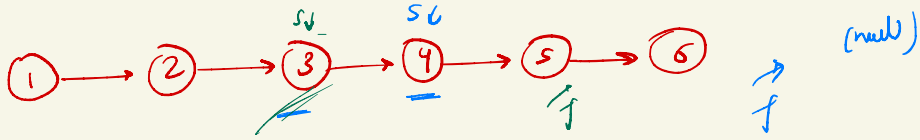
CPP

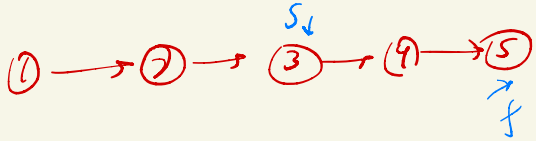
. →

Node Node*

null NULL

nullptr



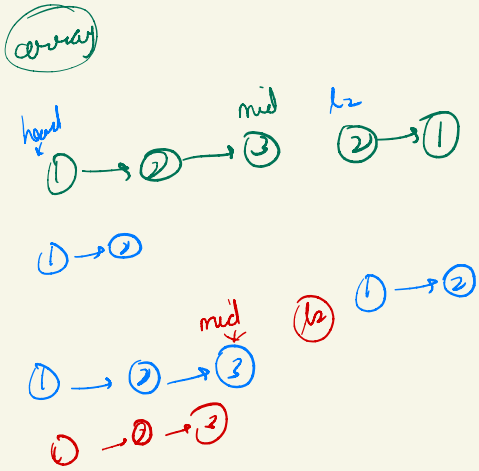


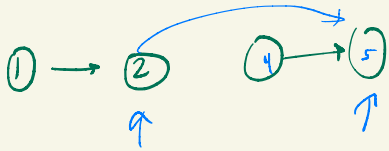
find mid

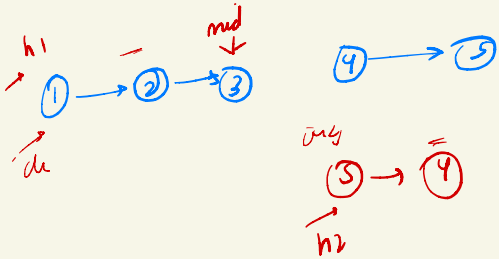
create l2

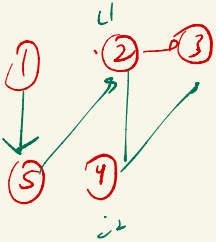
reverse l2

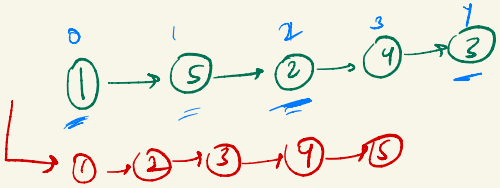
compare head, l2

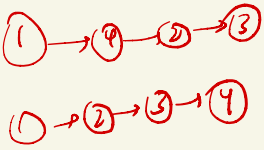










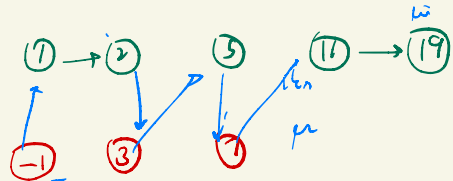


0, 2, 4 addLast (1 → 2 → 3)

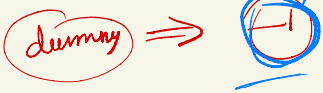
1, 3 addFirst (4 → 5)

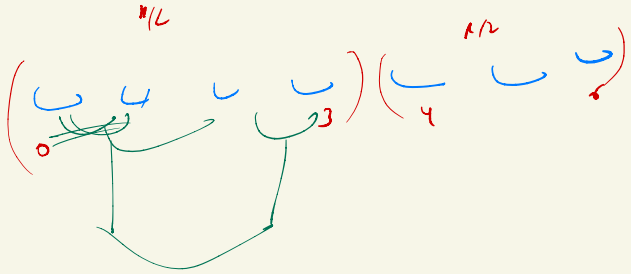
addFirst()

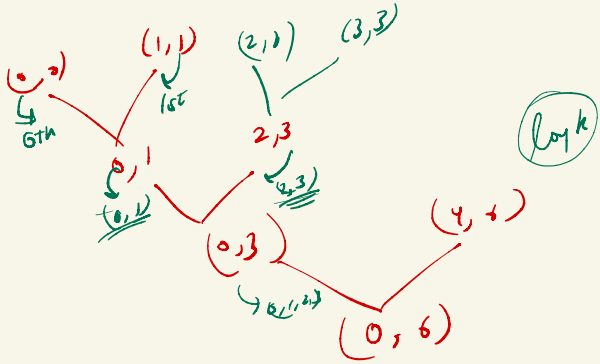
addLast()

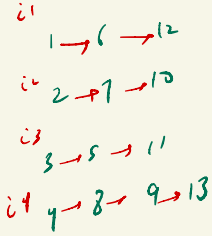


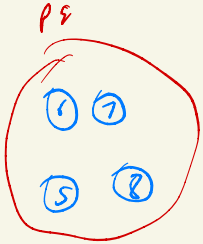

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode* dummy=new ListNode(-1);
    ListNode* itr=dummy;

    while(list1 && list2){
        if(list1->val < list2->val){
            itr->next=list1;
            list1=list1->next;
        } else {
            itr->next=list2;
            list2=list2->next;
        }

        itr=itr->next;
    }

    if(list1){
        itr->next=list1;
    }
    if(list2){
        itr->next=list2;
    }

    return dummy->next;
}
```

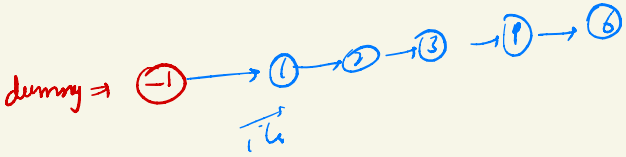


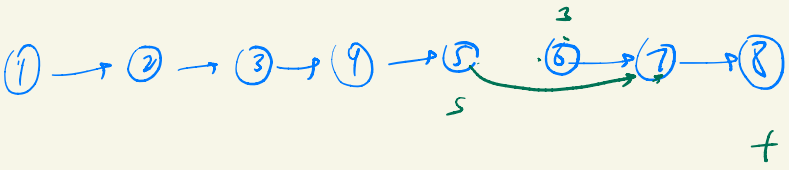

merge K lists


log k

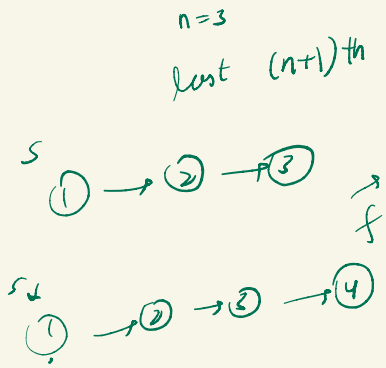


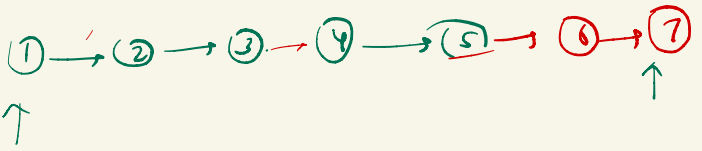


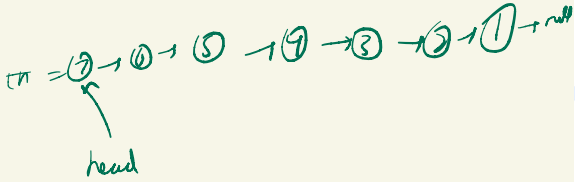

dummy



```cpp
ListNode* removeNthFromEnd(ListNode* head, int n) {
    if(n==0 || head==nullptr){
        return head;
    }

    ListNode* slow=head;
    ListNode* fast=head;

    while(n--){
        fast=fast.next;
    }

    if(fast==nullptr){
        return head->next;
    }

    while(fast->next){
        slow=slow->next;
        fast=fast->next;
    }

    slow->next=slow->next->next;

    return head;
}
```

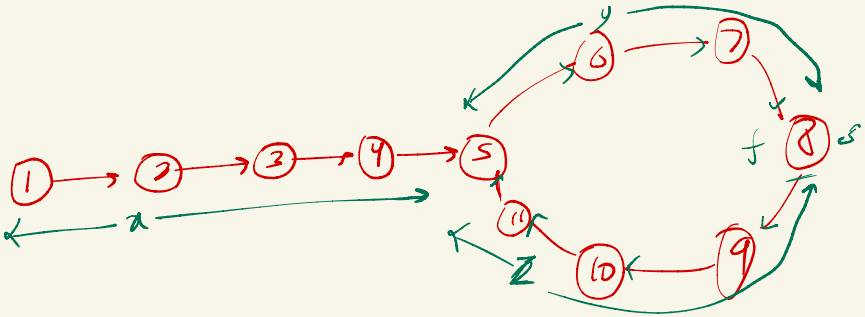






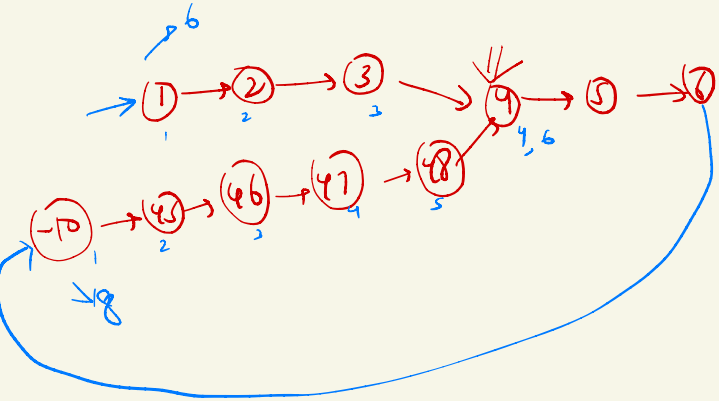
```cpp
ListNode* th=nullptr;
ListNode* tt=nullptr;

void addFirst(ListNode* node){
    if(th==nullptr){
        th=tt=node;
    } else {
        node->next=th;
        th=node;
    }
}

ListNode* reverseBetween(ListNode* head, int left, int right) {
    ListNode* itr=head;
    int curr_node=1;
    ListNode* left_prev=nullptr;

    while(itr){
        while(curr_node>=left && curr_node<=right){
            ListNode* itrKaNext=itr->next;
            itr->next=nullptr;
            addFirst(itr);

            itr=itrKaNext;
            curr_node++;
        }

        if(curr_node>right){
            if(left_prev==nullptr){ // head
                head=th;
            } else {
                left_prev->next=th;
            }

            tt->next=itr;
            break;
        }

        left_prev=itr;
        itr=itr->next;
        curr_node++;
    }

    return head;
}
```

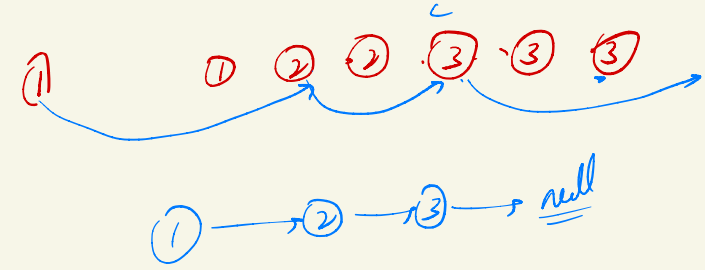
slow = slow .



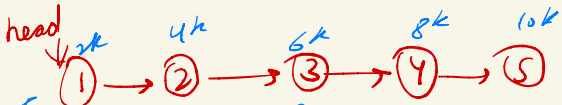
$x + y = slow$

$fast = x + 2y + 2$






```
    ListNode* deleteDuplicates(ListNode* head) {
        if(head==nullptr || head->next==nullptr){
            return head;
        }

        ListNode* curr=head;

        while(curr && curr->next){
            if(curr->val==curr->next->val){
                ListNode* currKaNext=curr->next;
                curr->next=nullptr;
                curr->next=currKaNext->next;

                currKaNext->next=nullptr; // not neccessory
            }
            if(curr->next && curr->val!=curr->next->val){
                curr=curr->next;
            }
        }

        return head;
    }
};
```



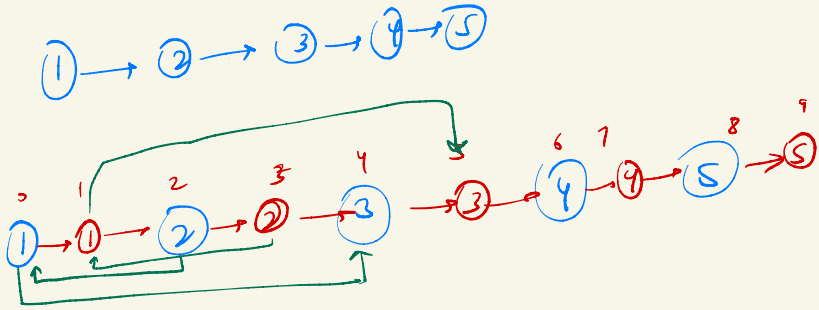
deep copy (real)

shallow copy (fake)

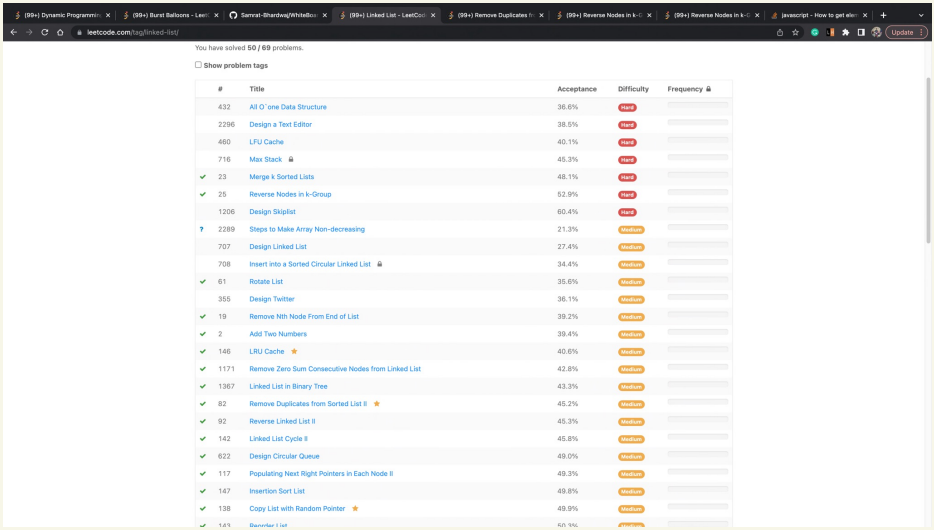
ListNode* copy = head

return copy;

int a = 5;
b = a;



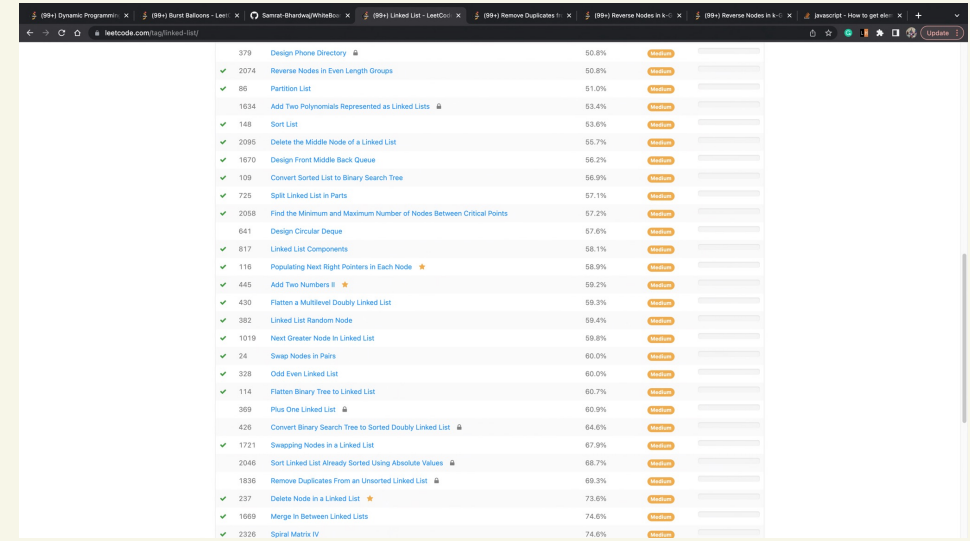
1) copy Nodes
2) attach Randoms
3) extract CL



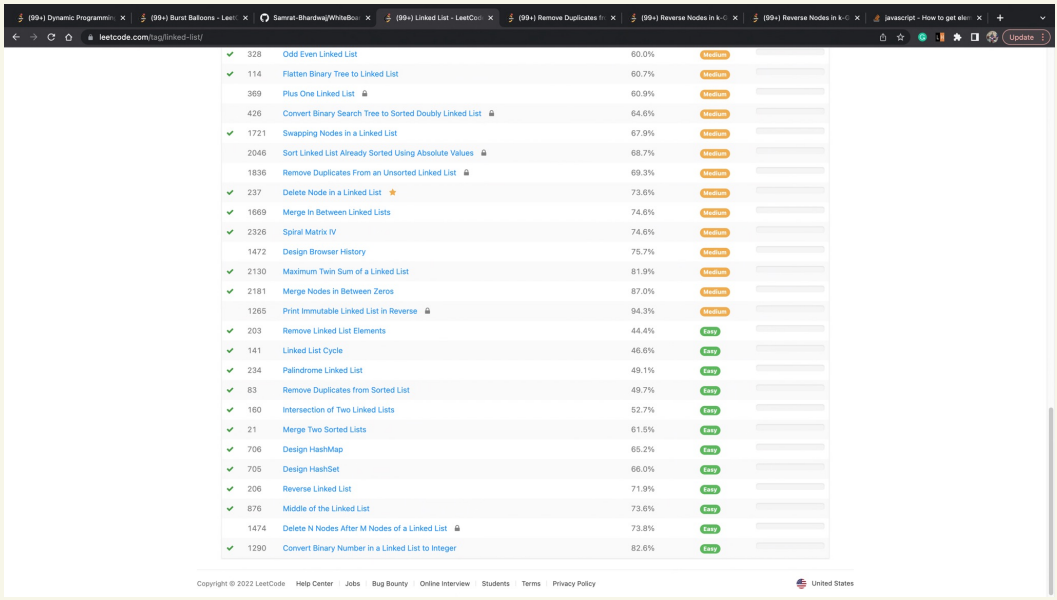
You have solved **50 / 69** problems.

Show problem tags

	#	Title	Acceptance	Difficulty	Frequency
	432	All O`one Data Structure	36.6%	Hard	
	2296	Design a Text Editor	38.5%	Hard	
	460	LFU Cache	40.1%	Hard	
	716	Max Stack	45.3%	Hard	
✓	23	Merge k Sorted Lists	48.1%	Hard	
✓	25	Reverse Nodes in k-Group	52.9%	Hard	
	1206	Design Skiplist	60.4%	Hard	
?	2289	Steps to Make Array Non-decreasing	21.3%	Medium	
	707	Design Linked List	27.4%	Medium	
	708	Insert into a Sorted Circular Linked List	34.4%	Medium	
✓	61	Rotate List	35.6%	Medium	
	355	Design Twitter	36.1%	Medium	
✓	19	Remove Nth Node From End of List	39.2%	Medium	
✓	2	Add Two Numbers	39.4%	Medium	
✓	146	LRU Cache	40.6%	Medium	
✓	1171	Remove Zero Sum Consecutive Nodes from Linked List	42.8%	Medium	
✓	1367	Linked List in Binary Tree	43.3%	Medium	
✓	82	Remove Duplicates from Sorted List II	45.2%	Medium	
✓	92	Reverse Linked List II	45.3%	Medium	
✓	142	Linked List Cycle II	45.8%	Medium	
✓	622	Design Circular Queue	49.0%	Medium	
✓	117	Populating Next Right Pointers in Each Node II	49.3%	Medium	
✓	147	Insertion Sort List	49.8%	Medium	
✓	138	Copy List with Random Pointer	49.9%	Medium	
✓	143	Reorder List	50.3%	Medium	



	379	Design Phone Directory	50.8%	Medium	
✓	2074	Reverse Nodes in Even Length Groups	50.8%	Medium	
✓	86	Partition List	51.0%	Medium	
	1634	Add Two Polynomials Represented as Linked Lists	53.4%	Medium	
✓	148	Sort List	53.6%	Medium	
✓	2095	Delete the Middle Node of a Linked List	55.7%	Medium	
✓	1670	Design Front Middle Back Queue	56.2%	Medium	
✓	109	Convert Sorted List to Binary Search Tree	56.9%	Medium	
✓	725	Split Linked List in Parts	57.1%	Medium	
✓	2058	Find the Minimum and Maximum Number of Nodes Between Critical Points	57.2%	Medium	
	641	Design Circular Deque	57.6%	Medium	
✓	817	Linked List Components	58.1%	Medium	
✓	116	Populating Next Right Pointers in Each Node	58.9%	Medium	
✓	445	Add Two Numbers II	59.2%	Medium	
✓	430	Flatten a Multilevel Doubly Linked List	59.3%	Medium	
✓	382	Linked List Random Node	59.4%	Medium	
✓	1019	Next Greater Node In Linked List	59.8%	Medium	
✓	24	Swap Nodes in Pairs	60.0%	Medium	
✓	328	Odd Even Linked List	60.0%	Medium	
✓	114	Flatten Binary Tree to Linked List	60.7%	Medium	
	369	Plus One Linked List	60.9%	Medium	
	426	Convert Binary Search Tree to Sorted Doubly Linked List	64.6%	Medium	
✓	1721	Swapping Nodes in a Linked List	67.9%	Medium	
	2046	Sort Linked List Already Sorted Using Absolute Values	68.7%	Medium	
	1836	Remove Duplicates From an Unsorted Linked List	69.3%	Medium	
✓	237	Delete Node in a Linked List	73.6%	Medium	
✓	1669	Merge In Between Linked Lists	74.6%	Medium	
✓	2326	Spiral Matrix IV	74.6%	Medium	



Stack

LIFO

([{

([! =

( )

O(1) space

O(n) - time

a++

[(([3))]

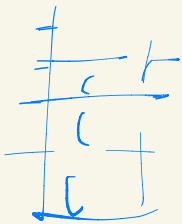
(( )) → true

a = 1 → 2 → 0

) (→ false

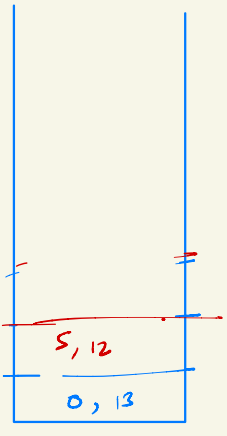
a = -1

( ) (( ) → false



arr [13, 4, 3, 7, 6, 12]

ngr = [-1, 7, 7, 12, 12, -1]



smaller

Candidate → correct window

greater

arr ⇒ [1, 3, -1, -3, 5, 3, 6, 7]

ngr ⇒ [1, 4, 4, 4, 6, 6, 7, 8]

6 3 4 5

idx = 1

nge [1, 5, 3, 4, 5, 6, 7, 8]

1, 1

6 + (k-1)

6, 2

1, 3

2, 4

ngr(idx)

idx = 4

2 - 4

idx < i

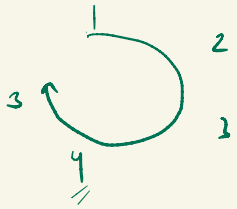
idx = i

[1, 2, 3, 4, 3]

1, 2, 3, 4, 3

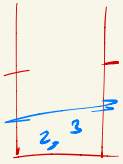
2, 3, 4, -1, 4

[2, 3



1, 6, 5, 4, 3, 2, 1

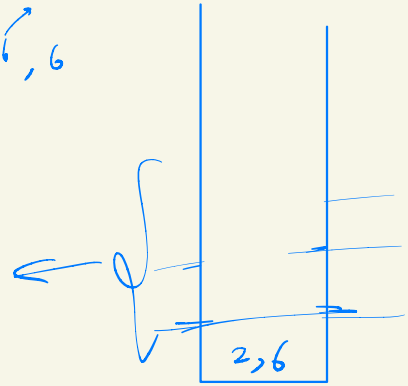
-1, 6, 6, 6, 6, 6



0 1 2 3 4 5

1, 2, 6, 3, 4, 3

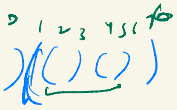
2, 6, -1, 4, 6, 6

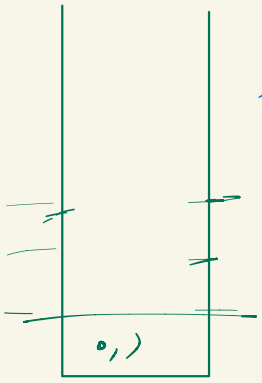


0-n

0 1 2 3 4 5 6

) ( ) ( ) ((





i-st.top

0, 2

0, 4

2-0 → 2

4-0 → 4

1, 3 ⇒ 2

1, 5 → 4

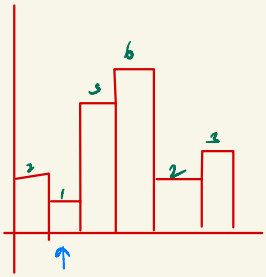
0, 6 → 6

100, 85, 60, 70

1, 1, 1, 2

next(70)

2, 1, 5, 6, 2, 3



nsr

nsl

-1 -1 1 2 1 4

1 6 4 4 6 6

nsl(i) = -1

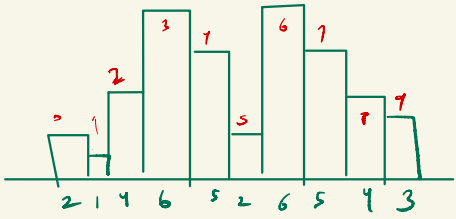
nsr(i) = 1

1 - (-1) - 1

⇒ 2 - 1 ⇒ 1

4 - (1) - 1 ⇒ 2 * 5 = 10

6 - (-1) - 1 ⇒ 6 * 1 ⇒ 6

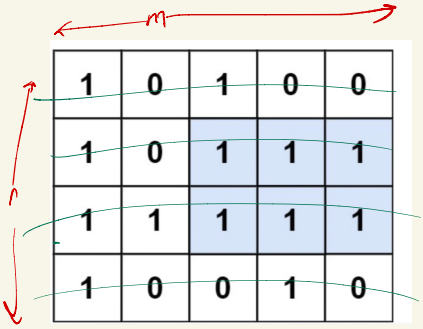


(i) → heights(i)

(i) → nsr

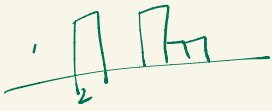
height

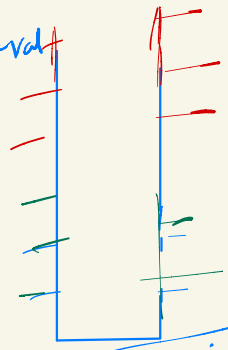
st.top → nsl



1	0	1	0	0	1
2	0	2	1	1	3
3	1	3	2	2	6
4	0	0	3	0	4

n x (m x m) → nm^2





Val < min

tp = val - min + val

min = 4 ~~3~~ ~~2~~ ~~3~~ 4

Val = 3

tp = st.top() = 2

old-min = 2 * val - tp

⇒ 6 - 2 ⇒ 4

tp = 0

Val = min = 1

new-min = 2 * 1 - 0

pmin = 2

tp < val < min

tp = (Val - min) + val

min = 2 * val - tp

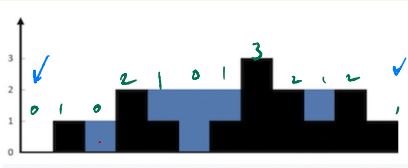
tp < val

st.top < min

1 - 3 + 1

-1

```cpp
int trap(vector<int>& height) {
    int n=height.size();
    int ans=0;

    stack<int> st;

    for(int i=0; i<n; i++){
        while(st.size()!=0 && height[st.top()]<=height[i]){
            int h=height[st.top()];
            st.pop();

            if(st.size()==0) break;

            int w=i-st.top()-1;
            ans+=w*(min(height[i],height[st.top()])-h);
        }
        st.push(i);
    }

    return ans;
}
main(){
```



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

lmax = 0

rmax = 1

lmax < rmax

i++

else

j--

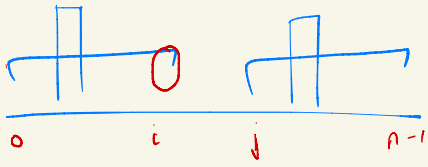
h = 0

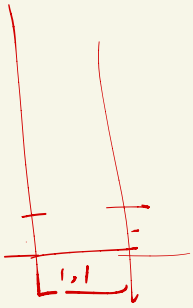
lmax - height(i)

rmax - height(j)

lmax < rmax

3 - 1 - 1 ⇒ 1





nums(i) < nums(k) < nums(j)

3, 1, 4, 2

msf 3 1 1 4

3 1 4 2

i < j < k

candidate

j th

nums(i) < nums

nums(k) < nums(j)

k → stack

if (msf(i) < nums(j))

i j k

1, 3, 2

i < j < k

j

2

Heap

arr 0 1 2 3 4 5 6 7 8

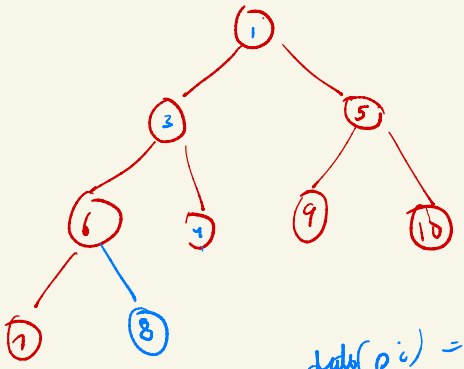
1, 3, 5, 6, 4, 9, 10, 7, 8,

pi = 0

lci = 2*pi + 1

rci = 2*pi + 2

minheap



ci = 8

pi = (8-1)/2

ci-1/2

pi

data(pi) < data(lci)

data(pi) - data(lci)

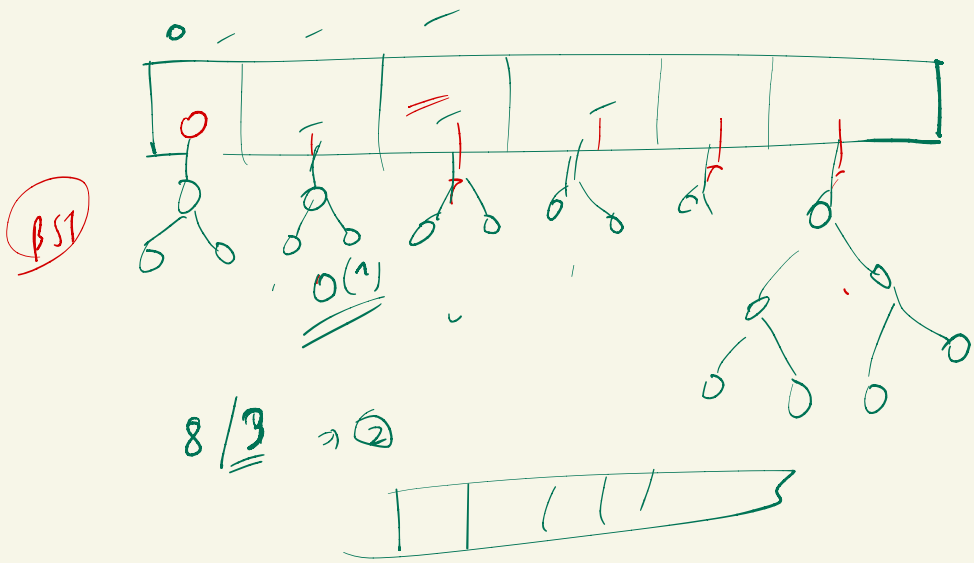
data(pi) = 1

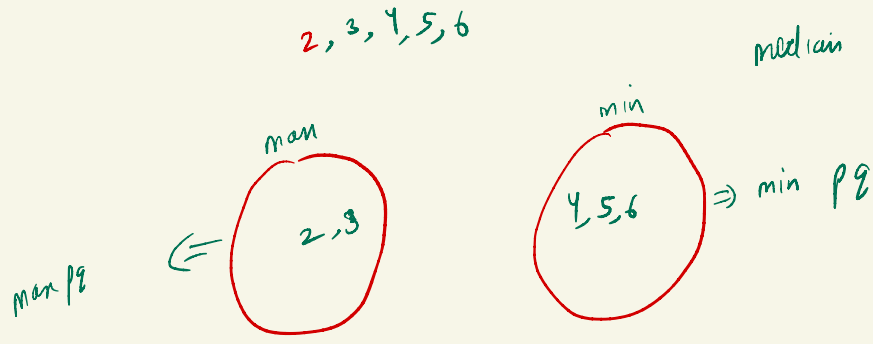
data(lci) ≥ 0

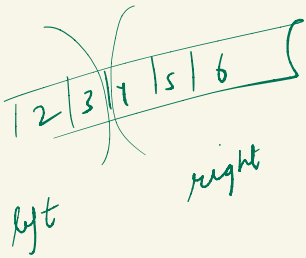
data(pi) > data(li)

data(li) - data(pi)

minheap = false







$\frac{3+4}{2} \rightarrow 3.5$

1 5 9

10 11 13

12 13 15

(10)

nxm

O(n+m)

find

kth smallest

$\log(k) + \log(n^2)$

$\log(n)$

4

1 → 1

2 → 1

3 → 1

4 → 4

5 → 5

6 → 3

7

→ 8

0 → 1 free

1 → 2 free

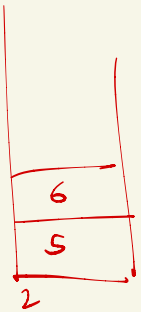
2 → 3 free

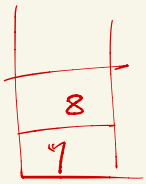
(3)

LIFO

stack

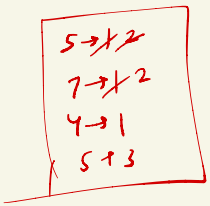
multiple stacks

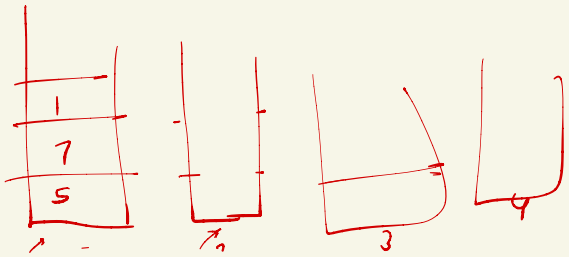




```cpp
class FreqStack {
public:
    unordered_map<int,int> fre;
    unordered_map<int, stack<int>> stacks;
    int maxFre=0;

    FreqStack() {

    }

    void push(int val) {
        fre[val]++;

        maxFre=max(maxFre,fre[val]);
        stacks[fre[val]].push(val);
    }

    int pop() {
        int rv=stacks[maxFre].top();
        stacks[maxFre].pop();

        if(stacks[maxFre].size()==0) maxFre--;
        fre[rv]--;

        // if(fre[rv]==0) fre.erase(rv);

        return rv;
    }
};
```

Input
["FreqStack", "push", "push", "push", "push", "push", "push", "pop", "pop", "pop", "pop"]
[[], [5], [7], [5], [7], [4], [5], [], [], [], []]





(1,2) (3,1) (7,3) 1,1

m = 4

key, node

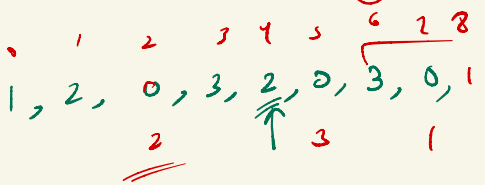
10,12

dll

set

1, 2
3,

5



1,1 = 5,8 9,10 10,12

1

0, 0, 1

-1, -1, -1

0

2

0, 5

0, 6

snap

1, 5

1, 7

snap

0, 9

0, 11

snap

sz = 2

[0][0] → 5

[0][0] → 6

idx, snap → val

[1][1] → 5

[1][1] → 7

(0)(2) → 9

(0)(2) → 11

vector < map <int, int>>;

idx

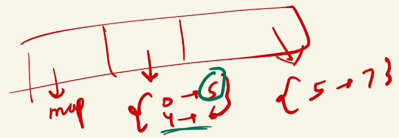
(idx) (snap) = val:

0

1

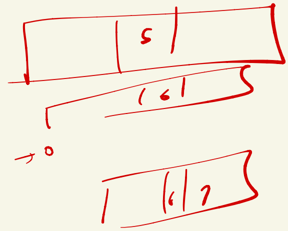
1, 2 → 7

1, 0



```
class SnapshotArray {
public:
    int snap_id;
    vector<map<int,int>> arr;

    SnapshotArray(int length) {
        arr.resize(length);
        snap_id=0;
    }

    void set(int index, int val) {
        arr[index][snap_id]=val;
    }

    int snap() {
        return snap_id++;
    }

    int get(int index, int snap_id) {
        auto itr=arr[index].upper_bound(snap_id);

        if(itr==arr[index].begin()) return 0;
        return prev(itr)->second;
    }
};
```

1, 3



ind, i

1, 0 → 5 → 0

5
5
5
5

1, 4 → 6

5
2, 5 → 7

0, 4

1, 1, 1, 1, 1, -1, -1, 1, 1, 1

sum=3

0, 1, 2, 3, 1, 5, 6, 5, 4

0 → 1
1 → 1
2 → 1
3 → 1
4 → 1
5 → 1
6 → 1

1, 1,

a → 0
b → 1
c → 6
d → 3
e → 4
z → 25

e, z

0 2 4
acacacacacadadadaeaeaezezez

a → 1900
b → 25 00

e → 4, 0 4
z → 41 25

ans = 5

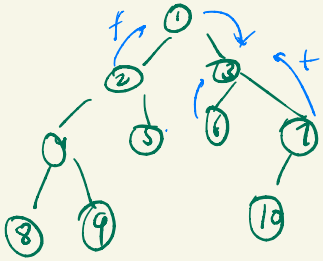
[4,12,2,7,3,18,20,3,19], bricks = 10, ladders = 2

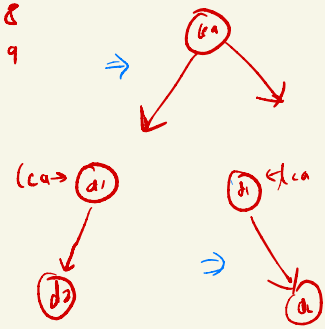
TREE

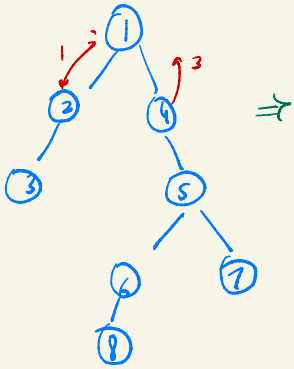
⇒ LCA

self = t
l = f
r = t

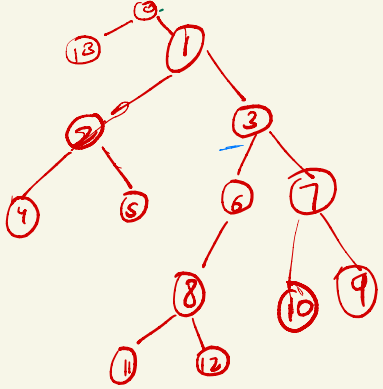
LCA = null
self = false
left
right







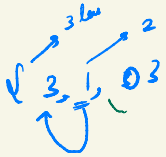
k far



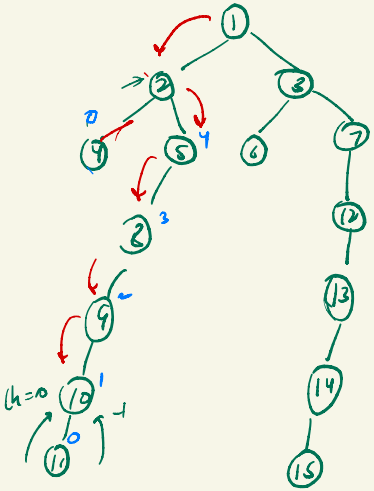
4, 5, 11, 12, 13

node = 3

3



diameter



O(n)

9

lh = -1

rh = -1

dia = 0

11

lh = -1

rh = -1

he = 0

dia = 0

lh = 0

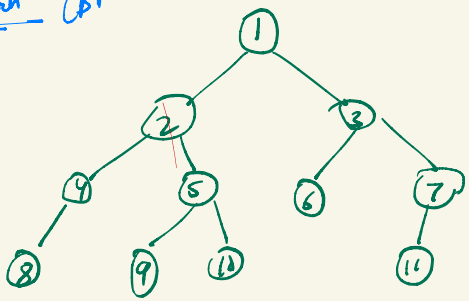
rh = -1

h = 1

dia = 0+(-1)+2 = 1

Breadth first search (BFS)

level order



(1)

2 3

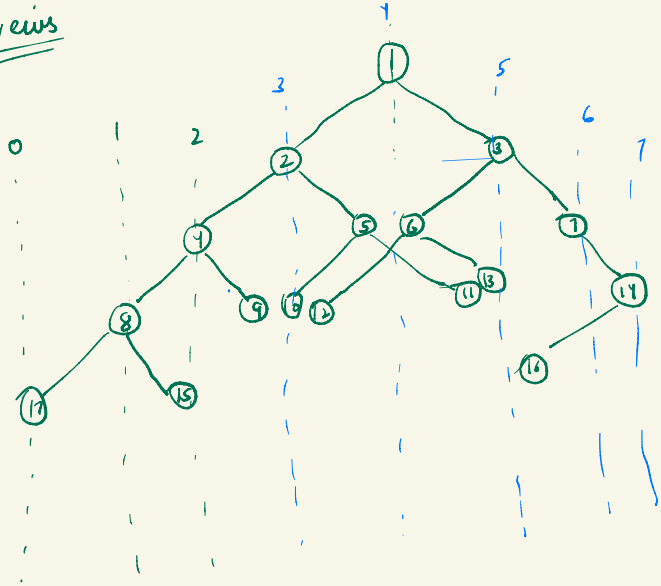
4 5 6 7

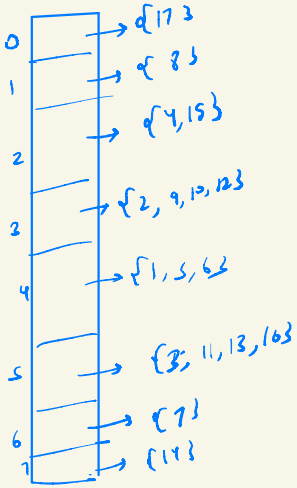
s = ~~4~~ ~~3~~ 2 1

8	9	10	11

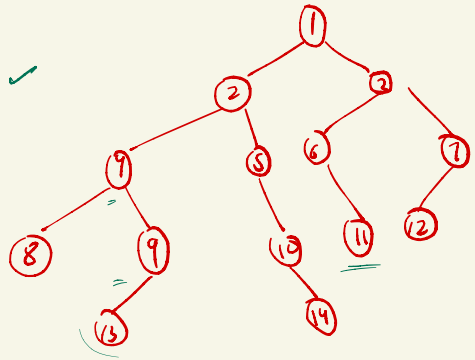
Views

(4) ↓ ref





Traversals




```
, see (root. left)
{  prev = root
see (root. right)
```

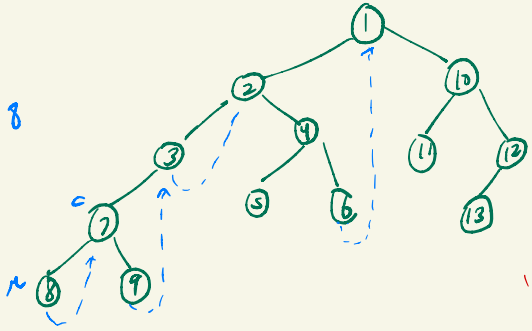
prev = 13 root = 9

8, 9, 13, 9, 2, 5, 10, 14, 1, 6, 11
3, 12, 7

```
root.data == data
prev = prev;
prev.data = data
succ = root
```

(4)

Morris traversals



1, 2, 3, 7, 8

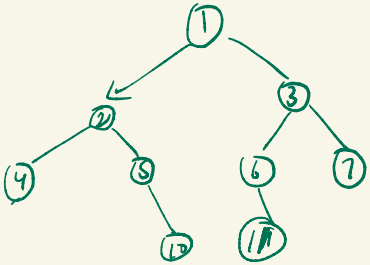
```
if (ln == null)
    curr = curr.right;
else {
    rem
⇒   if (rem.right == null)
        rem.right = curr;
        curr = ln
}
while (root.right != null
        && root.right != curr)
    root = root.right;
```

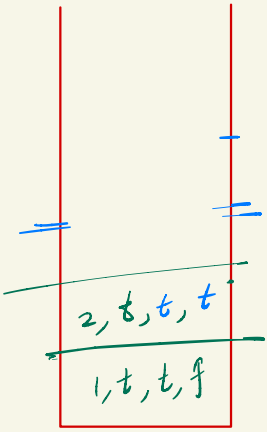
rem = 8

```
⇒ if (rem.right == curr)
    # thread break
    rem = curr.right;
}
```

#

pre → {1, 2, 4, 5, 3
post → {4, 5
in → {4, 2, 5





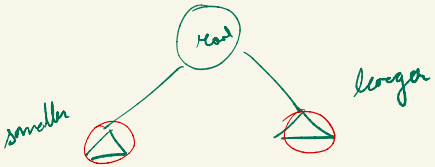
pre area

max. left

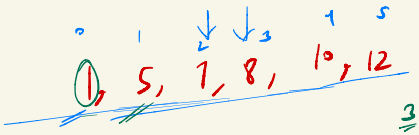
in area

right

Σ post area

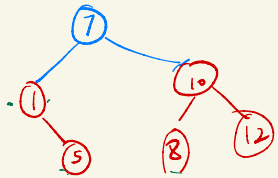


sorted



3

```
TreeNode* insertIntoBST(TreeNode* root, int val) {
    if(root==nullptr){
        return new TreeNode(val);
    }

    if(root->val < val){
        insertIntoBST(root->right,val);
    } else {
        insertIntoBST(root->left,val);
    }

    return root;
}
```



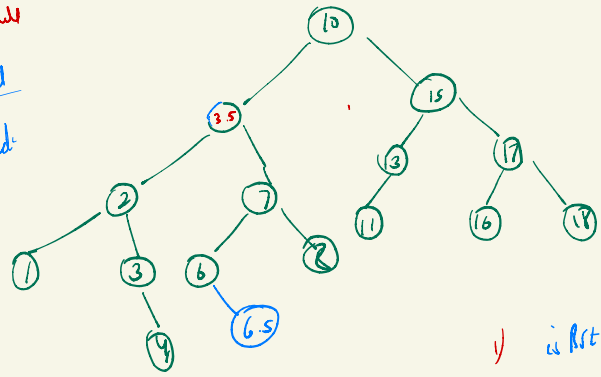
3

3

leaf → return null

→ one child

→ two child



→ just smaller

→ just longer

prev = x ~~x~~ ~~3~~ y

1) isBst(root.left)

2) if(prev!=null && prev.val > root.val) return false

3) prev = root (inorder)

4) isBst(root.right)

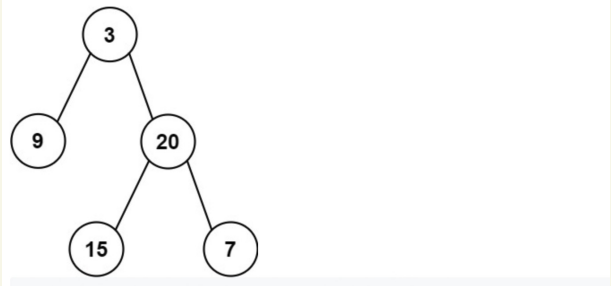
BST pair of
min
max
in BST

preorder ⇒ root, left, right.

inorder ⇒ left, root, right.

⇒ 3, 9, 20, 15, 7

⇒ 9, 3, 15, 20, 7



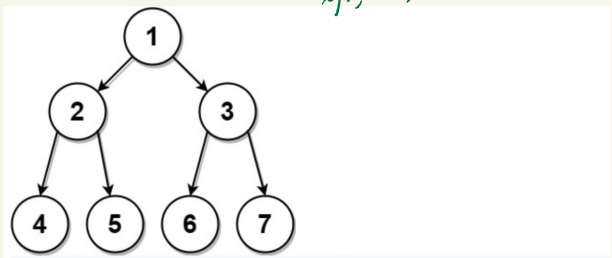
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

left ⇒ 1 left ⇒ 1

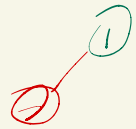
right → 3 right → 1

root, left, right

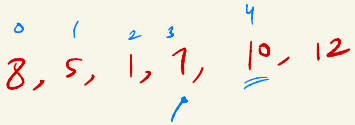
left, right, root



Input: preorder = [1,2,4,5,3,6,7], postorder = [4,5,2,6,7,3,1]
Output: [1,2,3,4,5,6,7]



idx = 0 1 2 3 4 5



(-∞, ∞)

1 3

```java
public TreeNode trimBST(TreeNode root, int low, int high) {
    if(root==null) return null;

    if(root.val < low){
        return trimBST(root.right,low,high);
    } else if(root.val > high){
        return trimBST(root.left,low,high);
    } else {
        root.left=trimBST(root.left,low,high);
        root.right=trimBST(root.right,low,high);
    }

    return root;
}
```