



Stacks → linear data structure

}



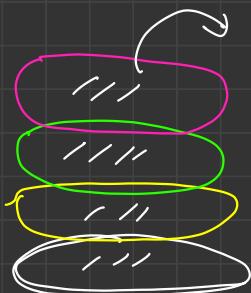
- ① Stack is a linear DS



Stack of plates



Stack Operations



1. addition of new data in a stack
is done on top

2. deletion of data from a stack
is done from top

3. data on top is only readable

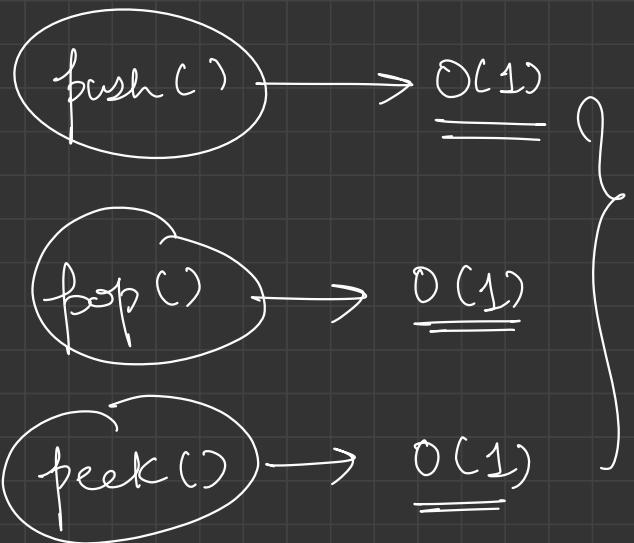
- { ① push() : add data in the stack
(Top of the Stack)
- { ② pop() : removes topmost element from the
stack .
- ③ peek() : See data present on the top of the stack

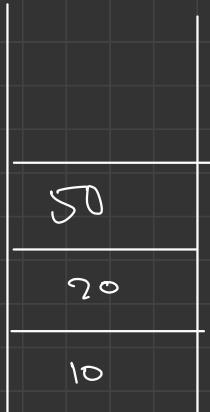
Stack < Integer > st = new Stack <>();



{ Integer, Float, Character, Long }
↓
Wrapper classes

OR
our own
defined
classes





St

St. peek() \rightarrow 30

St. pop()

St. peek() \rightarrow 20

St. push(50)

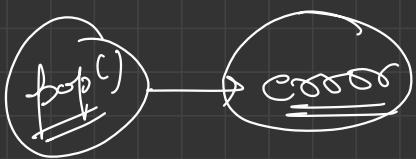
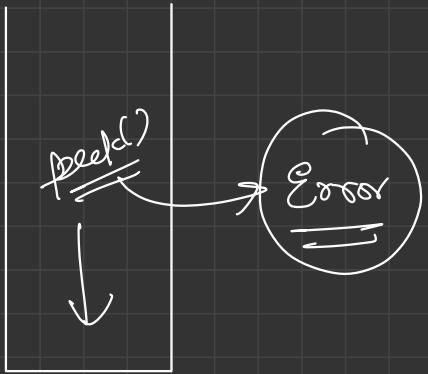
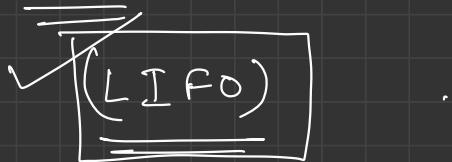
St. peek() \rightarrow 50

St. size() \rightarrow 3

① size()

size of the stack

Stack follows, last in, but first out .



stack underflow

ST

push (20)

push (50)

push (40)

push (30)

pop ()

50

50
40
30
20

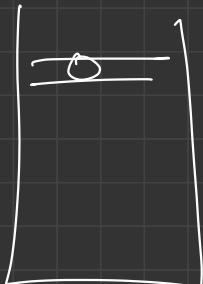
(last person in , first out)

ArrayList

- ① add at last
- ② get any element
- ③ remove any element
- ④ size
- ⑤ Replace any element

Stack

- ① push() on top
- ② pop() from top
- ③ peek() See on top()
- ④ size of stack



linked list



add last()

add first()

remove last()

remove first()

size of linked list()



Q

$$\underline{(a + (b \times c) - (d \times f))} \rightarrow \underline{\text{false}} \quad (\text{No extra brackets})$$

$$(a + ((b + c)) - (d \times f)) \rightarrow \text{true}$$

extra bracket

* Every bracket is balanced and we don't have
any extra bracket !

$$((a+b) \times (c+d))$$
$$\downarrow \qquad \qquad \qquad \downarrow$$
$$(A \times B)$$
$$\downarrow \qquad \qquad \qquad \downarrow$$
$$C$$

$$(((a+b) \times (c+d)))$$
$$\overbrace{\qquad\qquad}^{\qquad\qquad} \qquad \qquad \overbrace{\qquad\qquad}^{\qquad\qquad}$$
$$\downarrow \qquad \qquad \qquad \downarrow$$
$$((A \times B))$$
$$\downarrow$$
$$\overbrace{C}^{\qquad\qquad}$$

Not Required

~~A A A A A A A A A A A A~~

$(a + (b \times c) - (d \times f))$

① any operator, or opening bracket,
possible part of a exp.

$(+)$ \rightarrow 'c' or operator char.

—
+
a
(

Last person opened has to
be closed first.

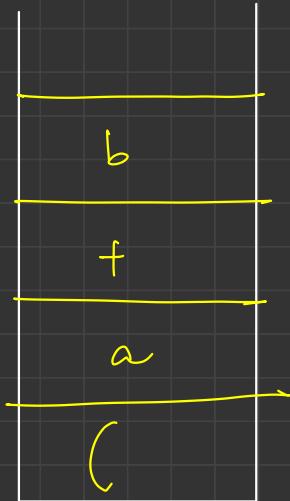
$(LIFO)$

$(-)$ \rightarrow ')'
search for corresponding st
'(bracket in stack
if st had some value
use '(bracket' in stnd.

means, we have an exp.

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

~~a b + c d + c~~



St

```
if (ch == ')')
{
    st.push(ch);
}
```

```
else
{
    if (st.peek() == '(')
        return true;
```

else

```
{
    while(st.peek() != '(')
        st.pop();
    st.pop(); → corresponding opening
```



```

public boolean ExtraBrackets(String exp) {
    // Write your code here
    // true when extra bracket else false

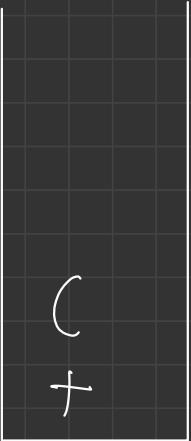
    Stack<Character> st = new Stack<>();
    for (int i = 0; i < exp.length(); i++) {
        char ch = exp.charAt(i);

        // if '(' or ')' or Operator -> push in stack
        if (ch != ')') {
            st.push(ch);
        }

        else {
            // we got a closing bracket
            if (st.peek() == '(') {
                // no expression in between
                return true;
            } else {
                // removal of exp in between
                while (st.peek() != '(') {
                    st.pop();
                }
                // corresponding opening bracket for a closing bracket with exp in between
                st.pop();
            }
        }
    }
    return false;
}

```

ex² $(a+b) + ((c+d))$



st

TC : $O(N)$]
SC : $O(N)$]

$a+b+c+d$

(5)

for ($i = 0 \rightarrow n$)

}

while()

$\boxed{N \text{ times in its life span}}$

g

O(2N)

$i = 0, 1, 2, 3, 4$

↓ ↓ ↓ ↓ ↓

- - - - -

$N \text{ times}$

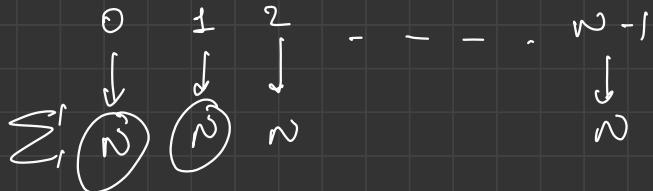
$O(N^2)$

for ($i \rightarrow 0 \rightarrow n$)

}

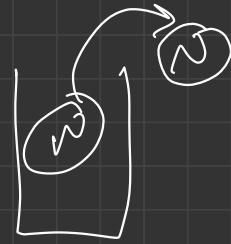
 while ($\leq N$)

}



$$= N + N + N + \dots + N \text{ times}$$

$\overbrace{N \times N} = \underline{\underline{N^2}}$



$O(N)$

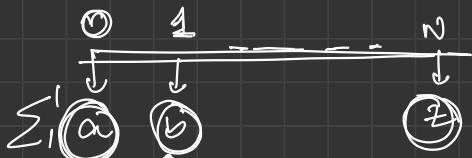
for ($i = 0 \rightarrow n$)

}

 while ($i < n$)

}

(N time in life)



= $a + b + \dots + z$ operator

$\Rightarrow \underline{\underline{N}} \text{ times}$

Next Greater Element on Right

arr[] = [2, 5, 9, 3, 1, 12, 6, 8, 7]

nger[] : [5, 9, 12, 12, 12, -1, 8, -1, -1]

Solve? ① BruteForce

→ int[] nger = new - - -

for (int i=0; i < n)

{ for (int j=i+1; j < n)

if (arr[j] > arr[i])

{ nger[i] = arr[j]
break;

$$\sum \text{TC} \stackrel{\text{def}}{=} \underline{\underline{O(N^2)}}$$

$$\sum (N-1) + (N-2) + \dots + 1$$

$$= \frac{(N-1)N}{2} = \underline{\underline{O(N^2)}}$$

$\text{Sc: } O(1)$
 $\text{TC: } O(N^2)$



$\text{arr}[] = [2, 5, 9, 3, 1, 12, 6, 8, 7]$

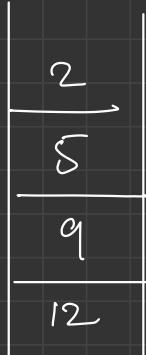
$\text{nger}[] = [5, 9, 12, 12, 12, -1, 8, -1, -1]$

Approach 1

① traverse array from last to start.

$\begin{array}{ccccccccc} \downarrow & \downarrow & \downarrow & \swarrow & \nwarrow & \searrow & \nearrow & \searrow & \downarrow \\ [2, 5, 9, 3, 1, 12, 6, 8, 7] \\ \rightarrow [5, 9, 12, 12, 12, -1, 8, -1, -1] \end{array}$

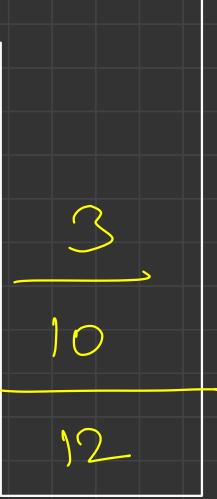
{
 if ($\text{st.size}() == 0$)
 $\text{nger}[i] = -1$;
 else
 remove element from the stack
 if ($\text{st.size} == 0$) $\text{nger}[i] = -1$ till anything bigger
 else $\text{nger}[i] = \text{st.pop}();$
 $\text{st.push}(\text{arr}[i])$;



st

10 12 4 5 [5 9 12 12 -1 8 -1 -1
 3 10, 1, 4 [2, 5 9, 3, 1, 12, 6, 8, 7]

```
class Solution {
    public static long[] nextLargerElement(long[] arr, int n) {
        // Write code here and print output
        long[] nger = new long[n];
        Stack<Long> st = new Stack<>();
        for (int i = n - 1; i >= 0; i--) { → N + 2 loops
            if (st.size() == 0) {
                nger[i] = -1;
            } else {
                while (st.size() > 0 && st.peek() <= arr[i]) {
                    st.pop(); ← N time
                }
                if (st.size() == 0) {
                    nger[i] = -1; ✓
                } else {
                    nger[i] = st.peek(); ← life time
                }
            }
            st.push(arr[i]); ←
        }
        return nger;
    }
}
```



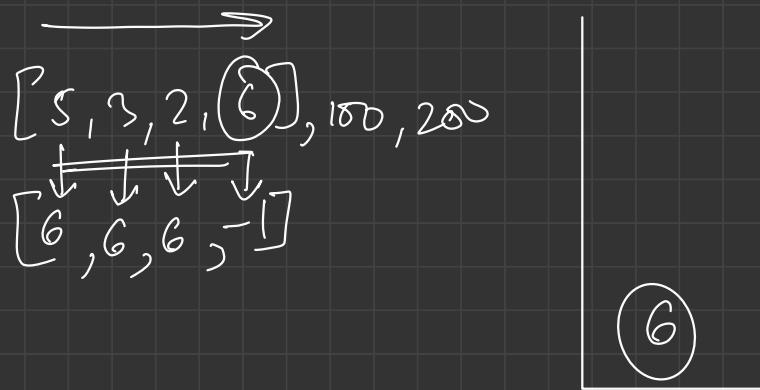
Time Complexity : $O(N)$

Space Complexity : $O(N)$ → Stack Space

Stack

$\text{arr}[] = [2, 5, 9, 3, 1, 12, 6, 8, 7]$

Approach 2



push → when topmost element of stack is greater than you

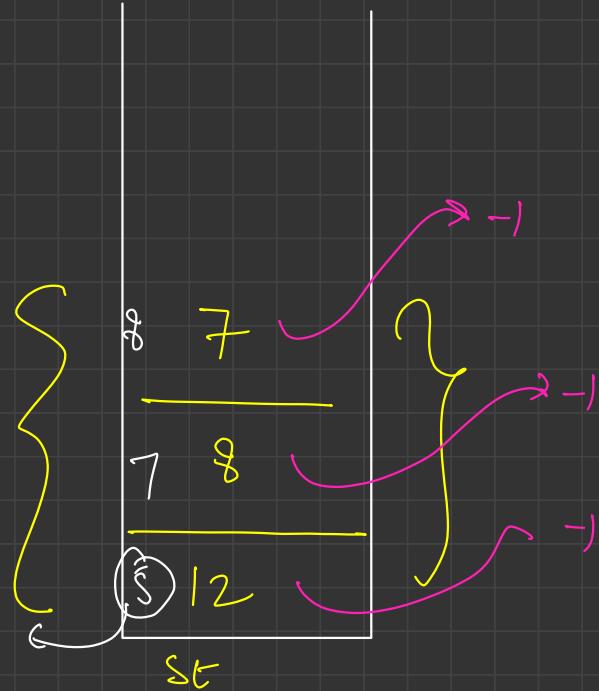
pop() → when some greater than me was encountered,

$\text{arr}[] = [5, 9, 12, 12, 12, -1, 8, -1, -]$

(+) when stack top is greater than me, or empty.

(-) when curr ele, is greater than peek(), and curr is peek's neighbor.

$$\underline{\underline{\text{ngtr}(8)}} = -1$$



```

// Approach 2:
public static long[] nextLargerElement(long[] arr, int n) {
    // Write code here and print output
    long[] nger = new long[n];
    Stack<Integer> st = new Stack<>();

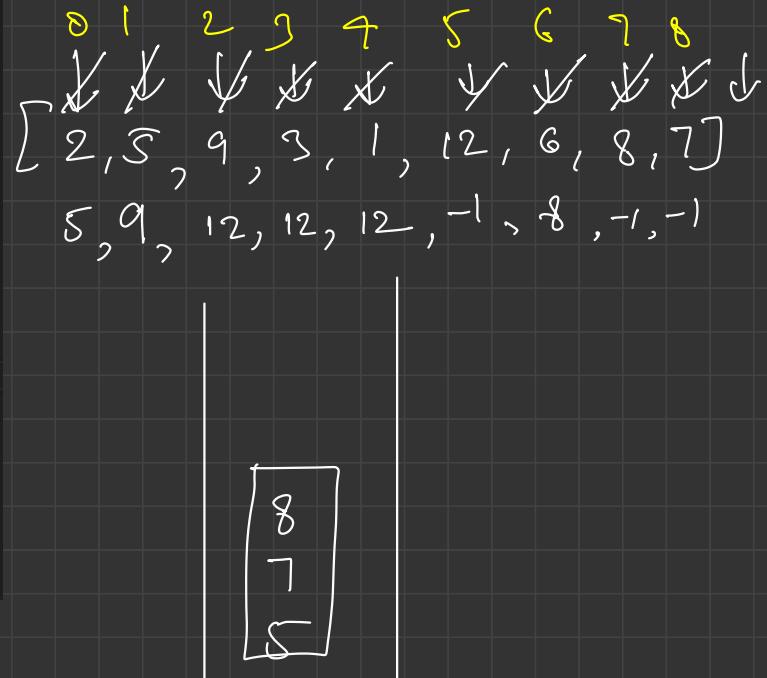
    for (int i = 0; i < n; i++) {
        while (st.size() > 0 && arr[st.peek()] < arr[i]) {
            int idx = st.pop();
            nger[idx] = arr[i];
        }
        st.push(i);
    }

    while (st.size() > 0) {
        int idx = st.pop();
        nger[idx] = -1;
    }

    return nger;
}

```

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

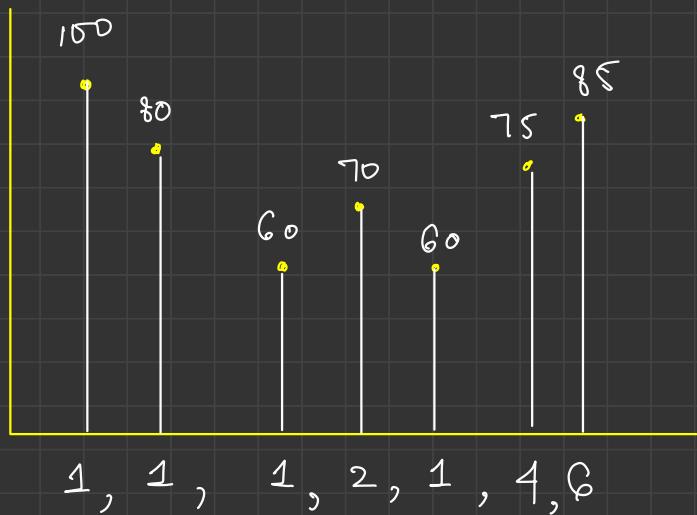


Stock Span

Stock[] = [100, 80, 60, 70, 60, 75, 85]
0th 1st 2nd 3rd 4th 5th 6th

2 1 4

= 1 - sum

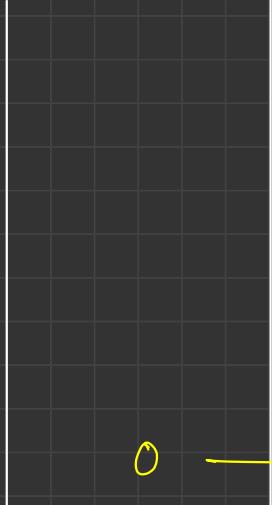


\downarrow [100, 80, 60, 70, 60, 75, 85]

0th 1st 2nd 3rd 4th 5th 6th
0 0 1 2 3 1 0

ngeli [-1, 0, 1, 1, 3, 1, 0]

span[2] [1, 1, 1, 2, 1, 4, 6]



0 → 100

SC: $O(N)$

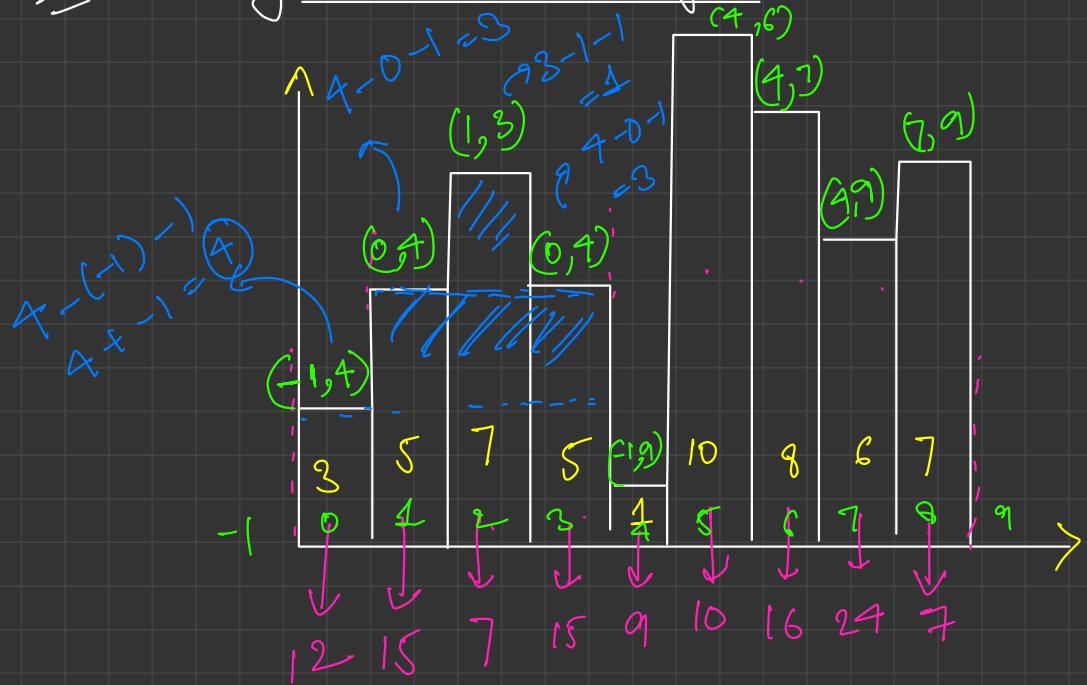
```
class Solution {  
    static int[] nextGreaterEleOnLeftIdx(int[] a, int n) {  
        int[] ngeli = new int[n];  $O(N)$   
        Stack<Integer> st = new Stack<>();  $O(N)$   
        for (int i = n - 1; i >= 0; i--) {  
            if (st.size() == 0) {  
                st.push(i);  
            } else {  
                while (st.size() > 0 && a[st.peek()] < a[i]) {  
                    int idx = st.pop();  
                    ngeli[idx] = i;  
                }  
                st.push(i);  
            }  
        }  
  
        while (st.size() > 0) {  
            int idx = st.pop();  
            ngeli[idx] = -1;  
        }  
  
        return ngeli;  
    }  
  
    static int[] stockSpan(int[] a) {  
        int[] ngeli = nextGreaterEleOnLeftIdx(a, a.length);  
        int[] span = new int[a.length];  
        for (int i = 0; i < a.length; i++) {  
            span[i] = i - ngeli[i];  
        }  
        return span;  
    }  
}
```

TC: $O(N)$

SC: $O(N)$

Q

largest Area Histogram



$$\text{width} = r - l - 1$$

Step 5
① get next smaller element by ${}^{\circ}\text{idx}$ on
left

② get next smaller ele by idx on
Right

③ width = $\gamma - l - 1$ area = height $\times \underline{\text{width}}$



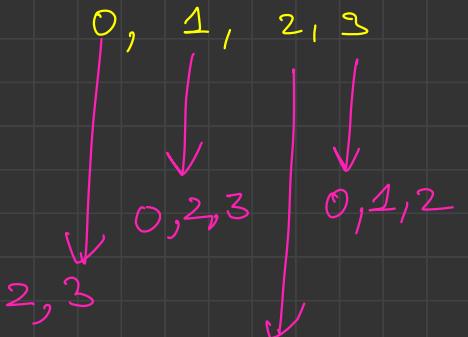
Celebrity Problem

	0	1	2	3
0	0 (0,0)	0 (0,1)	1 (0,2)	1 (0,3)
1	1 (1,0)	0 (1,1)	1 (1,2)	1 (1,3)
2	0 (2,0)	0 (2,1)	0 (2,2)	0 (2,3)
3	1 (3,0)	1 (3,1)	1 (3,2)	0 (3,3)

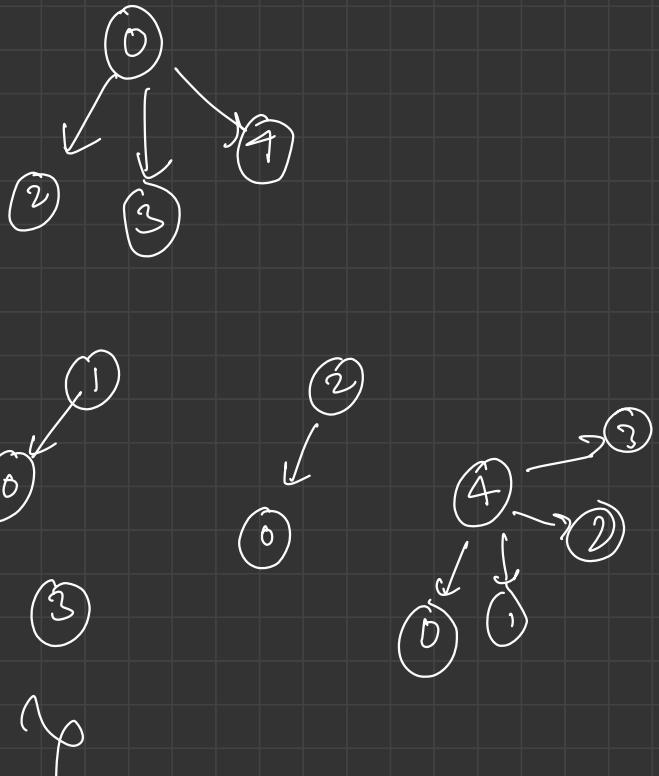
$\left\{ \begin{matrix} n \times n \\ n \text{ different persons} \end{matrix} \right.$

If i knows j $\text{arr}[i][j] = 1$

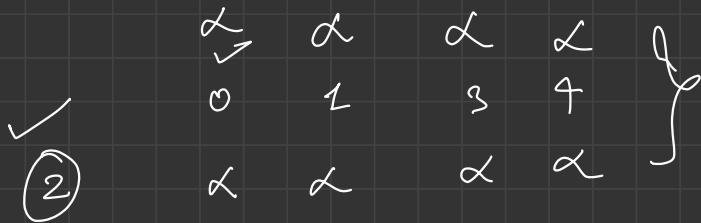
If i doesn't know j $\text{arr}[i][j] = 0$



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



- ① He should not know anyone
- ② He should be known by everyone



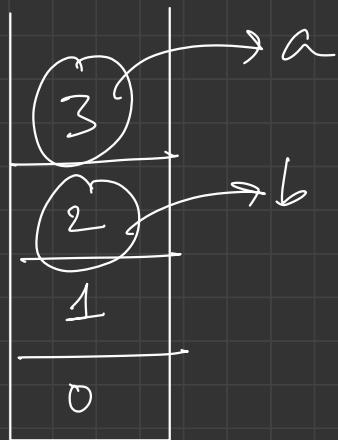
Only One cell at Max possible!

Brute force $\underline{\underline{TC: O(N^2)}}$

elimination technique

~~if~~ $\text{arr}[a][b] = 1$ {a knows b}
b can be my
 cells

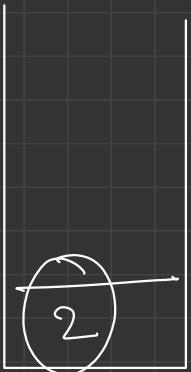
	0	1	2	3
0	0	0	1	1
1	1	0	1	1
2	0	0	0	0
3	1	1	1	0



Stack

if $\text{arr}[a][b] = 0$
 ↗ a doesn't
 know b ↘
 a can be my
 cells

	0	1	2	3
0	0 (0,0)	0 (0,1)	1 (0,2)	1 (0,3)
1	1 (1,0)	0 (1,1)	1 (1,2)	1 (1,3)
2	0 (2,0)	0 (2,1)	0 (2,2)	0 (2,3)
3	1 (3,0)	1 (3,1)	1 (3,2)	0 (3,3)



celeb = 2

```

class Solution {
    int findCelebrity(int M[][], int n) {
        // step 1: add everyone to the stack
        Stack<Integer> possibleCelebs = new Stack<Integer>();
        for (int i = 0; i < n; i++) {
            possibleCelebs.push(i);
        }

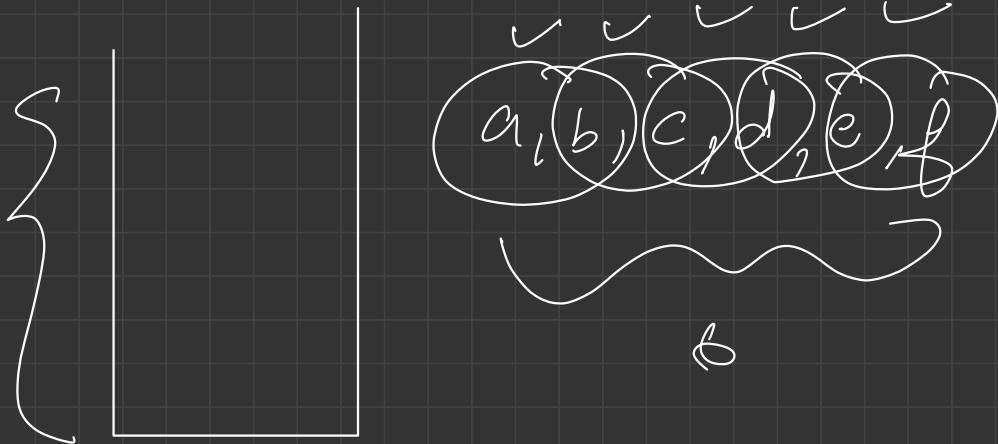
        while(possibleCelebs.size() >= 2) {
            int p1 = possibleCelebs.pop();
            int p2 = possibleCelebs.pop();

            if (M[p1][p2] == 1) {
                // p1 knows p2, hence p2 can be celeb but p1 cann't be
                possibleCelebs.push(p2);
            } else {
                // p1 doesn't know p2, hence p2 can't be a celeb, but p1 can be one
                possibleCelebs.push(p1);
            }
        }

        int celeb = possibleCelebs.pop();
        // check row
        for (int c = 0; c < n; c++) {
            if(M[celeb][c] == 1) {
                return -1;
            }
            if (c != celeb && M[c][celeb] == 0) {
                return -1;
            }
        }
    }
}

```

return celeb;



$$\begin{array}{c}
 \left. \begin{array}{l} TC: O(N) + O(N) + O(N) = O(3N) = O(N) \\ SC: O(N) \end{array} \right\} \\
 \hline
 \end{array}$$

Evaluate Infix Expressions.

Rule BODMAS

#

$$2 * 4 / (\underline{4 / 2}) + 6$$

① first Brackets
are evaluated

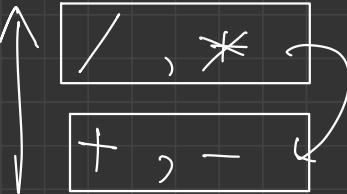
①

$$2 * 4 / 2 + 6$$

② precedence:

②

$$8 / 2 + 6$$



③

$$4 + 6$$

④

10

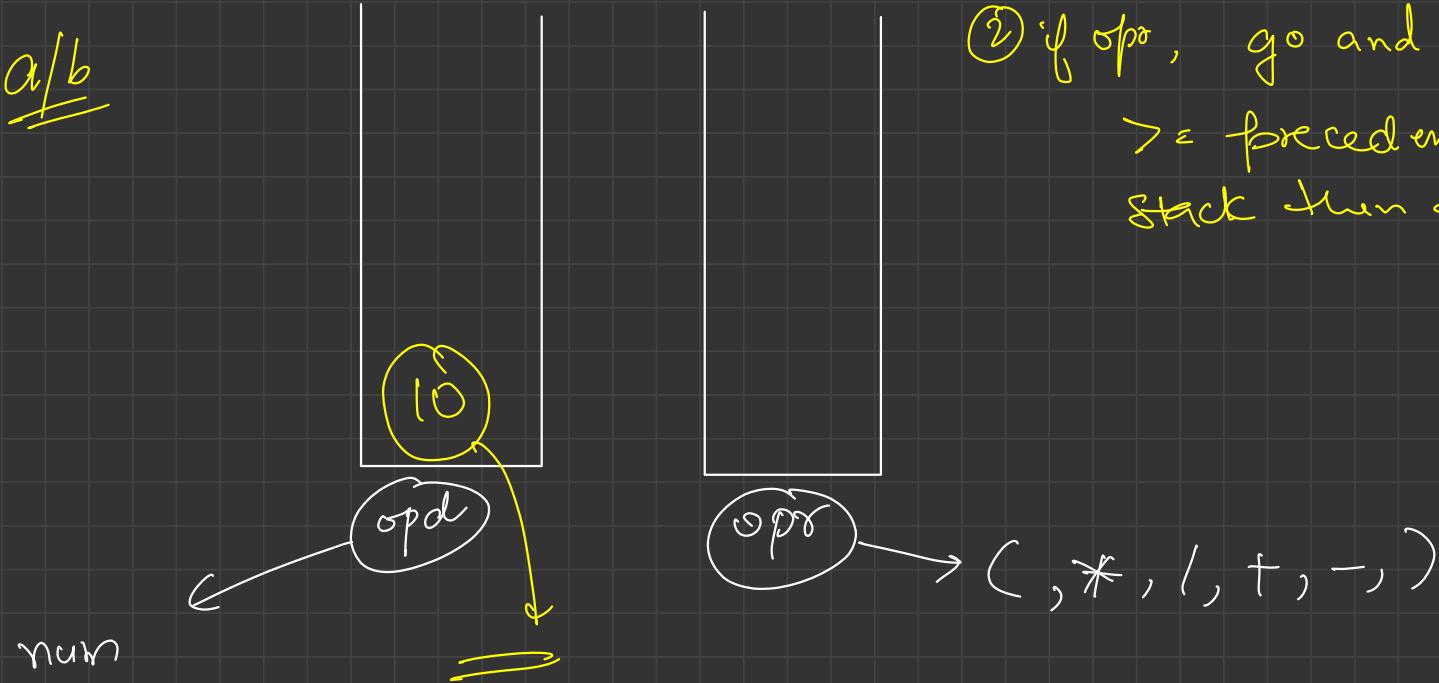
$$\begin{array}{ll} / - 2 & + \rightarrow 1 \\ * - 2 & - \rightarrow 1 \end{array}$$

$$2 * 4 / (4 / 2) + 6$$


① iq opd, opd. push (ch)

② if so, go and solve

$> \in$ precede from
stack then add me.




```

public int precedence(char o) {
    if (o == '+') {
        return 1;
    } else if (o == '-') {
        return 1;
    } else if (o == '/') {
        return 2;
    } else {
        // o -> *
        return 2;
    }
}

```

```

public int eval(int v1, int v2, char o) {
    if (o == '+') {
        return v1 + v2;
    } else if (o == '-') {
        return v1 - v2;
    } else if (o == '/') {
        return v1 / v2;
    } else {
        // o -> *
        return v1 * v2;
    }
}

```

$2 \times 4 / (7 / 2) + 6$

```

public int solve(String exp) {
    // consist of operands
    Stack<Integer> opd = new Stack<>();
    // consist of operators
    Stack<Character> opr = new Stack<>();

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

        if (Character.isDigit(ch) == true) {
            opd.push(ch - '0');
        } else if (ch == '(') {
            opr.push(ch);
        } else if (ch == ')') {
            while (opr.peek() != '(') {
                int v2 = opd.pop();
                int v1 = opd.pop();

                char o = opr.pop();

                int ans = eval(v1, v2, o);

                opd.push(ans);
            }
            opr.pop();
        } else {
            // ch -> *, /, +, -
            while (opr.size() > 0 && opr.peek() != '(' && precedence(ch) <= precedence(opr.peek())) {
                int v2 = opd.pop();
                int v1 = opd.pop();

                char o = opr.pop();

                int ans = eval(v1, v2, o);

                opd.push(ans);
            }
            opr.push(ch);
        }
    }
}

```

```

while (opr.size() > 0) {
    int v2 = opd.pop();
    int v1 = opd.pop();

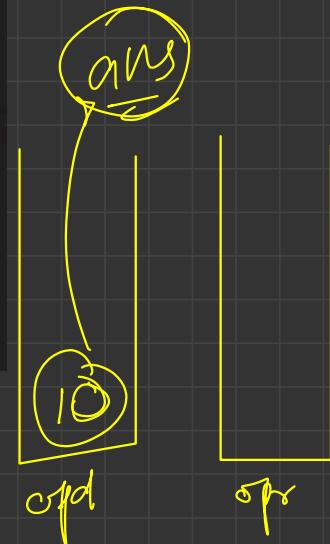
    char o = opr.pop();

    int ans = eval(v1, v2, o);

    opd.push(ans);
}

return opd.peek();
}

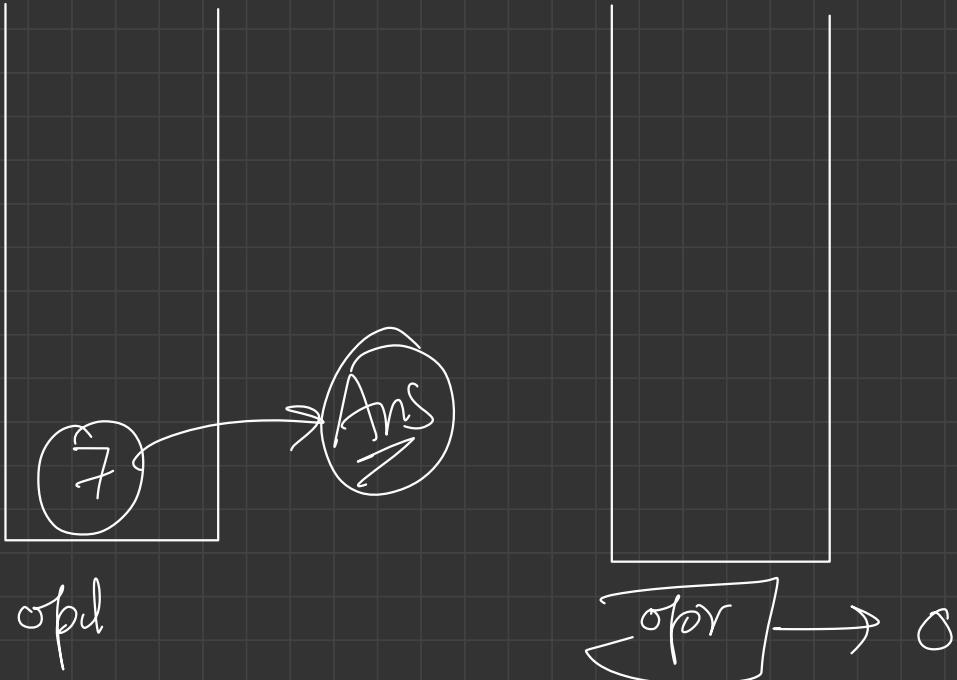
```



$$2 * 4 / ((4 / 2) + (2 * 3)) + 6$$

Diagram illustrating the expression evaluation:

- The expression is $2 * 4 / ((4 / 2) + (2 * 3)) + 6$.
- Arrows point from each character to its corresponding memory location:
 - Two locations for 2
 - Four locations for $*$
 - One location for 4
 - One location for $/$
 - Two locations for $($
 - One location for 4 (inside the first parenthesis)
 - One location for $/$ (inside the first parenthesis)
 - Two locations for $)$ (inside the first parenthesis)
 - Two locations for $+$
 - Two locations for 2
 - One location for $*$
 - Three locations for 3
 - One location for $)$ (inside the second parenthesis)
 - One location for $+$
 - One location for 6
- A yellow arrow points to the 2 in $2 * 3$, indicating it is being multiplied by the current value of 2 .
- The result of the multiplication is stored at $V1 * V2 = 8$.



Convert Infix to postfix and prefix.

$$2 * 4 / (4 / 2) + 6 \rightarrow \underline{\text{Infix}}$$

postfix

prefix

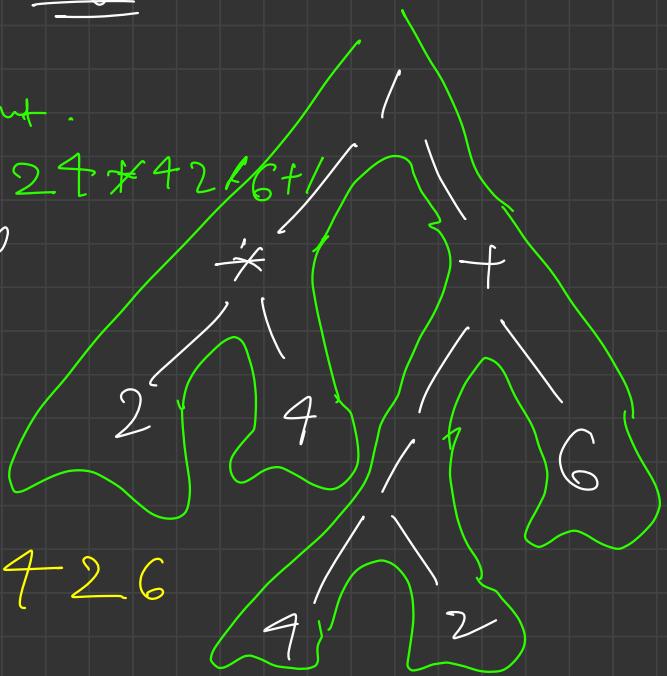
left → right & point.

opd + opr

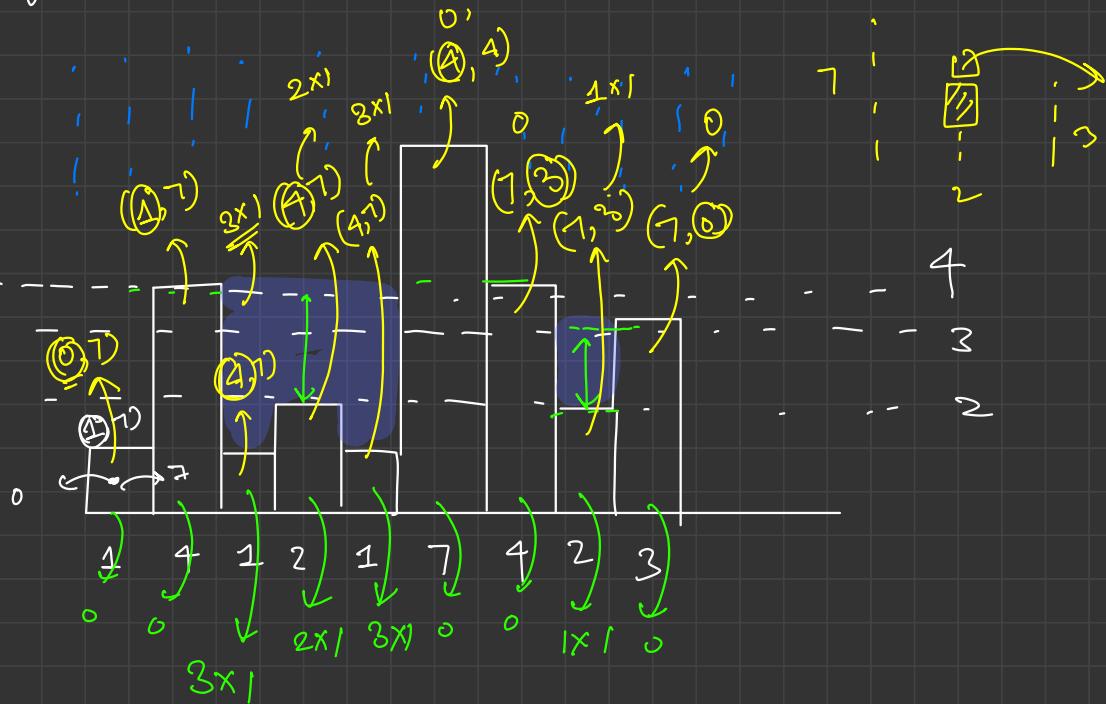
opr + opd

point → left → right

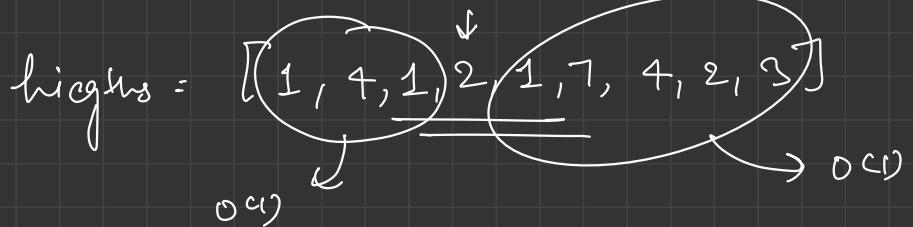
/ * 2 4 + / 4 2 6



Trapping Rain Water.



$$0+0+\cancel{3}+\cancel{2}+\cancel{3}+0+0+\cancel{1}+0 = \textcircled{9} \text{ units of water}$$



maxWater
 $t = (\min(lm, rm) - h) \times$

$O(n)$ → work for Each Element

Brun Force
 Method

TC: $\underline{\underline{O(N \times N)}} \approx \underline{\underline{O(N^2)}}$

SC: $\underline{\underline{O(1)}}$

0 0 3

heights = [1, 4, 1, 2, 1, 7, 4, 2, 3]

lmax[] = [1, 4, 4, 4, 7, 7, 7, 7]

rmax[] = [7, 7, 7, 7, 7, 4, 3, 3]

1 4 4-1

currWater = $\left[\frac{\min(lmax[i], rmax[i]) - height[i]}{1} \right] \times 1$

for (int i=1; i<n; i++)
{
 lmax[i] = Math.max(lmax[i-1], heights[i]);
}

for (int i=n-2; i>=0; i--)

{
 rmax = Math.max(rmax[i+1], height[i]);

0

heights = [1, 4, 1, 2, 1, 7, 4, 2, 3] } TC: O(3N) = O(N)

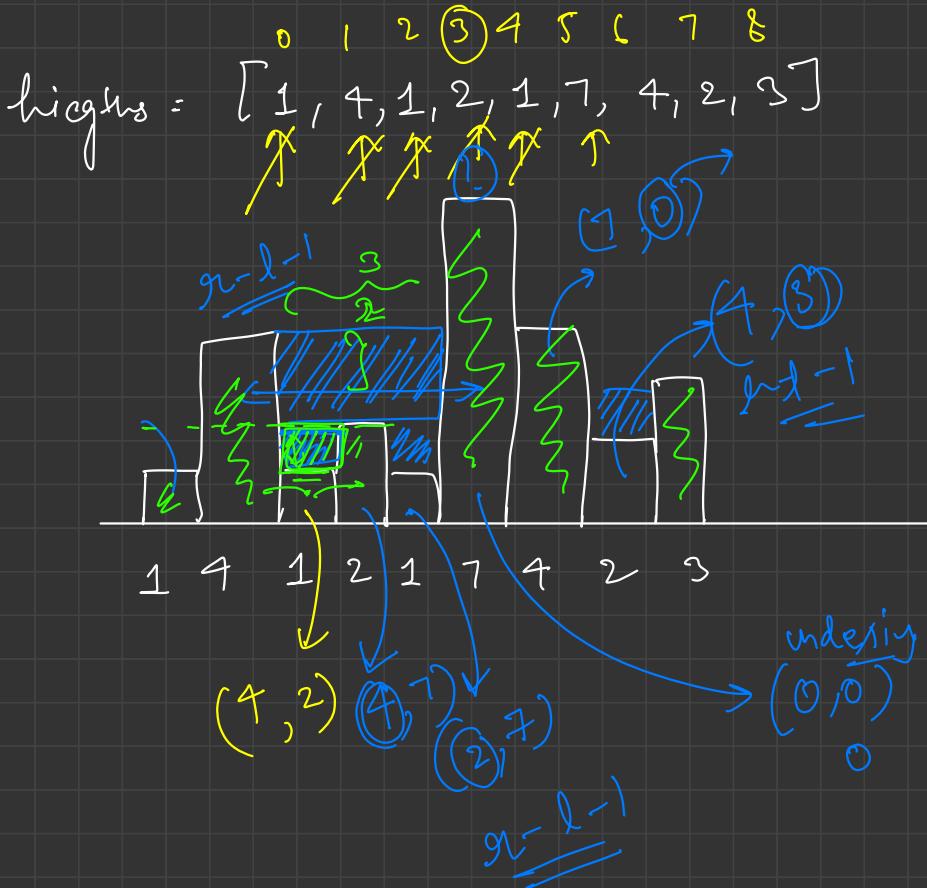
✓
lmax[] = [1, 4, 4, 4, 4, 7, 7, 7, 7]

✓
rmax[] = [7, 7, 7, 7, 7, 4, 3, 3]

maxheight 1

water height: 1 - 1

Water height
 \approx_1 area = height × 1



ngc
 ① till you can't be ngc for
 peek
 ② curr > arr[peek()]

```

// Approach 2
public void TappingWater(int[] arr, int n) {
    // Write code here and print output
    Stack<Integer> st = new Stack<>();
    int totalWater = 0;

    for (int i = 0; i < n; i++) {
        while (st.size() > 0 && arr[i] > arr[st.peek()]) {
            int r = i;
            int cVal = arr[st.pop()];

            if (st.size() == 0) {
                // don't have next greater element on left
                break;
            }

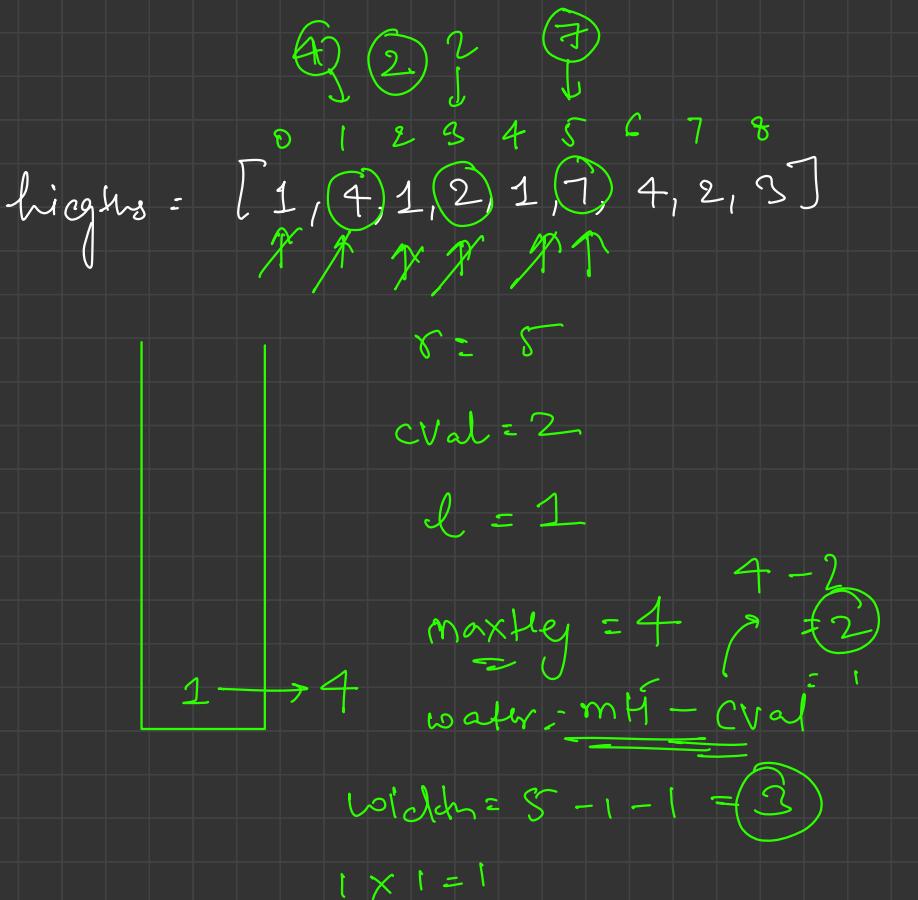
            int l = st.peek();
            int width = r - l - 1;
            int totalHeight = Math.min(arr[l], arr[r]);
            int waterHeight = totalHeight - cVal;

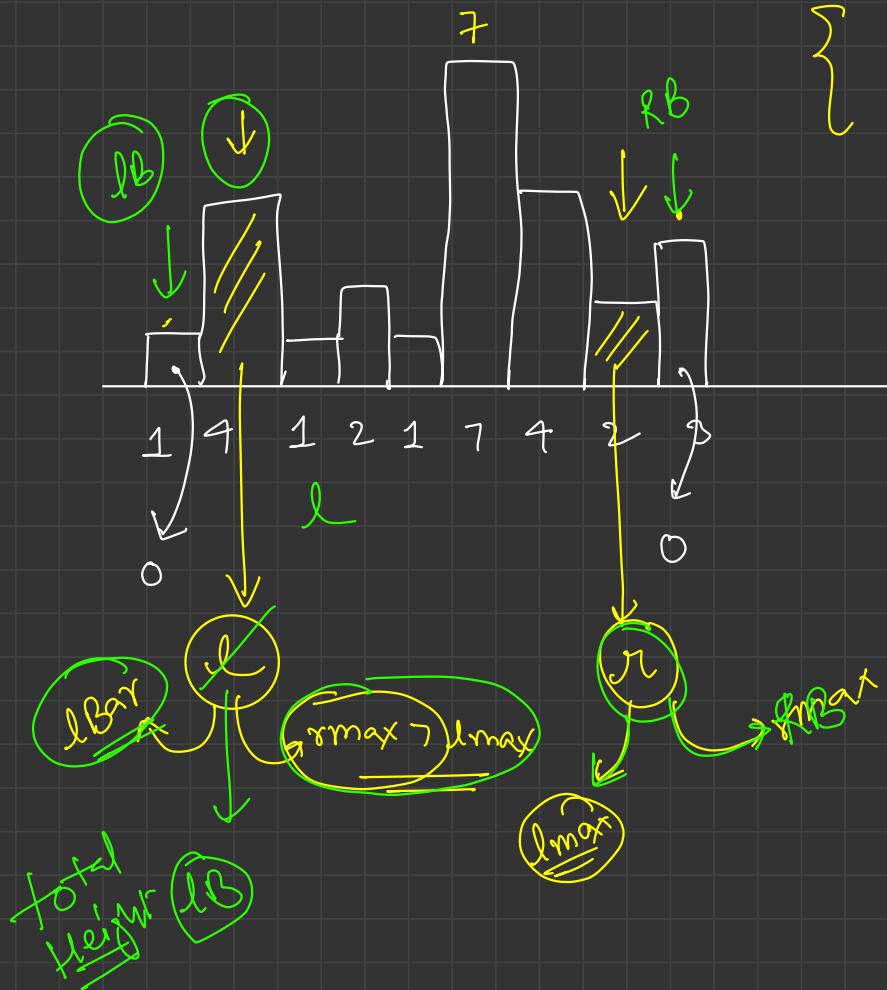
            totalWater += (width * waterHeight);
        }
        st.push(i);
    }

    System.out.println(totalWater);
}

```

STC : O(N)
SC : O(N)



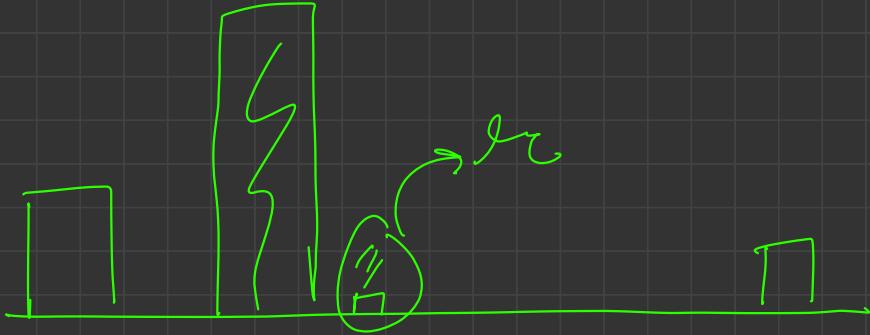


```

    } int leftBar = arr[0]; 1
    } int rightBar = arr[n-1]; 3
    int l = 1; 
    int rc = n-2; 

    if (leftBar < Right Bar)
    {
        if (lB < arr[l])
            lB = arr[l]
        else water f = (lB - arr[l])
    }
    else
    {
        if (RB < arr[r])
            RB = arr[r];
        else water f = (RB - arr[r])
    }
}

```

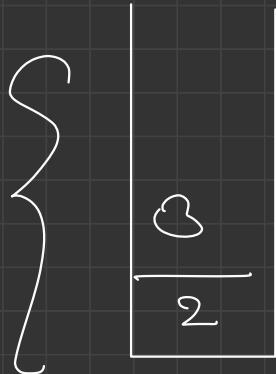
$$RB < LB$$

Height : RB

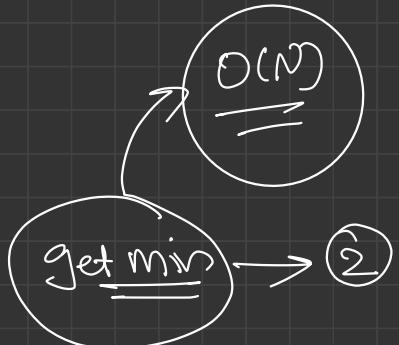
Minimum Stack

~~X~~ ↓

2, 3, 4, 1, 10, 5



Actual
St



$O(N)$

get min

②

minVal = 2
2

temptst

```

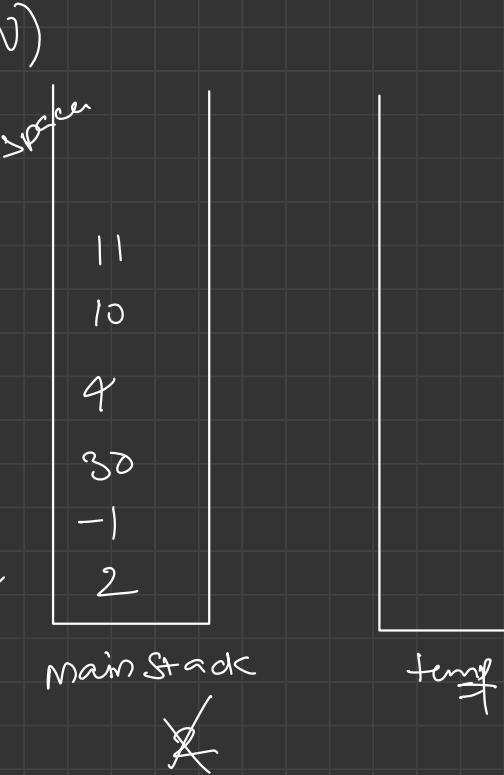
    // TC: O(N), SC: O(N)
    int getMin() {
        // Write Code here
        if (mainStack.size() == 0) {
            return -1;
        }

        int minValue = Integer.MAX_VALUE;
        Stack<Integer> tempStack = new Stack<>();
        while (mainStack.size() != 0) {
            int val = mainStack.pop();
            minValue = Math.min(minValue, val);
            tempStack.push(val);
        }

        while (tempStack.size() != 0) {
            int val = tempStack.pop();
            mainStack.push(val);
        }

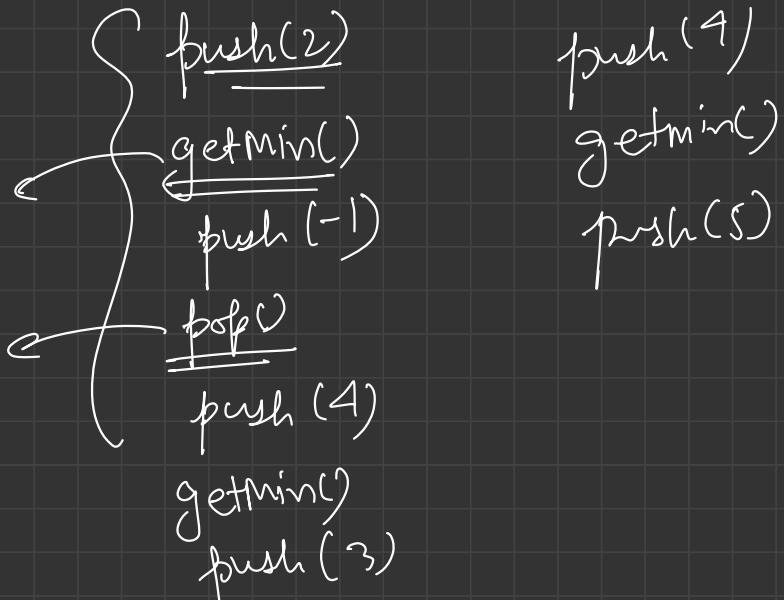
        return minValue;
    }
}

```

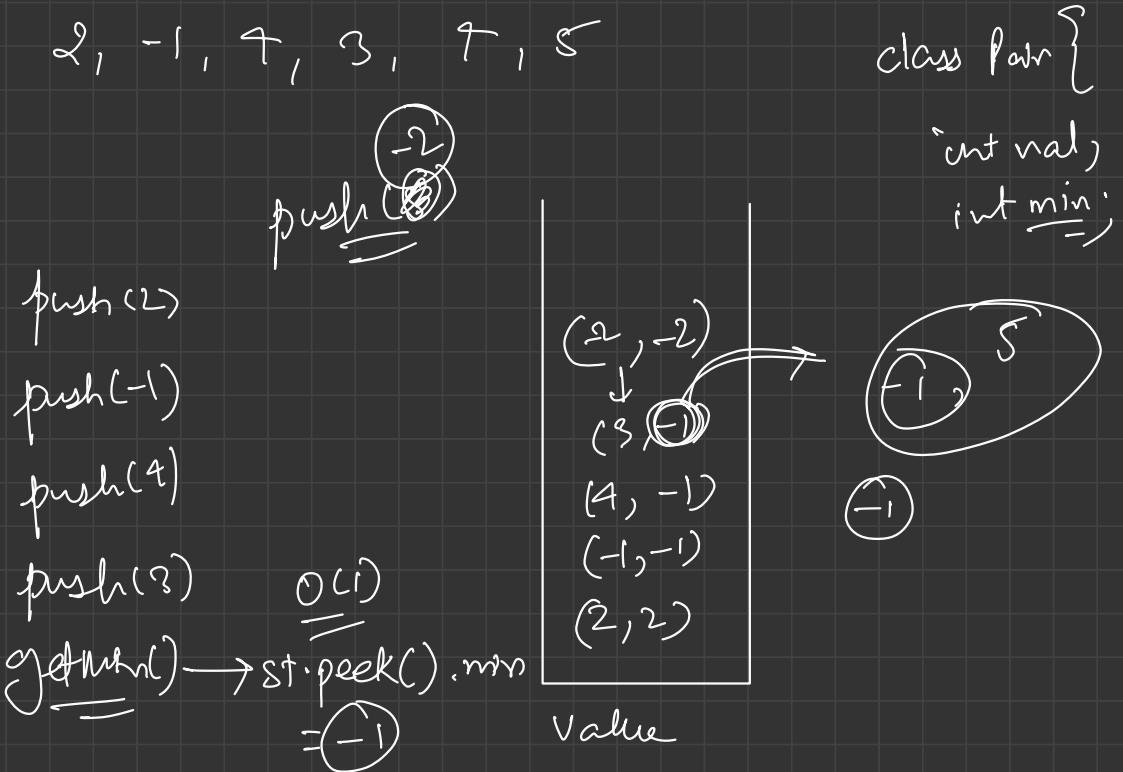


$$\minVal = \cancel{\text{MaxInt}} \neq 10 \neq \underline{-1}$$

(2) -1, 7, 3, 7, 5

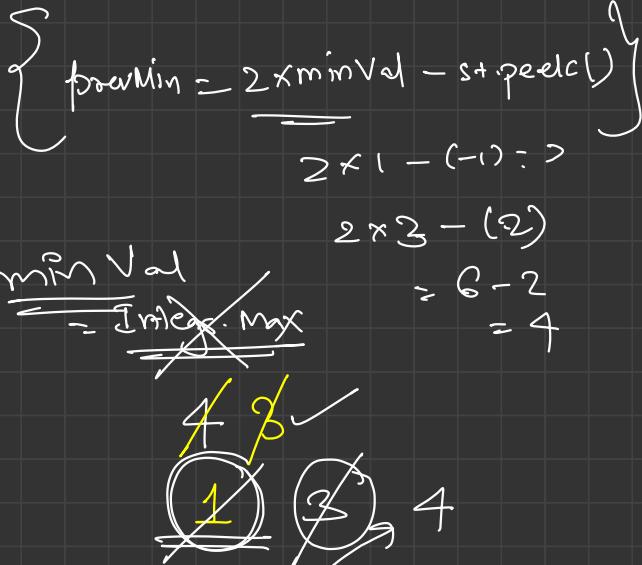


getMin() $\rightarrow O(1)$ time Complexity ?



- $\rightarrow \text{push}(4)$, $\text{getMin}() \rightarrow 4$
- $\rightarrow \underline{\text{push}(3)}$, $\underline{\text{getMin}()}$ $\rightarrow 3$
- $\rightarrow \underline{\text{push}(7)}$ $\rightarrow \underline{\text{getMin}()} \rightarrow 3$
- $\rightarrow \underline{\text{push}(5)}$, $\underline{\text{getMin}()} \rightarrow 3$
- $\rightarrow \underline{\text{push}(4)}$, $\underline{\text{getMin}()} \rightarrow 3$
- $\rightarrow \underline{\text{push}(1)}$, $\underline{\text{getMin}()} \rightarrow 1$
- $\rightarrow \text{pop}()$

$$2 \times 3 - 4 \\ = 2$$



* Whenever you are updating \minVal ,

then $\text{push}(\underline{2 \times x} - \underline{\minVal}) \rightarrow \underline{\text{val}}$

$\underline{\text{new min}} = \underline{x}$

$\minVal - \underline{x}$
 $\underline{=} \rightarrow \underline{\text{—}}$

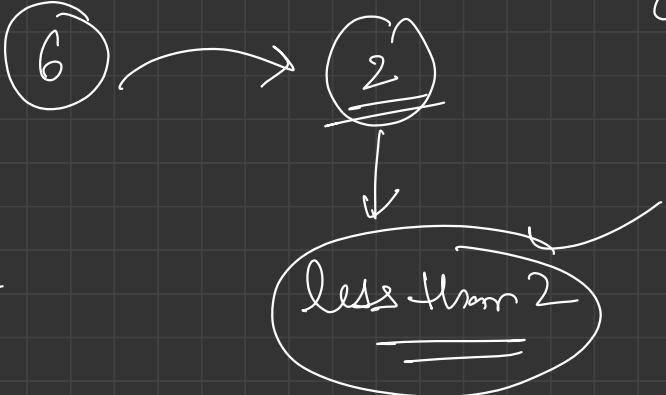
~~st.peek()~~ ← ~~MinValue~~
 int val = 2 * ~~x~~ - ~~minValue~~; ← forev
 st.push(val);
 minValue = x;

$$\begin{array}{l}
 \textcircled{A} = 2 \times \textcircled{B} \leftarrow C \\
 \boxed{\text{bovenMinValue} = 2 \times \cancel{\text{minValue}} - \cancel{\text{st.peek()}}}
 \end{array}$$

$$= 2 \times 1 - (-1)$$

$$\Rightarrow 2 + 1 = \textcircled{3}$$

1 -



$$2 - 6 = \underline{\underline{-4}}$$

curr Min $\underline{\underline{2}}$

$$\underline{\underline{2}} \times \underline{\underline{2}} - \underline{\underline{C}} = \underline{\underline{-2}}$$

$$2 \times 2 = 4 =$$

Min

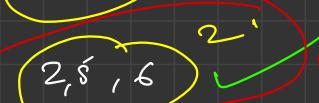
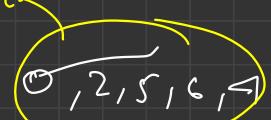
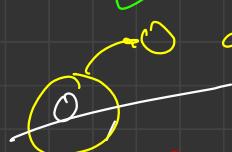
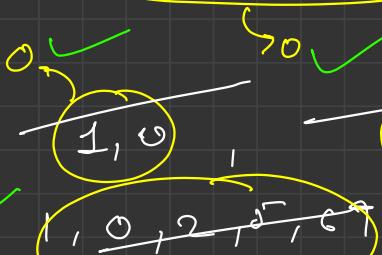
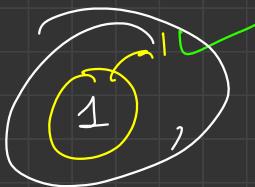
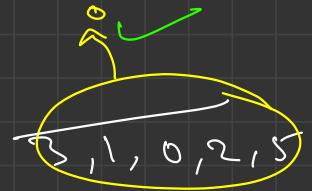
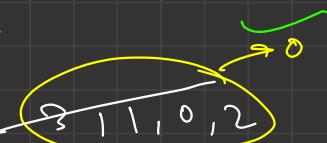
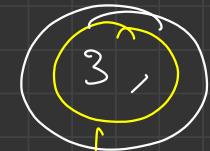
$$a + b - c = d =$$

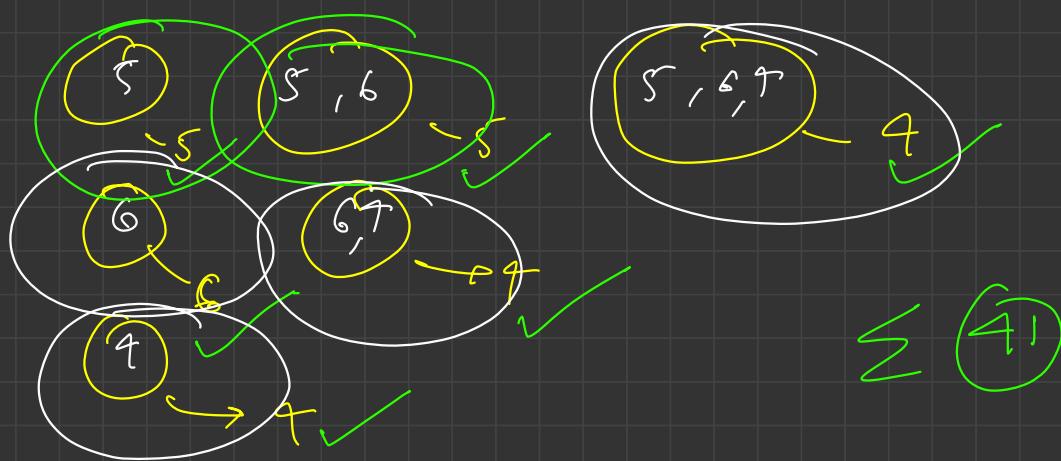
$$\underline{\underline{b < c}}$$

$$\underline{\underline{d < a}}$$

Sum of Subarrays Minimum

arr[] = [3, 1, 0, 2, 5, 6, 4]





$\leq \hat{O}(1)$

Brute force :

$$TC: N \times \underline{\underline{N^2}} = \underline{\underline{O(N^3)}} \quad \left. \right\}$$

Smarter Brute-force

$$\underline{\underline{TC: O(N^4)}}$$

$$\left\{ \text{TC}: O(N) \quad \text{SC}: \underline{\underline{O(N)}} \right\}$$

$$\text{arr}[] = [3, 1, 0, 2, 5, 6, 4]$$

$$\left\{ 2 \quad 2, 5 \quad 2, 5, 6 \quad 2, 5, 6, 4 \right\} \rightarrow 2$$

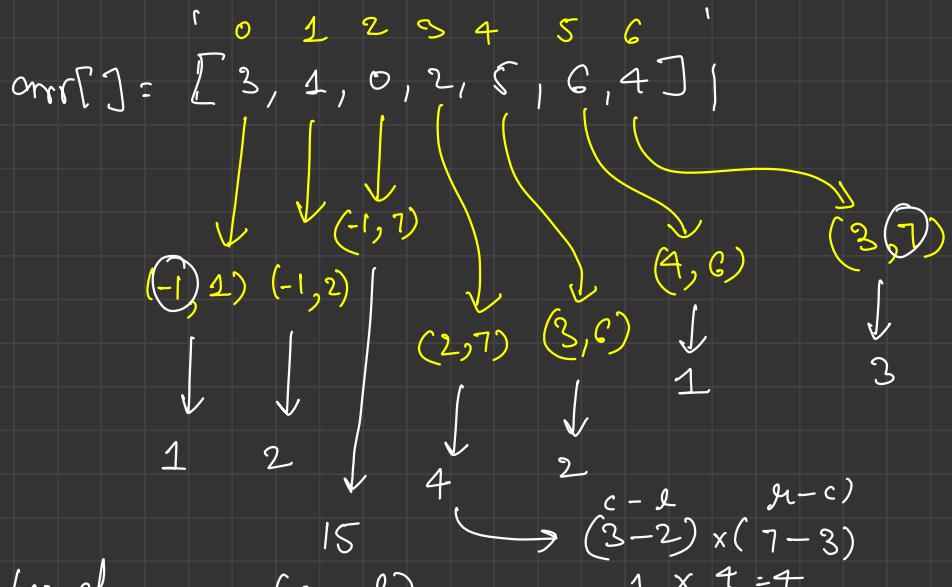
$$\left\{ 5 \quad 5, 6 \right\} \rightarrow 5$$

$$\left\{ 7 \quad 6, 4 \quad 5, 6, 4 \right\} \rightarrow 4$$

$$\left\{ 6 \right\} \rightarrow 6$$

$$\left\{ 3 \right\} \rightarrow 3$$

$$\left\{ 1, 3, 1 \right\} \rightarrow 1$$



total Number of
 Subarrays with
curr as min

$$\rightarrow \boxed{(r - cidx)(cidx - l)}$$

$\sum \min \text{ of } \sum = \underline{x} \times \underline{\text{currVal}}$

