Stack And Queue {level 1}

lecture 1

- Stack Basics
  - → Normal Stack
  - → Dynamic Stack
- Queue Basics
  - → Normal Queue
  - → Dynamic Queue
- Two Stacks in Array

lecture 2

- Stack using Queue
  - → Push efficient
  - → Pop efficient
- Queue using Stacks
  - → Push efficient
  - → Pop efficient
- Duplicate Brackets
- Balanced Brackets

Lecture (3)

- Next Greater to Right

HW
- NGE to Left
- NSE to Right
- NSE to Left

- Stock Span
- Largest Histogram
- Sliding window Maximum

Lecture (4)

Lecture (5)

- Celebrity Problem
- Minimum Stack
  - → with Extra Space
  - → w/o Extra Space
- Smallest No following Pattern

Stack & Queue

{ level ② }

{ Thursday & Weekends }

{ optional for FJP-2 }

lecture → ①, ②, ③

→ Circular Queue

→ Circular Deque

→ Maximum Stack

→ Max Frequency Stack

→ Middle Queue

→ Stack - Increment OP

→ K Stacks in Array

→ Reverse Stack

→ Reverse Queue

→ Sort Stack

→ Interleave Queue

→ Remove Outermost Parantheses

→ Longest valid Parantheses

→ Minimum Additions

→ minimum Removals - I & II

→ Score of Parantheses

→ Reverse Substrings

→ Trapping Rain Water - I & II

→ Maximum Area Container

→ Remove K Digits

→ Remove Duplicate letters

→ Lexico Smallest Subset

→ Basic Calculator - I, II, III

→ Expression Tree - Construct & Evaluate

→ Validate Stack Sequence

→ Asteroid Collision

→ Exclusive Time of Fn

→ No of Recent Calls

→ Valid Word After Substitution

{ Saturday → Evening }

h2 - lecture (1)

→ K Stacks in Array

→ Circular Queue (LL)

→ Circular Deque

→ Maximum Stack

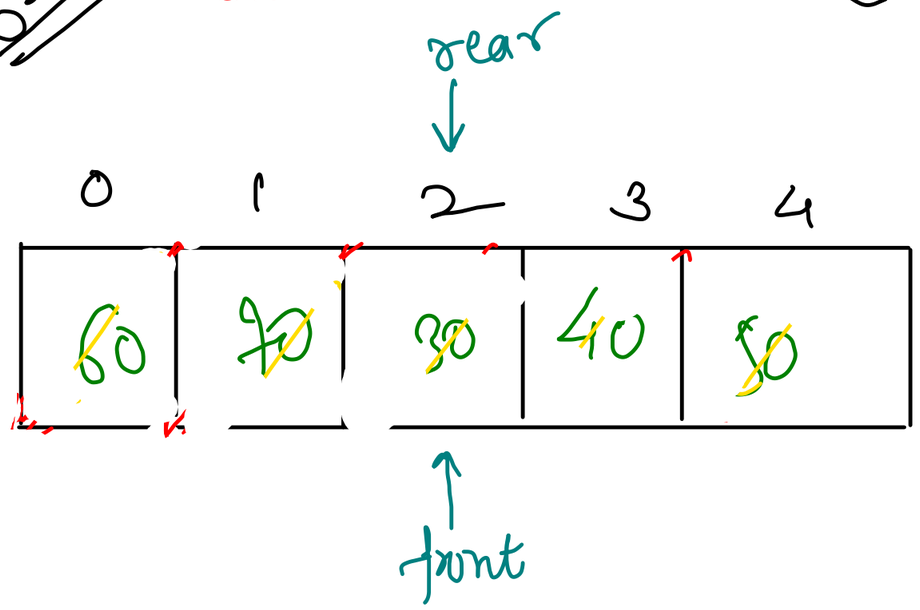
→ Max Frequency Stack

LL Remaining Ques

- → Big Integers
- → Rotate List
- → LFU
- → Flatten LL

LC 622

Circular Queue { Implementation of Queue using Array }



Size = 0 1 2 3 3 3 2 3 4 5 4
3 2 1 0

arr[rear] = val;
rear = (rear+1)%n;

push(10)
push(20)
push(30)
pop()
front = (front+1)%n
push(40)
pop()
push(50)
push(60)
push(70)
push(80) → Circular overflow

pop()
pop()
pop()
pop()
pop()
pop() → Queue underflow

```
int front = 0, rear = 0;
int size = 0;
int[] arr;

public MyCircularQueue(int k) {
    arr = new int[k];
}

public boolean enQueue(int value) {
    if(size == arr.length){
        // Stack overflow
        return false;
    }

    arr[rear] = value;
    rear = (rear + 1) % arr.length;
    size++;
    return true;
}
```

isFull()

```
public boolean deQueue() {
    if(size == 0){
        // Stack underflow
        return false;
    }

    front = (front + 1) % arr.length;
    size--;
    return true;
}

public int Front() {
    if(size == 0) return -1;
    return arr[front];
}

public int Rear() {
    if(size == 0) return -1;
    return arr[(rear - 1 + arr.length) % arr.length];
}
```

isEmpty()

isEmpty()

```
public boolean isEmpty() {
    return (size == 0);
}

public boolean isFull() {
    return (size == arr.length);
}
```

Deque (Doubly ended Queue)

Stack	Queue
top delete - O(1)	front delete → O(1)
top insert - O(1)	rear insert → O(1)

Singly Linked List as Deque

because remove last → O(n)

Deque

front insertion - O(1)
front removal - O(1)
rear insert - O(1)
rear delete - O(1)

Doubly Linked list {head, tail}

addFirst
removeLast
addLast
removeLast
→ O(1)

I&P-4 (Level 2)

Fees → 10-12

DP → 15

Graphs → 10

Bit → 5-7

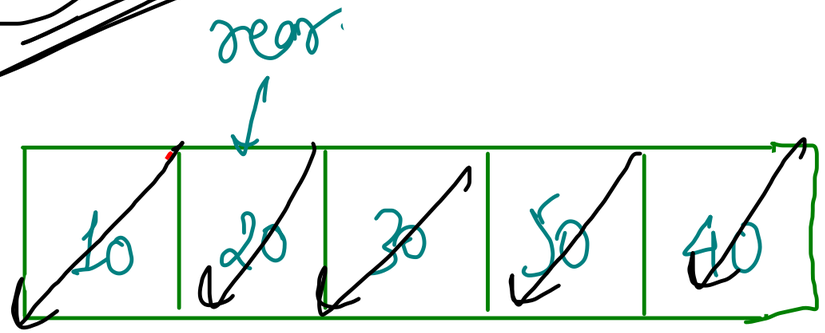
{HM, ARS} → 5-7

2 months

52 classes = 10 weeks

5 classes

(11) Deque



(1) addLast (10)

(2) addLast (20)

(3) addLast (30)

(4) addFirst (40)

(5) addFirst (50)

(6) removeFirst()

(7) removeFirst()

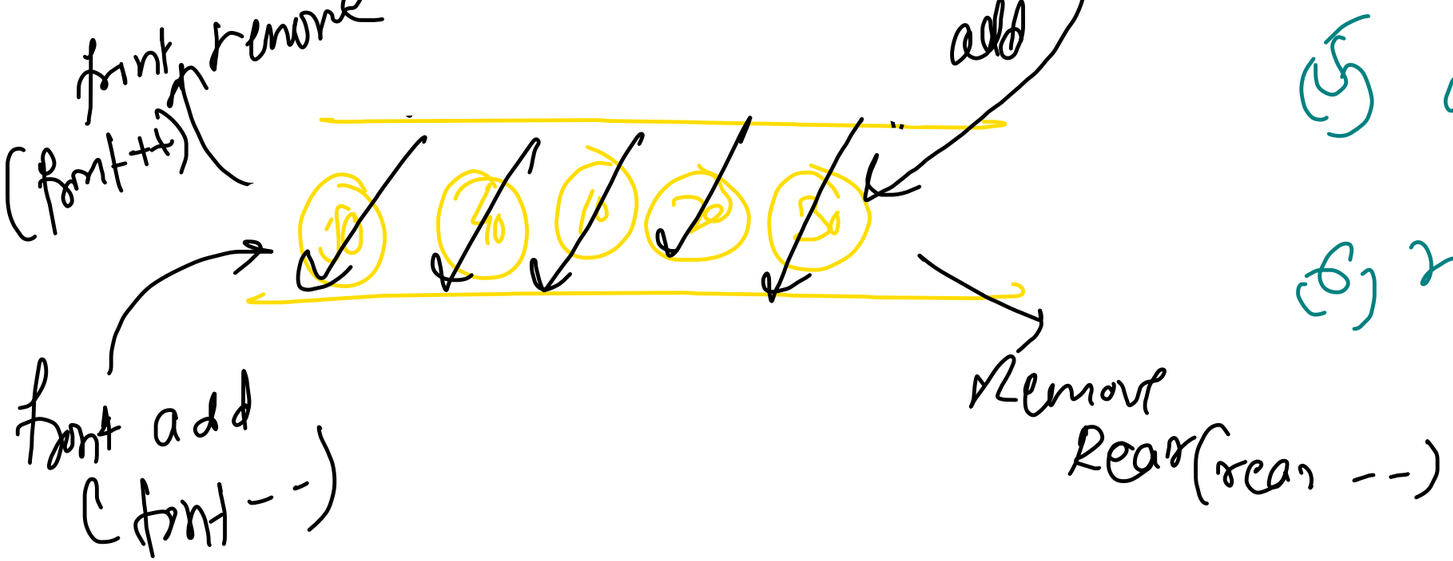
(8) removeLast()

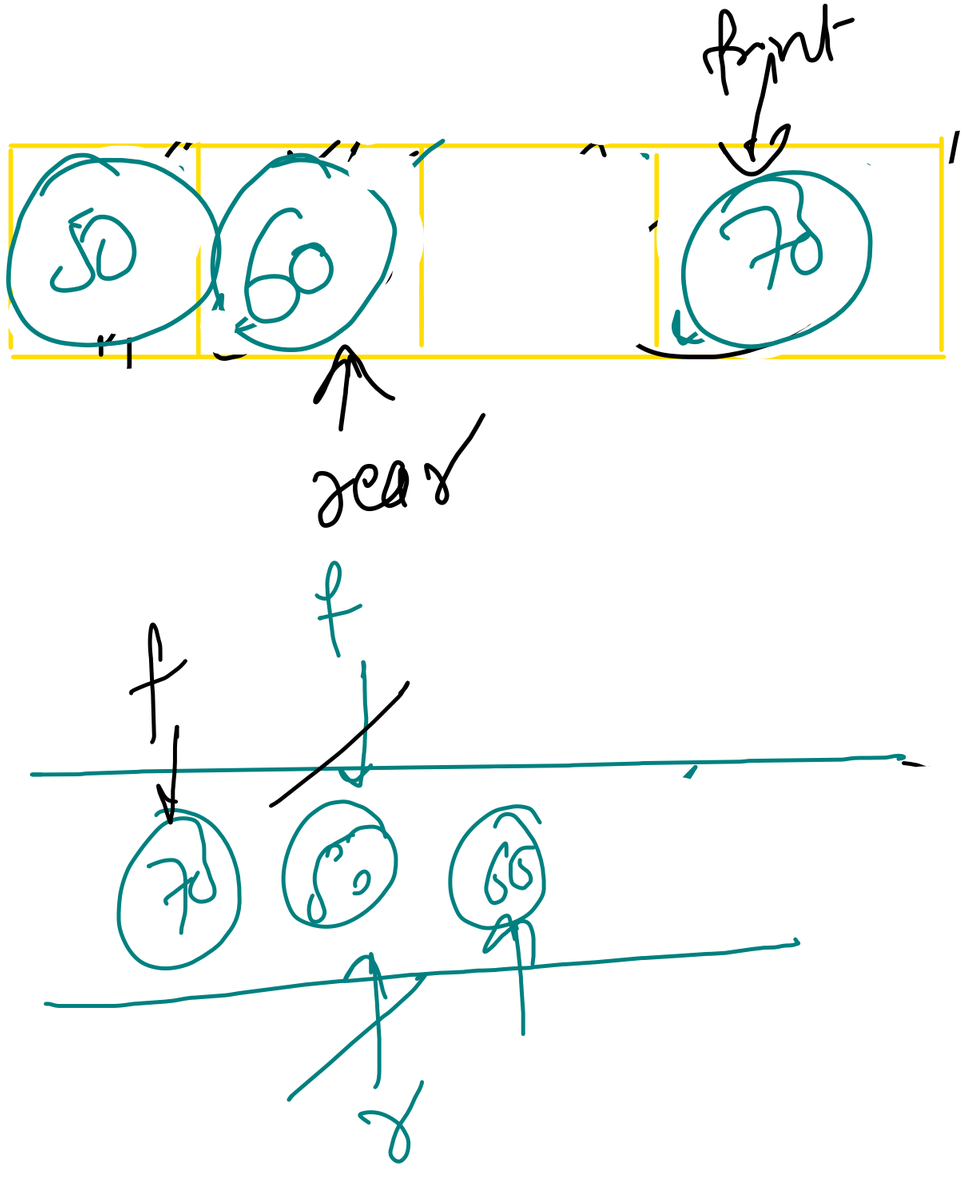
(9) removeLast()

(10) removeFirst()

rear--
return arr[rear]

↳ int val
= arr[front]
front++





addLast (10)

addLast (20)

addLast (30)

addFirst (40)

removeLast()

removeFirst()

removeFirst()

removeLast()

addFirst (50)

addLast (60)

addFirst (70)

641

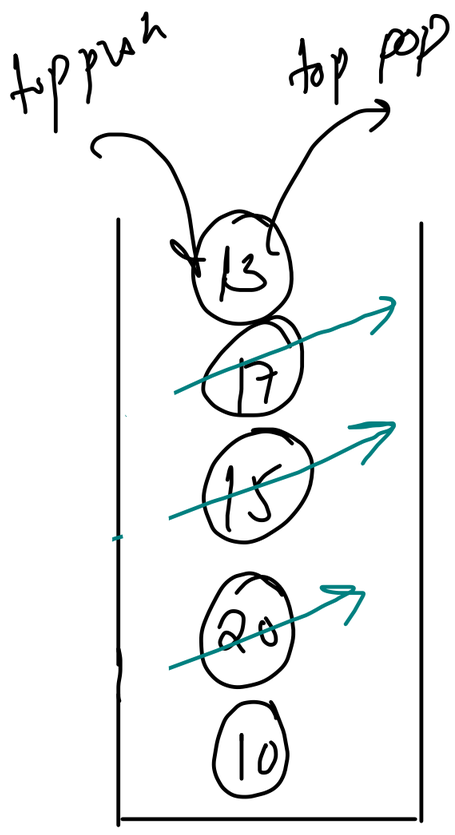
```java
int[] arr;
int front, rear, size;
public MyCircularDeque(int k) {
    arr = new int[k];
    front = rear = size = 0;
}
```

```java
public int getFront() {
    if(isEmpty()) return -1;
    return arr[front];
}

public int getRear() {
    if(isEmpty()) return -1;
    return arr[rear];
}

public boolean isEmpty() {
    return (size == 0);
}

public boolean isFull() {
    return (size == arr.length);
}
```

```java
public boolean insertFront(int value) {
    if(isFull()){
        // Deque Overflow
        return false;
    }

    if(size == 0) front = rear = 0;
    else front = (front - 1 + arr.length) % arr.length;

    arr[front] = value;
    size++;
    return true;
}

public boolean insertLast(int value) {
    if(isFull()){
        // Deque Overflow
        return false;
    }

    if(size == 0) front = rear = 0;
    else rear = (rear + 1) % arr.length;

    arr[rear] = value;
    size++;
    return true;
}
```

```java
public boolean deleteFront() {
    if(isEmpty()){
        // Deque underflow
        return false;
    }

    front = (front + 1) % arr.length;
    size--;
    return true;
}

public boolean deleteLast() {
    if(isEmpty()){
        // Deque underflow
        return false;
    }

    rear = (rear - 1 + arr.length) % arr.length;
    size--;
    return true;
}
```

Maximum Stack { LRU Cache }



① push (10)

② push (20)

③ push (15)

④ push (17)

⑤ push (13)

⑥ peek max () → 20

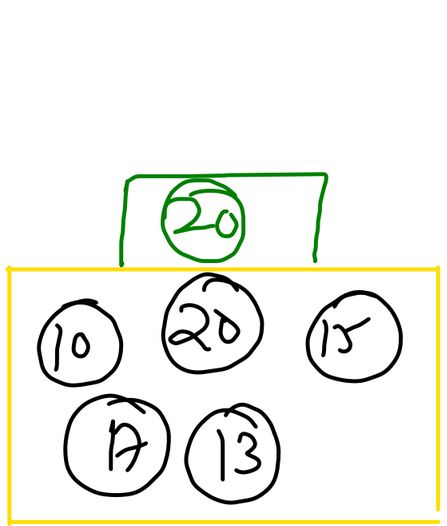
⑦ popMax ()

⑧ popMax()

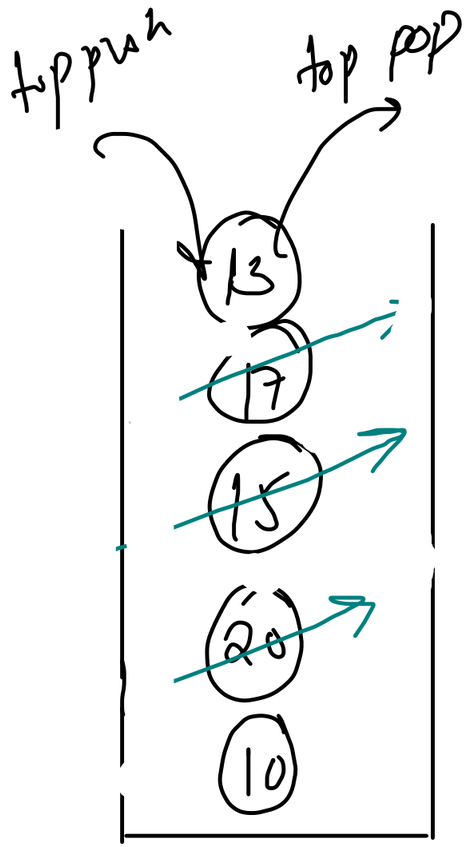
⑨ popMax()

Stack { push()
pop()
top/peek()
peekMax()
popMax() } O(n)

① Instead of Stack implementation using array, we will use DLL.



Max Heap



DLL

① push (10) → O(1) DLL, O(log N) PQ

② push (20)

③ push (15)

④ push (17)

⑤ push (13)

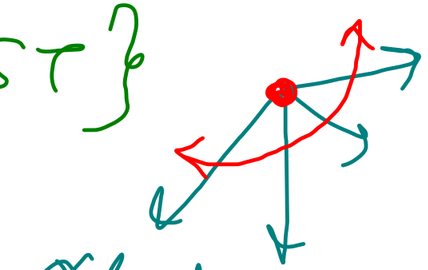
⑥ peek max() → 20 → O(1) Max get

⑦ popMax() → O(1) getmax, O(log n) delet, O(1) DLL removal

⑧ pop() → random ele from PQ is not possible

TreeMap { Self Balancing BST }

< Int, A< Node >>
↳ insertion

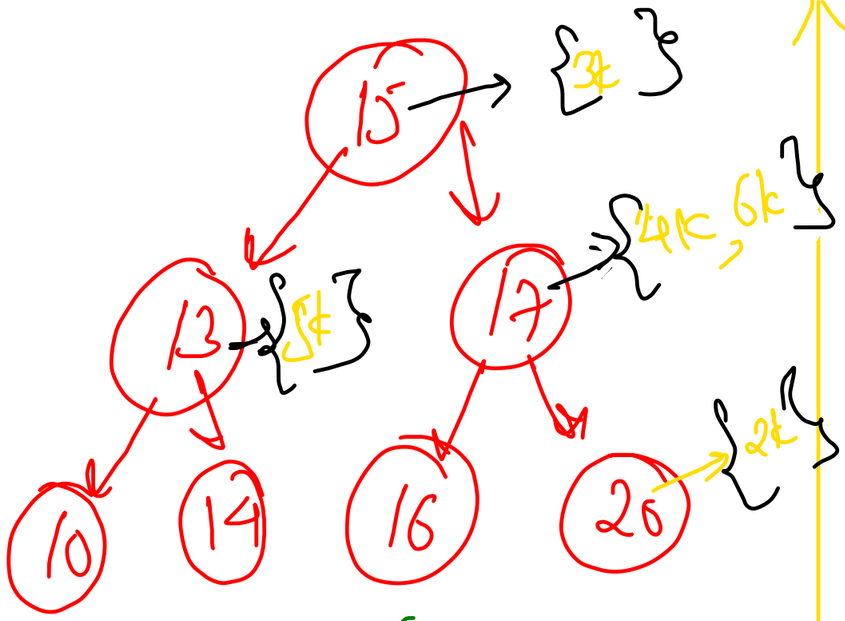


O(logN) pop

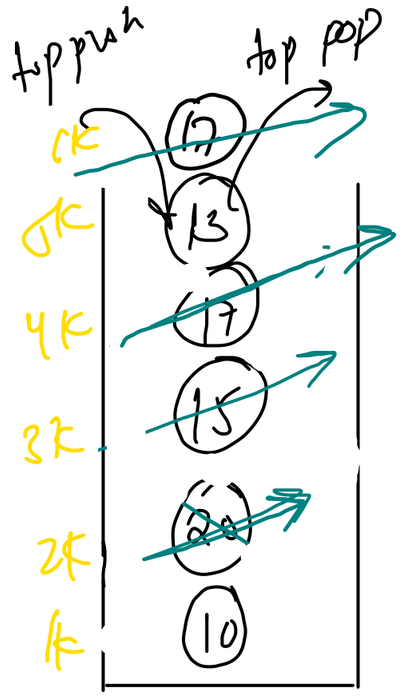
O(logN) peekmax

O(logN) push

O(logN) popmax



BST (Increasing order)



DLL (Insertion Stack)

② Instead of using max heap, use BST.

③ Duplicates handle cases

↓

Tree < Int, A<Node>>
map

Design

```java
public static class Node{
    int val;
    Node prev;
    Node next;
    Node(int val) {this.val = val;}
    Node(int val, Node prev, Node next){
        this.val = val;
        this.prev = prev;
        this.next = next;
    }
}

TreeMap<Integer, ArrayList<Node>> map;
Node head, tail;

public MaxStack() {
    head = new Node(-1);
    tail = new Node(-1);
    head.next = tail;
    tail.prev = head;
    map = new TreeMap<>();
}
```

```java
public void push(int x) {
    // Insert At Last in DLL
    Node curr = new Node(x, tail.prev, tail);
    tail.prev.next = curr;
    tail.prev = curr;

    // Insert in TreeMap
    if(map.containsKey(x) == false){
        map.put(x, new ArrayList<>());
    }
    map.get(x).add(curr);
}

public int pop() {
    // Delete Last Element From DLL
    Node curr = tail.prev;
    curr.prev.next = curr.next;
    curr.next.prev = curr.prev;

    // Delete Node From TreeMap
    ArrayList<Node> arr = map.get(curr.val);
    arr.remove(arr.size() - 1);
    if(arr.size() == 0) map.remove(curr.val);
    else map.put(curr.val, arr);
    return curr.val;
}
```

```java
public int top() {
    return tail.prev.val;
}

public int peekMax() {
    return map.lastEntry().getKey();
}

public int popMax() {
    int maxVal = peekMax();
    ArrayList<Node> arr = map.get(maxVal);
    Node curr = arr.get(arr.size() - 1);

    // Delete Last Element From DLL
    curr.prev.next = curr.next;
    curr.next.prev = curr.prev;

    // Delete Node From TreeMap
    arr.remove(arr.size() - 1);
    if(arr.size() == 0) map.remove(curr.val);
    else map.put(curr.val, arr);
    return curr.val;
}
```

Design a stack-like data structure to push elements to the stack and pop the most frequent element from the stack.

Implement the `FreqStack` class:

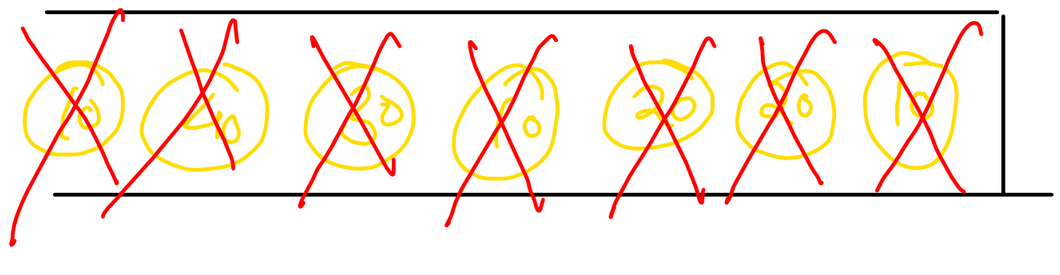
- `FreqStack()` constructs an empty frequency stack.
- `void push(int val)` pushes an integer `val` onto the top of the stack.
- `int pop()` removes and returns the most frequent element in the stack.
  - If there is a tie for the most frequent element, the element closest to the stack's top is removed and returned.

Stack

push(10) push(40)
push(20) push(10)
push(20)
push(10)
push(30)

pop() all nodes

Node vs Freq Int, Int



Frequency stack

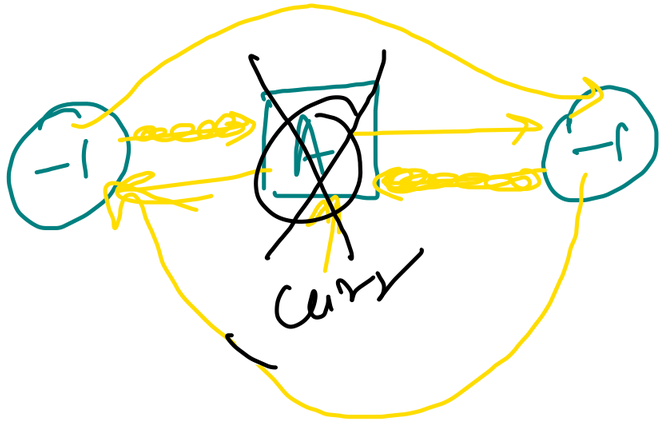
↳ bucketSort

freq vs Nodes

<int, A<int>>

```java
HashMap<Integer, Integer> freq = new HashMap<>();
HashMap<Integer, ArrayList<Integer>> freqNodes = new HashMap<>();
int maxFreq = 0;

public void push(int val) {
    freq.put(val, freq.getOrDefault(val, 0) + 1);
    int currFreq = freq.get(val);

    if(freqNodes.containsKey(currFreq) == false){
        freqNodes.put(currFreq, new ArrayList<>());
    }

    freqNodes.get(currFreq).add(val);
    if(currFreq > maxFreq) maxFreq = currFreq;
}
```

```java
public int pop() {
    ArrayList<Integer> nodes = freqNodes.get(maxFreq);

    int val = nodes.get(nodes.size() - 1);
    nodes.remove(nodes.size() - 1);
    freq.put(val, freq.get(val) - 1);

    if(nodes.size() == 0) {
        freqNodes.remove(maxFreq);
        maxFreq--;
    }

    return val;
}
```

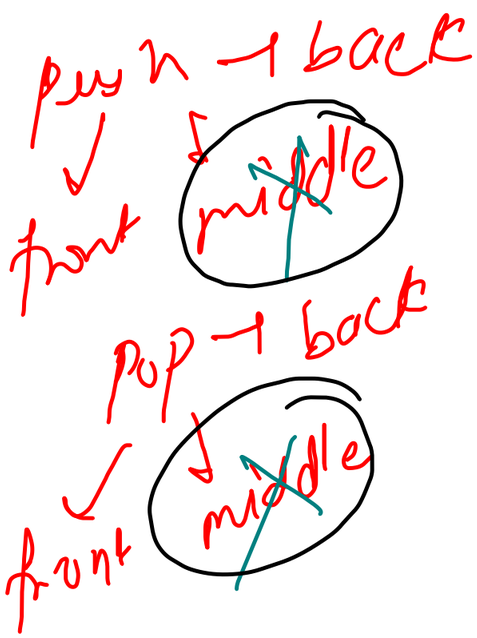


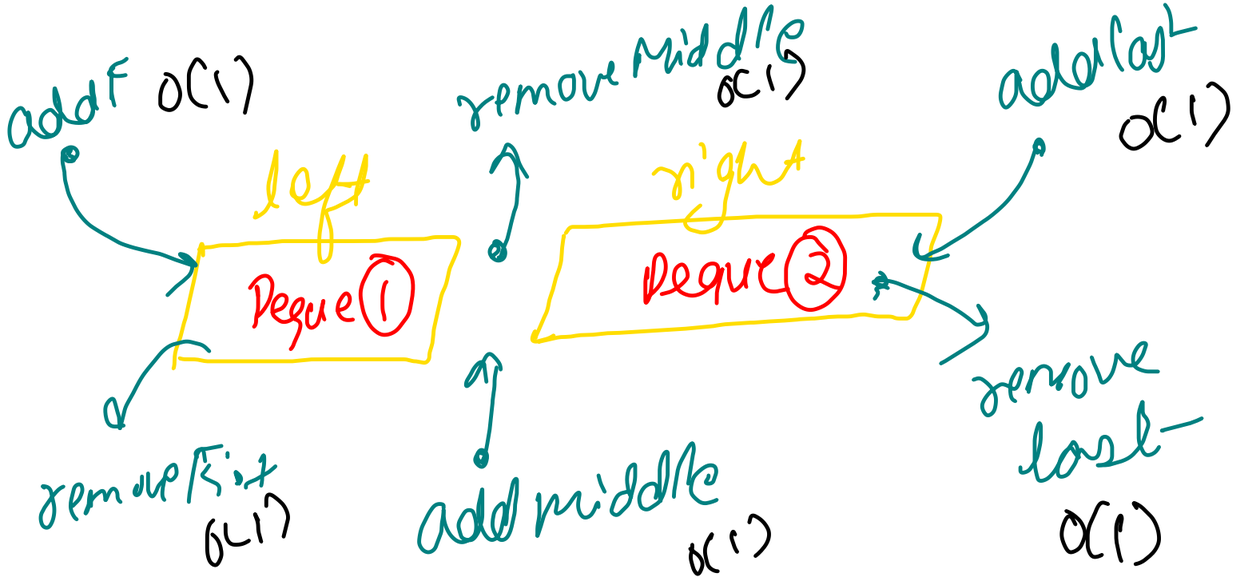
→ curr.prev.next = curr.next

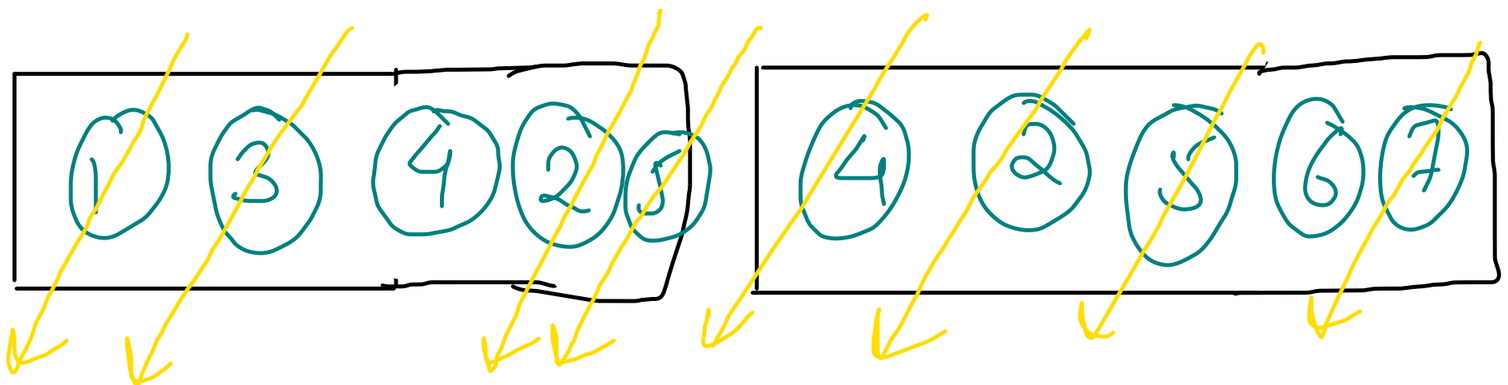
→ curr.next.prev = curr.prev

Front Middle Back Queue

KC 1670



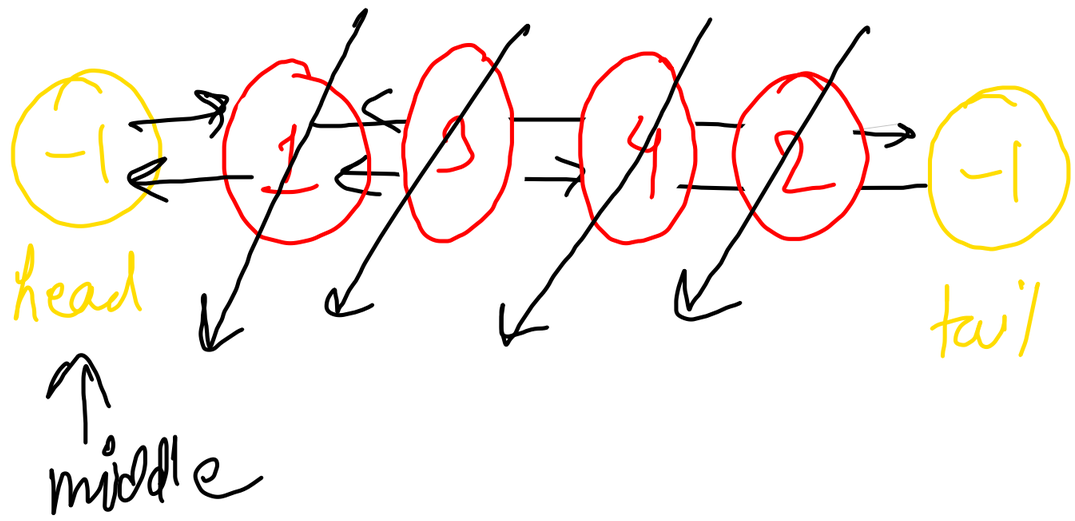





```
q.pushFront(1);
q.pushBack(2);
q.pushMiddle(3);
q.pushMiddle(4);
q.popFront();
q.popMiddle();
q.popBack();
q.popFront();
```

q.pushBack(5)
q.pushBack(6)
q.pb(7)

```
q.pushFront(1);
q.pushBack(2);
q.pushMiddle(3);
q.pushMiddle(4);
q.popFront();
q.popMiddle();
q.popMiddle();
q.popBack();
q.popFront();
```



2nd approach

using DLL

q.pushBack(5)

q.pushBack(6)

q.pb(7)

RC 1670

```
Deque<Integer> left = new ArrayDeque<>();
Deque<Integer> right = new ArrayDeque<>();

public void balance(){
    if((left.size() + right.size()) % 2 == 0){
        while(left.size() > right.size()){
            int val = left.removeLast();
            right.addFirst(val);
        }
        while(right.size() > left.size()){
            int val = right.removeFirst();
            left.addLast(val);
        }
    } else {
        while(left.size() > right.size() + 1){
            int val = left.removeLast();
            right.addFirst(val);
        }
        while(right.size() + 1 > left.size()){
            int val = right.removeFirst();
            left.addLast(val);
        }
    }
}
```

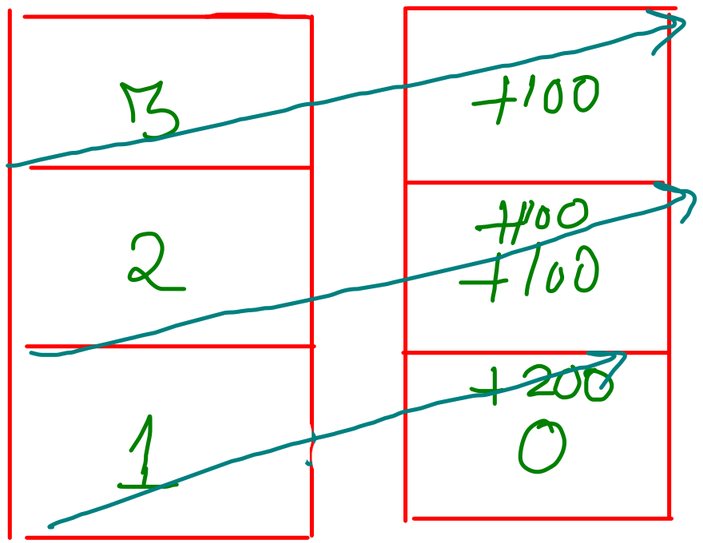
```
public void pushFront(int val) {
    left.addFirst(val);
    balance();
}

public void pushMiddle(int val) {
    if(left.size() > right.size()){
        right.addFirst(left.getLast());
        left.removeLast();
    }

    left.addLast(val);
    balance();
}

public void pushBack(int val) {
    right.addLast(val);
    balance();
}
```

```
public int popFront() {
    if(left.size() + right.size() == 0) return -1;
    int val = left.getFirst();
    left.removeFirst();
    balance();
    return val;
}

public int popMiddle() {
    if(left.size() + right.size() == 0) return -1;
    int val = left.getLast();
    left.removeLast();
    balance();
    return val;
}

public int popBack() {
    if(left.size() + right.size() == 0) return -1;
    if(right.size() == 0) {
        int val = left.getLast();
        left.removeLast();
        balance();
        return val;
    } else {
        int val = right.getLast();
        right.removeLast();
        balance();
        return val;
    }
}
```

Design Stack with Increment operation

LC 1381

```
3
push 1
push 2
pop
push 2
push 3
push 4
increment 2 100
increment 5 100
pop
pop
pop
pop
```

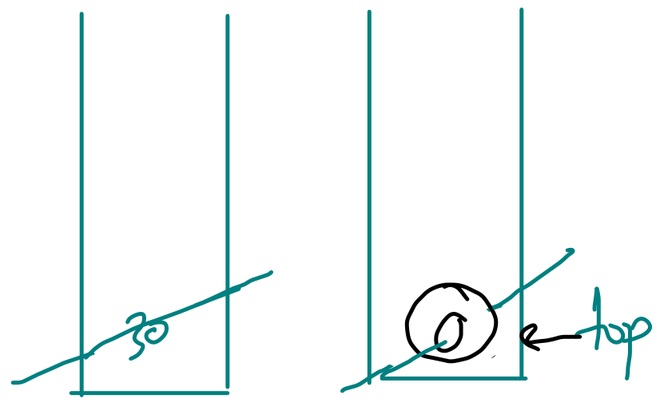


- `CustomStack(int maxSize)` Initializes the object with `maxSize` which is the maximum number of elements in the stack or do nothing if the stack reached the `maxSize`.
- `void push(int x)` Adds `x` to the top of the stack if the stack hasn't reached the `maxSize`.
- `int pop()` Pops and returns the top of stack or **-1** if the stack is empty.
- `void inc(int k, int val)` Increments the bottom `k` elements of the stack by `val`. If there are less than `k` elements in the stack, just increment all the elements in the stack.

```java
int[] arr;
int[] inc;
int top = -1;

public CustomStack(int maxSize) {
    arr = new int[maxSize];
    inc = new int[maxSize];
}

public void push(int x) {
    if(top == arr.length - 1){
        // Stack overflow
        return;
    }

    top++;
    arr[top] = x;
}
```

```java
public int pop() {
    if(top == -1){
        // Stack underflow
        return -1;
    }

    int val = arr[top] + inc[top];
    if(top - 1 >= 0)
        inc[top - 1] += inc[top];

    inc[top] = 0;
    top--;
    return val;
}
```

```java
public void increment(int k, int val) {
    if(top == -1) return;

    k = Math.min(k, top + 1);
    inc[k - 1] += val;
}
```




```
["CustomStack","pop","increment","push","increment","increment","increment","pop","increment"]
[[30],[],[3,40],[30],[4,63],[2,79],[5,57],[],[5,32]]
```

output = -1, 229

12 Stacks in Array

GFG { love Babbar } HARD

arr:

0	1	2	3	4	5
10	20	30	40	50	60

next / prev:

3	4	1	2	5	-1

top:

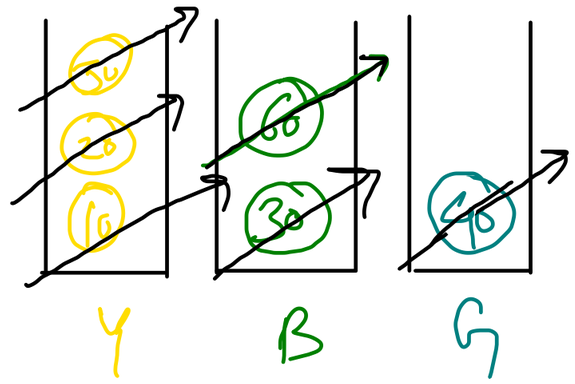
Yellow	Blue	Green
-1	-1	-1

K = 3

free idx = 0

- push (10, Y)
- push (20, Y)
- push (30, B)
- push (40, G)
- push (50, Y)
- push (60, B)
- push (70, G)
  - ↳ free == -1 : Stack overflow

- pop (B)
- pop (Y)
- pop (Y)
- pop (B)
- pop (G)
- pop (Y)
- pop (Y) → top(Y) == -1
  - ↳ Stack underflow



ith index is free,

push → next (i) = next free element's index

else

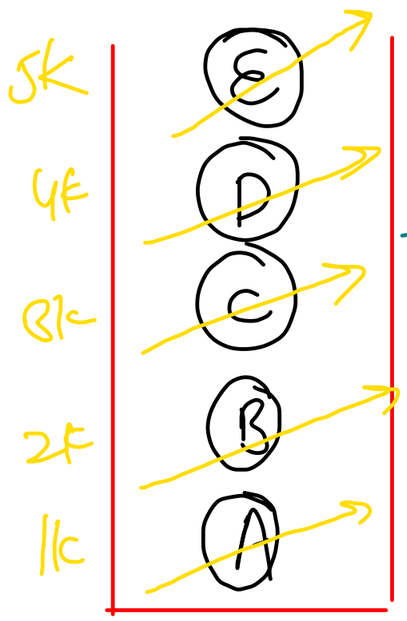
pop → next (i) = previous value' index of top stack

```java
int[] arr;
int[] next;
int[] top;
int freeIdx;

public NStack(int K, int N) {
    arr = new int[N];
    next = new int[N];
    for(int i=0; i<N; i++){
        next[i] = i + 1;
    }
    next[N - 1] = -1;

    top = new int[K];
    for(int i=0; i<K; i++){
        top[i] = -1;
    }

    freeIdx = 0;
}
```

```java
public boolean push(int x, int m) {
    if(freeIdx == -1){
        // Stack overflow
        return false;
    }

    int currIdx = freeIdx;

    // Fill Element in array
    arr[currIdx] = x;

    // Update freeIdx to next Freeidx
    freeIdx = next[freeIdx];

    // Update Next to previous element's index of mth stack
    next[currIdx] = top[m - 1];

    // Make current element's index as top of mth stack
    top[m - 1] = currIdx;

    return true;
}
```

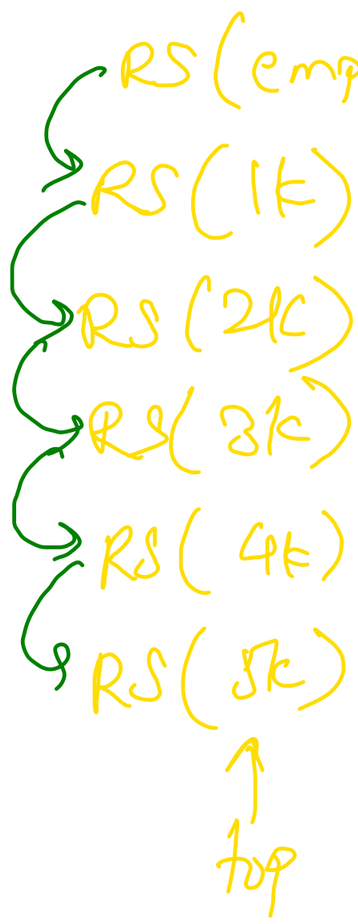
```java
public int pop(int m) {
    if(top[m - 1] == -1){
        // Stack underflow
        return -1;
    }

    int currIdx = top[m - 1];

    // Update Top of mth Stack as Previous Element's Index
    top[m - 1] = next[currIdx];

    // Update Next of CurrIdx as next Free Slot
    next[currIdx] = freeIdx;

    // Make Current Index as Free Index
    freeIdx = currIdx;

    return arr[currIdx];
}
```

Reverse Stack {using Recursion}



1u + 2u + 3u + 4u + 5u

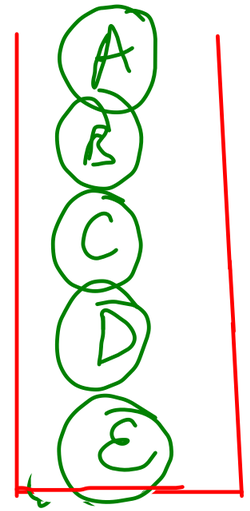
$= \frac{n*(n+1)}{2} = O(n^2)$



for Reverse Stack

(calls)^{height} + (Pre + Post) * height

$(1)^N + (1+N)*N = O(N^2)$

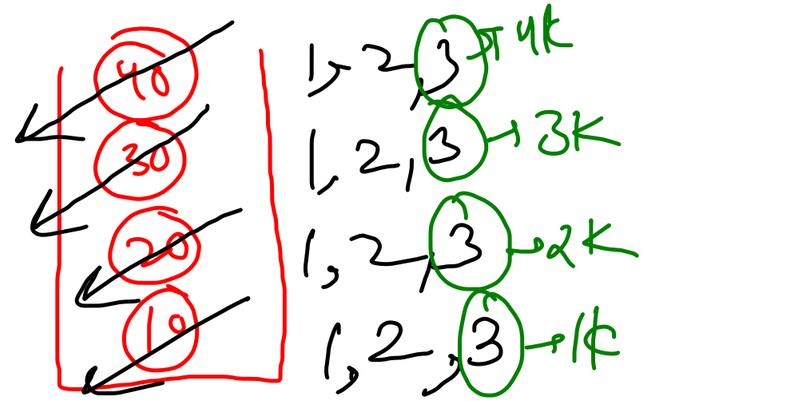


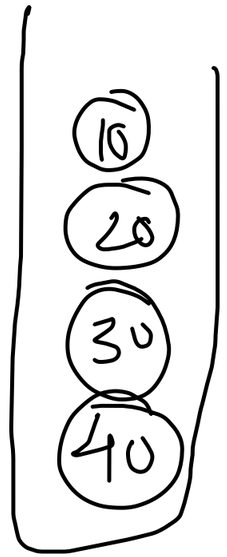
Pop() → Pre

ReverseStack() → call

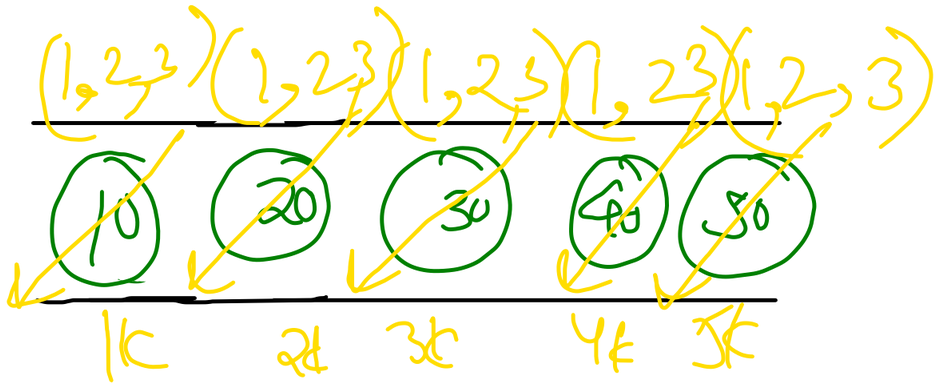
InsertAtBottom() → Post

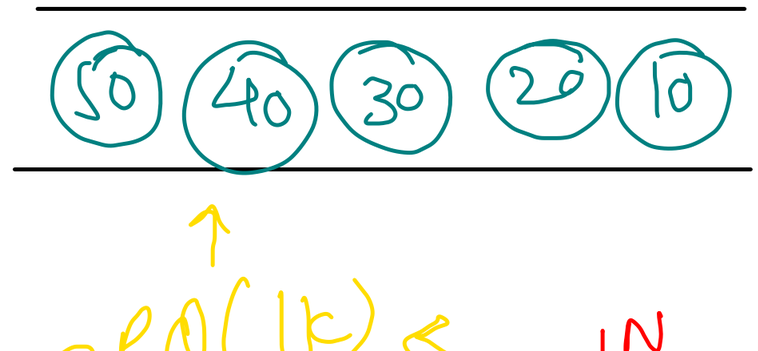
```java
public static void insertAtBottom(Stack<Integer> stack, int bottom){
    if(stack.isEmpty()){
        stack.push(bottom);
        return;
    }

    int val = stack.pop();
    insertAtBottom(stack, bottom);
    stack.push(val);
}

public static void reverseStack(Stack<Integer> stack) {
    if(stack.empty()){
        return;
    }

    int val = stack.pop();
    reverseStack(stack);
    insertAtBottom(stack, val);
}
```





Reverse Queue (using Recursion)





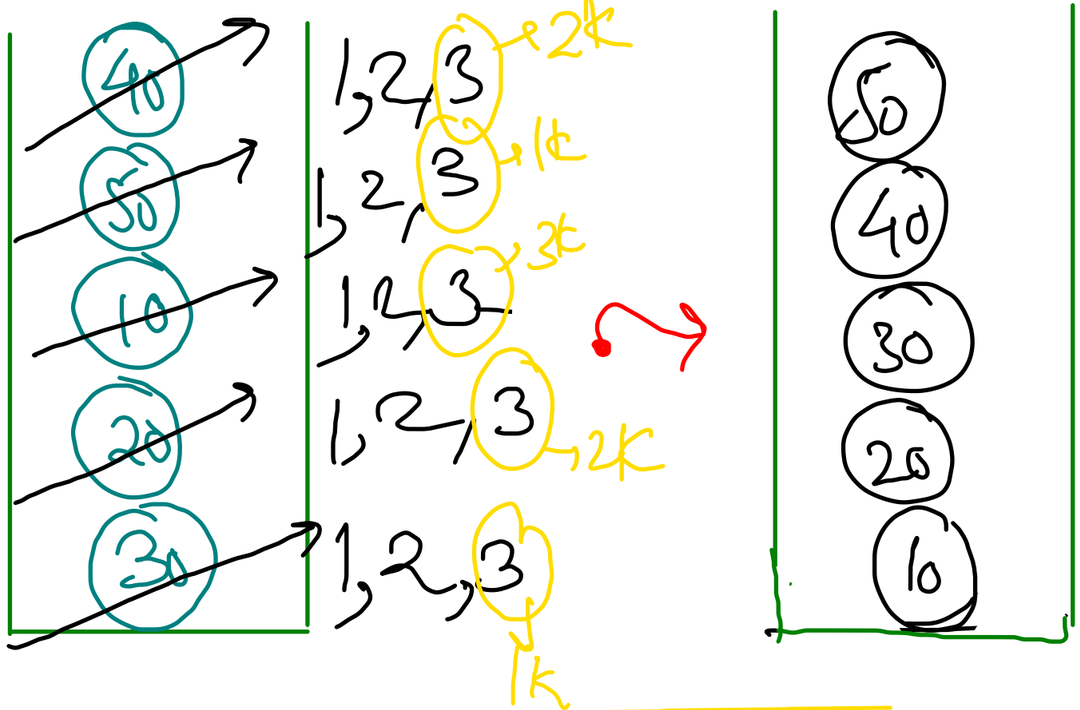

```java
public class Solution {
    public static void reverse(Queue < Integer > q) {
        if(q.size() == 0){
            return;
        }

        int val = q.remove();
        reverse(q);
        q.add(val);
    }
}
```

→ O(1) precede
→ faith
→ O(1) postorder

$1^N + (1+1)*N$

⇒ O(N)

Sort Stack {using Recursion}



worst case → N^2

best case → N

pop() → Pop

sort Stack() → call → faith

insertAtSorted()

↳ first → O(N) worst { increasing order ≈ reverse

O(1) best

{ reverse sorted }

```java
public static void insertAtSorted(Stack<Integer> stack, int val){
    if(stack.isEmpty() || stack.peek() <= val){
        stack.push(val);
        return;
    }

    int top = stack.pop();
    insertAtSorted(stack, val);
    stack.push(top);
}
```

```java
public static void sortStack(Stack<Integer> stack) {
    if(stack.isEmpty()) return;

    int top = stack.pop();
    sortStack(stack);
    insertAtSorted(stack, top);
}
```

Lecture (24) {Sunday Evening}

→ Interleave Queue

→ Valid Parantheses - 1

→ Remove Outermost Parantheses

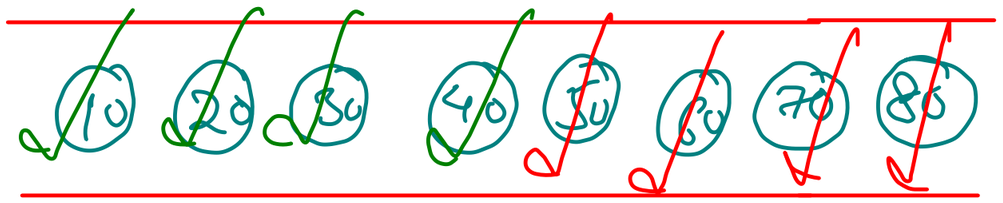
→ Longest Valid Parantheses

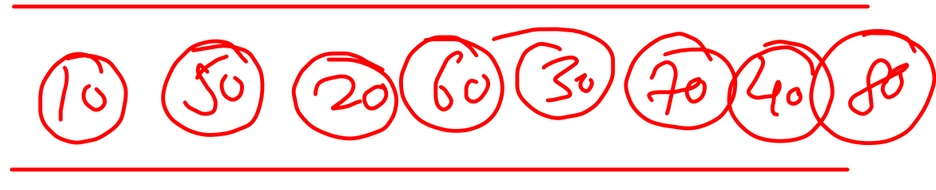
→ Minimum Additions - 1

→ Minimum Removals - 1

Parantheses Evaluation → Stack

Interleave Queue



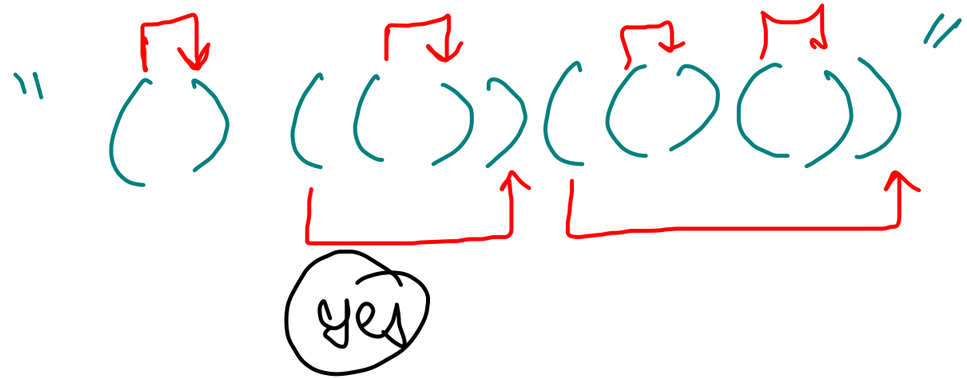


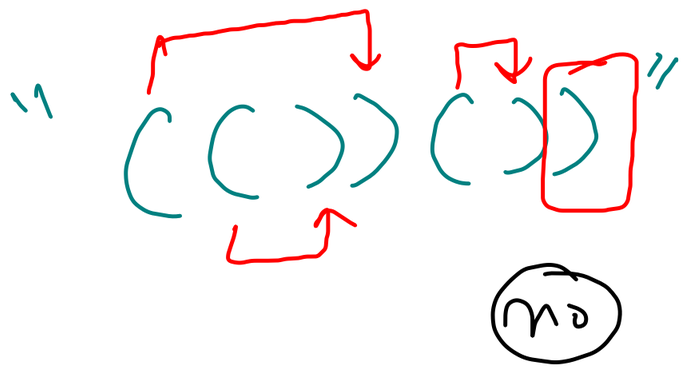
O(N) extra space

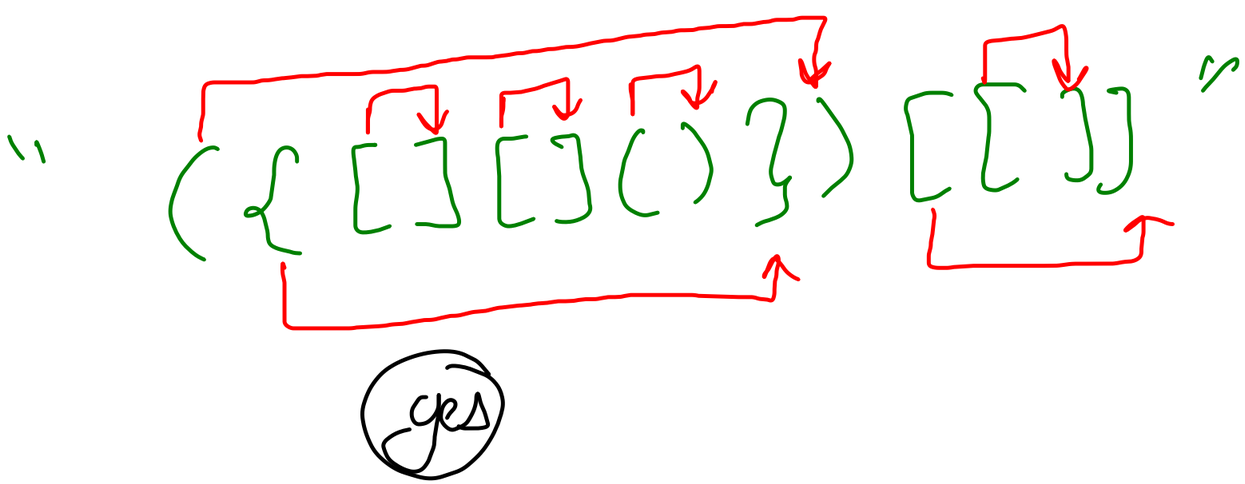
O(N) time

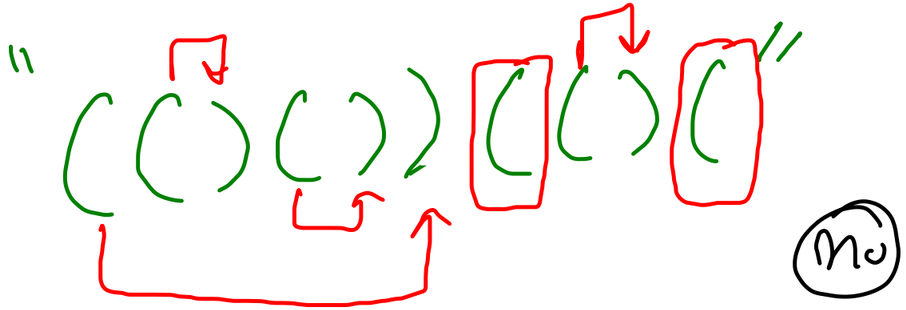
Parantheses problems

hC (20)

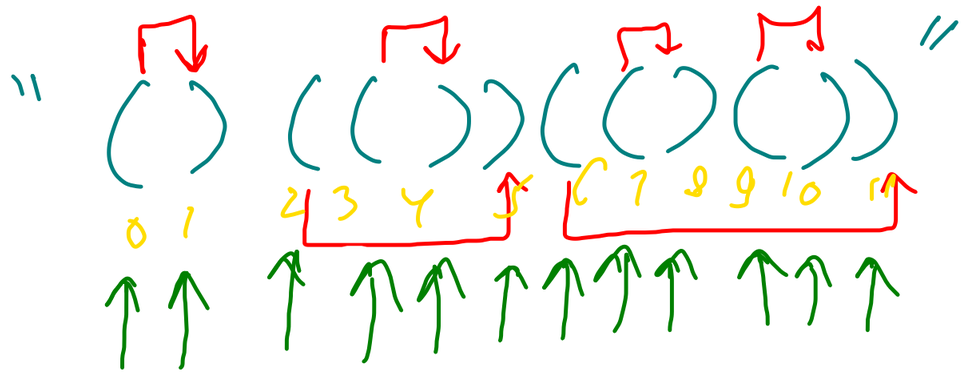


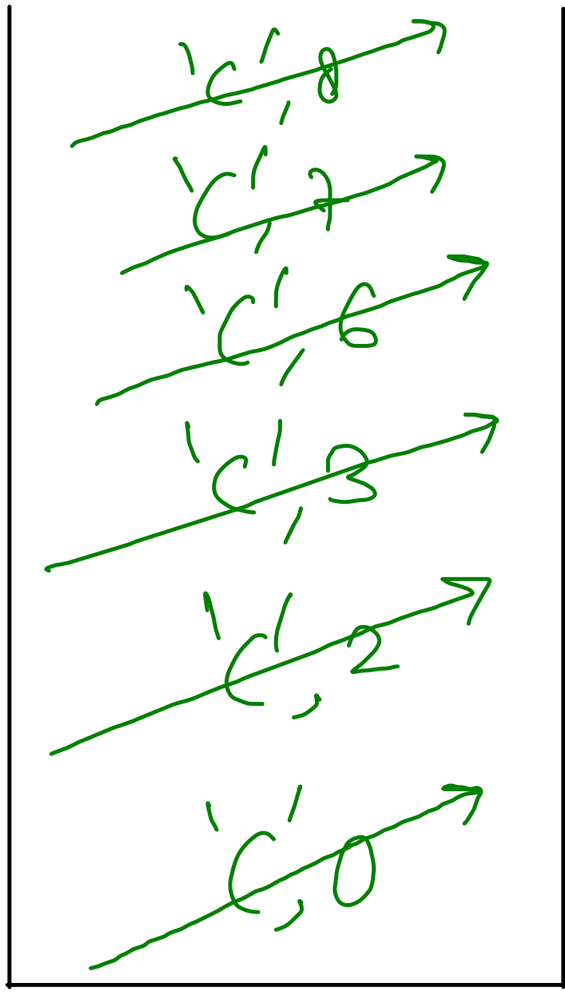


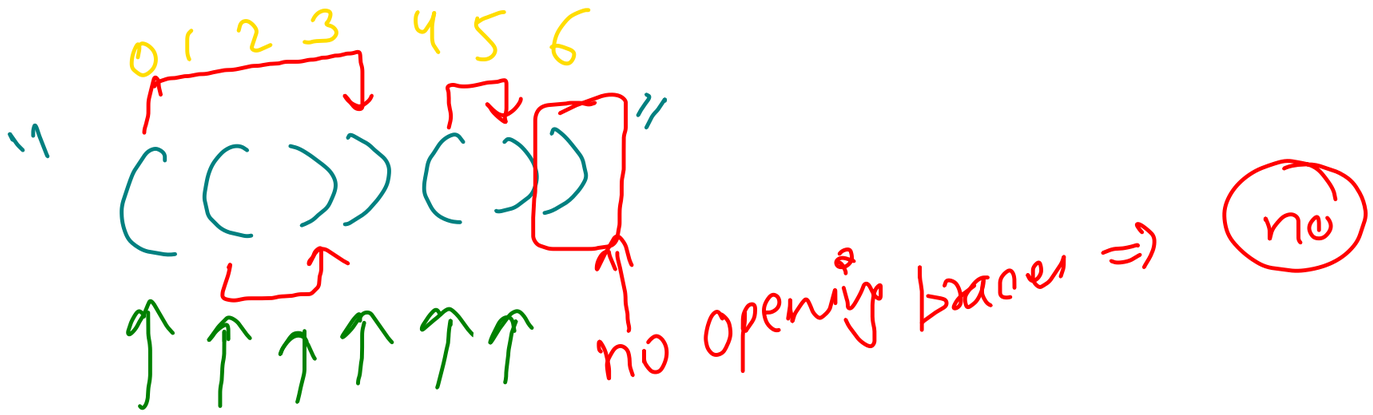


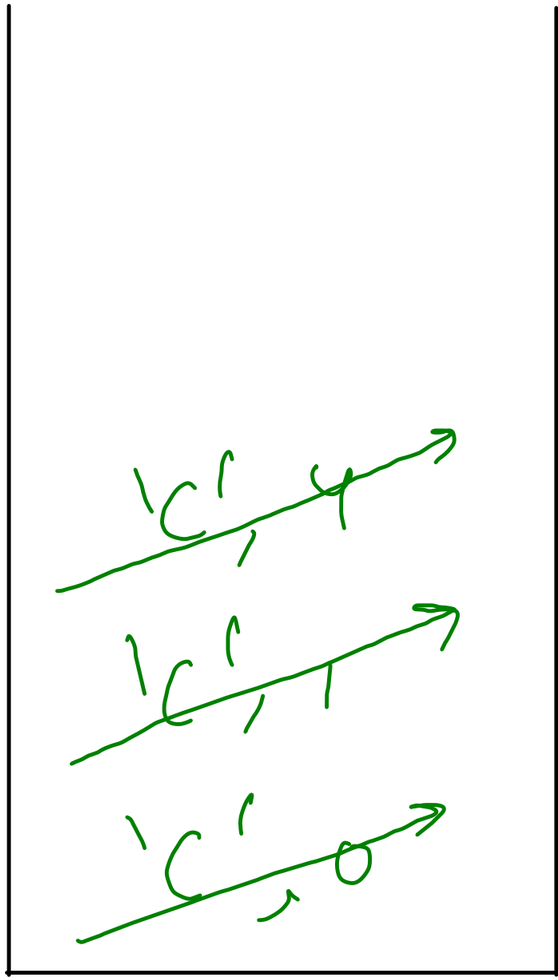


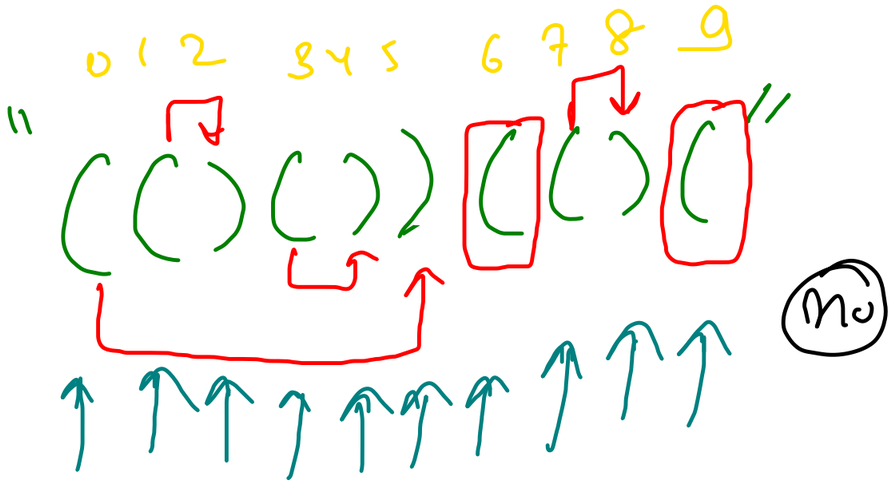
```
Stack<Integer> stk = new Stack<>();

for(int i=0; i<s.length(); i++){
    Character ch = s.charAt(i);

    if(ch == ')'){

        if(stk.isEmpty()) return false;
        if(s.charAt(stk.peek()) == '(')
            stk.pop();
        else return false;

    } else if(ch == '}'){

        if(stk.isEmpty()) return false;
        if(s.charAt(stk.peek()) == '{')
            stk.pop();
        else return false;

    } else if(ch == ']'){
        if(stk.isEmpty()) return false;
        if(s.charAt(stk.peek()) == '[')
            stk.pop();
        else return false;

    } else {
        stk.push(i);
    }
}

if(stk.size() == 0) return true;
return false;
```


0 1
2 3 4 5
6 7 8 9 10 11


8
7
6
3
2
0


0 1 2 3 4 5 6
no opening braces ⇒ no

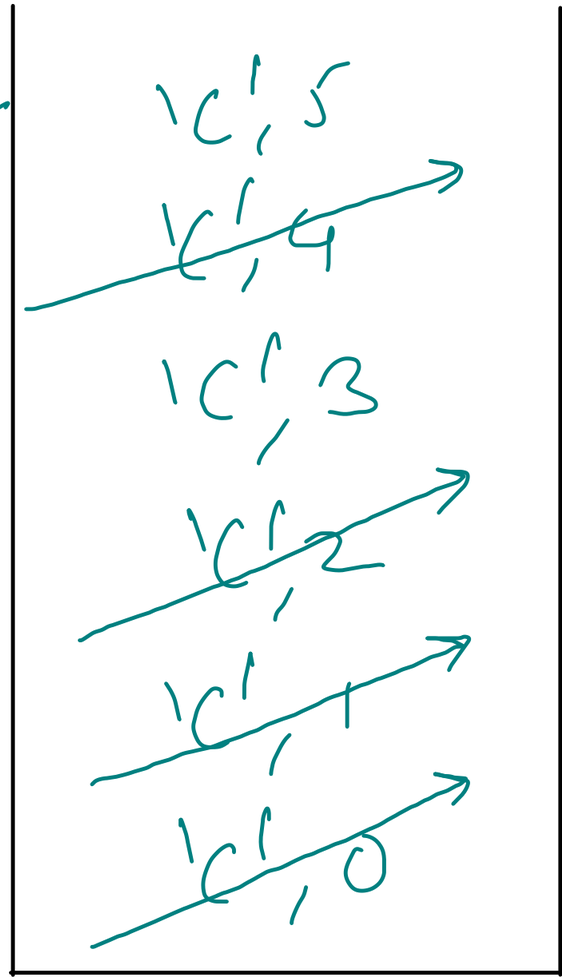


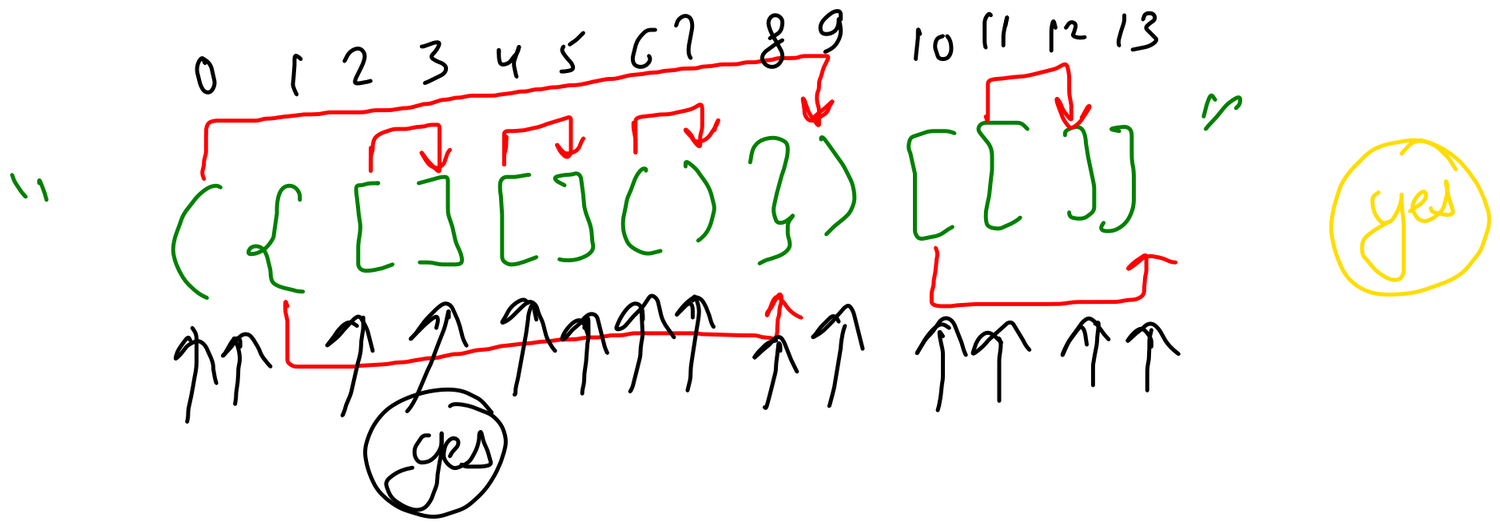


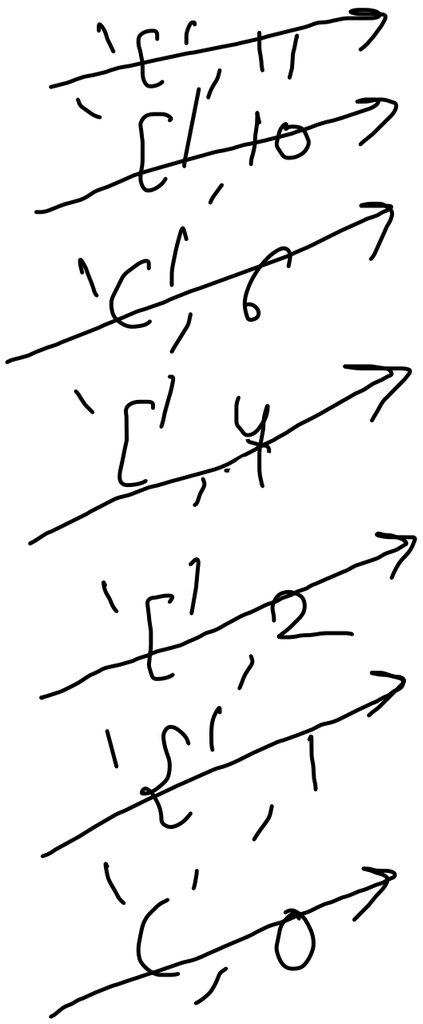
Stk.isEmpty() > 0

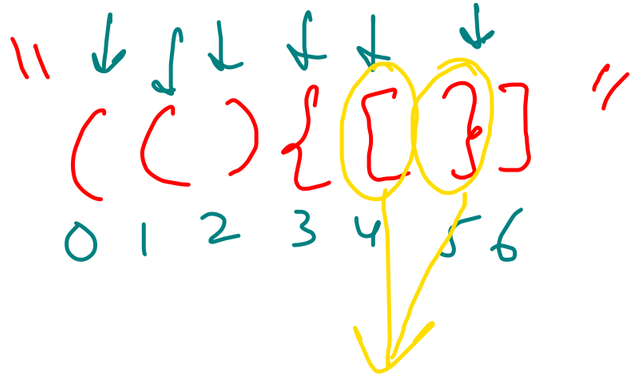
⇒ openings brackets are imbalanced

⇒ false

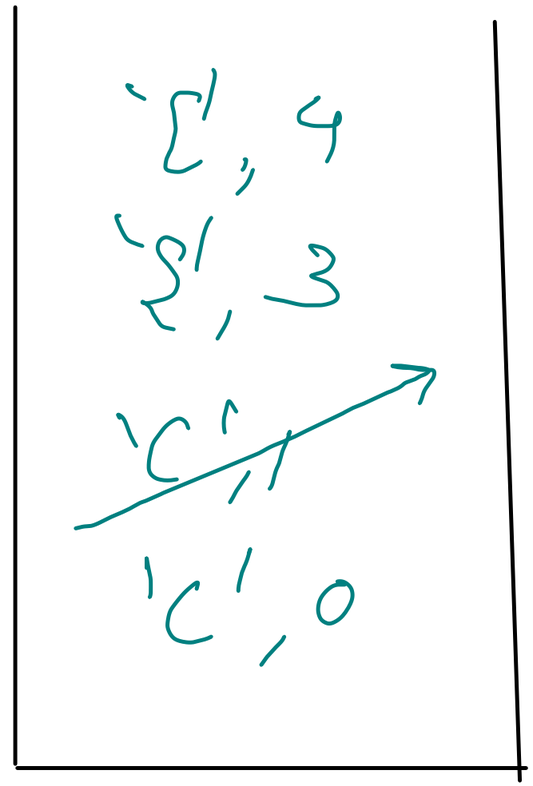



0 1 2 3 4 5 6 7 8 9 10 11 12 13
yes
yes

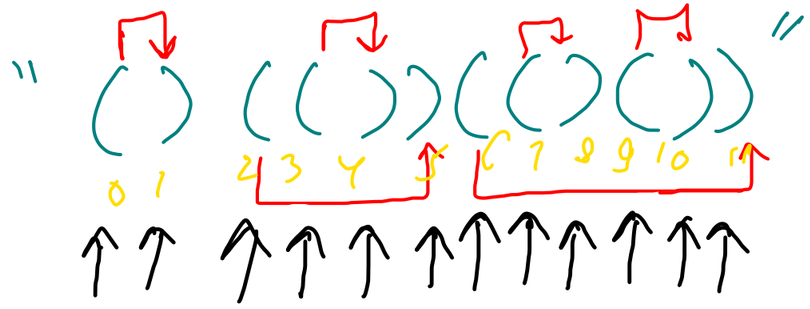




mismatching pair ⇒ no



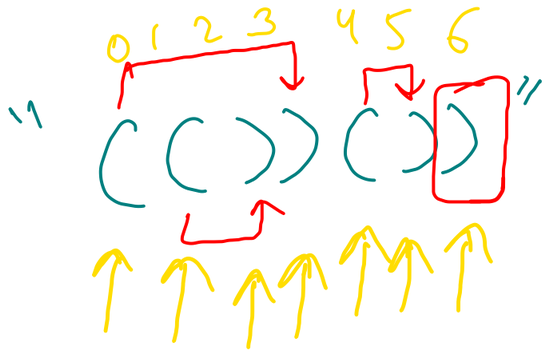
Constraint → ( ) same type



open = 0 1 0 1 2 1 6

1 2 1 2 1 0

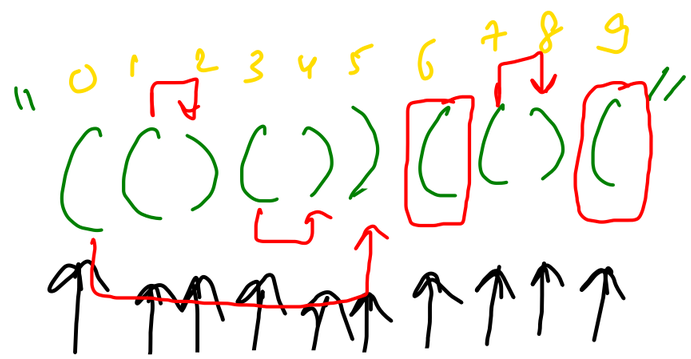
yes



open = 0 1 1 1 0

1 0 -1

no



open = 0 1 1 1 1 1 0

1 1 1 2

no

")) (("

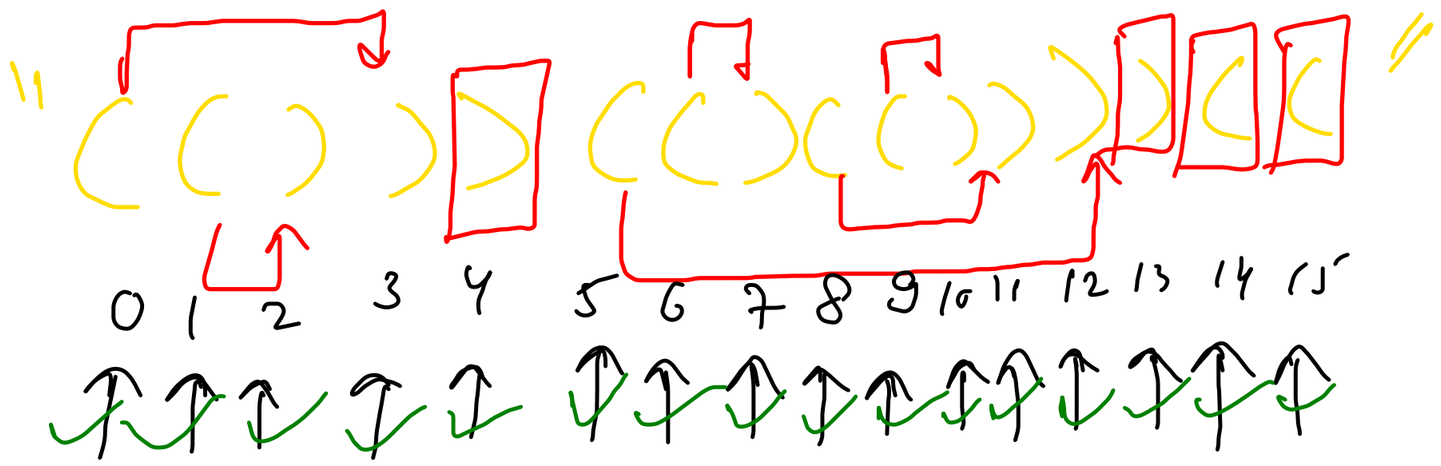
open = 0+1+2

close = 1+2

(921) LC minimum Additions/Removals

```java
public int minAddToMakeValid(String s) {
    int open = 0, close = 0;

    for(int i=0; i<s.length(); i++){
        if(s.charAt(i) == '('){
            open++;
        } else {
            if(open == 0) close++;
            else open--;
        }
    }

    return open + close;
}
```

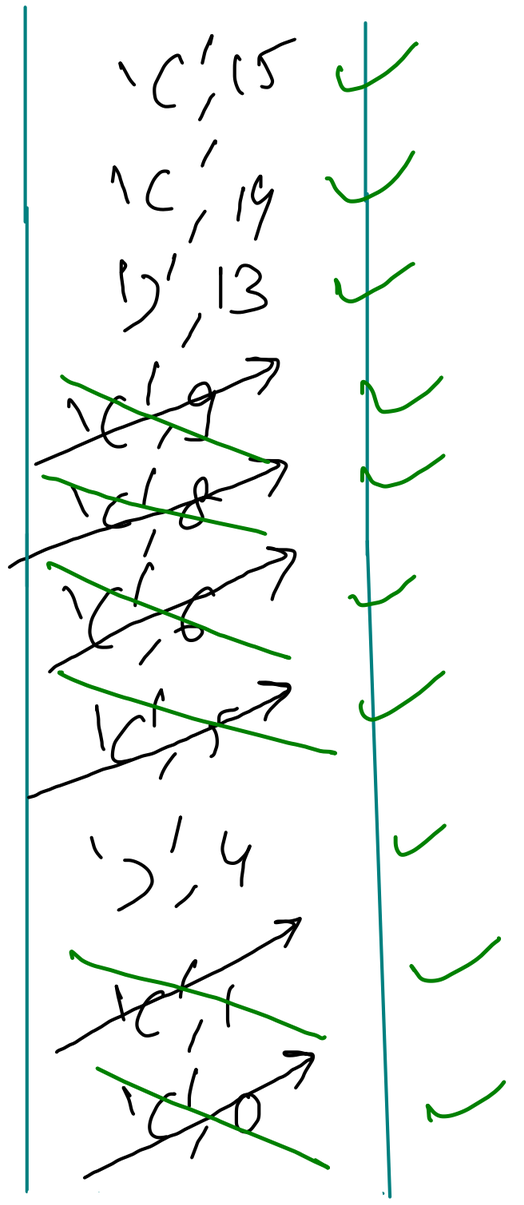
(1249) LC Minimum Removals



open → stk.push() {C imbalance}

close → stk.empty || stk.top == ')' ⇒ stk.push()
{) imbalance}

stk.pop() {balanced}



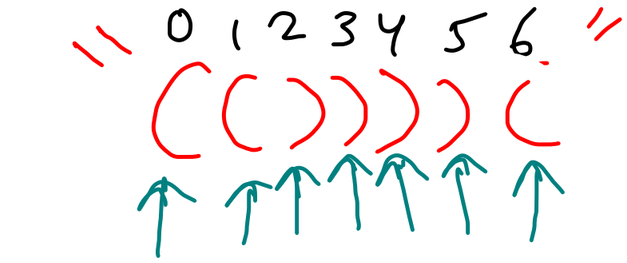
```java
Deque<Integer> q = new ArrayDeque<>();
for(int i=0; i<s.length(); i++){
    char ch = s.charAt(i);

    if(ch == '('){
        q.addLast(i);
    } else if(ch == ')'){
        if(q.size() == 0 || s.charAt(q.getLast()) == ')')
            q.addLast(i);
        else q.removeLast();
    }
}

StringBuilder res = new StringBuilder("");
for(int i=0; i<s.length(); i++){
    if(q.size() > 0 && q.getFirst() == i){
        q.removeFirst();
        continue;
    }
    res.append(s.charAt(i));
}
return res.toString();
```

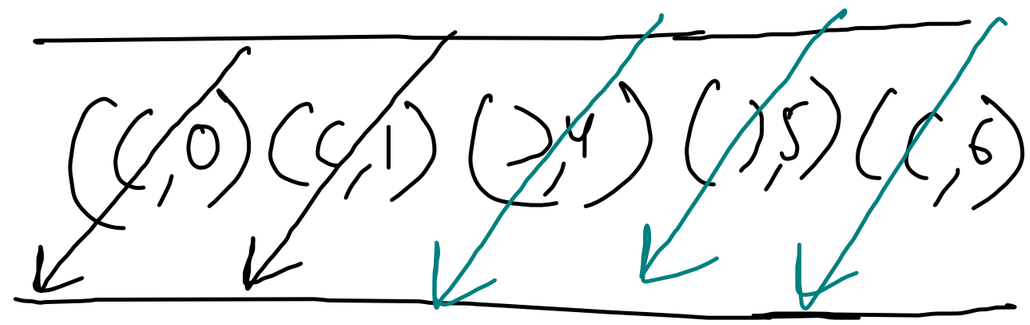
O(n) extra space

O(n)

O(n) output space

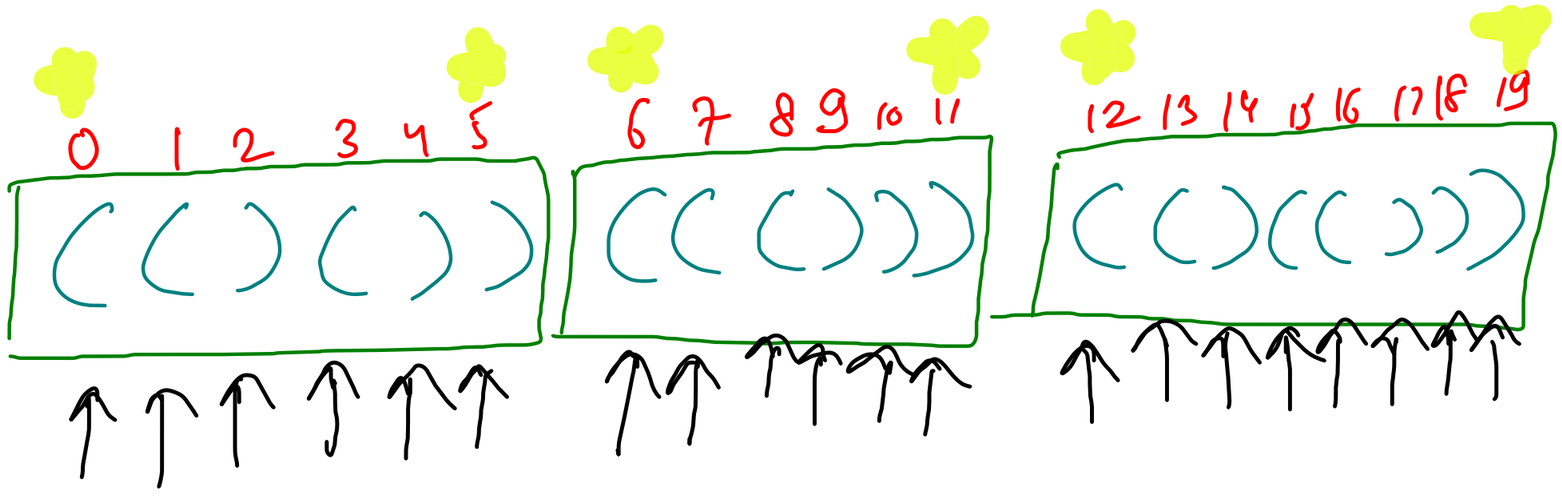
O(n)



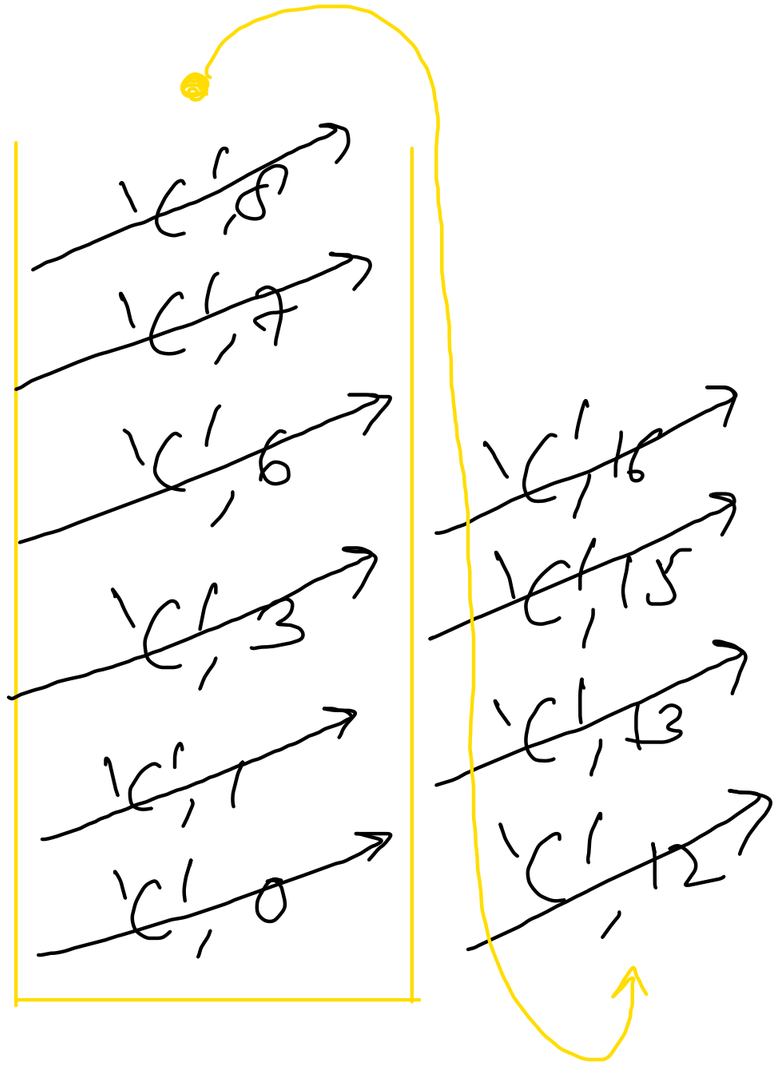


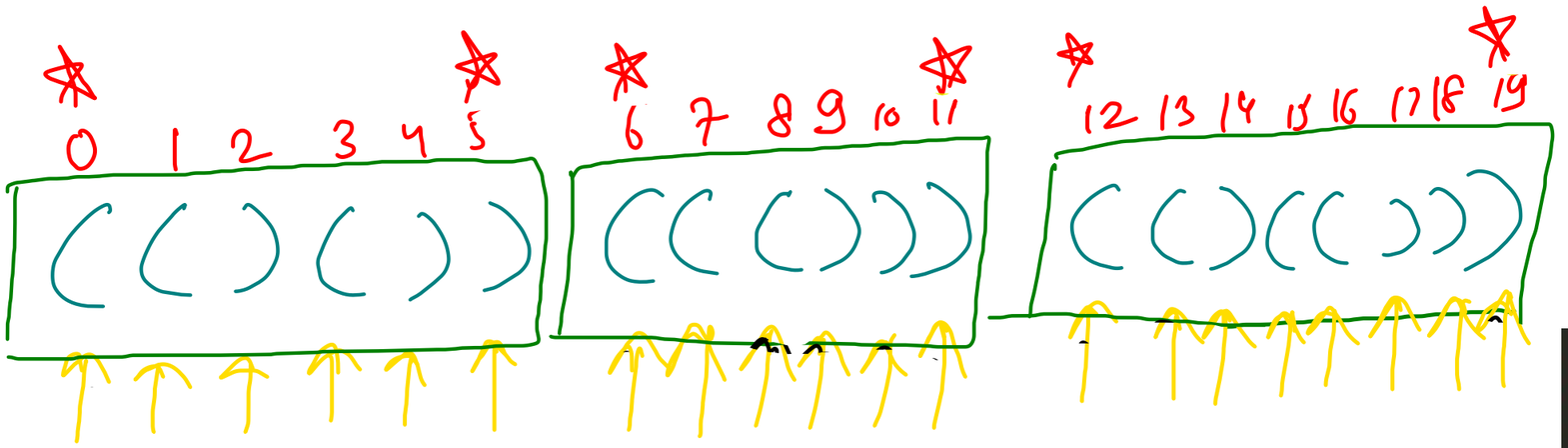
Remove Outermost Parantheses

LC 1021




```
Stack<Integer> stk = new Stack<>();

StringBuilder res = new StringBuilder("");
for(int i=0; i<s.length(); i++){
    char ch = s.charAt(i);

    if(ch == '('){
        stk.push(i);
    } else {
        int j = stk.pop();
        if(stk.empty())
            res.append(s.substring(j + 1, i));
    }
}

return res.toString();
```



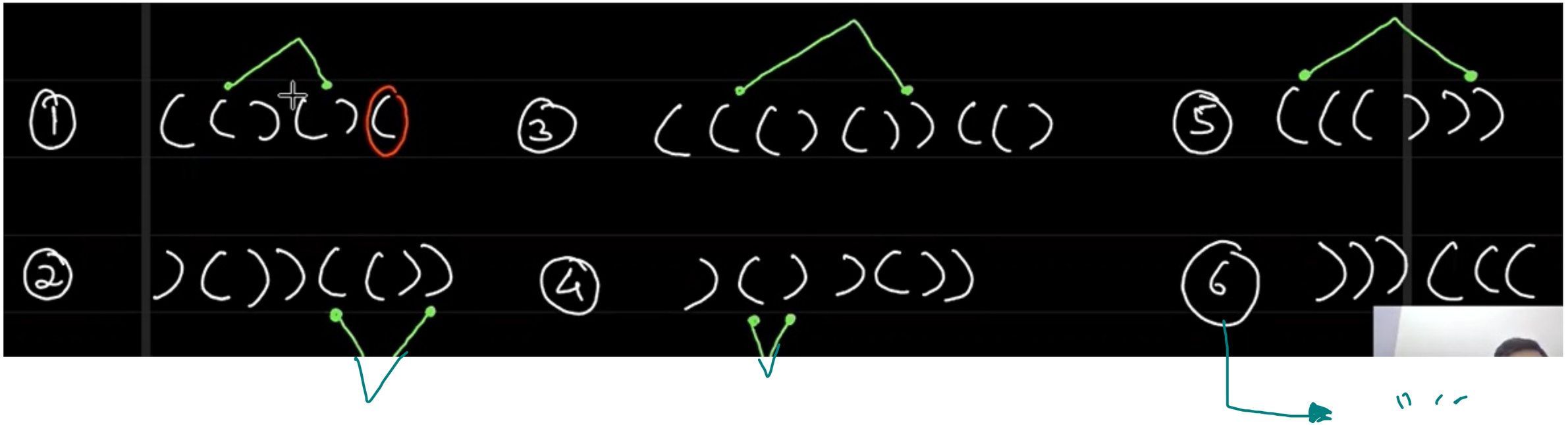


open = 0 1 2 1 0 1 2 3 2 1 0 1 2 1 2 3 2 1 0

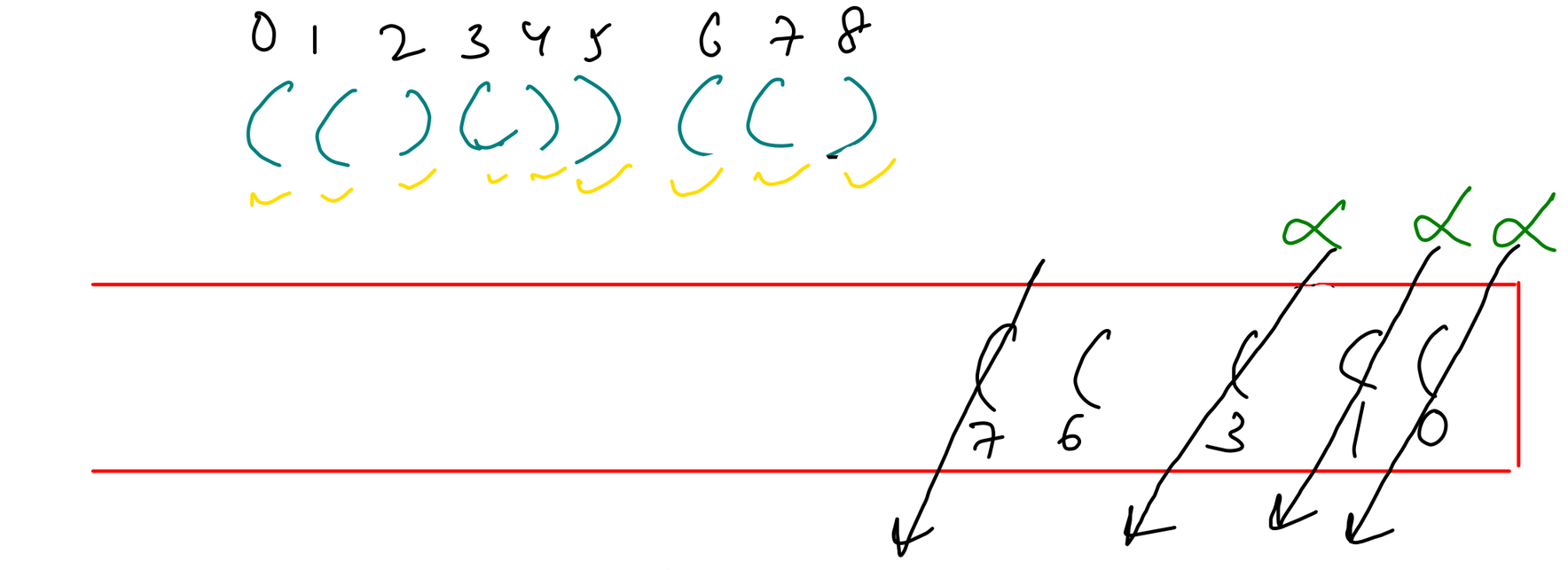
res = "( ) (( )) ( ) (( ))"

```java
int open = 0;
StringBuilder res = new StringBuilder("");

for(int i=0; i<s.length(); i++){
    char ch = s.charAt(i);

    if(ch == '('){
        if(open > 0) res.append(ch);
        open++;
    } else {
        open--;
        if(open > 0) res.append(ch);
    }
}

return res.toString();
```

(Q32) longest valid Parantheses Substring



7 (( ) ( )) ((( )))

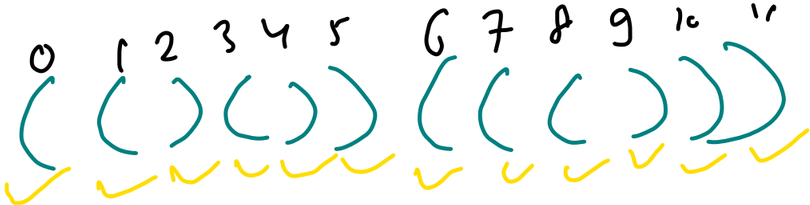


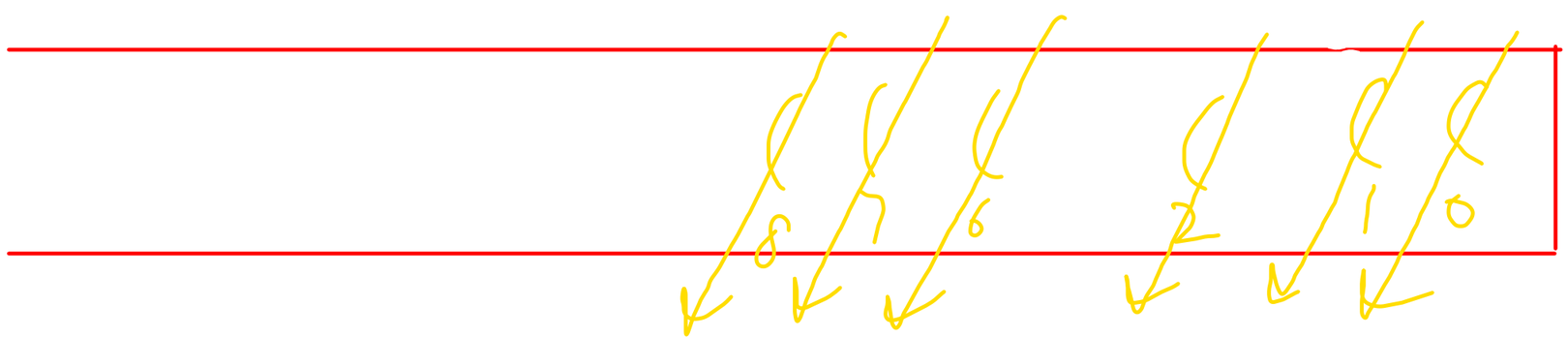
str = " "

"( )"
1 2

"( )( )"

"(( )( ))"





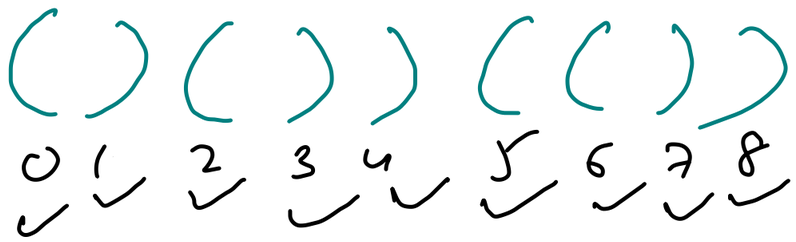
str = " "

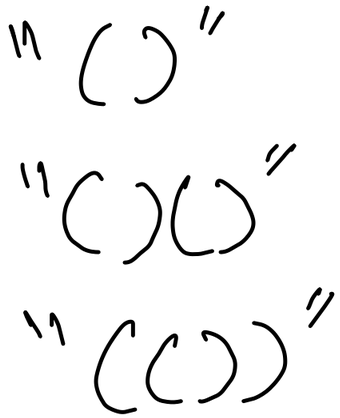
"( )"

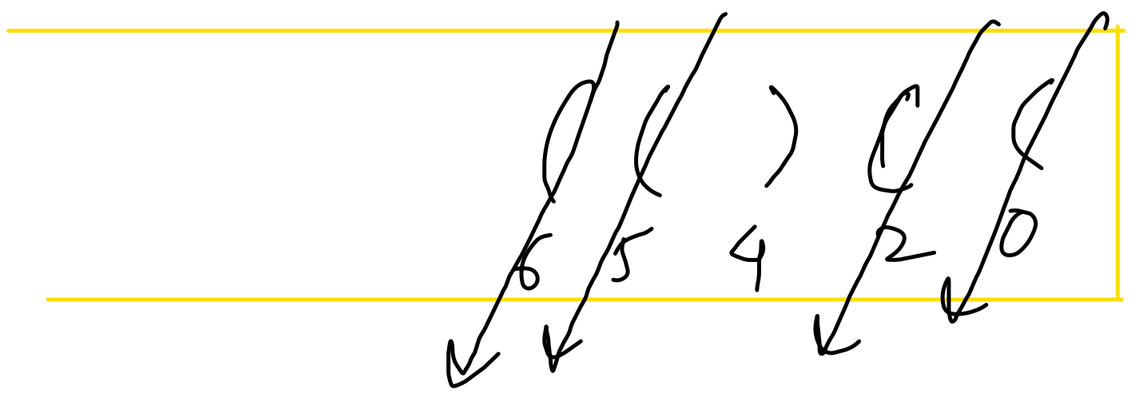
"( )( )"

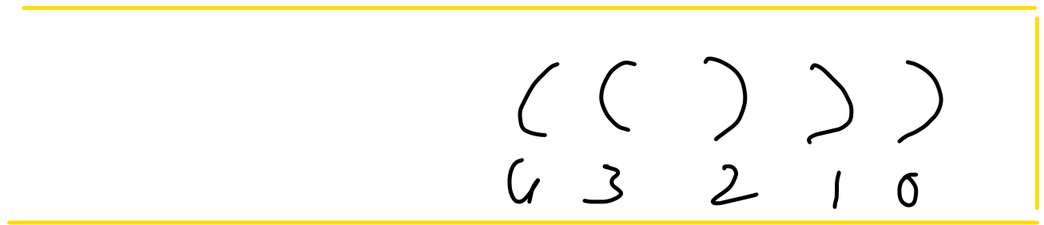
"(( )( ))"

"(( ) ( )) (( ))"









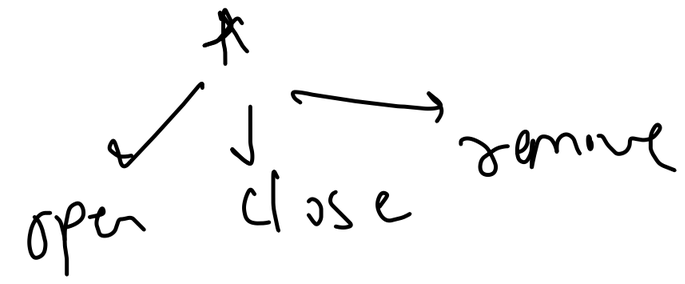

```java
public int longestValidParentheses(String s) {
    Stack<Integer> stk = new Stack<>();
    int maxLen = 0;

    for(int i=0; i<s.length(); i++){
        if(s.charAt(i) == '('){
            stk.push(i);
        } else {
            if(stk.size() > 0 && s.charAt(stk.peek()) == '(')
                stk.pop();
            else stk.push(i);
        }

        int j = (stk.size() == 0) ? -1 : stk.peek();
        maxLen = Math.max(maxLen, i - j);
    }
    return maxLen;
}
```

Stack & Queue - lecture (5)

- → Balanced Parantheses - II (678)
- → Minimum Insertions - II (1541)
- → Minimum Removals - II (301)
- → Score of Parantheses (856)
- → Reverse Substrings b/w brackets (1190)
- → Decode Strings (394)

Balanced Brackets - II

678

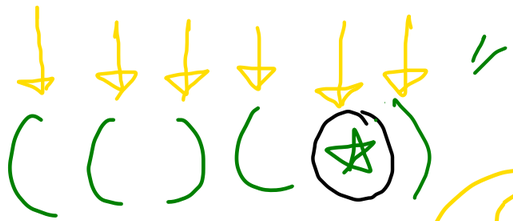


" (( ) (☆) "

" (( ) ☆ (( )) ☆) "

" (☆ ( )) "

eg 1



true

open = ϕ ≠ ≠ ≠ ≠ [1]

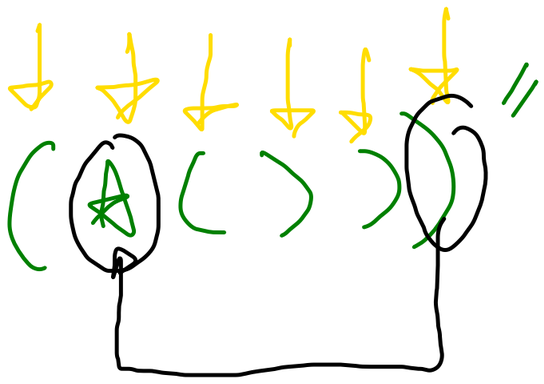
star = ϕ [1] → closing

open ≤ Star

↓

closing brace

eg 2

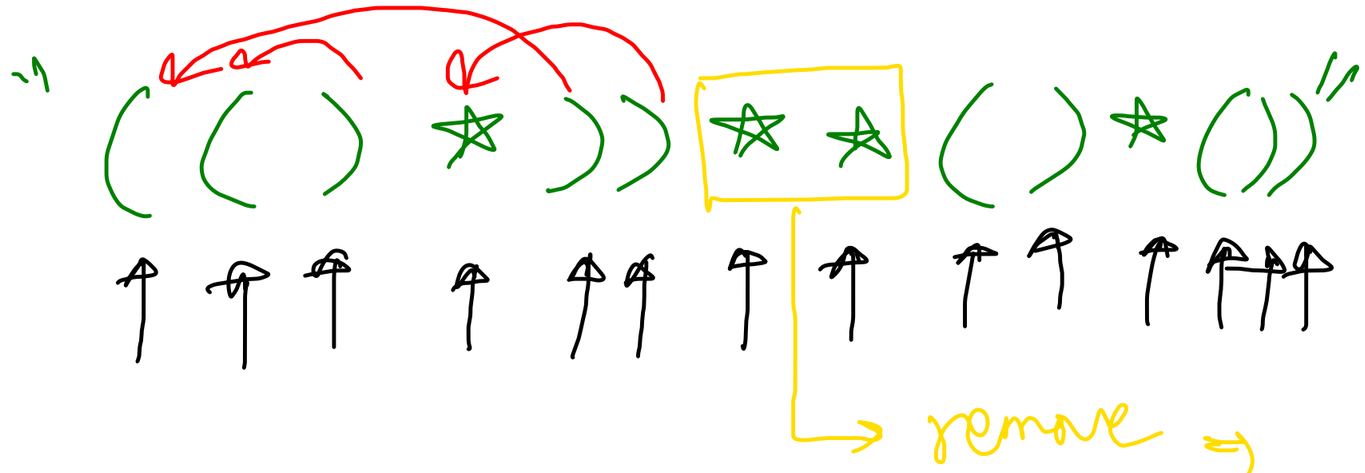


true

open = ϕ / ≠ / [0]

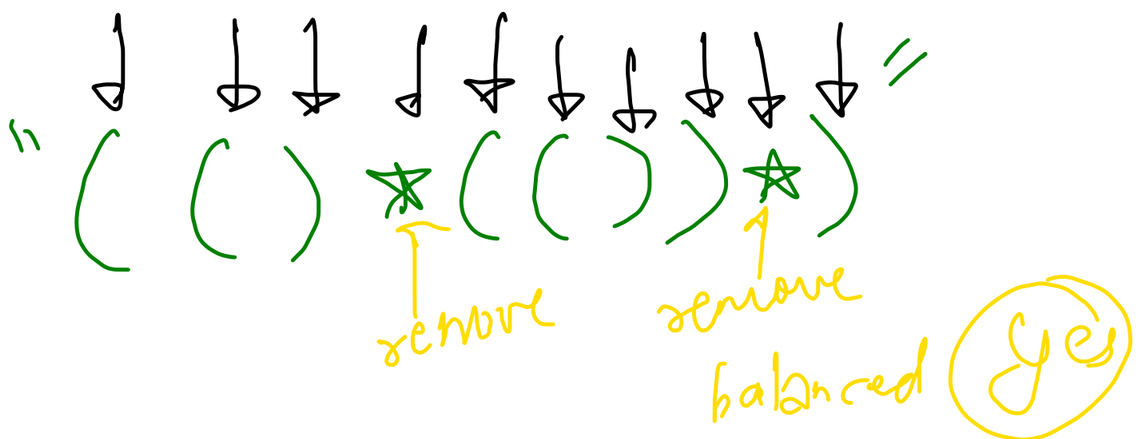
Star = / [0]

open ≤ star



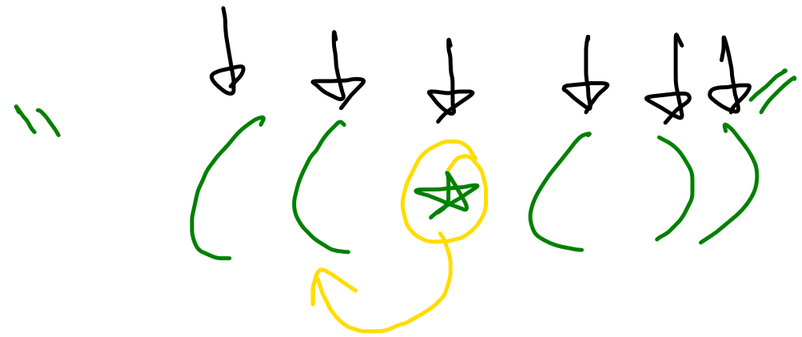
open = ∅ / (/ (/) / * /) /)

star = ∅ / * / ∅ / * / 2



open = ∅ / (/ (/) / * / (/ (/) /) /)

Star = ∅ / 2

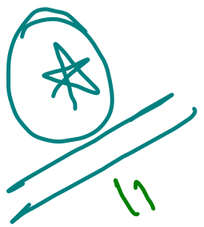


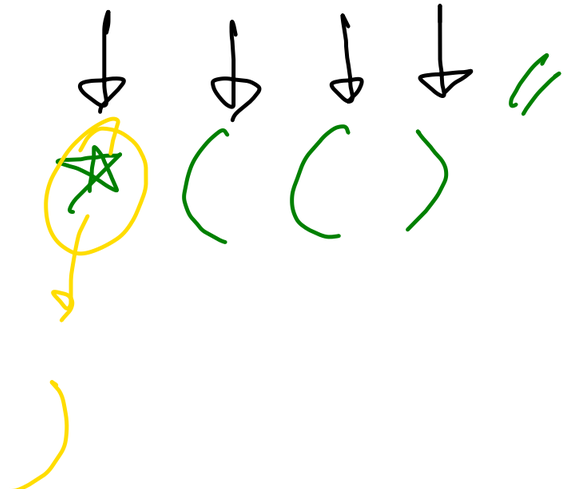
open = ∅ 1 ∅ 3 ∅ 1

star = ∅ 1

open ⊆ star

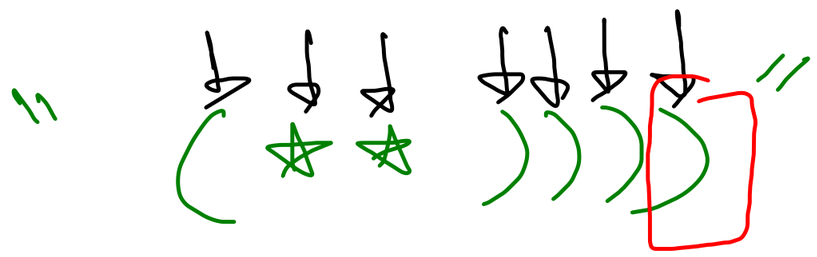
closing





open = ∅ X ∅ 1

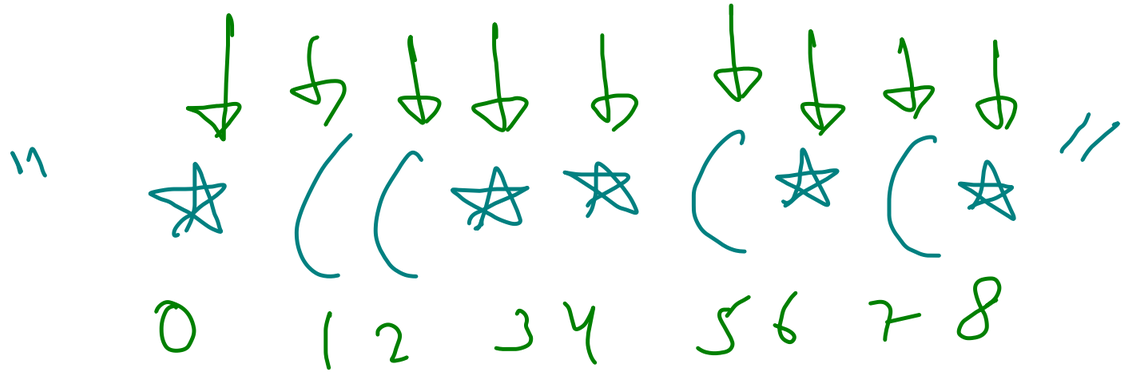
star = ∅ 1

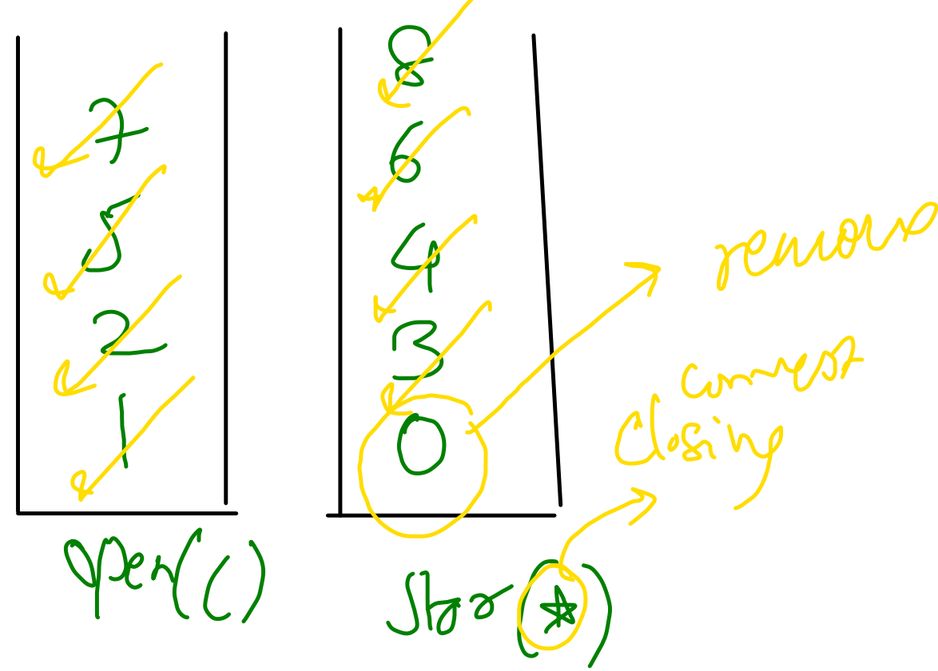


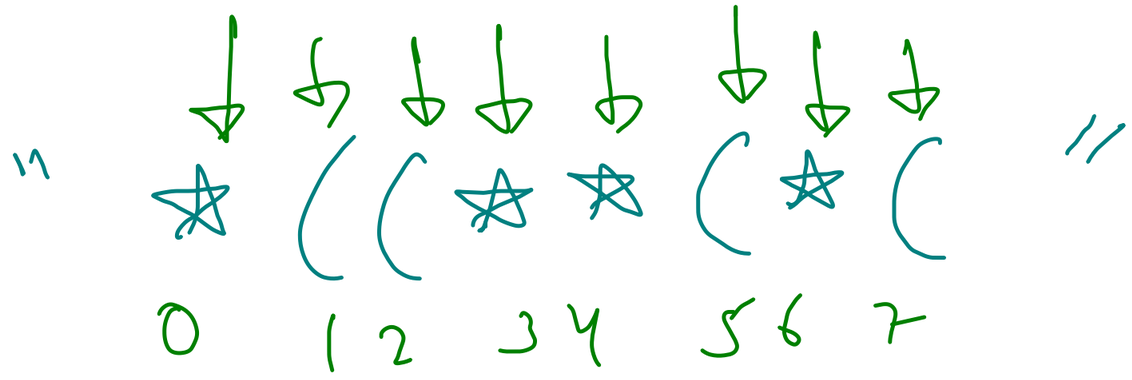
open = ∅ ✗ 0

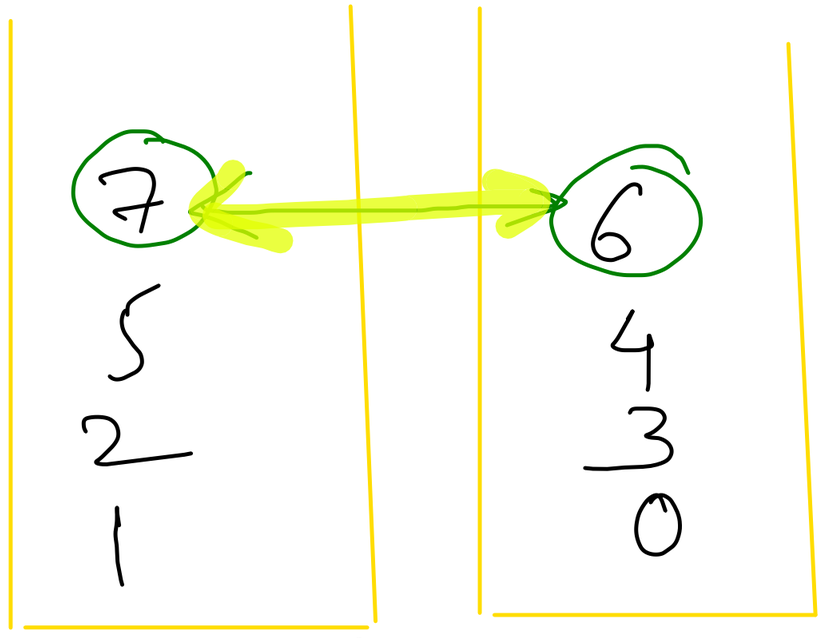
star = ∅ ✗ ☆ ✗ 0

no balanced









unbalanced

open > star ⟹ false

open = star ⟹ equivalent indices check

open < star ⟹ remove extra stars & equivalency check on remaining s

open(() star(*)

```java
public boolean checkValidString(String s) {
    Stack<Integer> open = new Stack<>();
    Stack<Integer> star = new Stack<>();

    for(int i=0; i<s.length(); i++){
        char ch = s.charAt(i);

        if(ch == '('){
            open.push(i);
        } else if(ch == '*'){
            star.push(i);
        } else {
            if(open.size() > 0) open.pop(); // opening balance with closing
            else if(star.size() > 0) star.pop(); // convert star to opening
            else return false; // unbalanced closing
        }
    }

    if(open.size() > star.size())
        return false; // opening will remain unbalance

    while(open.size() > 0){
        int openIdx = open.pop();
        int closeIdx = star.pop();

        if(openIdx > closeIdx) return false;
    }

    return true;
}
```

O(N) time
O(N) extra space
{stack}

str[i] == ')'

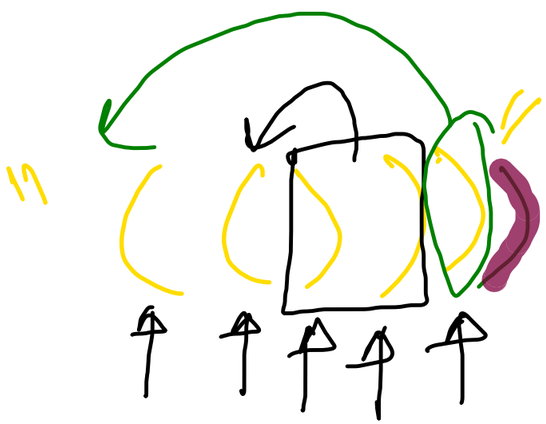
① "(((**(*" 0 1 2 3 4 5 6

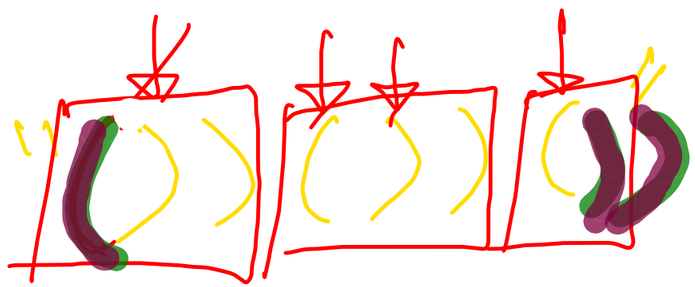
② "*((**(*(" 0 1 2 3 4 5 6 7

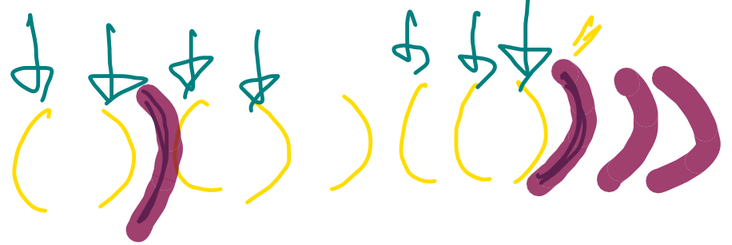
Minimum Insertions — (1)

1541

open = 2 close
(=))







open = 0

additions = 1

extra closing

open = 0 / 1

addition = 1

open = 1 / 0 / 0 / 1

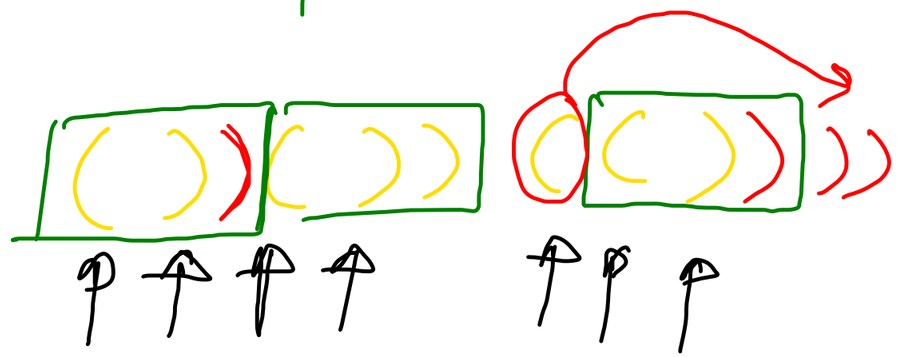
additions = 0 / 2

```java
public int minInsertions(String s) {
    int open = 0, additions = 0;

    for(int i=0; i<s.length(); i++){
        char ch = s.charAt(i);

        if(ch == '('){
            open++;
        } else if(ch == ')'){

            if(open > 0) open--;
            else additions++; // add one opening character

            if(i + 1 < s.length() && s.charAt(i + 1) == ')')
                i++;
            else additions++; // add one closing character
        }
    }

    return additions + 2 * open;
}
```

↳ add 2 closing char for each opening

O(N) Time

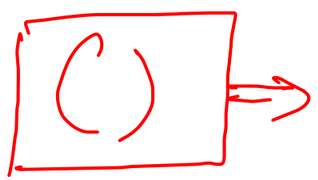
O(1) extra Space



open = ∅ / ∅ / ∅ / ∅ / 1 / 2 / 1

additions = ∅ / 2

Score of Parantheses



Score = 1

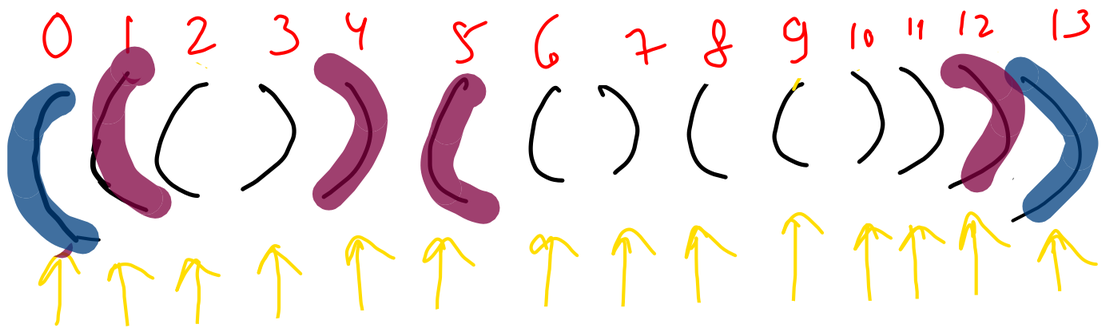
A + B → Score = A + B

(A) ⇒ Score ⇒ 2 * A

((( )) ( ) (( ))))

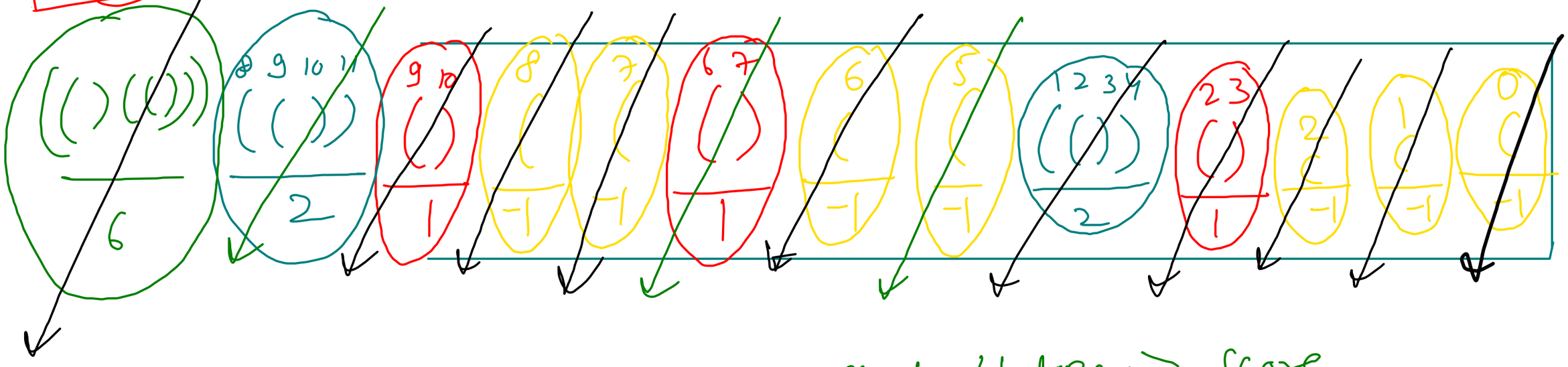
Opening index (char) → -1

Store → integer → >0



entiesting
16

var = 0+6+2



Stack <Integer> score

```java
public int scoreOfParentheses(String s) {
    Stack<Integer> stk = new Stack<>();

    for(int i=0; i<s.length(); i++){

        if(s.charAt(i) == '('){
            stk.push(-1);
        } else {

            int ans = 0;
            while(stk.peek() != -1){
                ans += stk.pop();
            }

            stk.pop(); // pop opening char (-1) as well

            if(ans == 0) stk.push(1); // push () score as 1
            else stk.push(2 * ans);
        }
    }

    int ans = 0;
    while(stk.size() > 0){
        ans += stk.pop();
    }
    return ans;
}
```

O(N) time comp

O(N) extra space {Stack}

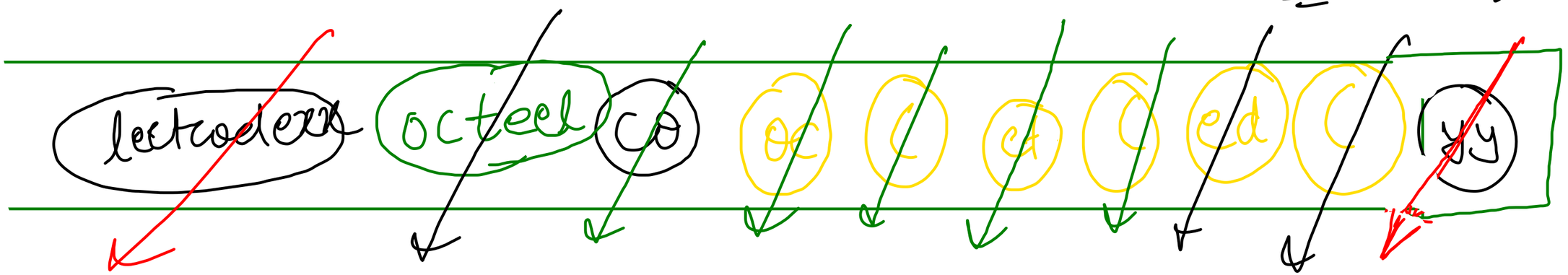
Reverse substring b/w pair of braces

9:30 - 12:30

'yy(ed(et(oc))el)xx

Str = "edfocteel"

= "leetcode"



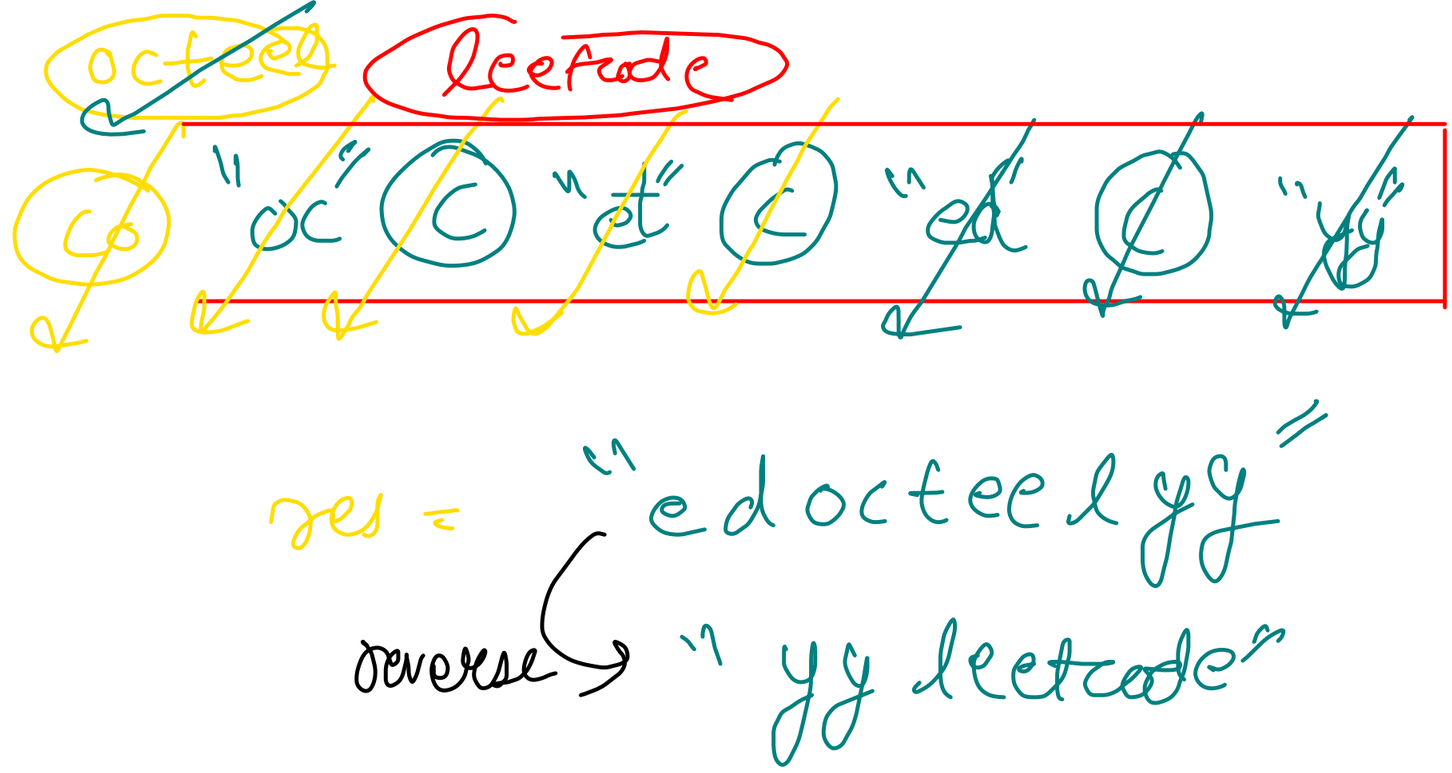
$O(N^2)$ worst case

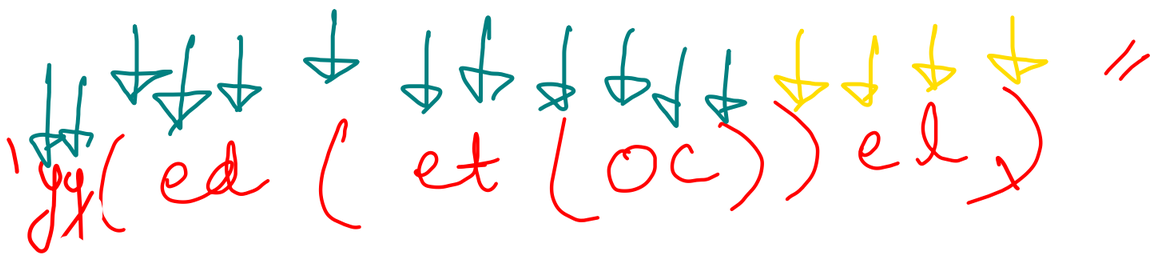
↳ traverse + reverse

yy leetcodexx

```java
Stack<StringBuilder> stk = new Stack<>();
StringBuilder open = new StringBuilder("(");

for(int i=0; i<s.length(); i++){
    char ch = s.charAt(i);

    if(ch == ')'){

        StringBuilder str = new StringBuilder("");
        while(stk.peek() != open){
            StringBuilder top = stk.pop();
            top.reverse();
            str.append(top);
        }

        stk.pop(); // pop opening
        stk.push(str);

    } else if(ch == '('){
        stk.push(open);
    } else {

        if(stk.size() == 0 || stk.peek() == open)
            stk.push(new StringBuilder(ch + ""));

        else stk.peek().append(ch);
    }
}
```

```java
StringBuilder res = new StringBuilder("");
while(stk.size() > 0){
    StringBuilder top = stk.pop();
    top.reverse();
    res.append(top);
}
return res.reverse().toString();
```





Stack & Queue - lecture (7)

→ Expression Tree ⟨ Construct, Evaluate

→ Decode String

→ Remove Invalid Parentheses

→ 132 Pattern

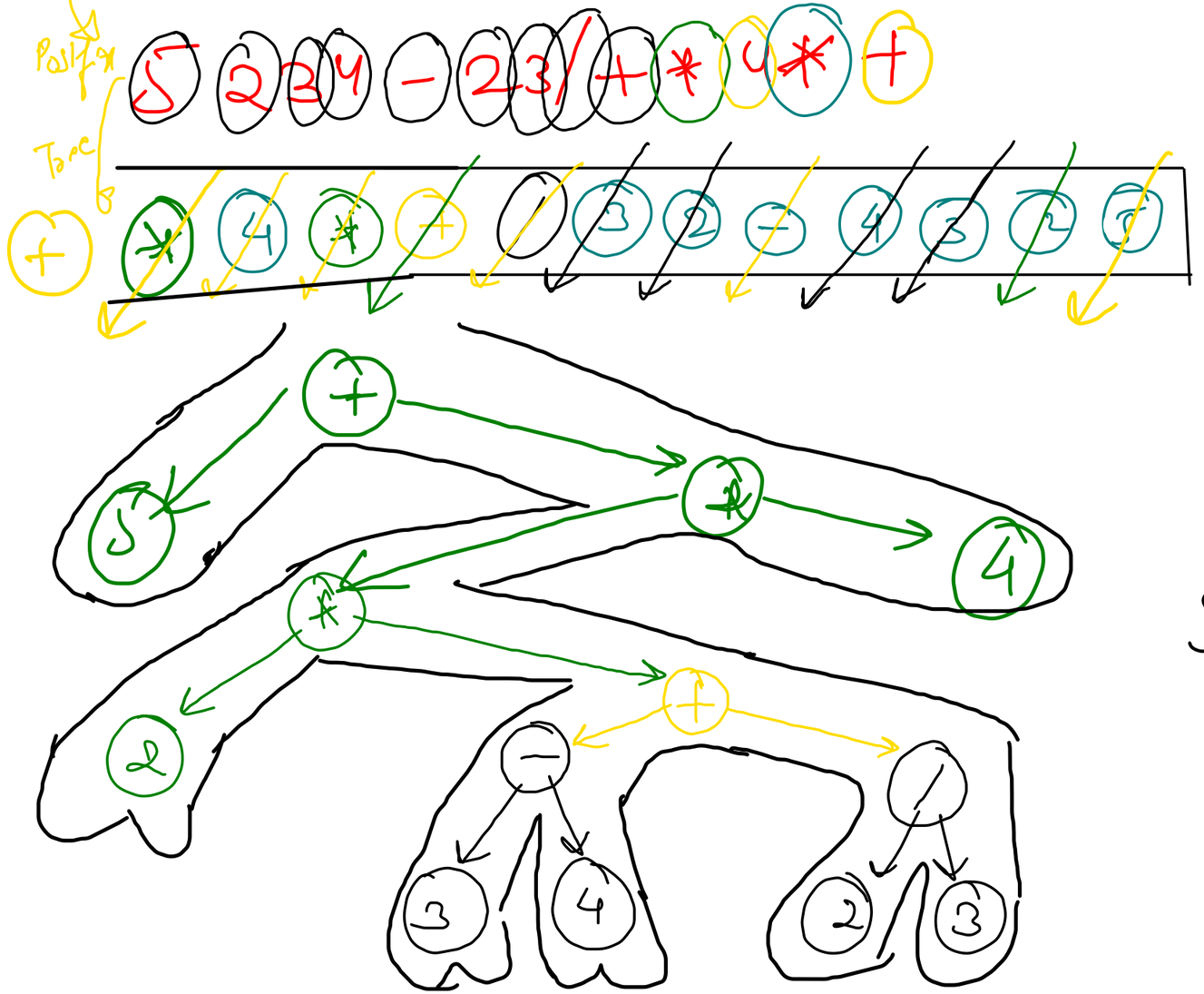
→ Sum of Subarray Minimums

Expression Tree Construction

Infix: 5 + 2 * (3 - 4 + 2 / 3) * 4

Postfix: 5 2 3 4 - 2 3 / + * 4 * +

Tree



→ Infix → Prefix, Postfix

Binary Tree

- Leaf Nodes: operands
- Non-leaf: operators

Preorder → Prefix

Postorder → Postfix

5 2 3 4 - 2 3 / + * 4 * +

```java
public static BinaryTreeNode binaryExpressionTree(String s) {
    String postfix = infixToPostfix(s);

    Stack<BinaryTreeNode> stk  = new Stack<>();
    for(int i=0; i<postfix.length(); i++){
        char ch = postfix.charAt(i);
        BinaryTreeNode node = new BinaryTreeNode(ch);

        if(ch == '+' || ch == '-' || ch == '/' || ch == '*'){
            if(stk.size() > 0){
                node.right = stk.pop();
                node.left = stk.pop();
            }
        }
        stk.push(node);
    }
    return stk.peek();
}
```

O(N)

O(N)

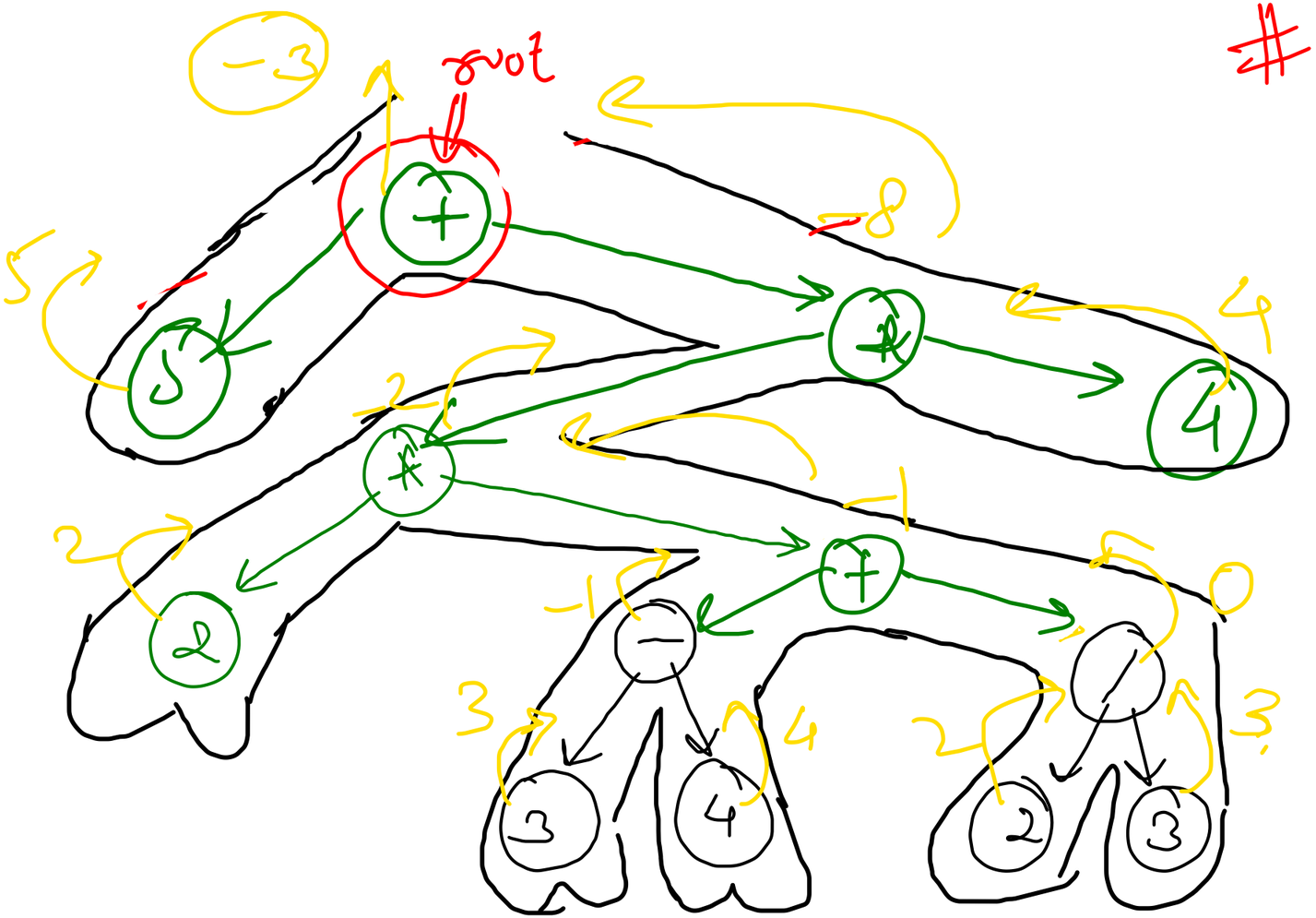
```java
public static int evaluateExpression(BinaryTreeNode<String> root)
{
    if(root.left == null && root.right == null)
        return Integer.parseInt(root.data);

    int left = evaluateExpression(root.left);
    int right = evaluateExpression(root.right);

    if(root.data.equals("+")) return left + right;
    if(root.data.equals("-")) return left - right;
    if(root.data.equals("*")) return left * right;
    return left / right;
}
```

Recursion

↓

Evaluation

Expression Tree Evaluation



Decode String

3 [a] 2 [bc]

aaabcbc

3[a2[c]]

= 3 [acc]

= accaccacc

2[abc]3[cd]ef

abcabc cdcdef

3 [a] 2 [bc]

① Count = 0*10+3 = 3

res = a+a+a

= aaa

② count = 0*10+1 = 1

= 1*10+2 = 12

res = aaabc---bc 12 times

count = 0*10 + 1 = 1*10 + 2 = 12

res (abc ...12 times)

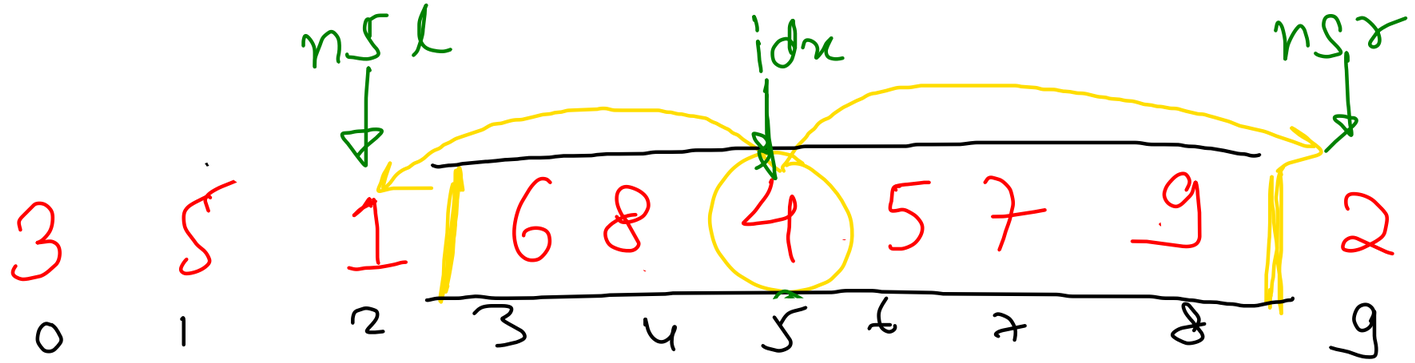
count = 0*10 + 3 = 3*10 + 4 = 34

res (abc ...12 times) (cdxx ...34 times) pp

```java
int idx = 0;
public String decodeString(String s) {
    StringBuilder res = new StringBuilder("");
    int count = 0;

    while(idx < s.length() && s.charAt(idx) != ']'){
        char ch = s.charAt(idx);
        idx++;

        if(ch >= '0' && ch <= '9'){
            count = count * 10 + (ch - '0');
        } else if(ch == '[') {

            String chotares = decodeString(s);
            for(int i=0; i<count; i++){
                res.append(chotares);
            }
            count = 0;

        } else {
            // characters from a to z
            res.append(ch);
        }
    }

    if(idx < s.length()) idx++;  // skip closing bracket
    return  res.toString();
}
```

O(n*k) time

O(n) extra space { Recursion call stack }



10^9+7 → Prime no in integer range

{ 6, 8, 4 }

{ 6, 8, 4, 5 }

{ 6, 8, 4, 5, 7 }

{ 6, 8, 4, 5, 7, 9 }

{ 8, 4 }

{ 8, 4, 5 }

{ 8, 4, 5, 7 }

{ 8, 4, 5, 7, 9 }

{ 4 }

{ 4, 5 }

{ 4, 5, 7 }

{ 4, 5, 7, 9 }

nsr - idx = 4

idx - nsl = 3

Sum of all subarrays minimum

Contribution of idx index = Value at idx * no of subarray = arr[idx] * (nsr - idx) * (idx - nsl)

```java
long[] nsl(int[] arr){
    long[] res = new long[arr.length];
    Stack<Integer> stk = new Stack<>();

    for(int i=0; i<arr.length; i++){
        while(stk.size() > 0 && arr[stk.peek()] > arr[i]){
            stk.pop();
        }

        if(stk.size() == 0) res[i] = -1;
        else res[i] = stk.peek();

        stk.push(i);
    }

    return res;
}
```

strictly smaller

```java
long[] nsr(int[] arr){
    long[] res = new long[arr.length];
    Stack<Integer> stk = new Stack<>();

    for(int i=arr.length - 1; i>=0; i--){
        while(stk.size() > 0 && arr[stk.peek()] >= arr[i]){
            stk.pop();
        }

        if(stk.size() == 0) res[i] = arr.length;
        else res[i] = stk.peek();

        stk.push(i);
    }

    return res;
}
```

smaller & equal

```java
public int sumSubarrayMins(int[] arr) {
    long[] nsl = nsl(arr);
    long[] nsr = nsr(arr);

    long res = 0;
    for(int i=0; i<arr.length; i++){
        res = res + (arr[i] * (nsr[i] - i) * (i - nsl[i]));
    }
    return (int)(res % 10000000007l);
}
```

→ 1e9+7

→ O(N) time

→ O(N) space extra of nsl, nsr, stack

{0, 1, 3, 1, 4, 2, 2, 1}

Subarray's min

[3] → 3

[3,1] → 1

[3,1,4] → 1

[3,1,4,2] → 1

✓ [3,1,4,2,1] → 1

✓ [3,1,4,2,1,2] → 1

[1] → 1

[1,4] → 1

[1,4,2] → 1

✓ [1,4,2,1] → 1

✓ [1,4,2,1,2] → 1

Contributings

Contributn of 1

Contributn of 4

Contributn of 2

ans = 3*1 + 1*6 + 4*1 + 2*3

[4] → 4

[4,2] → 2

[4,2,1] → 1

[4,2,1,2] → 1

[2] → 2

[2,1] → 1

[2,1,2] → 1

[1] → 1

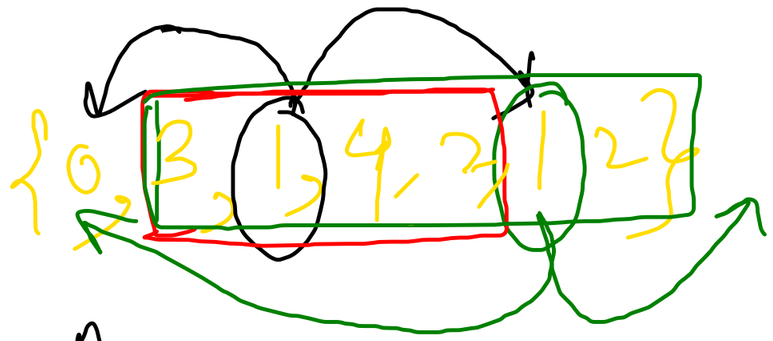
[1,2] → 1

[2] → 2

→ Next smaller to left (strictly)

→ Next Smaller to right (smaller and=)

mutually exclusive & exhaustive



intersection = ϕ

union = total

1st one is min

(3, 1)

(3, 1, 4)

(3, 1, 4, 2)

(1)

(1, 4)

(1, 4, 2)

2nd one is min

(3, 1, 4, 2, 1)

(3, 1, 4, 2, 1)

(1, 4, 2, 1)

(1, 4, 2, 1, 2)

(4, 2, 1)

(4, 2, 1, 2)

(2, 1)

(2, 1, 2)

(1)

(1, 2)

Stack & Queue - lecture (8)

→ Trapping Rain Water - I

→ Trapping Rain Water - II

→ Container with most water

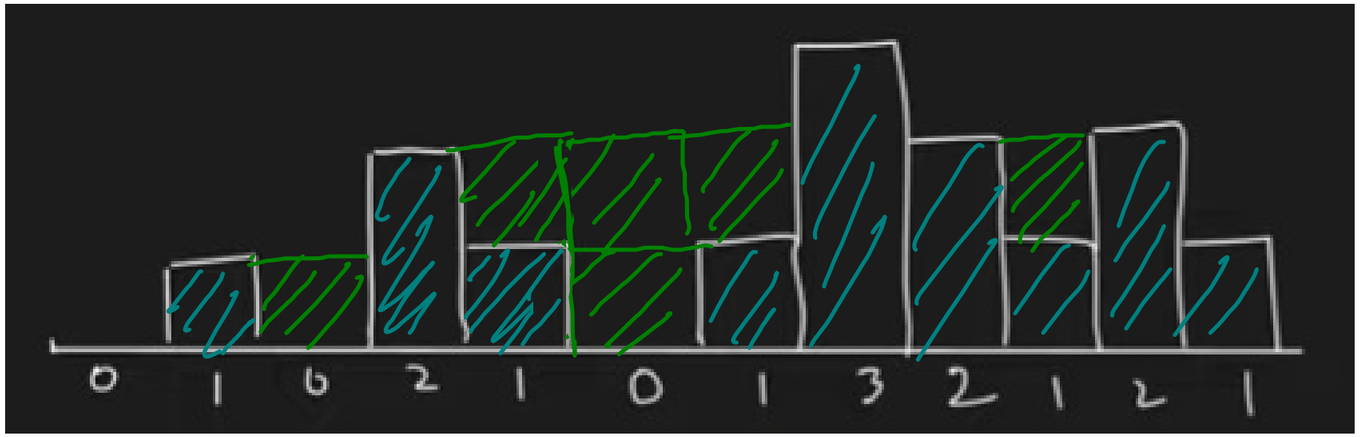
→ Remove Duplicate letters

→ Remove Invalid Parantheses

→ 132 Pattern

Trapping Rain Water

42 Leetcode



prefix → leftmax

suffix → rightmax

ans = min(lmax, rmax) — my height

lmax 0 1 1 2 2 2 2 3 3 3 3 3

↕ O(N) extra space

rmax 3 3 3 3 3 3 3 3 2 2 2 1

water 0 0 1 0 1 2 1 0 0 1 0 0

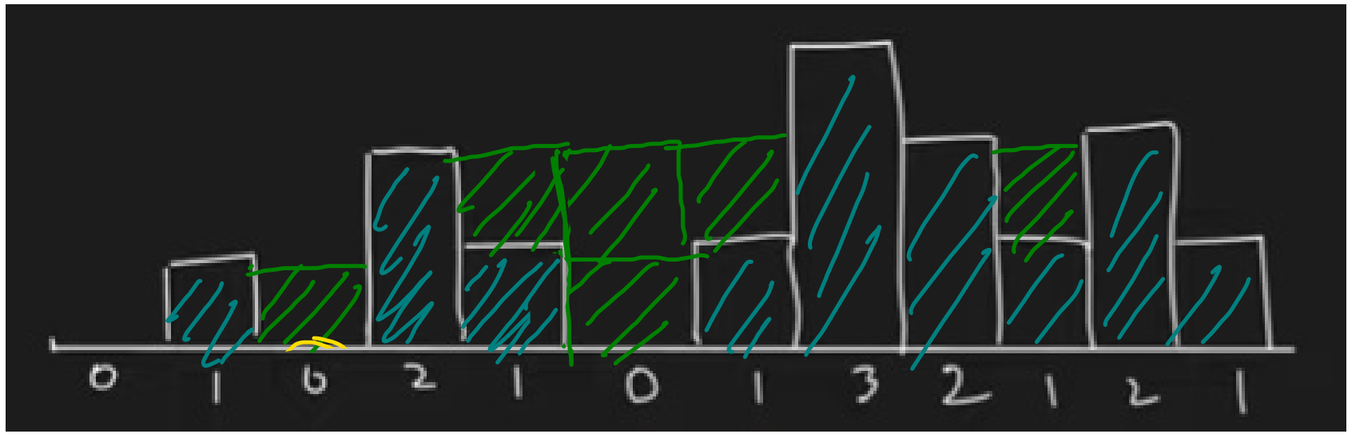
```java
int[] leftMax(int[] height){
    int[] res = new int[height.length];
    res[0] = height[0];

    for(int i=1; i<height.length; i++){
        res[i] = Math.max(height[i], res[i - 1]);
    }
    return res;
}

int[] rightMax(int[] height){
    int[] res = new int[height.length];
    res[height.length - 1] = height[height.length - 1];

    for(int i=height.length-2; i>=0; i--){
        res[i] = Math.max(height[i], res[i + 1]);
    }
    return res;
}
```

O(N) Time Comp

O(N) space comp

Prefix & Suffix Array

```java
public int trap(int[] height) {
    int[] leftMax = leftMax(height);
    int[] rightMax = rightMax(height);

    int water = 0;
    for(int i=0; i<height.length; i++){
        water += (Math.min(leftMax[i], rightMax[i]) - height[i]);
    }
    return water;
}
```



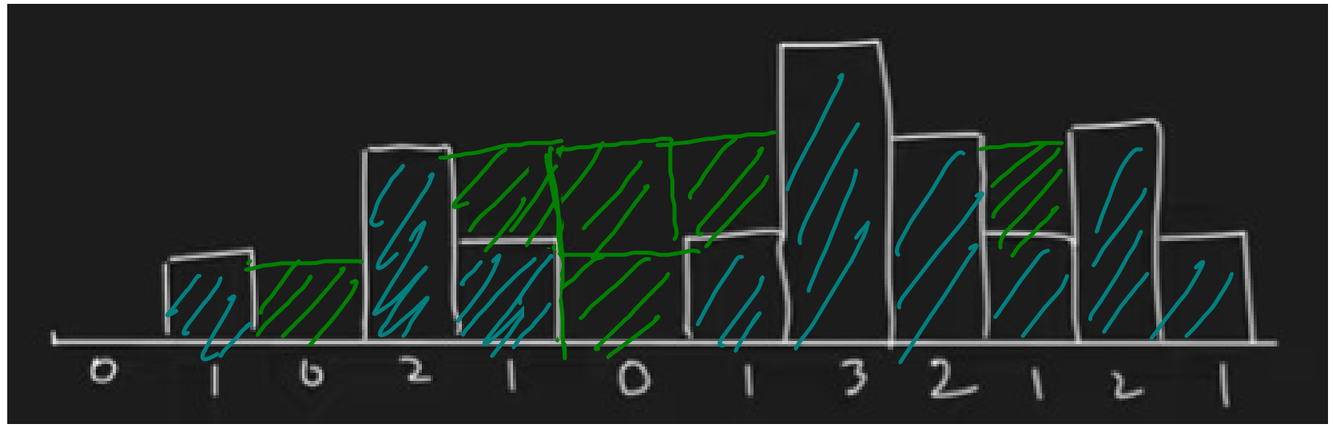
leftmax = 0 1 2 3

rightmax = 0 1 2

water = 0 1 +1 +2 +1 +1

Pseudo code

```
left = 0 , right = arr.length-1
while( left <= right){
    if( a[l] <= a[r]){
        → lm - a[l]
        → lm = a[l]
        left++;
    }
    else{
        → rm - a[r]
        → rm = a[r]
        right--;
    }
}
```



TWO POINTER

→ O(N) time

→ O(1) extra space

11:20

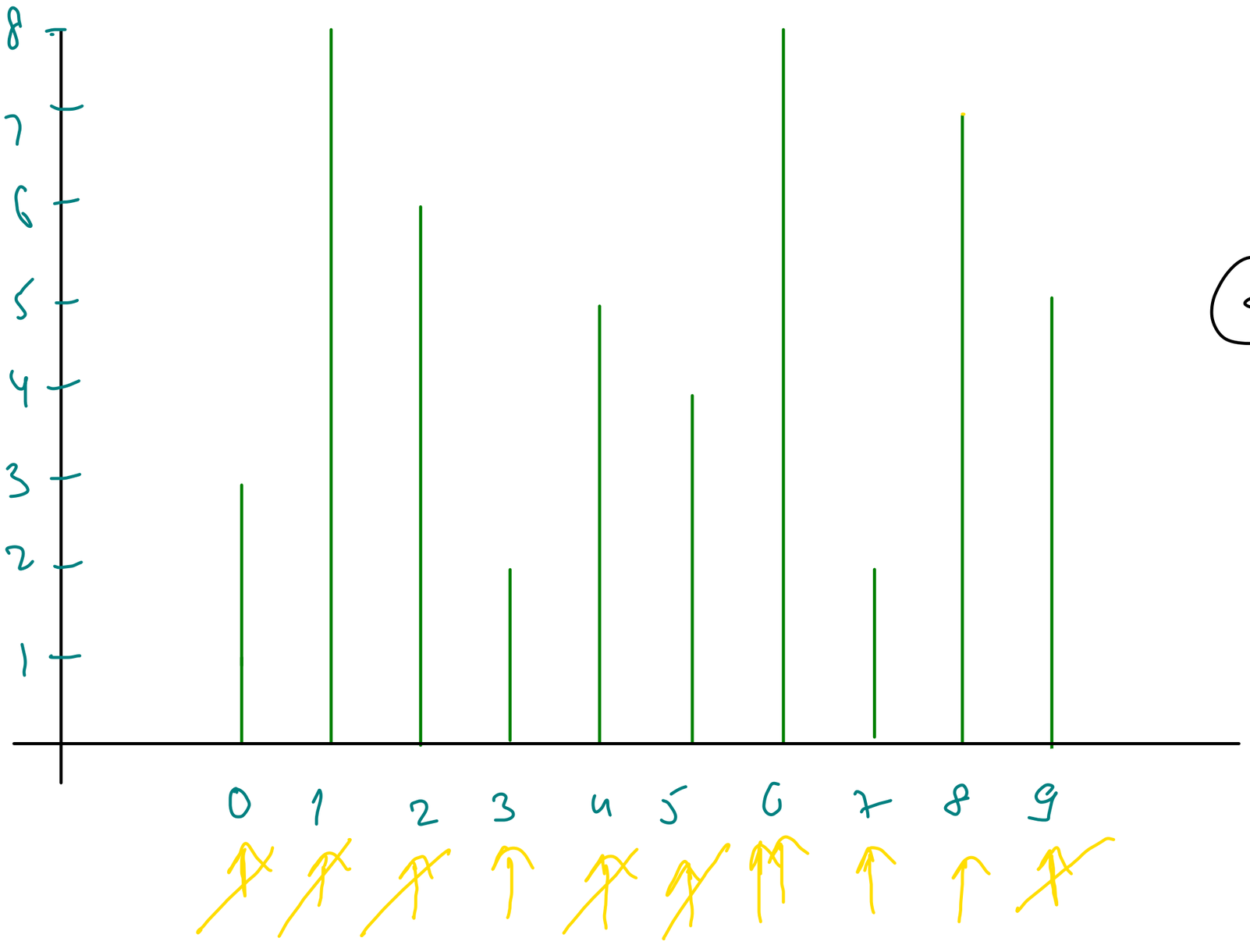
leftmax = 0 1 2

rightmax = 0 1 2

water = 0 + 1 + 1 + 2 + 1 + 1

```
int leftMax = 0, rightMax = 0;
int left = 0, right = arr.length - 1, water = 0;

while(left <= right){

    if(arr[left] <= arr[right]){
        if(arr[left] < leftMax) water += (leftMax - arr[left]);
        else leftMax = arr[left];
        left++;
    } else {
        if(arr[right] < rightMax) water += (rightMax - arr[right]);
        else rightMax = arr[right];
        right--;
    }
}

return water;
```

(11) Container with most water



(1) Brute force :→ $O(N^2)$

Forloop : first pillar

next-forloop : second pillar

(2) O(N) time O(1) space

easy two pointer

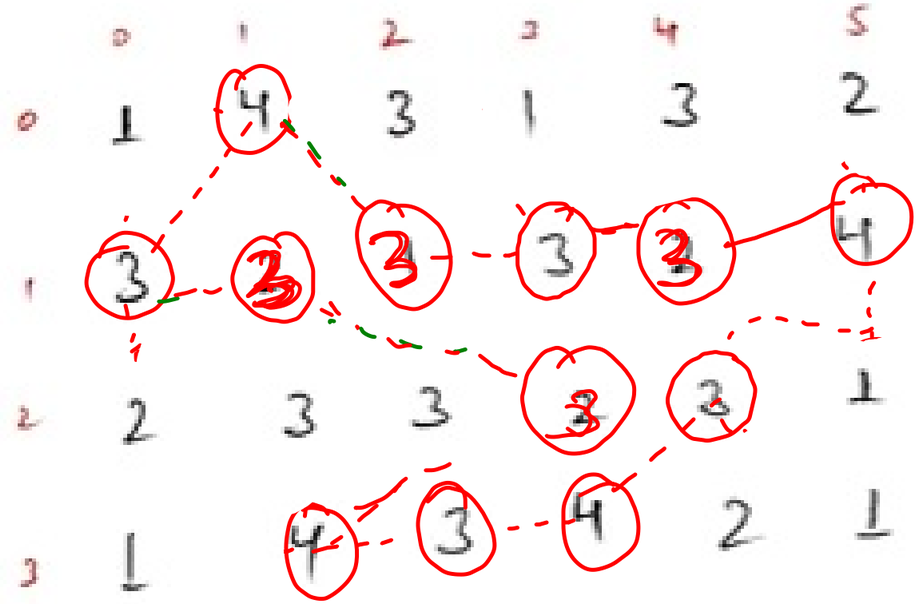
water = (9*3)

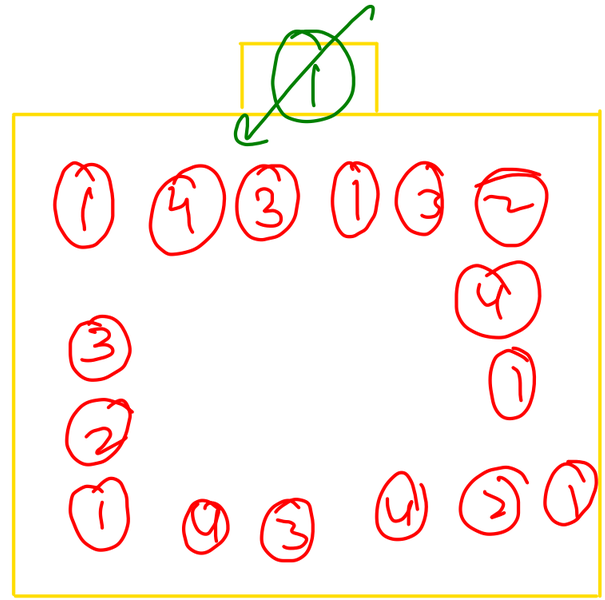
(8*8)

(7*7)

```java
public int maxArea(int[] arr) {
    int left = 0, right = arr.length - 1;
    int water = 0;

    while(left < right){
        int currWater = (right - left) * Math.min(arr[left], arr[right]);
        water = Math.max(water, currWater);

        if(arr[left] <= arr[right]) left++;
        else right--;
    }

    return water;
}
```

Trapping Rain Water - II





min heap
(PQ)

Water = 0 + 2 + 1 + 1 + 1

```java
public static class Pair implements Comparable<Pair>{}

int[] rows = {-1, 1, 0, 0};
int[] cols = {0, 0, 1, -1};

public int trapRainWater(int[][] mat) {
    PriorityQueue<Pair> q = new PriorityQueue<>();
    int row = mat.length, col = mat[0].length;

    for(int i=0; i<row; i++){}

    for(int j=0; j<col; j++){}

    int water = 0;
    while(q.size() > 0){
        Pair min = q.remove();

        for(int i=0; i<4; i++){
            int r = min.row + rows[i];
            int c = min.col + cols[i];

            if(r >= 0 && r < row && c >= 0 && c < col && mat[r][c] != -1){
                int newVal = mat[r][c];

                if(newVal < min.val){
                    water += (min.val - newVal);
                    q.add(new Pair(r, c, min.val));
                }
                else q.add(new Pair(r, c, newVal));
                mat[r][c]  = -1;
            }
        }
    }

    return water;
}
```

inserting boundary nodes in priority queue

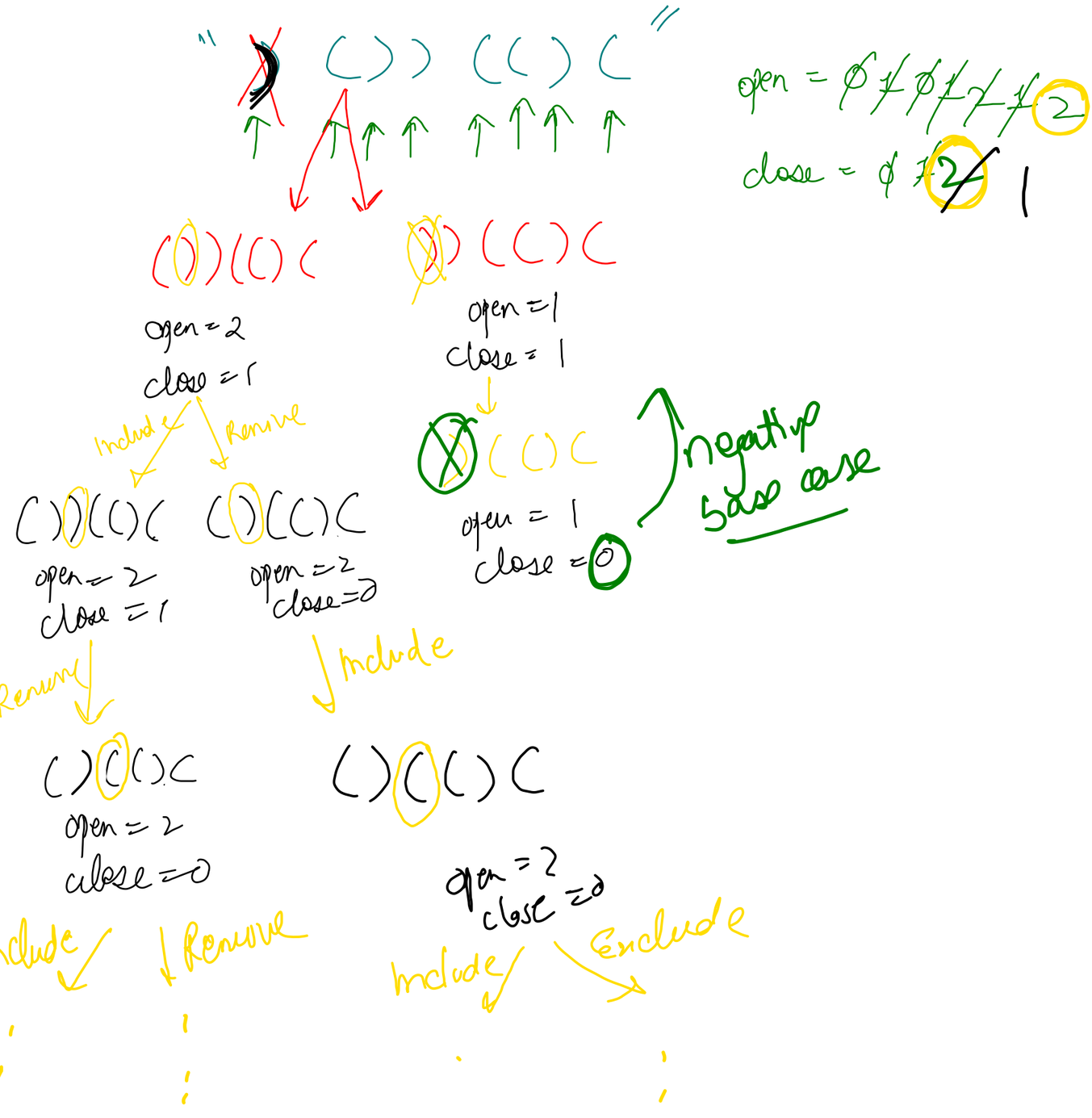
→ $O(n^2 \log n)$

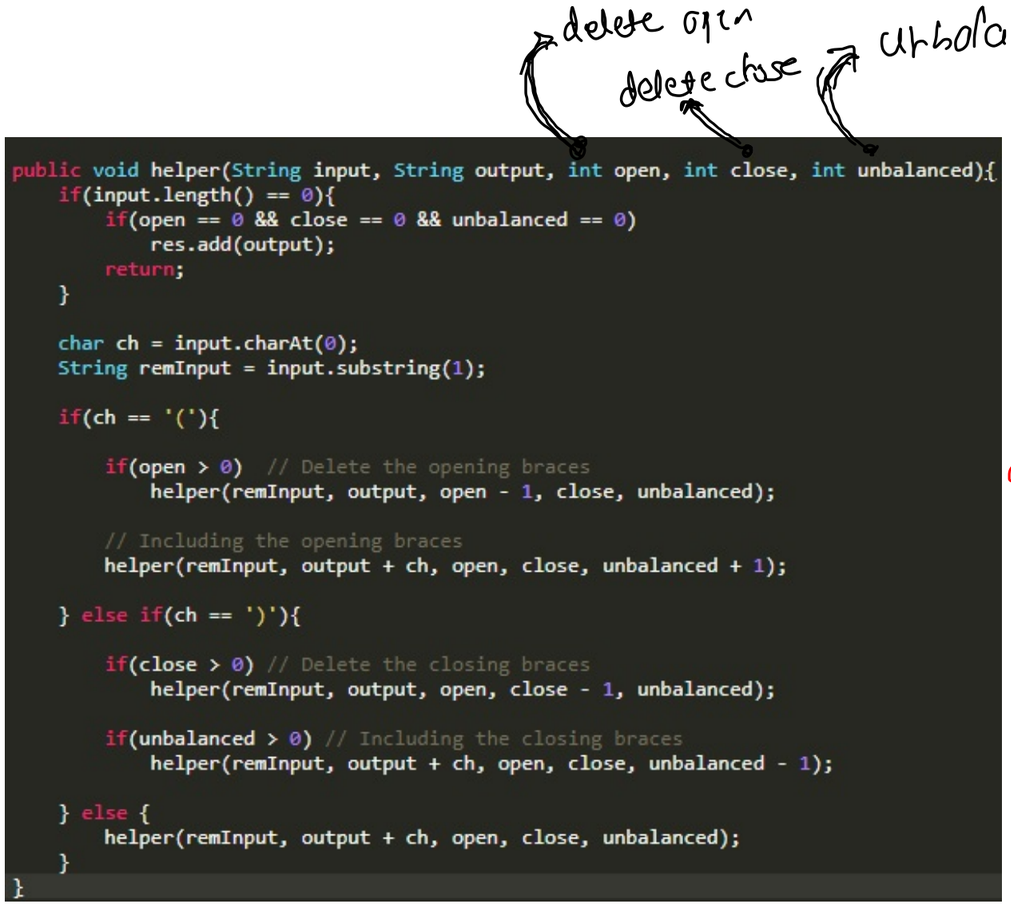
$row = col = n$

$4n$

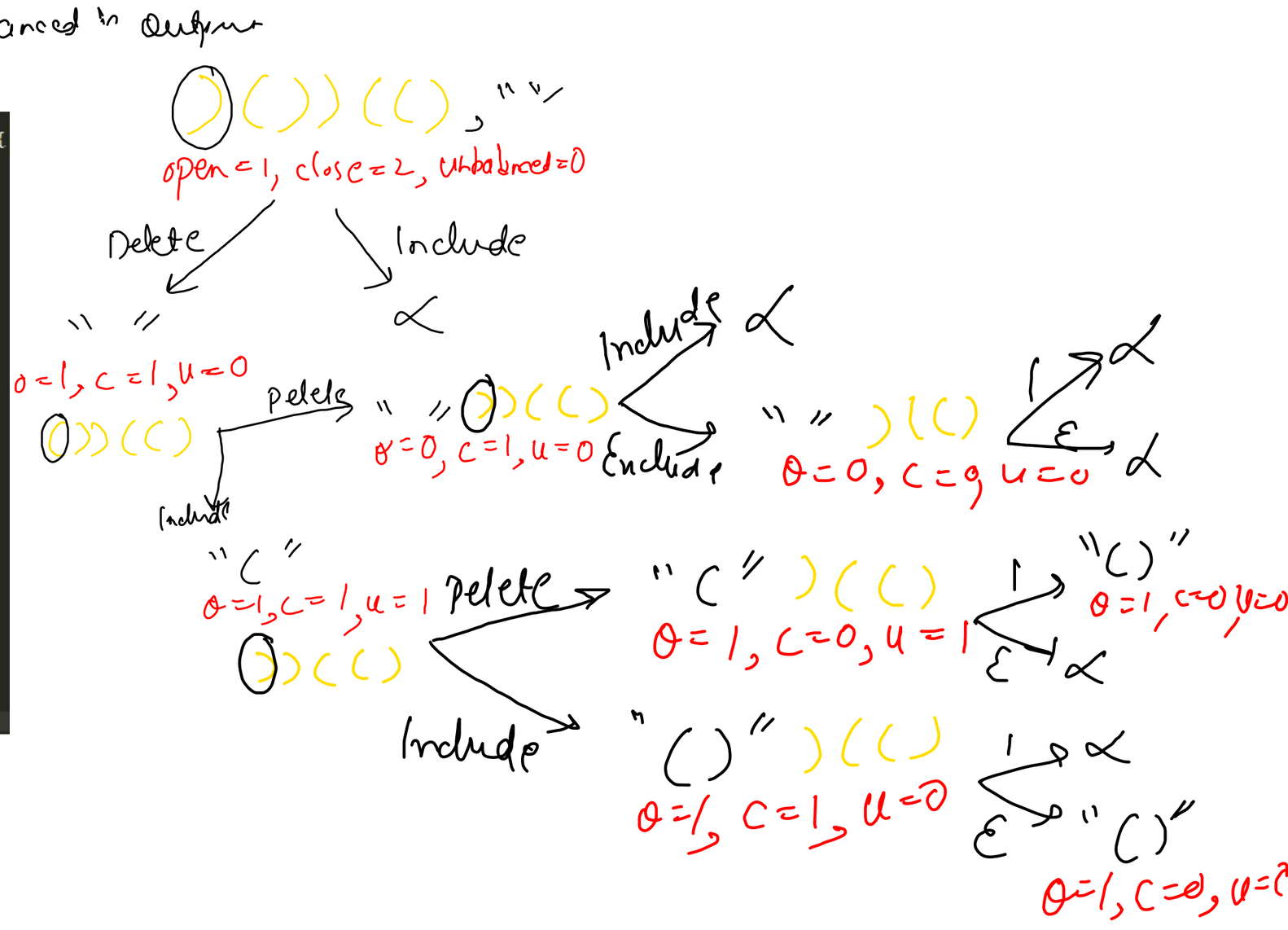
Priority Queue nodes

Remove Invalid Parentheses





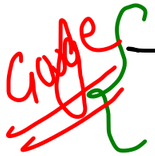
```java
public void helper(String input, String output, int open, int close, int unbalanced){
    if(input.length() == 0){
        if(open == 0 && close == 0 && unbalanced == 0)
            res.add(output);
        return;
    }

    char ch = input.charAt(0);
    String remInput = input.substring(1);

    if(ch == '('){

        if(open > 0)  // Delete the opening braces
            helper(remInput, output, open - 1, close, unbalanced);

        // Including the opening braces
        helper(remInput, output + ch, open, close, unbalanced + 1);

    } else if(ch == ')'){

        if(close > 0) // Delete the closing braces
            helper(remInput, output, open, close - 1, unbalanced);

        if(unbalanced > 0) // Including the closing braces
            helper(remInput, output + ch, open, close, unbalanced - 1);

    } else {
        helper(remInput, output + ch, open, close, unbalanced);
    }
}
```



Stack & Queue - L2 - Lecture (9)

→ Remove Duplicate letters

→ Remove K Digits
Most Competitive Subsequence → Same

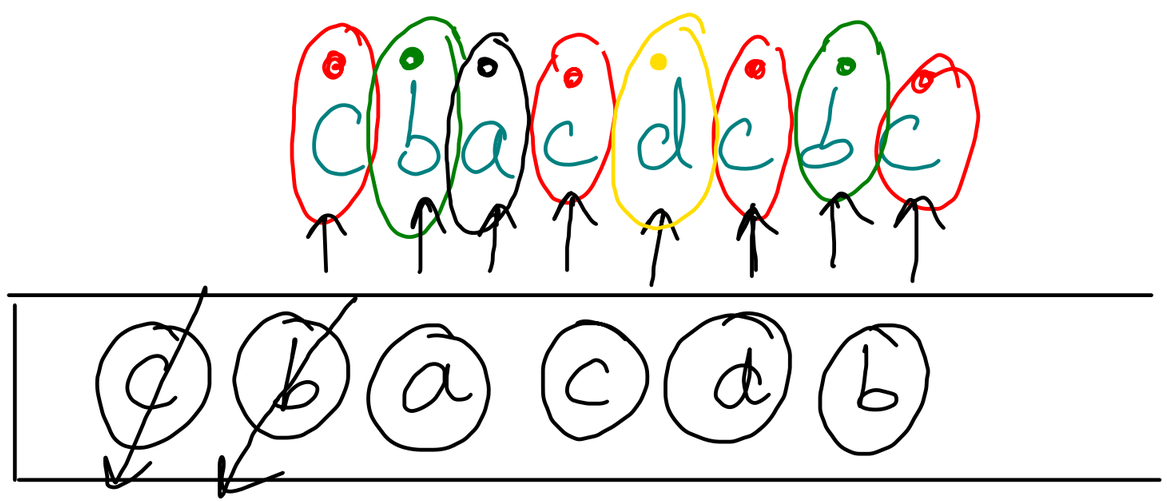


→ Asteroid Collision

→ Validate Stack Sequence

→ 132 Pattern

Remove Duplicate letters



lexicographically smallest subset

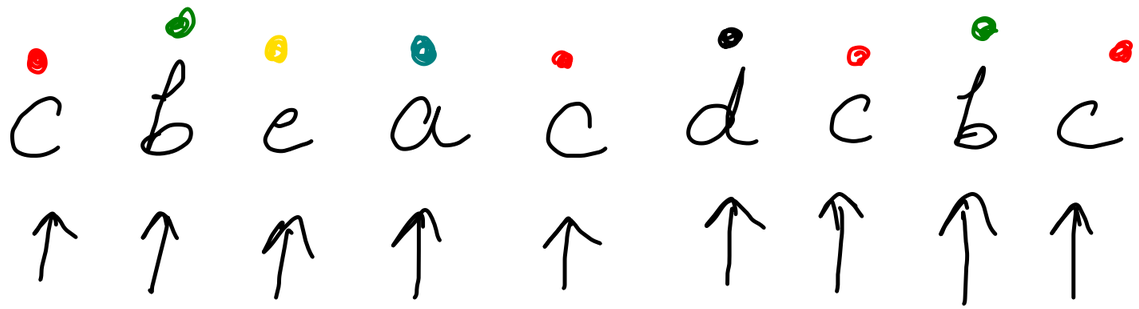
c → 4 3

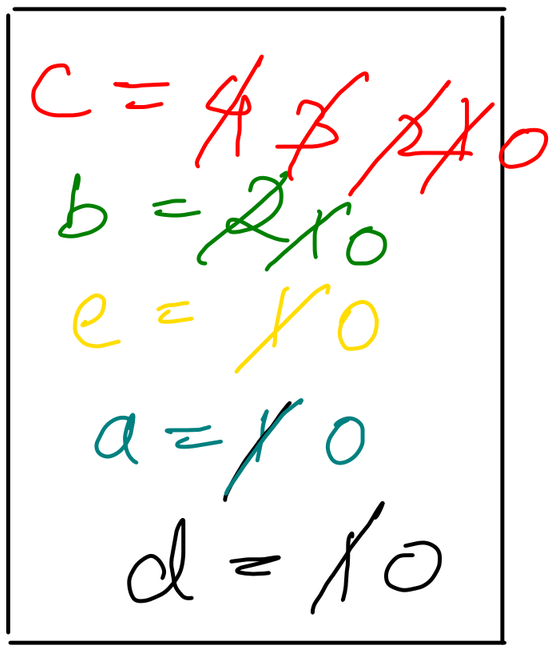
b → 2 1

a → 1

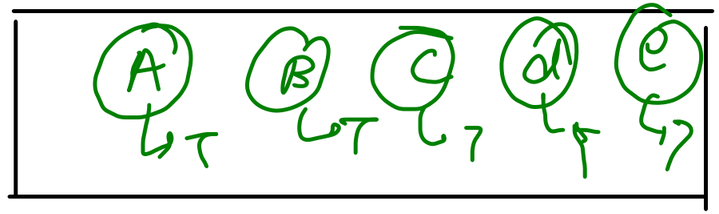
d → 1

cbad , cadb , bacd , acdb , - - - -

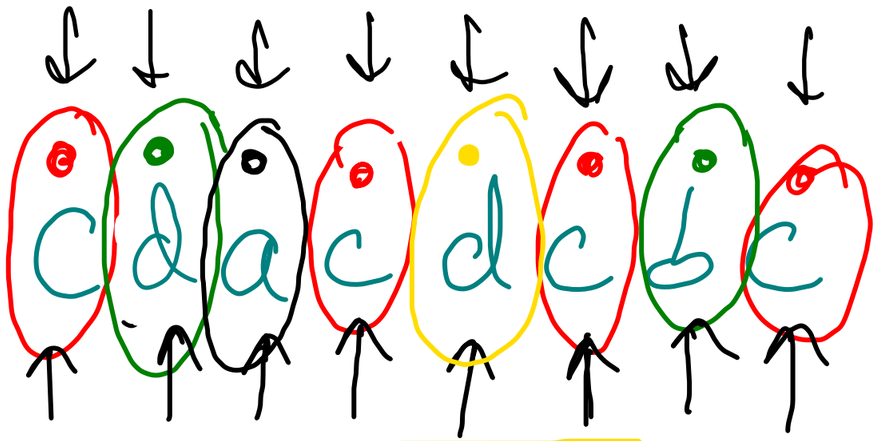


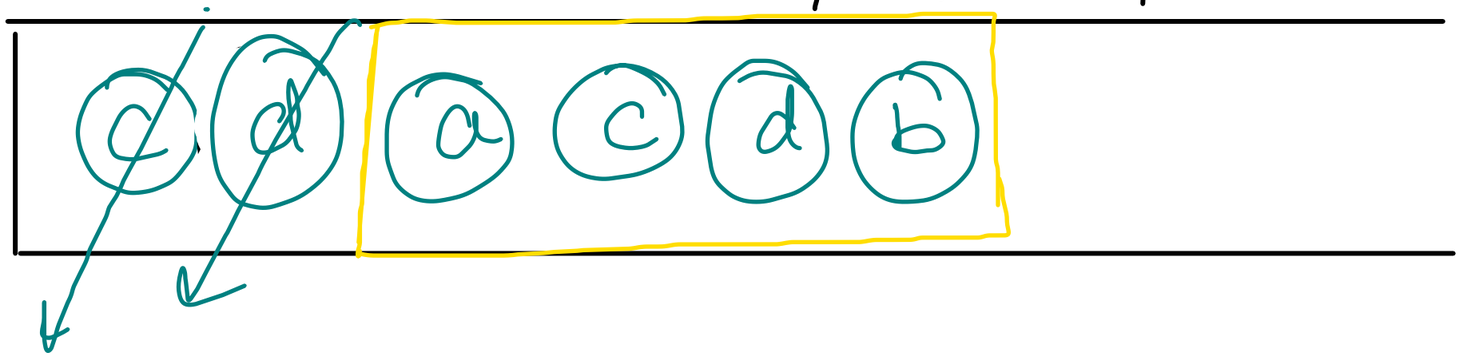


free



instack



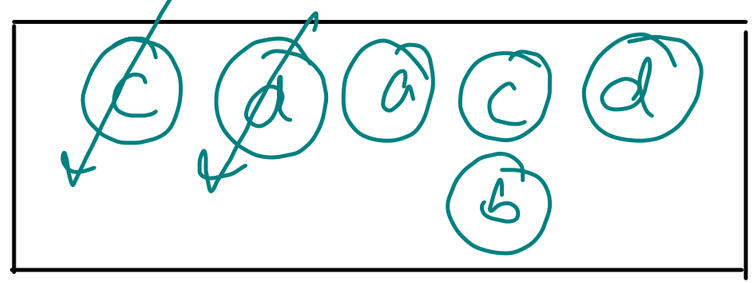


c = 4 3 2 1 freq

b = 2 0

a = 1 0

d = 2 1 0



instack

```java
Deque<Character> q = new ArrayDeque<>();

int[] freq = new int[26];
boolean[] inQ = new boolean[26];

for(int i=0; i<s.length(); i++)
    freq[s.charAt(i) - 'a']++;

for(int i=0; i<s.length(); i++)
{
    char ch = s.charAt(i);
    freq[ch - 'a']--;
    if(inQ[ch - 'a'] == true)
        continue;

    while(q.size() > 0 && q.getLast() > ch && freq[q.getLast() - 'a'] > 0){
        inQ[q.removeLast() - 'a'] = false;
    }

    q.addLast(ch);
    inQ[ch - 'a'] = true;
}

StringBuilder str = new StringBuilder("");
while(q.size() > 0){
    str.append(q.removeFirst());
}
return str.toString();
```

already present in Stack

Lenico Greater characters with atleast 1 freq.

O(N) Time

O(N) extra space (deque)

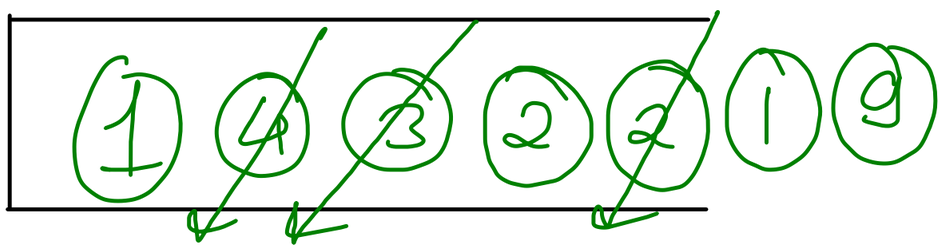
Remove K Digit

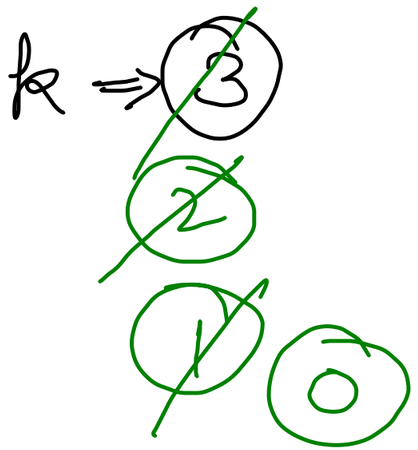
"Minimum Integer after removal"

k=0 in b/w

Case ①

"1432219"

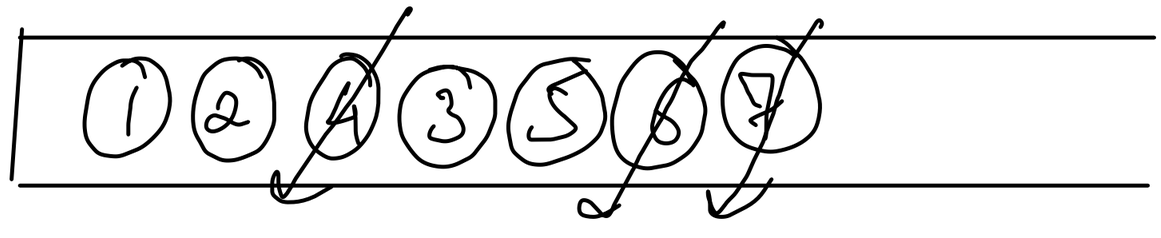


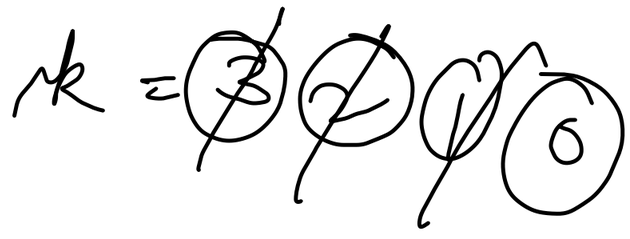


k>0 atlast

Case ②

"1243567"





```java
public String removeKdigits(String s, int removal) {
    Deque<Character> q = new ArrayDeque<>();

    for(int i=0; i<s.length(); i++){
        char ch = s.charAt(i);

        while(q.size() > 0 && removal > 0 && q.getLast() > ch){
            q.removeLast();
            removal--;
        }

        q.addLast(ch);
    }

    // removal > 0 even after entire string is traversed
    while(q.size() > 0 && removal > 0){
        q.removeLast();
        removal--;
    }

    // remove leading zeros
    while(q.size() > 0 && q.getFirst() == '0'){
        q.removeFirst();
    }

    if(q.size() == 0) return "0";
    StringBuilder str = new StringBuilder("");
    while(q.size() > 0){
        str.append(q.removeFirst());
    }
    return str.toString();
}
```

O(N) time

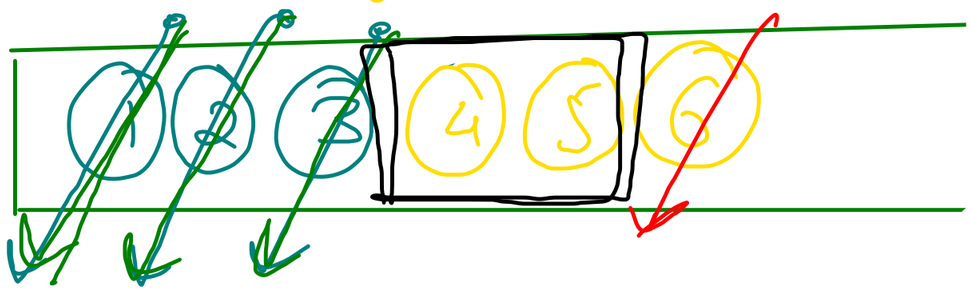
O(N) extra space { deque }

case ③

leading zeros

1 2 3 0 0 0 4 5 6

stack empty → leading zeros

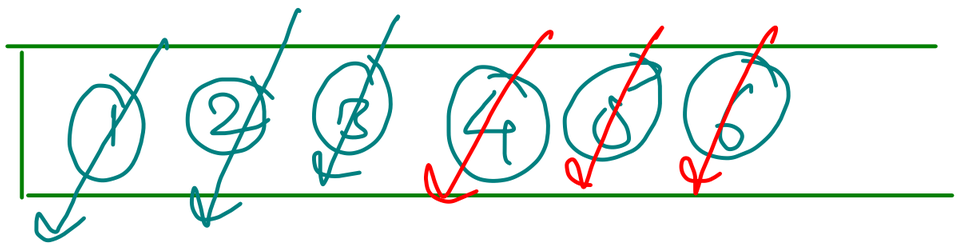


k = 4 3 2 1 0

case 4

corner case after removals stack is empty

1 2 3 0 0 4 5 6



k = 6 5 4 3

return "0"

Lecture - (10) LL + S&Q

- Validate Stack Sequence
- Asteroid Collision ⭐ Google
- Exclusive Time of Fn
- Subtract 2 LL, multiply 2 LL
- Flatten LL - I, II — Josh
- LFU Cache

Revision

Top 100

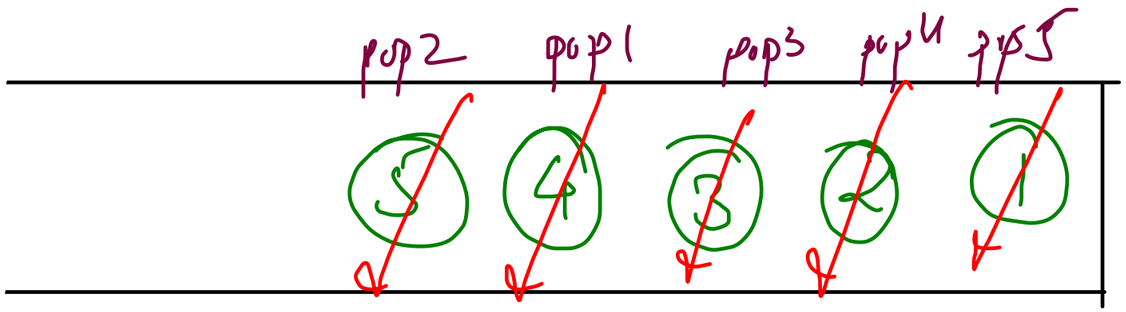
Amazon (80-90%)

Microsoft (80-90%)

Validate Stack Sequences {946}

Example 1:

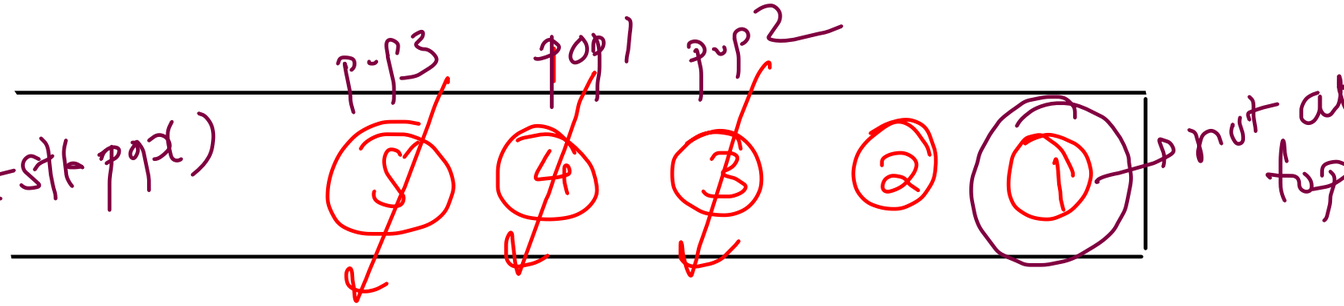
```
Input: pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
Output: true
Explanation: We might do the following sequence:
push(1), push(2), push(3), push(4),
pop() -> 4,
push(5),
pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```



Example 2:

```
Input: pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
Output: false
Explanation: 1 cannot be popped before 2.
```

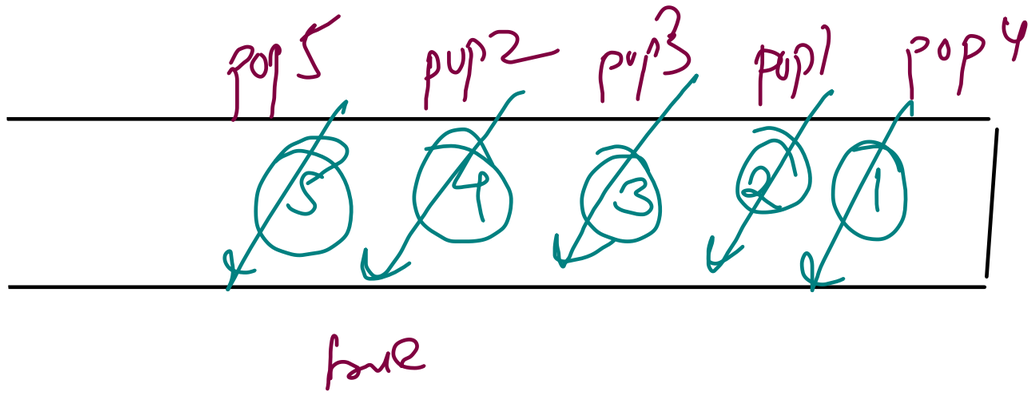
→ Not at stk top



Example 3:

push [1, 2, 3, 4, 5]

pop [2, 4, 3, 1, 5]



```java
public boolean validateStackSequences(int[] pushed, int[] popped) {
    Stack<Integer> stk = new Stack<>();
    HashSet<Integer> inStack = new HashSet<>();

    int push = 0;
    for(int pop = 0; pop < popped.length; pop++) {
        if(inStack.contains(popped[pop]) == true){
            if(stk.size() == 0 || stk.peek() != popped[pop])
                return false;
            else stk.pop();
        }
        else {
            while(push < pushed.length && pushed[push] != popped[pop]){
                stk.push(pushed[push]);
                inStack.add(pushed[push]);
                push++;
            }
            push++; // For the element to be popped
        }
    }


    return ((stk.size() == 0) ? true : false);
}
```

O(N) time

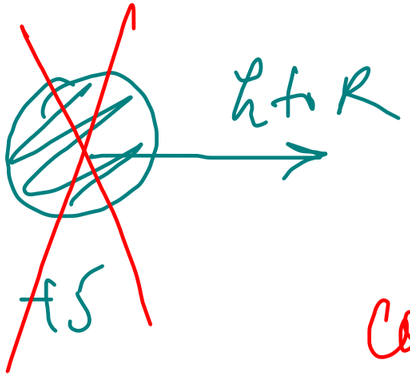
O(N) space

Asteroid Collision (735)

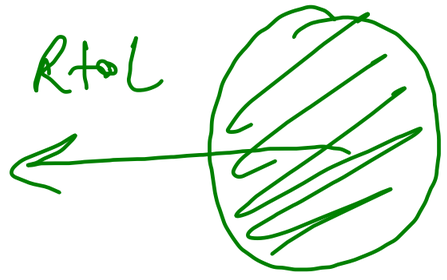
We are given an array `asteroids` of integers representing asteroids in a row.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

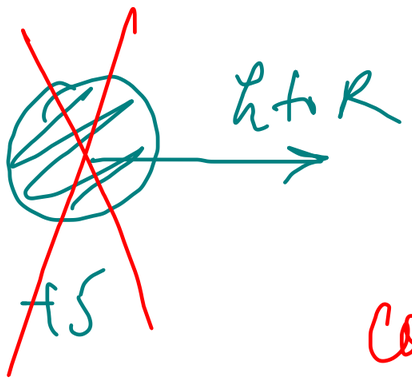
Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.



collision



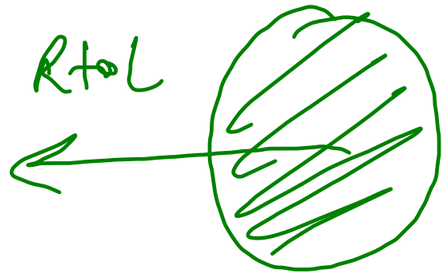
−10



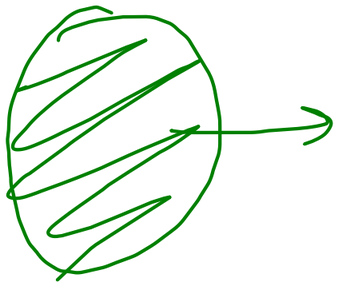
collision 1

S ↔ L

(pop)



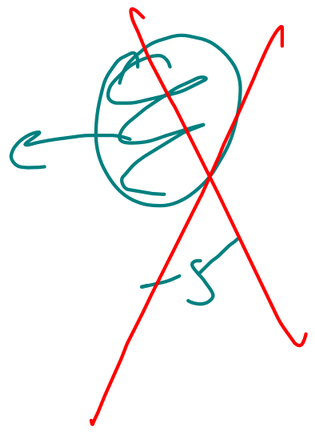
collision will not take place



collision 2

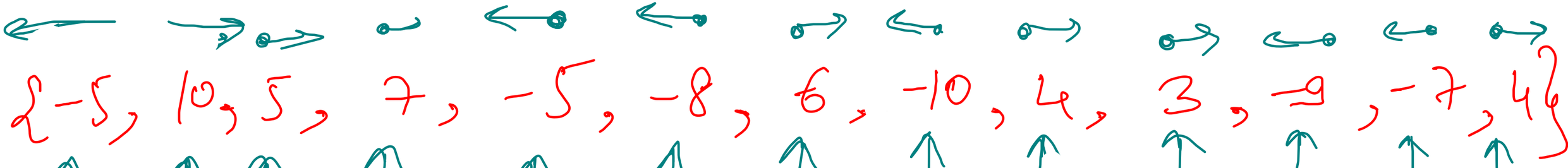
L ↔ S

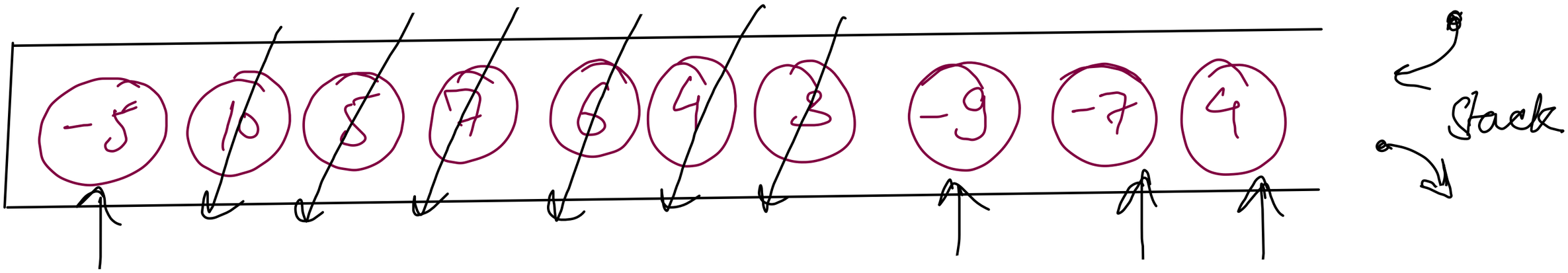
(no push & break)

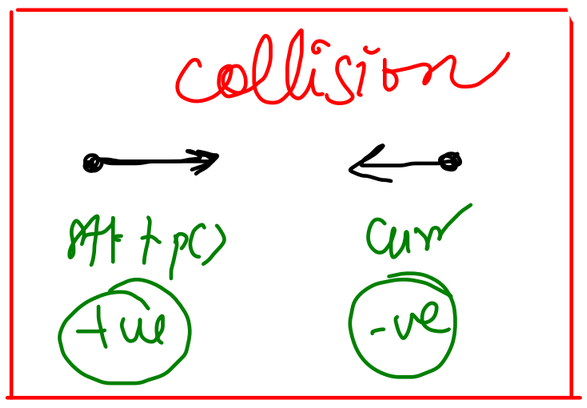


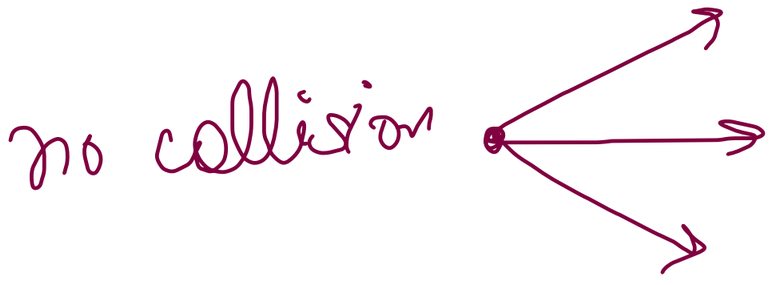
Collision 3

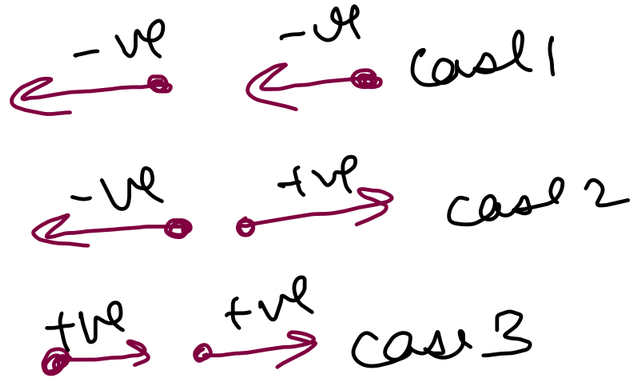
(pop & nopush & break)


{-5, 10, 5, 7, -5, -8, 6, -10, 4, 3, -9, -7, 4}


-5
10
5
7
6
4
3
-9
-7
4
Stack

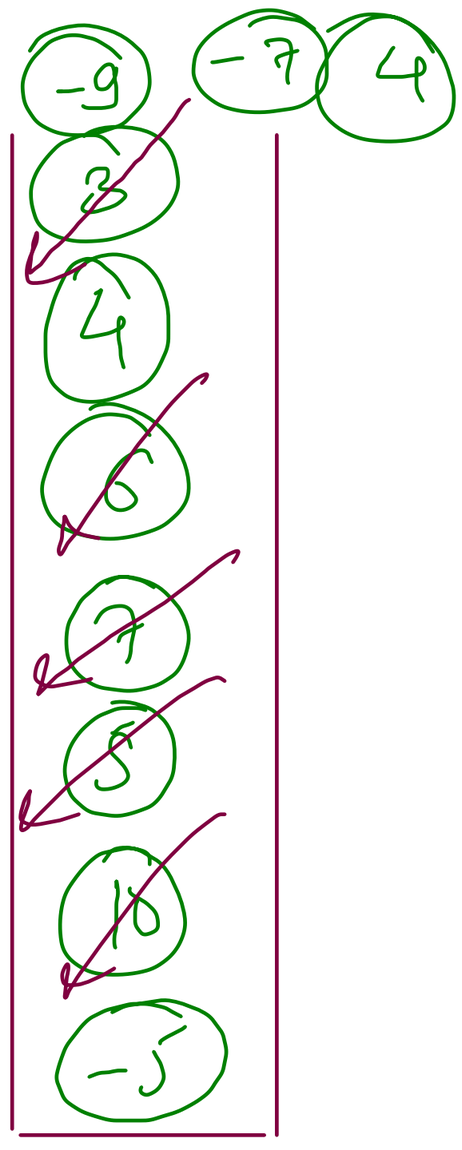

collision
stack.top()
curr
+ve
-ve


no collision


-ve
-ve
case 1
-ve
+ve
case 2
+ve
+ve
case 3

{-5, 10, 5, 7, -5, -8, 6, -10, 4, 3, -9, -7, 4}

```java
public int[] asteroidCollision(int[] asteroids) {
    Stack<Integer> stk = new Stack<>();

    for(int val: asteroids){
        if(val < 0){
            boolean destroy = false;
            while(stk.size() > 0 && stk.peek() > 0){
                if(stk.peek() < -val){
                    stk.pop();
                } else if(stk.peek() > -val){
                    destroy = true;
                    break;
                } else {
                    stk.pop();
                    destroy = true;
                    break;
                }
            }

            if(destroy == false)
                stk.push(val);
        }
        else stk.push(val);
    }

    int[] res = new int[stk.size()];
    int idx = res.length - 1;
    while(idx >= 0){
        res[idx--] = stk.pop();
    }
    return res;
}
```



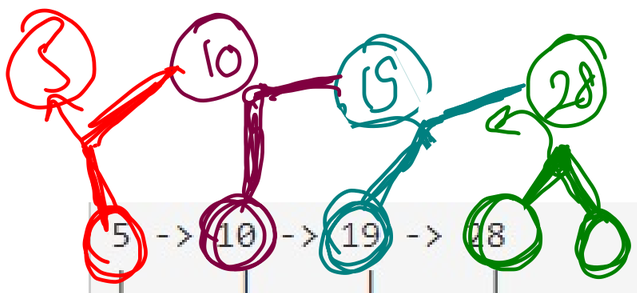
10:05

O(N) time ↘ 1 push, 1 pop } for each ele

O(N) space → Stack

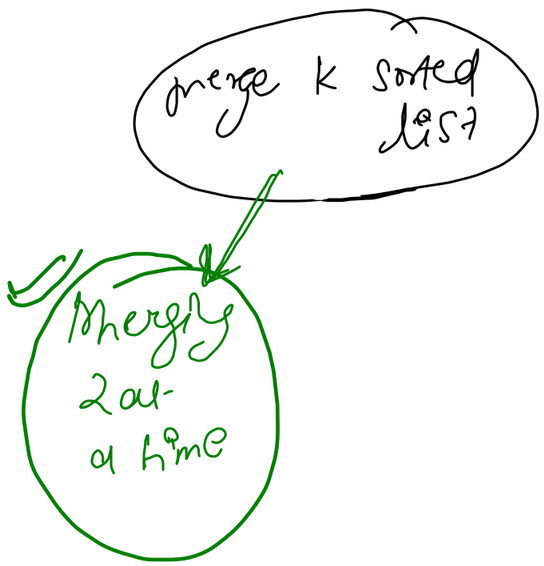
Flatten LL - (1)

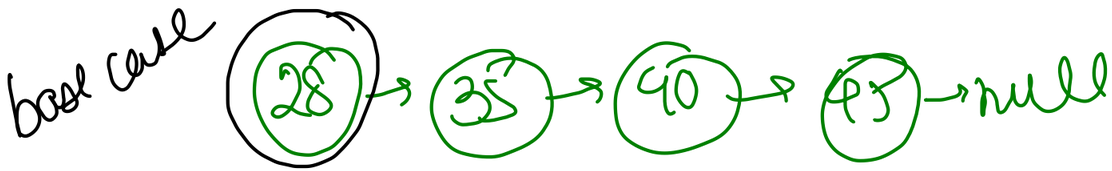
All these pics are bottom in res

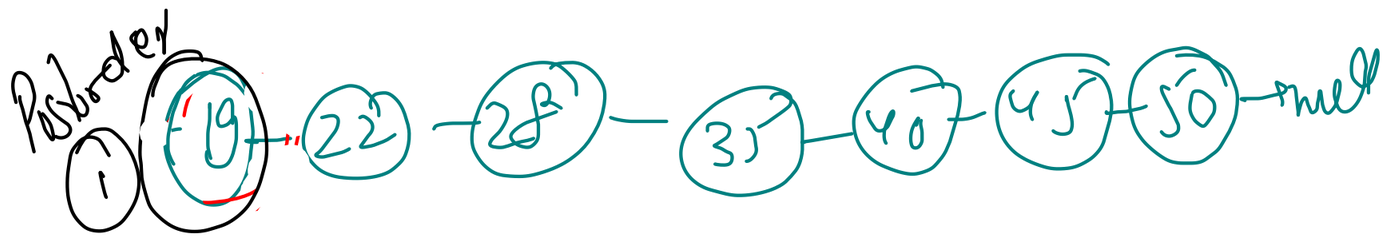


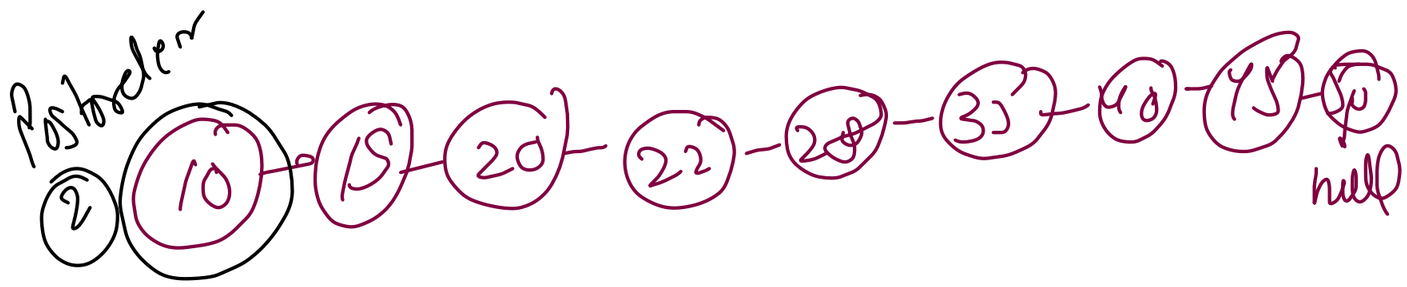
```
5 -> 10 -> 19 -> 28
|     |     |     |
7     20    22    35
|           |     |
8           50    40
|                 |
30                45
```

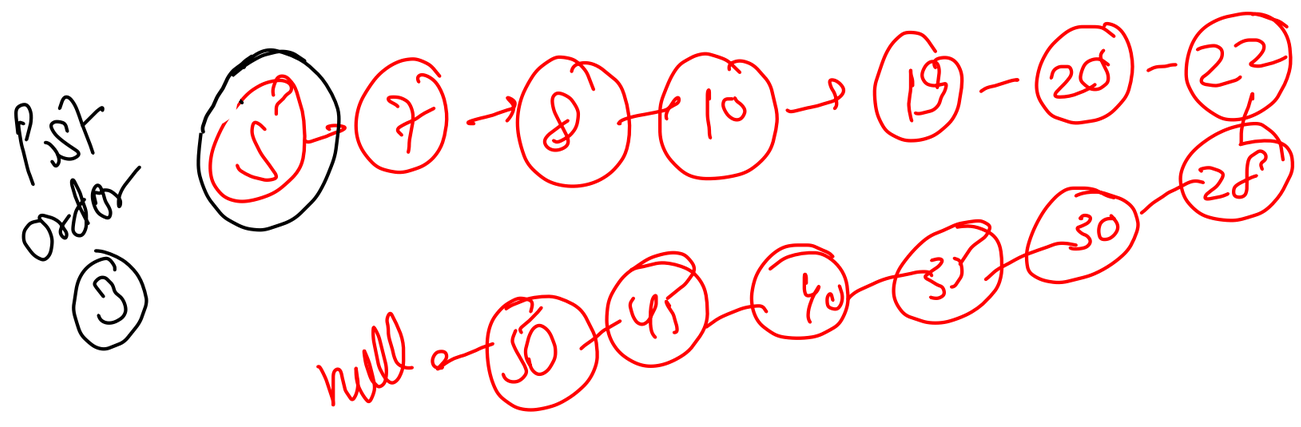
Output: 5-> 7-> 8- > 10 -> 19-> 20->
22-> 28-> 30-> 35-> 40-> 45-> 50.









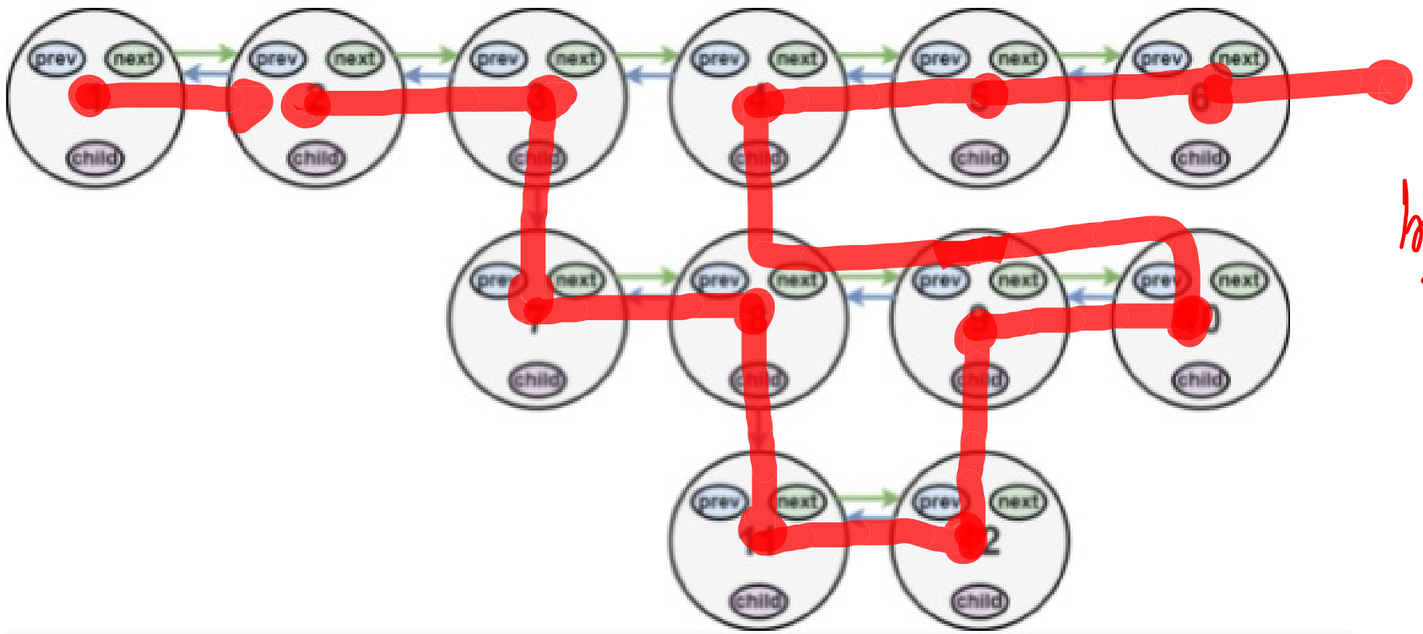


```
Node merge(Node a, Node b){

    Node dummy = new Node(-1);
    Node tail = dummy;

    while(a != null && b != null){
        if(a.data <= b.data){
            tail.bottom = a;
            tail = a;
            a = a.bottom;
        } else {
            tail.bottom = b;
            tail = b;
            b = b.bottom;
        }
    }

    while(a != null){
        tail.bottom = a;
        tail = a;
        a = a.bottom;
    }

    while(b != null){
        tail.bottom = b;
        tail = b;
        b = b.bottom;
    }

    return dummy.bottom;
}
```

```
Node flatten(Node root)
{
    if(root == null || root.next == null){
        return root;
    }

    Node b = flatten(root.next);
    return merge(root, b);
}
```

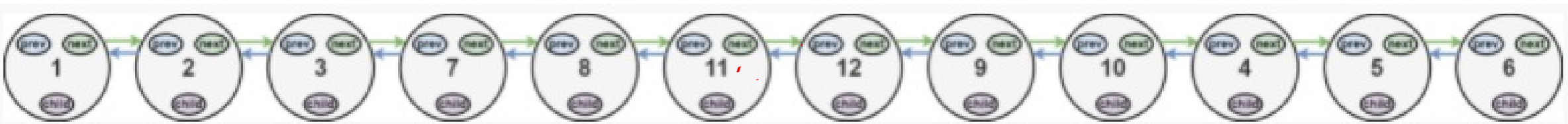
Flatten (II) (Multilevel DLL)

11:10

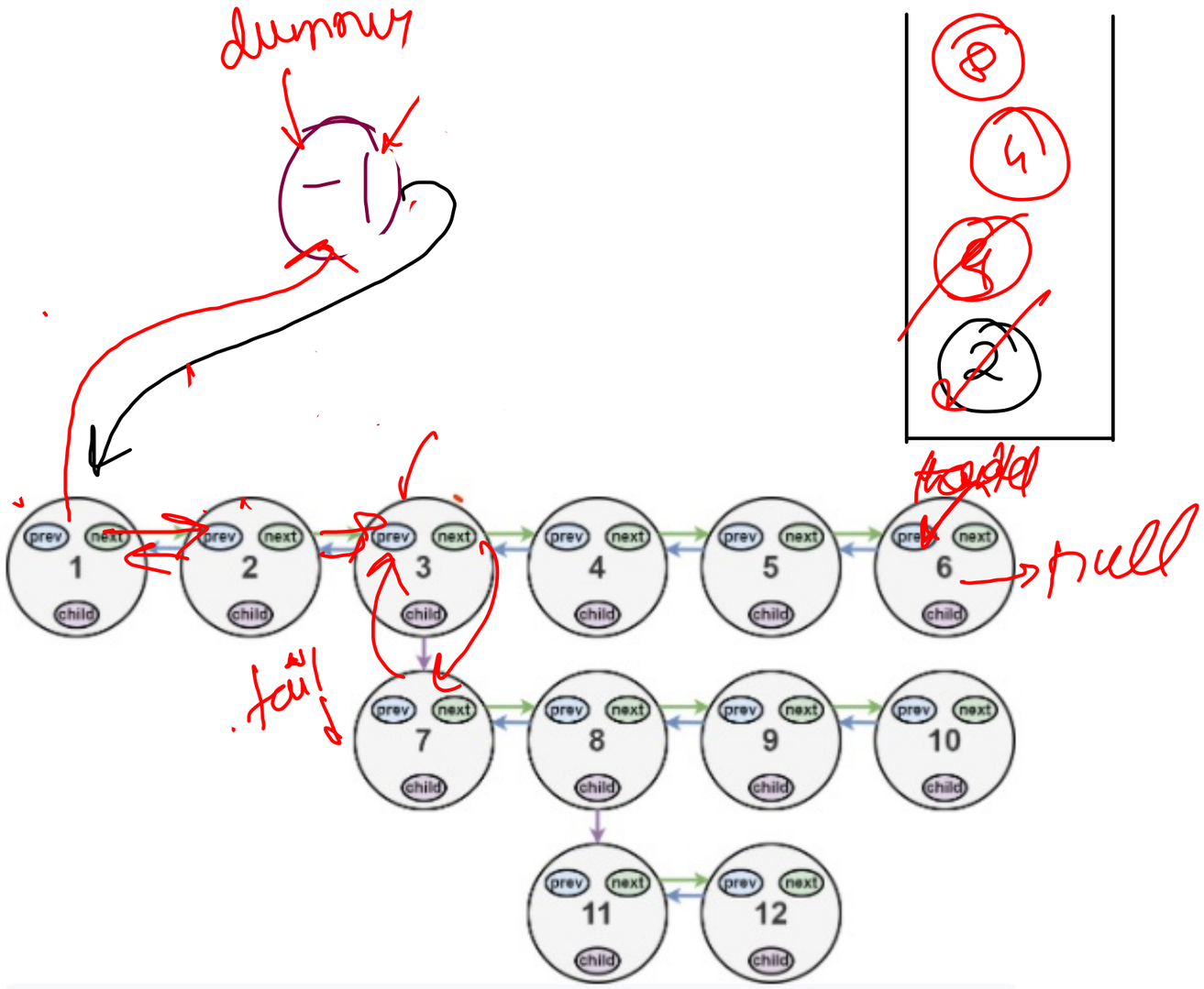


Input

similar to moves inorder



Output




```
Stack<Node> stk = new Stack<>();
Node dummy = new Node(-1);
Node tail = dummy;

while(stk.size() > 0 || head != null){
    if(head != null){
        if(head.next != null)
            stk.push(head.next);

        tail.next = head;
        head.prev = tail;
        tail = head;

        head = head.child;
    }
    else {
        Node temp = stk.pop();
        head = temp;
    }
}

return dummy.next;
```