Your Next Week

Saturday May 2	Sunday May 3	Monday May 4	Tuesday May 5
6:30 PM **— DUE Class 14 Mock Interviews** **— DUE Class 14 Lab** **— DUE Class 15 Reading** — Class 15 — Interview Prep 02 *MIDNIGHT* **— DUE Class 15 Learning Journal**	*MIDNIGHT* **— DUE CCW #1 Completed Personal Pitch** **— DUE CCW #1 Completed Resume** **— DUE Class 14-15 Feedback**		*6:30 PM* **— DUE Class 15 Lab** **— DUE Class 16 Reading** — Class 16A
Wednesday May 6	**Thursday May 7**	**Friday May 8**	**Saturday May 9**
6:30 PM — Class 16B *MIDNIGHT* **— DUE Class 16 Learning Journal**	*6:30 PM* — Co-working		*6:30 PM* **— DUE Class 16 Code Challenge** **— DUE Class 16 Lab** **— DUE Class 17 Reading** — Class 17 — Interview Prep 03 *MIDNIGHT* **— DUE Class 17 Learning Journal**

What We've Covered

Module 01

Javascript Fundamentals and Data Models

C01 — Node Ecosystem, TDD, CI/CD
C02 — Classes, Inheritance, Functional Programming
C03 — Data Modeling & NoSQL Databases
C04 — Advanced Mongo/Mongoose
C05 — DSA: Linked Lists

Module 02

API Servers

C06 — HTTP and REST
C07 — Express
C08 — Express Routing & Connected API
C09 — API Server
C11 — DSA: Stacks and Queues

Module 03

Auth/Auth

C10 — Authentication
C12 — OAuth
C13 — Bearer Authorization
C14 — Access Control (ACL)
C15 — DSA: Trees

Module 04

Realtime

C16 — Event Driven Applications
C17 — TCP Server
C18 — Socket.io
C19 — Message Queues
C20 — Midterms Prep

Midterms

Module 05

React Basics

C21 — Component Based UI
C22 — React Testing and Deployment
C23 — Props and State
C24 — Routing and Component Composition
C25 — DSA: Sorting and HashTables

Module 06

Advanced React

C26 — Hooks API
C27 — Custom Hooks
C28 — Context API
C29 — Application State with Redux
C30 — DSA: Graphs

Module 07

Redux State Management

C31 — Combined Reducers
C32 — Asynchronous Actions
C33 — Additional Topics
C34 — React Native
C35 — DSA: Review

Module 08

UI Frameworks

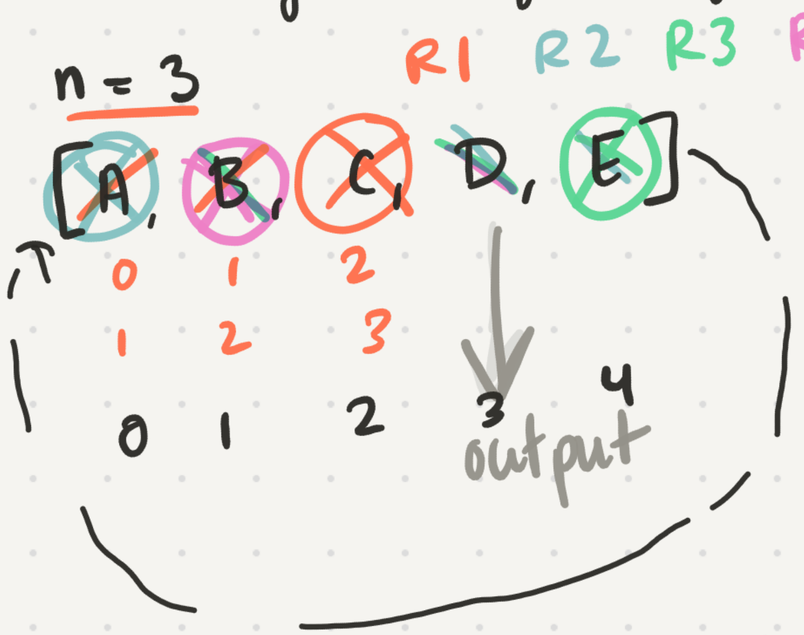
C36 — Gatsby and Next
C37 — JavaScript Frameworks
C38 — Finals Prep

Finals

Lab 14 Review

Code Challenge 14 Review

eeney meeney miney moe



length = 5

Array that we want to treat circularly

% modulo "mod"

i = 0

0 % 5 = 0

1 % 5 = 1

2 % 5 = 2

3 % 5 = 3

4 % 5 = 4

5 % 5 = 0

$$5 \overline{)\,5} \quad -5 \quad 0 \qquad \text{quotient } 1$$

$$5 \overline{)\,1} \quad -0 \quad 1 \qquad \text{quotient } 0$$

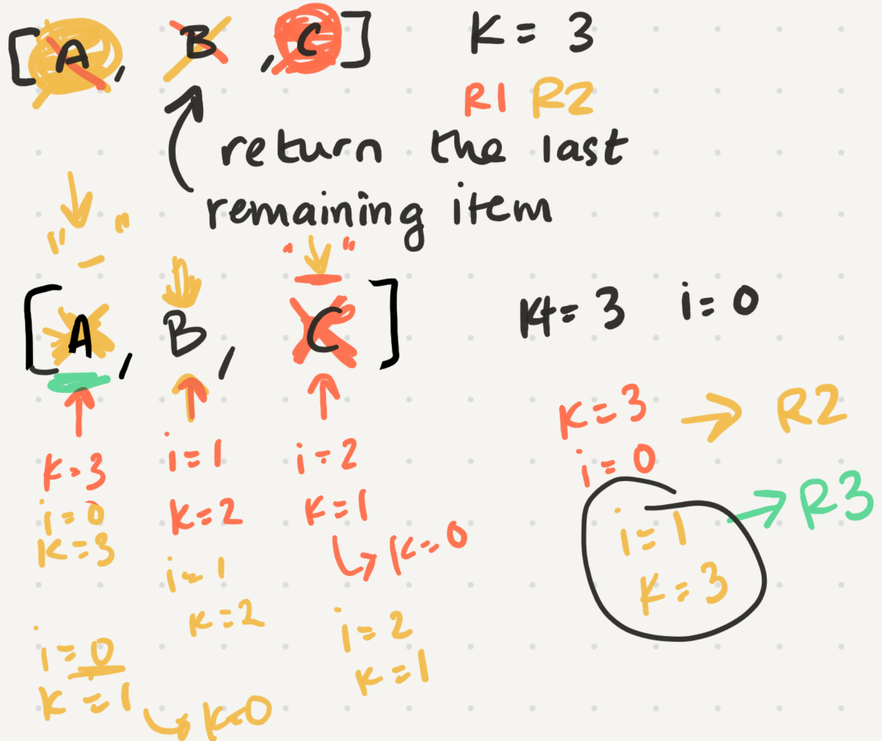
$$5 \overline{)\,2} \quad -0 \quad 2 \qquad \text{quotient } 0$$

Problem Domain

- Implement EMMM
- so that we can narrow down from a list & select one

Input ← "number of syllables", arr
Output → An item from the arr

Visual



Pseudo

```
foo( arr, k)
  i= 0;   ktracker = k;   li = -1;
while( i !== li    )
  → i = i % arr.length;
    if( arr[i] != "_" )
        ktracker --;
        li = i;
    if ( ktracker === 0 )
    → arr[i] = "_";
      ktracker = k;
    i++;
if( li === -1) throw err!
return arr[i];
```

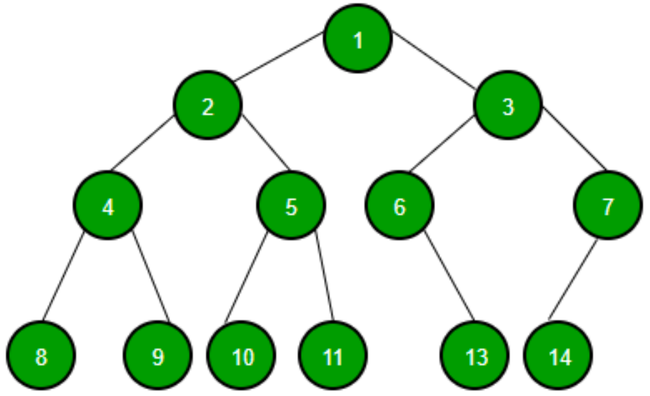
Class 15

DSA: Trees

seattle-javascript-401n16

Trees

- The **tree** data structure consists of parent and child nodes
- A parent can have any number of children, and each child can have its own children
- The top of the tree is the **root**
- The childless ends of the tree are the **leaves**
- We like **binary trees** the most!

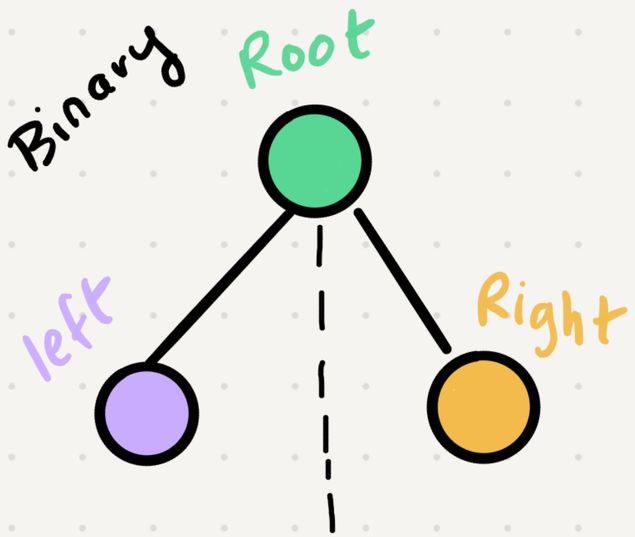


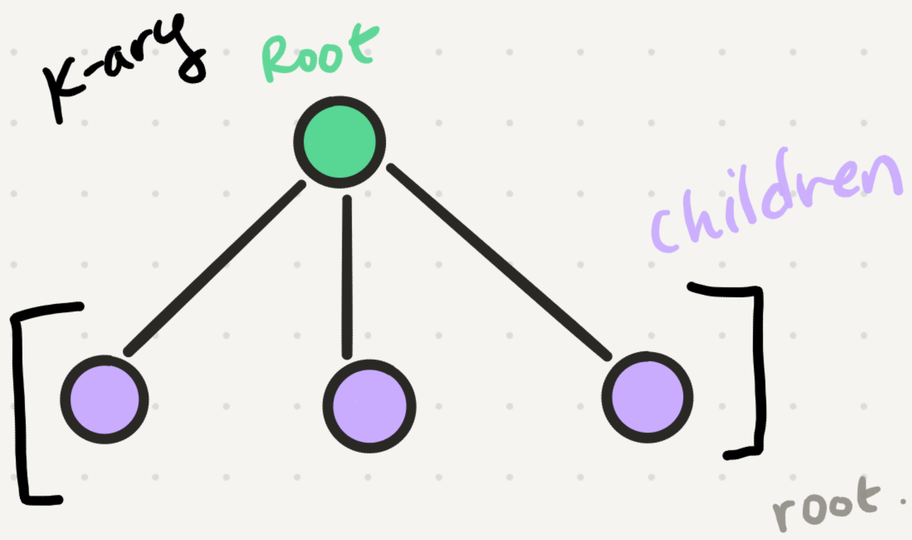
Traversal

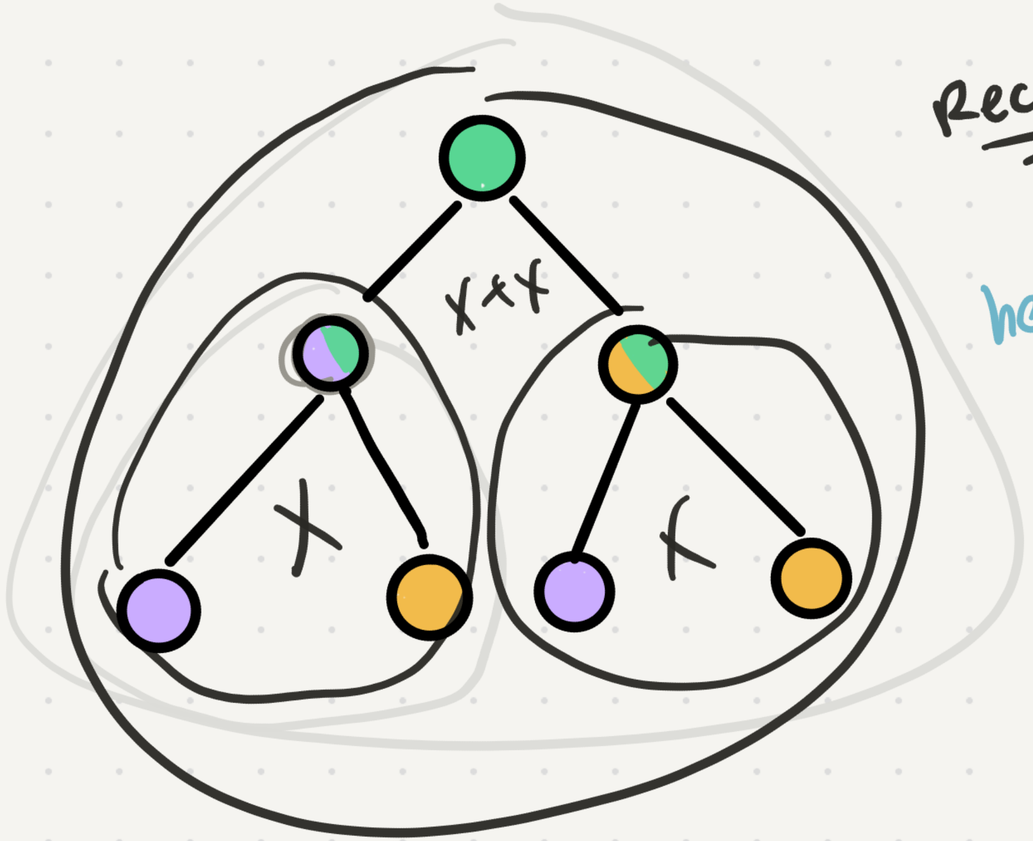
- Trees have multiple ways to **traverse** them
- **Depth-first traversal** is going top-down, and there are three types
  - PreOrder, InOrder, PostOrder
  - Uses recursion
- **Breadth-first traversal** is going left to right, using a queue

Balanced Trees

- A **binary search tree** is a sorted binary tree
    - We can find things quicker because we can cut out half of the tree each layer
- Time complexities for binary search tree searching is better when a tree is **balanced**
    - `O(n)` for unbalanced, `O(logn)` for balanced
- A balanced tree has the same number of left and right `descendants` (two nodes per parent)





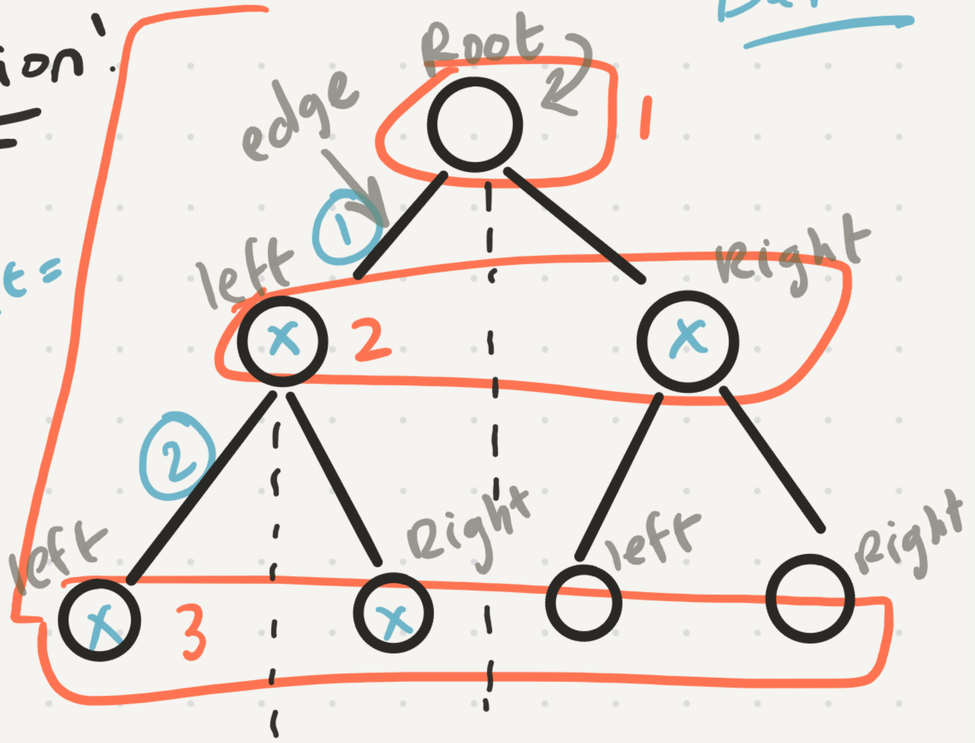


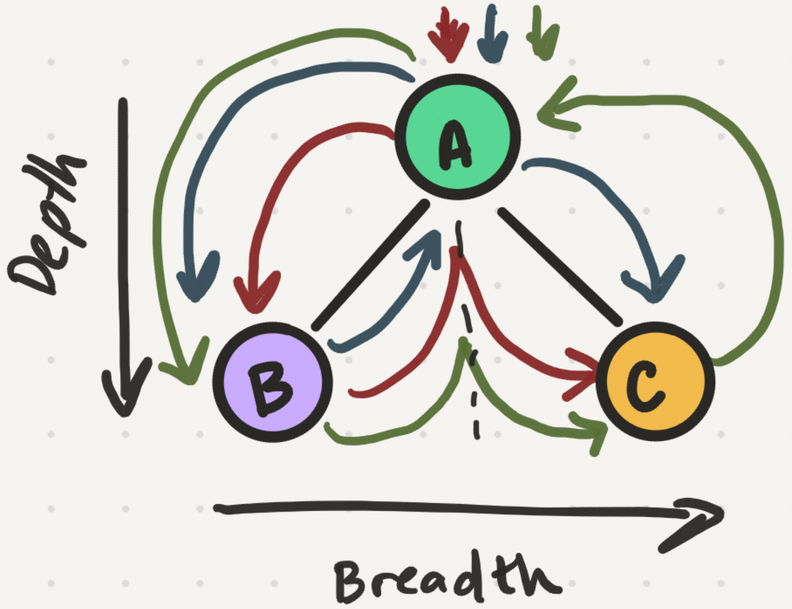
Recursion!

height = 2

root.left = Node

Balanced!





Depth first
Breadth first

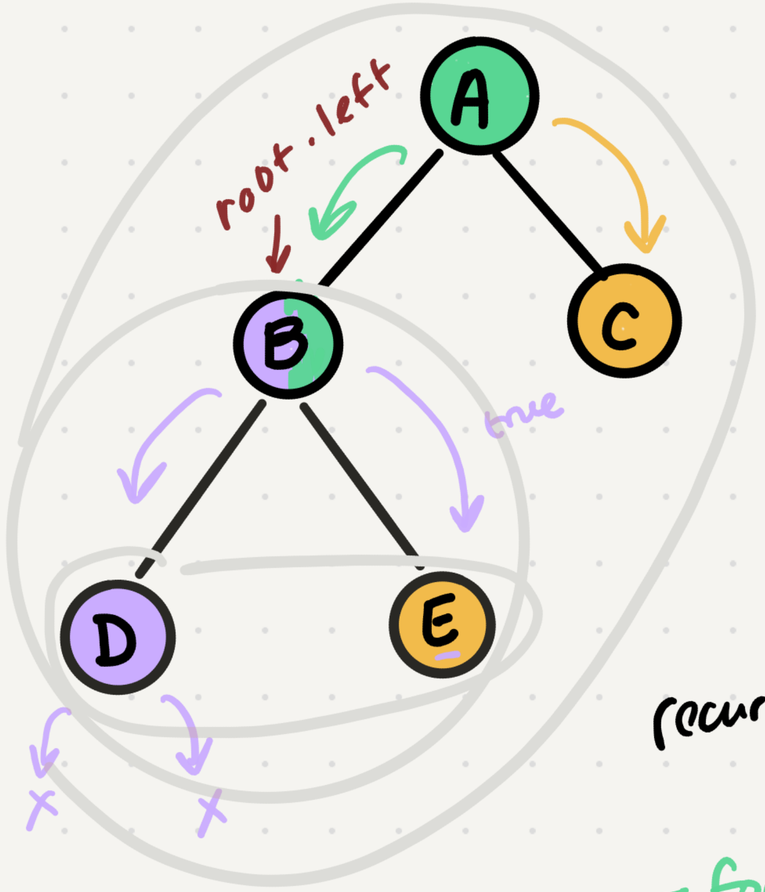
- preorder
- in order
- post order

pre, in, post relate to the root

Pre order = root first A B C

In order = root middle B A C

Post order = root last B C A



Pre order Traversal (Depth first)

↳ Root first

A B D E C

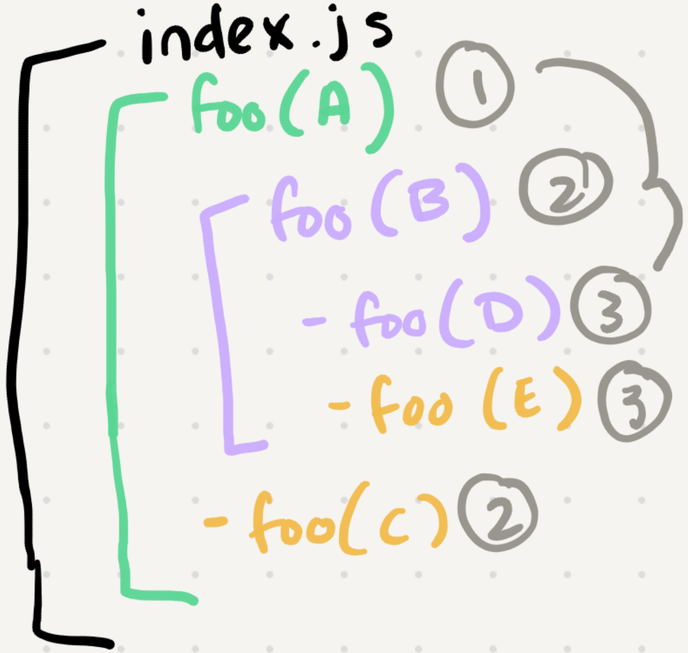
```
foo ( root)
  console.log (root.val);
  if (root.left)        true
    foo (root.left)     B D
  if (root.right)       true true
    foo (root.right)
```

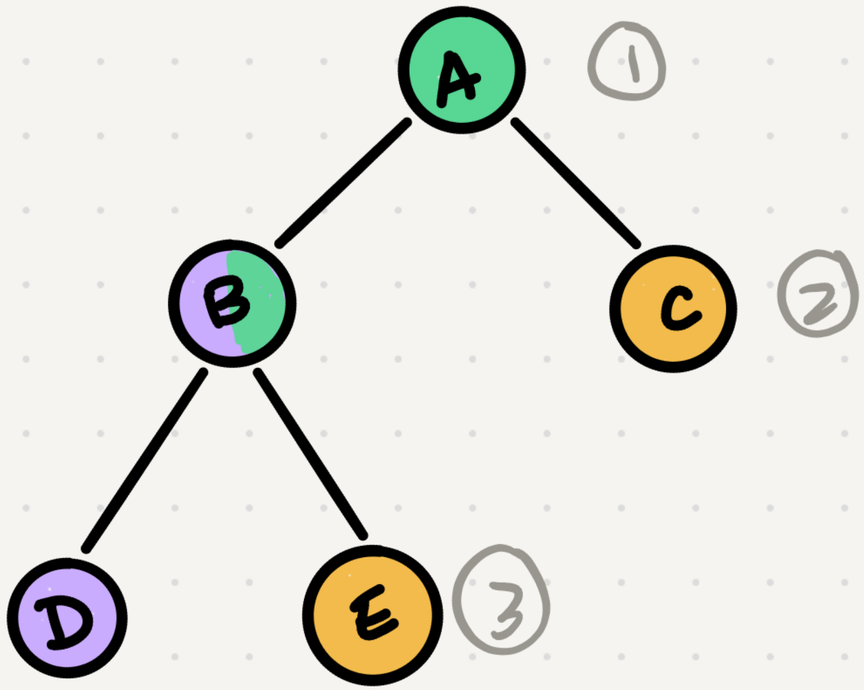
recursion!

Console

A B D E C

```
foo(A)
  foo(B)
    foo(D)
    foo(E)
  foo(C)
```

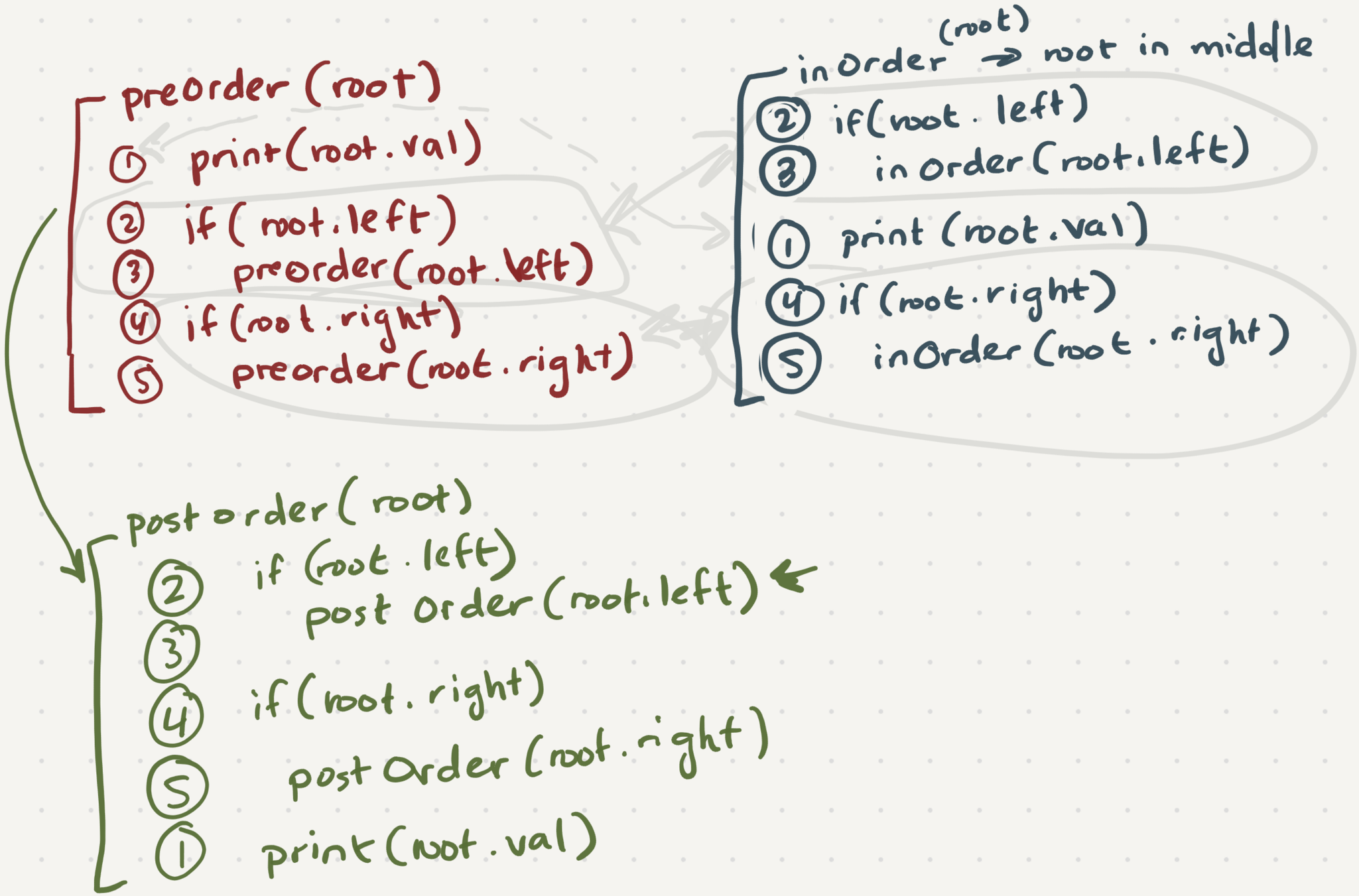


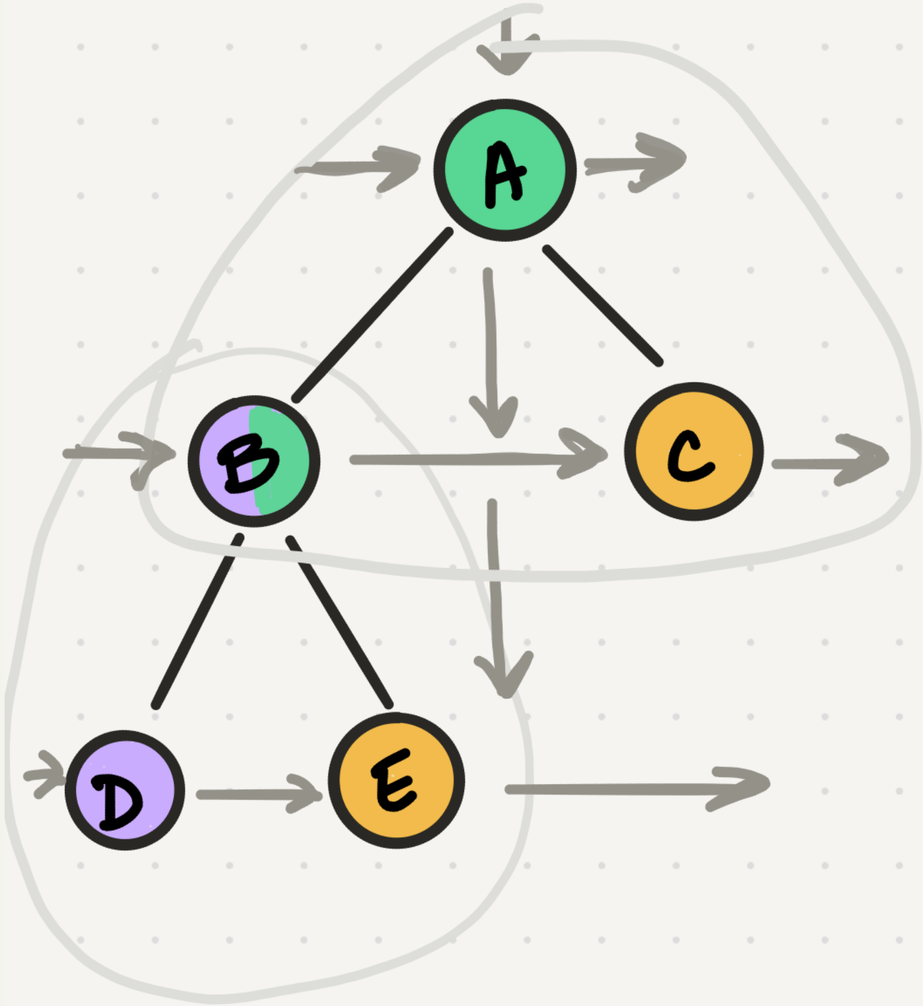


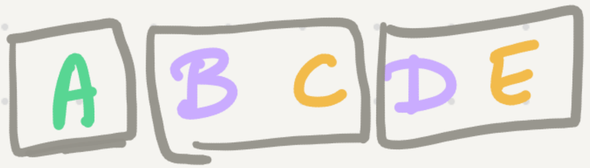
> node index.js

A B D E C

On the stack = running, not finished

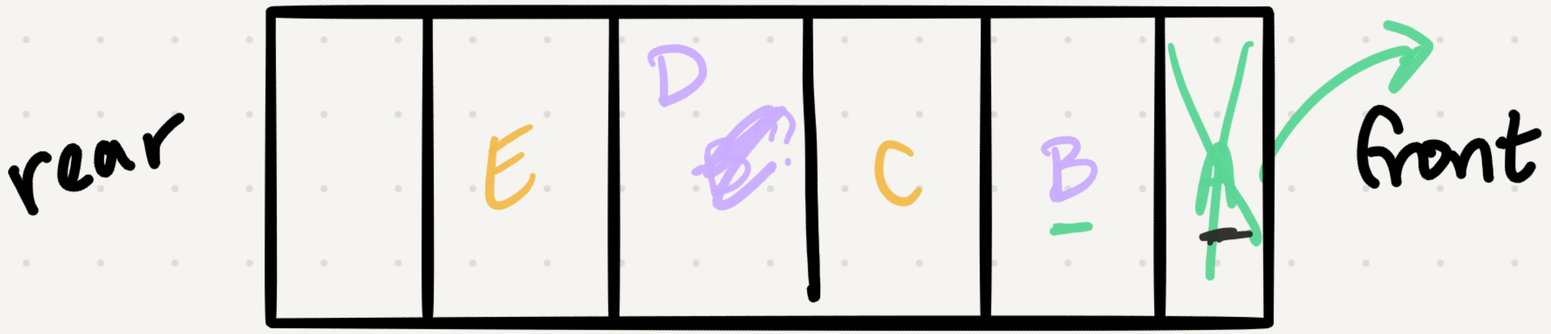

preorder (root)
① print(root.val)
② if (root.left)
③ preorder(root.left)
④ if (root.right)
⑤ preorder(root.right)
inOrder (root) → root in middle
② if(root.left)
③ inOrder(root.left)
① print (root.val)
④ if (root.right)
⑤ inOrder (root.right)
postorder (root)
② if (root.left)
post Order (root.left)
③
④ if (root.right)
⑤ post Order (root.right)
① print (root.val)





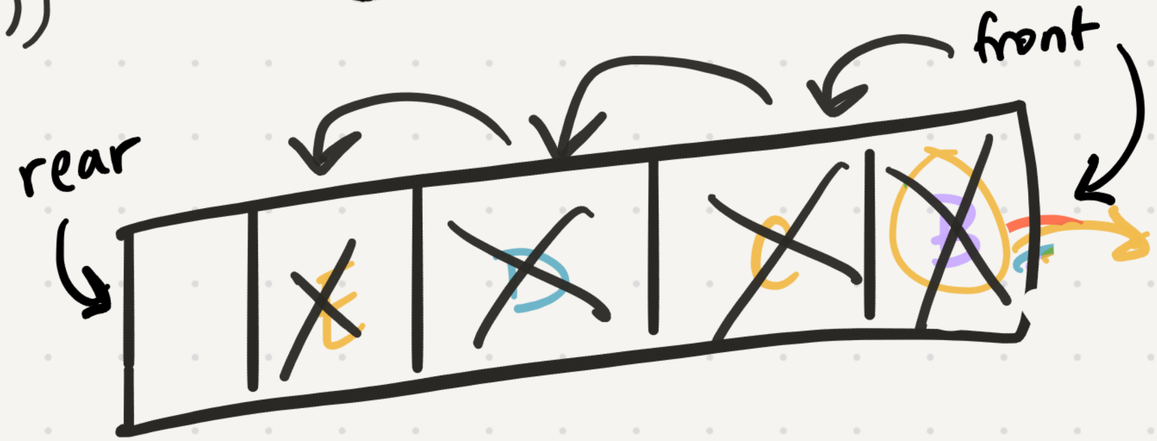

```
breadth first(root)
    enqueue(root);
    [if(root.left)
        enqueue(root.left)
    [if(root.right)
        enqueue(root.right)
    dequeue();
```

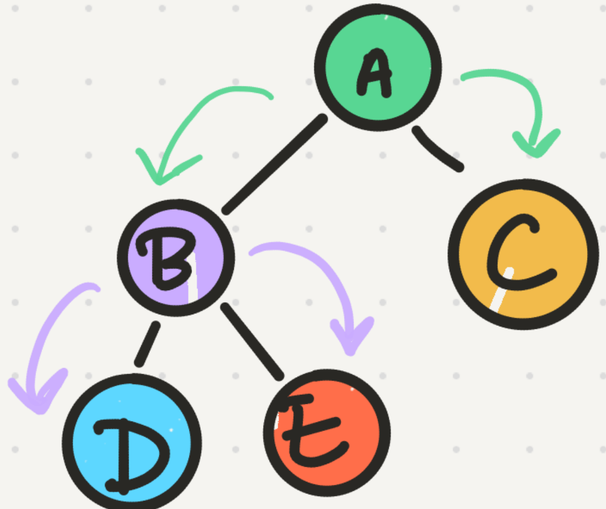
→ if queue was empty to start with else print root



A B C D E null exists exists exists

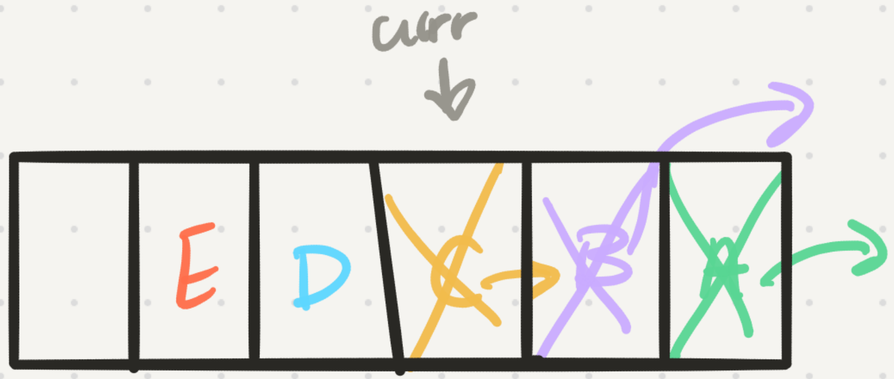
```
bf( root , queue?)
    let q = queue ? queue : new Queue()
    if (q.isEmpty())
        console.log(root.val)
    else
        console.log(q.dequeue())
    if(root.left)
      q.enqueue(root.left)
    if (root.right)
      q.enqueue(root.right)
    if (q.front)
        bf(q.front, q)
```

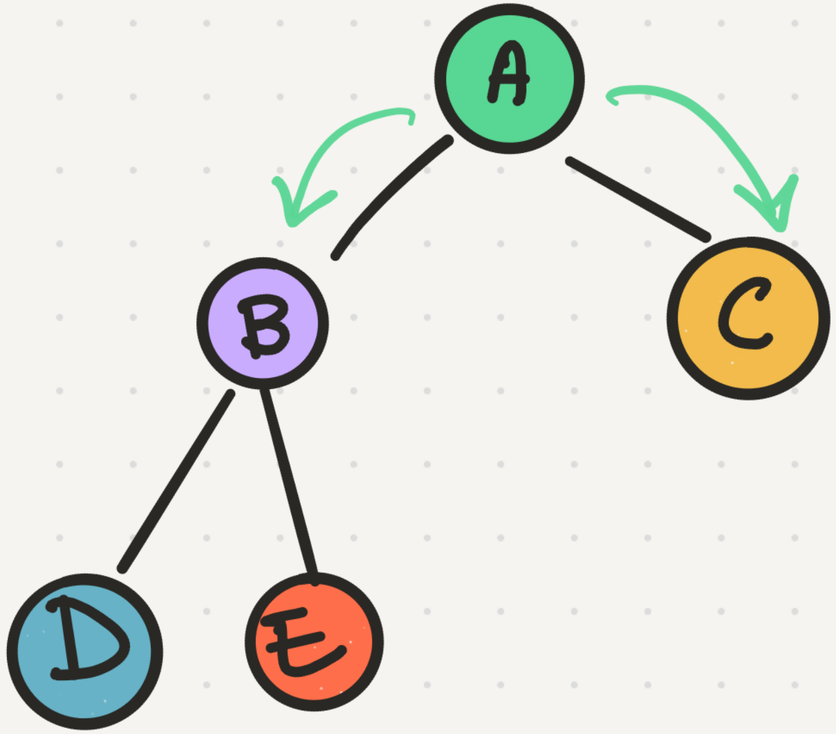


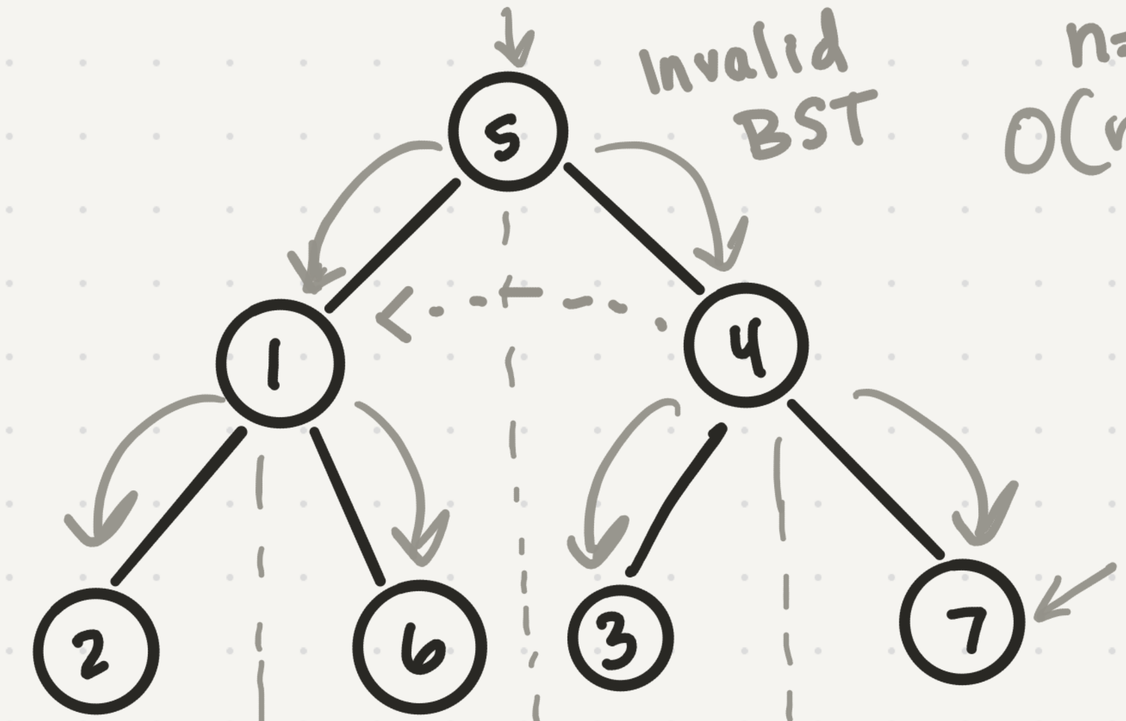


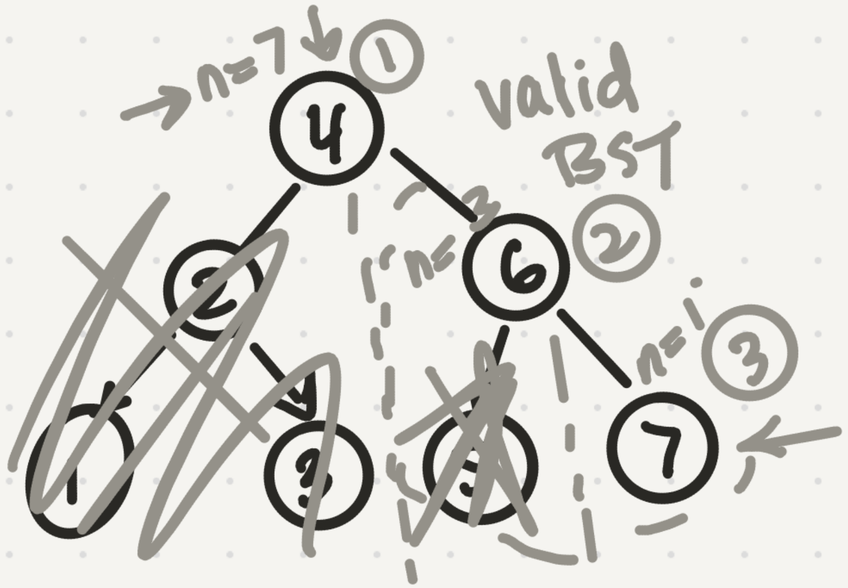
A

```
bf(root)
    let q = new Queue()
    q.enqueue(root)
    while(!q.isEmpty())
        let currNode = q.front;
      → q.dequeue();
        if(currNode.left)
      →   q.enqueue(currNode.left)
        if(currNode.right)
      ⇒   q.enqueue(currNode.right)
```

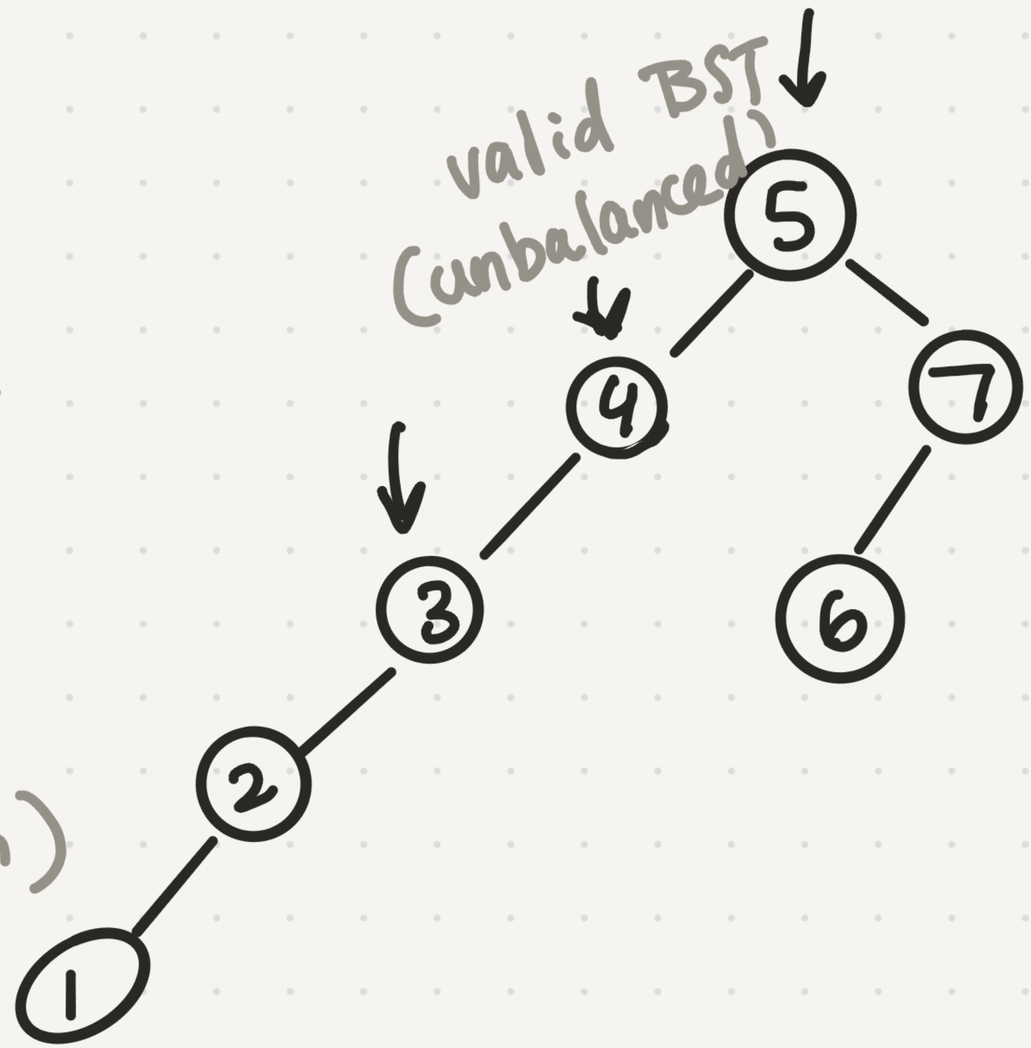





Invalid BST
n=7
O(n) all nodes checked
5
1
4
2
6
3
7


n=7
1
4
valid BST
2
6
n=3
n=1
3
7

n=7
O(3)
$O(\log_2 n)$


valid BST
(unbalanced)
5
4
7
3
6
2
1

Lab 15 Overview