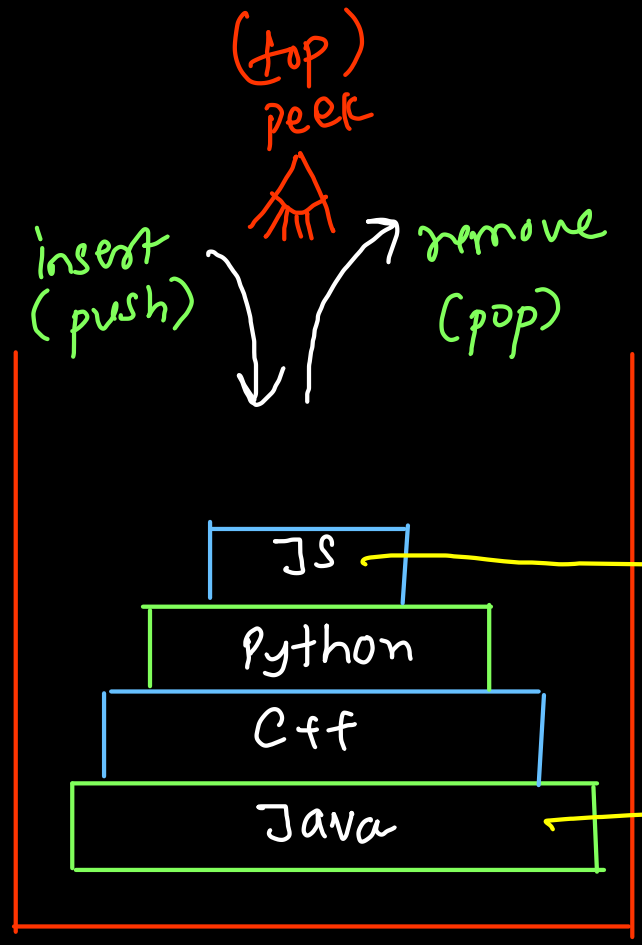
Stack



Depth First Traversal (DFS)

Programming ⇒ Recursion call stack
or Function call stack

Text Editor (undo & Redo)

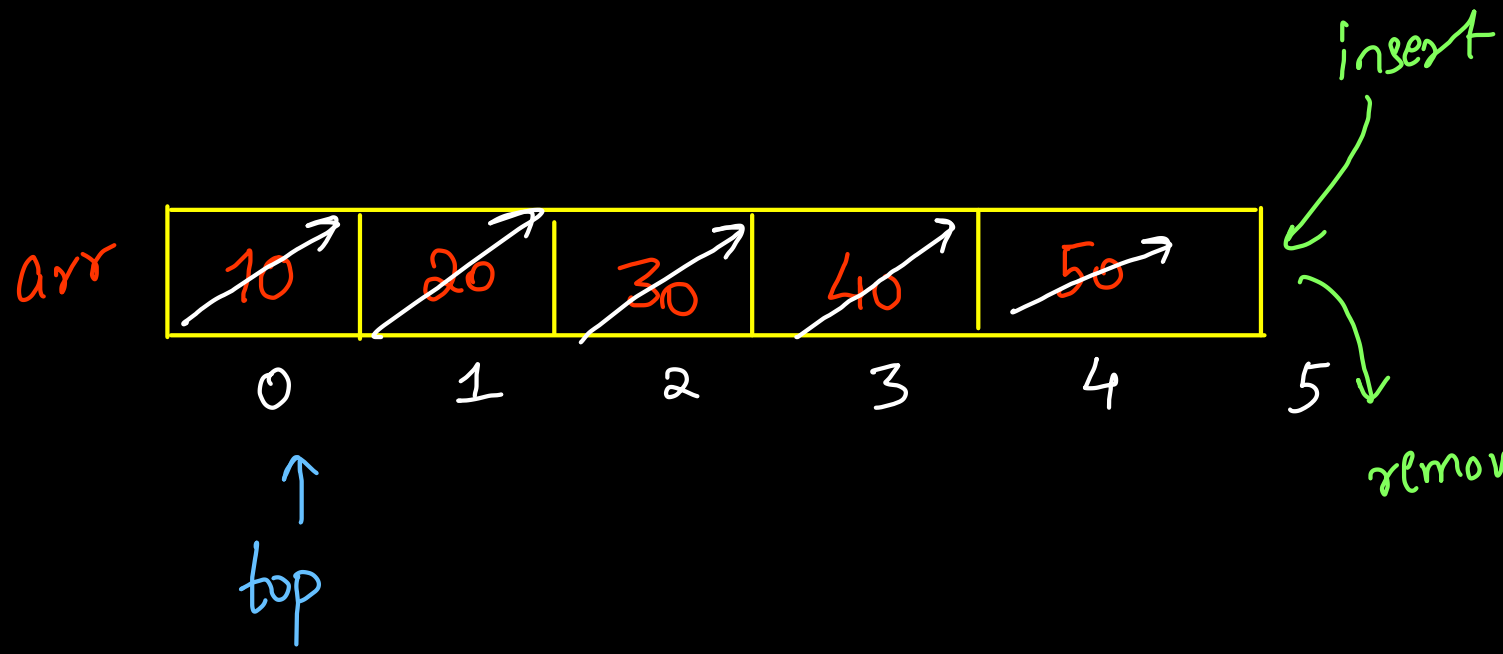
Expressions Evaluation → Parantheses
Operators

Last In / First out {LIFO}

First In / Last out {FILO}

class (Collection Framework)

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

stk.push(10); //O(1)
stk.push(20);
stk.push(30);
stk.push(40);

System.out.println(stk.peek()); // O(1)
System.out.println(stk); // O(n)

// Total Loop = O(n)
while(stk.size() > 0){
    System.out.println(stk.pop()); // O(1)
    System.out.println(stk);
}
```



array

push → O(1) from right

pop → O(1) from right

push

```
arr[top] = value;
top++;
```

pop

```
top--;
return arr[top];
```

push(10)
push(20)
push(30)
push(40)
push(50)

push(60)
↳ Stack Overflow

pop() = 50
pop() = 40
pop() = 30
pop() = 20
pop() = 10
pop() ⇒ Stack Underflow

Stack using Array

```java
class MyStack
{
    int top = 0;
    int arr[] = new int[1000];

    void push(int value)
    {
        if(top == arr.length){
            return; // Stack Overflow
        }
        arr[top] = value;
        top++;
    }

    int pop()
    {
        if(top == 0){
            return -1; // Stack Underflow
        }
        top--;
        return arr[top];
    }
}
```

O(1)

O(1)

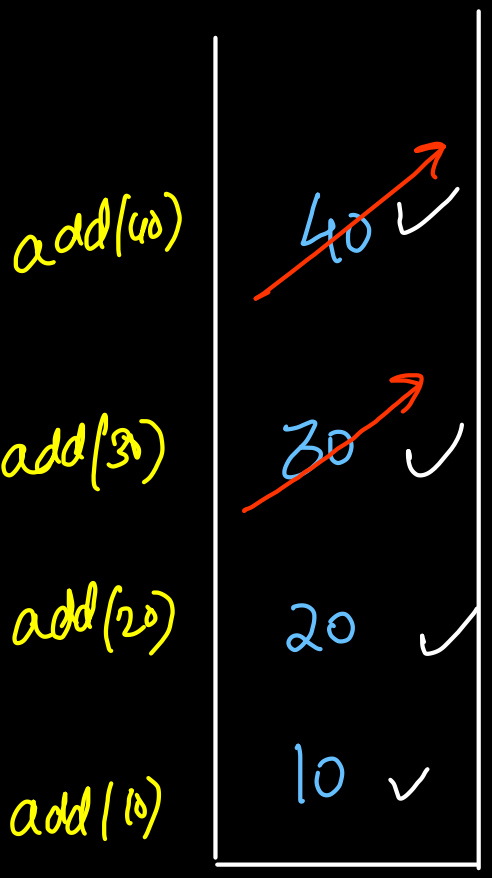
Stack using Arraylist ! (composition) ☺

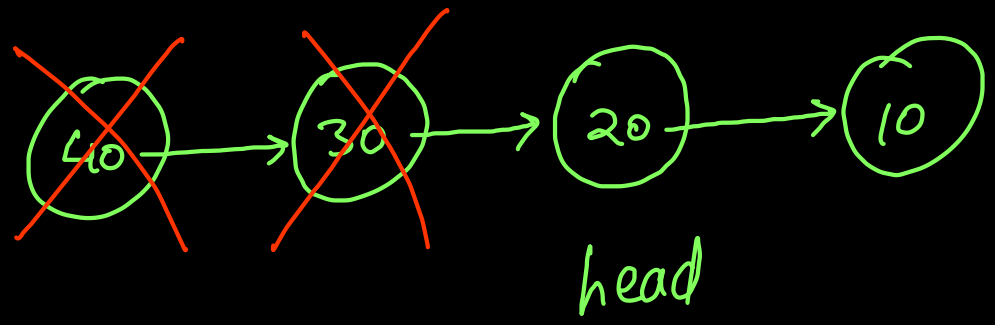
has a

```java
class MyStack
{
    ArrayList<Integer> arr = new ArrayList<>();

    void push(int value) {
        arr.add(value); // O(1) Insertion At End
    }

    int pop()
    {
        if(arr.size() == 0){
            return -1; // Stack Underflow
        }

        return arr.remove(arr.size() - 1);
        // O(1) Deletion From End
    }
}
```

$10^7 - 10^8$

Stack using list





Singly linked list with only head

① addAtHead, removeAtHead
↳ O(1) ↳ O(1)

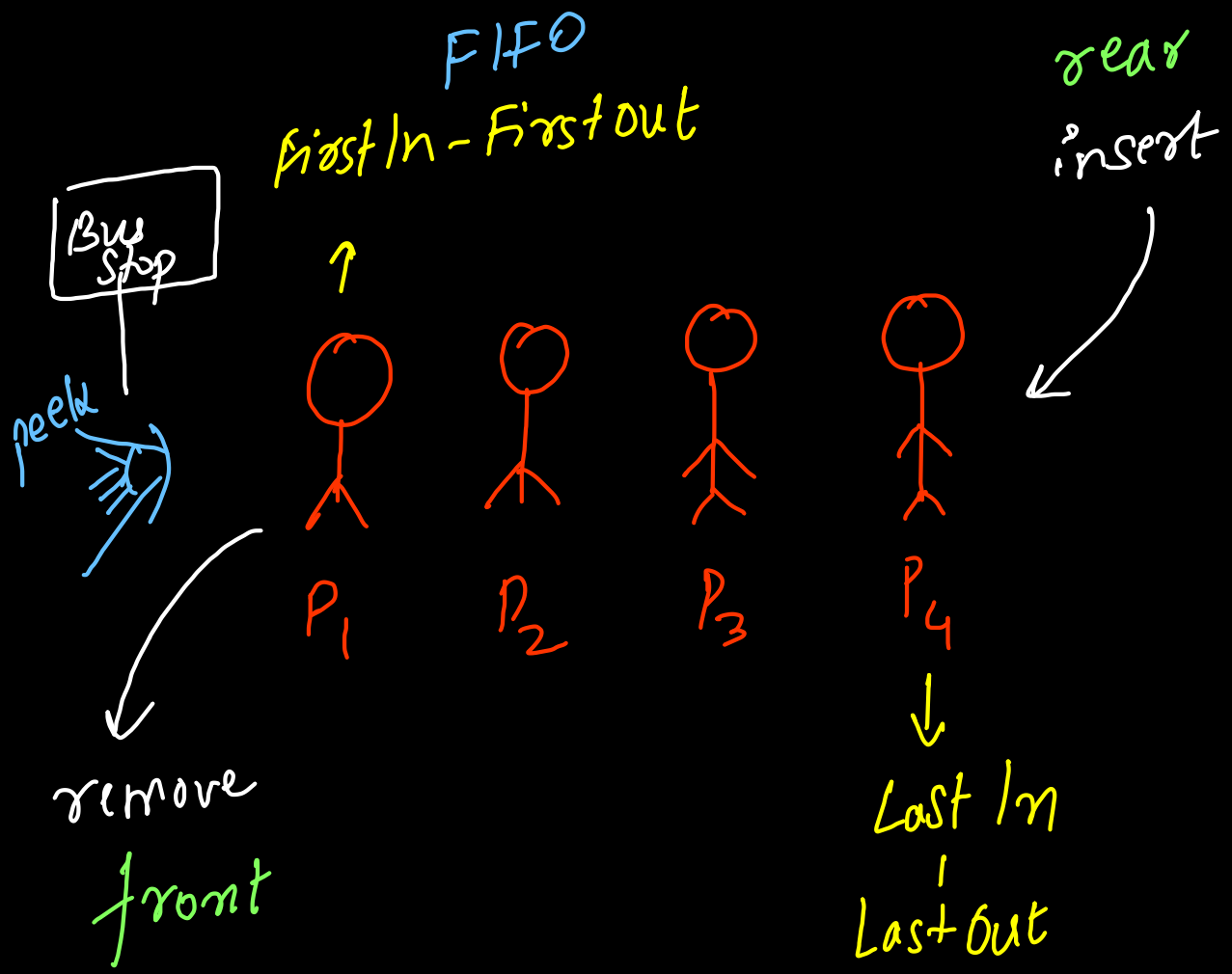
② addAtTail, removeAtTail
↳ O(n) ↳ O(n)

```
class MyStack
{
    StackNode head = null;

    void push(int value)  // insert at head
    {
        StackNode node = new StackNode(value);
        node.next = head;
        head = node;
    }

    int pop()     // remove at head
    {
        if(head == null){
            return -1; // Stack Underflow
        }
        int ans = head.data;
        head = head.next;
        return ans;
    }
}
```

O(1)

O(1)

Queue



Breadth First Traversal or BFS

Sliding window

Binary Nos (1-N) Generatn

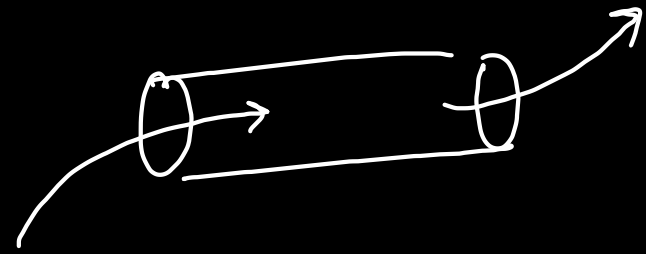
Security ⇒ bag {airport}

Queue using Linked list

{ Singly linked List with head & tail }

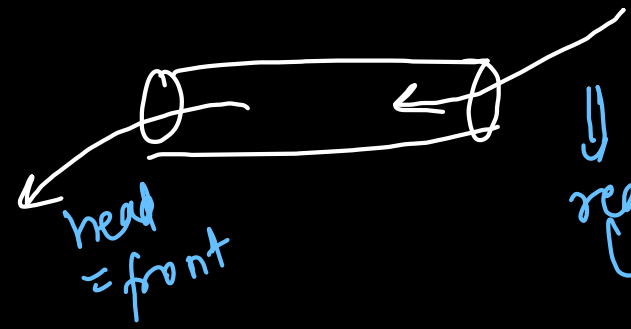
(1) insertAtHead
↳ O(1)

/ removeAtTail
↳ O(n)



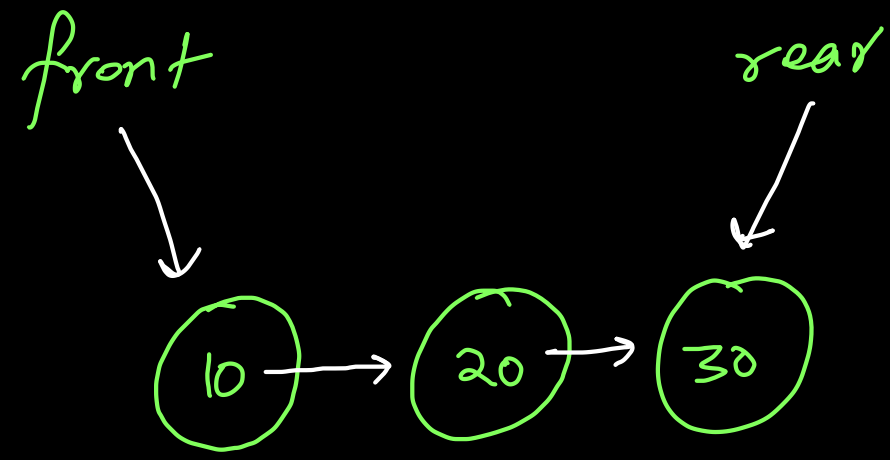
(2) insertAtTail
↳ O(1)

/ removeAtHead
↳ O(1)



```
QueueNode front, rear;

//Function to push an element into the queue.
void push(int value)
{
    // Insert at Tail/Rear
    QueueNode node = new QueueNode(value);

    if(head == null){
        front = rear = node;
    } else {
        rear.next = node;
        rear = node;
    }
}
```

O(1)



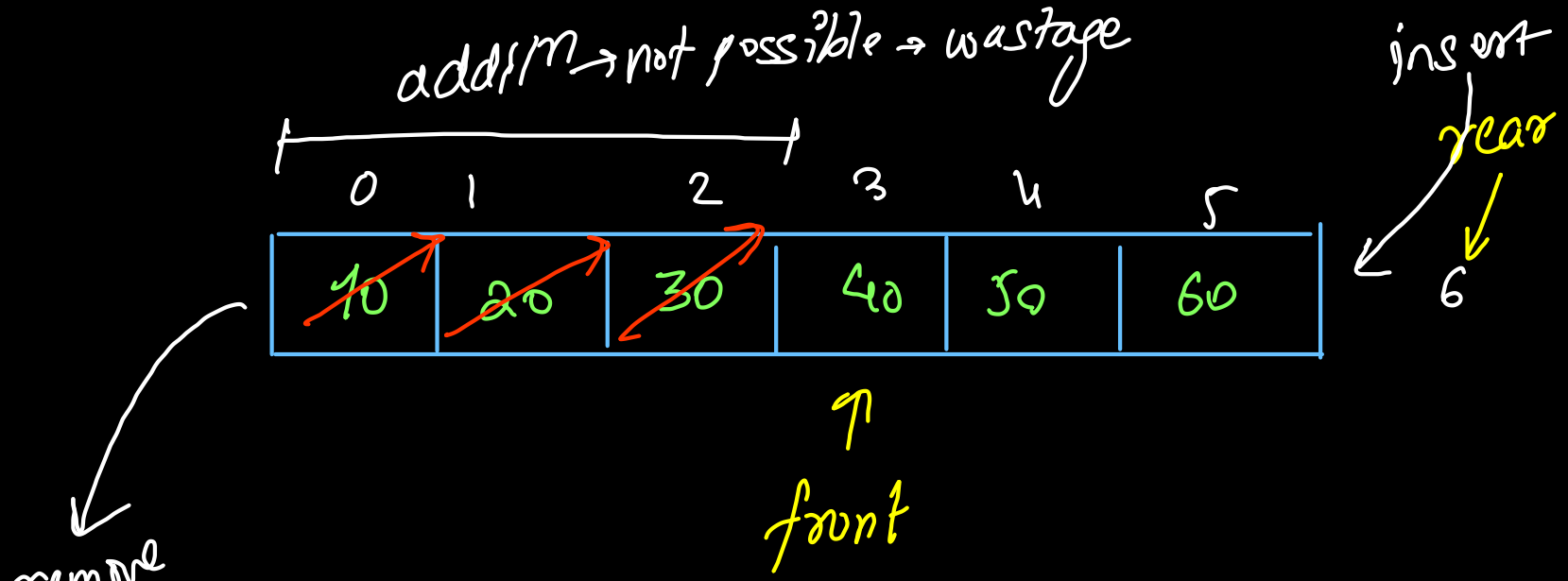
push (10)
push (20)
push (30)

Queue using SLL

```
class MyQueue
{
    QueueNode front, rear;

    //Function to push an element into the queue.
    void push(int value) {
        // Insert at Tail/Rear
        QueueNode node = new QueueNode(value);
        if(front == null){
            front = rear = node;
        } else {
            rear.next = node;
            rear = node;
        }
    }

    int pop() {
        // Delete at Head
        if(front == null){
            return -1; // Queue Underflow
        }
        int ans = front.data;
        front = front.next;
        return ans;
    }
}
```

O(1)

O(1)



capacity = 6
(= arr.length)

size = 0 1 2 ... 6
(3)

push(10)
push(20)
push(30)
push(40)
push(50)
push(60)
push(70)
↳ Queue overflow

remove()
remove()
remove()

push

```
arr[rear] = value;
rear ++
```

pop

```
ans = arr[front]
front ++
return ans;
```

Queue using normal array

```
class MyQueue {

    int front = 0, rear = 0, size = 0;
    int arr[] = new int[100005];

    void push(int value)
    {
        if(size == arr.length) {
            return; // Queue Overflow
        }
        arr[rear] = value;
        rear++;
        size++;
    }

    int pop()
    {
        if(size == 0){
            return -1; // Queue Underflow
        }
        int ans = arr[front];
        front++;
        size--;
        return ans;
    }
}
```

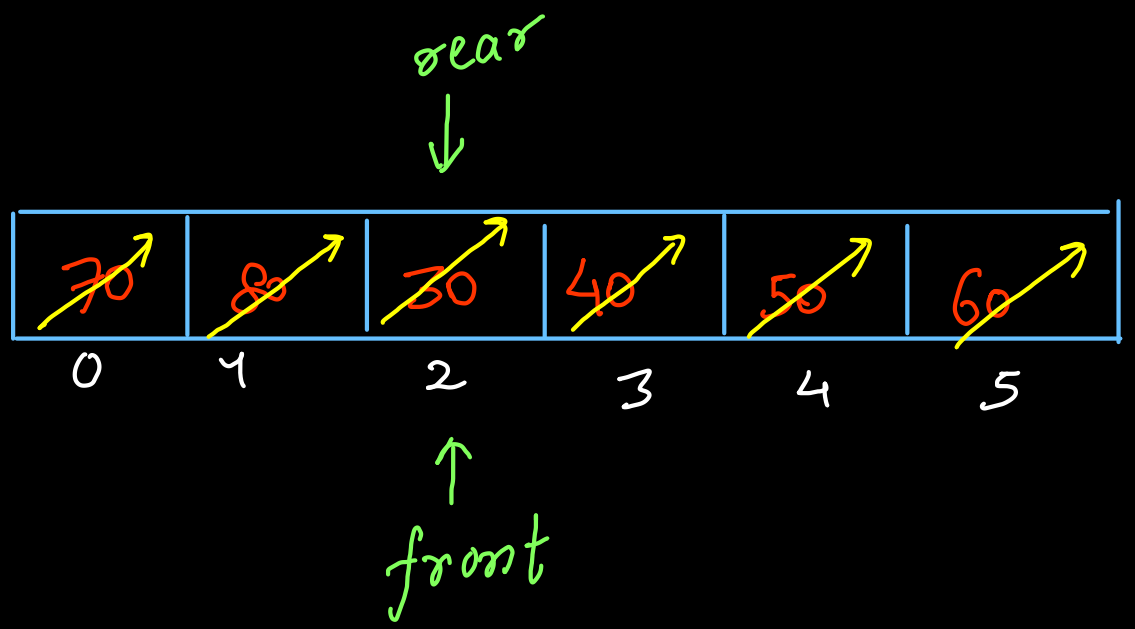
O(1)

O(1)

but wastage of space might occur

Queue using Circular Array

LC G22



$ptr = (ptr+1)\%n$

0	1
1	2
2	3
3	4
4	5
5	0

size = 0 1 2 3 4 5 6

Capacity = 6

add(10)	remove()
add(20)	remove()
add(30)	remove()
add(40)	remove()
add(50)	remove()
add(60)	remove
add(70)	remove()
add(80)	remove()
add(90)	remove()
⇓	⇓
overflow	underflow

```java
class MyCircularQueue {
    int[] arr;
    int front = 0, rear = 0, size = 0;

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

    public boolean enQueue(int value) {
        if(isFull()) return false; // Queue Overflow

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

    public boolean deQueue() {
        if(isEmpty()) return false; // Queue Underflow

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

→ O(1)

→ O(1)

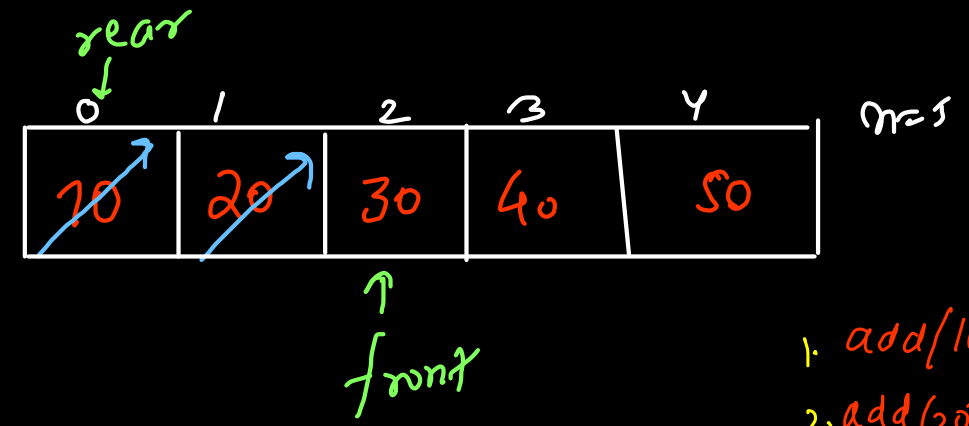
```java
public boolean isEmpty() {
    return (size == 0);
}

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

→ O(1)

→ O(1)

4-1%5
4 → 3
3-1%5
3 → 2
2-1%5
2 → 1
1-1%5
1 → 0
0-1%5
0 → 4

(rear-1+n) % n



1. add(10)
2. add(20)
3. add(30)
4. add(40)
5. add(50)
6. remove()
7. remove()

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

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

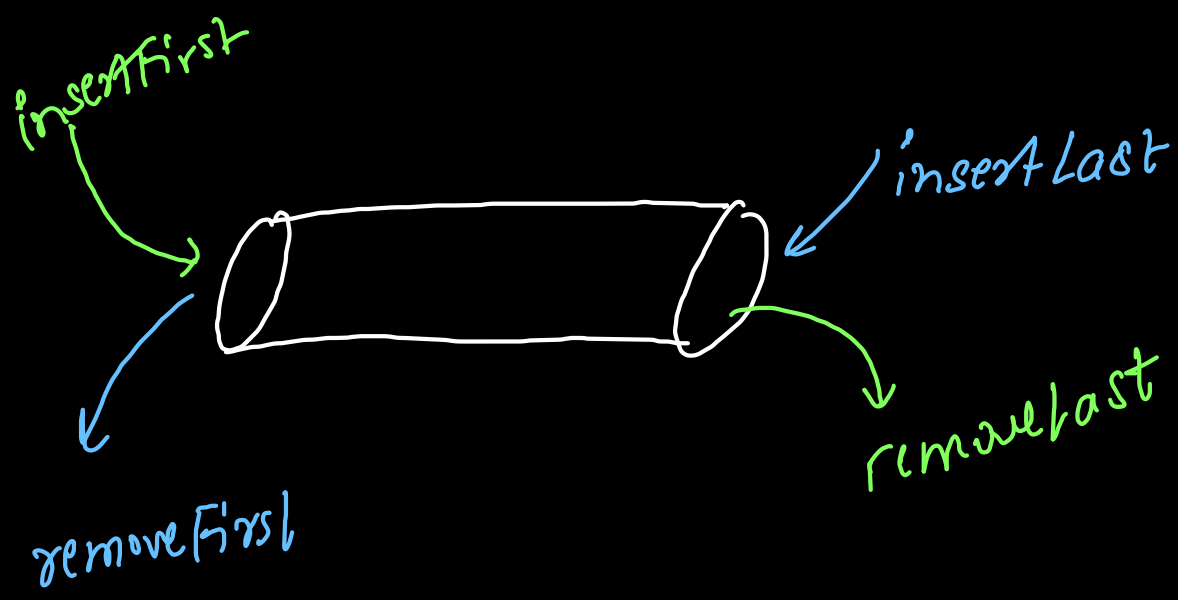
→ O(1)

↳ O(1)

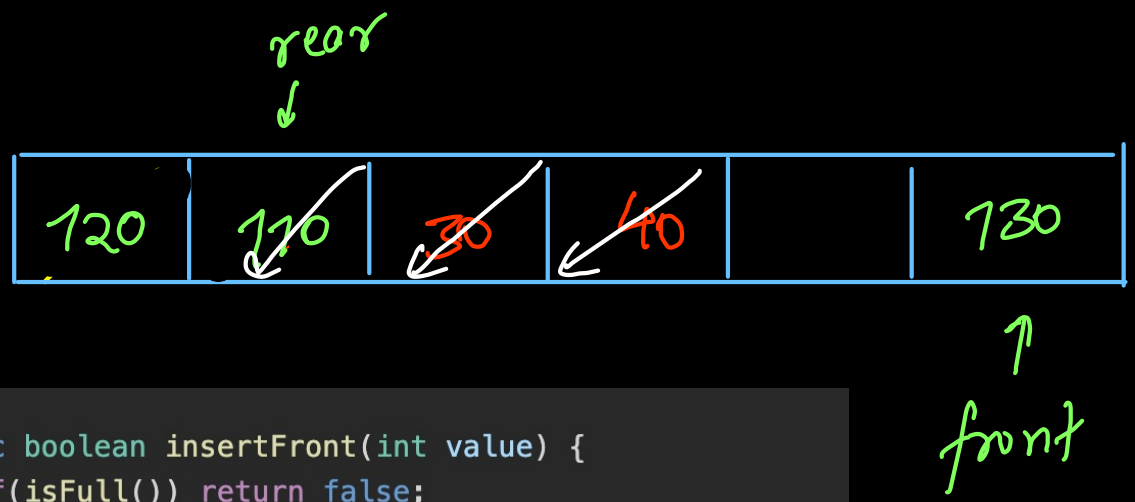
Design Circular Deque

Leetcode 641

→ Doubly ended queue



→ Stack




```java
public boolean insertFront(int value) {
    if(isFull()) return false;

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

O(1)

```java
public boolean deleteLast() {
    if(isEmpty()) return false;

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

O(1)

rear++
- addLast(10)
- addLast(20)
- addLast(30)
- addLast(40)

front++
- removeFirst()
- removeFirst()

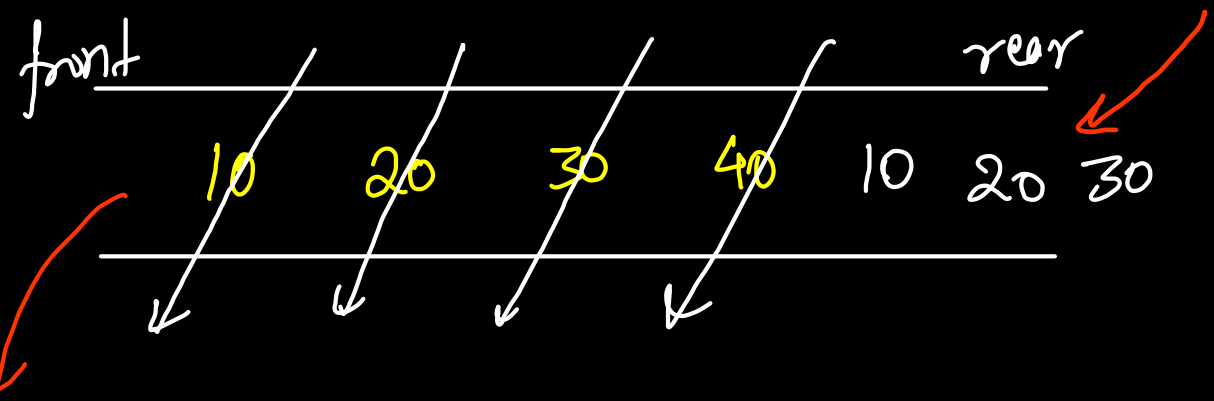
front--
- addFirst(110)
- addFirst(120)
- addFirst(130)

rear--
- removeLast()
- removeLast()
- removeLast

Implement Stack using Queue

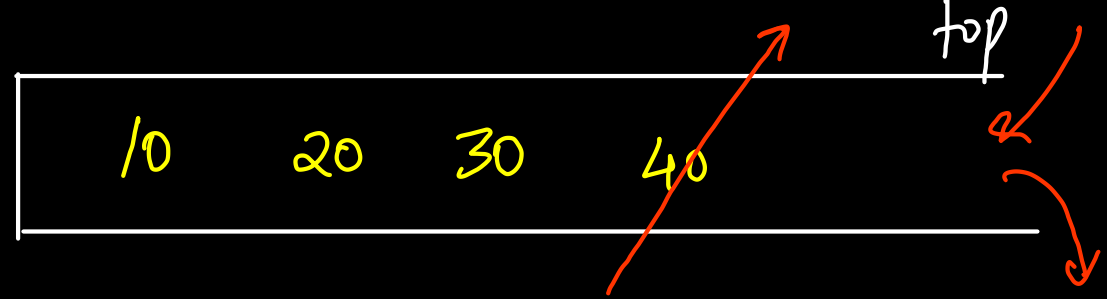
real:

Queue (FIFO)



logic:

Stack (LIFO)



add(10)

add(20)

add(30)

add(40)

peek() n pop + n push

remove() n pop + (n-1) push

Approach 1

Push Efficient → O(1)

Pop Heavy → O(n)

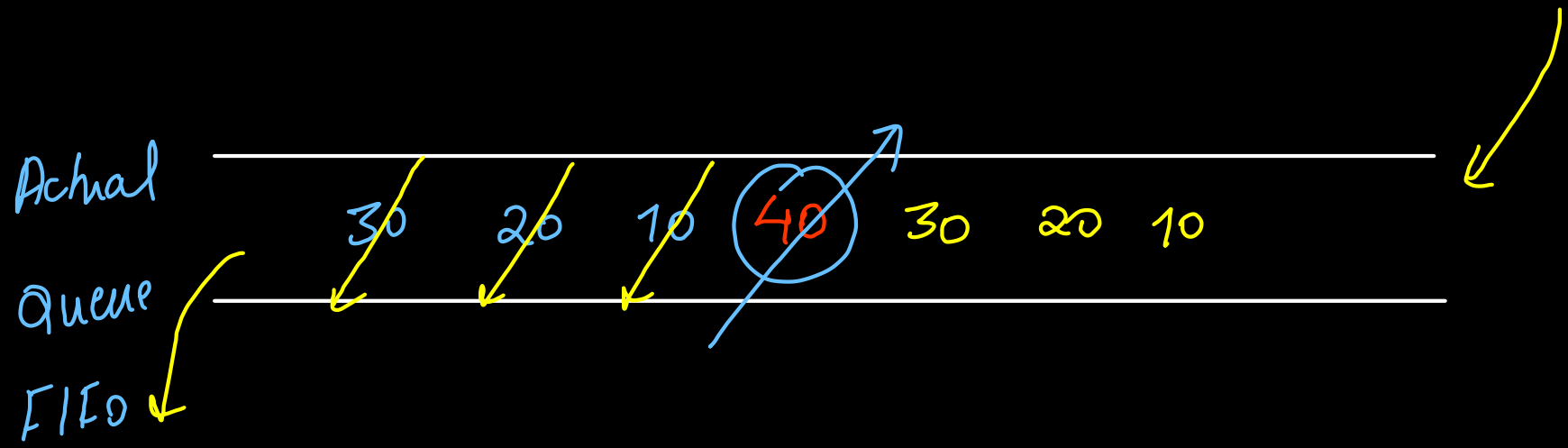
```java
class MyStack {
    Queue<Integer> q = new ArrayDeque<>();

    public void push(int x) {
        q.add(x);
    }

    public int pop() {
        for(int idx = 1; idx < q.size(); idx++){
            q.add(q.remove());
        }
        return q.remove();
    }

    public int top() {
        for(int idx = 1; idx < q.size(); idx++){
            q.add(q.remove());
        }
        int ans = q.remove();
        q.add(ans);
        return ans;
    }

    public boolean empty() {
        return q.size() == 0;
    }
}
```

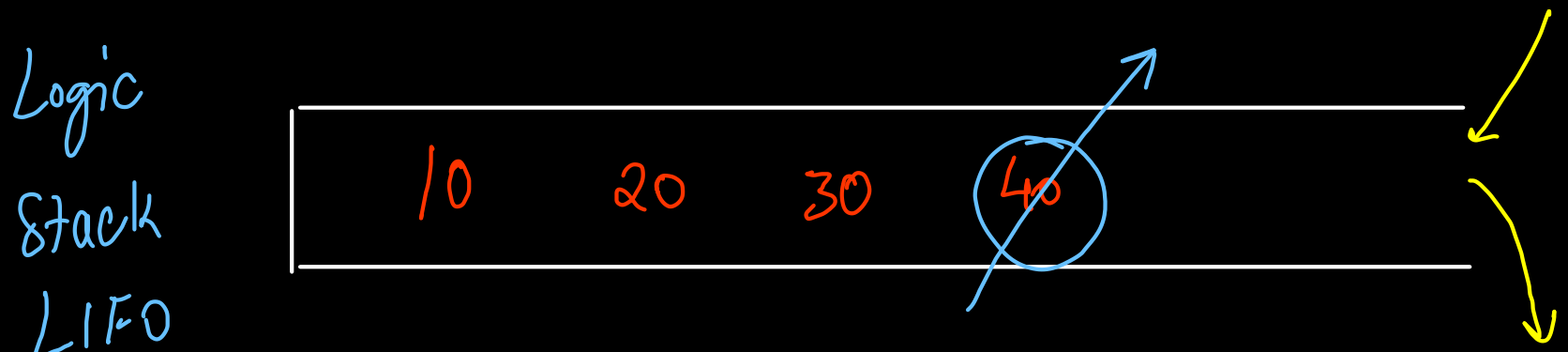
O(1)

O(n)

O(n)

O(1)





add(10)

add(20)

add(30)

add(40)

peek()

remove()

Approach 2

Push Heavy → O(n)

Pop Efficient → O(1)

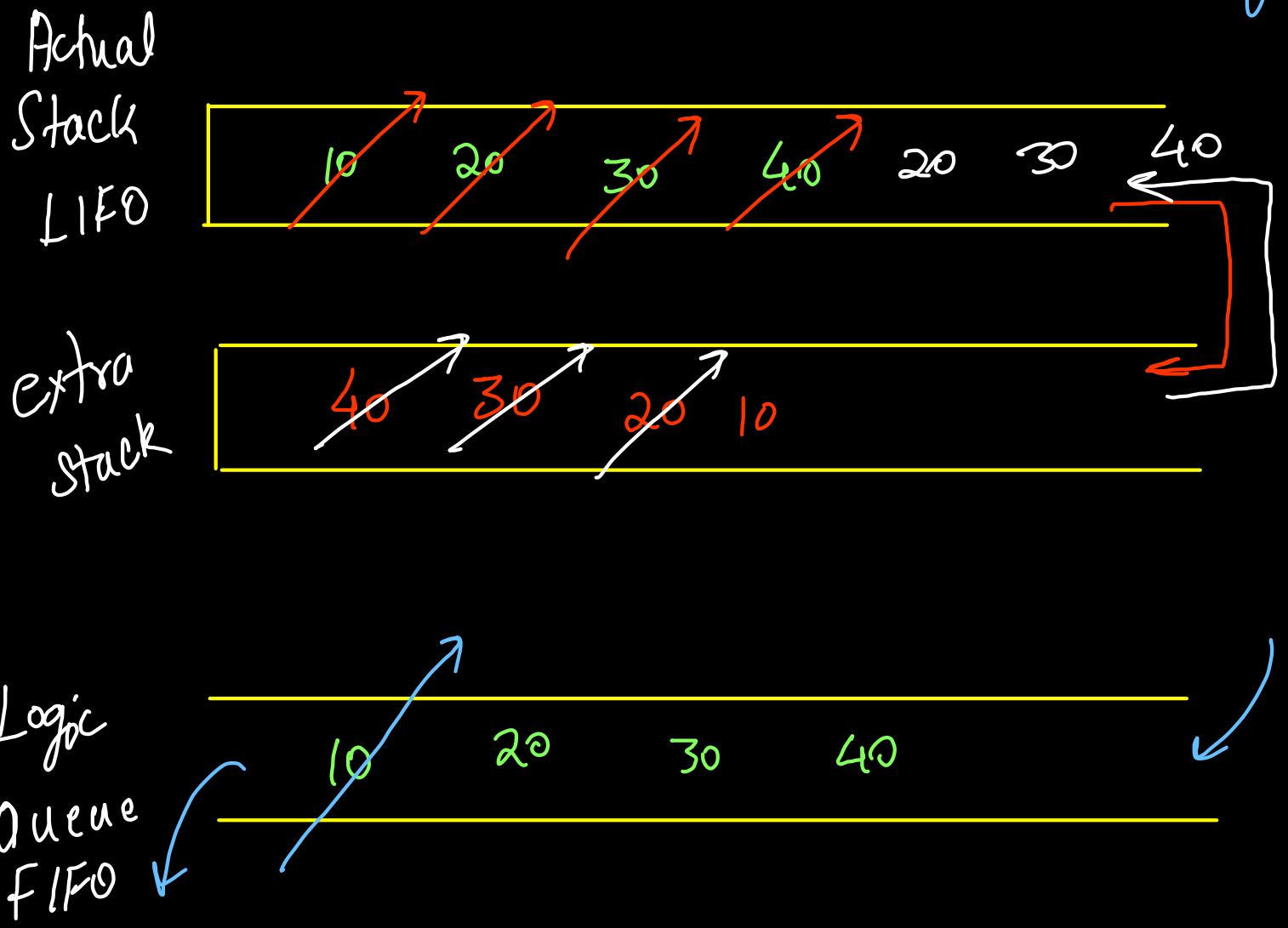
```
class MyStack {
    Queue<Integer> q = new ArrayDeque<>();

    public void push(int x) {
        q.add(x);
        for(int idx = 1; idx < q.size(); idx++){
            q.add(q.remove());
        }
    }

    public int pop() {
        return q.remove();
    }

    public int top() {
        return q.peek();
    }

    public boolean empty() {
        return q.size() == 0;
    }
}
```

O(n)

O(1)

O(1)

Implement Queue using Stack



add(10)

add(20)

add(30)

add(40)

peek()

remove()

Approach 1

Push Efficient → O(1)

Pop Heavy → O(n)

```java
class MyQueue {
    Stack<Integer> actual = new Stack<>();
    Stack<Integer> extra = new Stack<>();

    public void push(int x) {
        actual.push(x);
    }

    public int pop() {
        while(actual.size() > 0)
            extra.push(actual.pop());
        int ans = extra.pop();
        while(extra.size() > 0)
            actual.push(extra.pop());
        return ans;
    }

    public int peek() {
        while(actual.size() > 0)
            extra.push(actual.pop());
        int ans = extra.peek();
        while(extra.size() > 0)
            actual.push(extra.pop());
        return ans;
    }

    public boolean empty() {
        return actual.size() == 0;
    }
}
```

→ O(1)

→ O(n)

→ O(n)

→ O(1)

Approach 2 Push → O(n) Pop → O(1)

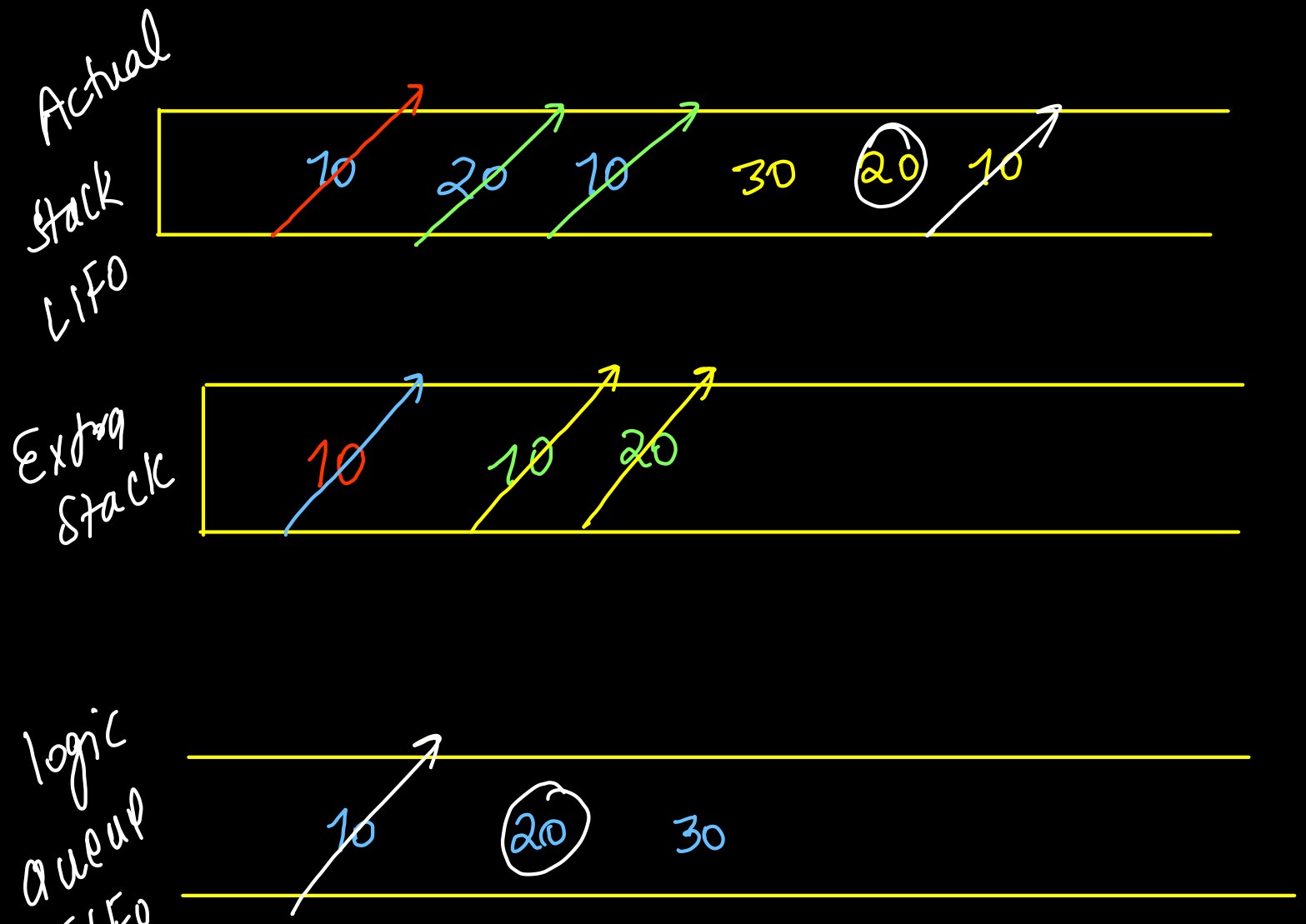
```java
class MyQueue {
    Stack<Integer> actual = new Stack<>();
    Stack<Integer> extra = new Stack<>();

    public void push(int x) {
        while(actual.size() > 0){
            extra.push(actual.pop());
        }
        actual.push(x);
        while(extra.size() > 0){
            actual.push(extra.pop());
        }
    }

    public int pop() {
        return actual.pop();
    }

    public int peek() {
        return actual.peek();
    }

    public boolean empty() {
        return actual.size() == 0;
    }
}
```

O(n)

O(1)

O(1)

O(1)



Approach 2

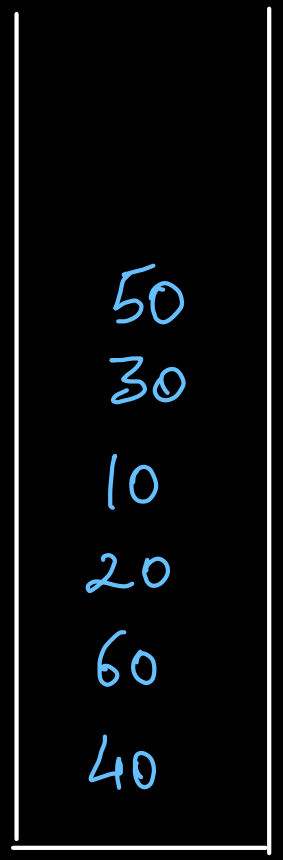
Push Heavy → O(n)

Pop Efficient → O(1)

add(10, 20, 30)

Design Minimum Stack

Leetcode 155



add(40)

top() = 40 min() = 40

add(60)

top() = 60 min() = 40

add(20)

top() = 20 min() = 20

add(10)

top() = 10 min() = 10

add(30)

top() = 30 min() = 10

add(50)

top() = 50 min() = 10

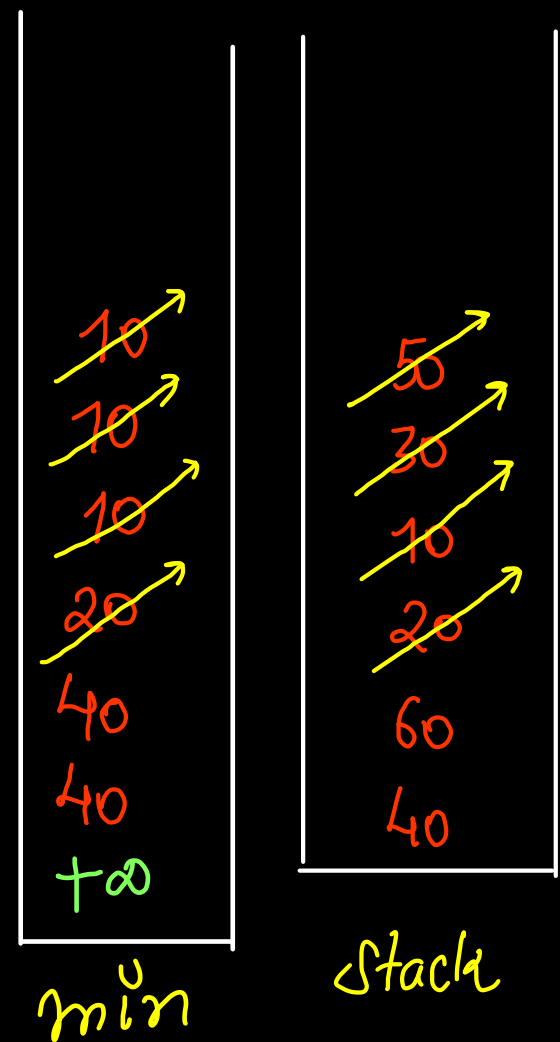
```java
class MinStack {
    Stack<Integer> min, stk;
    public MinStack() {
        min = new Stack<>();
        stk = new Stack<>();
        min.push(Integer.MAX_VALUE);
    }

    public void push(int val) {
        stk.push(val);
        int newMin = Math.min(min.peek(), val);
        min.push(newMin);
    }

    public void pop() {
        stk.pop();
        min.pop();
    }

    public int top() { return stk.peek(); }
    public int getMin() { return min.peek(); }
}
```

O(1) push

O(1) pop

O(1) top

O(1) getMin



add(40)
top() = 40 min() = 40

add(60)
top() = 60 min() = 40

add(20)
top() = 20 min() = 20

add(10)
top() = 10 min() = 10

add(30)
top() = 30 min() = 10

add(50)
top() = 50 min() = 10

ExtraSpace ⇒ O(n)
minStack

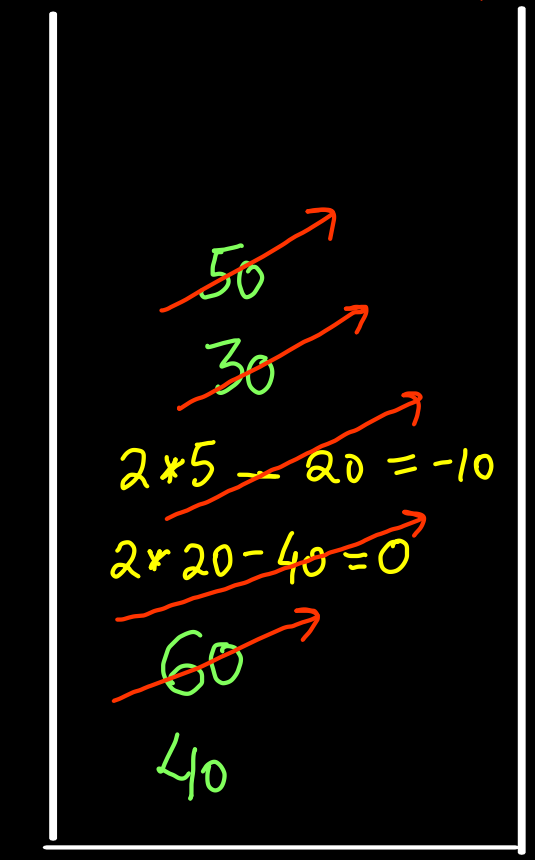
val < min

Encrypted

$= val + (val - min)$ (-ve)

$= 2 * val - min$

decrypt = 2*min - top



Stack <Long>

5
20
40
min

2*5 - (-10)

2*20 - 0

✓ add(40)
top() = 40 min() = 40

✓ add(60)
top() = 60 min() = 40

✗ add(20)
top() = 20 min() = 20

✗ add(5)
top() = 5 min() = 5

✗ add(30)
top() = 30 min() = 10

✗ add(50)
top() = 50 min() = 10

~~add(20)~~

Extra space ⇒ O(1)

```java
class MinStack {
    Stack<Long> stk = new Stack<>();
    long min = Integer.MAX_VALUE;

    public void push(int val) {
        if(val >= min){
            stk.push(1l * val);
        } else {
            stk.push(2l * val - min);
            min = val;
        }
    }

    public void pop() {
        long top = stk.pop();
        if(top >= min) return;
        min = 2l * min - top;
    }

    public int top() {
        long top = stk.peek();
        if(top < min)
            return (int)min;
        return (int)top;
    }

    public int getMin() {
        return (int)min;
    }
}
```

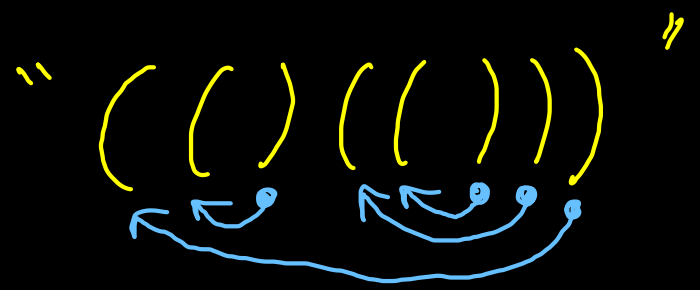
O(1)

O(1)

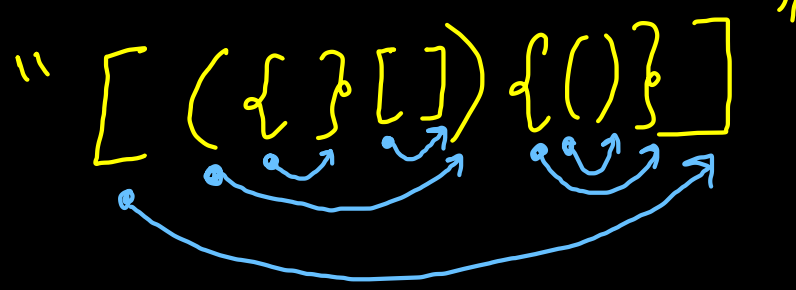
O(1)

O(1)

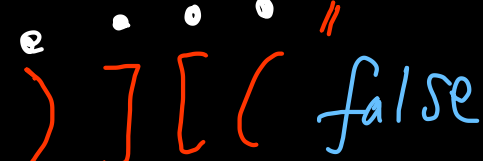
Balanced/Valid Parantheses leetcode 20



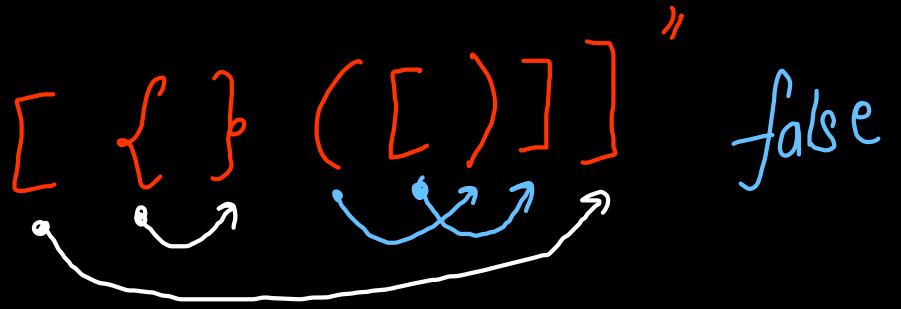
true



true

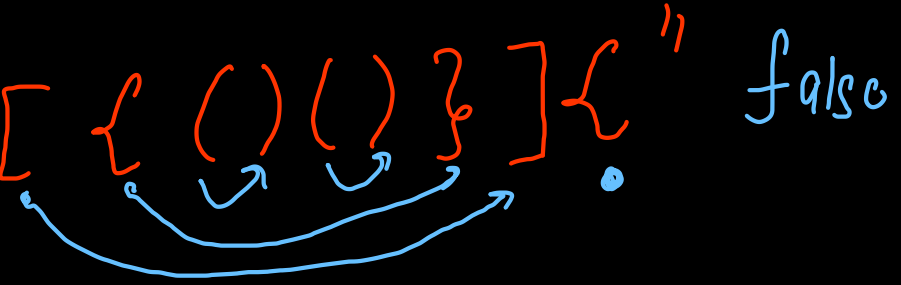


false

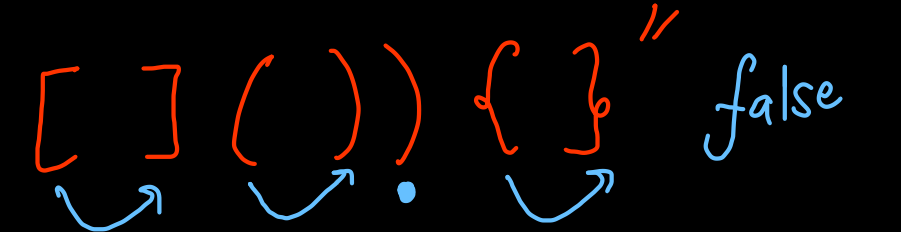


false

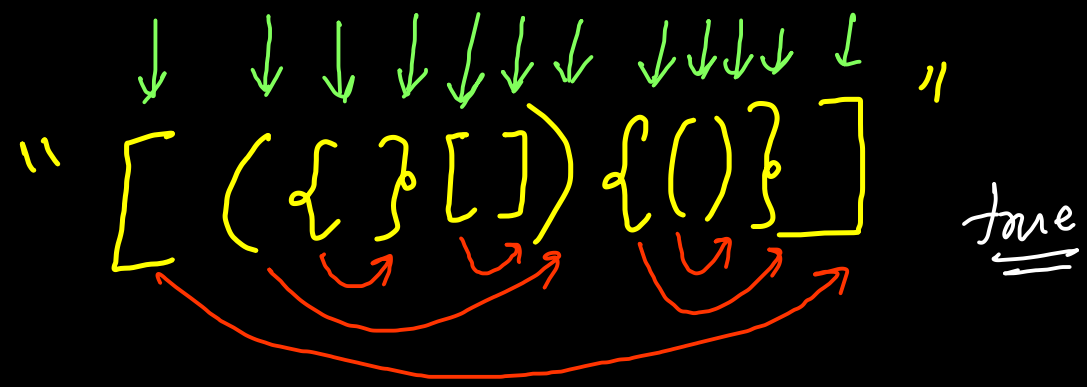
- → open == closed
- → order matching
- → type matching

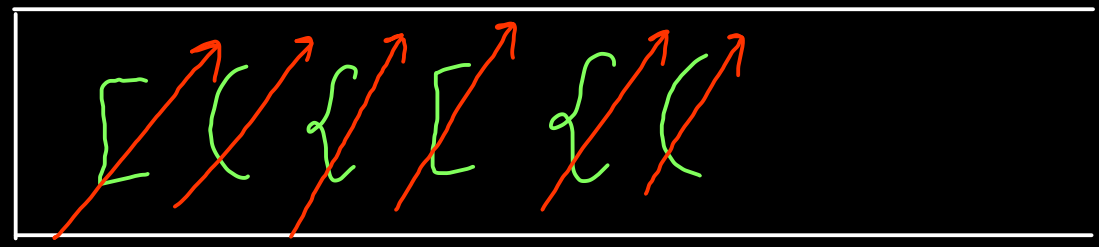


false

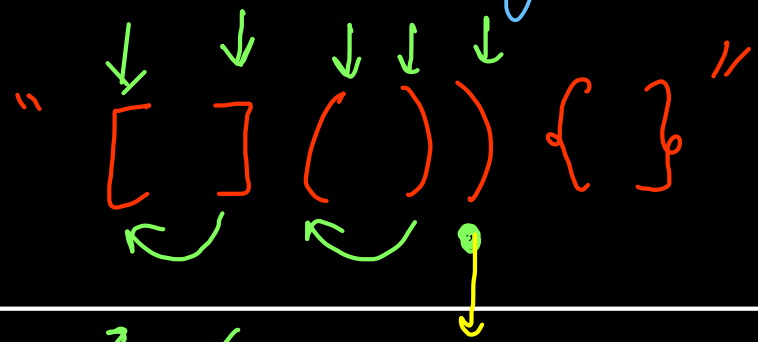


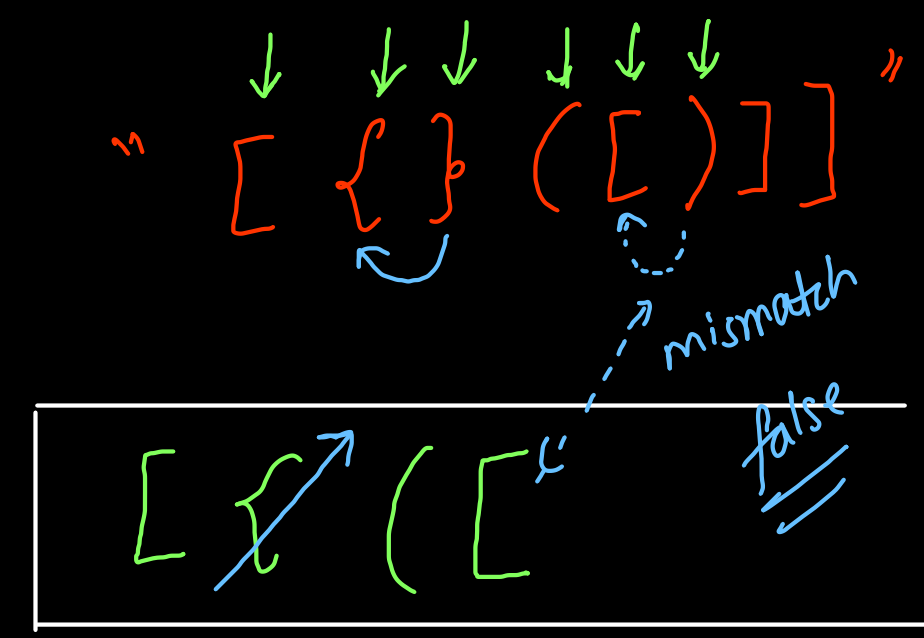
false

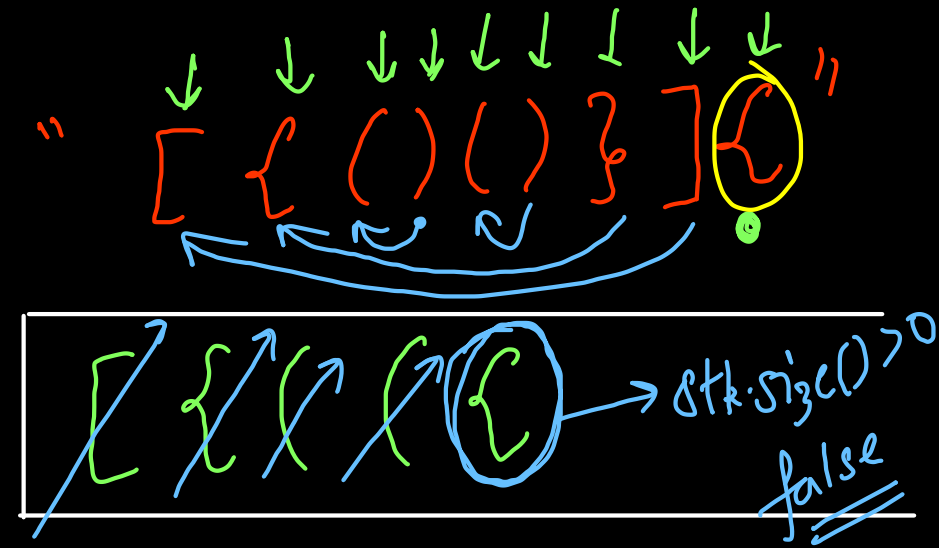




unbalanced opening braces





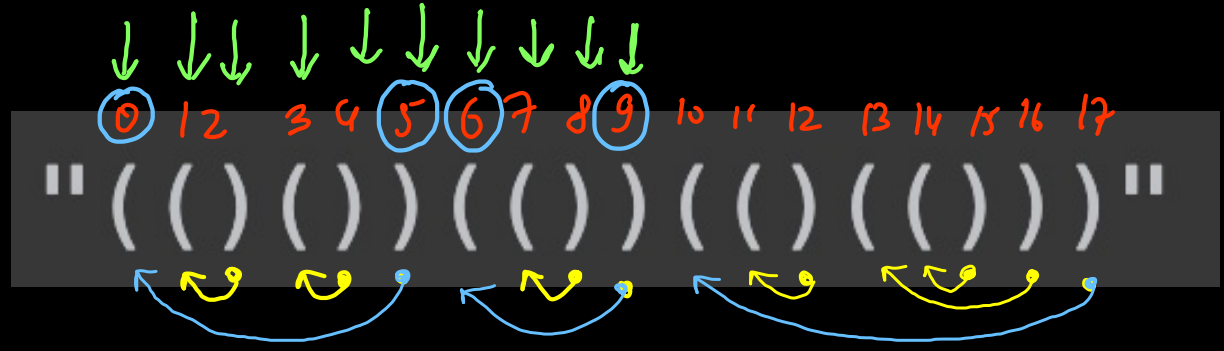


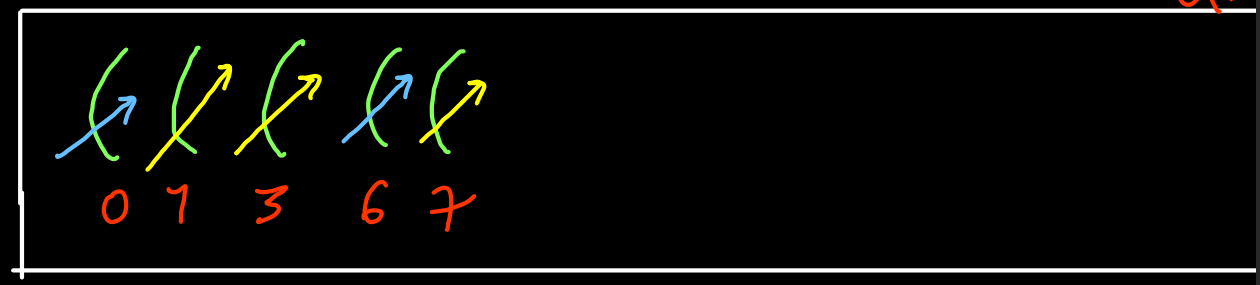
```java
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stk = new Stack<>();

        // for(char ch: s.toCharArray())
        for(int idx = 0; idx < s.length(); idx++){
            char ch = s.charAt(idx);

            if(ch == '(' || ch == '[' || ch == '{'){
                stk.push(ch);
            } else {
                if(stk.size() == 0) return false; // extra closing
                if(ch == ')' && stk.peek() == '(') stk.pop();
                else if(ch == '}' && stk.peek() == '{') stk.pop();
                else if(ch == ']' && stk.peek() == '[') stk.pop();
                else return false; // type or order mismatch
            }
        }

        return (stk.size() == 0); // stk.size() > 0: extra open
    }
}
```

Time ⇒ O(N)

Space ⇒ O(N) stack

Remove Outermost Parantheses





O(N)

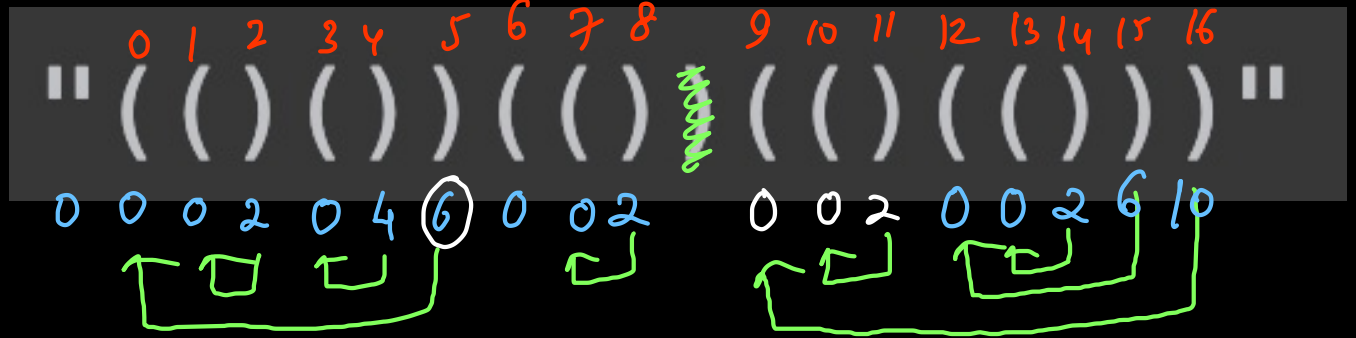
```java
public String removeOuterParentheses(String s) {
    boolean[] isOuter = new boolean[s.length()];

    Stack<Integer> stk = new Stack<>();
    for(int idx = 0; idx < s.length(); idx++){
        if(s.charAt(idx) == '('){
            stk.push(idx);
        }
        else {
            int open = stk.pop();
            if(stk.size() == 0){
                // remove outermost
                isOuter[idx] = isOuter[open] = true;
            }
        }
    }

    StringBuilder res = new StringBuilder();
    for(int idx = 0; idx < s.length(); idx++){
        if(isOuter[idx] == false){
            res.append(s.charAt(idx));
        }
    }
    return res.toString();
}
```

↳ O(1)

time = O(n)

space = O(n)

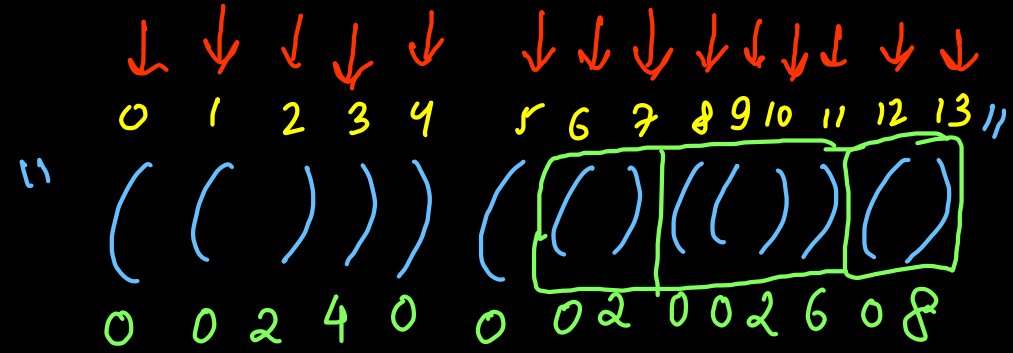
Longest Valid Parantheses



```java
class Solution {
    public int longestValidParentheses(String s) {
        int[] res = new int[s.length()];
        Stack<Integer> stk = new Stack<>();

        int max = 0;
        for(int idx = 0; idx < s.length(); idx++){
            if(s.charAt(idx) == '(' ){
                stk.push(idx);
            } else {
                if(stk.size() == 0) continue;
                int open = stk.pop();
                res[idx] += (idx - open + 1);
                if(open > 0) res[idx] += res[open - 1];
                max = Math.max(max, res[idx]);
            }
        }

        return max;
    }
}
```

Time ⇒ O(n)

Space ⇒ O(n)



0, 1, 5, 6, 8, 9, 12

max = 0 2 4 8

Monotonic stack

Next Greater Element

arr	200	30	10	20	70	50	60	90	10	160	130	130	150
	-1	70	20	70	90	60	90	160	160	-1	150	150	-1

1) Right Side
2) Strictly Greater
3) Nearest

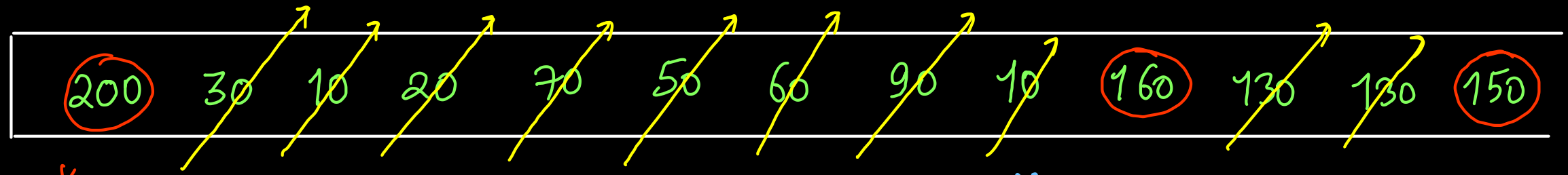
Brute force :→ Nested loops

```
for (i: 0 to N-1)
    for (j: i+1 to N-1)
        if(arr[j] > arr[i])
            nge[i] = arr[j];
            break;
```

Time ⇒ $O(n^2)$ → TLE

Space ⇒ $O(1)$

arr:	200	30	10	20	70	50	60	90	10	160	130	130	150
nge:	-1	70	20	70	90	60	90	160	160	-1	150	150	-1



Stack
NGE
yet to
be discovered

monotonically
decreasing

```
Stack<Integer> stk;
for(int idx:0 to N-1){
    while(stk.size() > 0 && arr[top] < arr[idx]
        nge[pop] = arr[idx];
    stk.push(idx);
}
```

Approach 1)

Left to Right

Data → NGE yet to be discovered

```
public static long[] nextLargerElement(long[] arr, int n)
{
    Stack<Integer> stk = new Stack<>();
    long[] nge = new long[n];
    Arrays.fill(nge, -1);

    for(int idx = 0; idx < n; idx++){
        while(stk.size() > 0 && arr[idx] > arr[stk.peek()]){
            nge[stk.pop()] = arr[idx];
        }
        stk.push(idx);
    }

    return nge;
}
```

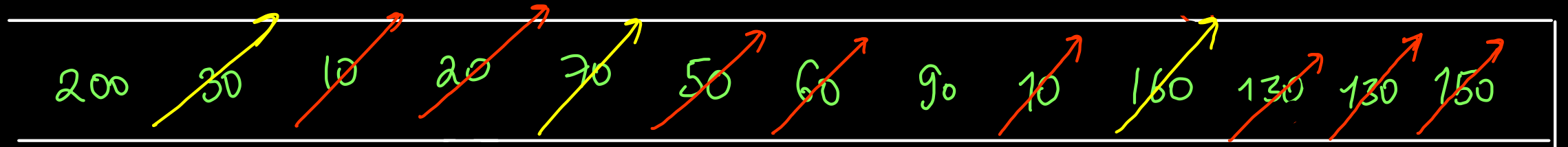
→ O(n) push + → O(n) pop ← independent

Time → O(N)

Space → O(N)

extra space or auxiliary datastr ⇓ Stack

200	30	10	20	70	50	60	90	10	160	130	130	150
-1	70	20	70	90	60	90	160	160	-1	150	150	-1




```
Stack<Integer>
for(idx: n-1 to 0){
    while(stk.size()>0 && arr[top] <= arr[idx]
        stk.pop();
    nge[idx] = (size>0)? arr[stk.peek()] : -1;
    stk.push(idx);
}
```

Pop smaller or equal

Find NGE of curr →

Push curr →

Stack {Potential answers}

monotonic decreasing

Approach 2)

Right to Left

Data → Potential NGE answers

Time ⇒ O(n)

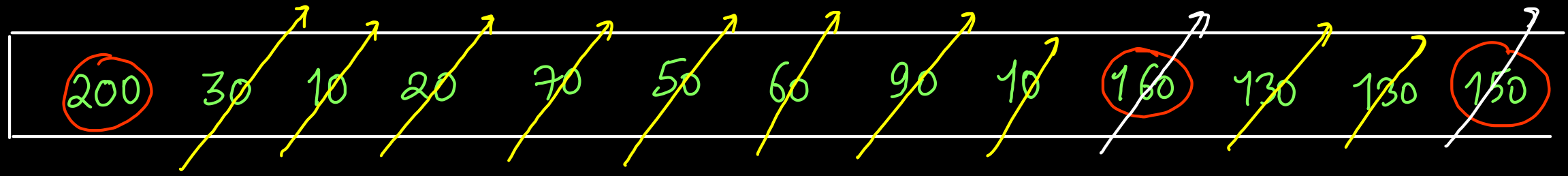
Space ⇒ O(n) extra (stack)

```java
public static long[] nextLargerElement(long[] arr, int n)
{
    Stack<Integer> stk = new Stack<>();
    long[] res = new long[n];

    for(int idx = n - 1; idx >= 0; idx--){
        while(stk.size() > 0 && arr[stk.peek()] <= arr[idx]){
            stk.pop();
        }
        res[idx] = (stk.size() > 0) ? arr[stk.peek()] : -1;
        stk.push(idx);
    }

    return res;
}
```

→ O(n) push + O(n) pop

NGE - Circular Array

200	30	10	20	70	50	60	90	10	160	130	130	150
-1	70	20	70	90	60	90	160	160	~~-1~~ 200	150	150	~~-1~~ 200



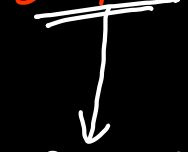
```java
public int[] nextGreaterElements(int[] arr) {
    int n = arr.length;
    int[] nge = new int[n];
    Arrays.fill(nge, -1);
    Stack<Integer> stk = new Stack<>();

    for(int idx = 0; idx < n; idx++){
        while(stk.size() > 0 && arr[idx] > arr[stk.peek()]){
            nge[stk.pop()] = arr[idx];
        }
        stk.push(idx);
    }

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

    return nge;
}
```

Time ⇒ O(n)

Space ⇒ O(n)

→ Due to circular array

Online Algorithm



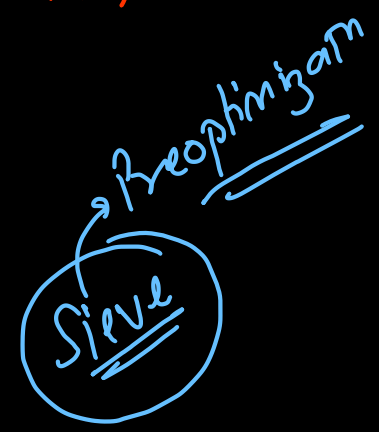
server

packets / discrete

Offline Algorithm

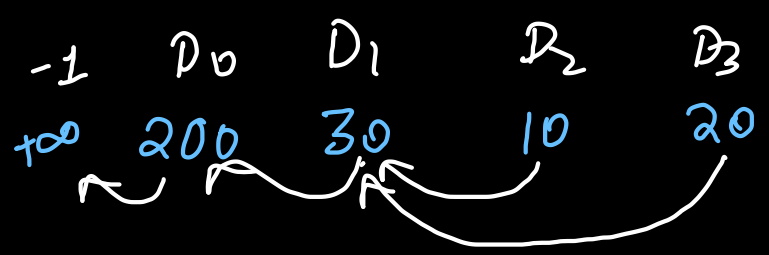
download

complete info



Online Stock span

Online Stock Span



D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
70	50	60	90	10	160	130	130	150				
1	1	1	2	4	1	2	7	1	9	1	2	3

Span = Maxm consecutive days in the left for which the current price is maxm = current index − next greater in the left index

				D_3	D_2	D_1	D_0
200	30	10	20	70	50	60	90
				3	1	1	1

potential answers

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

public int next(int price) {
    while(stk.size() > 0 && arr.get(stk.peek()) <= price){
        stk.pop();
    }

    int span = (stk.size() > 0) ? arr.size() - stk.peek()
                                : arr.size() + 1;

    stk.push(arr.size());
    arr.add(price);
    return span;
}
```

Stack : 2 1 0

arr : 50 60 90

size = ~~0~~ 1 2

N queries :→ O(N) total time

:→ N/N = O(1) avg

739. Daily Temperatures

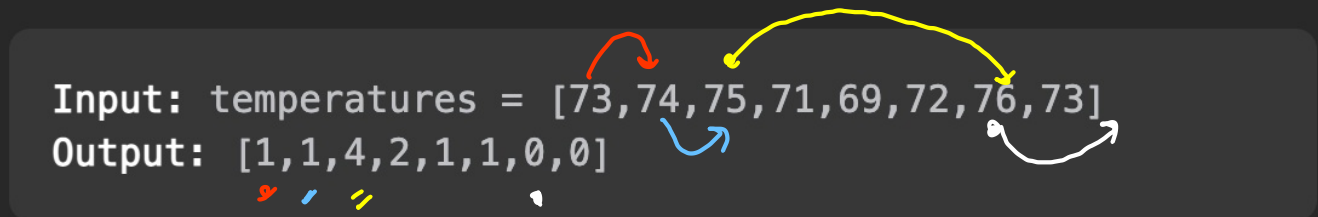
Hint

Medium 9.9K 227

Companies

Given an array of integers `temperatures` represents the daily temperatures, return *an array* `answer` *such that* `answer[i]` *is the number of days you have to wait after the* `i^th` *day to get a warmer temperature*. If there is no future day for which this is possible, keep `answer[i] == 0` instead.

Example 1:



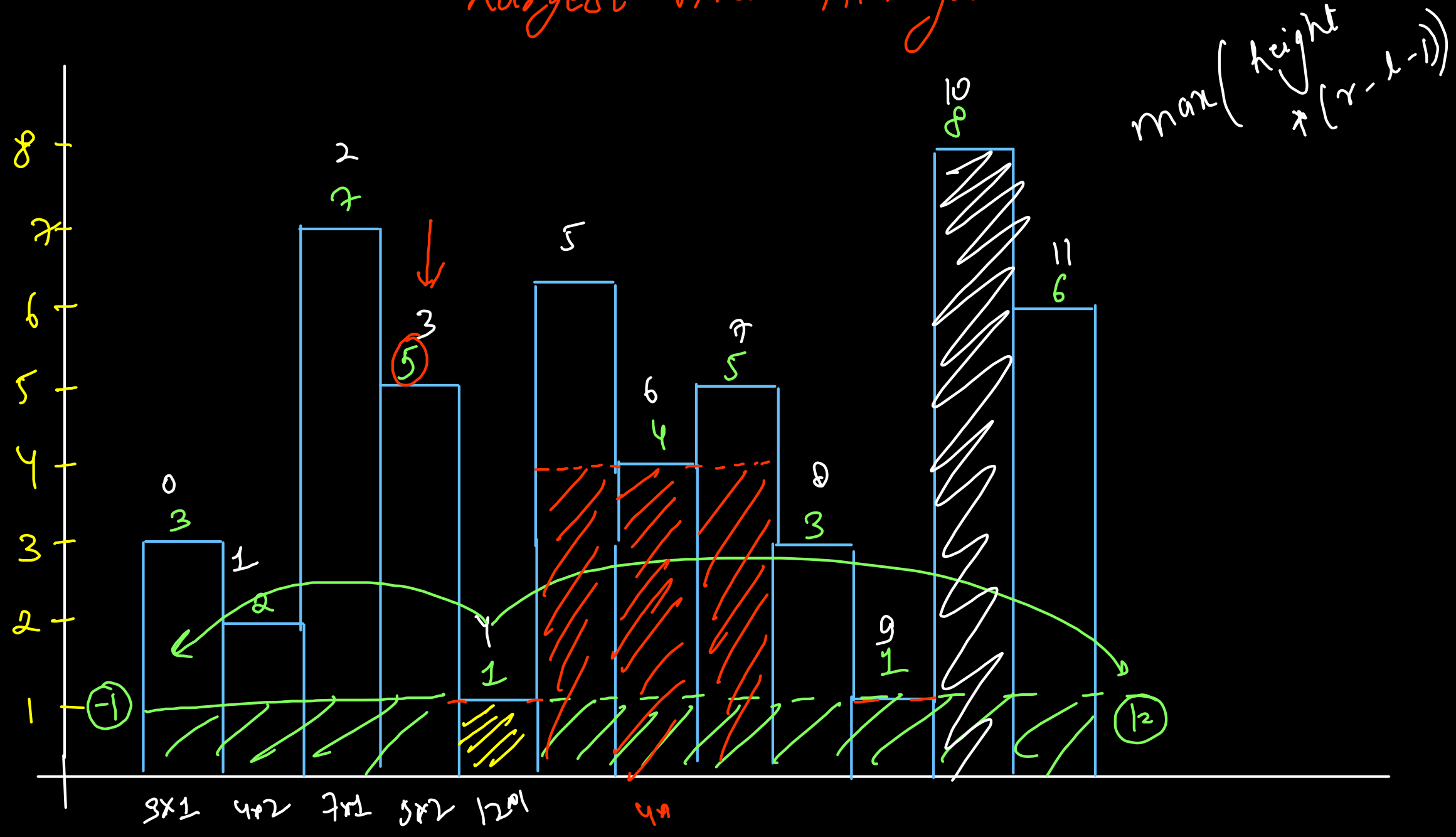
```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
```

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

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

        res[idx] = (stk.size() > 0) ? stk.peek() - idx : 0;
        stk.push(idx);
    }

    return res;
}
```

Time ⇒ O(n)

Space ⇒ O(n)

Largest Area Histogram



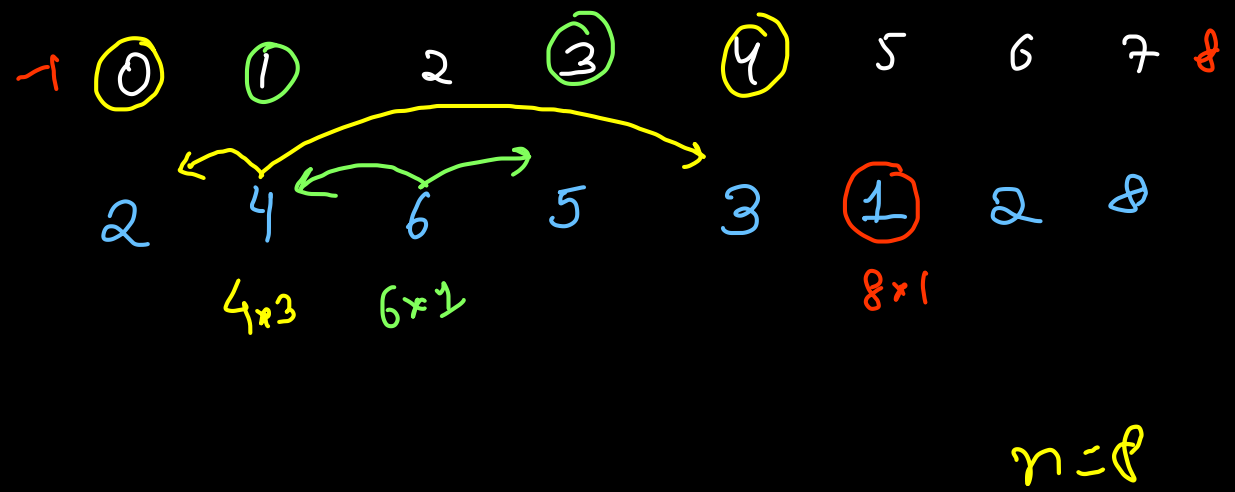
```java
public int[] rightNSE(int[] arr){
    int n = arr.length;
    Stack<Integer> stk = new Stack<>();
    int[] res = new int[n];

    for(int idx = n - 1; idx >= 0; idx--){
        while(stk.size() > 0 && arr[stk.peek()] >= arr[idx]){
            stk.pop();
        }
        res[idx] = (stk.size() > 0) ? stk.peek() : n;
        stk.push(idx);
    }

    return res;
}
```

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

    for(int idx = 0; idx < n; idx++){
        while(stk.size() > 0 && arr[stk.peek()] >= arr[idx]){
            stk.pop();
        }
        res[idx] = (stk.size() > 0) ? stk.peek() : -1;
        stk.push(idx);
    }

    return res;
}
```

```java
public int largestRectangleArea(int[] arr) {
    int[] left = leftNSE(arr);→O(n)
    int[] right = rightNSE(arr);→O(n)
    int max = 0;
    for(int idx = 0; idx < arr.length; idx++){
        int area = arr[idx] * (right[idx] - left[idx] - 1);
        max = Math.max(max, area);
    }
    return max;
}
```

O(n)

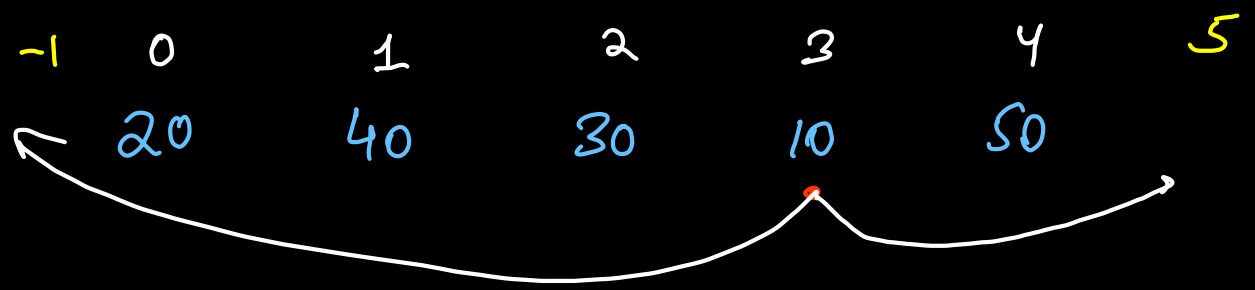
Time ⇒ O(n)
Space ⇒ O(n)



n = 8

Leetcode 907

Sum of Subarray minimum



Time $O(n^2)$ Approach (1)

Find all subarrays

Space $O(1)$

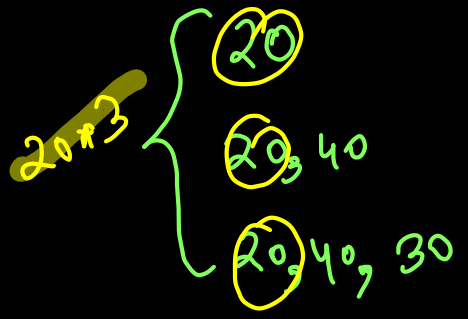
⇓

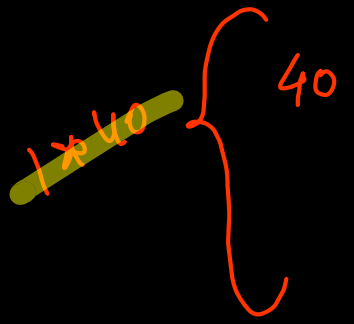
Find minm

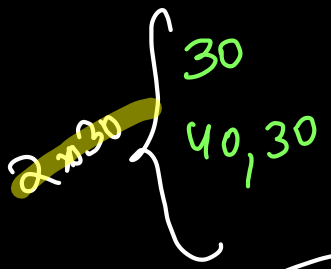
↓

Take sum of all minm

Brute force





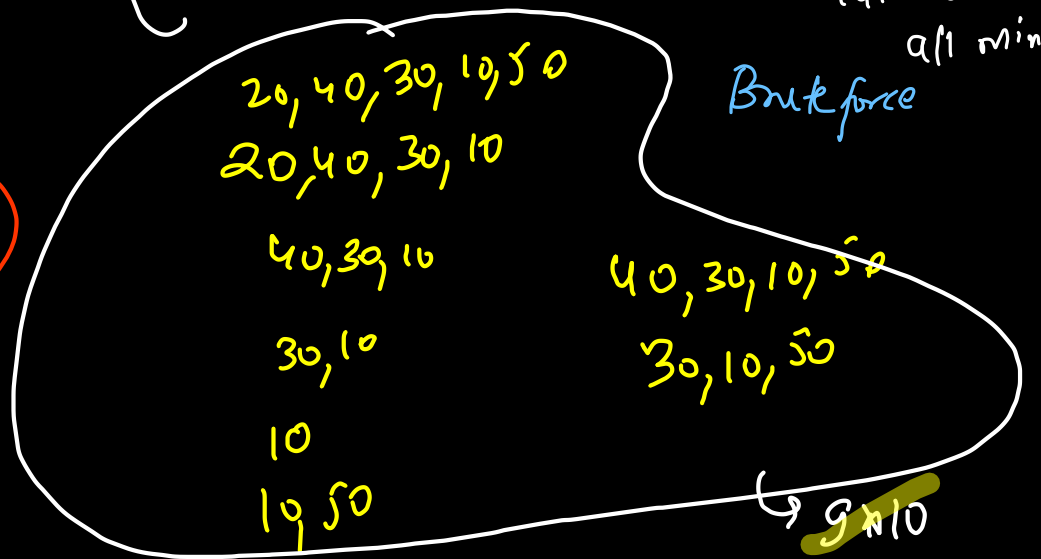


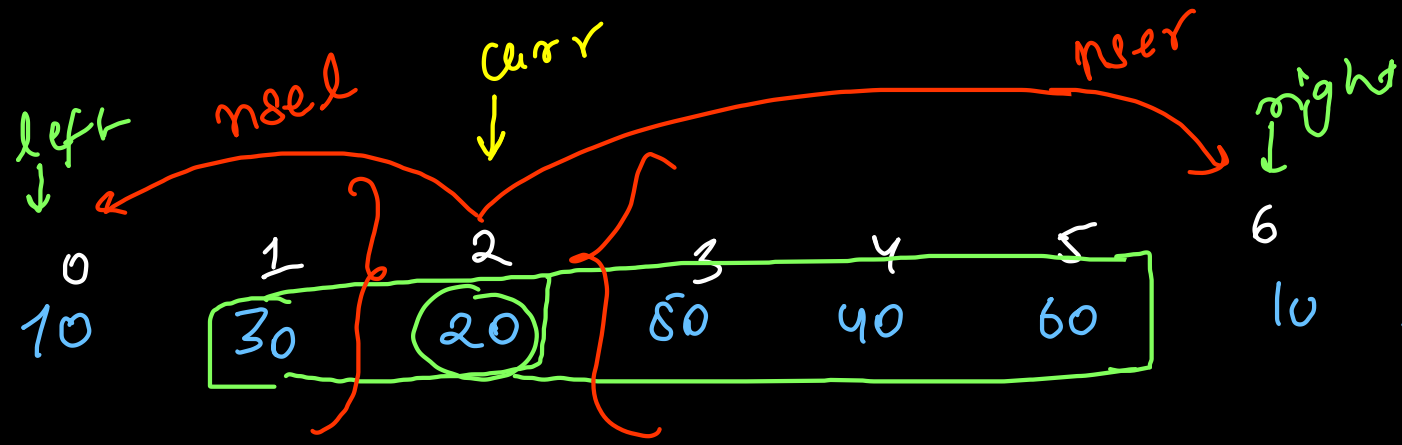
$(r-c)(c-l)$

$(3-0)*(0-(-1))$

$(2-1)(1-0)$

$= 1*1$





How many subarrays are there which are necessary to have

① curr index included

② start after left-(nse)

③ end before right(nse)

exclude subarrays ending before curr

~~30~~	20
30,20	20,50
30,20,50	20,50,40
30,20,50,40	20,50,40,60
30,20,50,40,60	

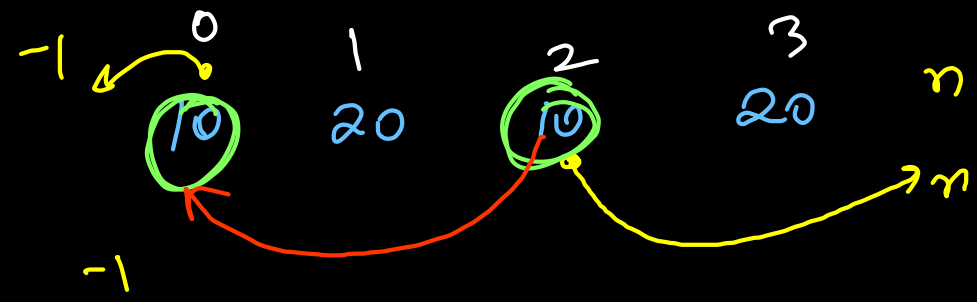
starting after curr

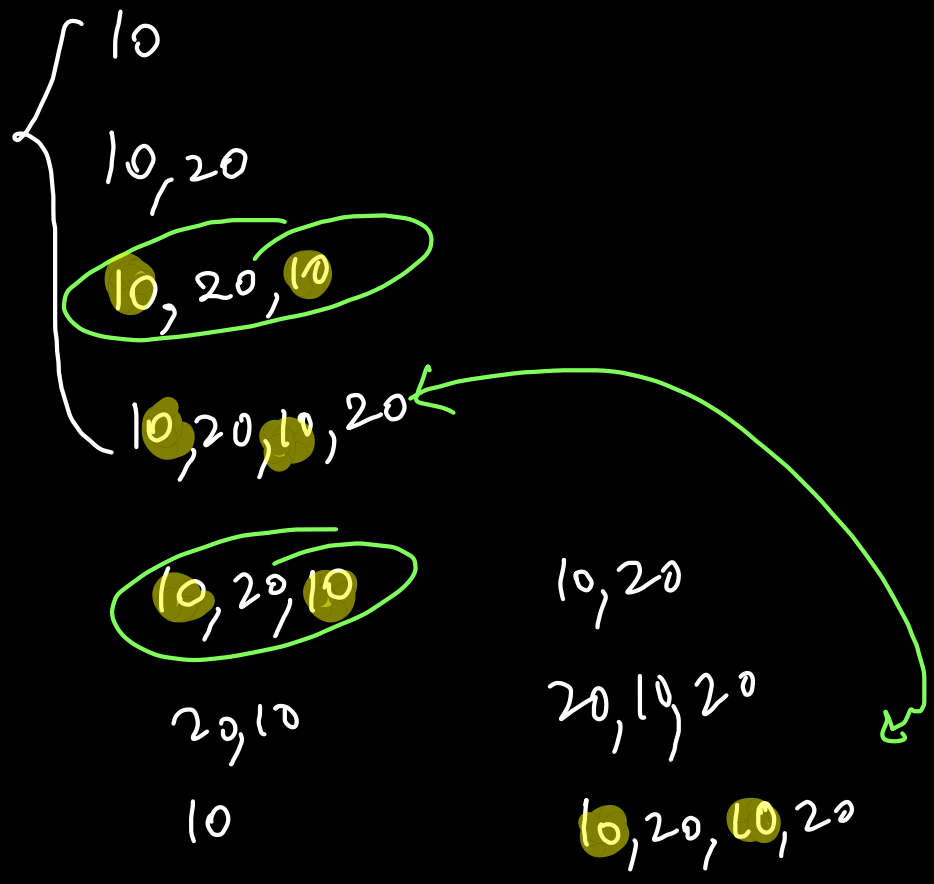
~~10~~	~~40~~	~~60~~
~~50~~,40	~~40~~,60	
~~50,40~~,60		

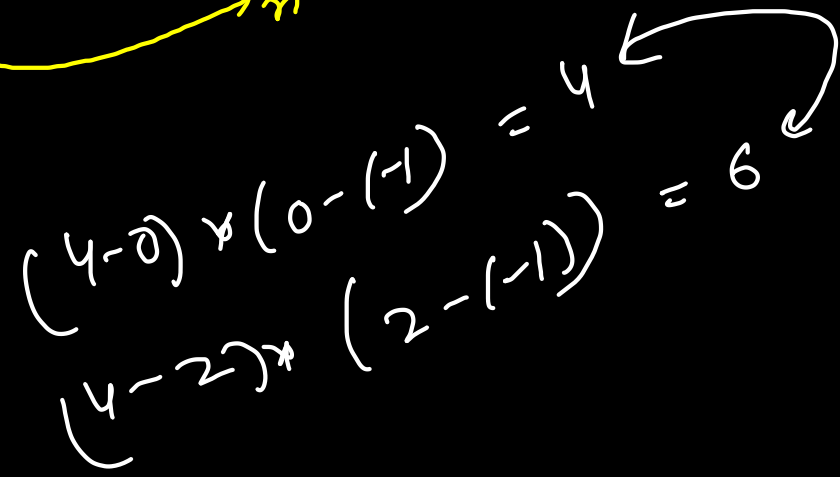
$$n = right - left - 1$$

$$\frac{(r-l-1)(r-l)}{2} - \frac{(curr.left-1)*(curr.left)}{2} - \frac{(right-curr-1)*(right-curr)}{2}$$

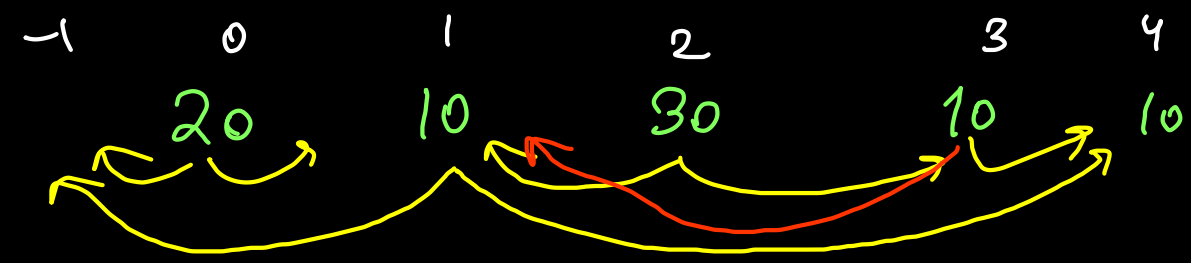
$$= (right - curr) * (curr.left)$$








```java
public int sumSubarrayMins(int[] nums) {
    int[] left = leftNSE(nums);
    int[] right = rightNSE(nums);
    long ans = 0;

    for(int curr = 0; curr < nums.length; curr++){
        int count = (right[curr] - curr) * (curr - left[curr]);
        ans += count * nums[curr];
    }
```



$20 * (1-0) * (0-(-1)) + 10 * (4-1) * (1-(-1)) + 30 * (3-2) * (2-1) + 10 * (4-3) * (3-1)$

{20} {20,10} {20,10,30} {20,10,30,10} {10} {10,30} {10,30,10} {30} {30,10} {10}

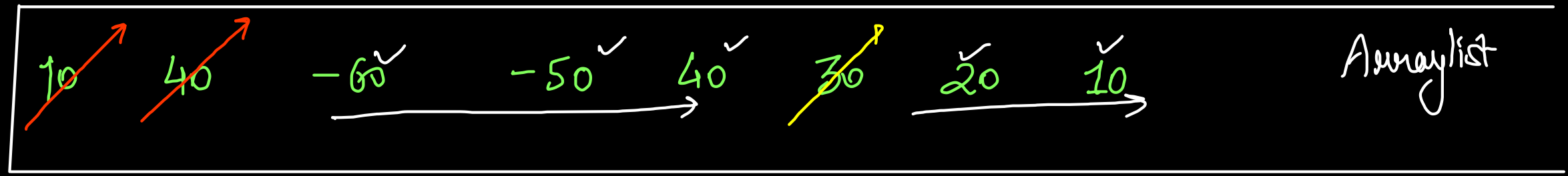
Corner case :→

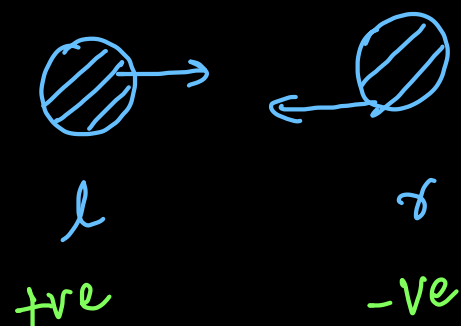
① modular arithmetic (answer can be very large)

② duplicate values


```java
public int sumSubarrayMins(int[] nums) {
    int[] left = leftNSE(nums);
    int[] right = rightNSE(nums);
    long ans = 0;

    for (int curr = 0; curr < nums.length; curr++) {
        long count = (right[curr] - curr) * (curr - left[curr]);
        ans = (ans + (count * nums[curr]) % 1000000007) % 1000000007;
    }

    return (int) ans;
}
```

Asteroid collision

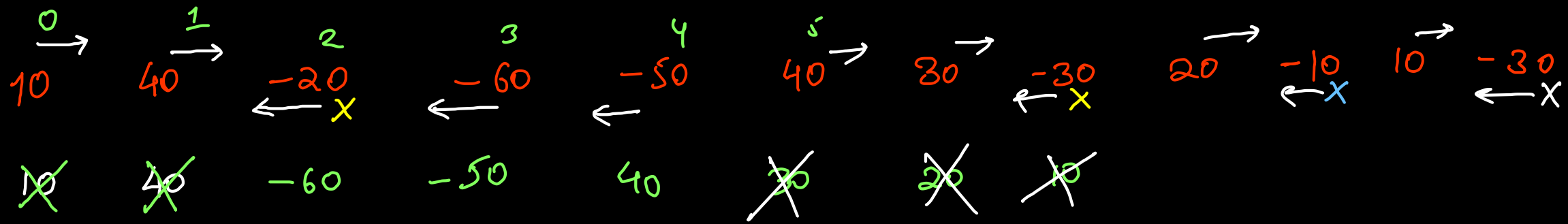




$|l| > |r|$ ⇒ destroy r ⇒ dont push(r)

$|l| = |r|$ ⇒ destroy l & r ⇒ pop left & dontpush(r)

$|l| < |r|$ ⇒ destroy l ⇒ pop(while) & push(r)




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

    for(int idx = 0; idx < asteroids.length; idx++){
        int right = asteroids[idx];

        if(right > 0) stk.add(right);
        else if(stk.size() == 0 || stk.get(stk.size() - 1) < 0){
                stk.add(right);
        }
        else if(stk.get(stk.size() - 1) < -right){
                stk.remove(stk.size() - 1);
                idx--; // to come back to same store
        }
        else if(stk.get(stk.size() - 1) == -right){
                stk.remove(stk.size() - 1);
        }
    }

    int[] res = new int[stk.size()];
    for(int i = 0; i < res.length; i++){
        res[i] = stk.get(i);
    }
    return res;
}
```

Time ⇒ O(n)

Space ⇒ O(n) {stack}

Celebrity Problem

	P_0	P_1	P_2	P_3	P_4
P_0	-	1	1	1	0
P_1	0	-	0	1	1
P_2	0	0	-	1	0
P_3	0	0	0	-	0
P_4	1	1	1	1	-

m[row][col]

1 ⇒ row knows col

0 ⇒ row doesn't col

Celebrity

① Everybody should him

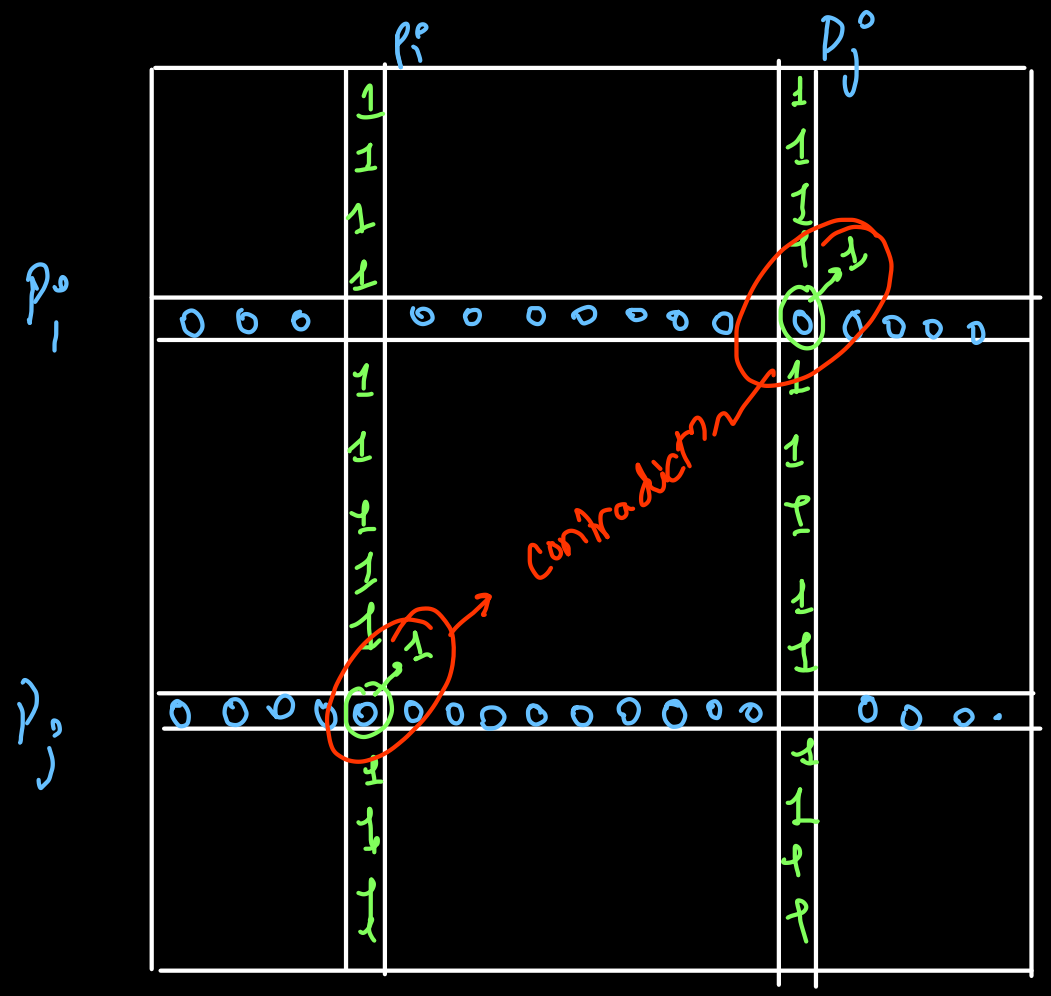
② He should know noone

Q1) Can be there more than 1 celebrities?

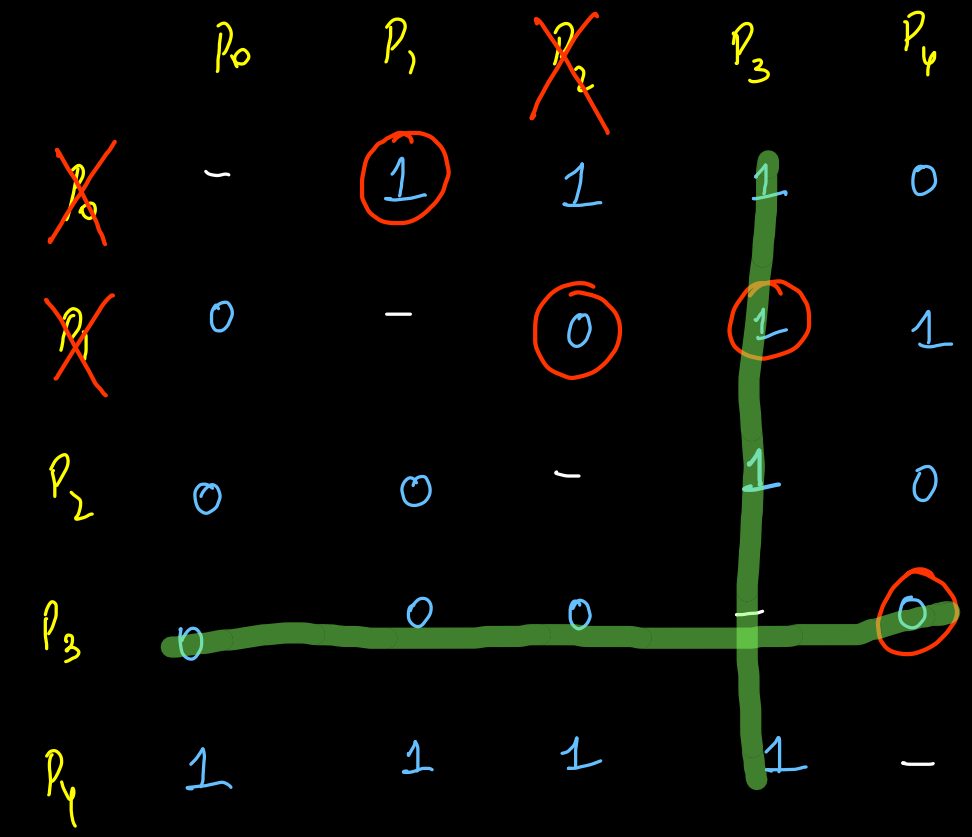
Q2) Can be there 0 celebrities?

↳ Yes

Assume 2 celebrities are there



P_i & P_j are celebrities



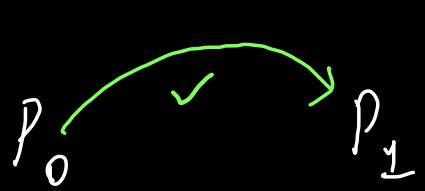
Brute force

↳ Check row == 0 & col == 1 for every person

⇒ Time ⇒ $O(n^2)$

Space ⇒ $O(1)$

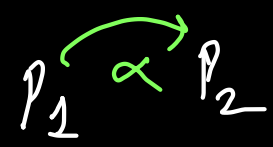
Elimination Answer



$m[P_0][P_1] = 1$

P_0 knows P_1

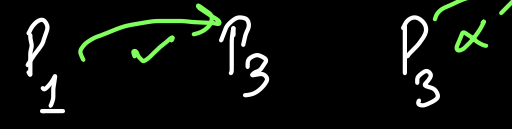
Eliminate P_0



$m[P_1][P_2] = 0$

P_1 don't know P_2

Eliminate P_2



$m[P_1][P_3] = 1$

P_1 know P_3

Eliminate P_1

$m[P_3][P_4] = 0$

P_1 don't know P_4

Eliminate P_4

celebrity = 3

```
class Solution
{
    int celebrity(int M[][], int n) {
        int ans = 0;
        for(int idx = 1; idx < n; idx++){
            if(M[idx][ans] == 0){
                ans = idx;
            }
        }

        for(int i = 0; i < n; i++){
            if(i == ans) continue;
            if(M[ans][i] != 0) return -1;
            if(M[i][ans] != 1) return -1;
        }

        return ans;
    }
}
```

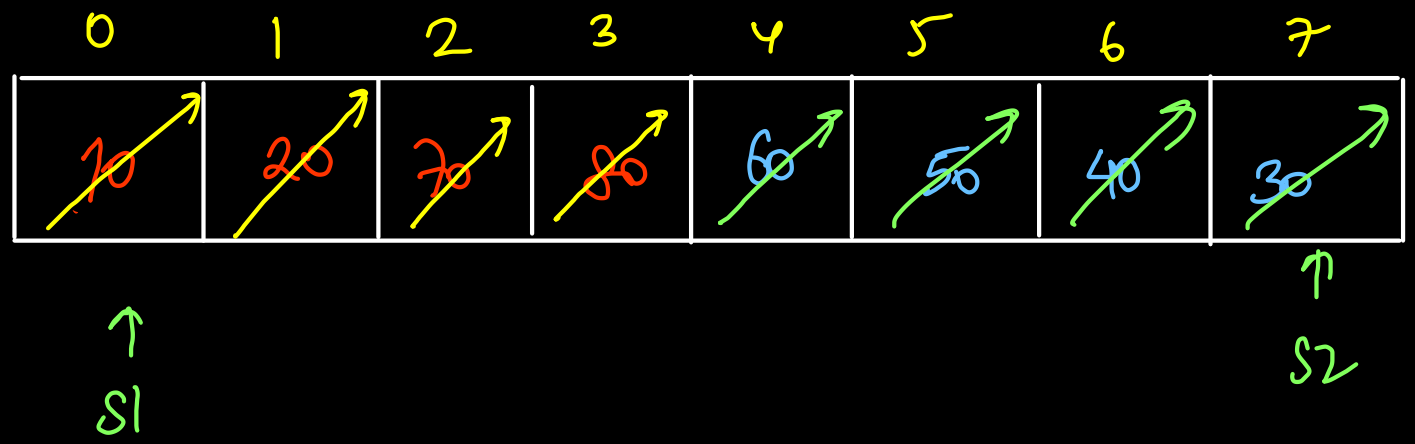
Finding the potential celebrity

No celebrity

Time => O(n)

Space => O(1)

Implement Two Stacks in Array



$S_1 > S_2$: Stack overflow

$S1 == 0$: pop : s1 underflow

$S2 == n-1$: pop : s2 underflow

S1.add(10)

S1.add(20)

S2.add(30)

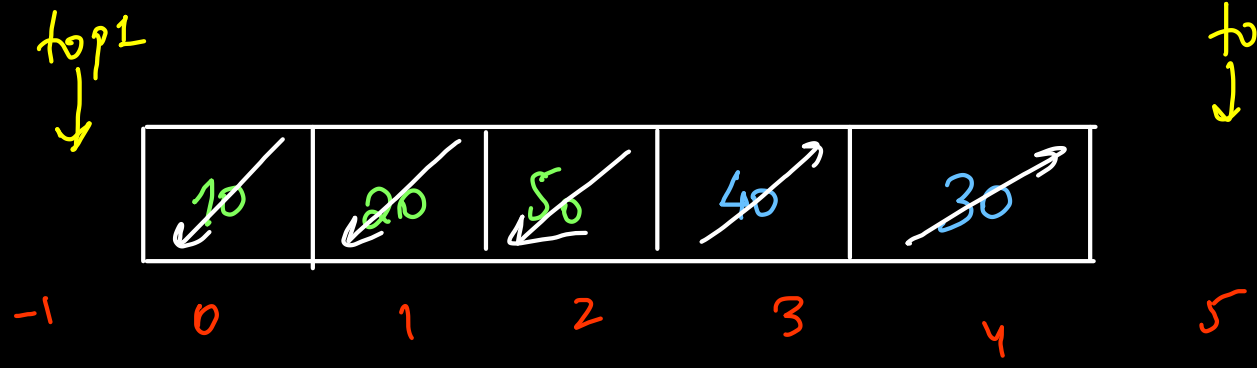
S2.add(40)

S2.add(50)

S2.add(60)

S1.add(70)

S1.add(80)

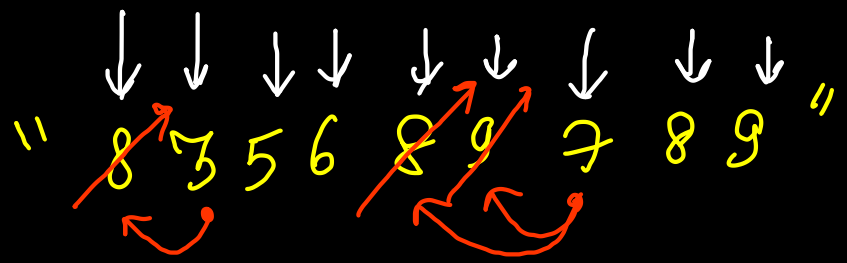



```
/* Structure of the class is
class TwoStack
{

    int size;
    int top1,top2;
    int arr[] = new int[100];

    TwoStack()
    {
        size = 100;
        top1 = -1;
        top2 = size;
    }
}*/
```

```
class Stacks
{
    void push1(int x, TwoStack sq)
    {
        if(sq.top2 == sq.top1 + 1) return; // Overflow
        sq.top1++;
        sq.arr[sq.top1] = x;
    }

    void push2(int x, TwoStack sq)
    {
        if(sq.top2 == sq.top1 + 1) return; // Overflow
        sq.top2--;
        sq.arr[sq.top2] = x;
    }

    int pop1(TwoStack sq)
    {
        if(sq.top1 == -1) return -1;
        return sq.arr[sq.top1--];
    }

    //Function to remove an element from top of the stack2.
    int pop2(TwoStack sq)
    {
        if(sq.top2 == sq.size) return -1;
        return sq.arr[sq.top2++];
    }
}
```

Remove K Digits

leetcode 402

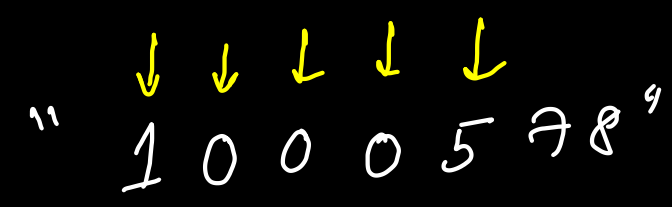


k = 5

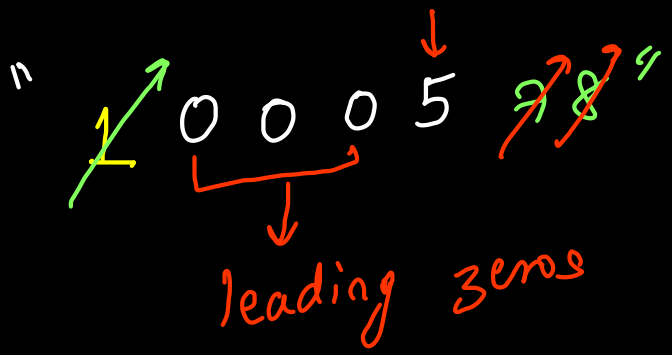
(1) place value
higher ↑ deletion
smaller ↓

answer :→ " 8 3 5 6 8 9 7 8 9 "

✓(2) digit ↑
deletion ↓ smaller



$k = 3$



Time $\Rightarrow O(n)$

Space $\Rightarrow O(1)$

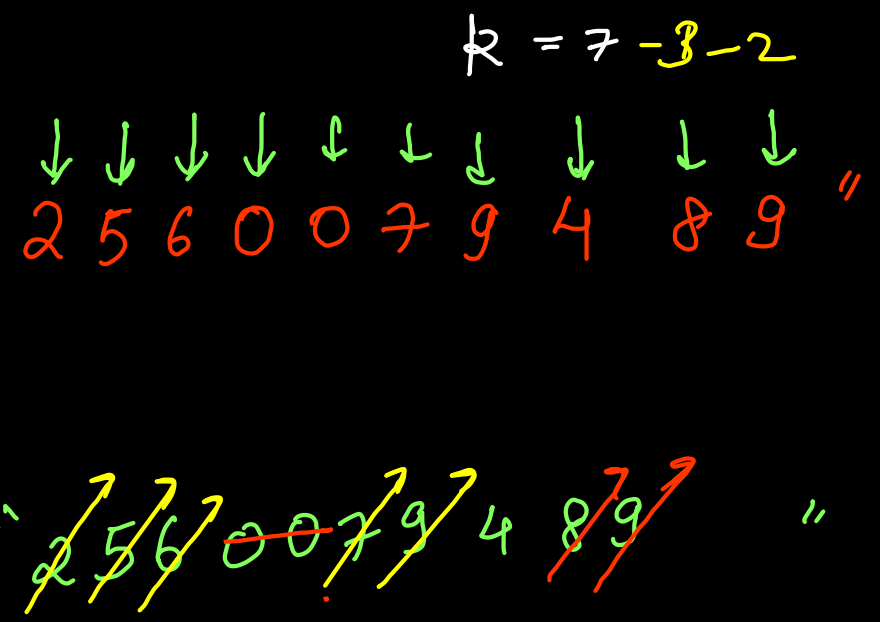
```java
class Solution {
    public String removeKdigits(String num, int k) {

        StringBuilder res = new StringBuilder();

        char[] chs = num.toCharArray();
        for(char ch: chs){
            while(k > 0 && res.length() > 0 && res.charAt(res.length() - 1) > ch){
                res.deleteCharAt(res.length() - 1);
                k--;
            }
            res.append(ch);
        }

        while(k-- > 0){
            res.deleteCharAt(res.length() - 1);
        }

        int idx = 0;
        while(idx < res.length() && res.charAt(idx) == '0')
            idx++;

        if(idx == res.length()) return "0";
        return res.toString().substring(idx);
    }
}
```

monotonic stack

delete last digits

remove leading zeros

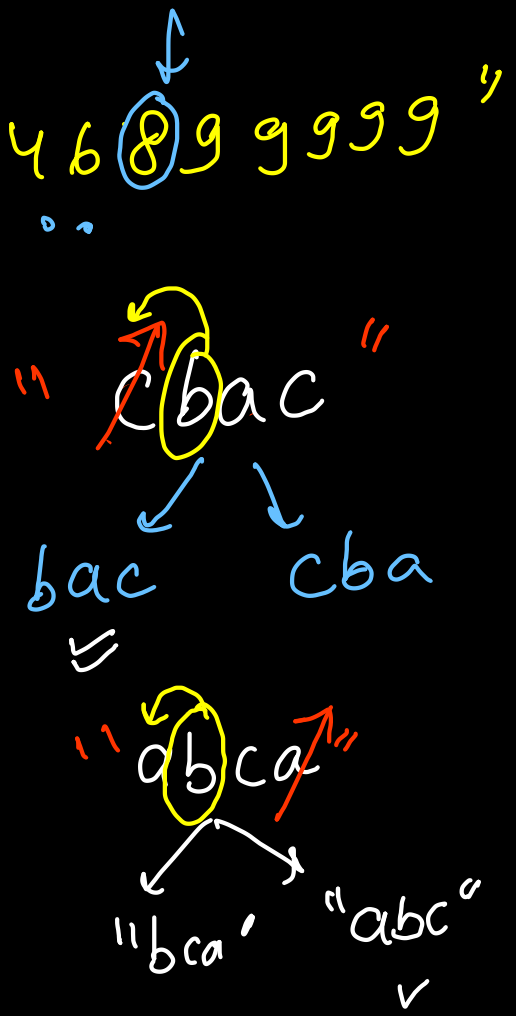
```java
class Solution {
    public String removeKdigits(String num, int k) {

        StringBuilder res = new StringBuilder();

        char[] chs = num.toCharArray();
        for(char ch: chs){
            while(k > 0 && res.length() > 0 && res.charAt(res.length() - 1) > ch){
                res.deleteCharAt(res.length() - 1);
                k--;
            }
            res.append(ch);
        }

        while(k-- > 0){
            res.deleteCharAt(res.length() - 1);
        }

        int idx = 0;
        while(idx < res.length() && res.charAt(idx) == '0')
            idx++;

        if(idx == res.length()) return "0";
        return res.toString().substring(idx);
    }
}
```

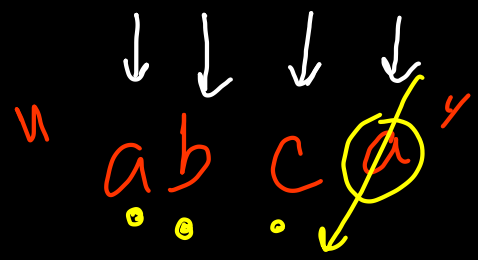


Remove Duplicate letters

lexicographical order / dictionary order

"72469" > "724689999"





a b c

a : 2

b : 1

c : 1

① monotonic stack (String Builder)

② frequency array (duplicate)

③ visited array

" a d b c d a e f a d e "

" a d b c d e f "

a : ~~3~~ ~~2~~ 1 X ✓

b : ~~1~~ 0 X ✓

c : ~~1~~ 0 X ✓

d : ~~3~~ 2 ✓

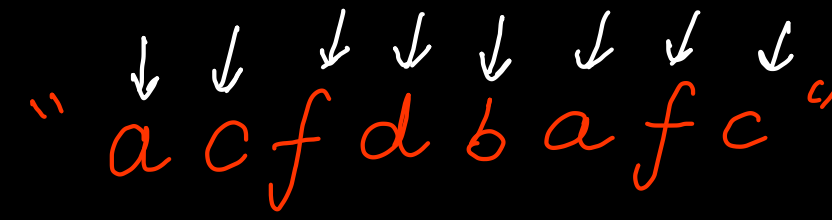
e : ~~2~~ 1 ✓

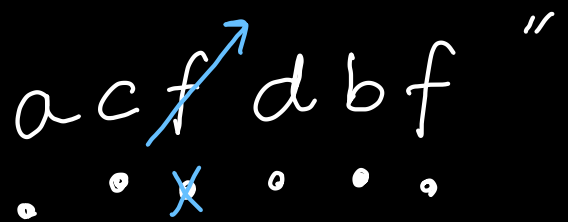
f : ~~1~~ 0 ✓

① duplicate

② lexicographical

```java
public String removeDuplicateLetters(String s) {
    int[] freq = new int[26];
    boolean[] vis = new boolean[26];

    char[] chs = s.toCharArray();
    for(char ch: chs) freq[ch - 'a']++;

    StringBuilder res = new StringBuilder();

    for(char ch: chs){
        freq[ch - 'a']--;
        if(vis[ch - 'a'] == true) continue;
        vis[ch - 'a'] = true;

        while(res.length() > 0){
            char top = res.charAt(res.length() - 1);
            if(top > ch && freq[top - 'a'] > 0){
                res.deleteCharAt(res.length() - 1);
                vis[top - 'a'] = false;
            } else break;
        }

        res.append(ch);
    }

    return res.toString();
}
```



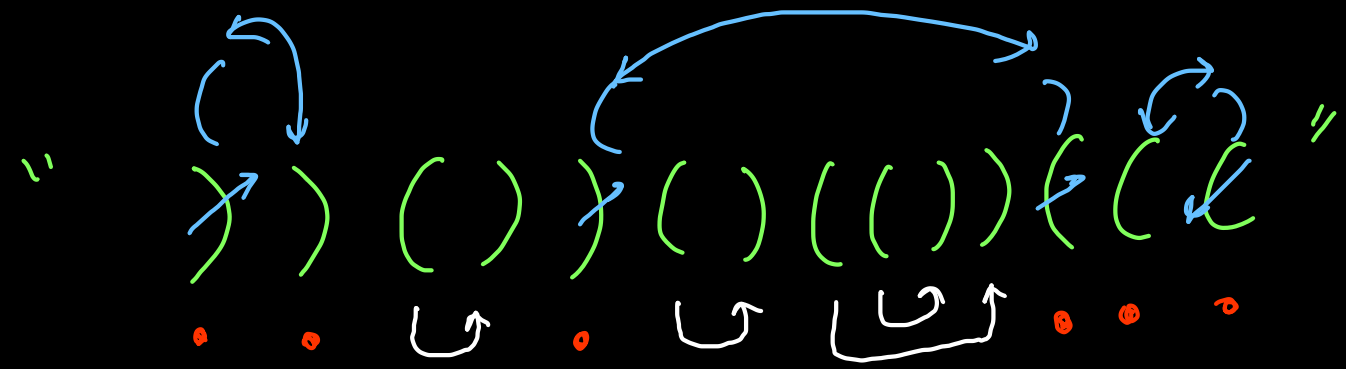


a : ~~2~~ 1 0
b : 1 0
c : ~~2~~ 1
d : 1 0
f : ~~2~~ 1 0

Time ⇒ O(n)

Space ⇒ O(26) = O(1)

Min^m Additions/Removals/Swaps to make it Balanced



⑥ Additions ⇒ number of unbalanced characters

⑥ Removals ⇒ number of unbalanced characters

Swaps ⇒ no of open = no of closing ; no of open/close

③

5th month → Recursion & Backtracking

→ Hashmap & Heaps

6th month → Trees

→ Graphs

7th month → Dynamic Programming

Tries

Bit manipulation

↓

CP batch

```java
public int minAddToMakeValid(String s) {
    int unbalanced_open = 0, unbalanced_close = 0;

    for(char ch: s.toCharArray()){
        if(ch == '(') unbalanced_open++;
        else {
            if(unbalanced_open > 0) unbalanced_open--;
            else unbalanced_close++;
        }
    }

    return unbalanced_open + unbalanced_close;
}
```

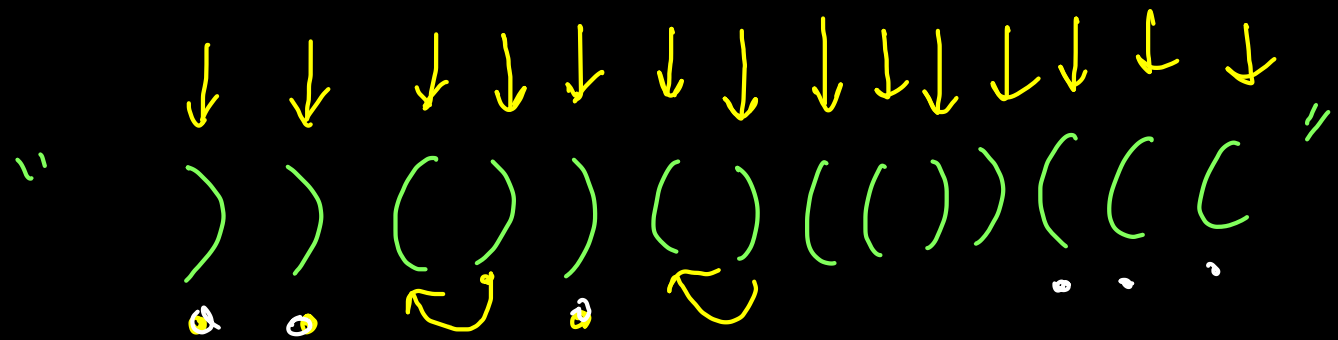
open = 0 1 2 1 0 1 2

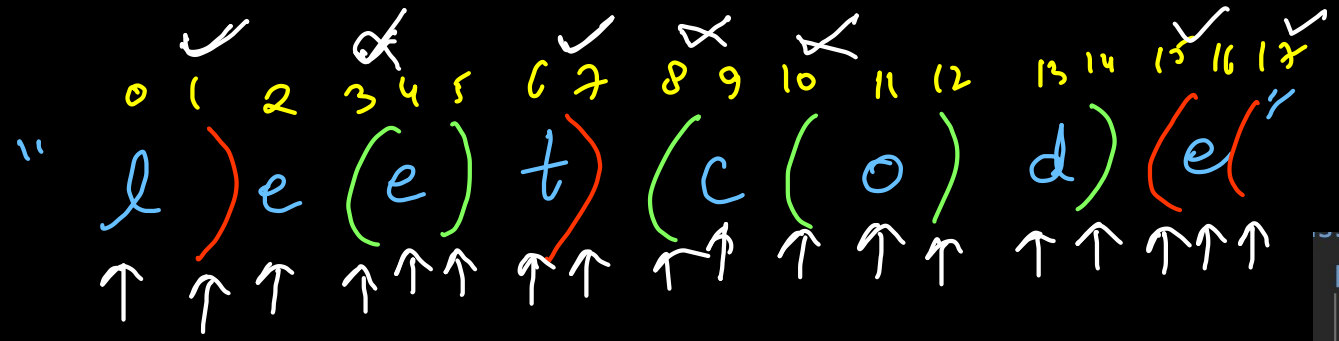
close = 0 1 2 3

3

Time ⇒ O(n)

Space ⇒ O(1)



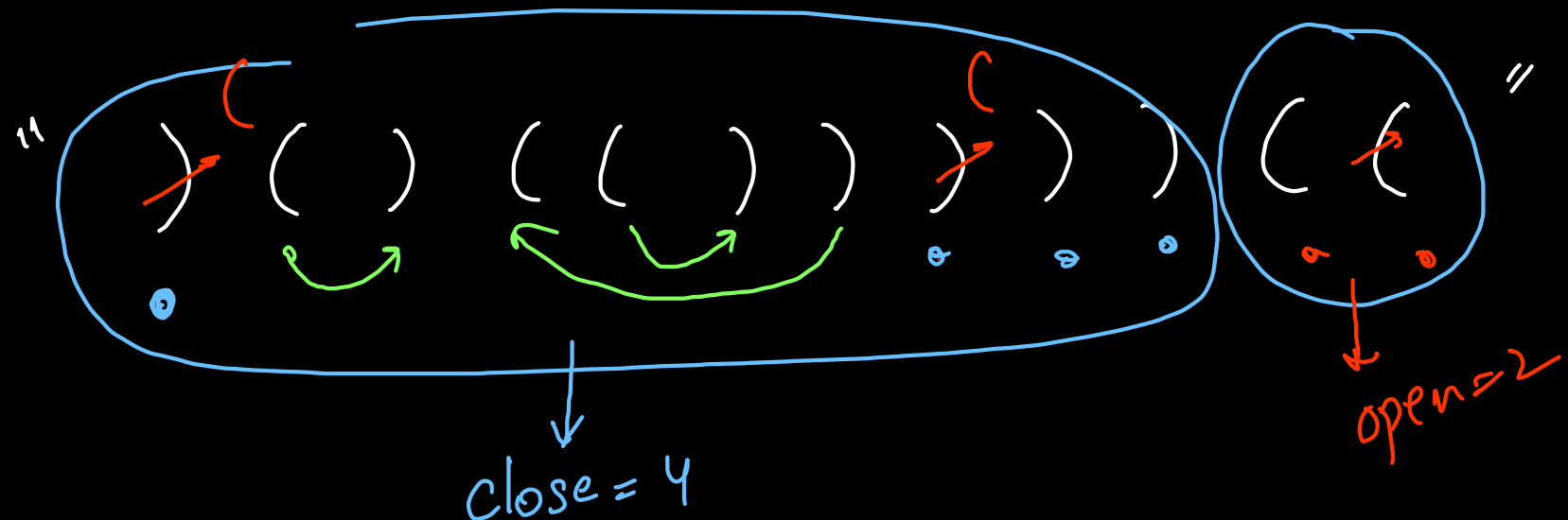


Time ⇒ O(n)

Space ⇒ O(n)

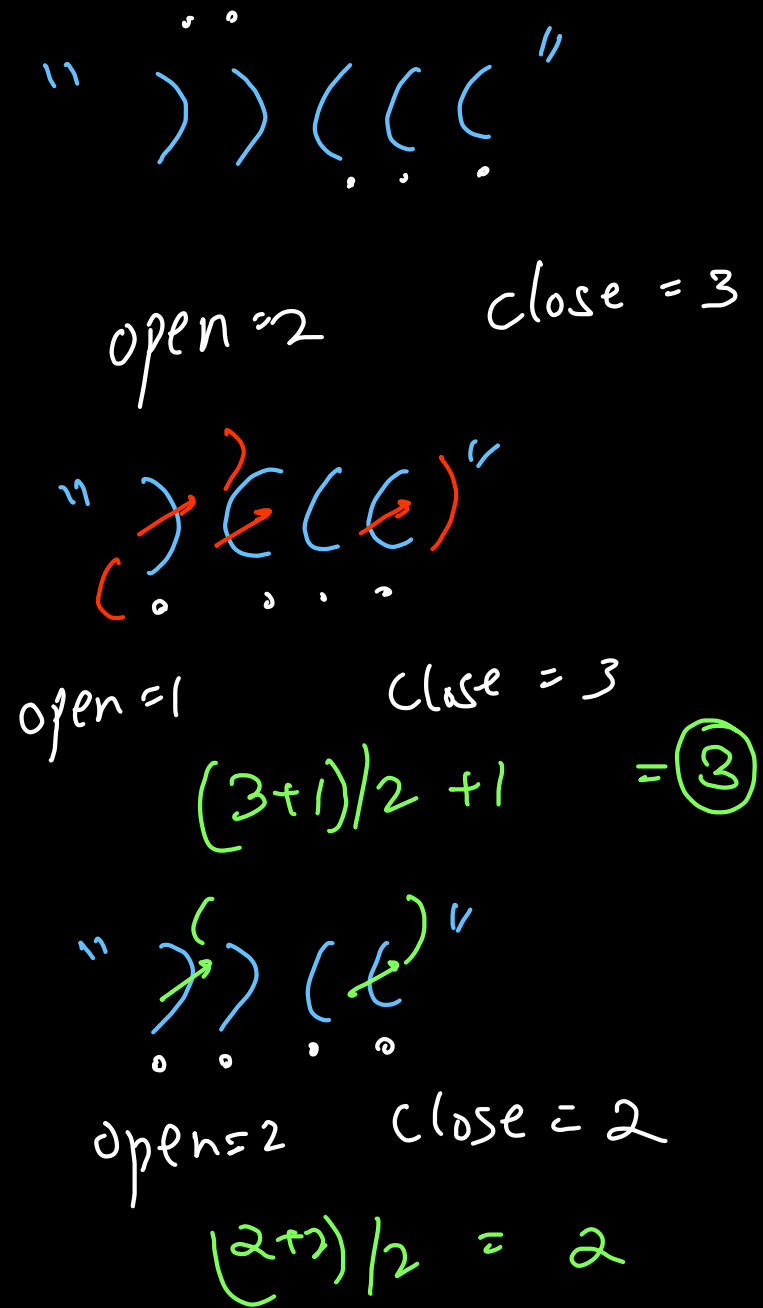
boolean Stack

```java
public String minRemoveToMakeValid(String s) {
    boolean[] unbalanced = new boolean[s.length()];

    Stack<Integer> stk = new Stack<>();
    for(int idx = 0; idx < s.length(); idx++){
        if(s.charAt(idx) == '('){
            stk.push(idx);
            unbalanced[idx] = true;
        } else if(s.charAt(idx) == ')'){
            if(stk.size() > 0){
                unbalanced[stk.pop()] = false;
            } else {
                unbalanced[idx] = true;
            }
        }
    }

    StringBuilder res = new StringBuilder();
    for(int idx = 0; idx < s.length(); idx++){
        if(unbalanced[idx] == false){
            res.append(s.charAt(idx));
        }
    }
    return res.toString();
}
```

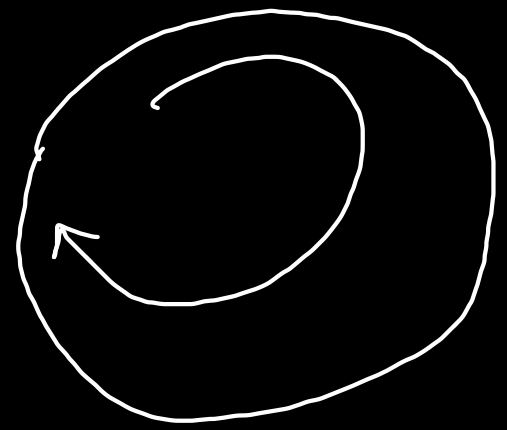

close = 4
open = 2


close = 1
open = 3

```
int countRev (String s)
{
    int open = 0, close = 0;

    for(char ch: s.toCharArray()){
        if(ch == '{') open++;
        else {
            if(open > 0) open--;
            else close++;
        }
    }

    int sum = open + close;
    if(sum % 2 == 1) return -1; //odd length
    if(open % 2 == 1) return sum / 2 + 1;
    return sum / 2;
}
```



Circular Tour / Gas Station LC 134



P_0 1l → 3

P_1 2l → 4

P_2 3l → 5

P_3 4l → 1

P_4 5l → 2

car = 4l + 2l = 6l

car = 6l + 4l = 7l

car = 2l + 3l = 5l

car = 3l + 5l = 8l

car = 0 + 4l

Start

```java
public int canCompleteCircuit(int[] gas, int[] cost) {
    int n = gas.length;
    for(int st = 0; st < n; st++){
        int fuel = 0;
        boolean ans = true;

        for(int jump = 0; jump < n; jump++){
            int curr = (st + jump) % n;
            fuel += gas[curr];

            if(fuel < cost[curr]){
                ans = false;
                break;
            }

            fuel -= cost[curr];
        }

        if(ans == true) return st;
    }
    return -1;
}
```

not able to complete circular tour.

no starting pt available

Brute force

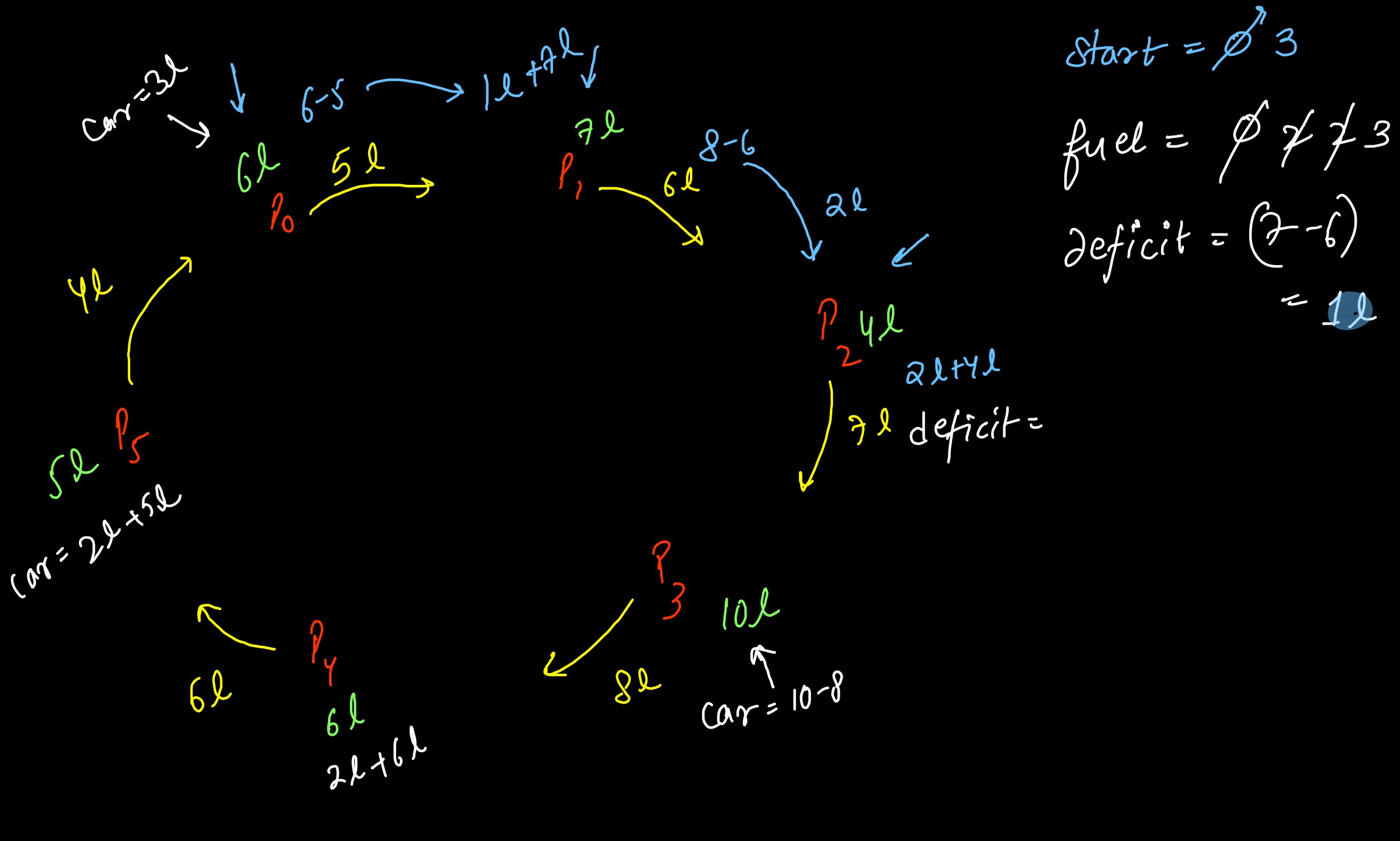
↳ Consider all pts as starting index

Time ⇒ $O(n^2)$ TLE

worst case

Space ⇒ $O(1)$

extra


car=3l
6-5
1l+7l
6l
5l
7l
8-6
2l
4l
2l+4l
7l deficit=
5l
car=2l+5l
6l
6l
2l+6l
8l
10l
car=10-8
start = 3
fuel = 3
deficit = (7-6)
= 1l

```java
class Solution {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int deficit = 0, fuel = 0, start = 0;

        for(int idx = 0; idx < gas.length; idx++){
            fuel += gas[idx] - cost[idx];

            if(fuel < 0){
                start = idx + 1;
                deficit += -fuel;
                fuel = 0;
            }
        }

        if(fuel >= deficit) return start;
        return -1;
    }
}
```

skip starting pt as P_1, P_2

circular tour (skip P_0 to P_3 again)

Time ⇒ O(n)

Space ⇒ O(1)

Expressions

BODMAS

a + b
a − b
a / b
a * b

⇓

Infix

v1 op v2

5 + (2 * (6 / 3))

5 * 3 / 2

Maths

5 * (3/2)

5 * 1

5

Coding

(5 * 3) / 2

15/2

7

⇒ Bracket

⇒ Division = multiplicam
- same priority
- left to right
- Associative