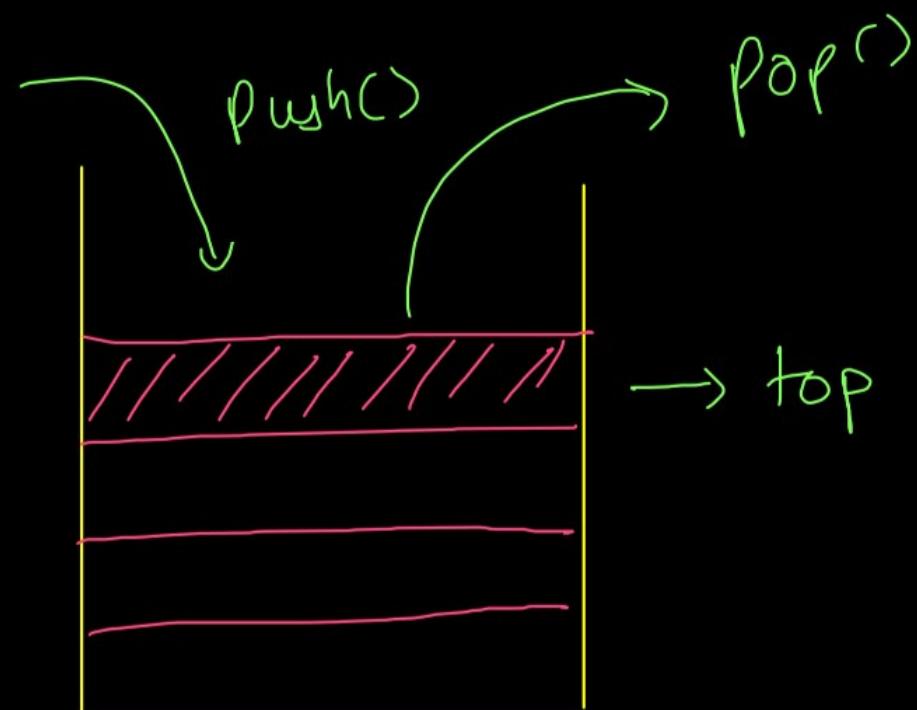


Stack

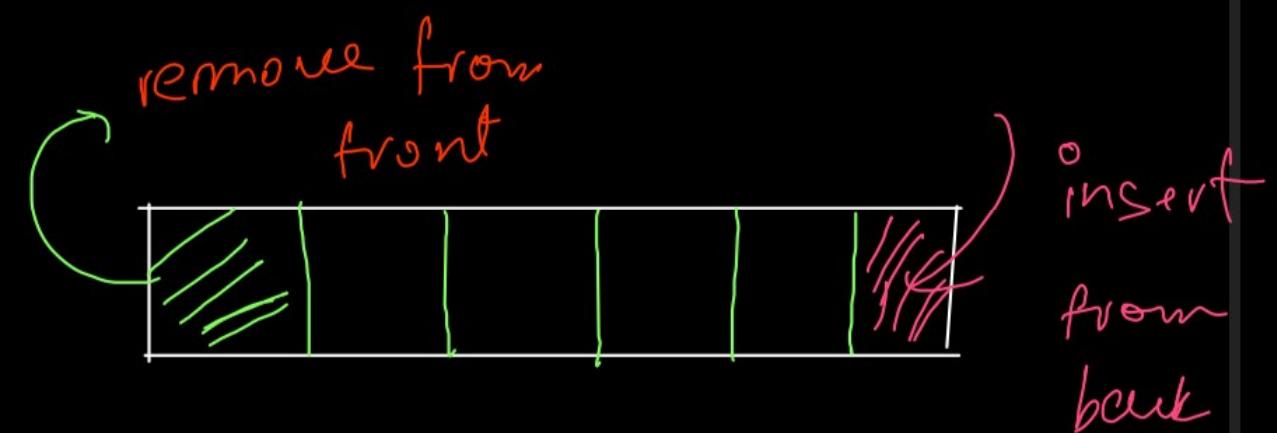
{ LIFO or FILO }



pop and push both
from the top

Queue

{ FIFO or LILO }





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

    public void push(int val) {
        if(stk.size() == 0) {
            stk.push(11 * val);
            min = (long)val;
        } else {
            //Encrypt the value if it is less than min
            if(val < min) {
                stk.push(21 * val - min);
                min = (long)val;
            } else {
                stk.push(11 * val);
            }
        }
    }
}
```

```
| public void pop() {
|     if(stk.size() == 0) return;
|
|     //check if top is encrypted
|     if(stk.peek() < min) {
|         //decrypt the prev min
|         min = 21 * min - stk.pop();
|     } else {
|         stk.pop();
|     }
|
|     public int top() {
|         if(stk.size() == 0) return -1;
|
|         //check if top is encrypted
|         if(stk.peek() < min) {
|             //decrypt the prev min
|             return (int)min;
|         }
|
|         long top = stk.peek();
|         return (int)top;
|     }
|
|     public int getMin() {
|         if(stk.size() == 0) return -1;
|
|         return (int)min;
|     }
| }
```



Without extra space.

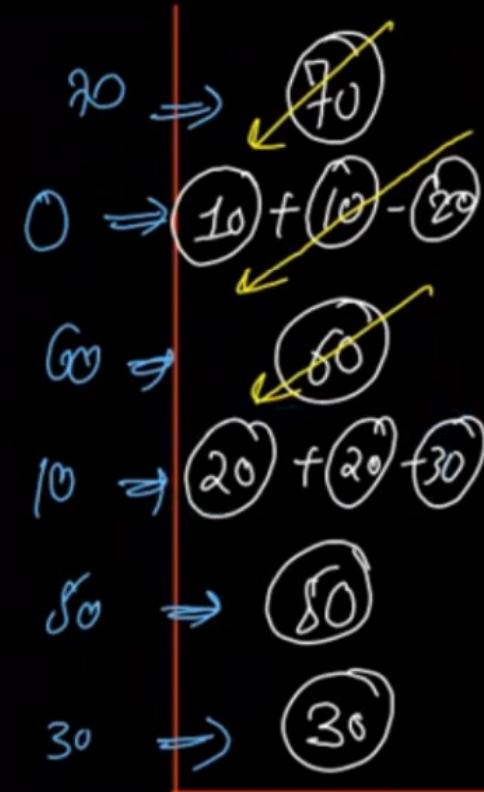
{without extra space}

| push (value) | peek |
|--------------|------|
| ✓ push (30) | 30 |
| ✓ push (50) | 50 |
| ✓ push (20) | 20 |
| ✓ push (60) | 60 |
| ✓ push (10) | 10 |
| ✓ push (70) | 70 |

```

graph TD
    30[30] --> 20[20]
    30 --> 10[10]
    20 --> 10
    20 --> 10

```



Decryption

$$\text{top} = \frac{\min \text{frnch} - \text{prev}}{\text{prev} = 2 * \min - \text{top}}$$

10

30
min

Encrypt \Rightarrow minimum node

$$\rightarrow \text{val} + (\text{val} - \min) \\ 20 + (20 - 30) \\ 20 + -ve \\ < 20$$



```
class MinStack {

    Stack<Integer> dataS;
    Stack<Integer> minS;
    int min;

    public MinStack() {
        dataS = new Stack<>();
        minS = new Stack<>();
        min = Integer.MAX_VALUE;
    }

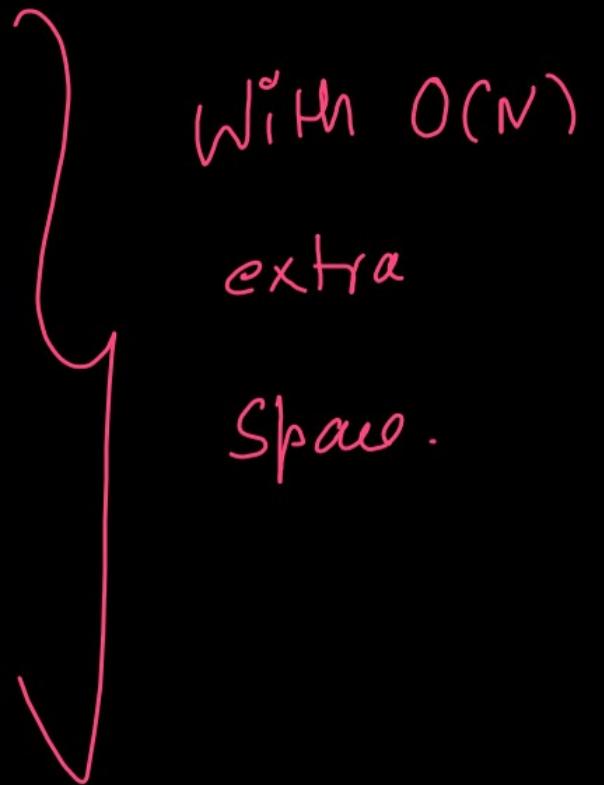
    public void push(int val) {
        dataS.push(val);
        min = (minS.size() == 0) ? Integer.MAX_VALUE : minS.peek();
        if(min < val)
            minS.push(min);
        else
            minS.push(val);
    }

    public void pop() {
        if(dataS.size() == 0) return;
        dataS.pop();
        minS.pop();

        if(minS.size() == 0) min = Integer.MAX_VALUE;
    }

    public int top() {
        if(dataS.size() == 0) return -1;
        return dataS.peek();
    }
}
```

```
    public int getMin() {
        if(dataS.size() == 0) return -1;
        return minS.peek();
    }
}
```



Minimum Stack

Without extra space

With extra space

Minimum Stack

with out extra space with extra space

| | peek() | min() |
|------------|--------|----------|
| push(30) | 30 | { 30 } |
| push(50) | 50 | { 30 } ↑ |
| push(20) | 20 | { 20 } ↑ |
| push(60) | 60 | { 20 } ↑ |
| push(10) | 10 | { 10 } ↑ |
| push(70) | 70 | |

pop()





```
2 class Solution
3 {
4     //Function to find if there is a celebrity in the party or not.
5     int celebrity(int M[][], int n)
6     {
7         // code here
8         Stack<Integer> stk = new Stack<>();
9
10        for(int i=0;i<M.length;i++) {
11            stk.push(i);
12        }
13
14        while(stk.size() > 1) {
15            int y = stk.pop();
16            int x = stk.pop();
17            if(M[x][y] == 1) {
18                stk.push(y); //y may or may not be a celeb
19            } else {
20                stk.push(x); //x may or may not be a celeb
21            }
22        }
23
24        int x = stk.pop();
25
26        for(int j=0;j<M.length;j++) {
27            if(j == x) continue;
28            if(M[x][j] == 1) {
29                return -1;
30            }
31        }
32
33        for(int i=0;i<M.length;i++) {
34            if(i == x) continue;
35            if(M[i][x] == 0) {
36                return -1;
37            }
38        }
39
40        return x;
41    }
42 }
```

Celebrity Problem Code.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 |

Last remaining ele
may or may not
be a
celeb. So,
we have
to check
for it.

possible Celebrities.

0 1 2 3 2 2 ②

This is an elimination technique.

$$T(n) = T(n-1) + O(1)$$

$O(n)$

if $\text{mat}[2][3] = 1$

↓
2 can't be a celebrity.

if $\text{mat}[2][3] = 0$

↓
3 can't be a celeb.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 |

We do not have any celebrity.
Hence 0 celebrity is possible.

Let us say x and y both are celebrities. So, x does not know y & y does not know x . However since x is a celebrity, y should know x & vice-versa. Hence, there is a contradiction. So, we cannot have more than 1 celebrity.

Celebrity Problem.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 |

mat [i] [j]

- ↳ 1 i^{th} person knows j^{th} person
- ↳ 0 i^{th} person does not know j^{th} person

Celebrity \rightarrow who knows nobody
 ↳ everybody knows him/her.

No celebrity ✓ Possible

More than ✗ Not
 1 celebrity possible.

Uski Row me all 0s
 Uske col me all 1s except itself.



```
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        int[] res = new int[nums.length - k + 1];
        Deque<Integer> q = new ArrayDeque<>();
        int idx = 0;
        for(int i=0;i<nums.length;i++) {
            //remove first
            if(q.size() > 0 && q.getFirst() <= i-k) {
                q.removeFirst();
            }

            //removeLast those elements that are smaller than us
            while(q.size() > 0 && nums[q.getLast()] < nums[i]) {
                q.removeLast();
            }

            //addLast -> we may be the answer of current or upcoming windows
            q.addLast(i);

            if(i >= k-1) {
                res[idx++] = nums[q.getFirst()];
            }
        }

        return res;
    }
}
```

$O(N)$ Time

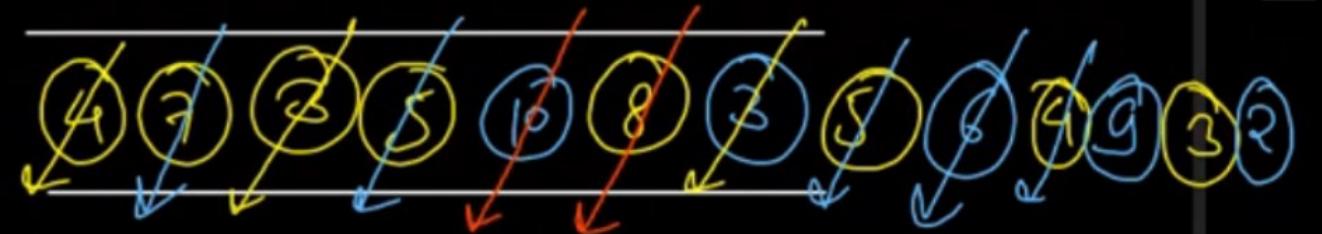
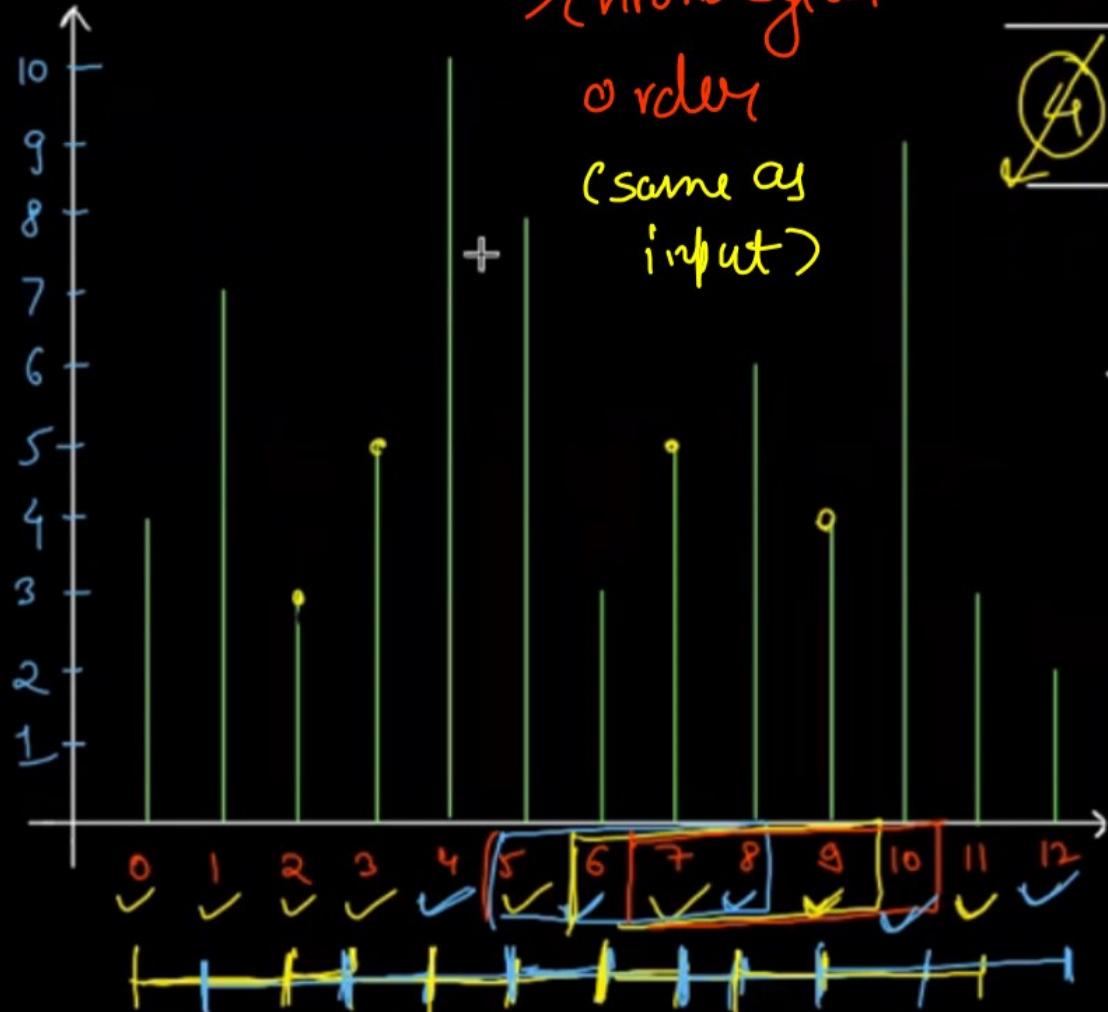
$O(k)$ Space

Deque is in \rightarrow decreasing order

\rightarrow chronological

order

(same as input)



$\{7, 10, 10, 10, 8, 6, 9, 9, 9\}$
 $\# \text{getFirst} \rightarrow$ maximum of curr window
 which get excluded from the window

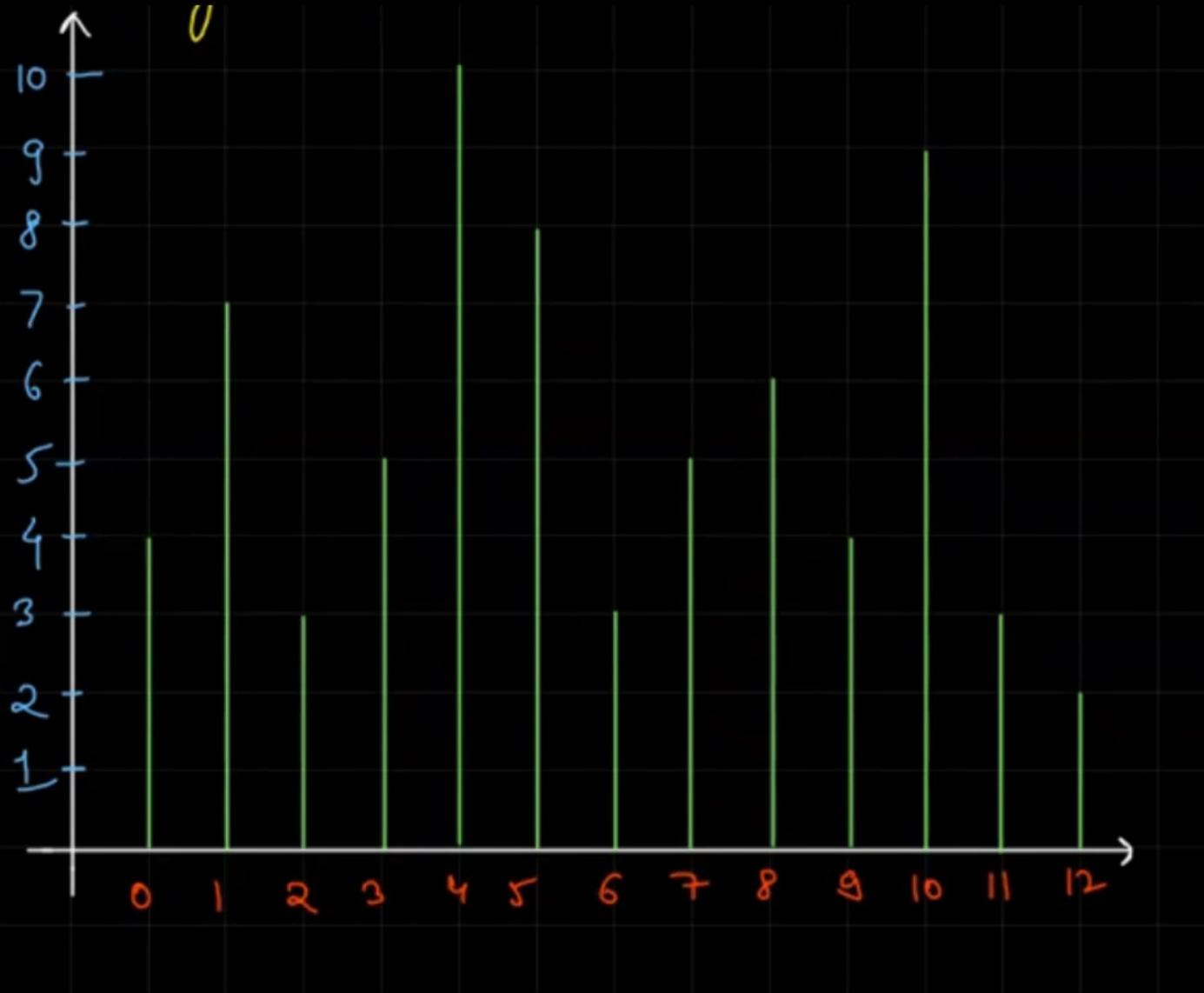
① removeFirst

→ smaller than oneself
 (can't be max of any window)

② removeLast

→ we may find max of upcoming windows

Sliding Window Maximum



Window = 4

Brute force: $O(N \times K)TC$

① Using Dequeue (Queue)

② Using Stack (NGE)

Dequeue is an ADT (Abstract Data Type). LL is itself an implementation, Dequeue is not.

```
public int calculate(String exp) {
    exp = refine(exp);

    Stack<Character> operator = new Stack<>();
    Stack<Integer> operand = new Stack<>();

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

        if(ch >= '0' && ch <= '9'){
            int num = 0;
            while(i < exp.length() && exp.charAt(i) >= '0' && exp.charAt(i) <= '9'){
                num = num * 10 + (exp.charAt(i) - '0');
                i++;
            }

            // operand -> push in operand wali stack
            operand.push(num);
            i--;
        } else if(ch == '('){
            operator.push(ch);
        } else if(ch == ')'){
            while(operator.peek() != '('){
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(performOp(a, op, b));
            }

            operator.pop(); // pop opening braces as well
        }
    }

    Stack<Character> operator = new Stack<>();
    Stack<Integer> operand = new Stack<>();

    for(int i=0; i<operator.size(); i++){
        char ch = operator.pop();

        if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){
            while(operator.size() > 0 && operator.peek() != '(' && precedence(operator.peek()) >= precedence(ch)){
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(performOp(a, op, b));
            }

            operator.push(ch);
        }
    }

    while(operator.size() > 0){
        int b = operand.pop();
        int a = operand.pop();
        char op = operator.pop();
        operand.push(performOp(a, op, b));
    }

    return operand.peek();
}
```



```
class Solution {
    int performOp(int a, char op, int b) {
        int out = 0;

        switch(op)
        {
            case '+': out = a + b; break;
            case '-': out = a - b; break;
            case '*': out = a * b; break;
            case '/': out = a / b; break;
        }
        return out;
    }

    int precedence(char op)
    {
        if(op == '+' || op == '-') return 1;
        else if (op == '*' || op == '/') return 2;
        else return 0;
    }
}
```

```
public String refine(String s){
    StringBuilder str = new StringBuilder("");

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

        if(ch == ' ') continue;
        if(ch == '+'){
            if(str.length() == 0 || str.charAt(str.length() - 1) == '('){
                // unary +
                str.append("0+");
            } else if(str.charAt(str.length() - 1) == '+' || str.charAt(str.length() - 1) == '-'){
                continue;
            } else str.append('+');
        }

        else if(ch == '-'){
            if(str.length() == 0 || str.charAt(str.length() - 1) == '('){
                // unary -
                str.append("0-");
            } else if(str.charAt(str.length() - 1) == '+')
                str.setCharAt(str.length() - 1, '-');
            else if(str.charAt(str.length() - 1) == '-')
                str.setCharAt(str.length() - 1, '+');
            else str.append('-');
        }

        else str.append(ch);
    }

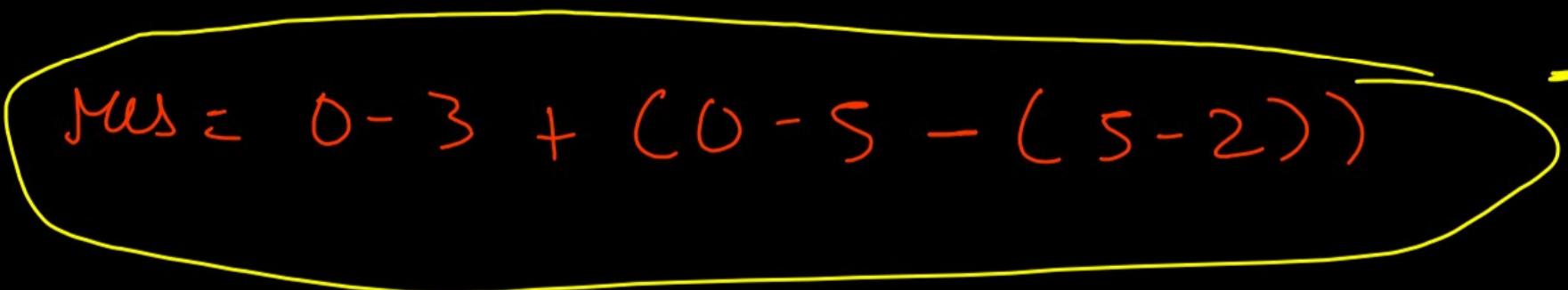
    return str.toString();
}
```

$$" -3 + (-5 - (+5 + -2)) "$$

Before this '-' we have '+'. We can also have *.
So, the safer thing is to do when there is an operator
before - is to take it into brackets

$$\text{res} = " \textcircled{0} - 3 + (\textcircled{0} - 5 - (\textcircled{5} + \textcircled{0} - 2))."$$

$$= -3 + (-5 - (+ 5 + -2))$$

Since '-' is just after '+', change the '+' symbol in resultant to '-'.


$$\text{Ans} = 0 - 3 + (0 - 5 - (5 - 2))$$

Evaluating
this string
is easy

& we have it in
infix evaluation.
Just have to make
some changes.

" $-3 + (-5 - (+5 + -2))$ " → Input String

Since the srs was empty initially, & we had '-' sign, we add a 0 before it

$$srs = 0 - 3 + ($$

" $-3 + (-5 - (+5 + -2))$ "

Since '-' is just after the opening bracket, add 0 before it.

$$srs = 0 - 3 + (0 - 5 - ($$

" $-3 + (-5 - (+5 + -2))$ "

Since '+' is just after the opening bracket, we can ignore it

$$srs = 0 - 3 + (0 - 5 - (5 +$$

Basic Calculator. 1, 2 & 3 → Lintcode

Ask the interviewer whether the string can or cannot the following

(), spaces, More than 1 digit nos , unary operators like (-s) etc.

further are there any operators apart from arithmetic operators

that have associativity & to L -

" -3 +(-5 -(+5 +-2)) " → Input String

The image shows a digital whiteboard interface with various tools and icons at the top. The main area contains two code snippets.

Left Snippet:

```
// code
Stack<Integer> value = new Stack<>();
Stack<String> infix = new Stack<>();
Stack<String> prefix = new Stack<>();

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

    if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
        int v2 = value.pop();
        int v1 = value.pop();
        int val = operation(v1,ch,v2);
        value.push(val);

        String inv2 = infix.pop();
        String inv1 = infix.pop();
        String inv = "(" + inv1 + ch + inv2 + ")";
        infix.push(inv);

        String prev2 = prefix.pop();
        String prev1 = prefix.pop();
        String prev = "" + ch + prev1 + prev2;
        prefix.push(prev);
    } else {
        value.push(ch - '0');
        infix.push("" + ch);
        prefix.push("" + ch);
    }
}

System.out.println(value.pop());
System.out.println(infix.pop());
System.out.println(prefix.pop());
```

Right Snippet:

```
public static int operation(int val1, char op, int val2) {
    if(op == '+') {
        return val1 + val2;
    } else if(op == '-') {
        return val1 - val2;
    } else if(op == '*') {
        return val1 * val2;
    }

    return val1/val2;
}
```



Postfix Evaluation and Conversion.

5234-231+*4*+

$$0 - 1 = -1$$

$$-2/3 = 0$$

$$(3-4) \\ = \textcircled{1}$$

-3

-8

42

-1

C
3
2

1

3
2

5

evaluation

$+5**2+-34/234 \Rightarrow$ prefix

* * 2 + -34 / 23 4

$$\begin{array}{r} \cancel{4} \\ \cancel{*} 2 + -34 | 23 \end{array}$$

~~+ -34 /23~~

123

3

$$\textcircled{-3y}$$

43

2

34

Prefix



```
import java.io.*;
import java.util.*;

public class Main{

    public static int precedence(char ch) {
        if(ch == '*' || ch == '/') return 2;
        if(ch == '+' || ch == '-') return 1;
        return 0;
    }

    public static void performOp(Stack<String> prefix,Stack<String> postfix,char op) {
        String prefixB = prefix.pop();
        String prefixA = prefix.pop();
        String prefixRes = "" + op + prefixA + prefixB;
        prefix.push(prefixRes);

        String postfixB = postfix.pop();
        String postfixA = postfix.pop();
        String postfixRes = "" + postfixA + postfixB + op;
        postfix.push(postfixRes);
    }
}
```

```
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String exp = br.readLine();

    // code
    Stack<Character> operator = new Stack<>();
    Stack<String> prefix = new Stack<>();
    Stack<String> postfix = new Stack<>();

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

        if(ch >= 'a' && ch <= 'z') {
            prefix.push(ch + "");
            postfix.push(ch + "");
        } else if(ch == '(') {
            operator.push(ch);
        } else if(ch == ')') {
            while(operator.peek() != '(') {
                char op = operator.pop();
                performOp(prefix,postfix,op);
            }
            operator.pop();
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            while(operator.size() > 0 && operator.peek() != '(' && precedence(operator.peek()) >= precedence(ch)) {
                char op = operator.pop();
                performOp(prefix,postfix,op);
            }
            operator.push(ch);
        }
    }

    while(operator.size() > 0) {
        char op = operator.pop();
        performOp(prefix,postfix,op);
    }

    System.out.println(postfix.peek());
    System.out.println(prefix.peek());
}
```



$$5 + 2 * (3 - 4 + 2 / 3) * 4$$

~~operator~~

~~*~~ ~~0~~ ~~+~~ ~~*~~ ~~0~~ ~~+~~ ~~*~~ ~~0~~ ~~+~~

operator

~~* * 2 + - 3 4 / 2 3 4~~ ~~0~~ ~~* 2 + - 3 4 / 2 3~~ ~~0~~ ~~+ - 3 4 / 2 3~~ ~~0~~ ~~1 2 3~~ ~~0~~ ~~3 0 2 - 3 4~~ ~~0~~ ~~4 0 2 0 5~~ ~~0~~

prefix

~~2 3 4 - 2 3 1 + * 4 4~~ ~~0~~ ~~2 3 4 - 2 3 1 + * 0~~ ~~0~~ ~~3 4 - 2 3 1 +~~ ~~0~~ ~~2 3 1~~ ~~0~~ ~~3 0 2 3 4 -~~ ~~0~~ ~~4 0 2 0 5~~ ~~0~~

postfix

Prefix : $+ 5 * * 2 + - 3 4 / 2 3 4$

Postfix : $5 2 3 4 - 2 3 1 + * 4 * +$

Infix Conversion

Prefix

Postfix

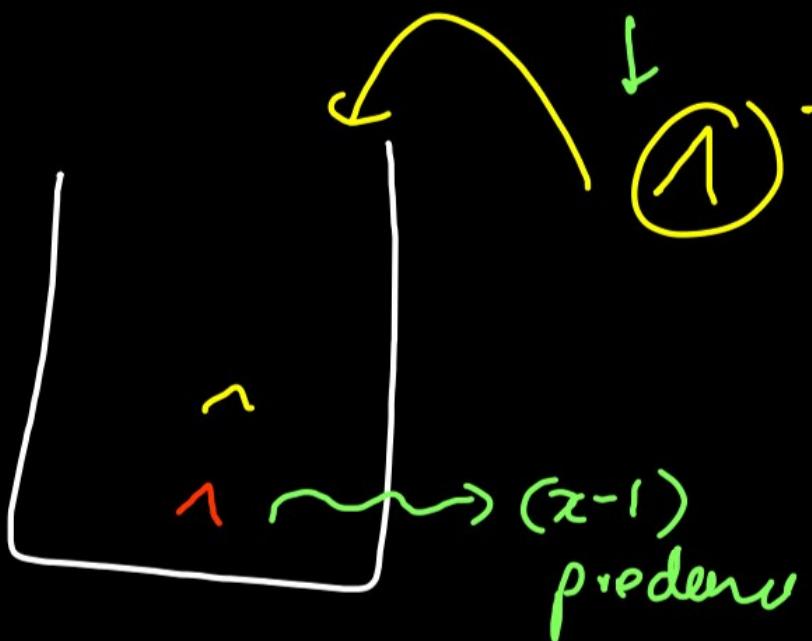
(Mostly compilers convert infix to
postfix)

Prefix : operator operand₁ operand₂ (+ab)

Postfix : operand₁ operand₂ operator (ab+)

Agar Seeme qus me right to left associativity bol dia
 to hume dekhna padega ki operators stack me jo hai
 aur jo stack k bahar hai unki precedence alag ho. Ex

2 1 3 ^ 1 4



x (precedence)

\sim
 $\sim \rightarrow (z-1)$
 precedence

this should
 have more
 precedence wrt
 \wedge which is
 inside the stack



```
public static int calculate(int a,char op, int b) {
    if(op == '+') {
        return a + b;
    } else if(op == '-') {
        return a - b;
    } else if(op == '*') {
        return a * b;
    } else {
        return a/b;
    }
}

public static int precedence(char op) {
    if(op == '*' || op == '/') return 2;
    else if(op == '+' || op == '-') return 1;
    return 0;
}
```

TC = O(N)

```
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String exp = br.readLine();

    // code
    Stack<Character> operator = new Stack<>();
    Stack<Integer> operand = new Stack<>();

    for(int i=0;i<exp.length();i++) {
        char ch = exp.charAt(i);
        if(Character.isDigit(ch) == true) {
            operand.push(ch-'0');
        } else if(ch == '(') {
            operator.push(ch);
        } else if(ch == ')') {
            while(operator.peek() != '(') {
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(calculate(a,op,b));
            }
            operator.pop();
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            while(operator.size() > 0 && operator.peek() != '(' && precedence(operator.peek()) >= precedence(ch)) {
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(calculate(a,op,b));
            }
            operator.push(ch);
        }
    }

    while(operator.size() > 0) {
        int b = operand.pop();
        int a = operand.pop();
        char op = operator.pop();
        operand.push(calculate(a,op,b));
    }

    System.out.println(operand.pop());
}
```



$$5 + 2 * (3 - 4 + \frac{2}{3}) * 4 + 2$$

```

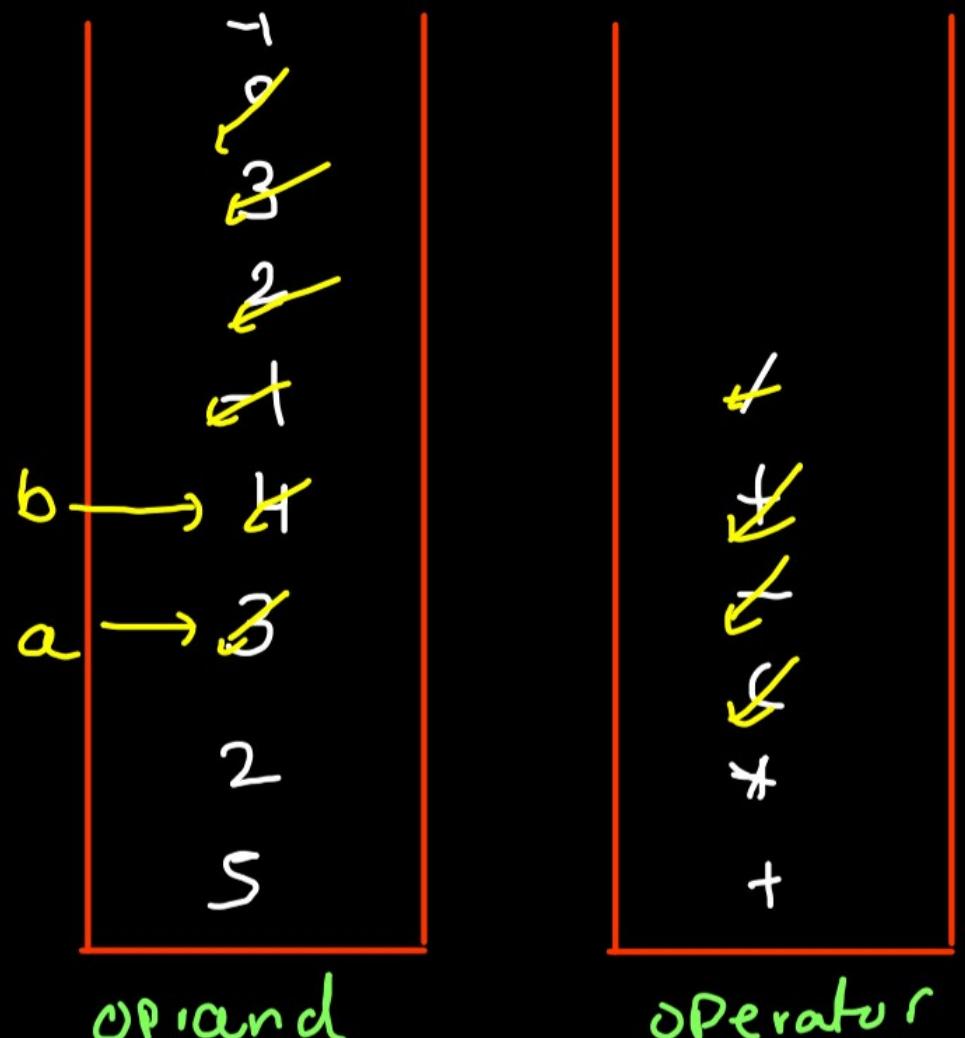
if (ch >='0' && ch <='9')
    operand. push(ch)
else if (ch == '+')
    operator. push(ch)
else if (ch == ')')
    pop until '(', perform, res.push operand
    >     pop ')' also
else {
    while ( stack.size > 0 && stack.peek() != '(' && prec(stack.peek()) >= prec(ch))
        pop & push
    push yourself()
}

```

3

Infix Evaluation

$$5 + 2 * (3 - 4 + 2 / 3) * 4 + 2$$



$$(3 - 4) = ①$$

↓
push

$$2 / 3 = 0$$

Priority

① Brackets

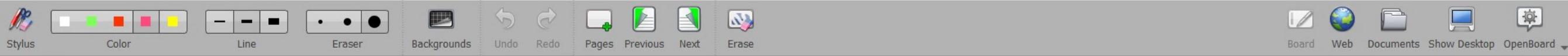
② Division & Multiplication

③ Addition & Subtraction

Left to Right Associativity

(+, -, *, /)

Power (^) is Right to Left association.



```
class Solution {
    public int largestRectangleArea(int[] heights) {
        int n = heights.length;
        Stack<Integer> st=new Stack<>();
        int[] rb=new int[n]; //nse on the right
        int[] lb=new int[n]; //nse on the left

        st.push(heights.length-1);
        rb[heights.length-1]=heights.length;

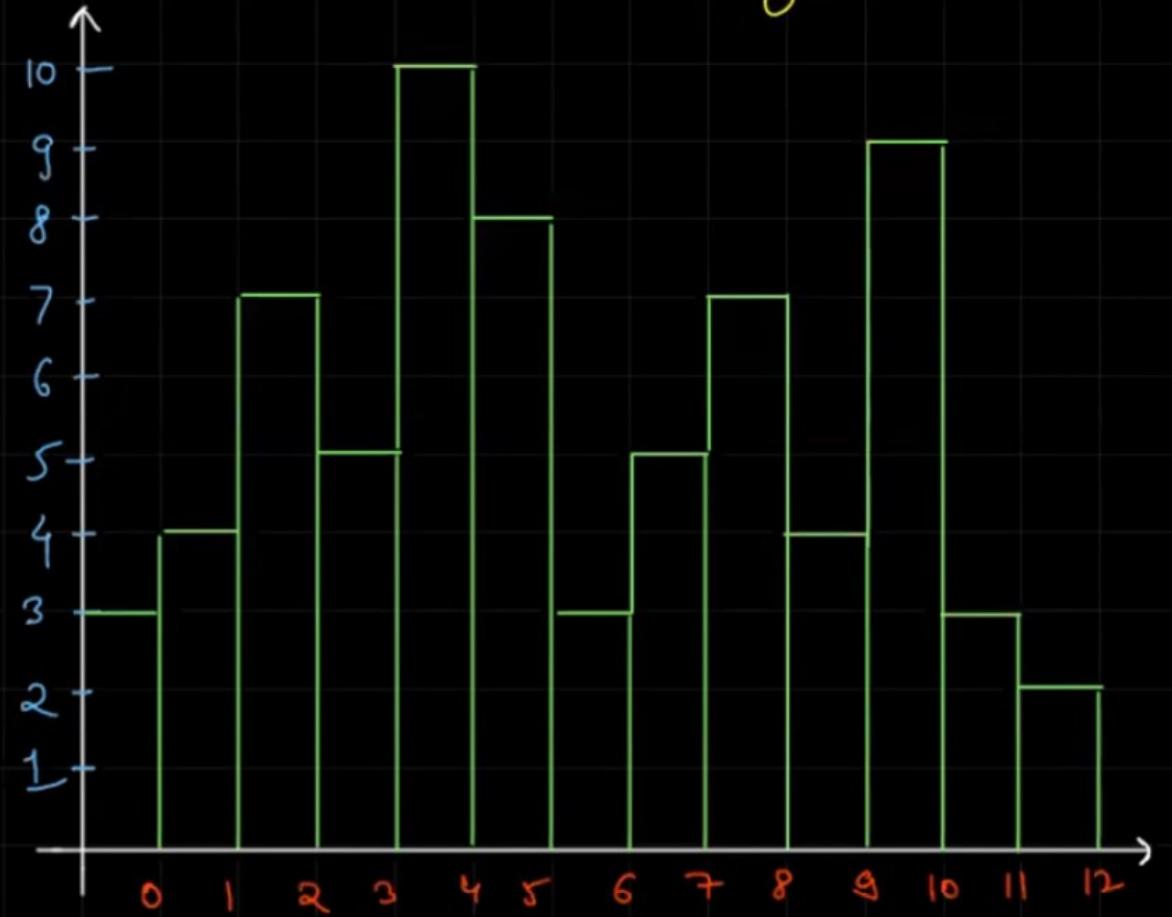
        for(int i=heights.length-2;i>=0;i--)
        {
            while(st.size()>0 && heights[st.peek()]>=heights[i])
            {
                st.pop();
            }
            if(st.size()==0)
            {
                rb[i]=heights.length;
            }
            else
            {
                rb[i]=st.peek();
            }
            st.push(i);
        }
    }
}
```

```
    st=new Stack<>();
    st.push(0);
    lb[0]=-1;
    for(int i=1;i<heights.length;i++)
    {
        while(st.size()>0 && heights[st.peek()]>=heights[i])
        {
            st.pop();
        }
        if(st.size()==0)
        {
            lb[i]=-1;
        }
        else
        {
            lb[i]=st.peek();
        }
        st.push(i);
    }

    int maxArea=0;
    for(int i=0;i<heights.length;i++)
    {
        int width=rb[i]-lb[i]-1;
        int area=heights[i]*width;
        if(area>maxArea)
        {
            maxArea=area;
        }
    }
    return maxArea;
}
```

Largest Area Histogram

Max Area Histogram



area = $\max[i] - \text{next_smaller}[i]$

q
key
* arr[i]
↑
height

at the right

Next Smaller Element
Variation.

$O(N)$ TC $O(N)$ SC

```
public static int[] solve(int[] arr){  
    int[] res = new int[arr.length];  
    Stack<Integer> st = new Stack<>();  
  
    for(int i=0;i<arr.length;i++) {  
        while(st.size() > 0 && arr[st.peek()] < arr[i]) {  
            st.pop();  
        }  
  
        if(st.size() == 0) {  
            res[i] = i + 1;  
        } else {  
            res[i] = i - st.peek();  
        }  
  
        st.push(i);  
    }  
  
    return res;  
}
```



Stock

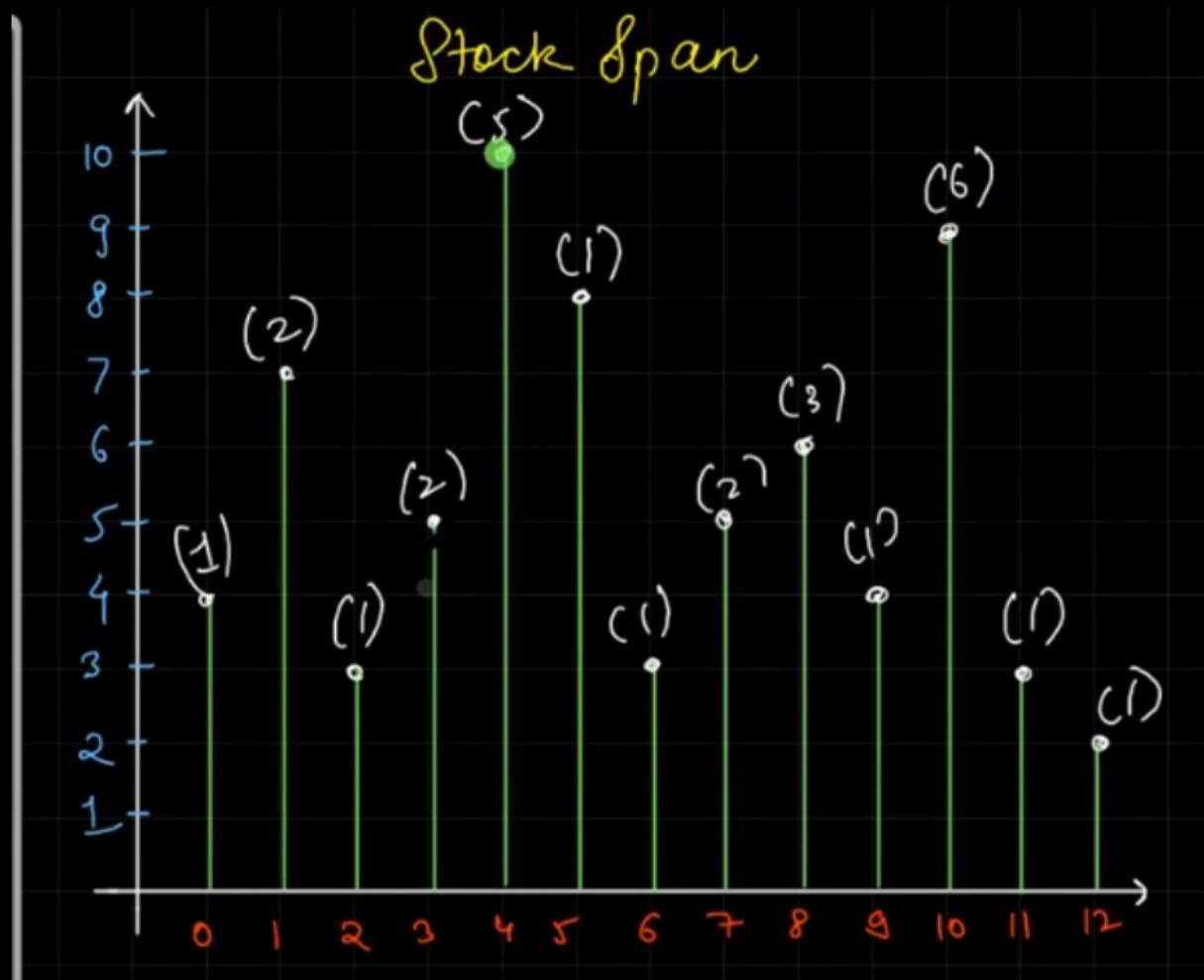
Span /

NGE to

left

Variation

Stock Span (NGE to left → variation)

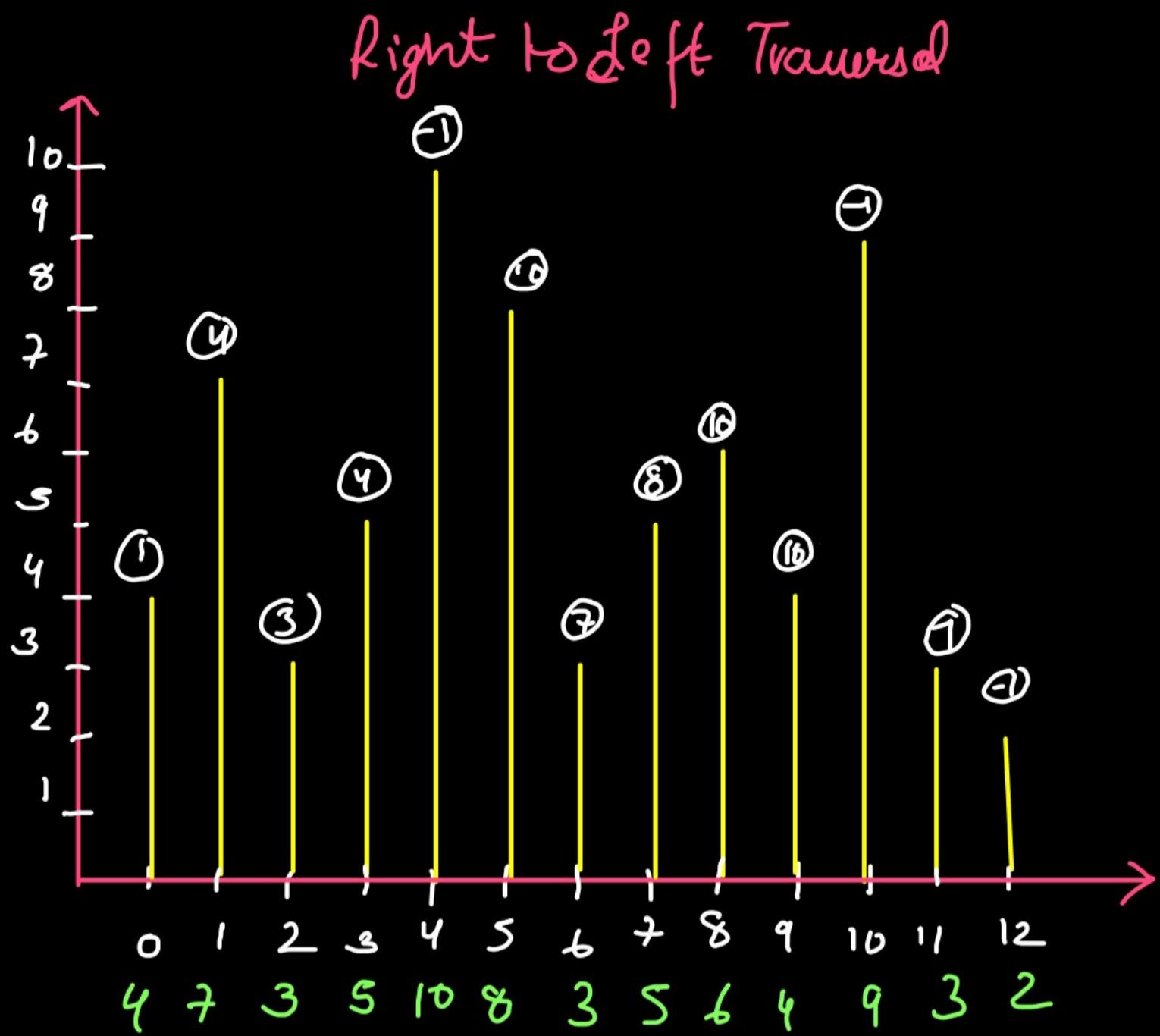


NGE to the left
house with door brac
Index wise, that is
Stock Span -

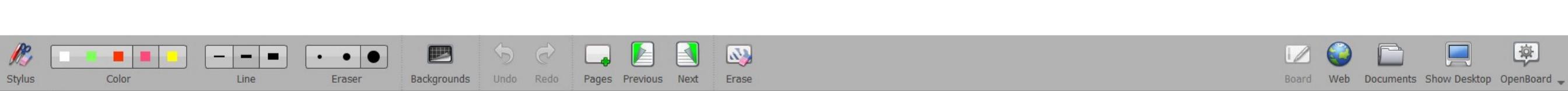


```
public static int[] solve(int[] arr){  
    int[] res = new int[arr.length];  
    Stack<Integer> st = new Stack<>();  
    res[res.length-1] = -1;  
    st.push(res.length-1);  
  
    for(int i=res.length-2;i>=0;i--) {  
        if(arr[st.peek()] > arr[i]) {  
            res[i] = arr[st.peek()];  
        } else {  
            while(st.size() > 0 && arr[st.peek()] < arr[i]) {  
                st.pop();  
            }  
            if(st.size() == 0) res[i] = -1;  
            else res[i] = arr[st.peek()];  
        }  
        st.push(i);  
    }  
  
    return res;  
}
```

Right to
left
traversal.



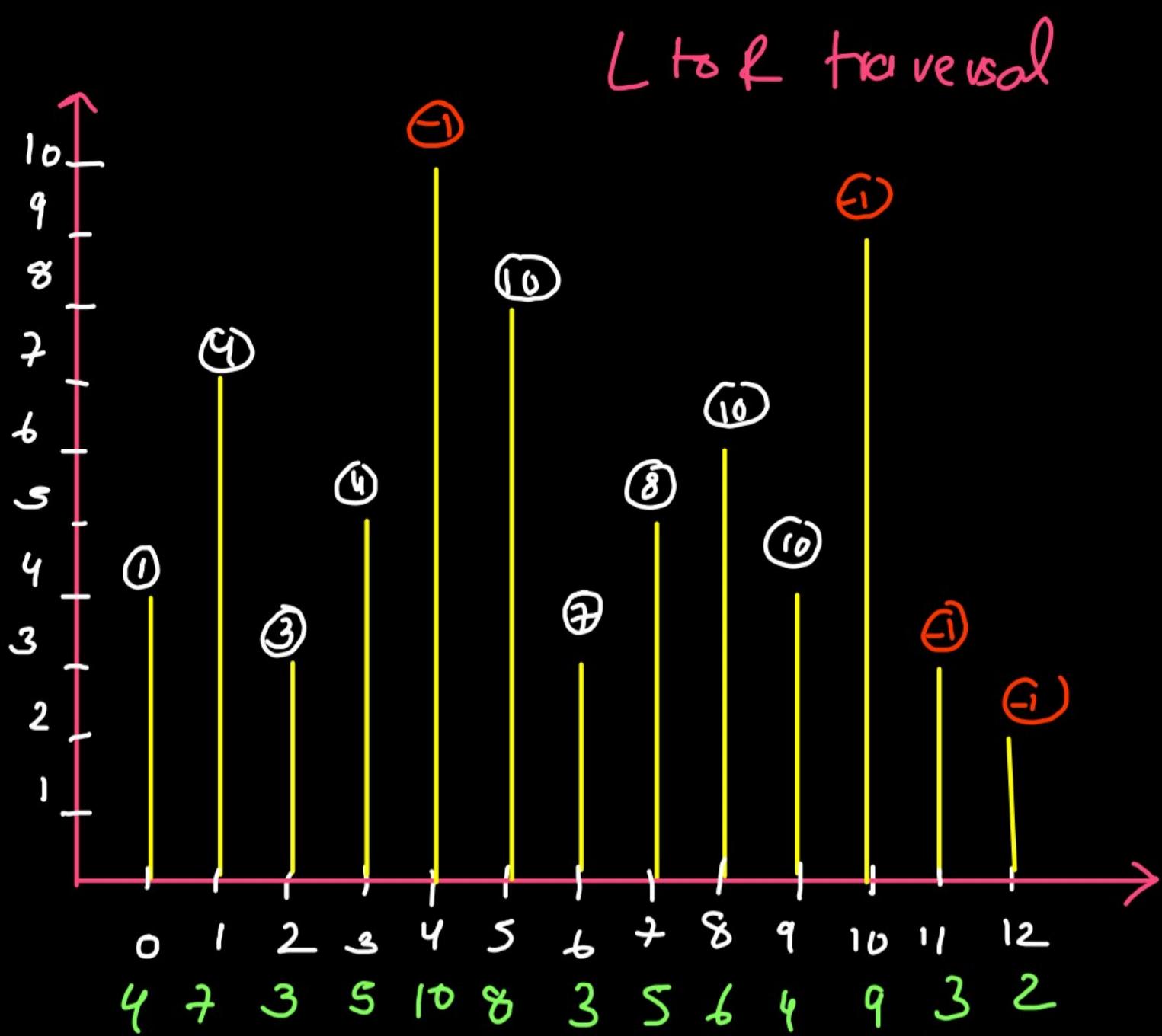
(0, 4)
~~(1, 7)~~
~~(2, 3)~~
~~(3, 5)~~
(4, 10)
~~(5, 8)~~
~~(6, 3)~~
~~(7, 5)~~
~~(8, 6)~~
~~(9, 7)~~
~~(10, 9)~~
~~(11, 3)~~
~~(12, 12)~~



```
public static int[] solve(int[] arr){  
    int[] res = new int[arr.length];  
    Arrays.fill(res,-1);  
    //we are storing indices in the stack  
    Stack<Integer> st = new Stack<>();  
    st.push(0);  
  
    for(int i=1;i<arr.length;i++) {  
        if(arr[i] < arr[st.peek()]) {  
            st.push(i);  
        } else {  
            while(st.size() > 0 && arr[st.peek()] < arr[i]) {  
                int idx = st.pop();  
                res[idx] = arr[i];  
            }  
            st.push(i);  
        }  
    }  
  
    return res;  
}
```

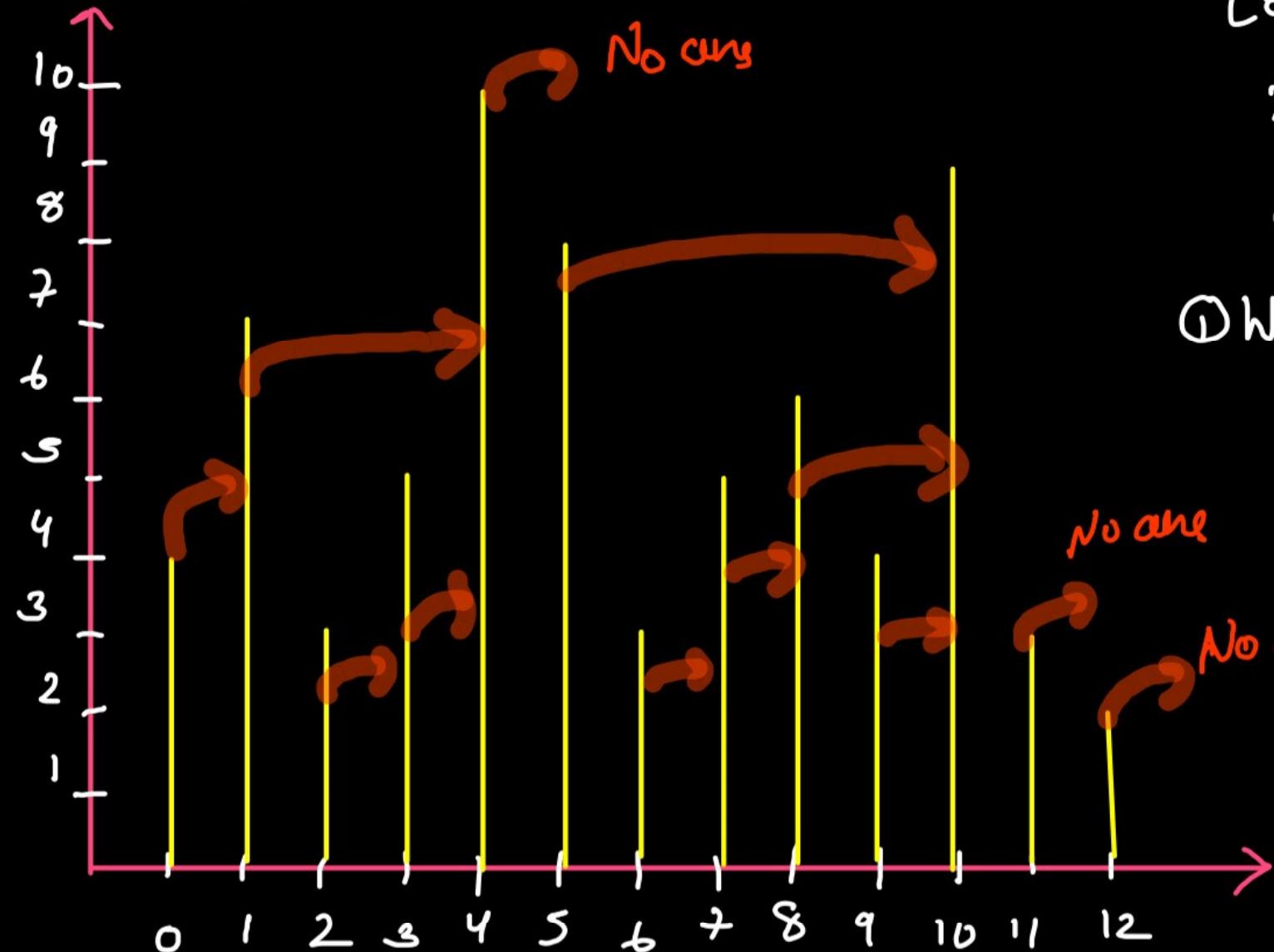
$T.C: O(n)$

$S.C: O(n)$



~~(12, 2)~~
~~(11, 3)~~
~~(10, 9)~~
~~(9, 4)~~
~~(8, 6)~~
~~(7, 5)~~
~~(6, 3)~~
~~(5, 8)~~
~~(4, 10)~~
~~(3, 5)~~
~~(2, 3)~~
~~(1, 7)~~
~~(0, 9)~~

NGE to the right



Left to right
traversal alg
on next page -

- ① While (`st.pop() < cur ele`)
pop
- ② push
cur ele



```
public static boolean solve(String str) {  
    Stack<Character> st = new Stack<>();  
    for(int i=0;i<str.length();i++) {  
        char ch = str.charAt(i);  
        if(ch == ')') {  
            if(st.peek() == '(') {  
                return true;  
            }  
  
            while(st.peek() != '(') {  
                st.pop();  
            }  
  
            st.pop();  
        } else st.push(ch);  
    }  
  
    return false;  
}
```

Duplicate brackets / Redundant / Needless { Brackets will be balanced for sure }

Redundant / Duplicate / Needless / Extra Brackets

$$((a+b) \cdot c) + d$$

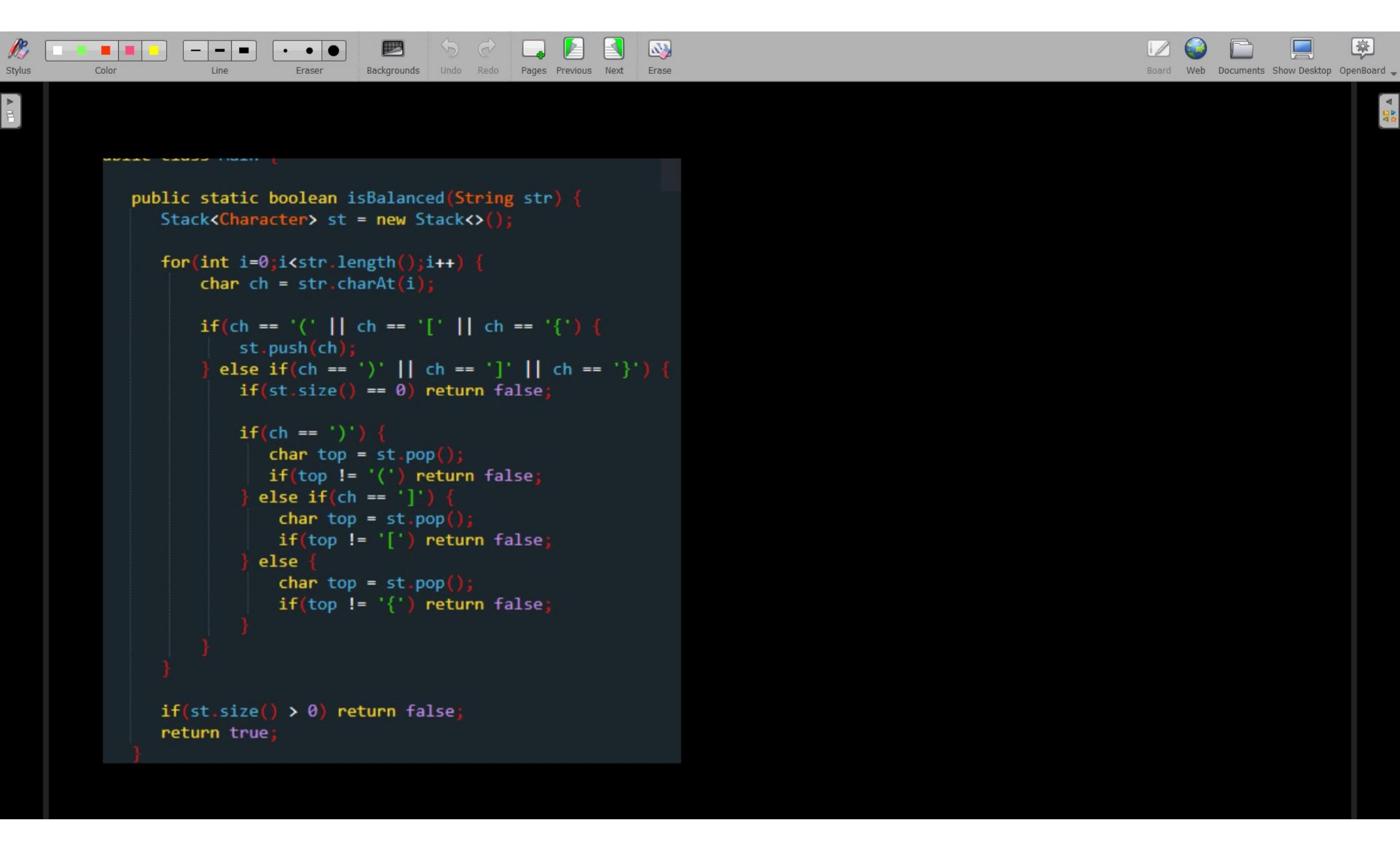
$$(a+b) \cdot (c+d)$$

$$(a+b) \cdot ((c+d))$$

"No independent content"

yes

needless/
redundant/
duplicate



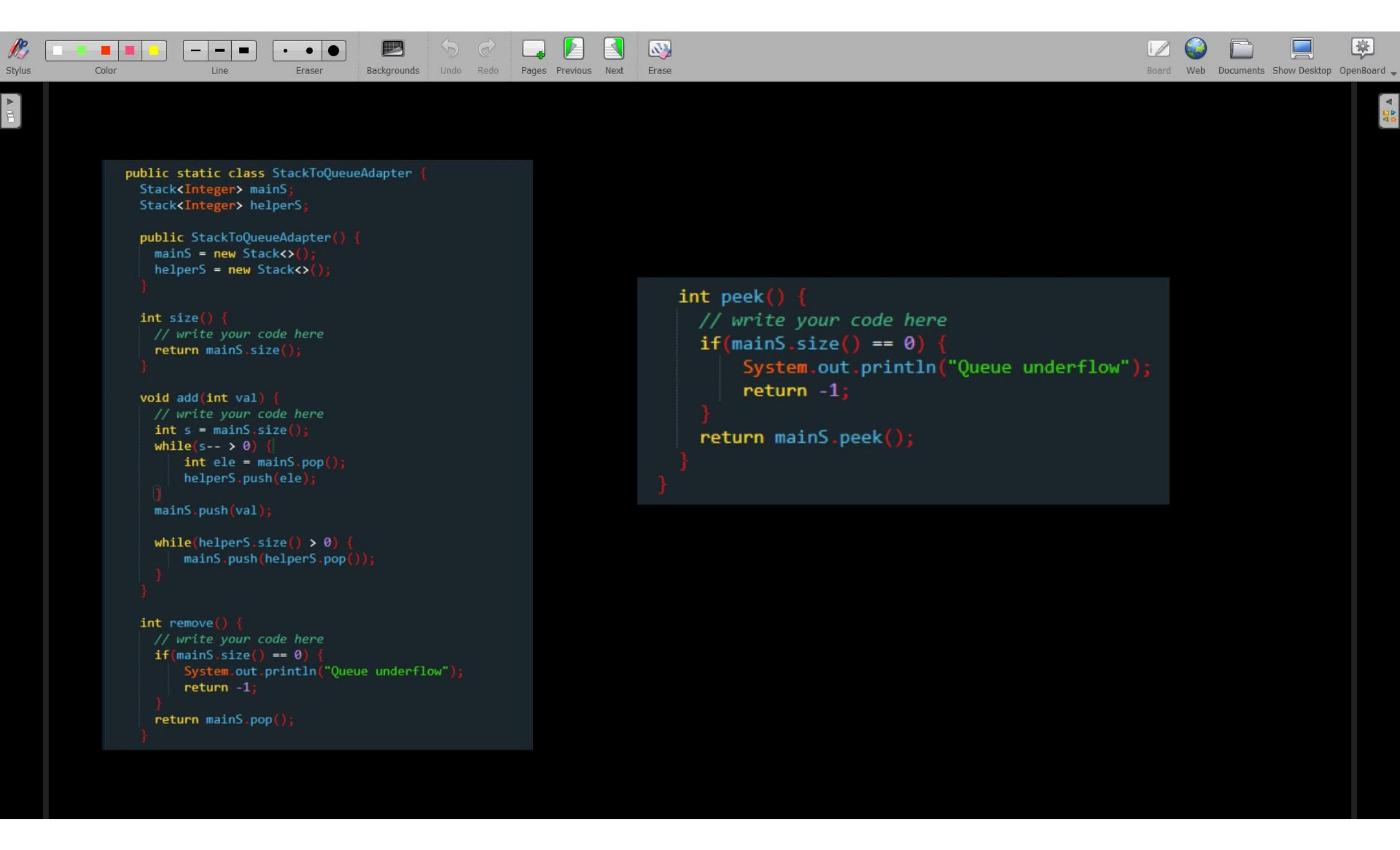
Balanced Brackets



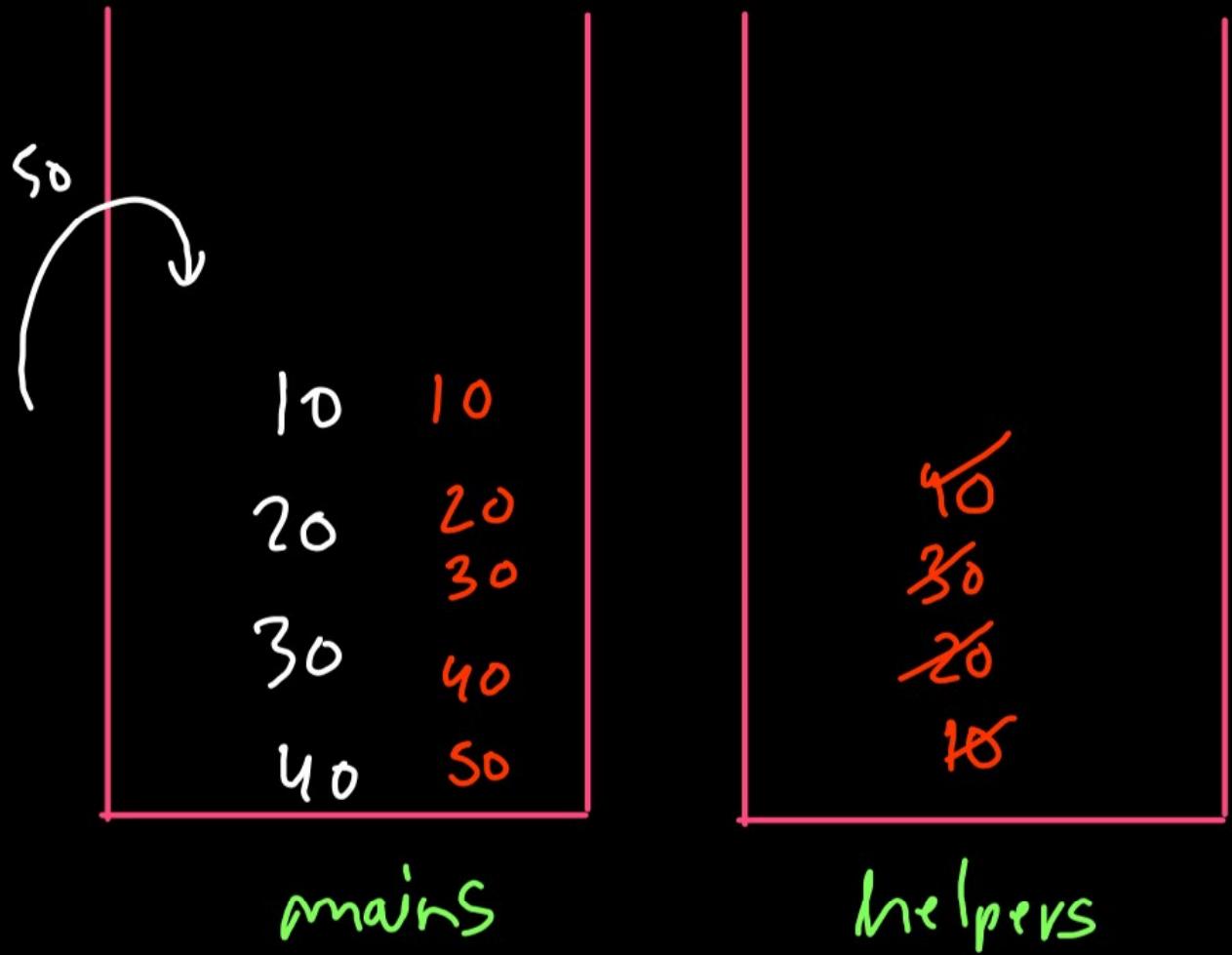
```
[(a + b) + {(c + d) * (e / f)}] -> true  
[(a + b) + {(c + d) * (e / f)}]} -> false  
[(a + b) + {(c + d) * (e / f)}] -> false  
([(a + b) + {(c + d) * (e / f)}]) -> false
```

opening bracket → push

closing → if st. empty → return false
else check if corresponding
opening bracket is at the top if yes
then pop it, if no → return false.

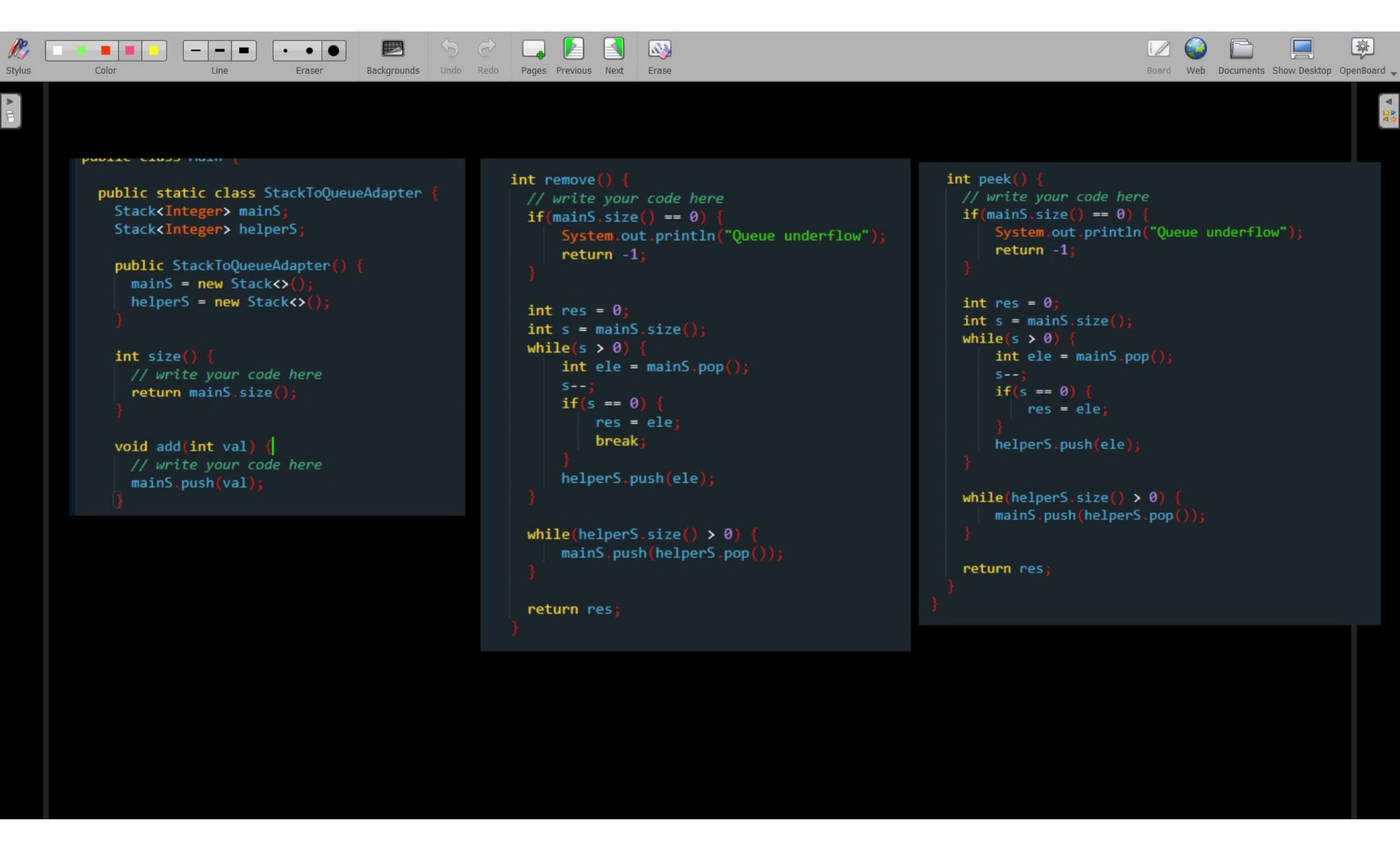


Queue using Stack \rightarrow Remove Efficient

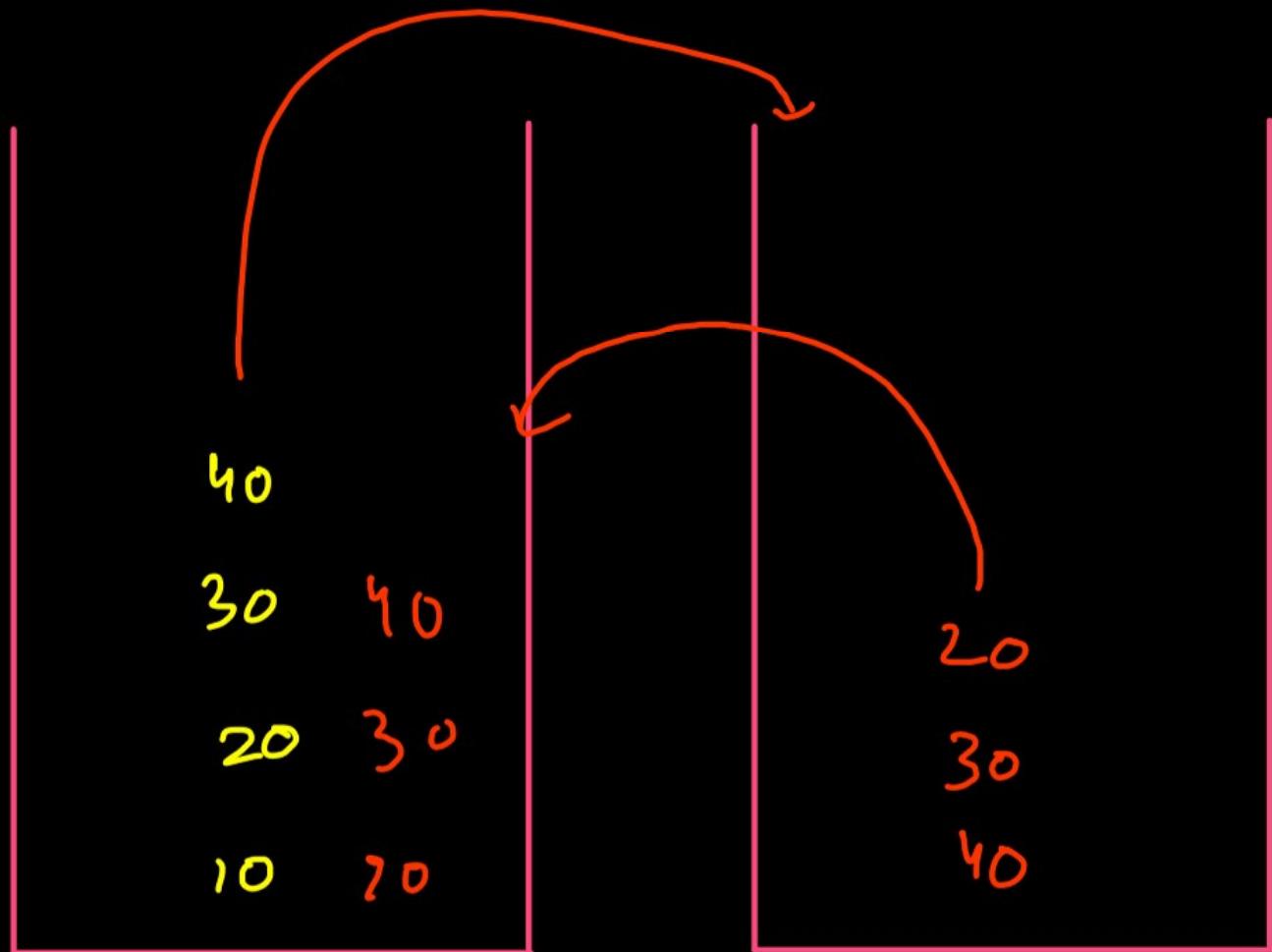


10 20 30 40 50

remove \rightarrow pop



Queue using Stack - Add Efficient



~~10 20 30 40~~

add → ms.push()

remove → pop ele from
ms & add it

to hs

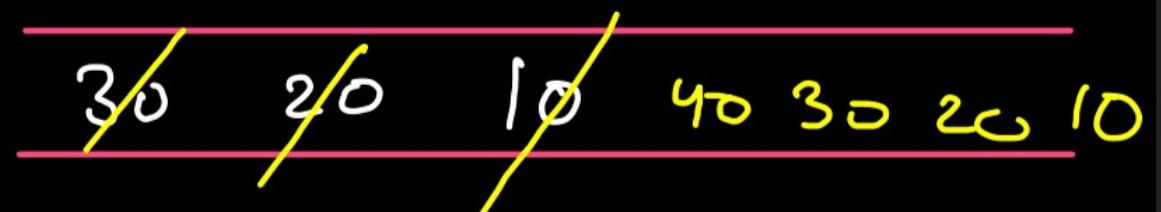
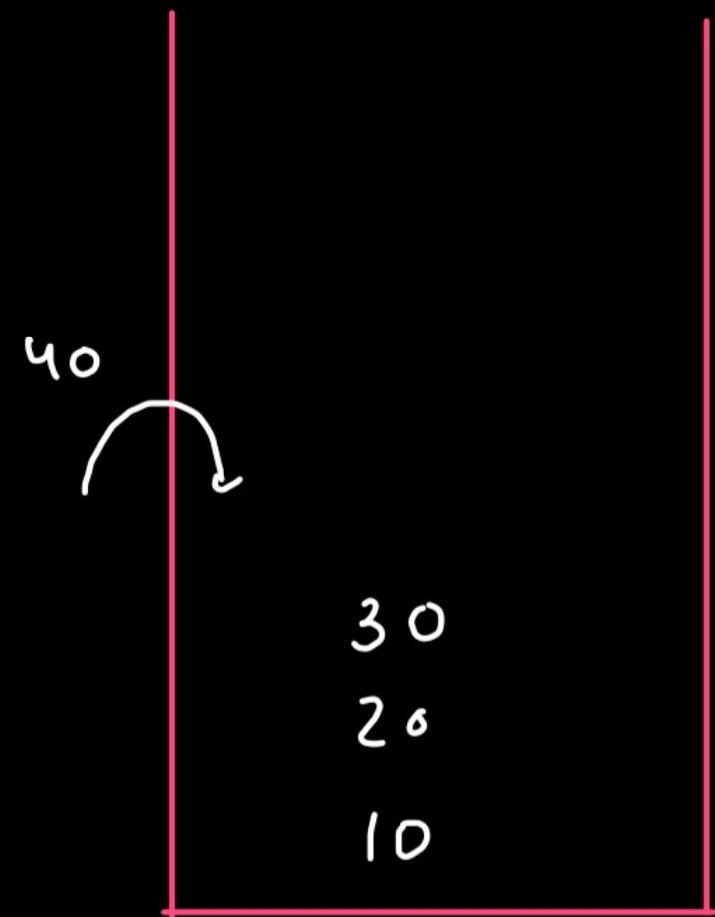
for every ele except
last remaining
pop ele from hs to ms.

```
public static class QueueToStackAdapter {  
    Queue<Integer> q;  
  
    public QueueToStackAdapter() {  
        q = new ArrayDeque<>();  
    }  
  
    int size() {  
        // write your code here  
        return q.size();  
    }  
  
    void push(int val) {  
        // write your code here  
        int s = q.size();  
  
        if(q.size() == 0) {  
            q.add(val);  
            return;  
        }  
  
        for(int i=1;i<=s;i++) {  
            int ele = q.remove();  
            if(i == 1) {  
                q.add(val);  
            }  
            q.add(ele);  
        }  
    }  
}
```

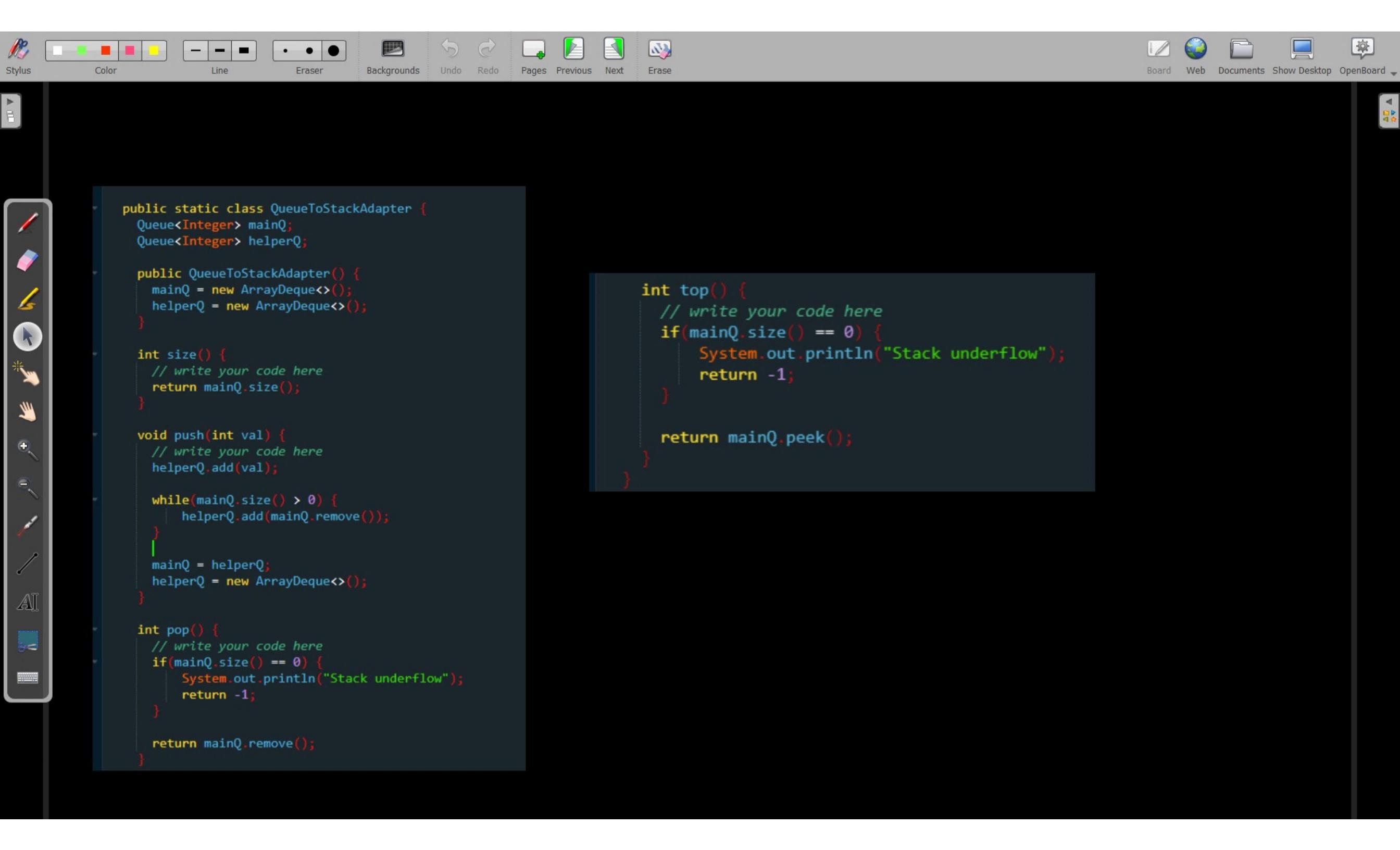
```
int pop() {  
    // write your code here  
    if(q.size() == 0) {  
        System.out.println("Stack underflow");  
        return -1;  
    }  
  
    return q.remove();  
}  
  
int top() {  
    // write your code here  
    if(q.size() == 0) {  
        System.out.println("Stack underflow");  
        return -1;  
    }  
  
    return q.peek();  
}
```

Stack using 1 queue
pop efficient

Stack Using 1 Queue → pop Efficient



When removing the first ele,
add curr ele first -

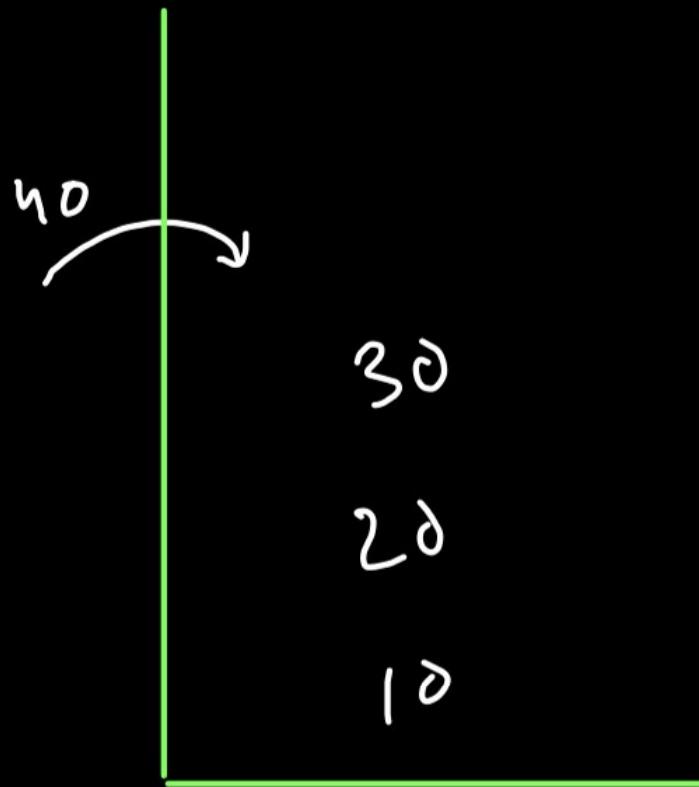


Stack using Queue \rightarrow Pop Efficient.

$\text{push} \rightarrow 1 \text{ helperQ.add(ele)}$
 $2 \text{ mainQ} \rightarrow \text{all pop \& push in helperQ}$
 $3 \text{ mainQ} = \text{helperQ}$

mainQ

30 20 10



helperQ

40 30 20 10

$\text{pop} \rightarrow \text{mainQ.remove()}$
 $\text{peek} \rightarrow \text{mainQ.peek()}$

The image shows a digital whiteboard interface with various tools and icons at the top. On the left, there's a vertical toolbar with icons for Stylus, Color, Line, Eraser, Backgrounds, Undo, Redo, Pages, Previous, Next, and Erase. The main area contains two code snippets in a monospaced font.

Left Snippet (Interface):

```
public static class QueueToStackAdapter {  
    Queue<Integer> q;  
  
    public QueueToStackAdapter() {  
        q = new ArrayDeque<>();  
    }  
  
    int size() {  
        // write your code here  
        return q.size();  
    }  
  
    void push(int val) {  
        // write your code here  
        q.add(val);  
    }  
  
    int pop() {  
        // write your code here  
        if(q.size() == 0) {  
            System.out.println("Stack underflow");  
            return -1;  
        }  
        int s = q.size();  
        int res = 0;  
        while(s > 0) {  
            int ele = q.remove();  
            s--;  
            if(s == 0) {  
                res = ele;  
                break;  
            }  
            q.add(ele);  
        }  
        return res;  
    }  
}
```

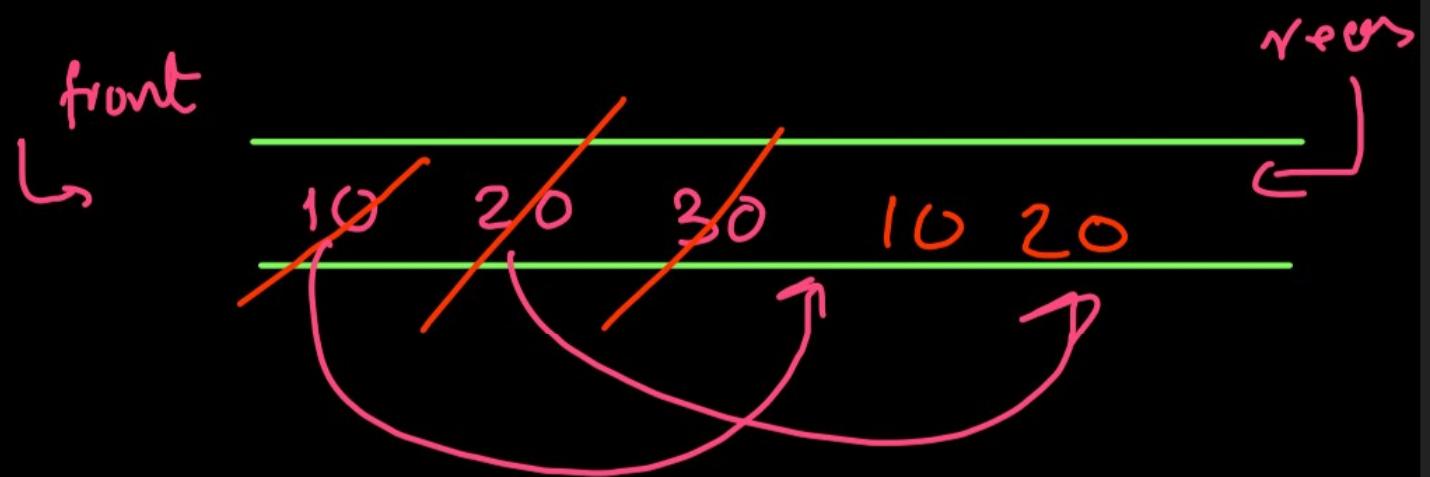
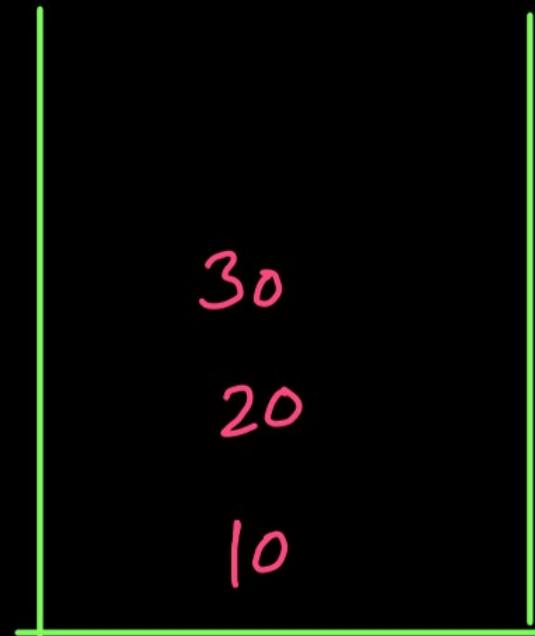
Right Snippet (Implementation):

```
int top() {  
    // write your code here  
    if(q.size() == 0) {  
        System.out.println("Stack underflow");  
        return -1;  
    }  
    int s = q.size();  
    int res = 0;  
    while(s > 0) {  
        int ele = q.remove();  
        s--;  
        if(s == 0) res = ele;  
        q.add(ele);  
    }  
    return res;  
}
```

Handwritten Annotations:

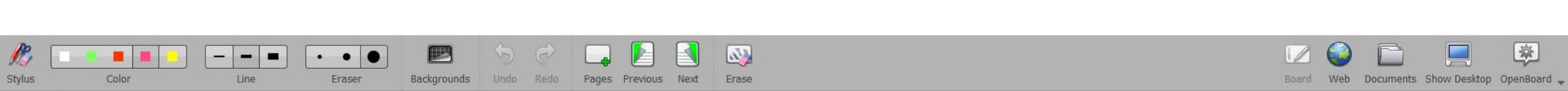
- A large curly brace on the right side groups the entire right-hand code block and spans from the `int top()` declaration down to the final closing brace of the method.
- To the right of this brace, the word "Stack" is written vertically.
- Below "Stack", the word "using" is written vertically.
- Below "using", the word "1 Queue" is written horizontally.
- On the far right, below the brace, the word "Push" is written vertically.
- At the bottom right, the word "Efficient." is written horizontally.

Push Efficient using 1 Queue



push → q.add

removal → remove & add ?? for all
else.



When we make push efficient, pop and peek() of
Stack become $O(N)$ both.

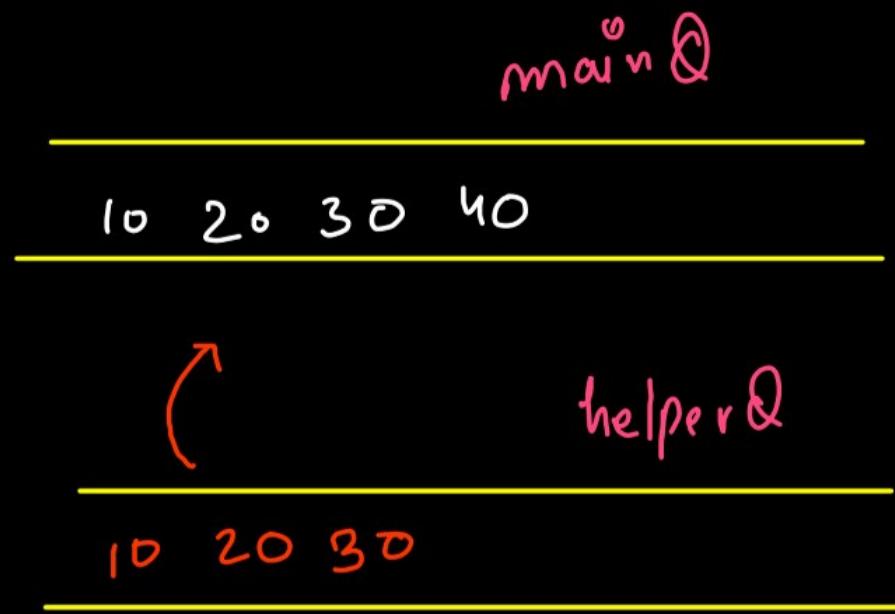
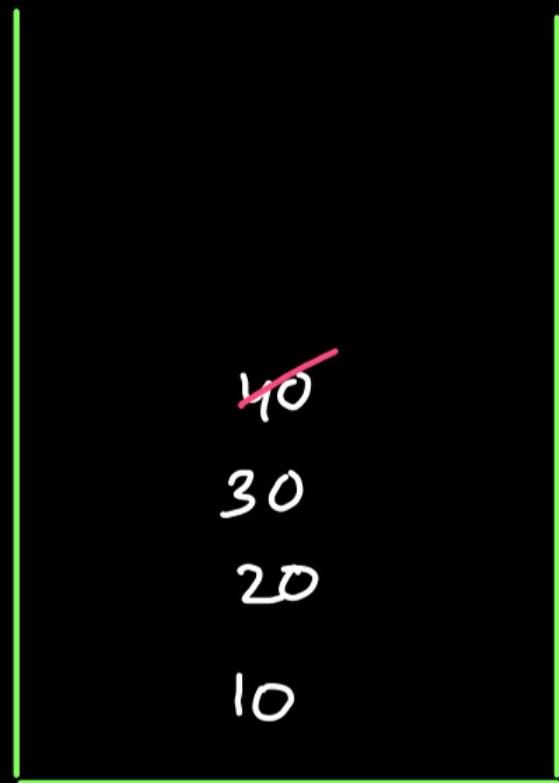
```
public static class QueueToStackAdapter {  
    Queue<Integer> mainQ;  
    Queue<Integer> helperQ;  
  
    public QueueToStackAdapter() {  
        mainQ = new ArrayDeque<>();  
        helperQ = new ArrayDeque<>();  
    }  
  
    int size() {  
        // write your code here  
        return mainQ.size();  
    }  
  
    void push(int val) {  
        // write your code here  
        mainQ.add(val);  
    }  
}
```

```
int pop() {  
    // write your code here  
  
    if(mainQ.size() == 0) {  
        System.out.println("Queue underflow");  
        return -1;  
    }  
  
    int res = -1;  
    while(mainQ.size() > 0) {  
        int ele = mainQ.remove();  
        if(mainQ.size() == 0) {  
            res = ele;  
            break;  
        }  
        helperQ.add(ele);  
    }  
  
    mainQ = helperQ;  
    helperQ = new ArrayDeque<>();  
    return res;  
}
```

```
int top() {  
    // write your code here  
    if(mainQ.size() == 0) {  
        System.out.println("Queue underflow");  
        return -1;  
    }  
  
    int res = -1;  
    while(mainQ.size() > 0) {  
        int ele = mainQ.remove();  
        if(mainQ.size() == 0) {  
            res = ele;  
        }  
        helperQ.add(ele);  
    }  
  
    mainQ = helperQ;  
    helperQ = new ArrayDeque<>();  
    return res;  
}
```

Stack using Queue \rightarrow Push Efficient { Strictly a Queue
Not a Dequeue }

push \rightarrow q.add



pop \rightarrow mainQ.remove & add helperQ
except last ele.
mainQ = helperQ -

top \rightarrow Same as pop, just add the last ele too.



Dynamic Queue

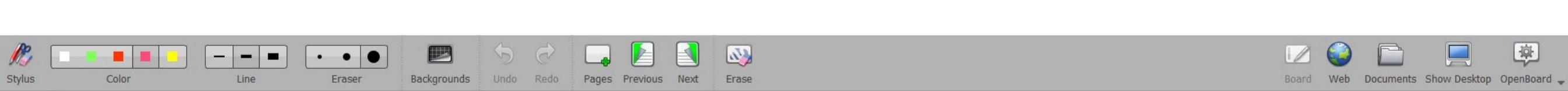
doubling

```
// change this code
void add(int val) {
    // write ur code here
    if(size == data.length){
        int n = data.length;
        int[] temp = new int[2*n];

        for(int i = 0; i < size; i++){
            int idx = (front + i) % data.length;
            temp[idx] = data[idx];
        }

        data = new int[2*n];
        data = temp;

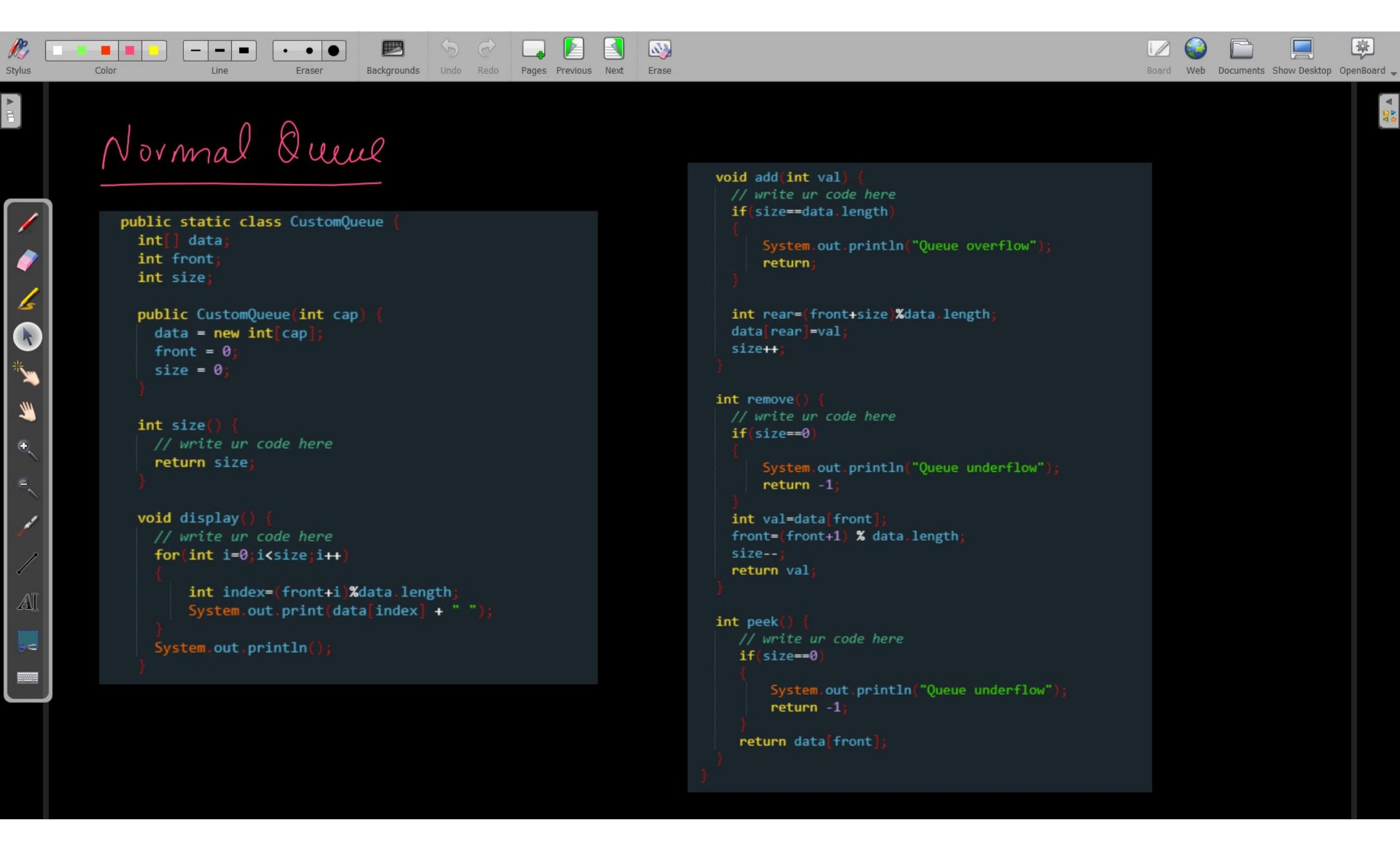
        int idx = (front + size) % data.length;
        data[idx] = val;
        size++;
    } else {
        int idx = (front + size) % data.length;
        data[idx] = val;
        size++;
    }
}
```



Leetcode → Implement Circular Queue -

```
class MyCircularQueue {  
  
    int front;  
    int rear;  
    int size;  
    int []arr;  
  
    public MyCircularQueue(int k) {  
        arr = new int[k];  
        size = 0;  
        front = rear = 0;  
    }  
  
    public boolean enqueue(int value) {  
        if(size == arr.length) return false;  
        arr[rear] = value;  
        rear = (rear + 1)%arr.length;  
        size++;  
        return true;  
    }  
  
    public boolean dequeue() {  
        if(size == 0) 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];  
        //this is because rear vala element rear se ek index peeche hai  
        //aur agar index 0 par rear hua to rear-1 karne se problem create ho  
        //jaegi. Isliye humne arr.length ko add krva liya  
    }  
  
    public boolean isEmpty() {  
        if(size == 0) return true;  
        return false;  
    }  
  
    public boolean isFull() {  
        if(size == arr.length) return true;  
        return false;  
    }  
}
```





Two Stacks in an Array -

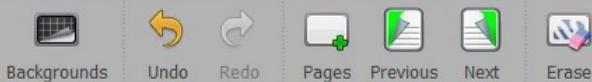
```
class Stacks
{
    //Function to push an integer into the stack1.
    void push1(int x, TwoStack sq)
    {
        if(sq.top1 == sq.top2 - 1) return;
        sq.top1++;
        sq.arr[sq.top1] = x;
    }

    //Function to push an integer into the stack2.
    void push2(int x, TwoStack sq)
    {
        if(sq.top1 == sq.top2 - 1) return;
        sq.top2--;
        sq.arr[sq.top2] = x;
    }

    //Function to remove an element from top of the stack1.
    int pop1(TwoStack sq)
    {
        if(sq.top1 == -1) return -1;
        int x = sq.arr[sq.top1];
        sq.top1--;
        return x;
    }

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





Queue using LL

```
public static class LLToQueueAdapter {
    LinkedList<Integer> list;

    public LLToQueueAdapter() {
        list = new LinkedList<>();
    }

    int size() {
        // write your code here
        return list.size();
    }

    void add(int val) {
        // write your code here
        list.addLast(val);
    }

    int remove() {
        // write your code here
        if(list.size() == 0) {
            System.out.println("Queue underflow");
            return -1;
        }

        return list.removeFirst();
    }

    int peek() {
        // write your code here
        if(list.size() == 0) {
            System.out.println("Queue underflow");
            return -1;
        }

        return list.getFirst();
    }
}
```

if tail also available

add last → new add

remove first → remove

↓

removeLast will be $O(N)$ always
as we need prev pointer



Stack using LL

```
public static class LLToStackAdapter {  
    LinkedList<Integer> list;  
  
    public LLToStackAdapter() {  
        list = new LinkedList<>();  
    }  
  
    int size() {  
        // write your code here  
        return list.size();  
    }  
  
    void push(int val) {  
        // write your code here  
        list.addFirst(val);  
    }  
  
    int pop() {  
        // write your code here  
        if(list.size() == 0) {  
            System.out.println("Stack underflow");  
            return -1;  
        }  
  
        return list.removeFirst();  
    }  
  
    int top() {  
        // write your code here  
        if(list.size() == 0) {  
            System.out.println("Stack underflow");  
            return -1;  
        }  
  
        return list.getFirst();  
    }  
}
```

push → add first } O(1)

pop → remove first

Dynamic Stack

```
// change the code of this function accordingly
void push(int val) {
    if (tos == data.length - 1) {
        int n = data.length;
        int[] arr = new int[n * 2];
        for(int i=0;i<n;i++) {
            arr[i] = data[i];
        }

        data = new int[2*n];

        for(int i=0;i<n;i++) {
            data[i] = arr[i];
        }

        tos = n;
        data[tos] = val;
    } else {
        tos++;
        data[tos] = val;
    }
}
```

Here amortized time

complexity of push is $O(1)$

doubling

avg

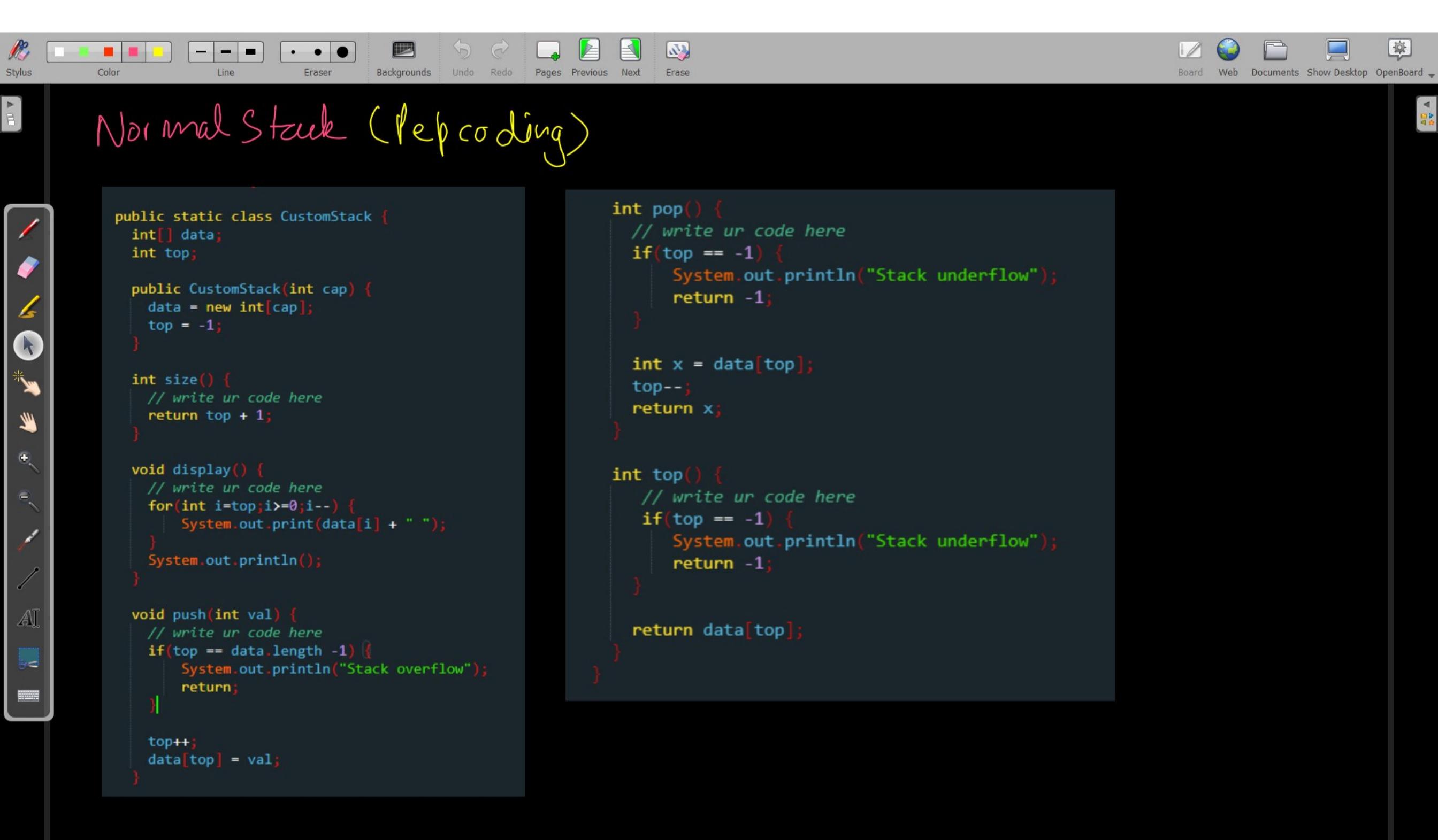
Σ Push Time

Σ Push

↓

no of times

push is performed





Queue is an interface hence it cannot be instantiated

Hence, we use ArrayDeque for creating the object while
the reference is of Queue only-

→ Stack is by default implemented using arrayList.

