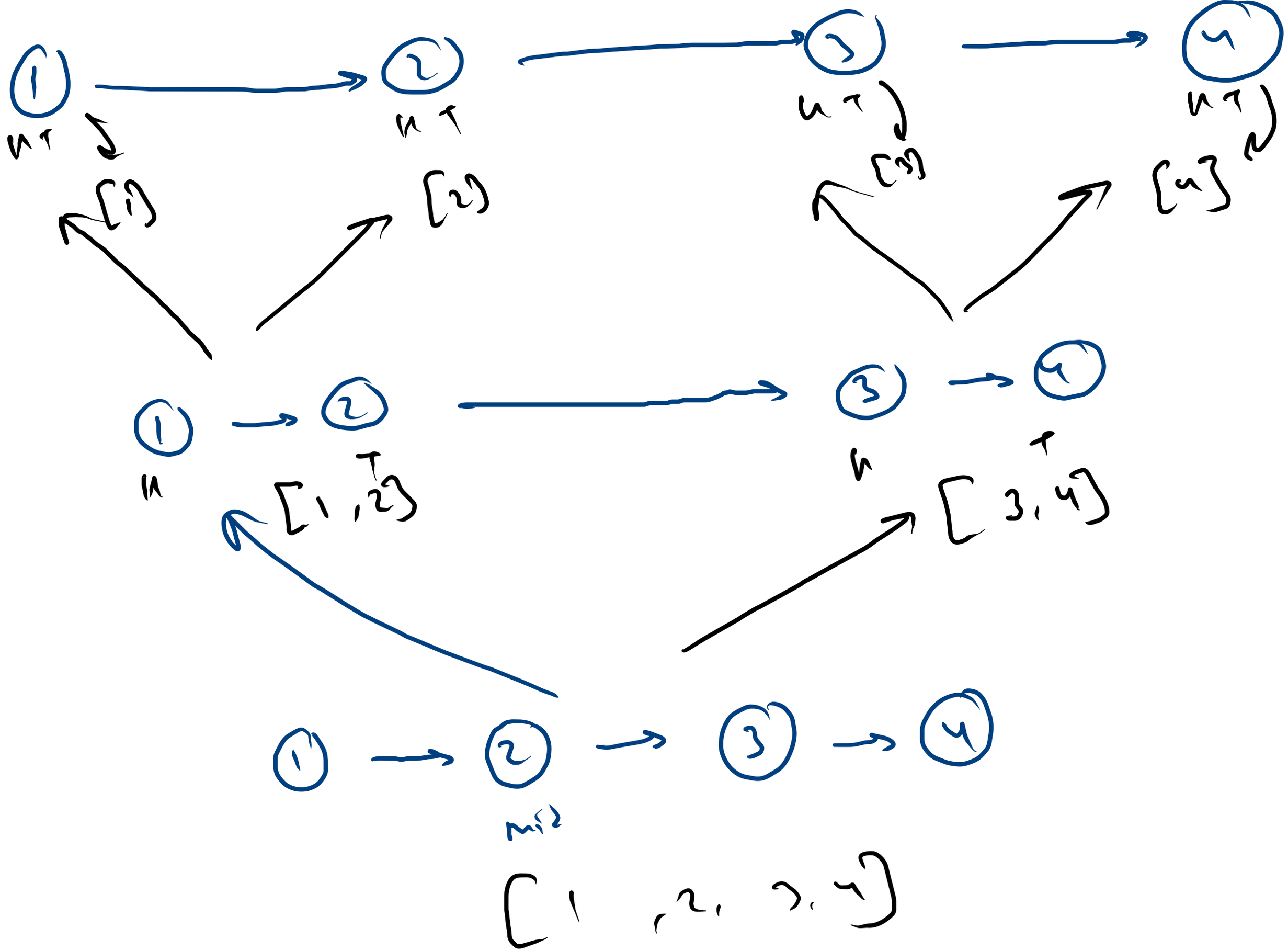
Algo

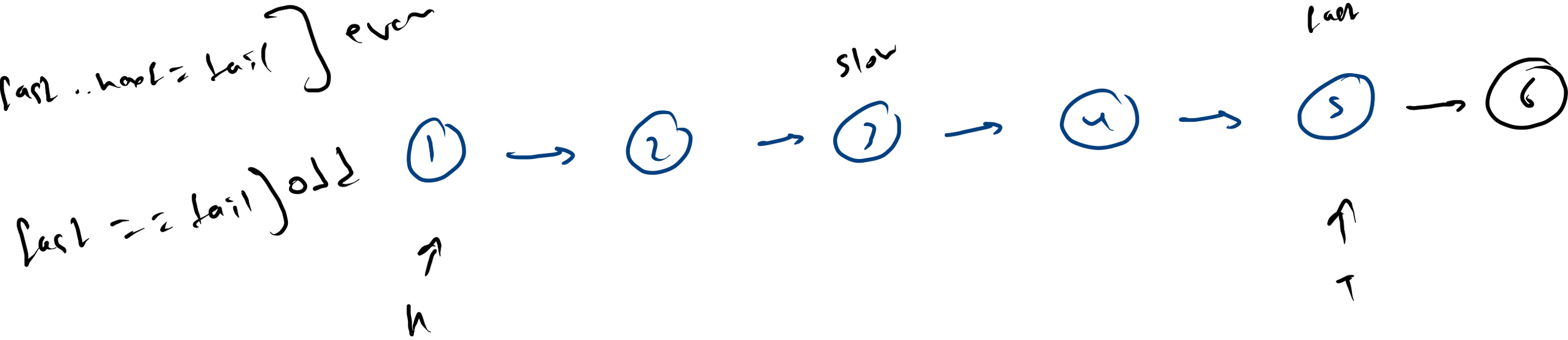
ans → 1 → 1 → 2

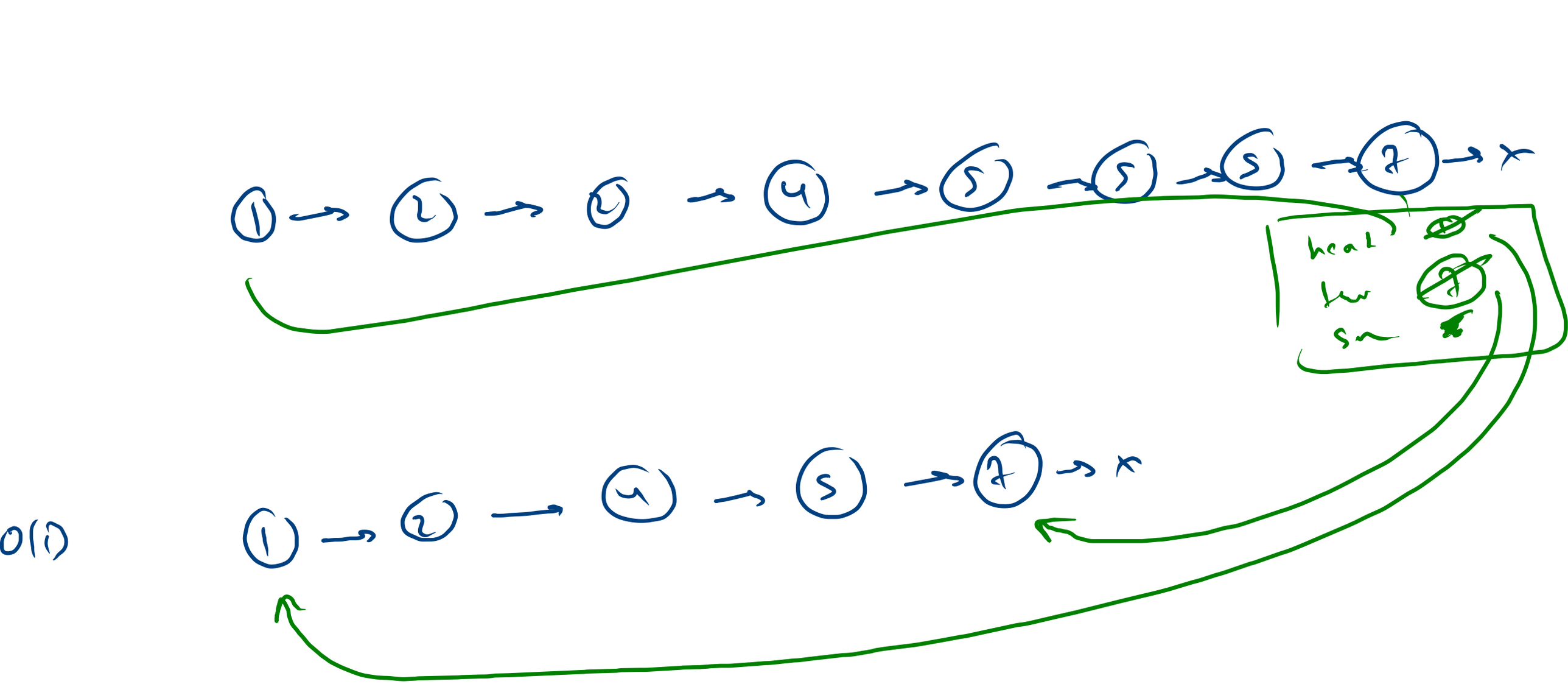
ans

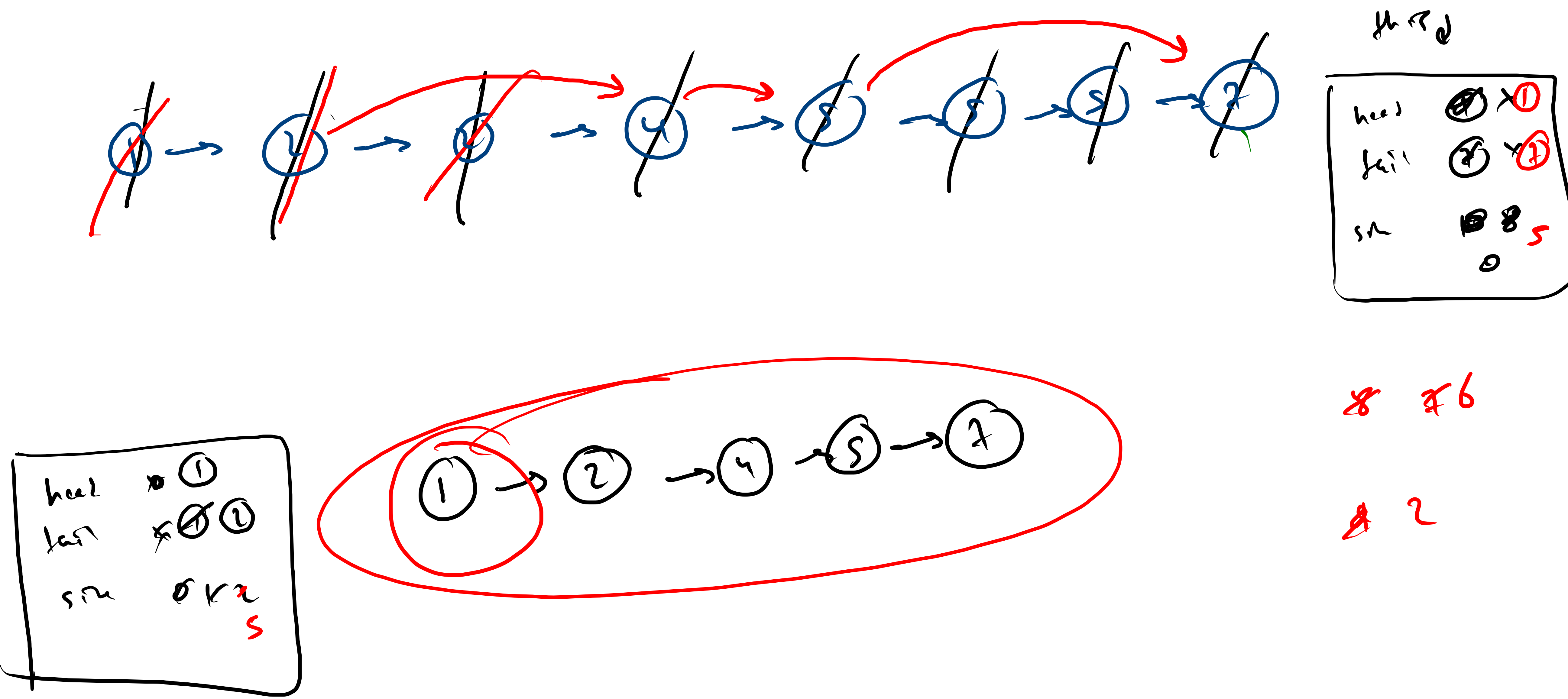


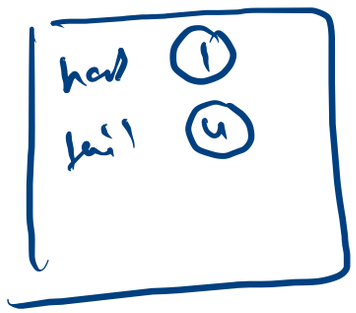

slow
fast
[1]
[3,4]
[1,3,4]
[2,2,6]
[1,2,2,3,4,6]

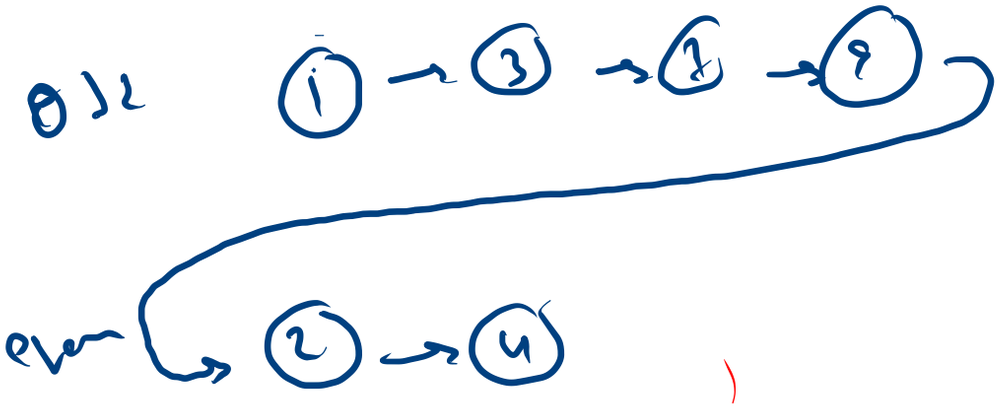

1
2
3
4
[1]
[2]
[3]
[4]
1
2
3
4
[1,2]
[3,4]
1
2
3
4
[1,2,3,4]

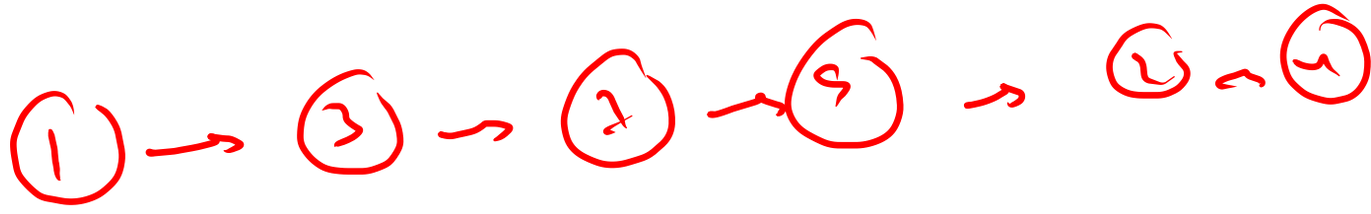

last..head = tail
even
last == tail
odd
1
2
3
4
5
6
slow
fast
h
T

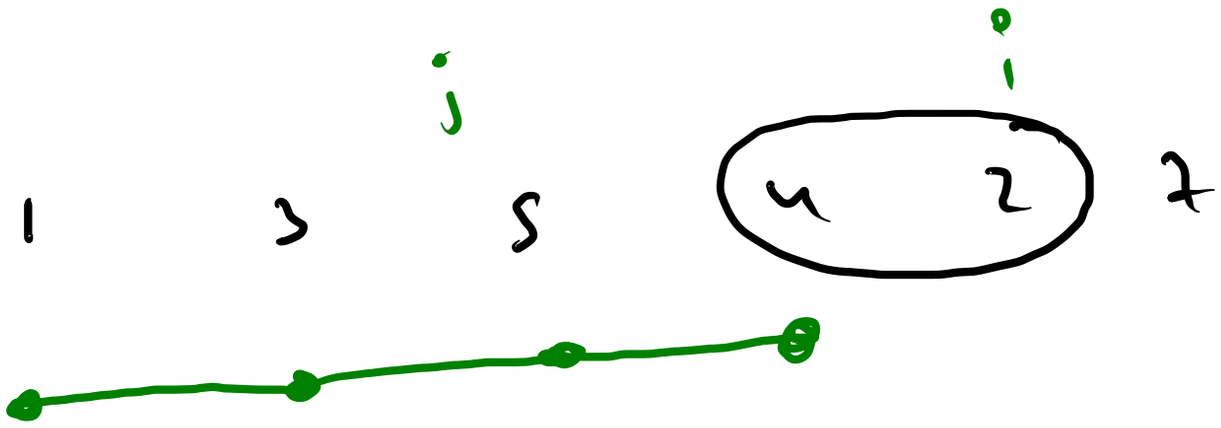

1
2
3
4
5
6
7
x
heal
lw
sw
1
2
4
5
7
x

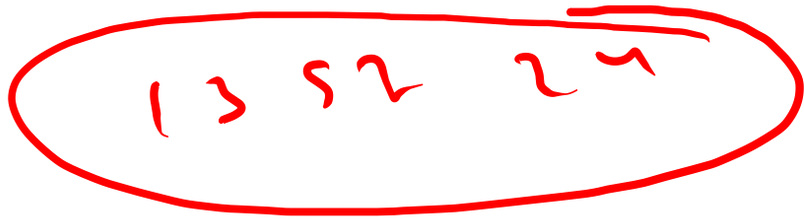

head
tail
size

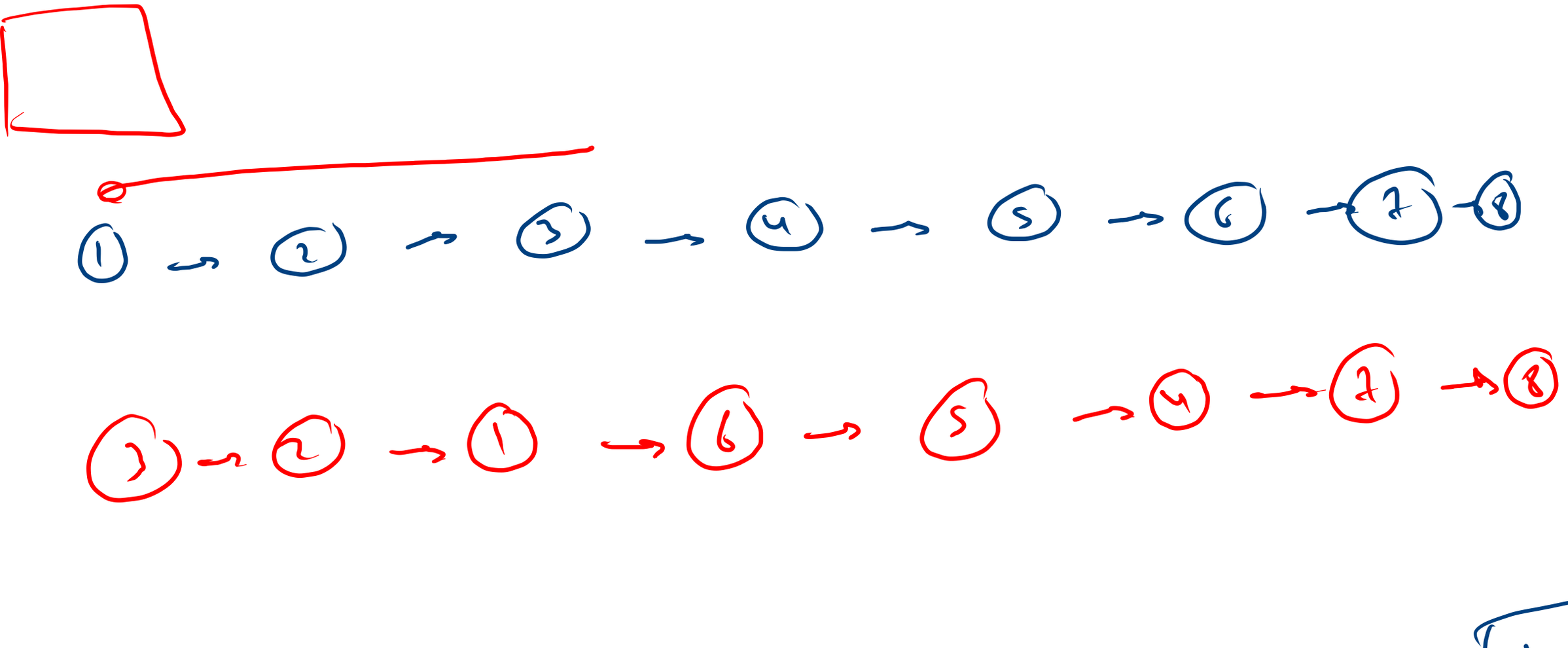

head
tail

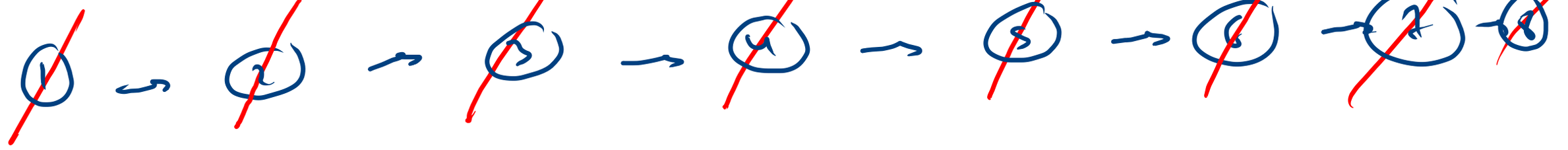


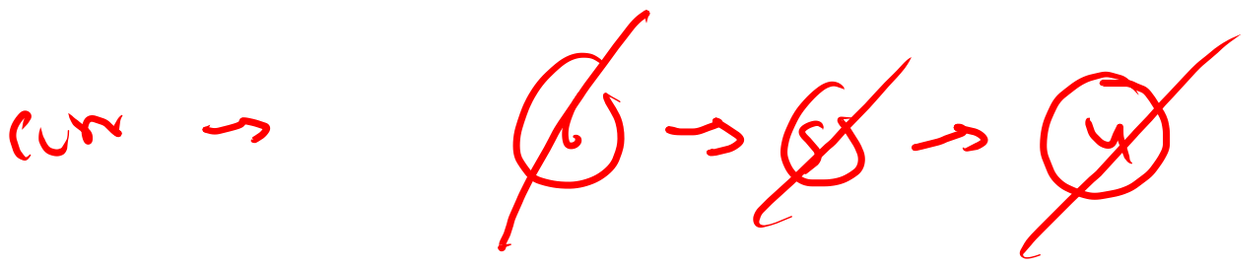


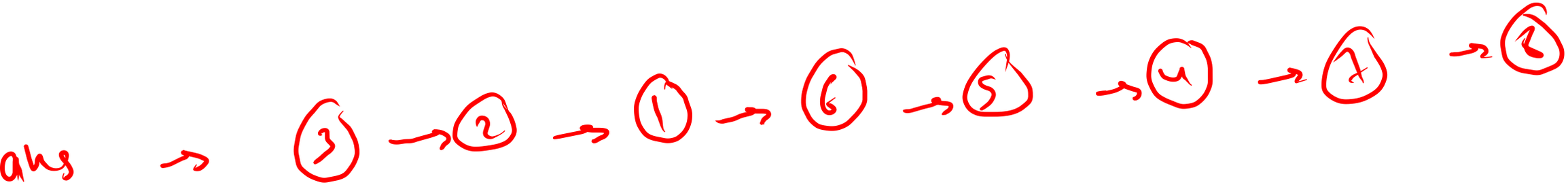

j
i
1
3
5
4
2
7



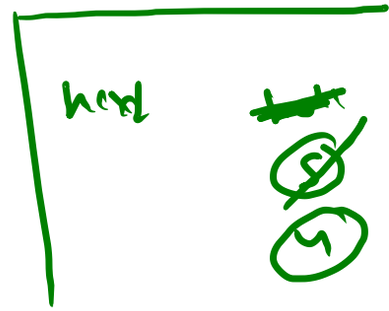

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8
3 → 2 → 1 → 6 → 5 → 4 → 7 → 8
k = 3

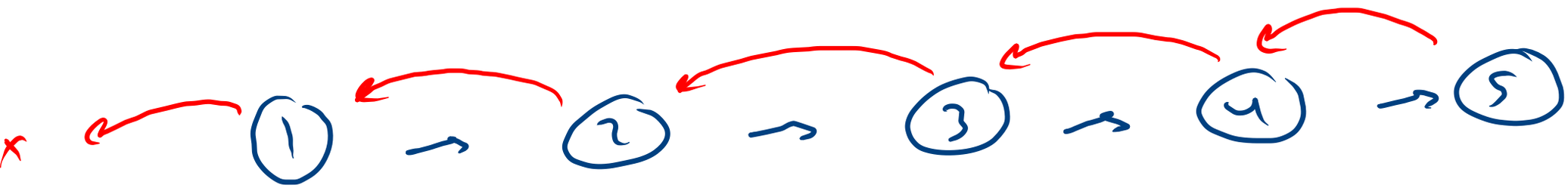


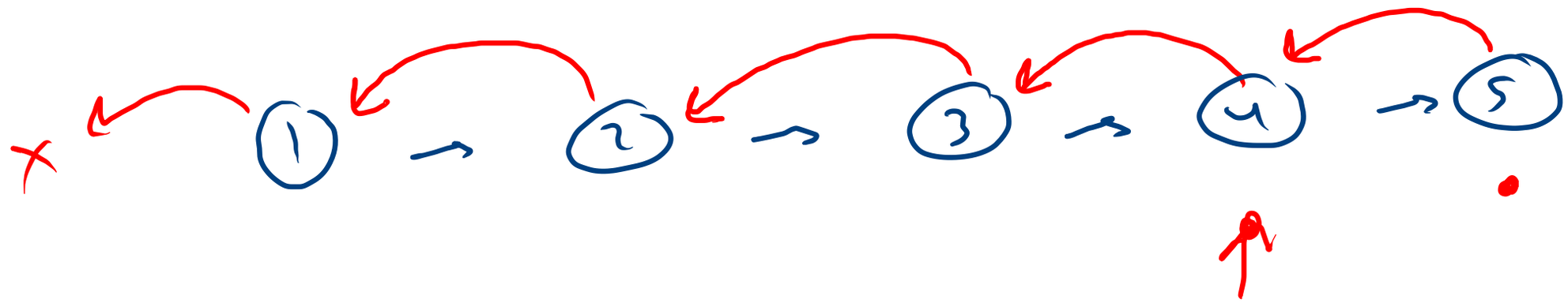




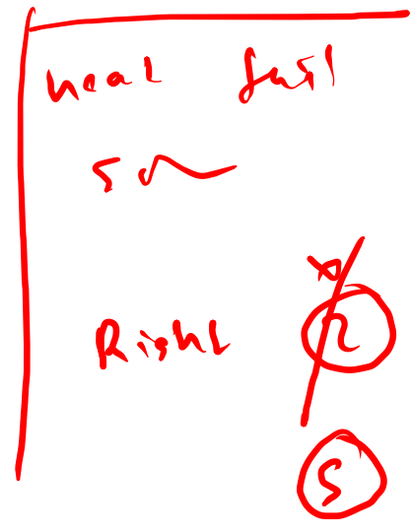

head
next
1
2
3
4
5
x

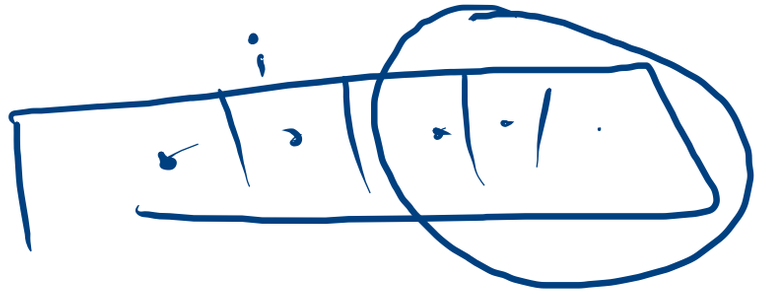

next
5
4

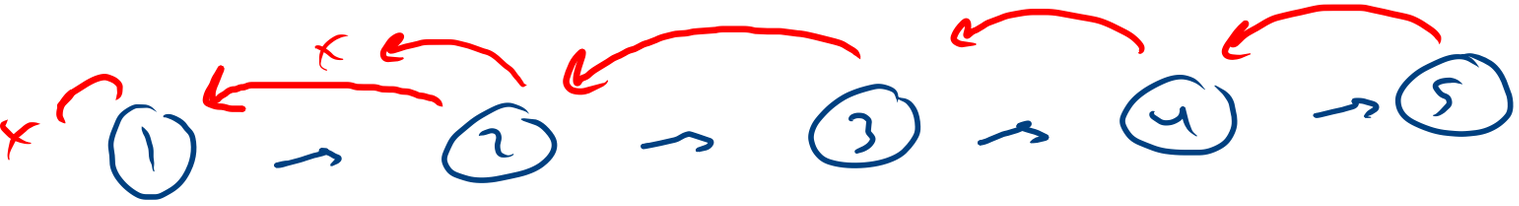

x
1
2
3
4
5



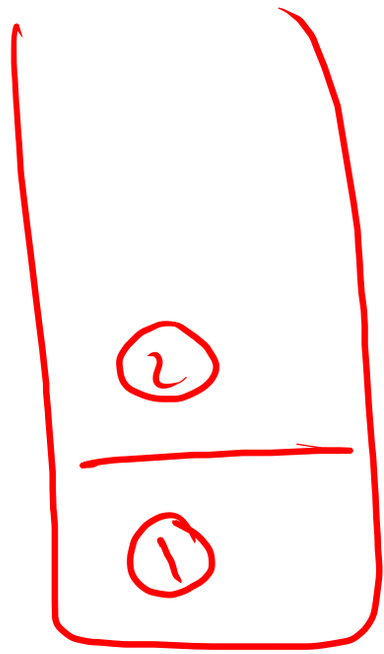
Right

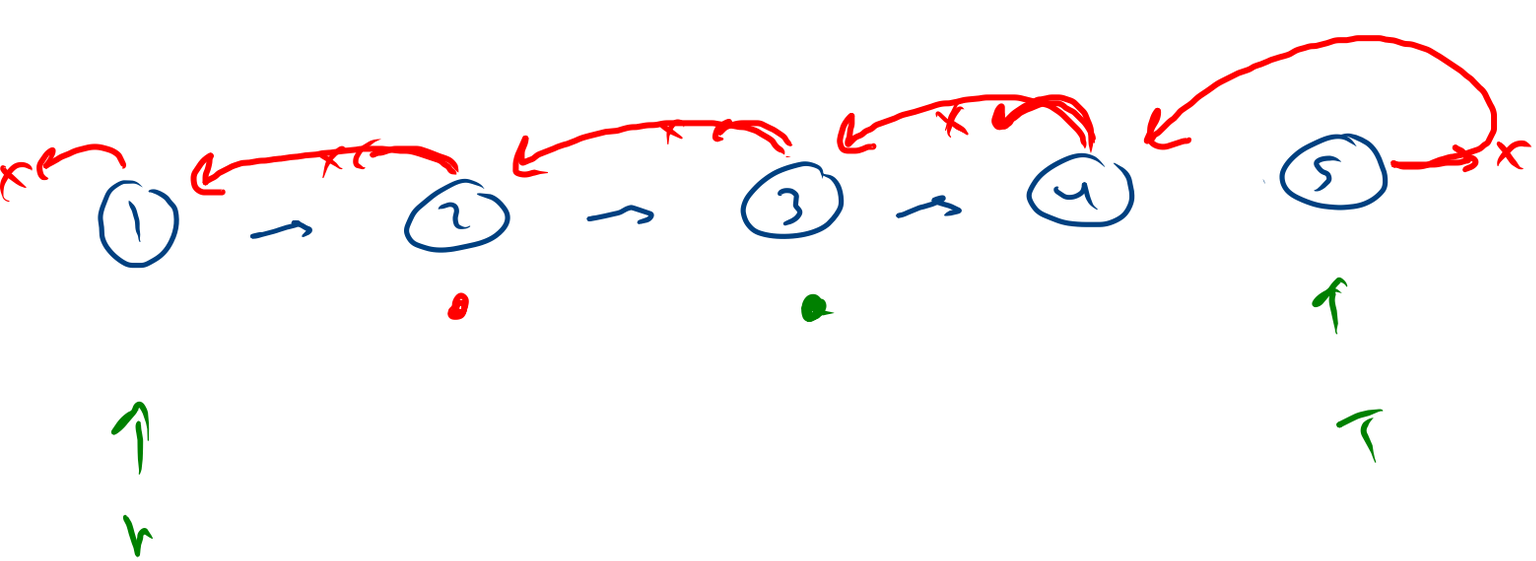




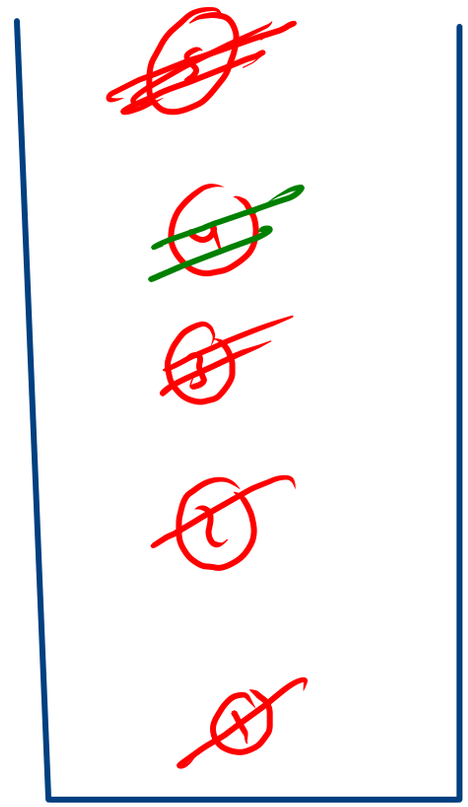



```
faitb(node.next);

node.next.next = node;
node.next = null;
```

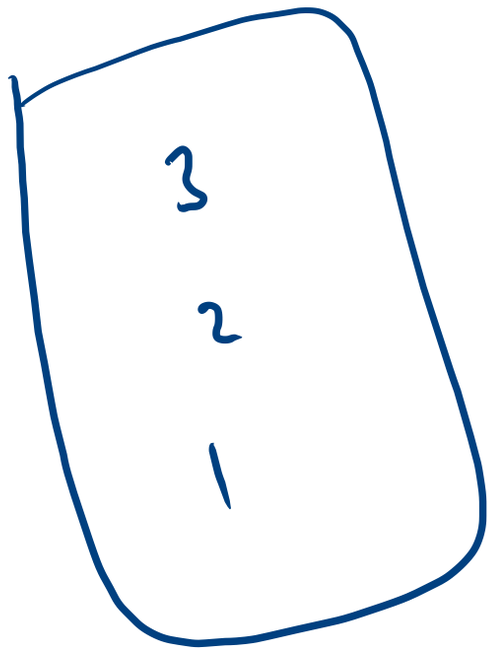


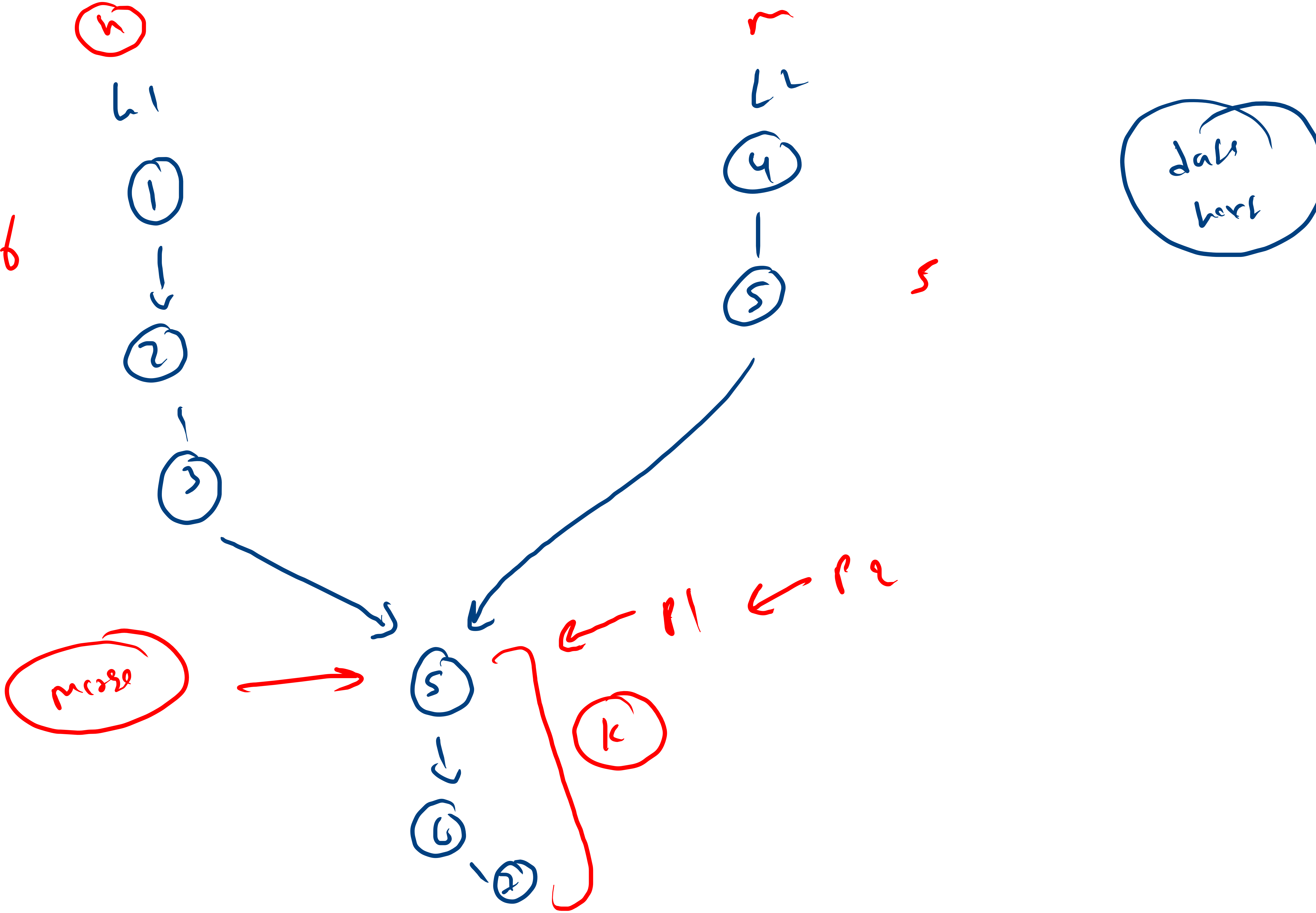


```
faitb(node.next);

node.next.next = node;
node.next = null;
```

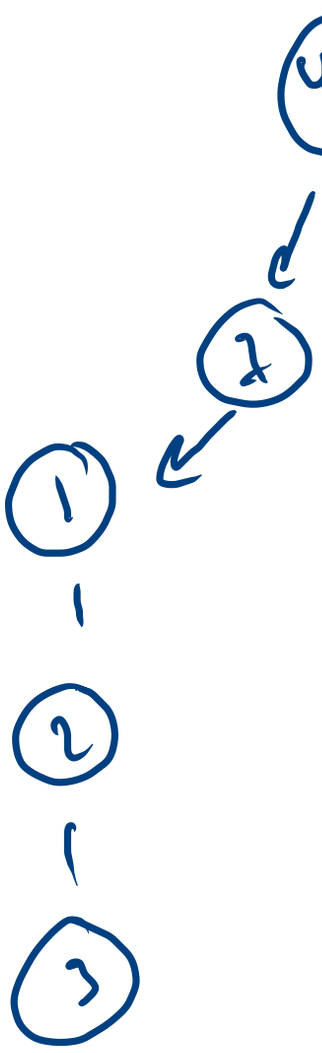


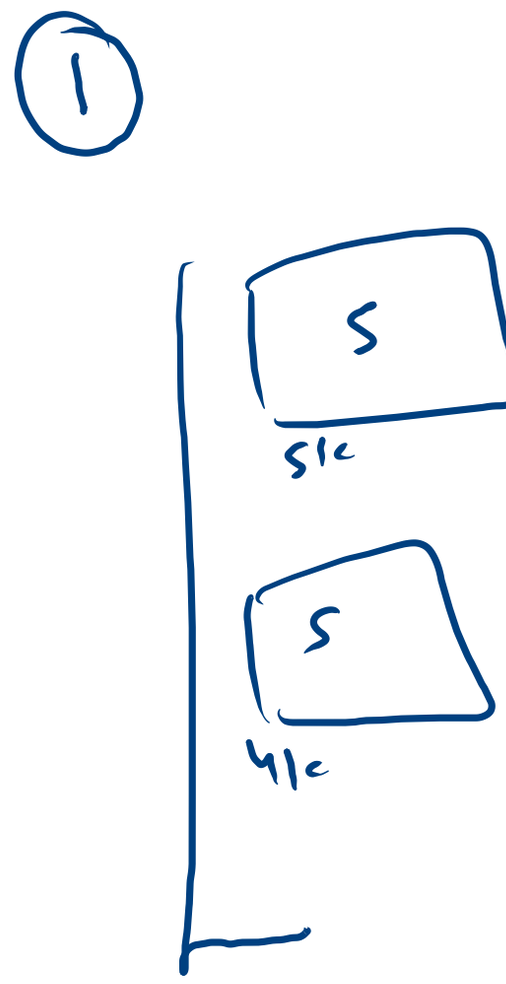
Print Reverse
Recursion

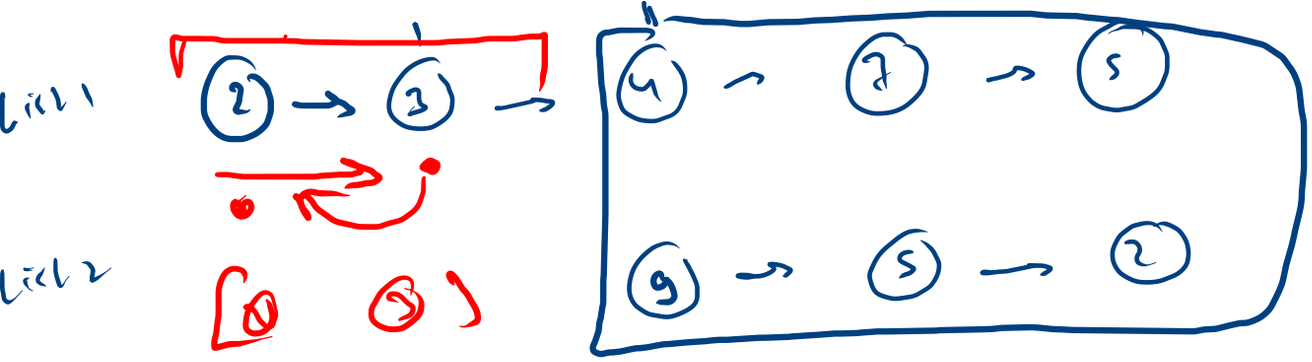


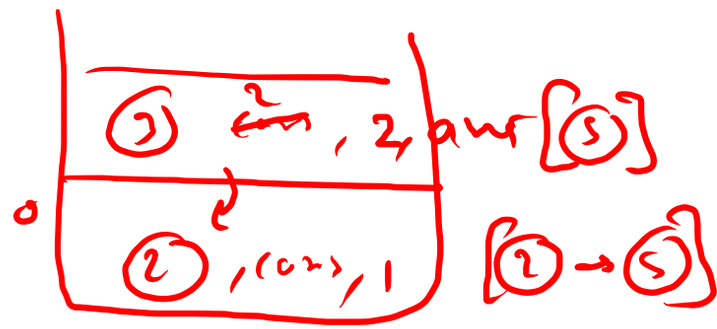

L1
L2
merge
p1
p2
k

b

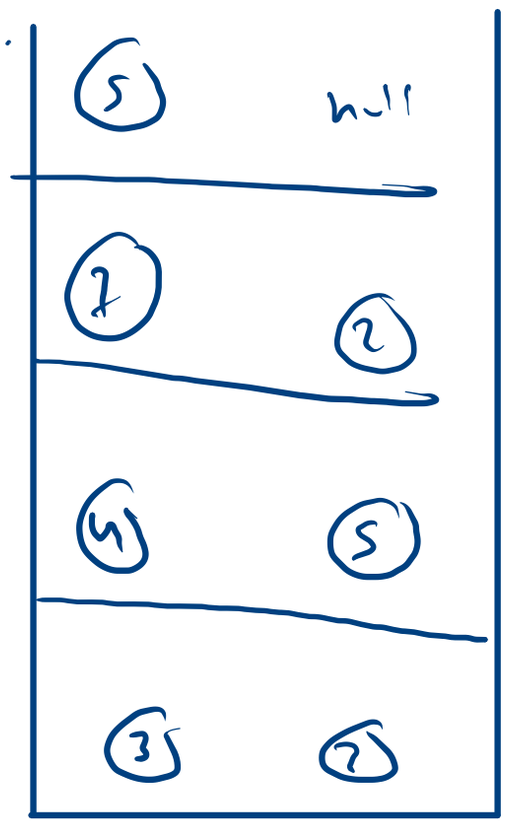






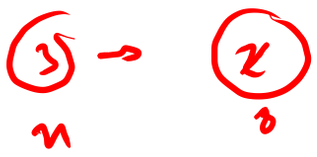


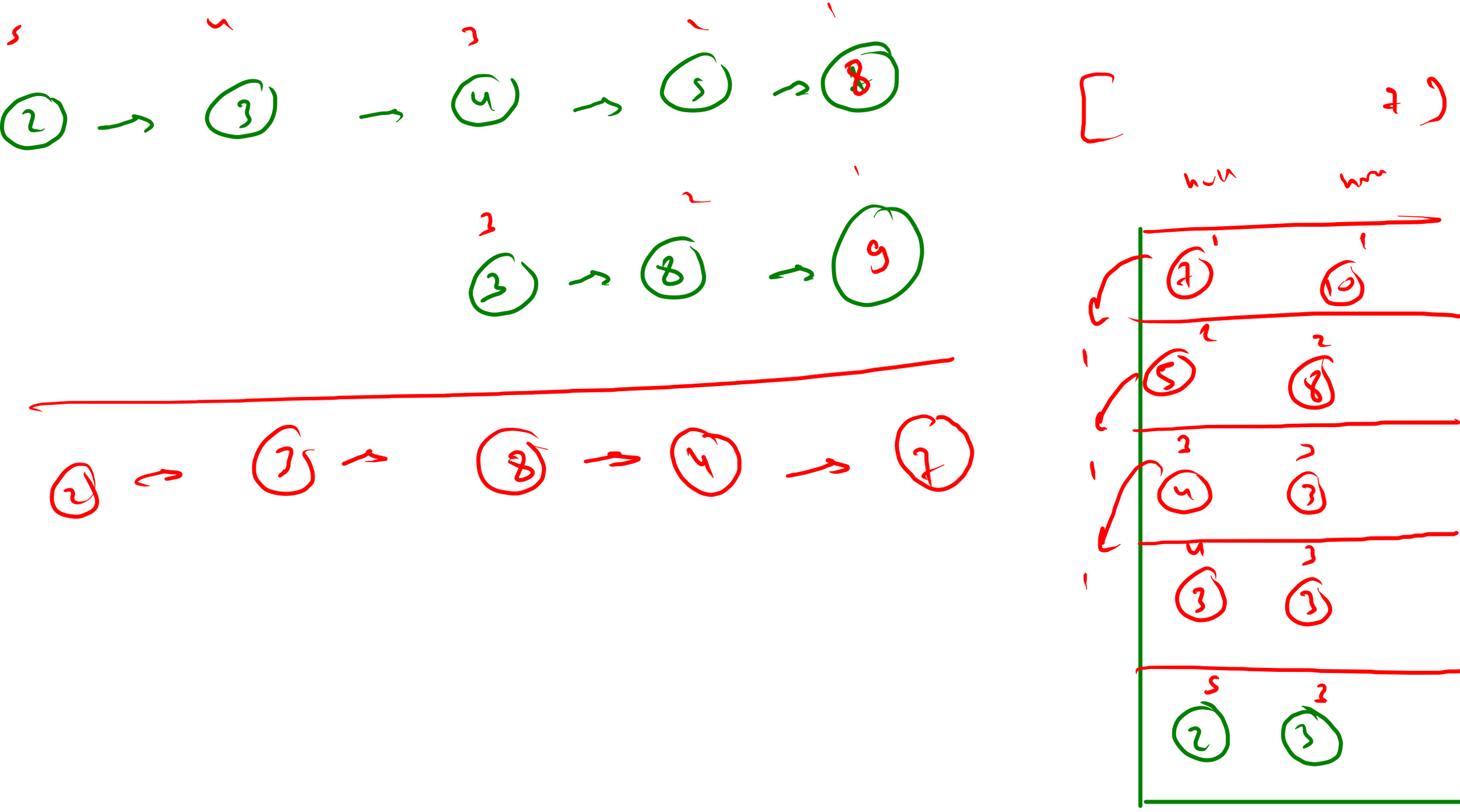
ans



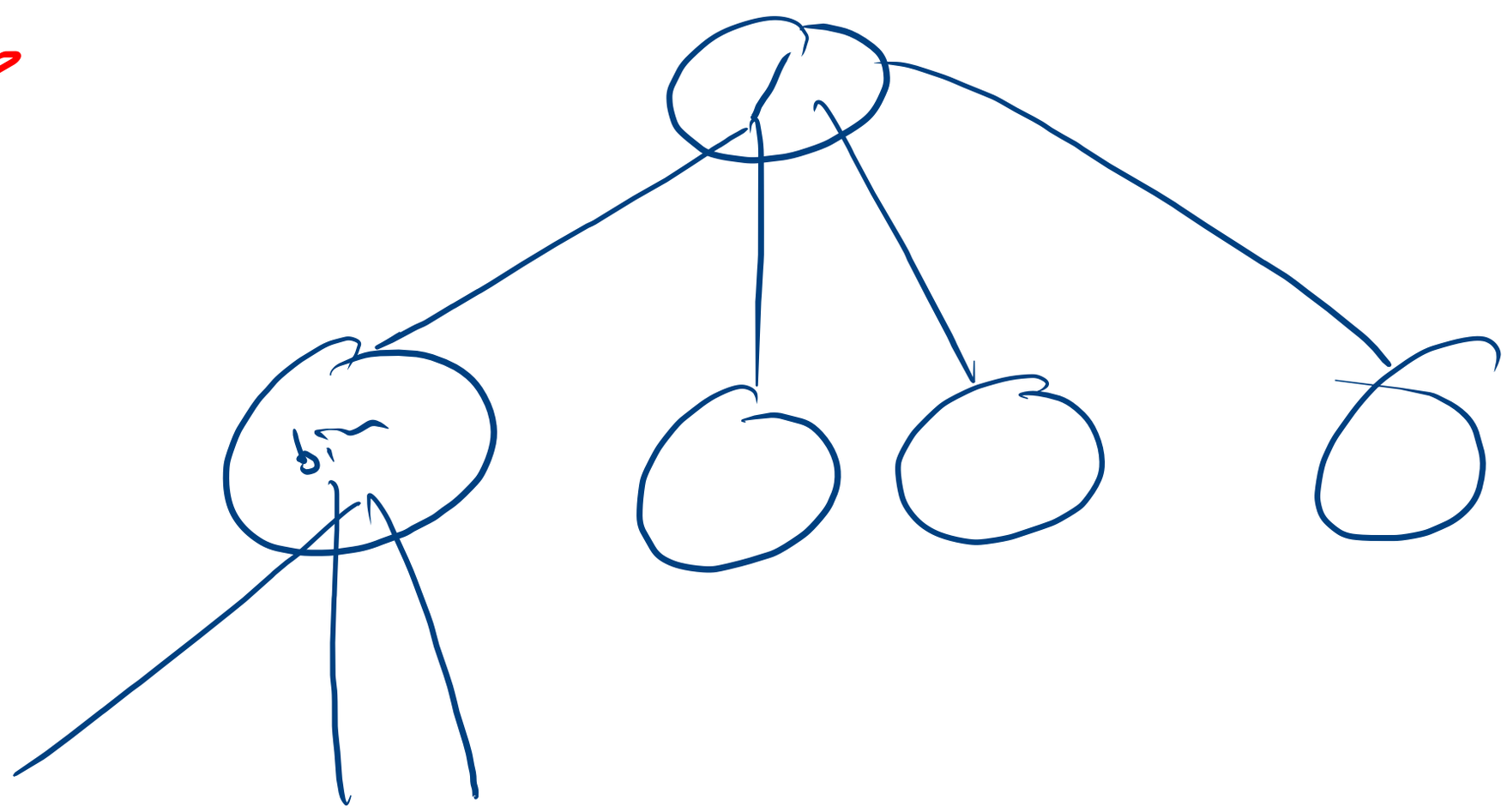
(cons)

2

handle 

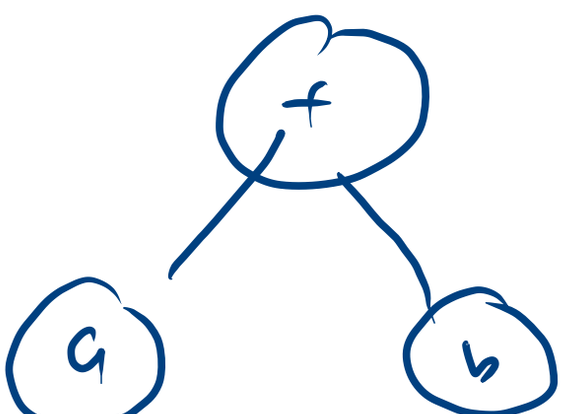


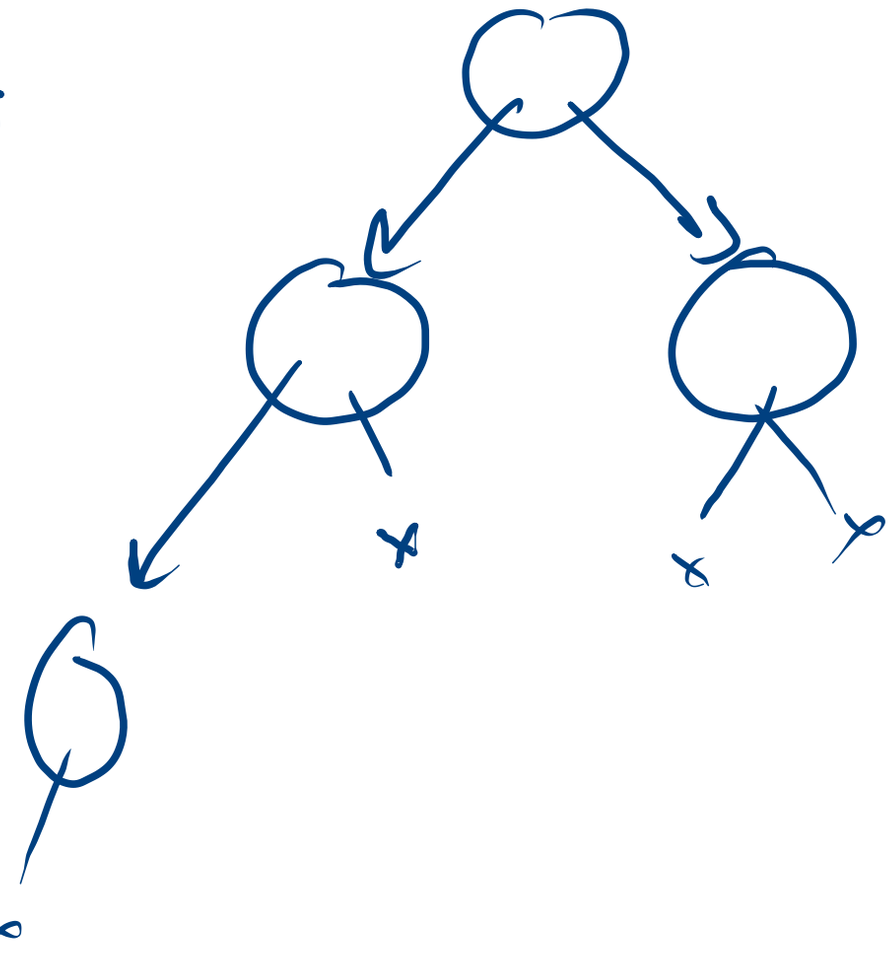
Binary Tree

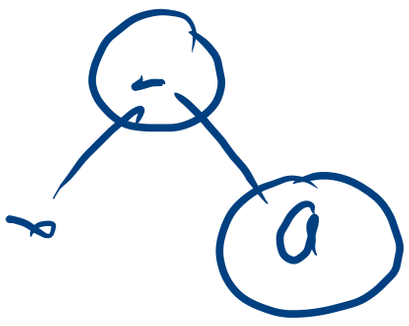



a
f
b


-b

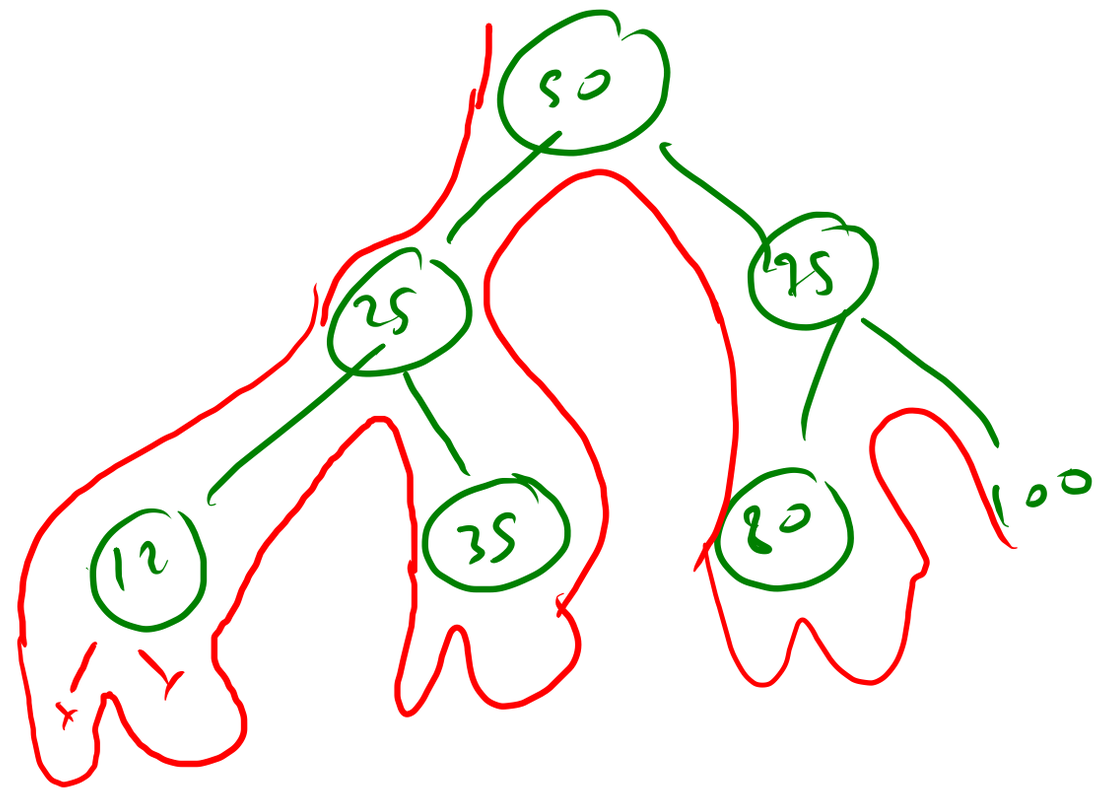

f
g
b


x
x
y

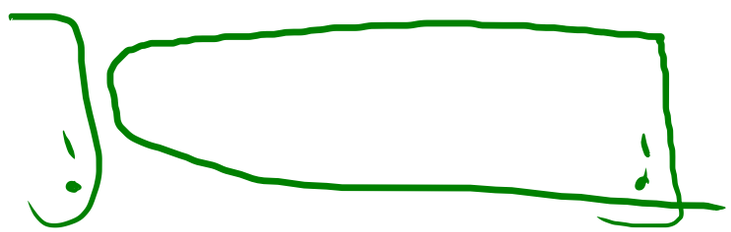

-
b
a

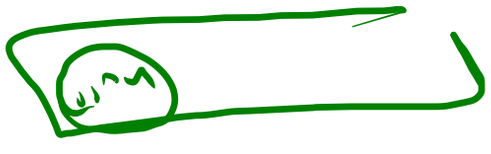
int data
Node next

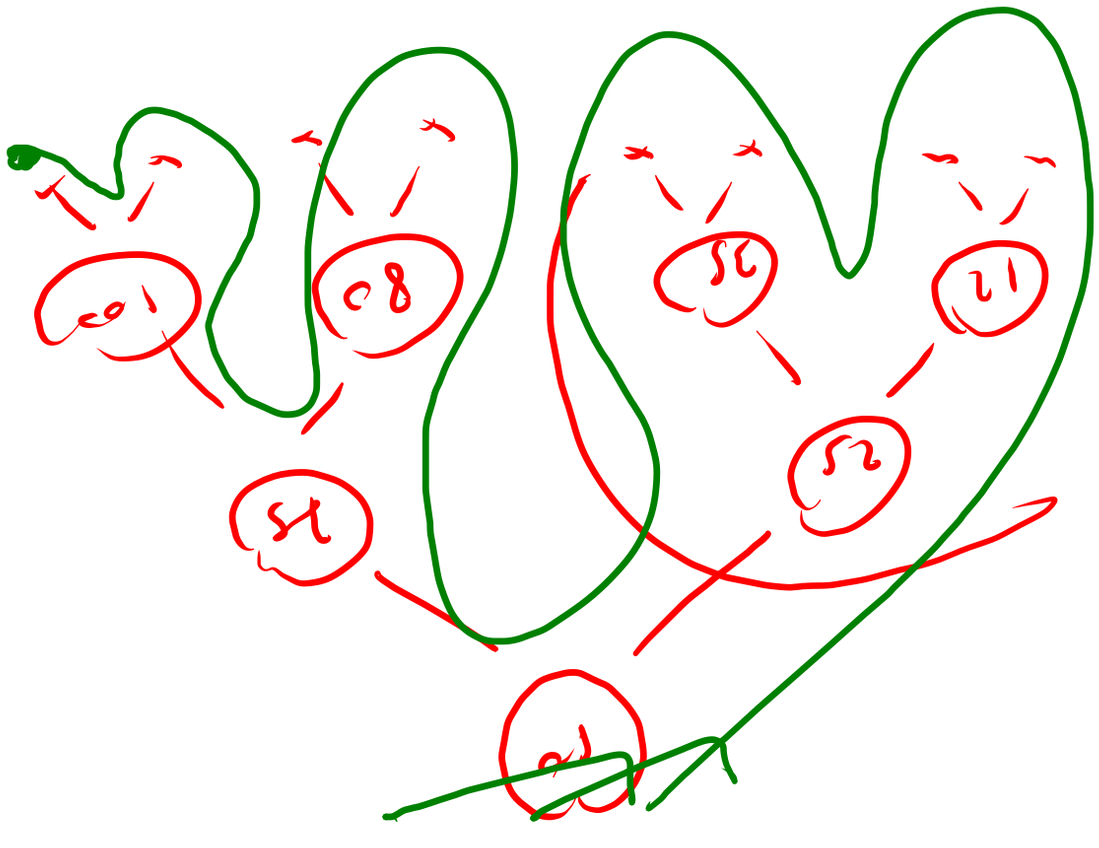
int data
Node left
Node right

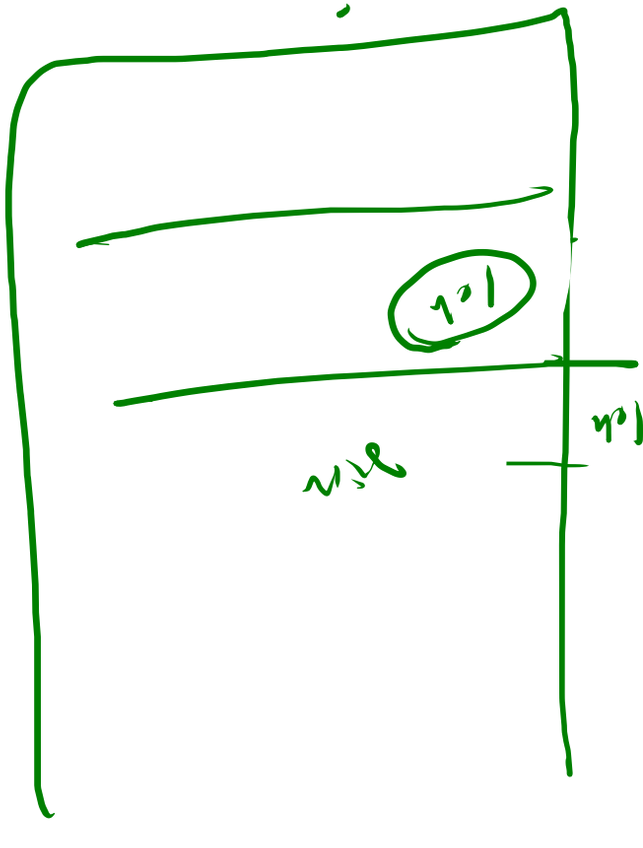


50 25 12 next 35 next 75 20
next next 100 next

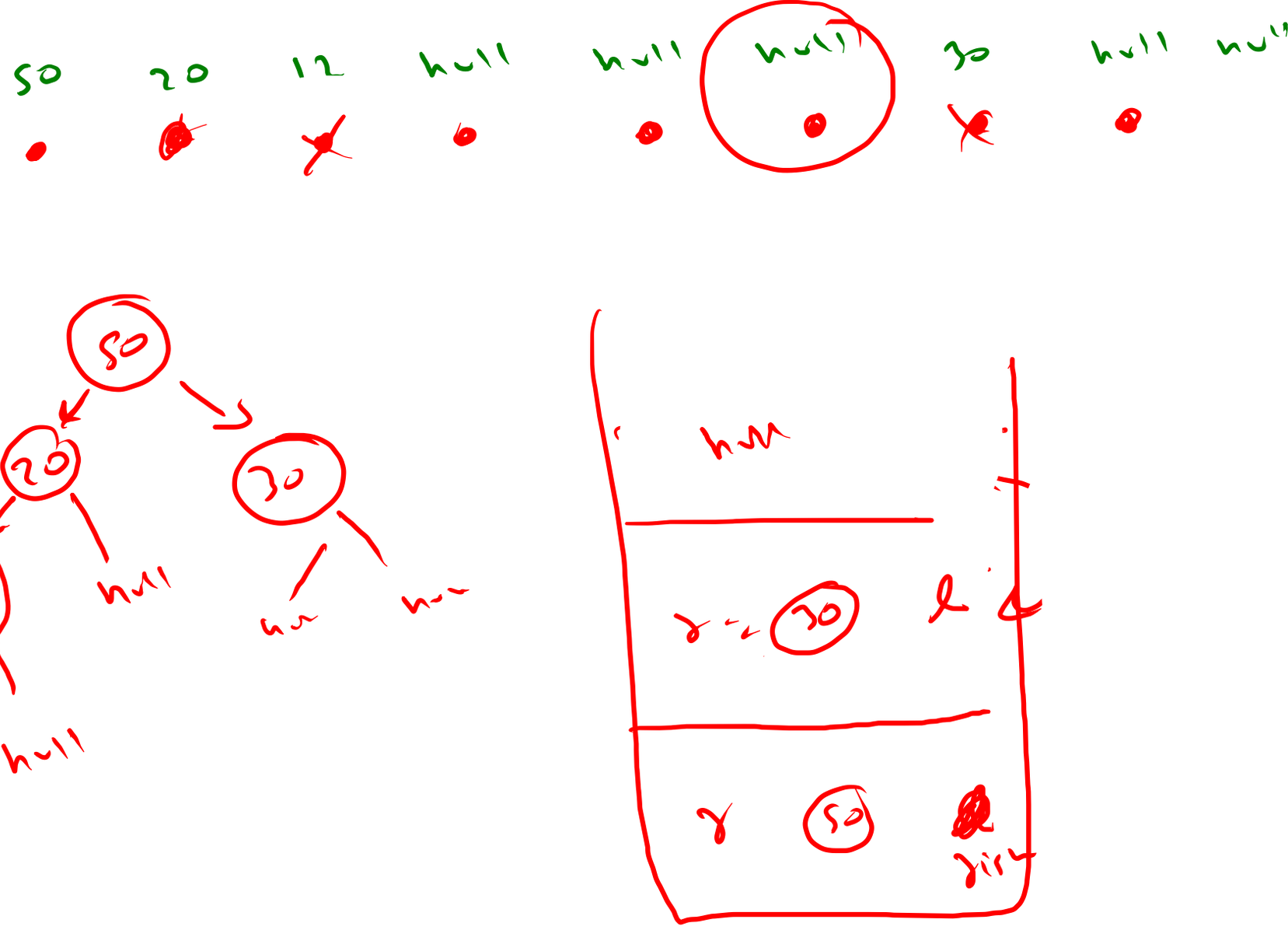


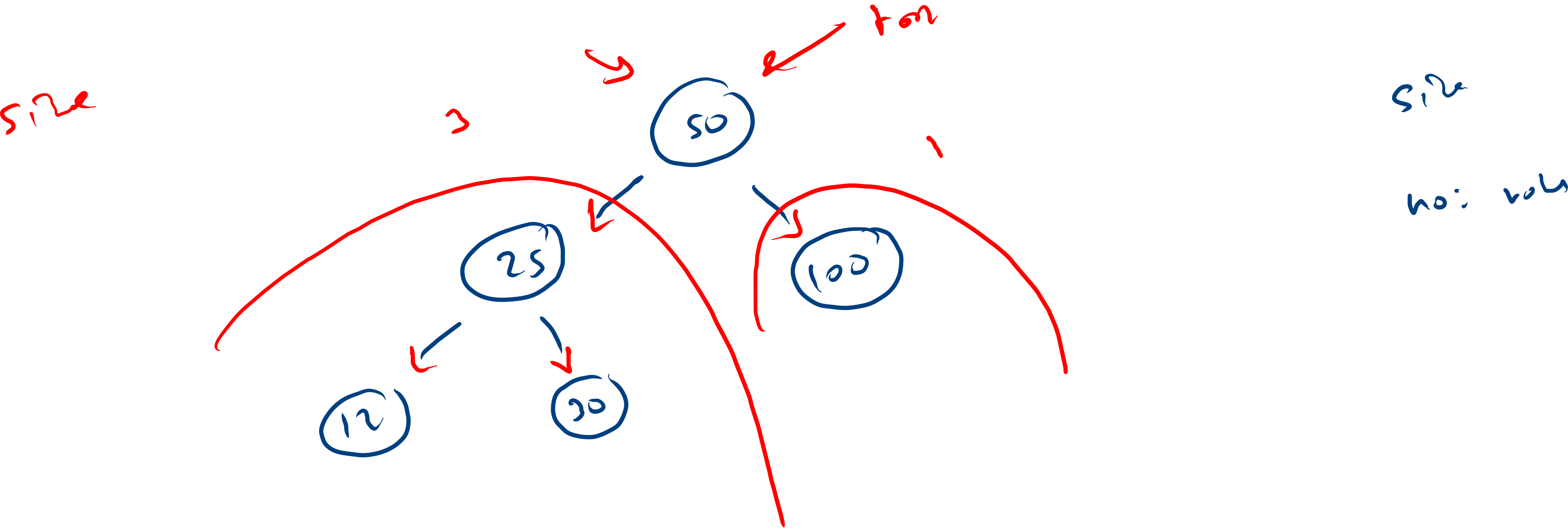


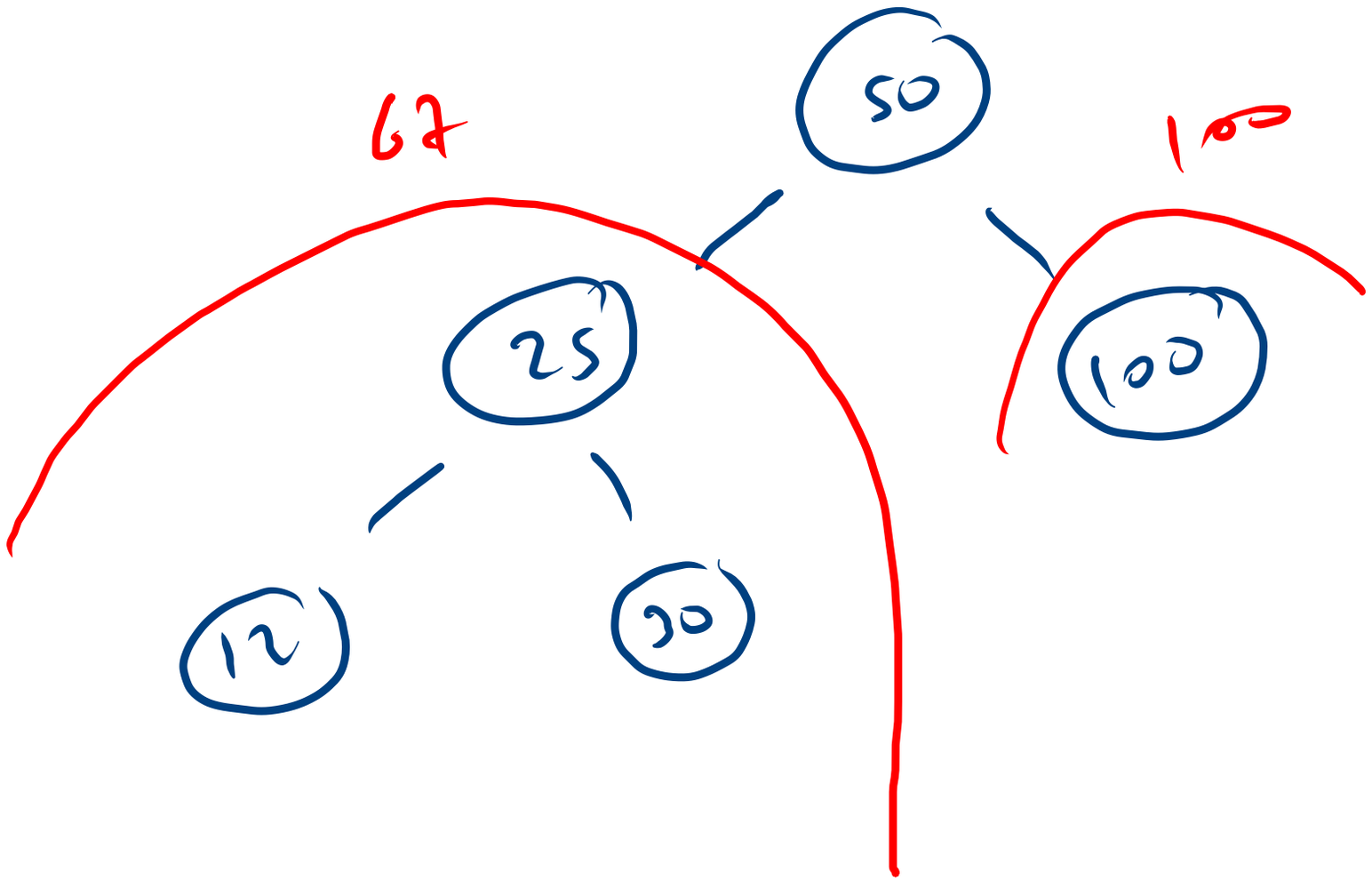


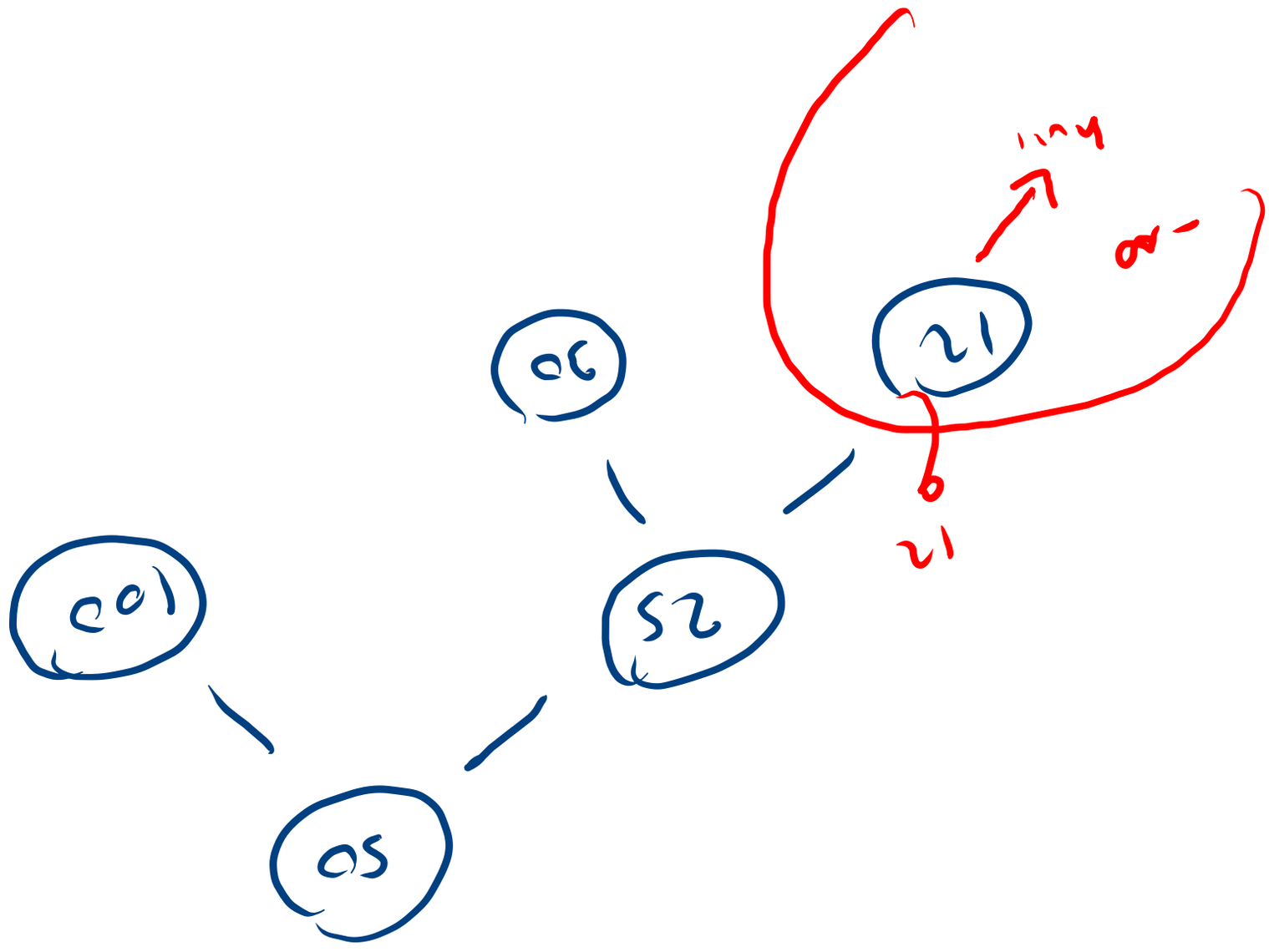


```java
static Integer data[] = new Integer[]{50,
static int i=0;

public static Node build(){
    if(data[i] == null){
        i++;
        return null;
    }


    Node r = new Node();
    r.data = data[i];
    i++;
    r.left = build();
    r.right = build();
    return r;
}

Run | Debug
public static void main(String[] args) {
```




size
3
50
25
100
1
12
30
size

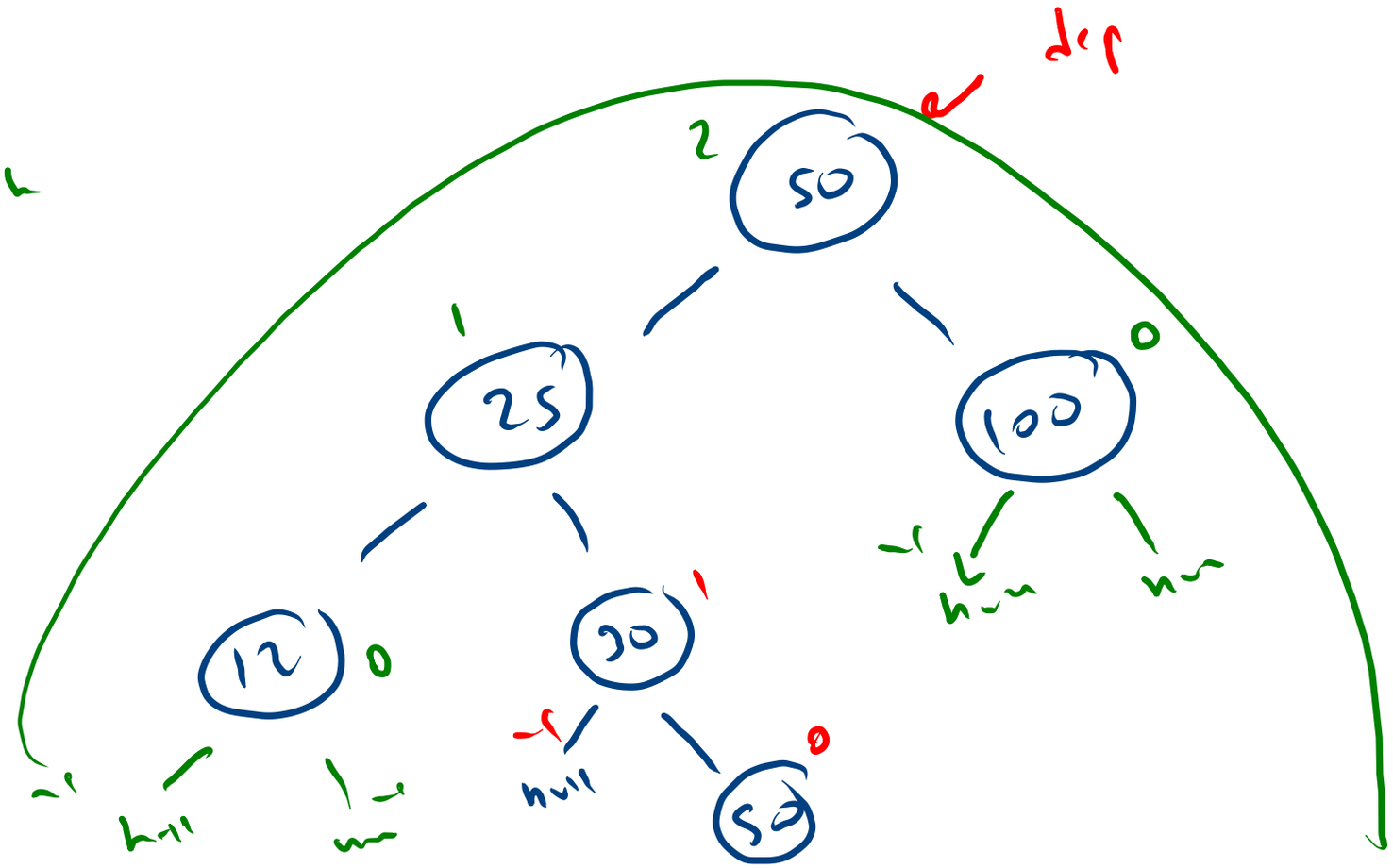

50
25
100
12
30


α_0
α_1
α_2
α_3
α_4
α_5
α_6
α_7

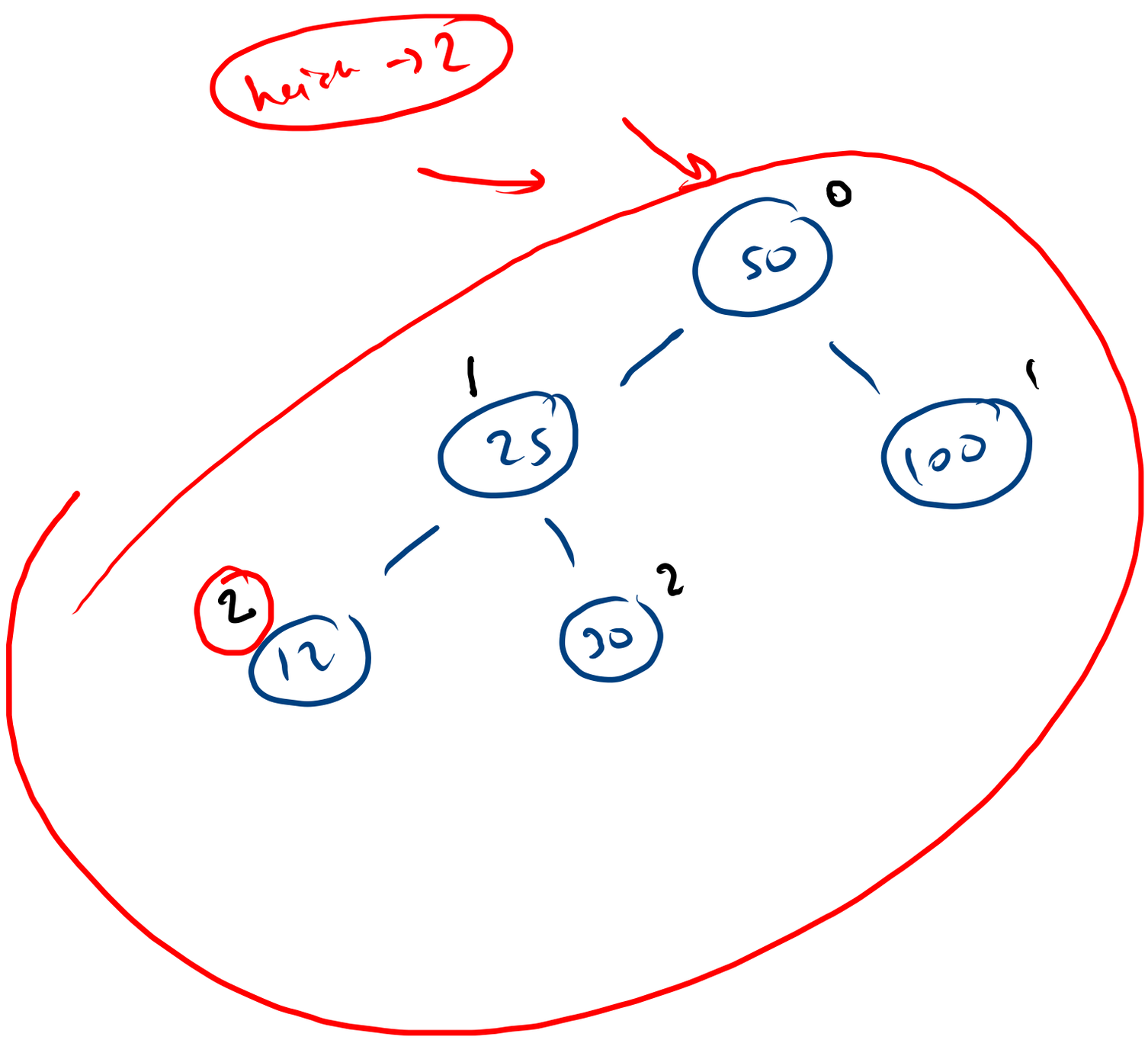
depth

height = max depth

$1 + \max(lh, rh)$



depth

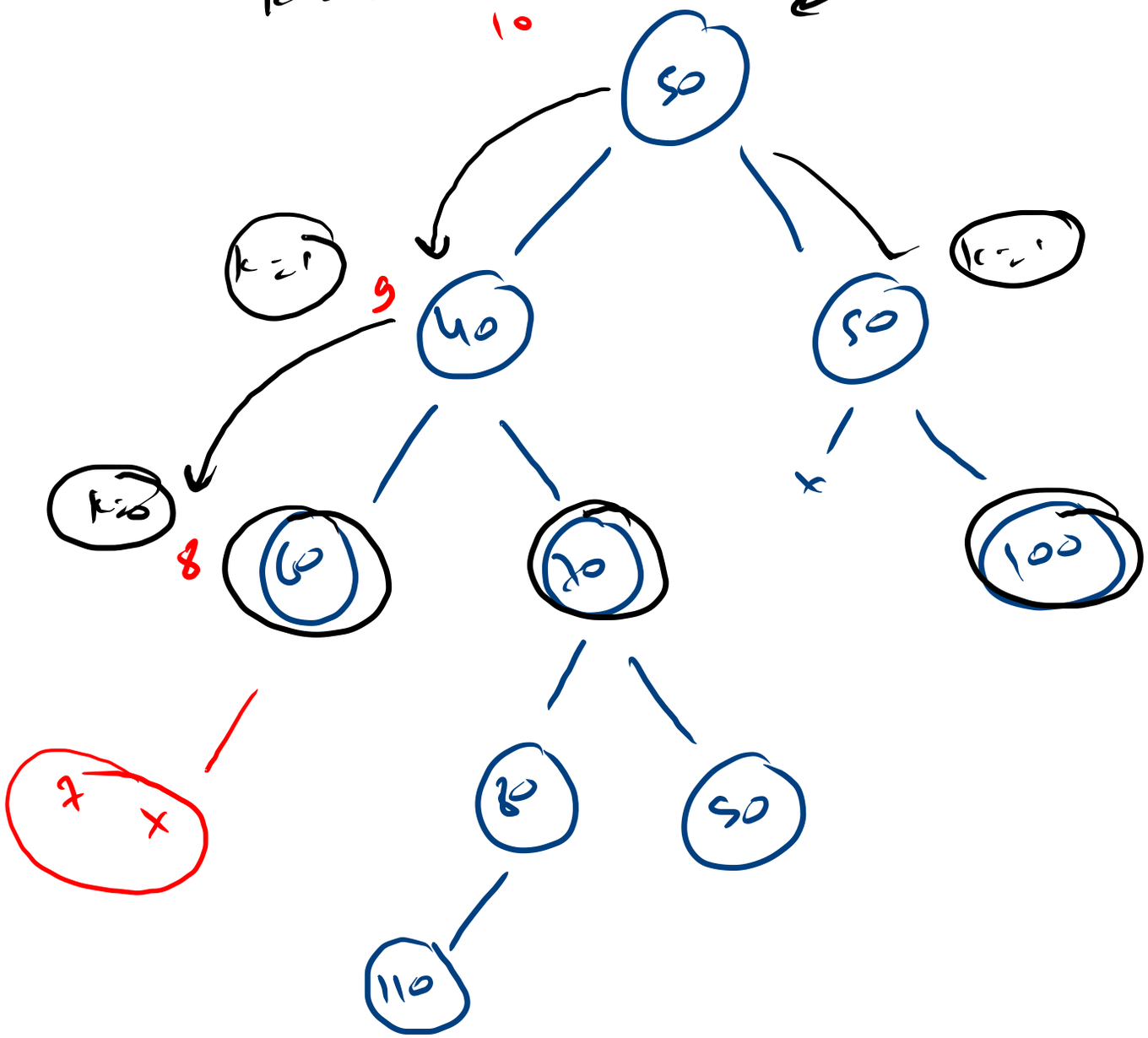


k=2 10

k=2

root

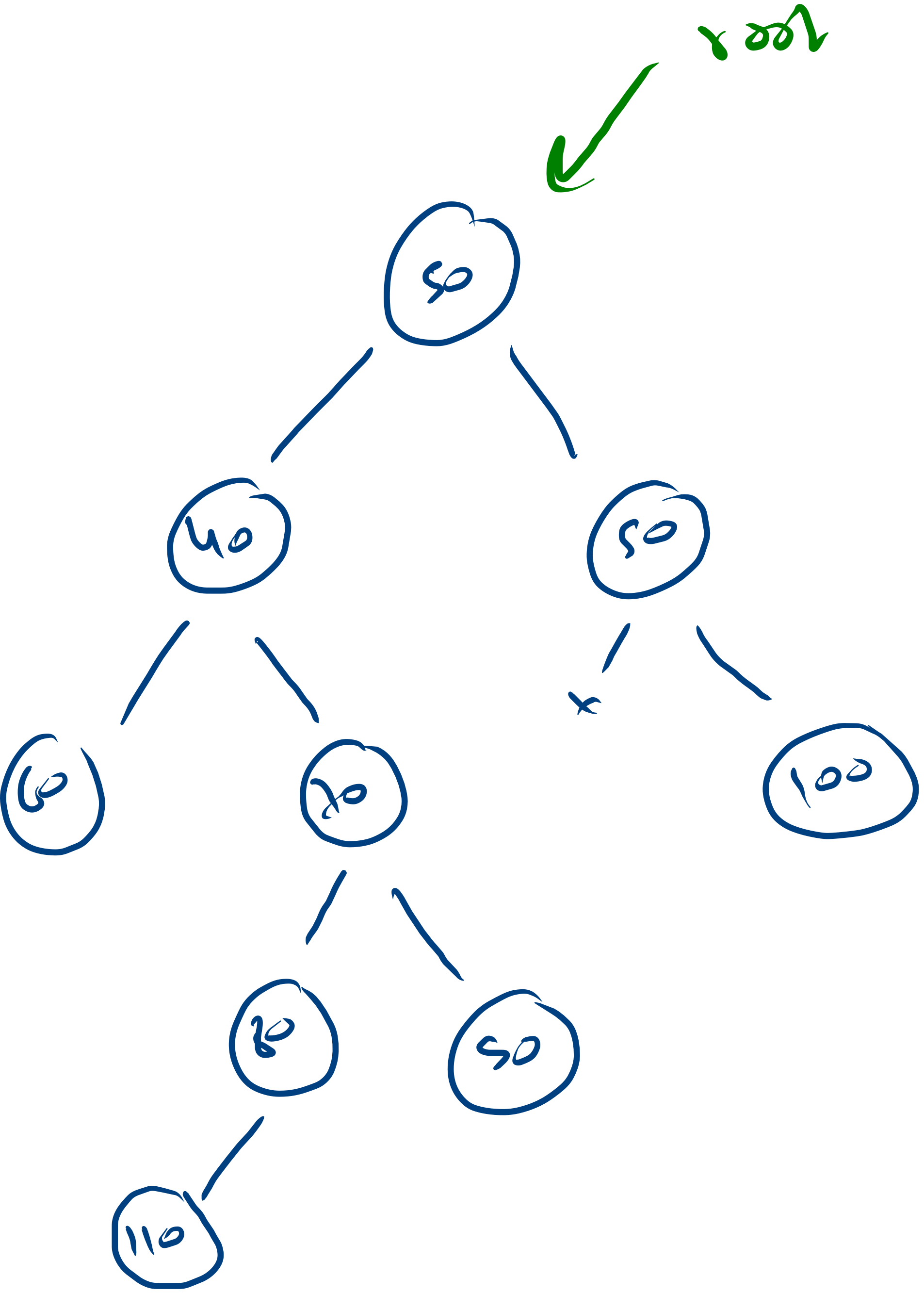
10



k = 2

60 70 100

100 70 60



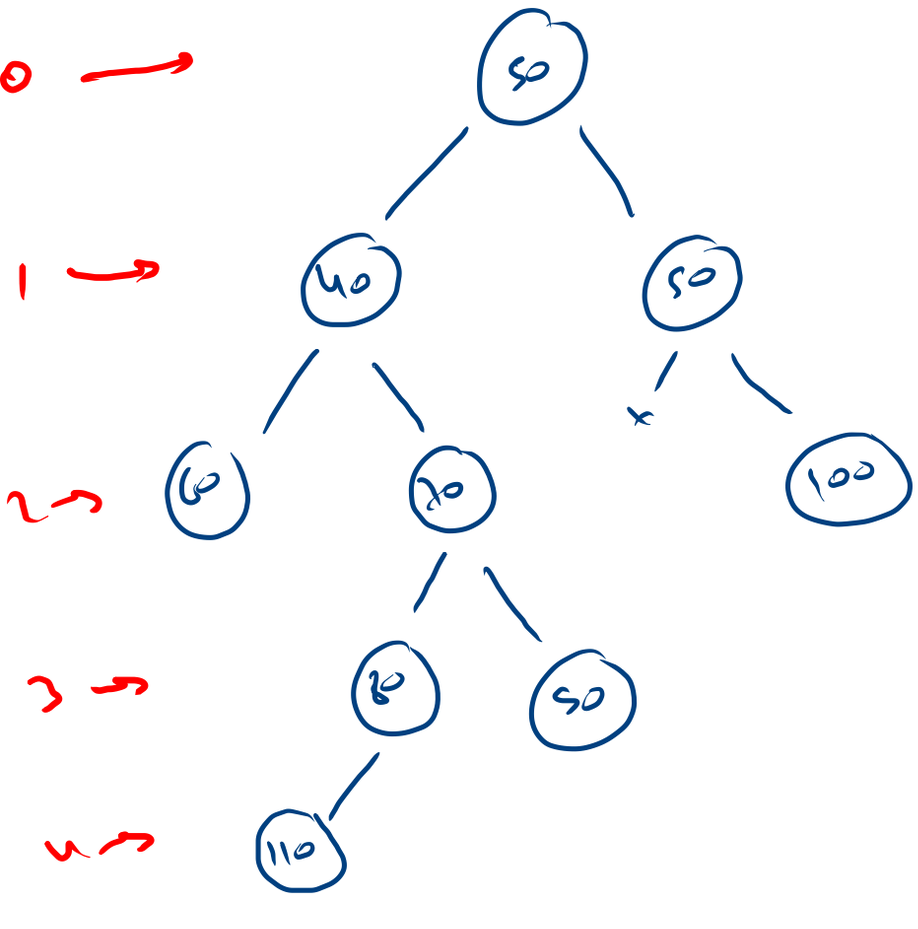
50

40 50

60 70 100

80 90

110



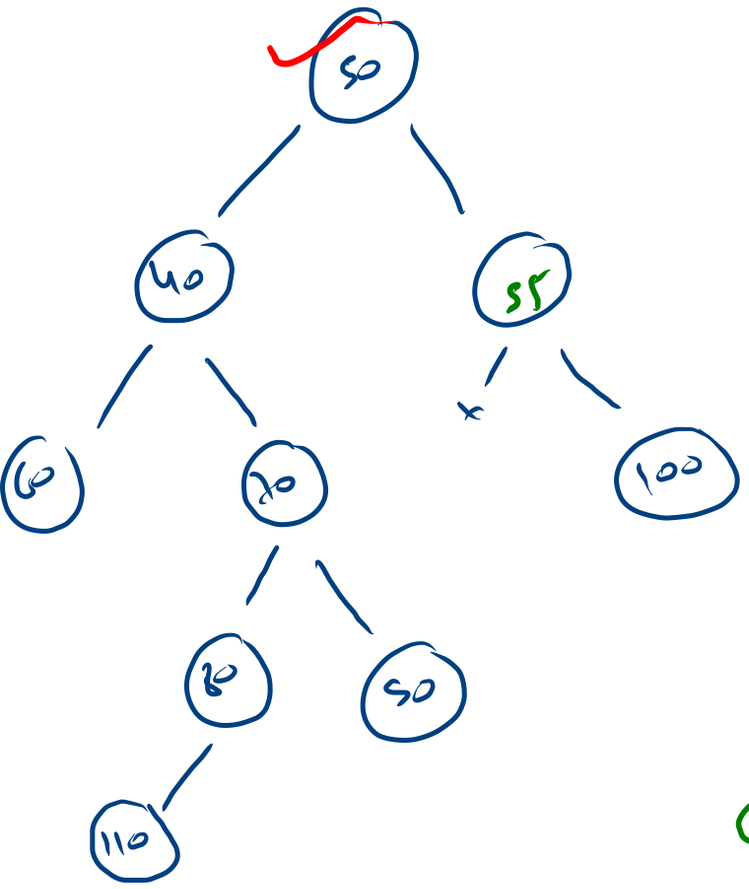
50

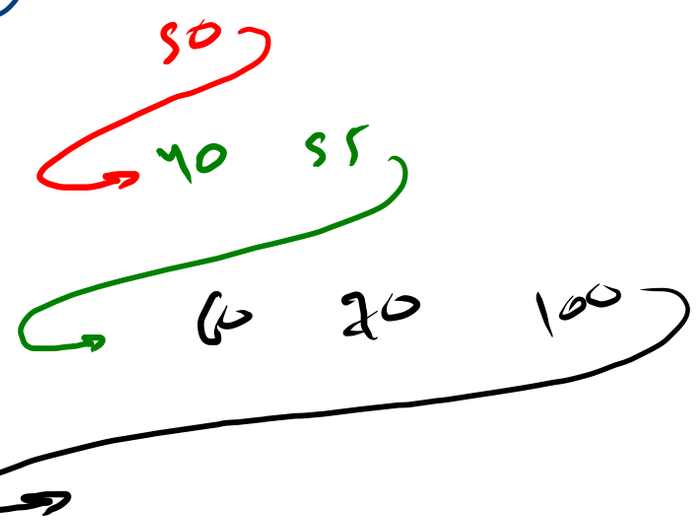
40 50

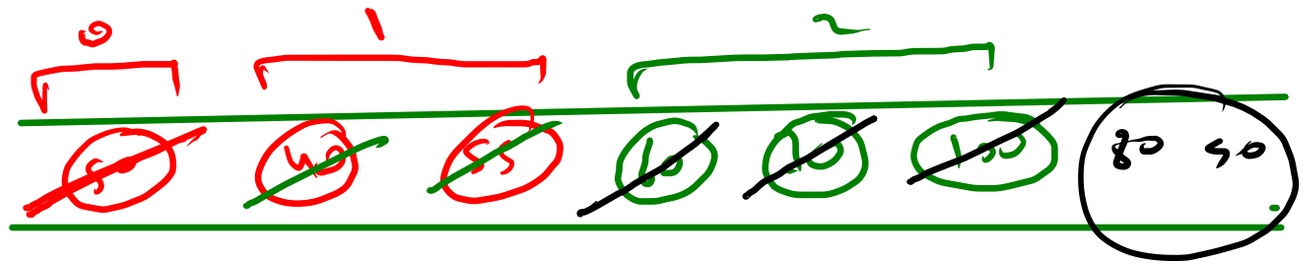
60 70 100

all (root)

count =







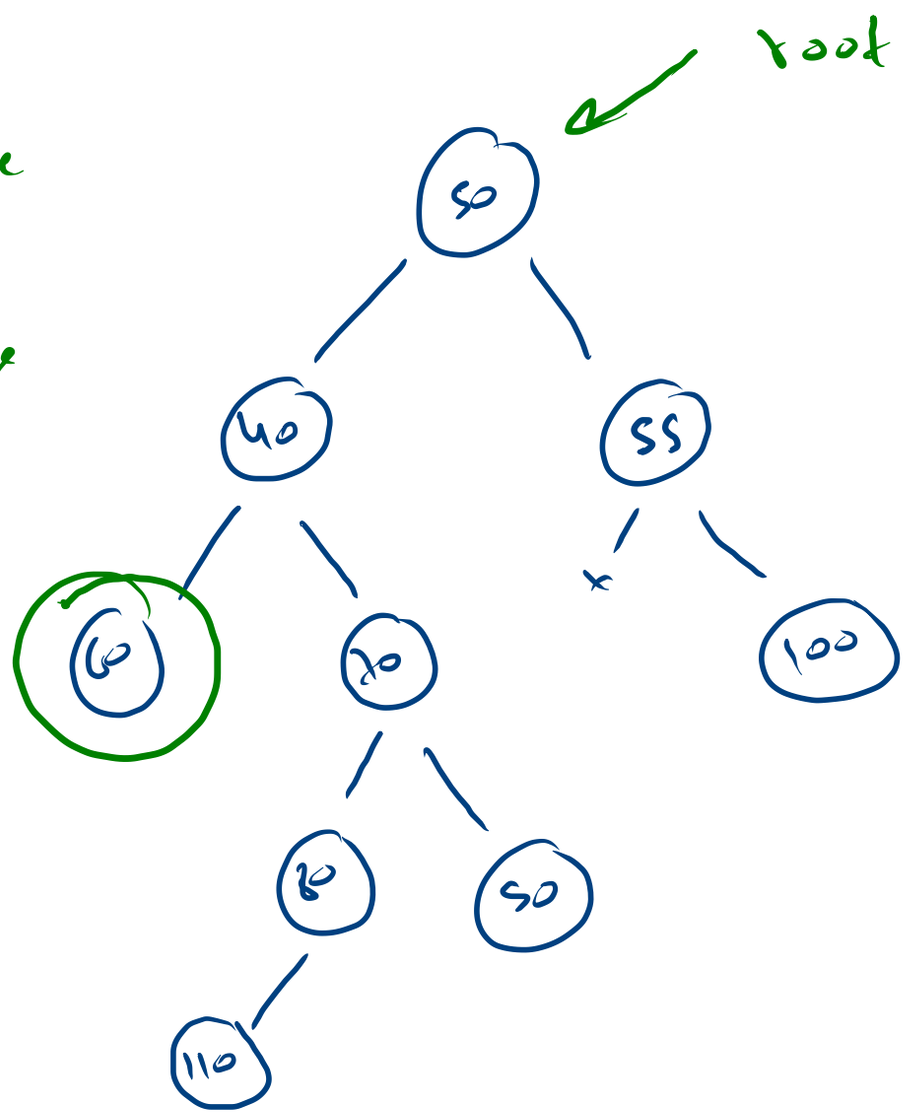
Curr = 3

```
Queue<Node> q = new LinkedList<>();

q.add(node);


while(q.size() > 0){

    int count = q.size();

    for(int i=0;i<count;i++){
        Node r = q.remove();
        System.out.print(r.data+" ");
        if(r.left!= null)
            q.add(r.left);
        if(r.right != null)
            q.add(r.right);
    }

    System.out.println();
}
```

data → 60 True

lst → 65 False

[60, 40, 50]



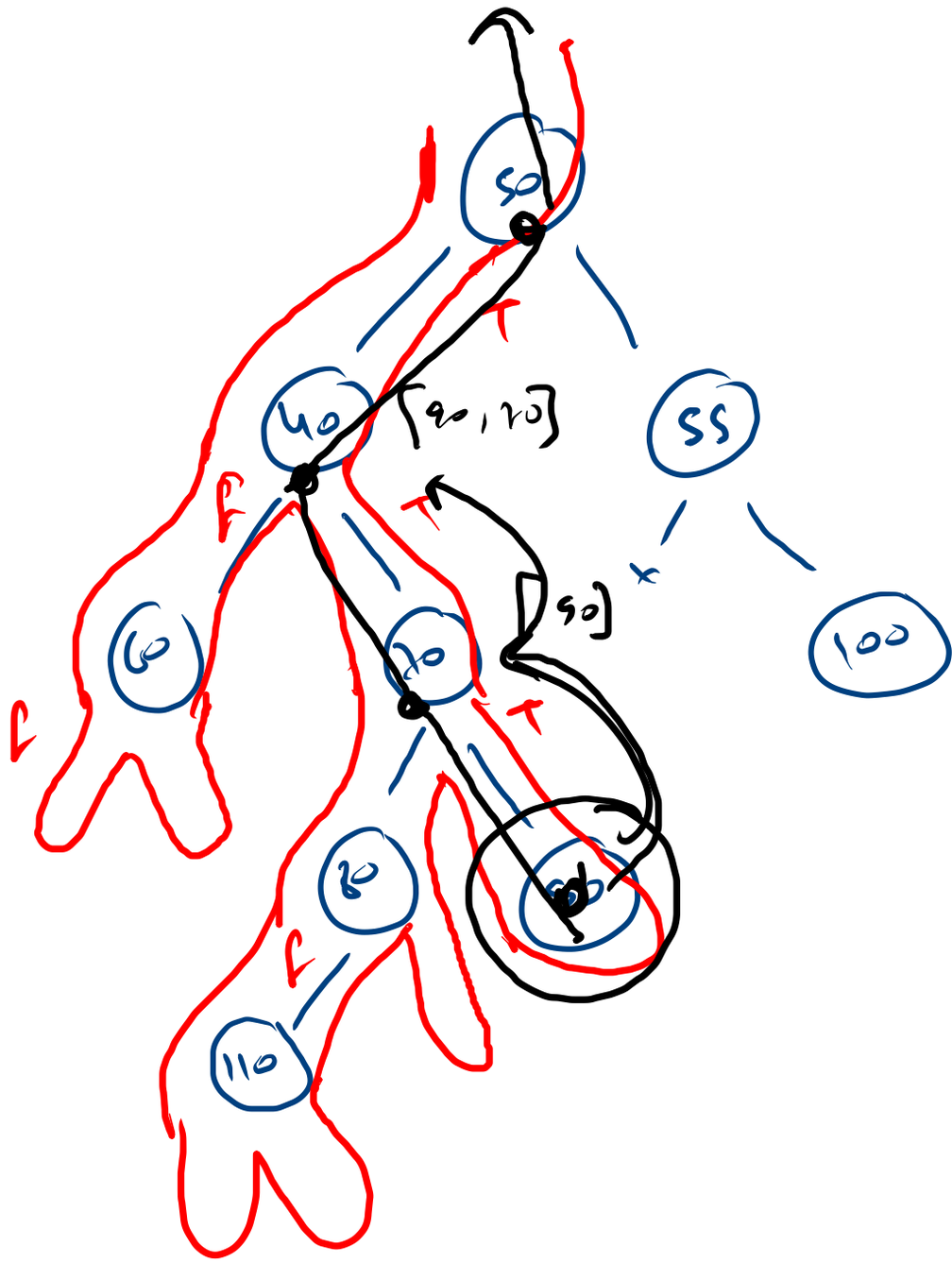
50

Node.data == data

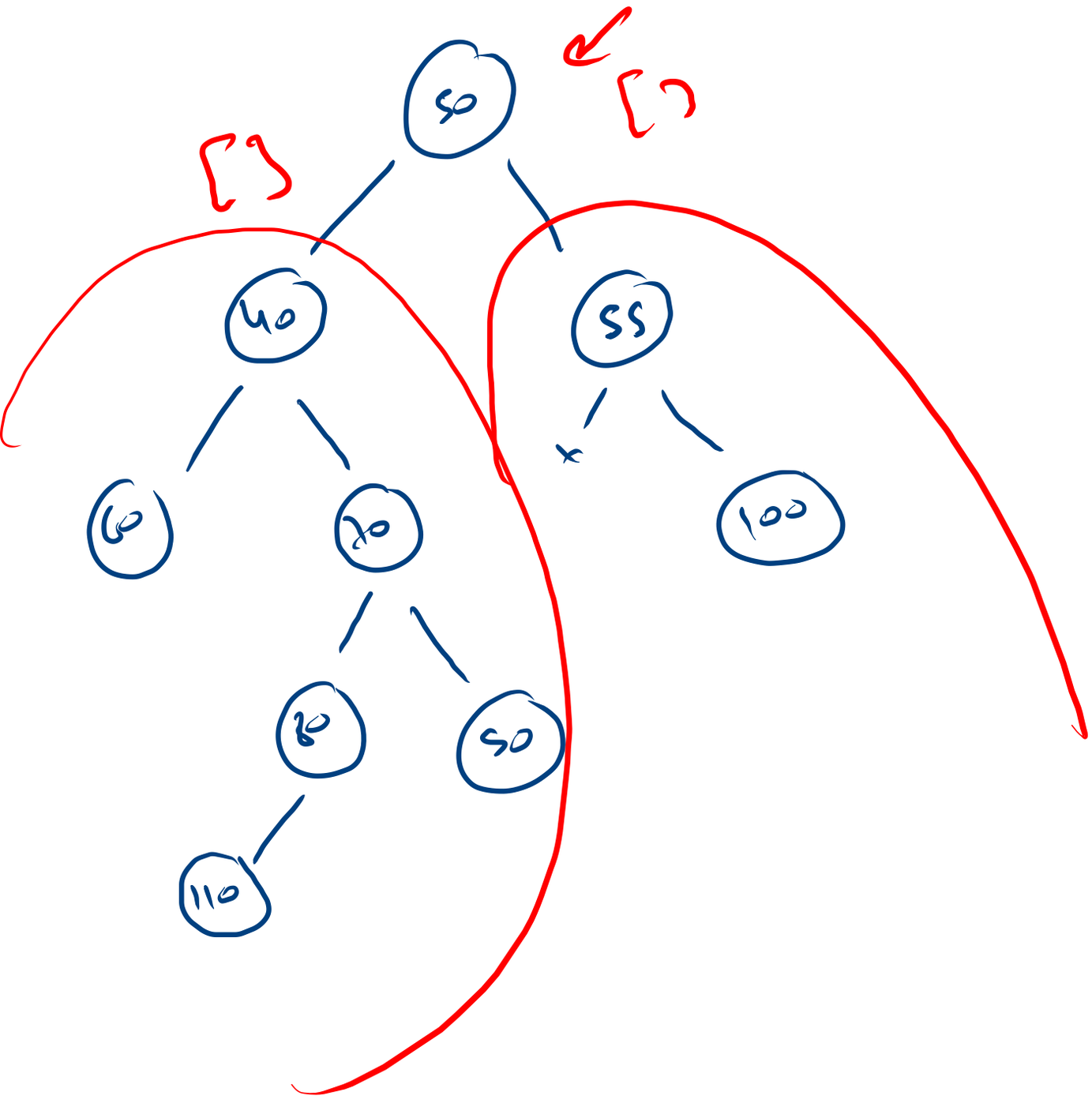
Left

right

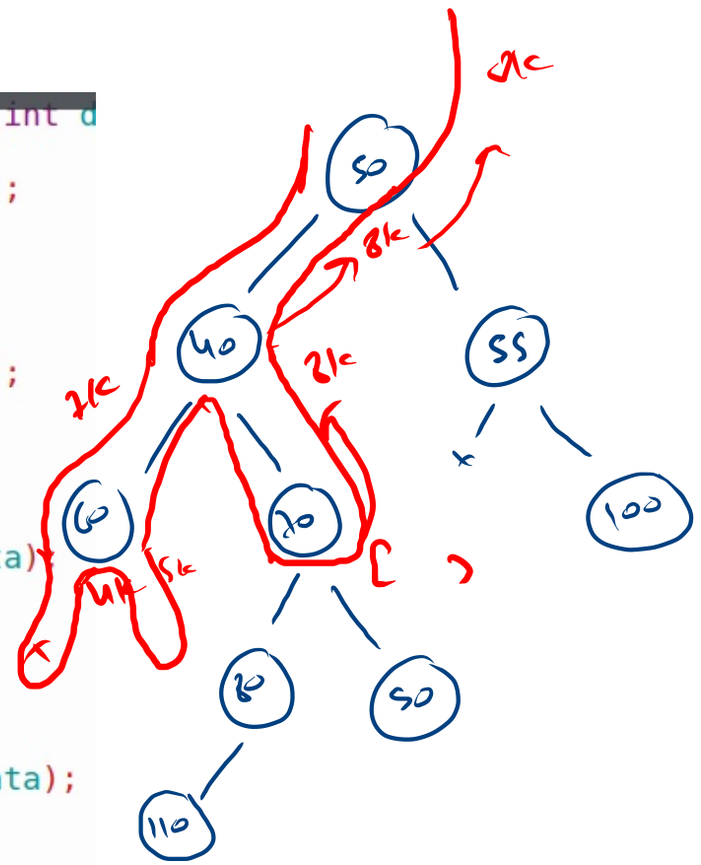
```java
public static boolean find(Node node, int data){
    if(node == null)return false;

    if(node.data == data)return true;

    boolean lf = find(node.left, data);
    if(lf)return true;

    boolean rf = find(node.right, data);

    return lf || rf;
}
```

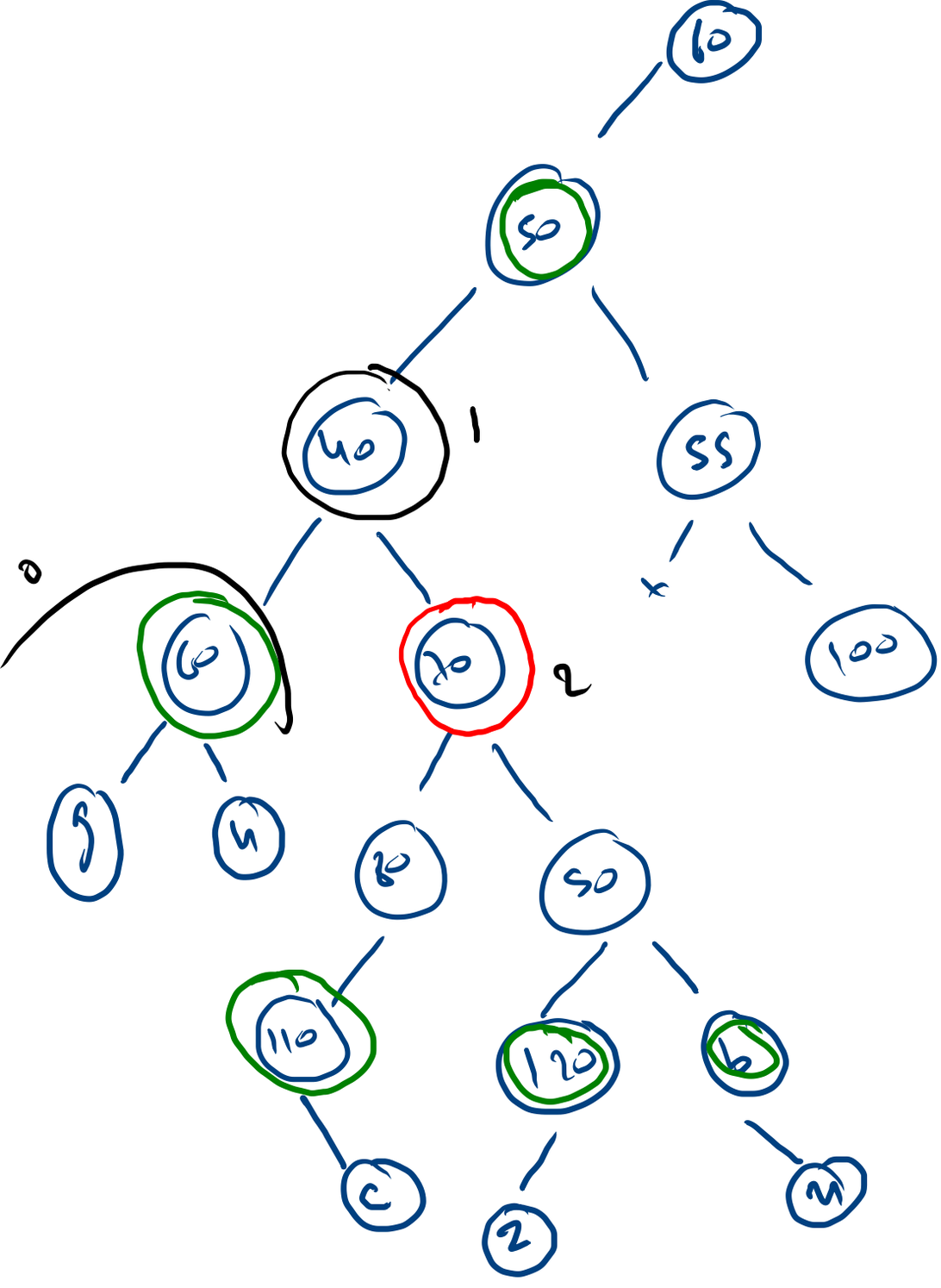


Id = 1100



```java
public static ArrayList<Integer> nodeToRootPath(Node node, int d
    if(node == null){
        ArrayList<Integer> ans = new ArrayList<Integer>();
        return ans;
    }

    if(node.data == data){
        ArrayList<Integer> ans = new ArrayList<Integer>();
        ans.add(node.data);
        return ans;
    }

    ArrayList<Integer> lf = nodeToRootPath(node.left, data);
    if(lf.size() > 0){
        lf.add(node.data);
        return lf;
    }

    ArrayList<Integer> rf = nodeToRootPath(node.right, data);

    if(rf.size() > 0){
        rf.add(node.data);
        return rf;
    }

    return new ArrayList<>();
}
```





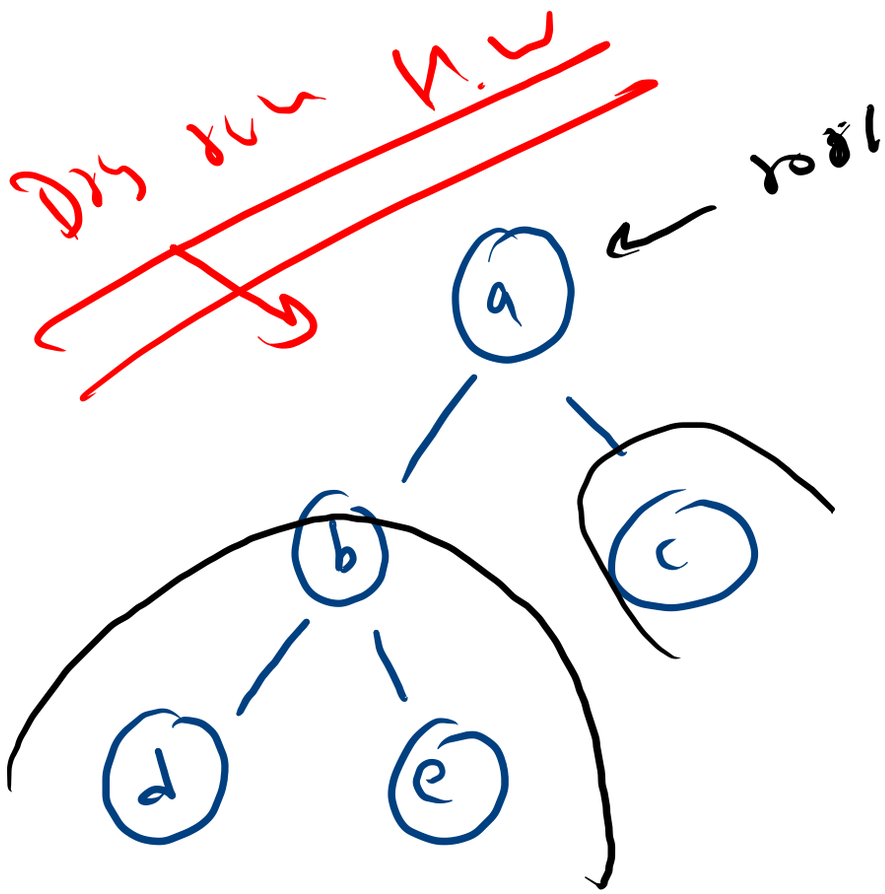
data → 70

k = 2

110
120
6
60
50

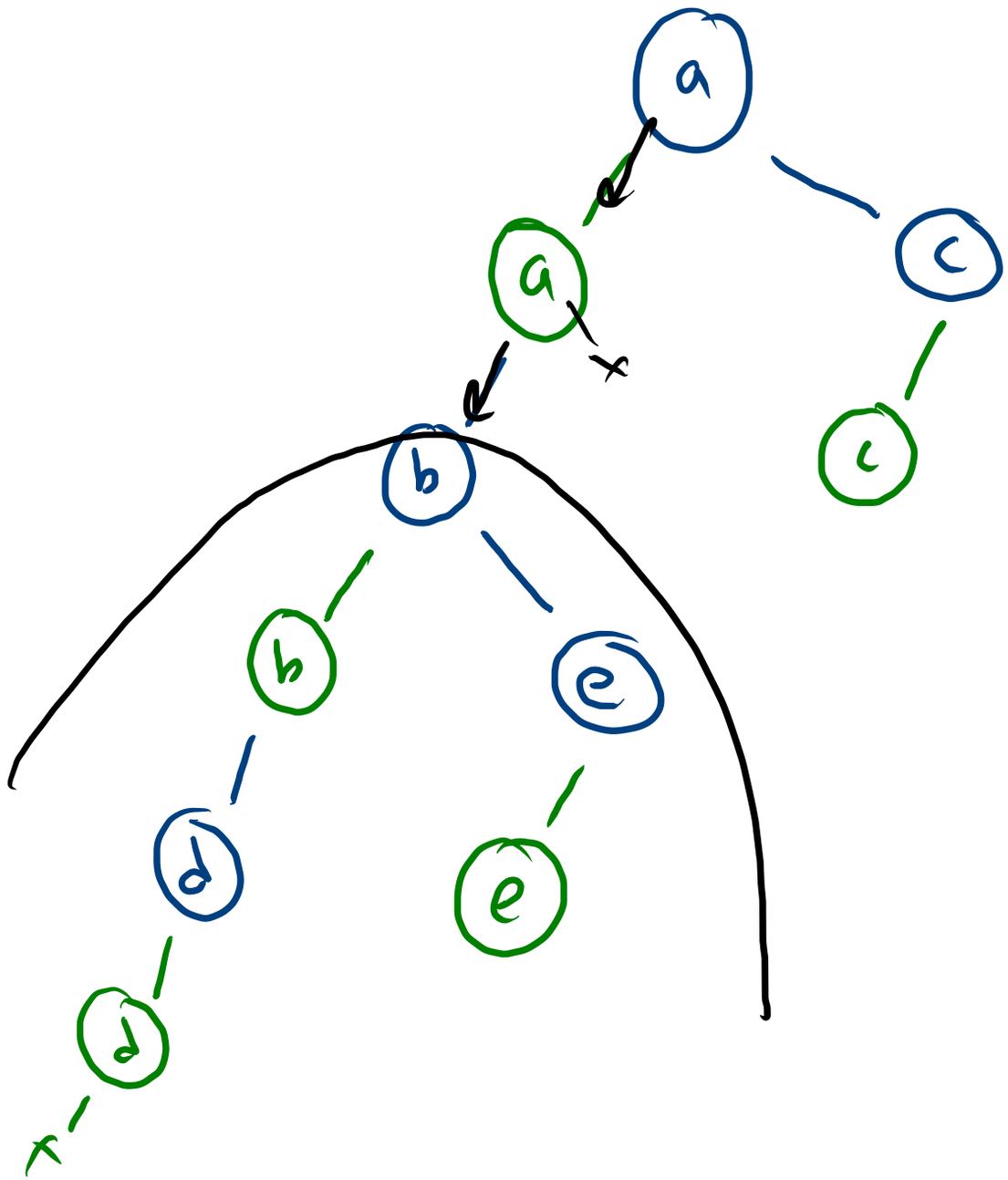
2 1 0

Dry run kr lo



node clone = new node();

clone.left = RLC(node.left)

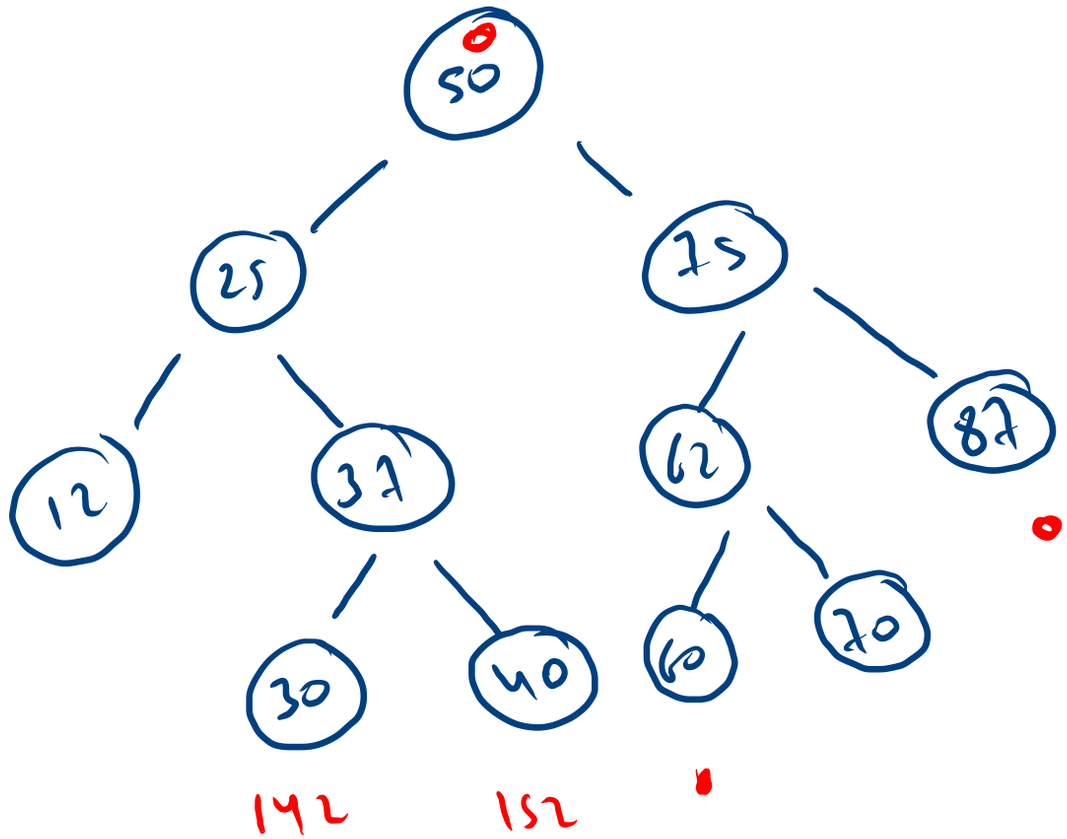


Transform back

H.W

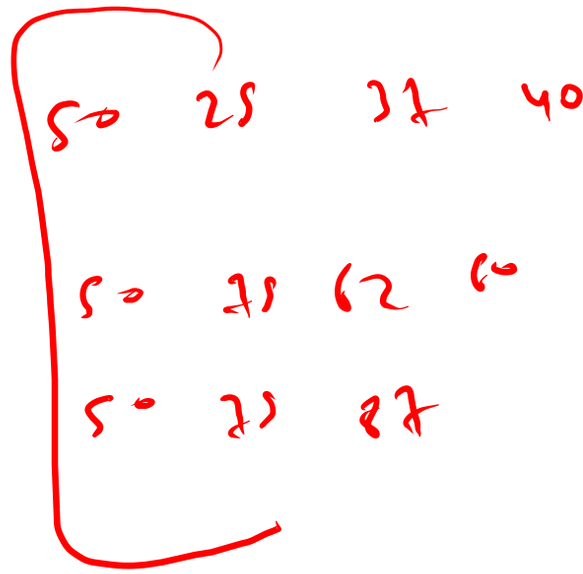
150

250

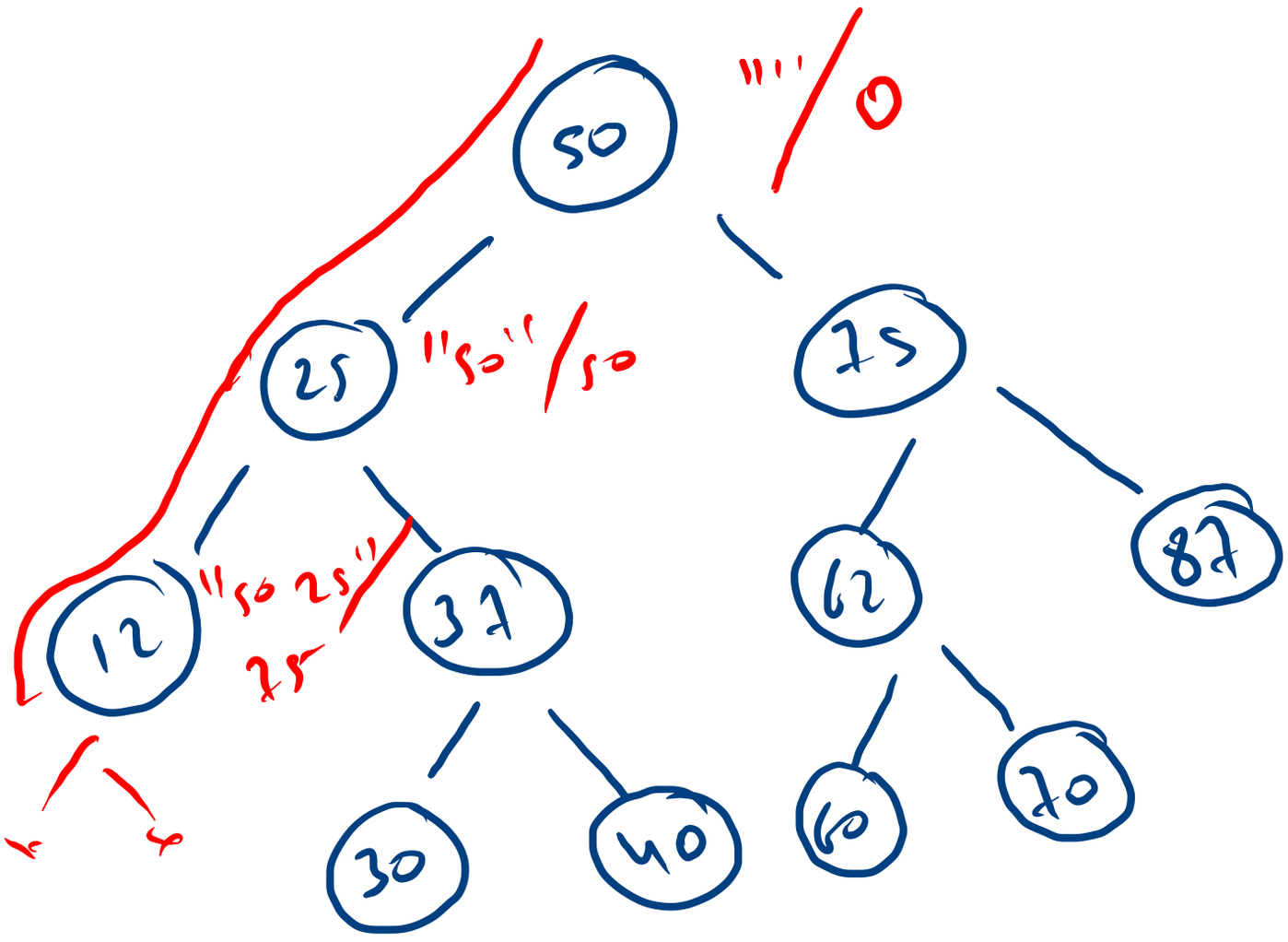


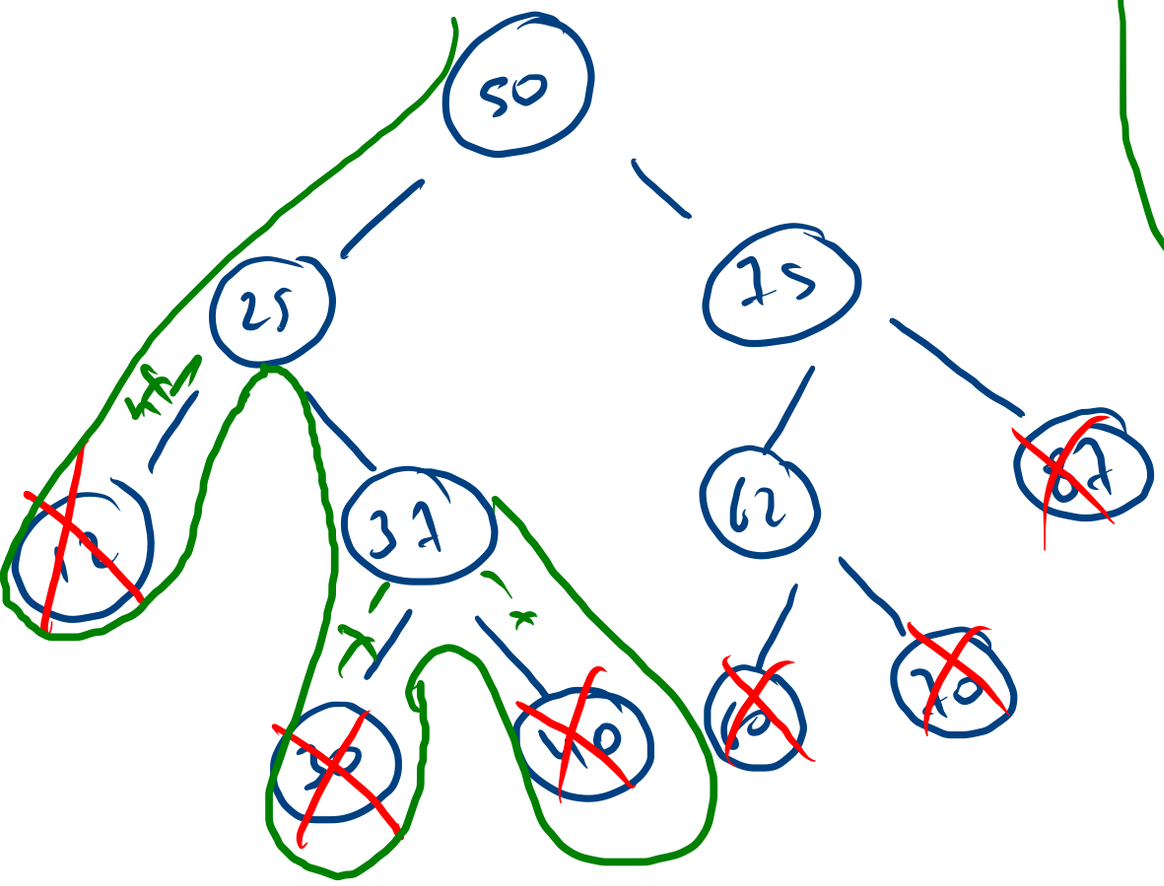
142 152

75
67

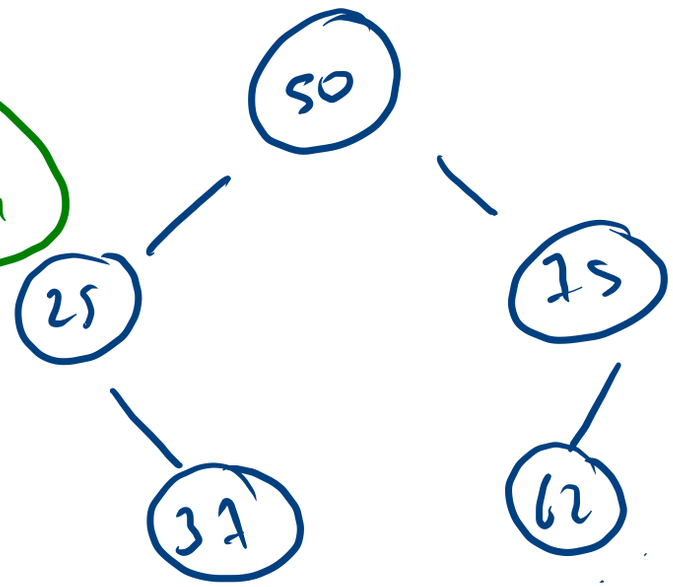


50 25 12 n n 37
30 n n 40 n n 75
62 60 n n 70 n n
87 n n 150 250

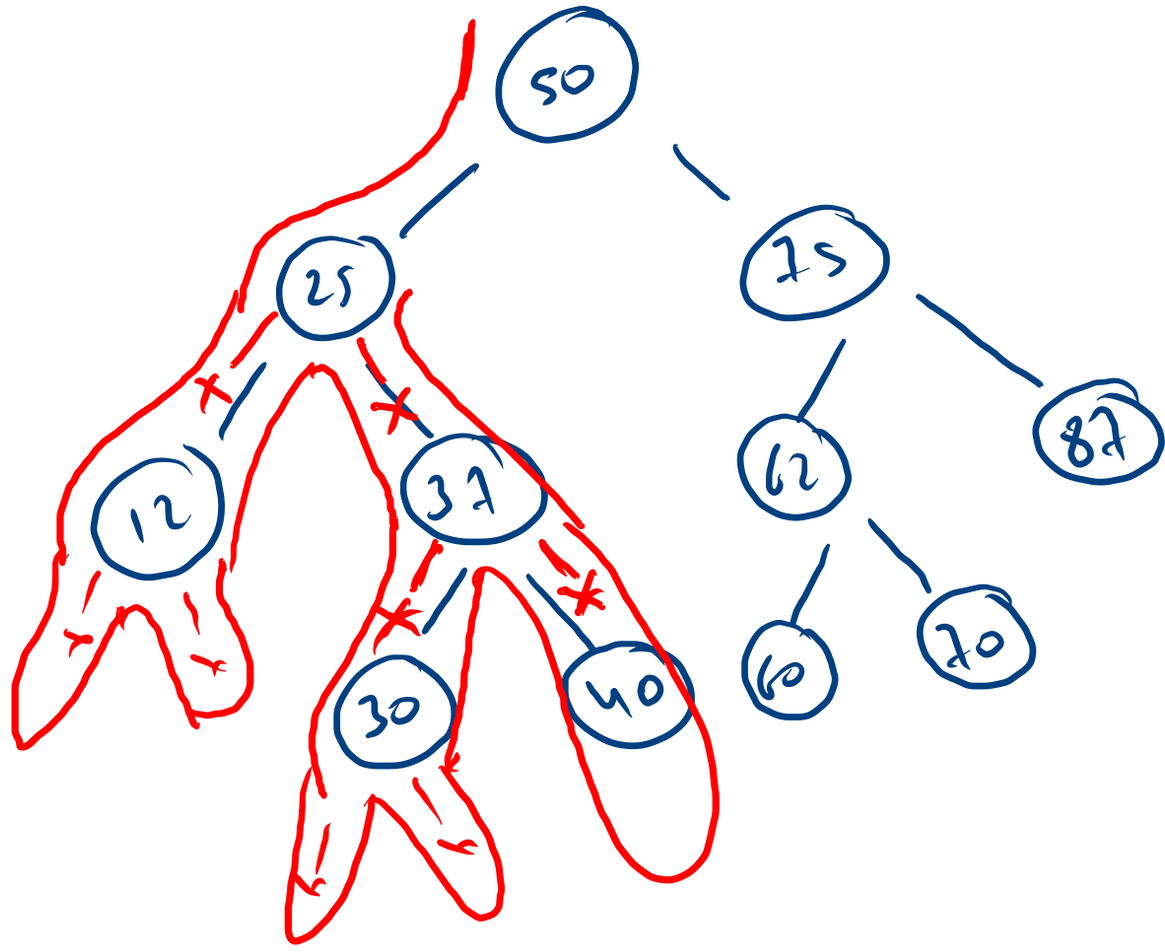

50
25
75
12
37
62
87
30
40
60
70

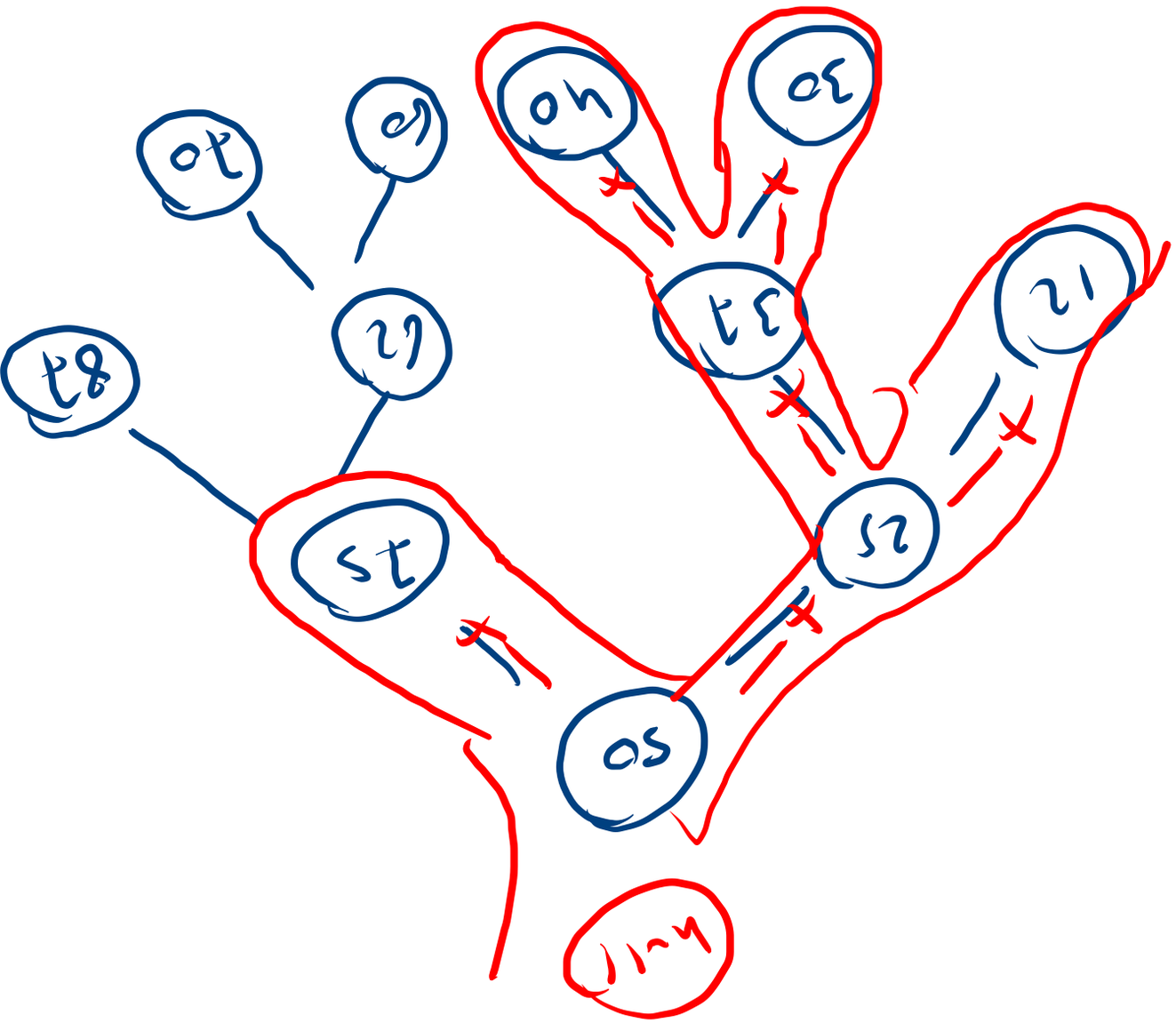


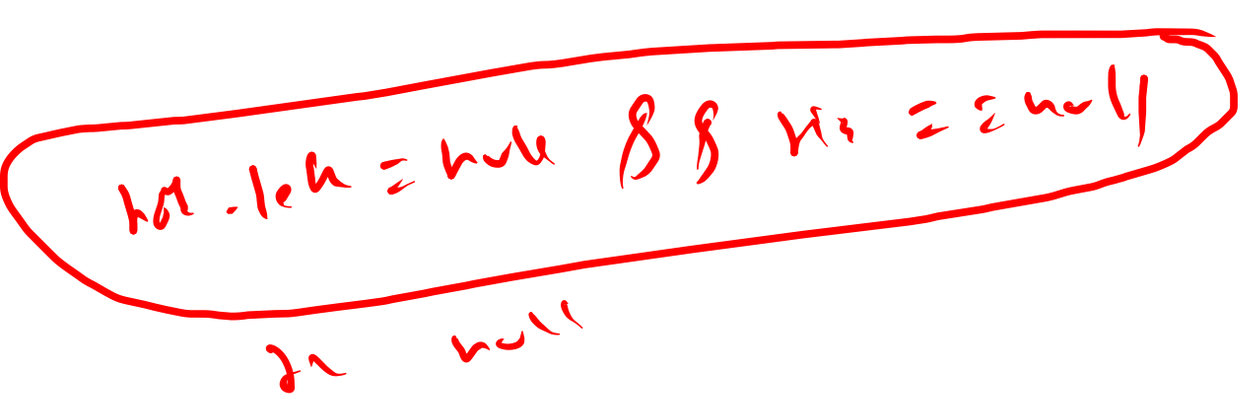

```
fun(left)
fun(right)

leaf == true or null
```

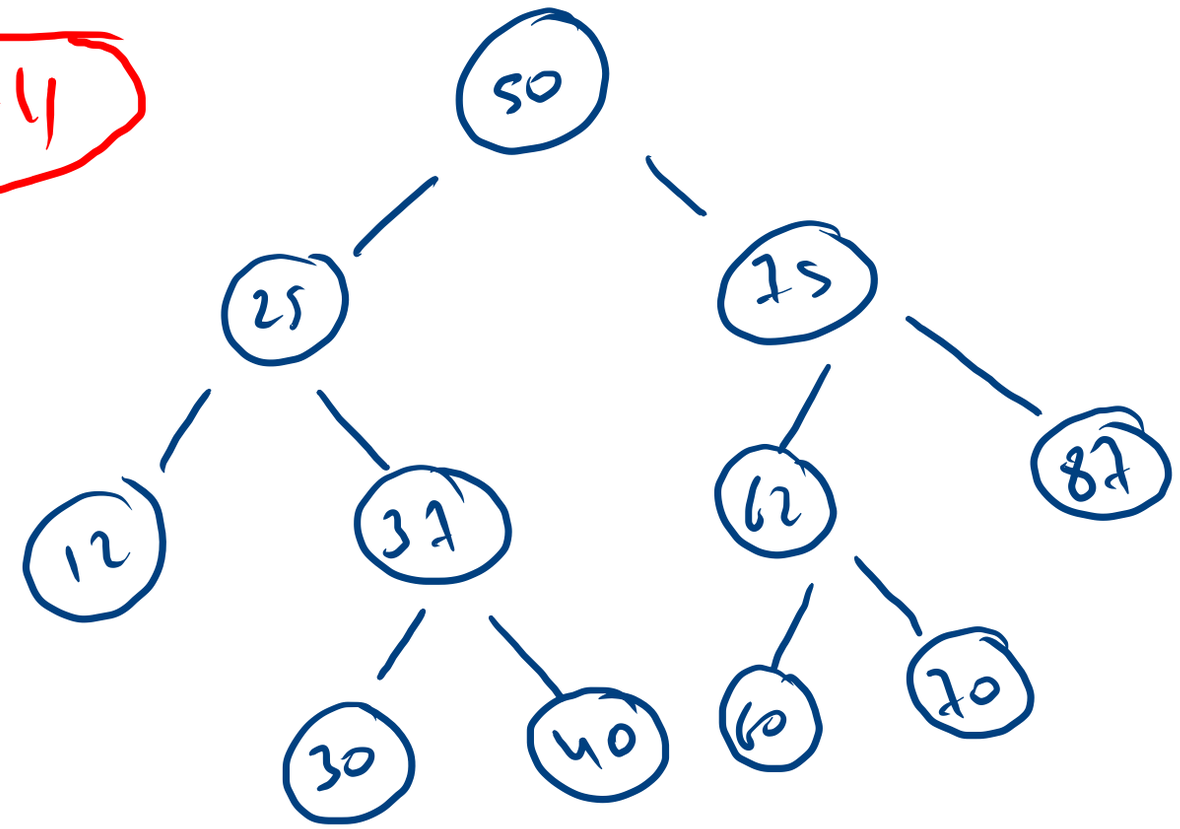


```java
public static Node removeLeaves(Node node){
    if(node == null)return null;

    node.left = removeLeaves(node.left);
    node.right= removeLeaves(node.right);

    if(l==null && r==null)return null;
    return node;
}
```

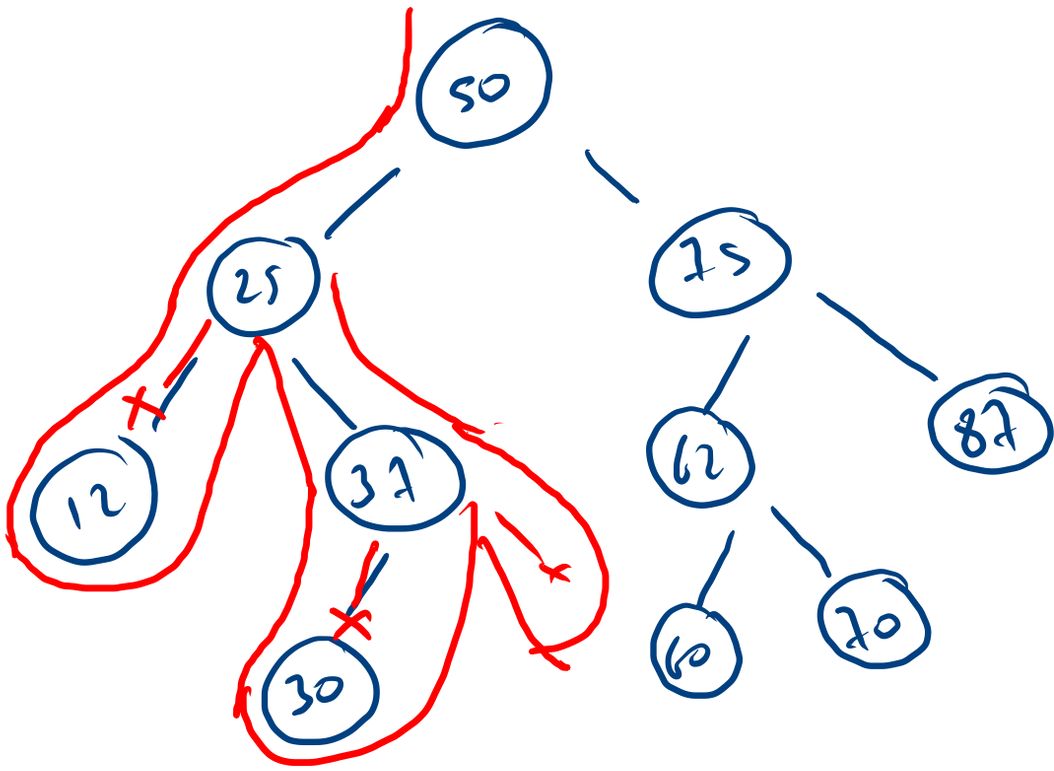


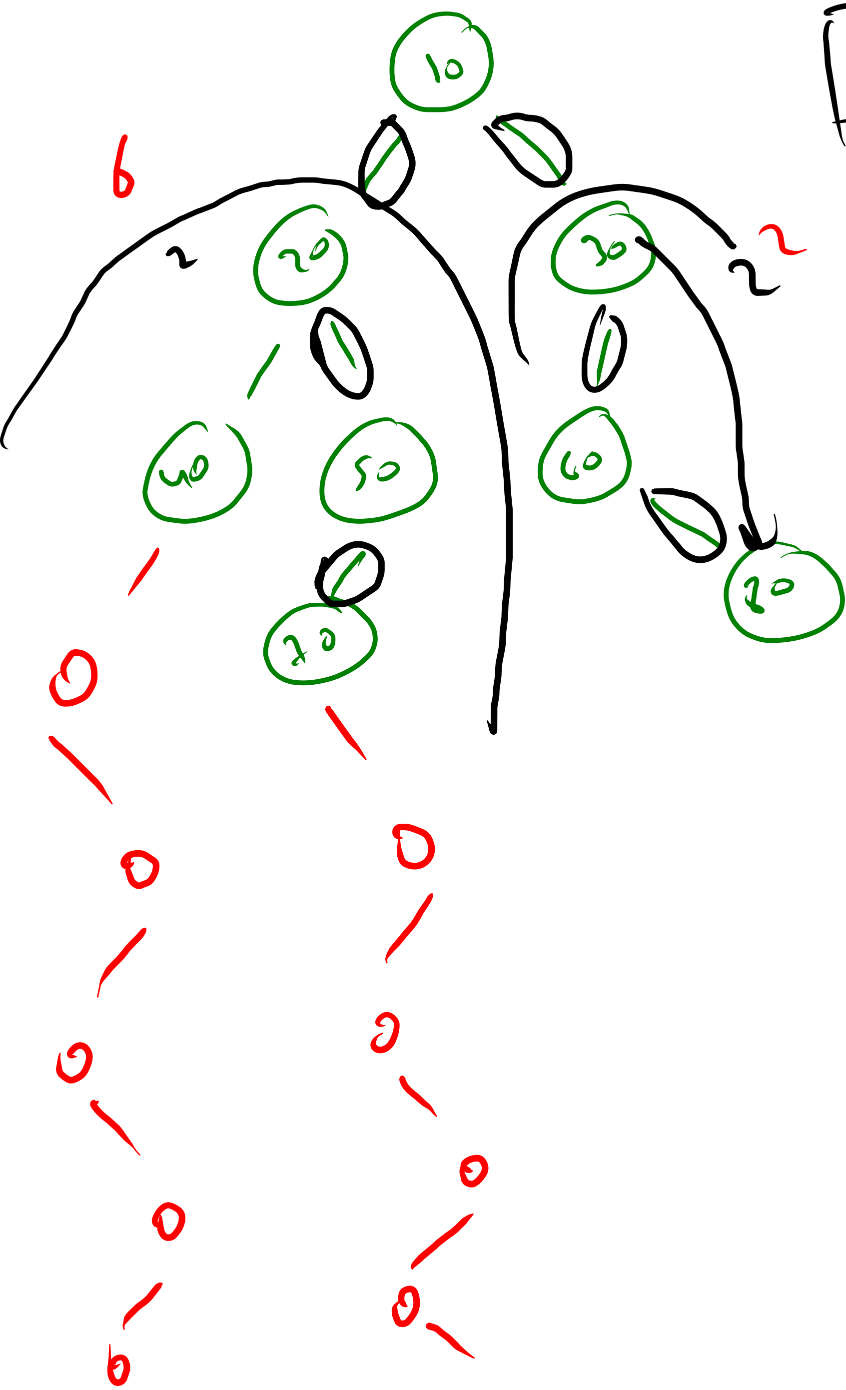





50
25
75
12
37
62
87
30
40
60
70

```java
public static Node removeLeaves(Node node){
    if(node == null)return null;
    if(node.left==null && node.right==null)return null;

    node.left = removeLeaves(node.left);
    node.right= removeLeaves(node.right);

    return node;
}
```



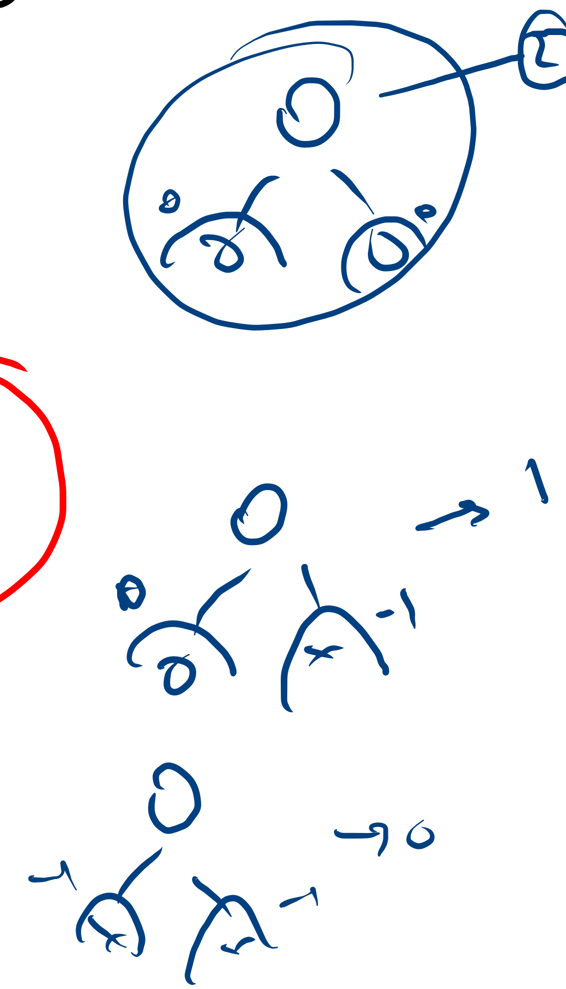


lheight

diameter = 6

10

ld , rd , lheight



```java
public static int height(Node node) {
  if (node == null) {
    return -1;
  }

  int lh = height(node.left);
  int rh = height(node.right);

  int th = Math.max(lh, rh) + 1;
  return th;
}
```

```java
public static int diameter1(Node node) {
    if(node == null)return 0;

    int ld = diameter1(node.left);
    int rd = diameter1(node.right);

    int md = height(node.left) + height(node.right) + 2;

    return Math.max(ld, Math.max(rd, md));
}
```