# Day 6: Arrays

Task:

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears once or twice, return an array of all the integers that appears twice.

Input: nums = [1,1,2]
Output: [1]

① Search in array

- lot of overlapping patterns with binary search and sorting

steps:

a. keep a dictionary with number/element occurences

b. store reoccurring elements in a list also

e.g: [4, 3, 2, 7, 8, 2, 3, 1]

dict = {4 : 1} $\xrightarrow[\text{element}]{\text{next}}$ {4:1, 3 : 1} $\xrightarrow[\text{ele.}]{\text{next}}$ {4 :1, 3 :1, 2:1} $\xrightarrow{\text{next}}$ . . .

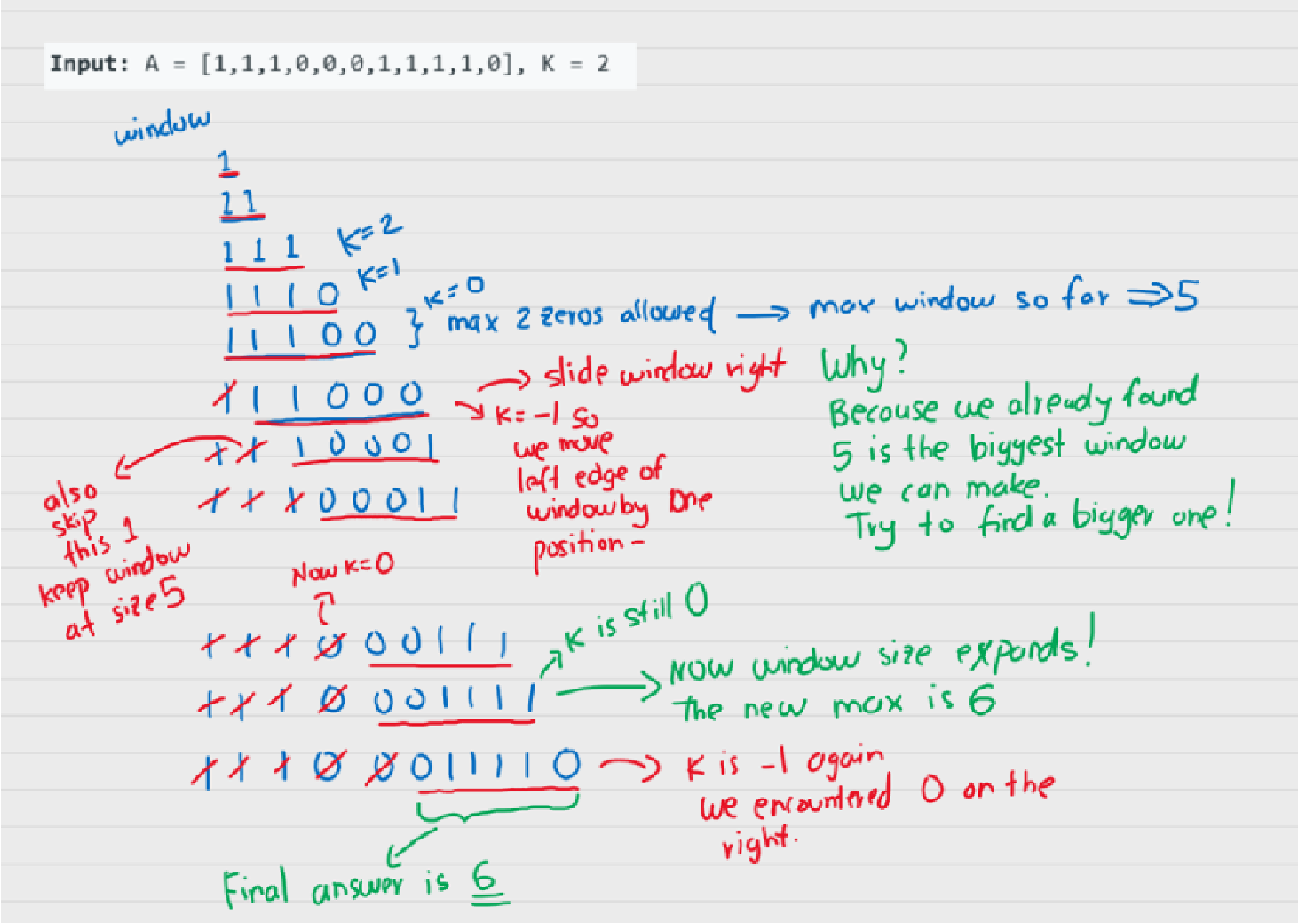
{4 : 1, 3 :1, 2:2, 7:1, 8:1} $\xrightarrow{\text{next}}$ {4:1, 3:2, 2:2, 7:1, 8: 1} $\xrightarrow{\text{next}}$ . . .

store in a list

store in a list

# ② Subarray Selection:

Task: Given a binary array nums and an integer k, return the maximum number of consecutive 1's in the array if you can flip at most k 0's.




https://leetcode.com/problems/max-consecutive-ones-iii/discuss/719833/Python3-sliding-window-with-clear-example-explains-why-the-soln-works

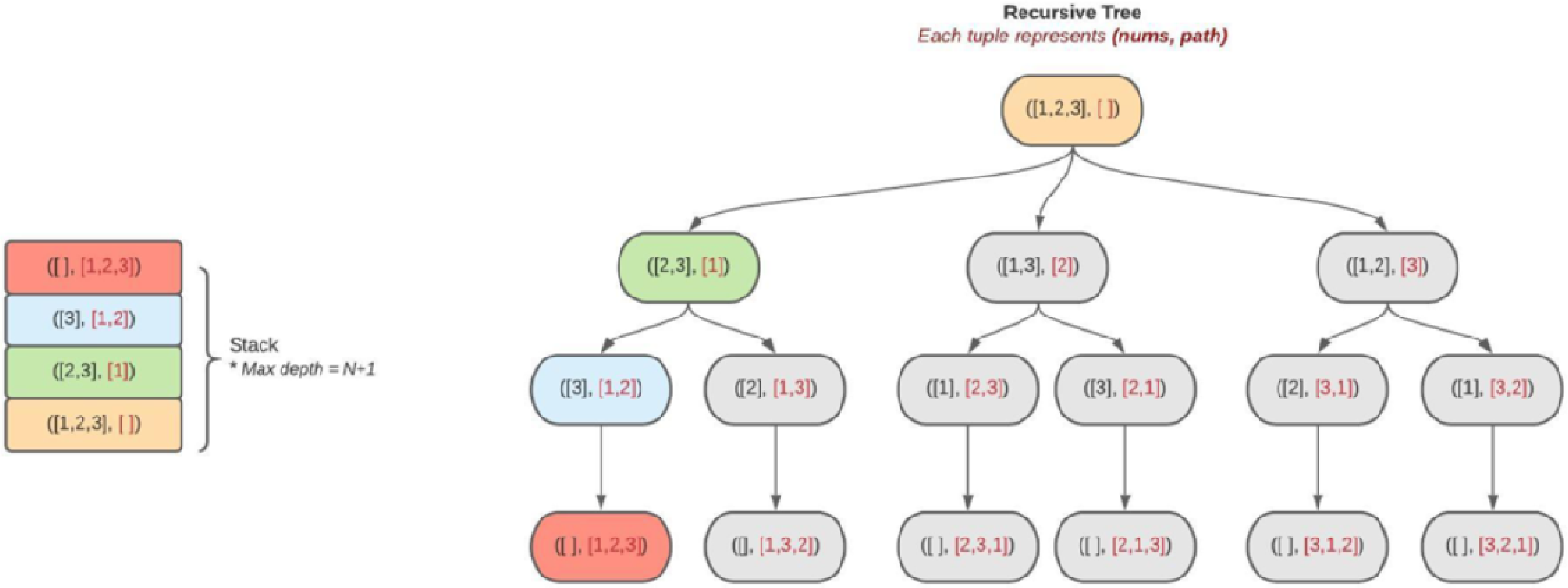
## Takeaway:

- subarrays can be managed with sliding window.
- besides, the size/length of the window can be adjusted as we gather more information from different subarrays

# ③ Reordering:

[a, b, c, d]

create all the permutations




https://leetcode.com/problems/permutations/discuss/993970/Python-4-Approaches-%3A-Visuals-%2B-Time-Complexity-Analysis

## 4) two arrays in tandem:

- merging without extra space
- use pointers for each array

Task: Given two sorted arrays arr1[] and arr2[] of sizes N and M in non-decreasing order. Merge them in sorted order without using any extra space. Modify arr1 so that it contains the first N elements and modify arr2 so that it contains the last M elements.

arr1 = [1, 3]

arr2 = [0, 2]

swap and keep track of which elements are not yet sorted

[1, 3]   [0, 2]

swap

[0, 3]   [1, 2]

swap

[0, 1]   [3, 2]

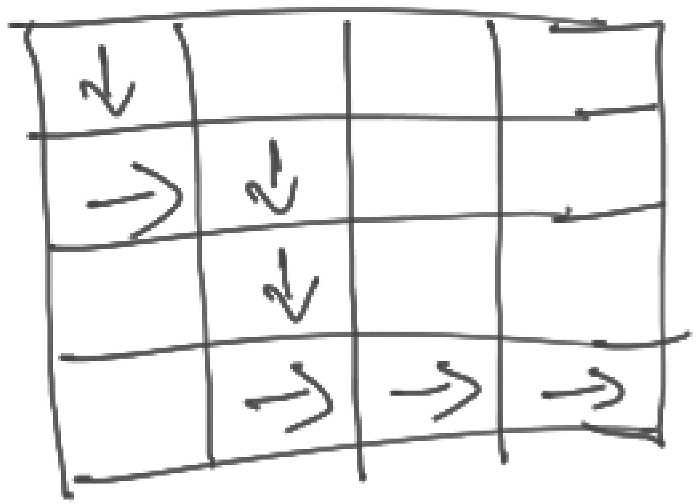
swap

[0, 1]   [2, 3]

# ⑤ matrices

## recursion and memoization.

**Task: Given a matrix, find the path from top left to bottom right with the greatest product by moving only down and right.**



one step = one recursive call

## In-place operations:

**Task: Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's, and return the matrix. You must do it in place.**

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

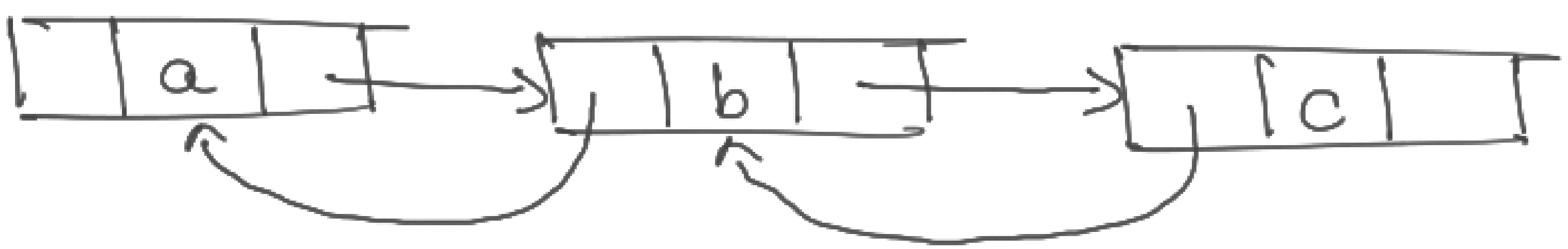
keep track, but do not immediately update the matrix in-place

# Day 7: Linked Lists:

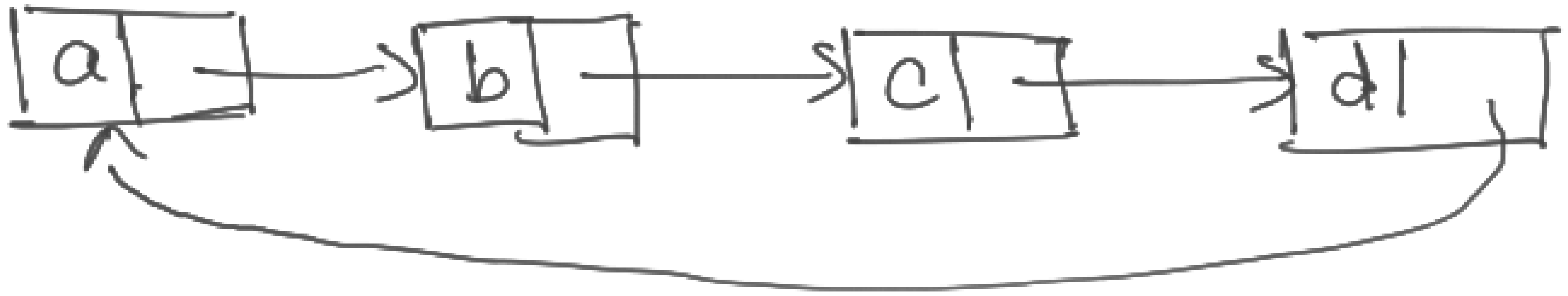
## ① Simple Implementations:

Single Linked List:

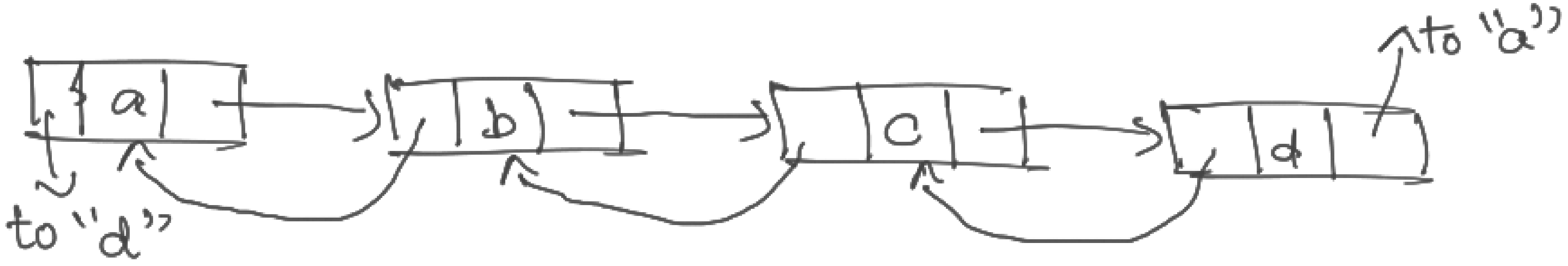

Double Linked List:




Circluar Linked List:




Double Circular Linked List,




make sure you get next & prev pointer values correctly, before modifying those.

## 2 Reversals

Q. 1 → 2 → 3

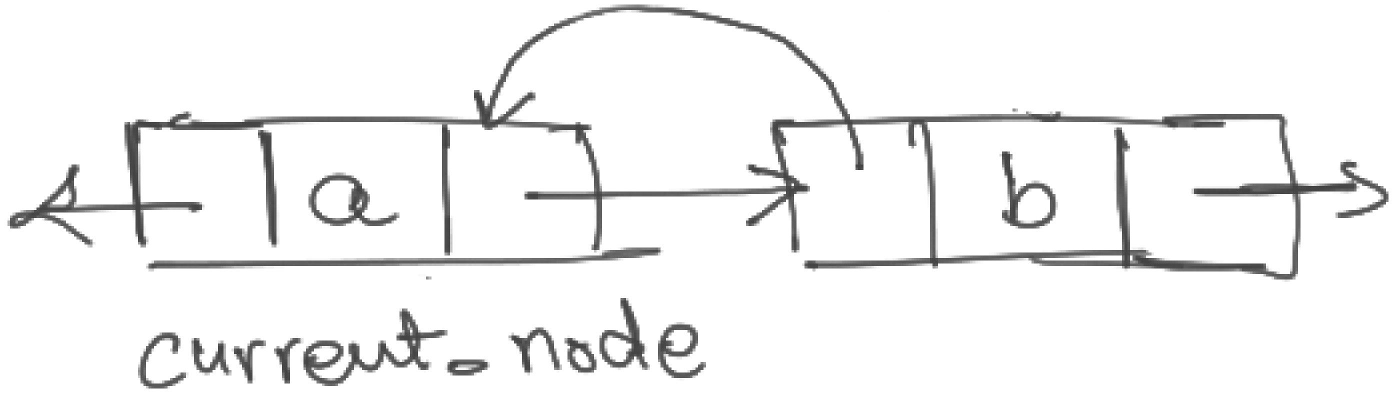
reverse a linked list

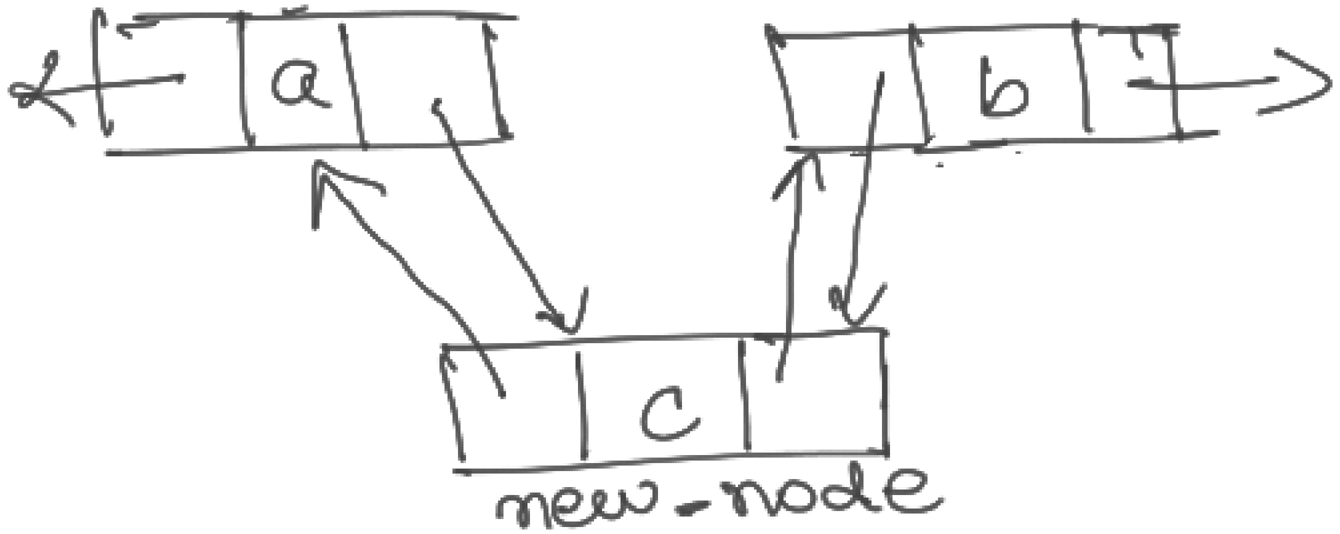
A. 3 → 2 → 1

we would need 3 pointers: prev, curr, next_

```
next_ = curr.next
curr.next = prev
prev = curr
```

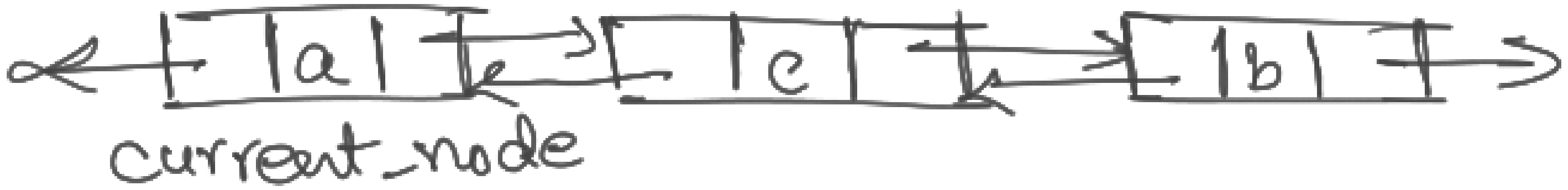
## 3 Insertions:




```
new_node = Node("c")
new_node.prev = current_node
new_node.next = current_node.next
current_node.next.prev = new_node
current_node.next = new_node
```

## Deletions:





```
current_node.next.next.prev
    = current_node
current_node.next
    = current_node.next.next
```

④ Cycles:

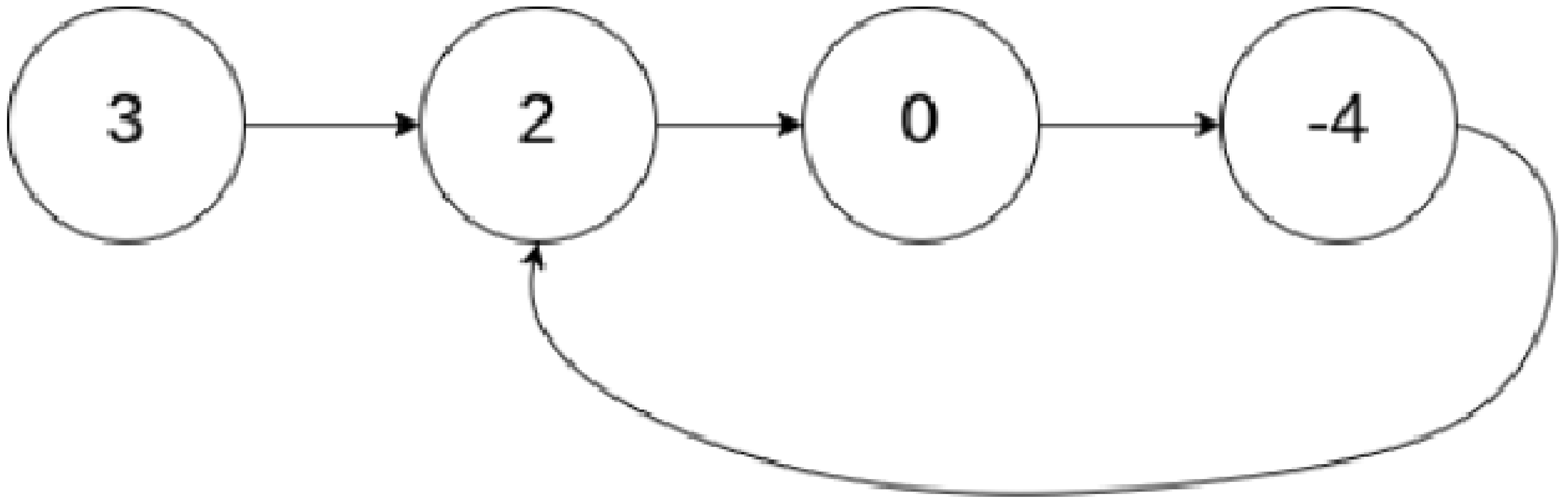
might not be distinct

Instead of storing /keeping track of values, or whole data structures, keep track of pointers/references.

memory efficient

**Task: Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return null.**

**Example 1:**



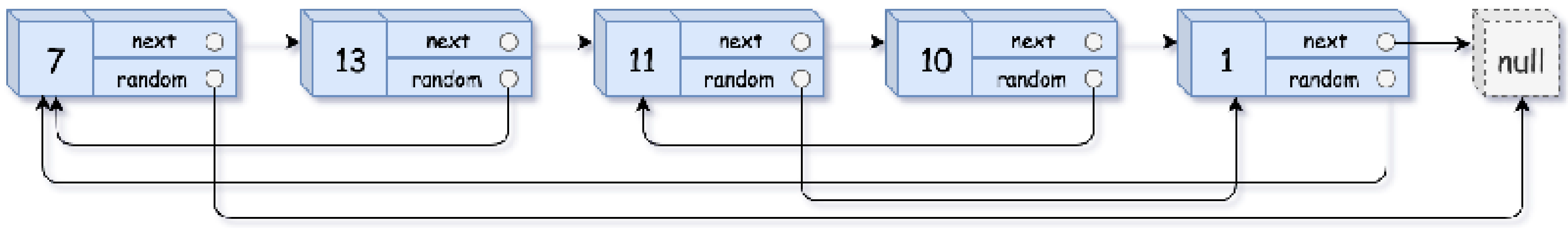

**Input:** head = [3,2,0,-4], pos = 1
**Output:** tail connects to node index 1
**Explanation:** There is a cycle in the linked list, where tail connects to the second node.

## ⑤ Split / Merge / Copy

usually when dealing with linked lists, keep track of nodes by their addresses and not by their stored values.

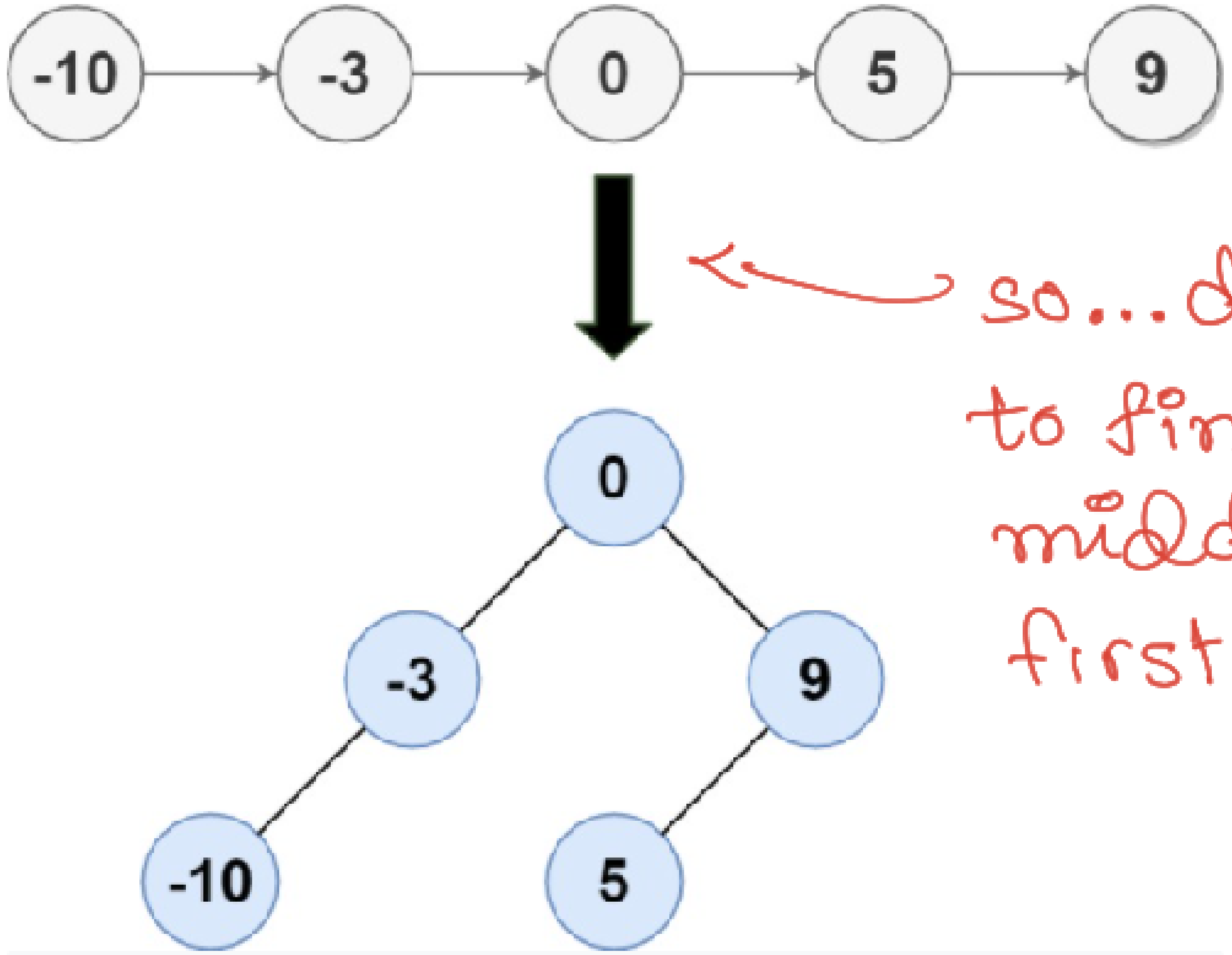
which data structure stores a value which can correspond to some other value?

**Task: A linked list of length n is given such that each node contains an additional random pointer, which could point to any node in the list, or null.**

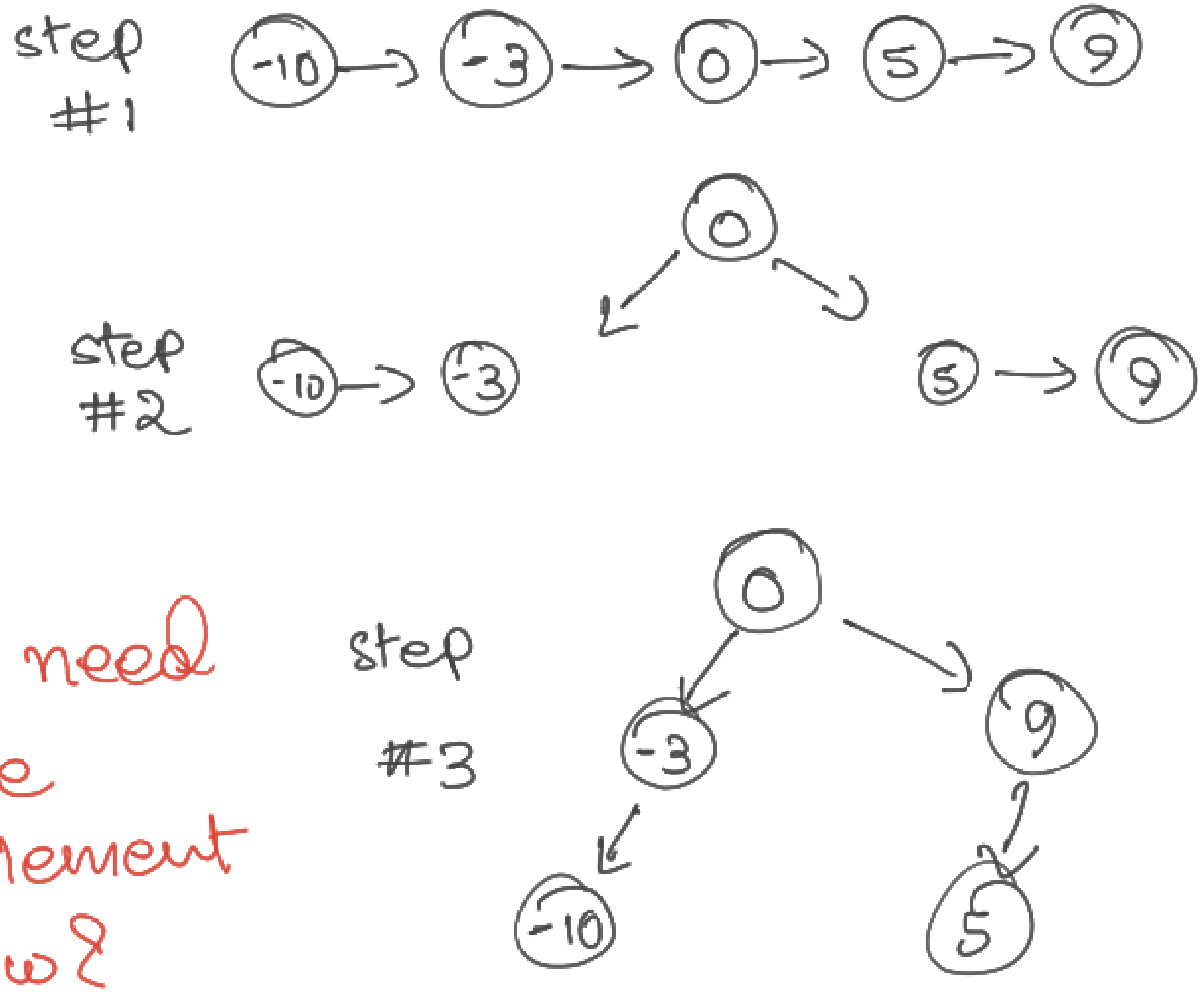


# ⑥ Linked List → Tree

**Task: Given the head of a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.**



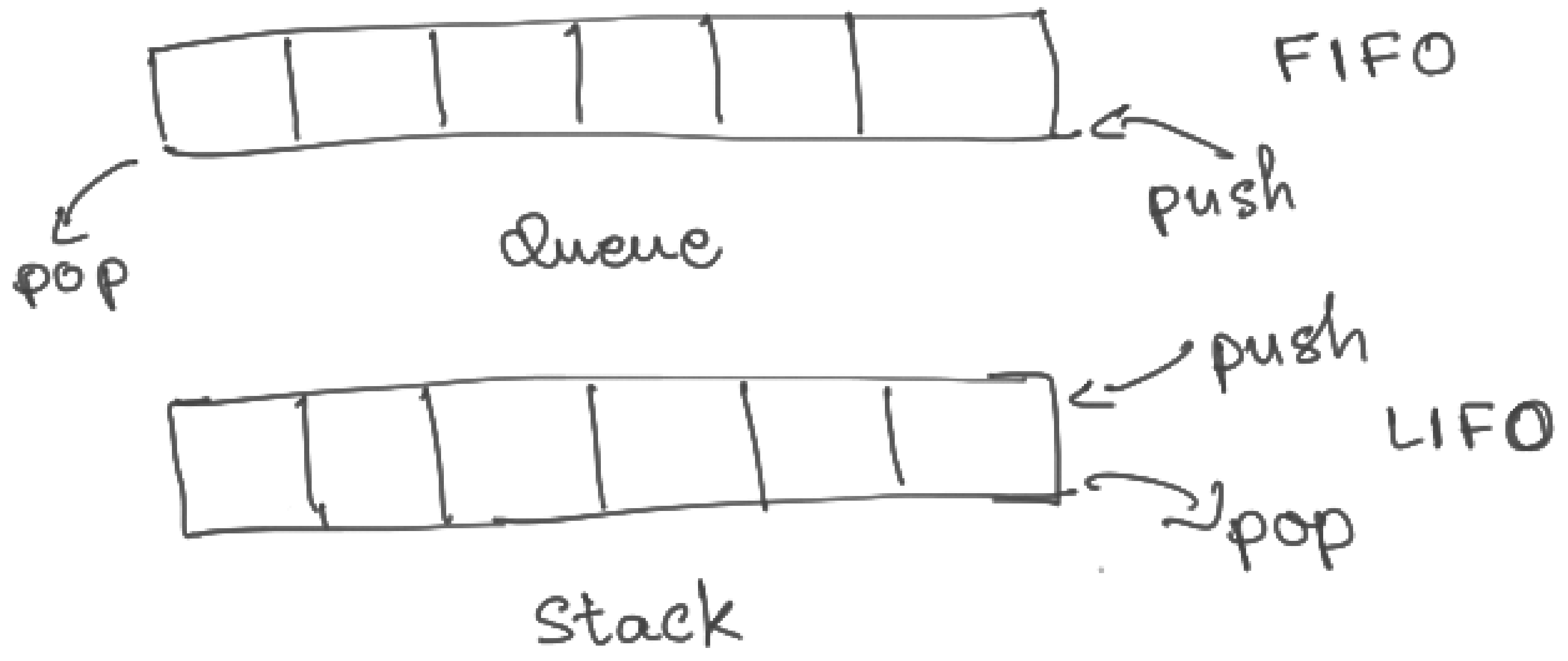

so...do we need to find the middle element first? how?




```
Input: head = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: One possible answer is [0,-3,9,-10,null,5], which represents the shown height balanced BST.
```

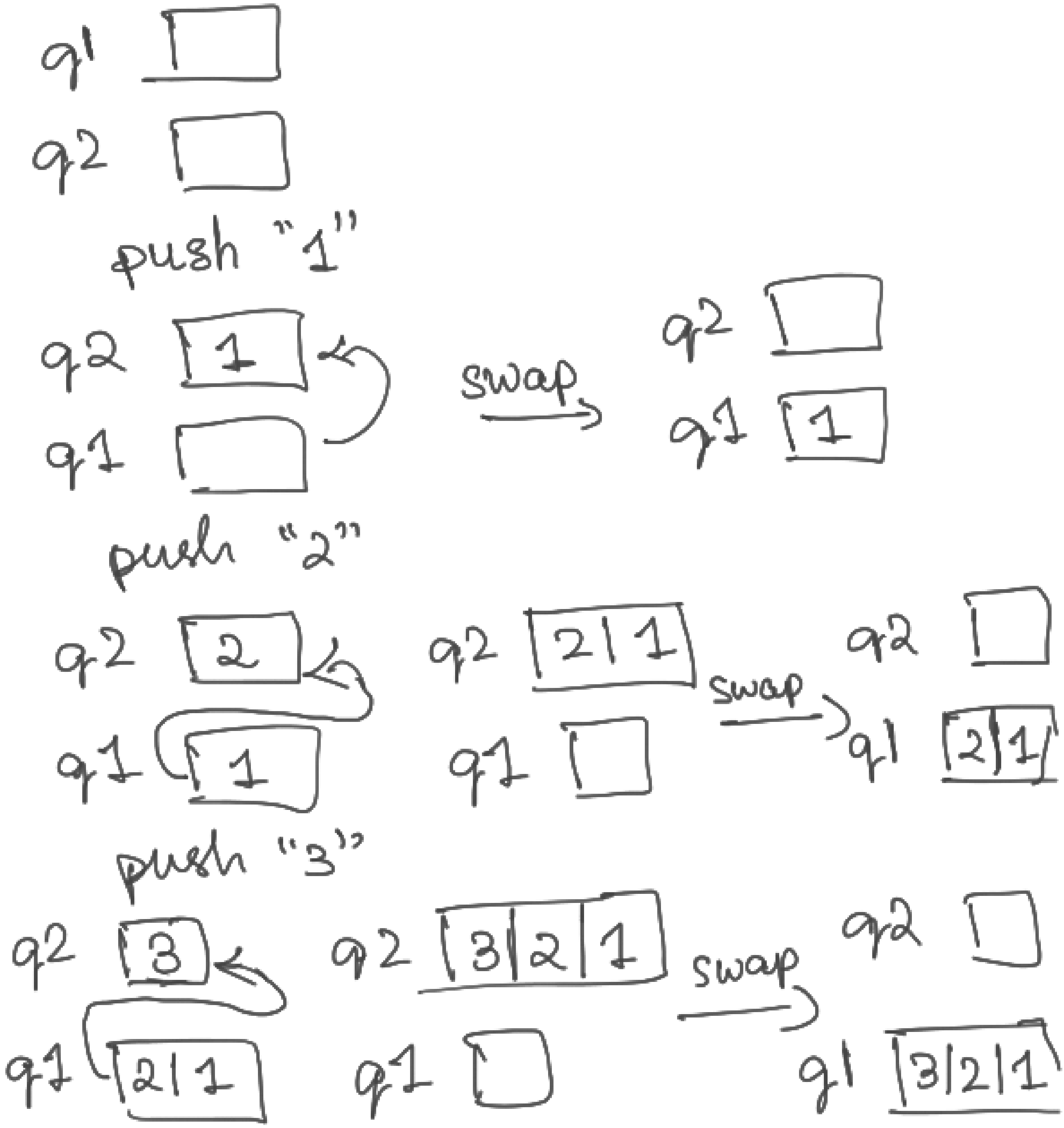
# Day 8: Stacks/Queues:

## 1 Implementation




Stack from Queue:

- Get 2 Queues: q1 & q2
- Add a new element to q2
- $q2 \xleftarrow{\text{add elements}} q1$
- swap q2 & q1



## ② Manipulations:

"Reverse a stack ... without creating any additional data structure."

input:


Recursion at rescue!

level #1

stack = [1, 2, 3]

temp = stack.pop() = 3

pop until bottom
add temp
stack = [3, 2, 1]

level #2

stack = [1, 2]

temp = stack.pop() = 2

pop until bottom
add temp
stack = [2, 1]

level #3

stack = [1]

temp = stack.pop() = 1

add temp
stack = [1]

## 3 Puzzles

"Sort a queue without extra space."

queue = [4, 2, 1, 3]

Do this N times:

1. Search for the minimum element in the unsorted part of the queue. Store it's value and the index.
2. Remove the elements occuring before the minimum element from the front of the queue and insert them at the end of the queue.
3. Once the index of the minimum element is reached, pop the minimum element ans store it in a variable.
4. Next, remove the elements occuring after the minimum element from the front of the queue and insert them at the end of the queue.
5. Finally, insert the minimum element (from 3) into the end of the queue.

unsorted

a. queue = [4, 2, 1, 3]

(step 2) = [1, 3, 4, 2]

pop & temp store

(step 3) = [3, 4, 2]

(step 4) = [4, 2, 3]

(step 5) = [4, 2, 3, 1]

b. queue = [4, 2, 3, 1]

= [2, 3, 1, 4]

pop & store

= [3, 1, 4]

= [4, 3, 1]

= [4, 3, 1, 2]

c. q = [4, 3, 1, 2]

= [3, 1, 2, 4]

pop & store

= [1, 2, 4]

= [1, 2, 4]

= [1, 2,

## 4 Typical uses

1. Queue : when you want to get things out in the order that you put them in
2. Stack : when you want to get things out in the reverse order that you put them in
3. Arrays : for randomly accessing any element.

---

Breadth-first walk : Queue (related to iterations)

Depth-first walk : Stack (related to recursion)

# Day 9: Trees:

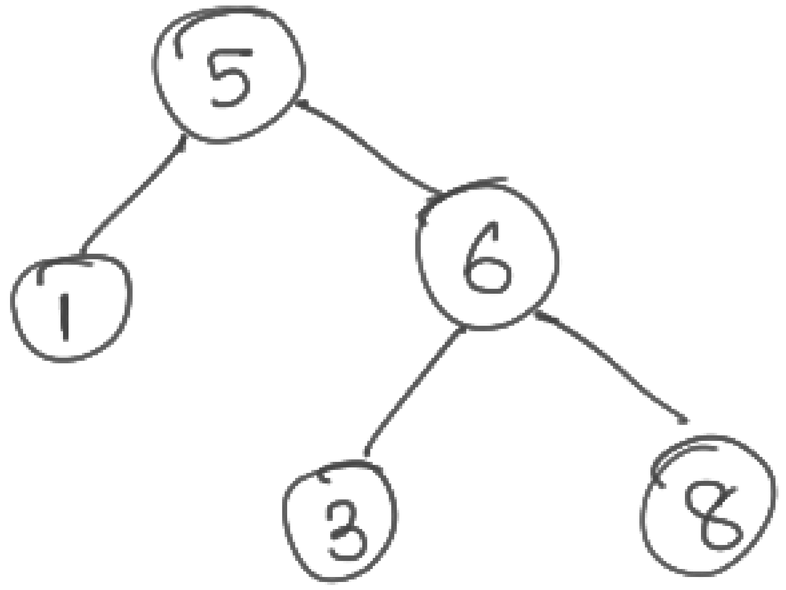
## 1 Binary Search Tree:

**Task: Given the root of a binary tree, determine if it is a valid binary search tree (BST).**

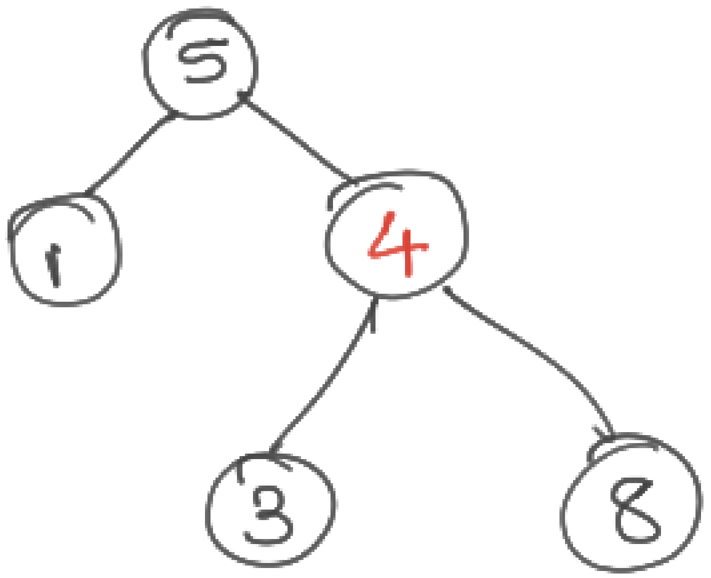
A valid BST is defined as follows:

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. Both the left and right subtrees must also be binary search trees.

BST ✓

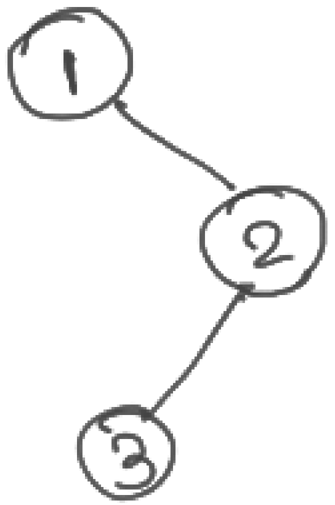



BST ✗




- recursively check for each subtree's root.
- return false if any of the 3 conditions of BST are violated.

## ② Depth First Search




In order: [1, 3, 2]

root value

(no left subtree)

right subtree

right val of right subtree

root of right subtree

In order: [left, root, right]

Pre order: [root, left, right]

Post order: [left, right, root]

- Can create copies of trees
- Can compare trees
- Can traverse in: in/pre/post orders

---

- Construct a tree from in- & post-order traversal

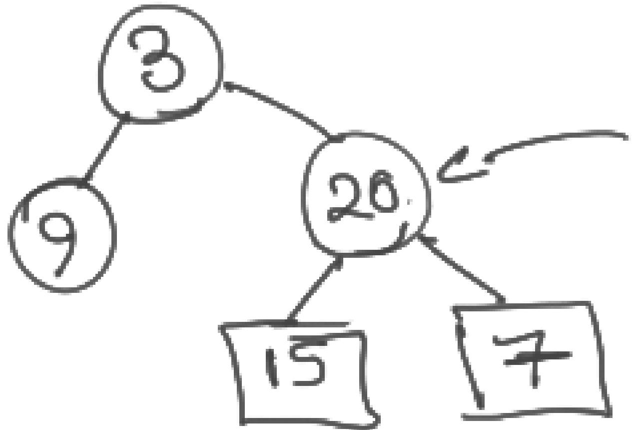
inorder: [9, 3, 15, 20, 7] post: [9, 15, 7, 20, 3]

a. Last element of post-order would be the root ③

b.


from in-order

c.

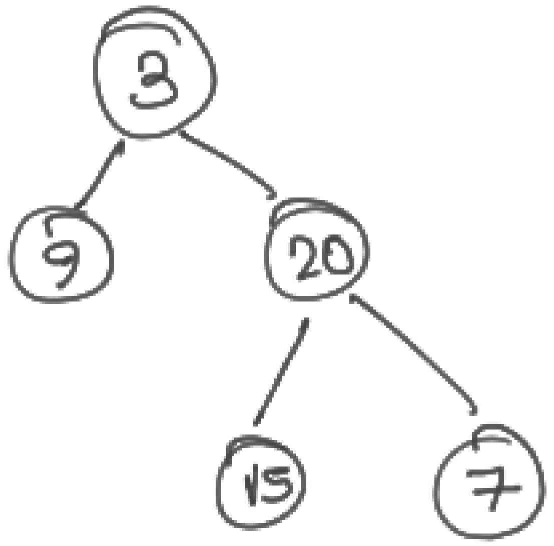



from post order

from in-order

# ③ Level Order Traversal

- just a fancy name for Breadth First Search




a. 3

b. 3, 9, 20

c. 3, 9, 20, 15, 7

- whenever you need to visit a tree level by level
  - max sum of level nodes
  - average of nodes on each level

## Speed:

BFS is slower than DFS for average cases.

## Time Complexity:

BFS : O(V + E)

#vertices #edges

DFS : O(V + E)

## Data Structures:

BFS : Queue

DFS : Stack

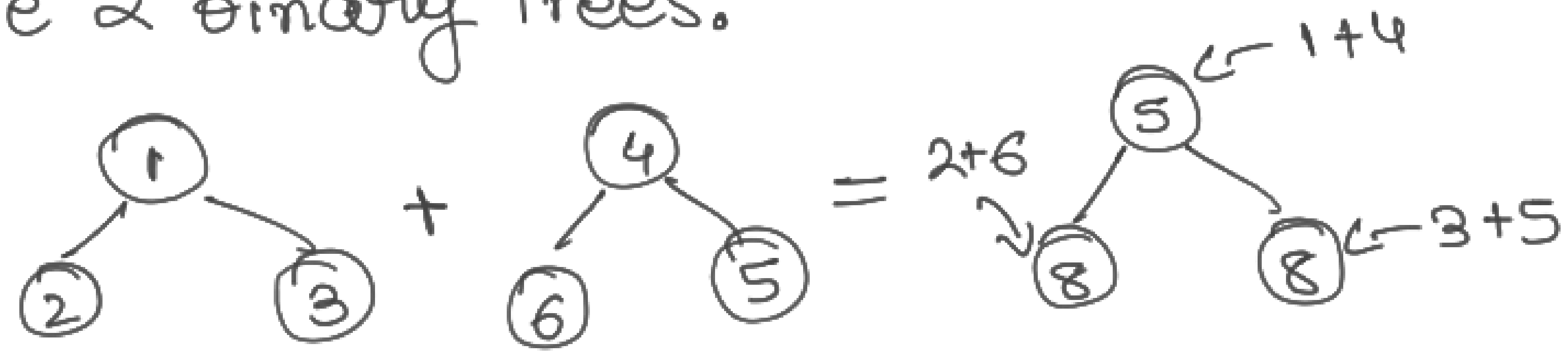
## (4) Additional Data

which search algo. would you use?

Q1: Find max depth of a binary tree.

Q2: Merge 2 binary trees.

e.g.




Q3: Find the largest value in each tree row.

Q4: Return all paths from a binary tree root to all the leafs.