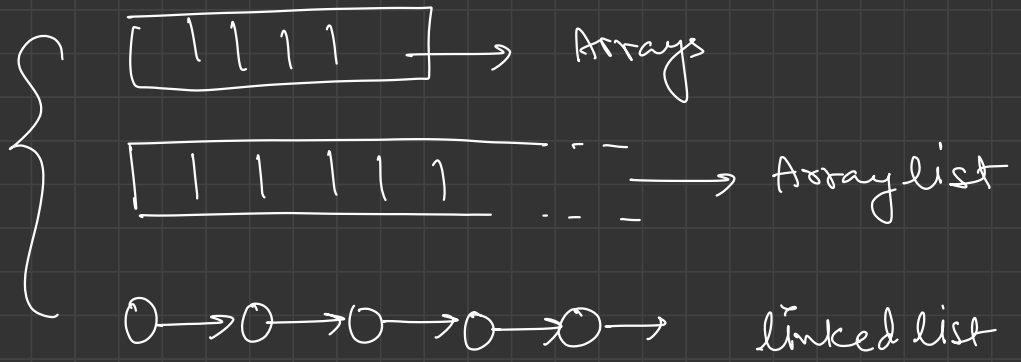
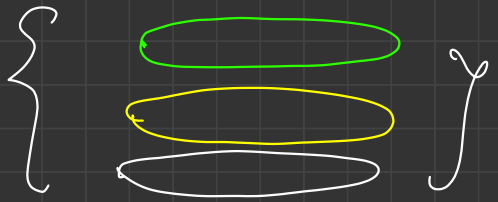
Stacks → linear data structure



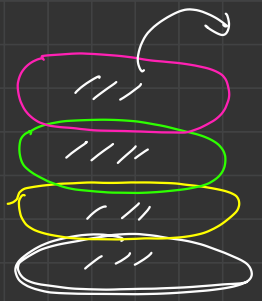
(1) Stack is a linear DS



dishes are stacked together!

Stack of plates

Stack Operations

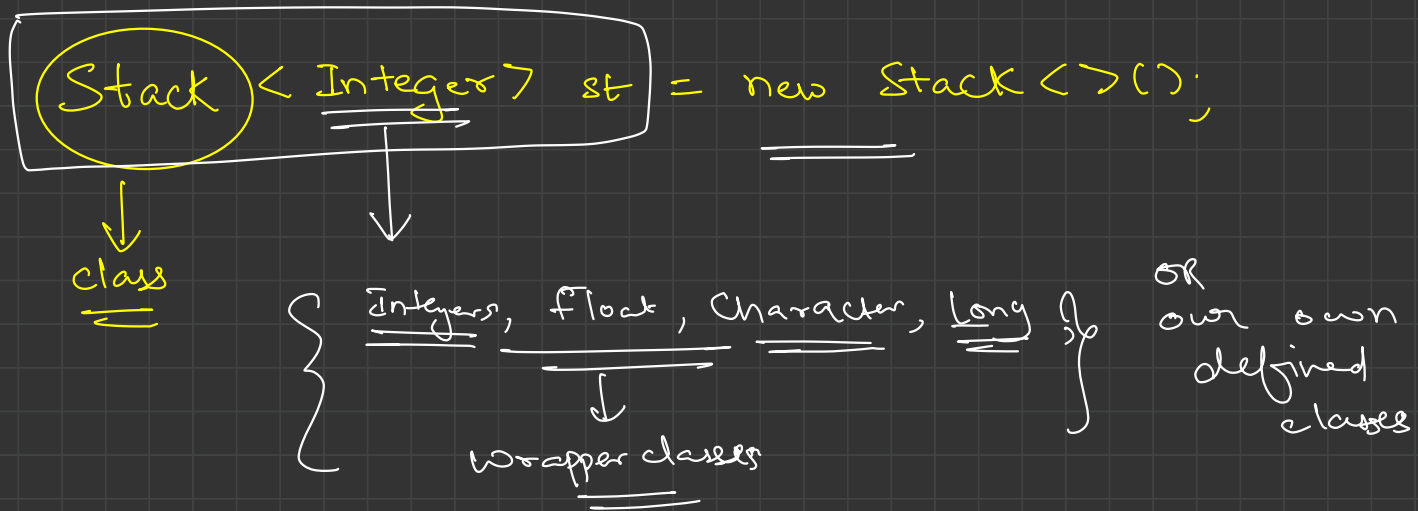


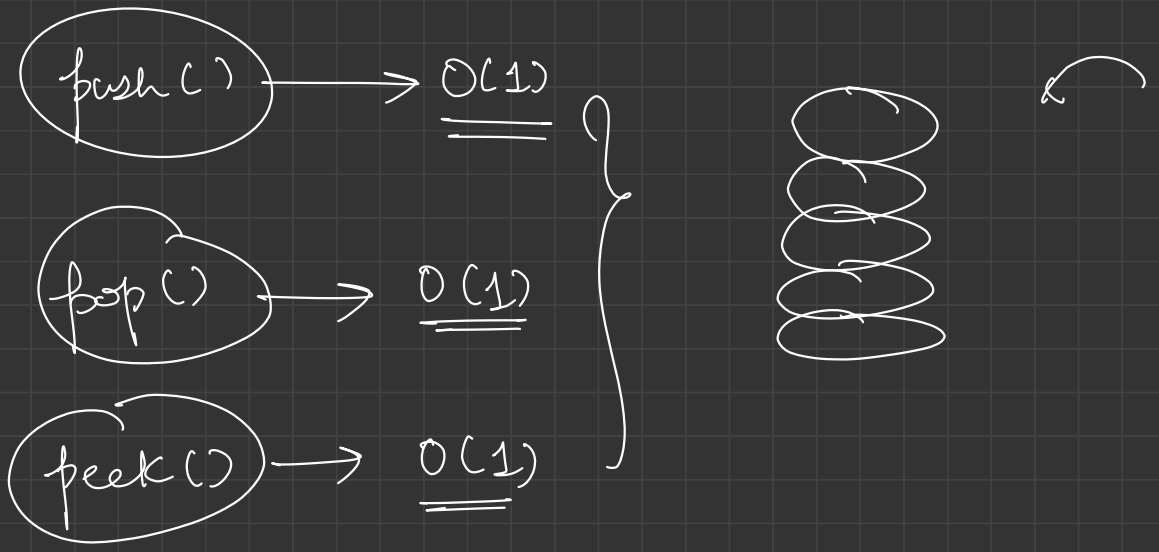
1. addition of new data in a stack is done on top
2. deletion of data from a stack is done from top
3. data on top is only readable

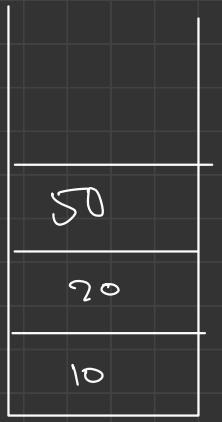
① push() : add data in the stack (top of the stack)

② pop() : removes topmost element from the stack.

③ peek() : see data present on the top of the stack


Stack < Integer > st = new Stack <>();
class
Integers, float, Character, Long
Wrapper classes
OR
our own defined classes


push()
O(1)
pop()
O(1)
peek()
O(1)



St.peek() → 30

St.pop()

St.peek() → 20

St.push(50)

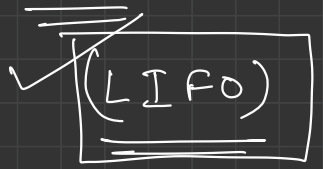
St.peek() → 50

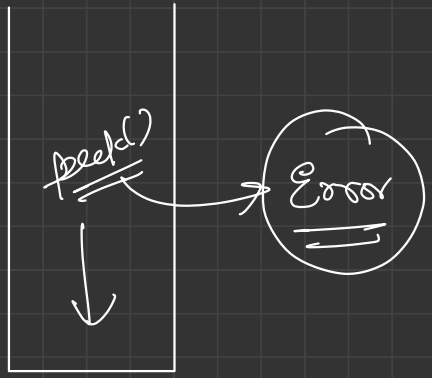
St.size() → 3

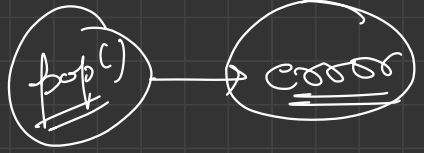
① size()

→ Size of the Stack!

Stack follows, last In, But first out.







Stack underflow

St

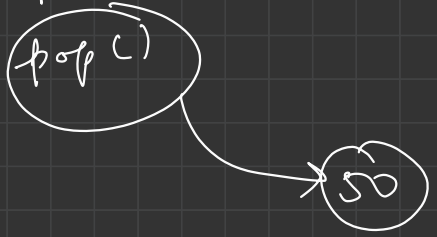
push (20)

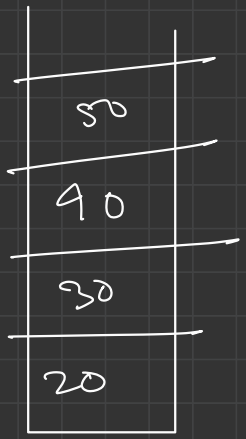
push (30)

push (40)

push (50)

pop ()





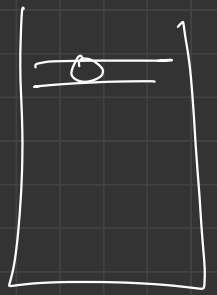
(last person In, first out)

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()

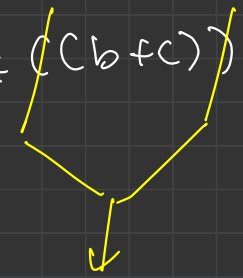
remove last()

remove first()

Size of linked list()

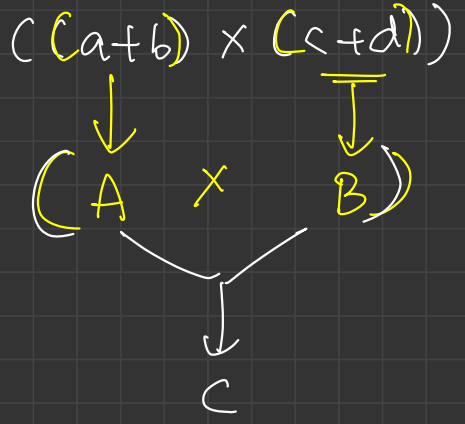
Q $(a+(b\times c)-(d\times f))$ ⟶ false (No extra brackets)

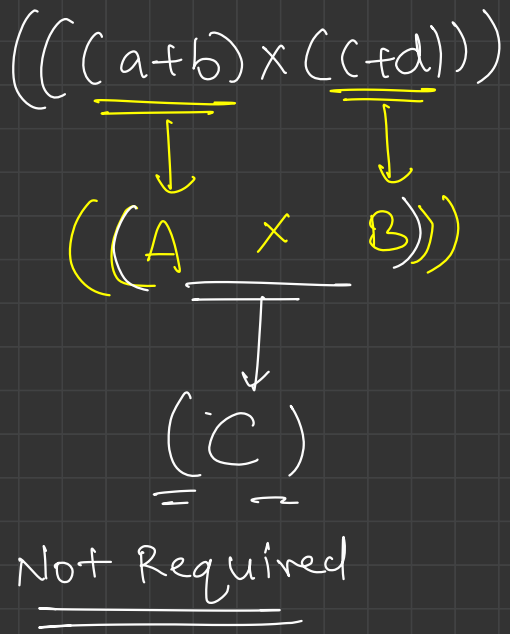
$(a+((b+c))-(d\times f))$ ⟶ true



extra bracket

✱ Every Bracket is balanced and we don't have any extra bracket.


((a+b) x (c+d))
(A x B)
C

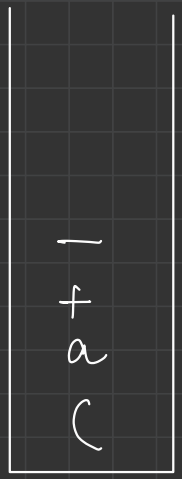

(((a+b) x (c+d)))
((A x B))
(C)
Not Required

(a + (b x c) − (d x f))

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

(+) → '(' or operator char.

(−) → ')'



Last person opened has to be closed first.

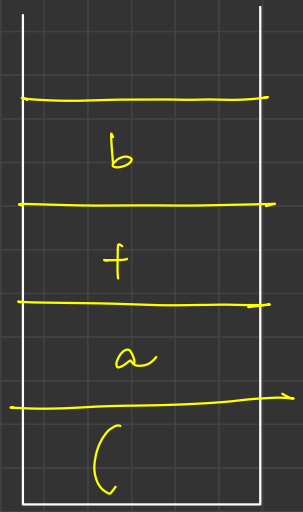
(LIFO)

search for corresponding '(' bracket in stack

if it had some other '(' bracket in stack.

means, we have an exp.

(a+b) + c +((d+e))



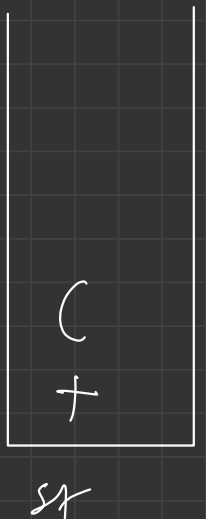
St

```
if (ch != ')')
{
    st.push(ch);
}
else
{
    if(st.peek() == '(')
        return true;
    else
    {
        while(st.peek() != '(')
            st.pop();
        st.pop(); ⟶ corresponding opening
```

broadcast:

```java
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 Char 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;
}
```

ex: (a+b)+((c+d))



a+b+(d)

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

(5)

```
for (i=0 → n)
{
    whileC()
}
```

whileC() → N times in its life span

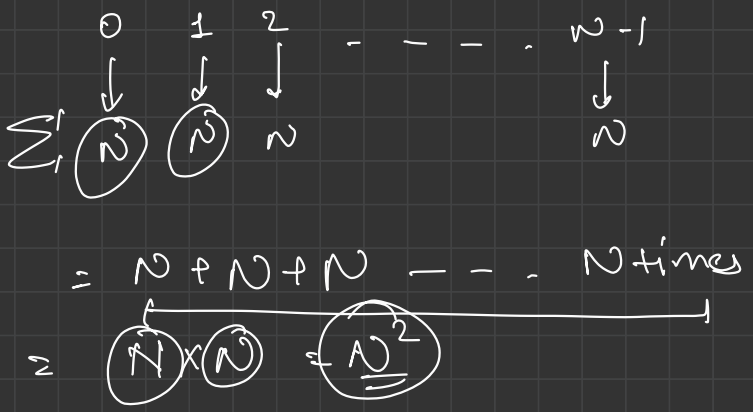
O(2N)

i = 0, 1, 2, 3, 4

N times

O(N²)

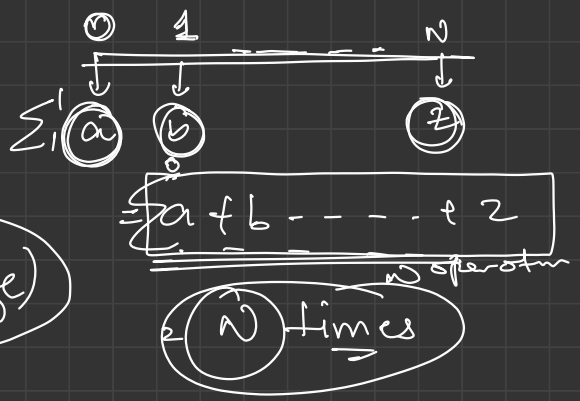
```
for (i -> 0 -> n)
{
    while (n)
}
```

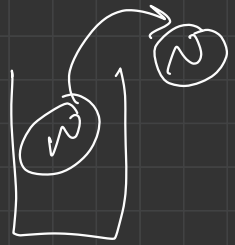


O(N)

```
for (i = 0 -> n)
{
    while ()
}
```

(N time in life)





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 ; n)
{
   for(int j= i+1 ; n)
      if(arr[j] > arr[i])
      {
         nger[i] = arr[j]
         break;
```

}

} TC: $O(N^2)$

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

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

SC: $O(1)$
TC: $O(N^2)$

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

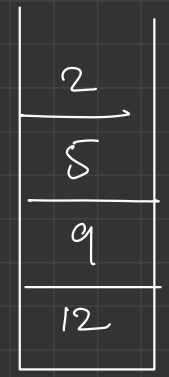
nger[] : [5, 9, 12, 12, 12, -1, 8, -1, -1]

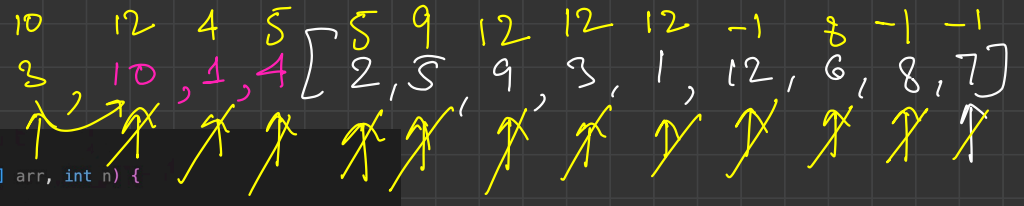
Approach 1

(1) traverse array from last to start.

```
[2, 5, 9, 3, 1, 12, 6, 8, 7]
→ [5, 9, 12, 12, 12, -1, 8, -1, -1]
```

```
if (st.size() == 0)
    nger[i] = -1;
else
    remove element from the stack till anything bigger present.
    if (st.size() == 0) nger[i] = -1
    else nger[i] = st.peek();
st.push(arr[i]);
```

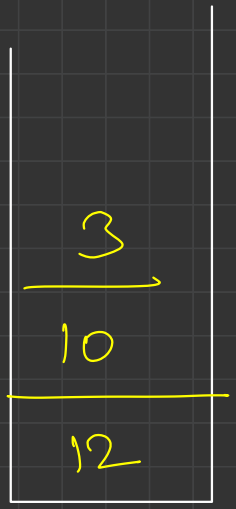





```java
class Solution {
    public static long[] nextLargerElement(long[] arr, int n) {
        // Write code here and print output
        long[] nger = new long[n];
        Stack<Long> st = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {
            if (st.size() == 0) {
                nger[i] = -1;
            } else {
                while (st.size() > 0 && st.peek() <= arr[i]) {
                    st.pop();
                }

                if (st.size() == 0) {
                    nger[i] = -1;
                } else {
                    nger[i] = st.peek();
                }
            }

            st.push(arr[i]);
        }

        return nger;
    }
}
```

N times

N time
life time



St

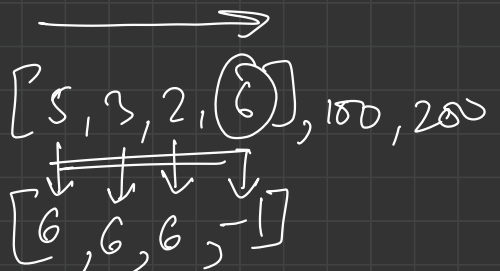
Time Complexity : O(N)

Space Complexity : O(N)

↳ Stack Space

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 ones encountered,

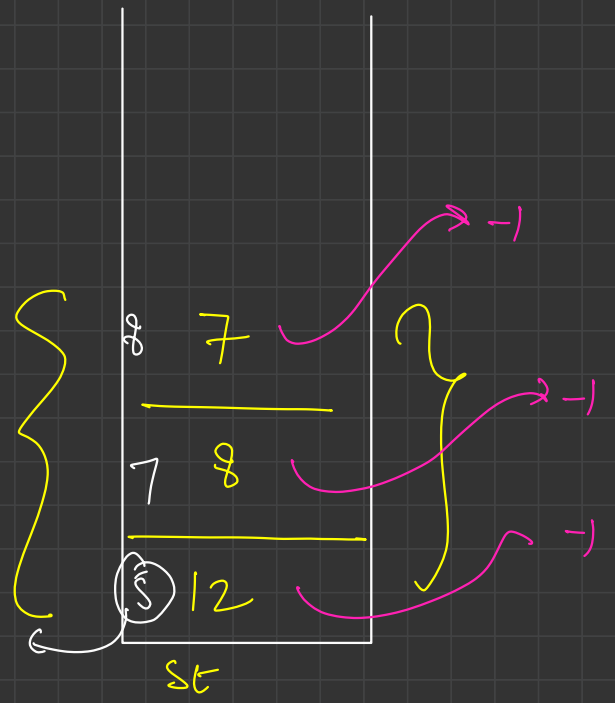
arr[] : [2, 5, 9, 3, 1, 12, 6, 8, 7]

5 9 12 12 12 -1 8 -1 -1

0 1 2 3 4 5 6 7 8

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

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



```java
// 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;
}
}
```

N times

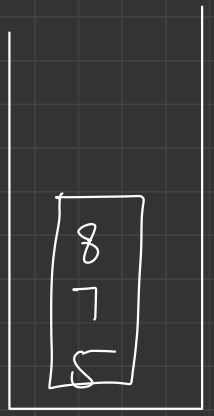
N times
Lifetime

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

0 1 2 3 4 5 6 7 8

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

5, 9, 12, 12, 12, -1, 8, -1, -1

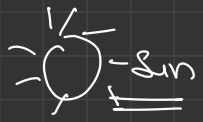


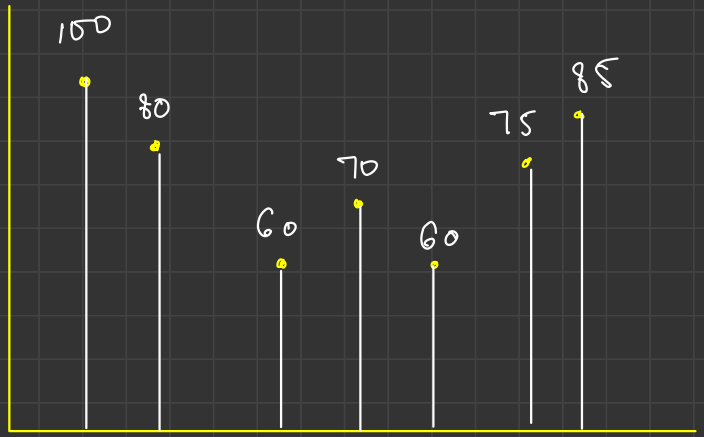
Stock span

Stock[] = [100, 80, 60, 70, 60, 75, 85]

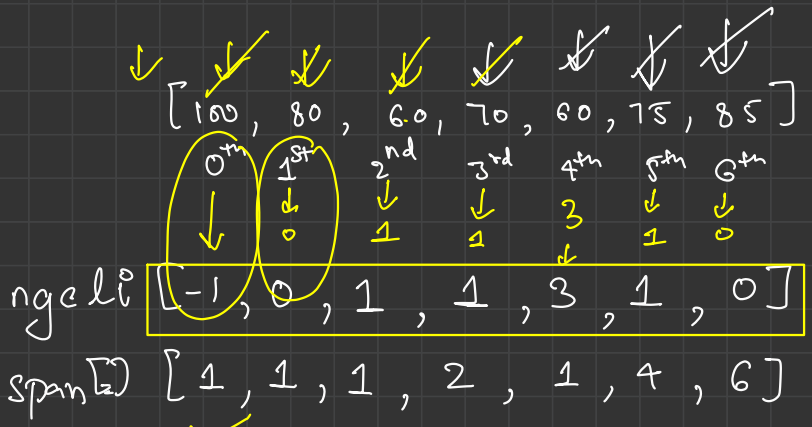
0th 1st 2nd 3rd 4th 5th 6th

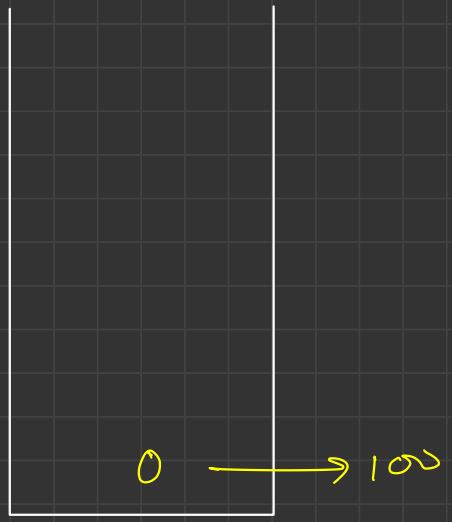
2 1 4





1, 1, 1, 2, 1, 4, 6


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


0
100

```java
class Solution {
    static int[] nextGreaterEleOnLeftIdx(int[] a, int n) {
        int[] ngeli = new int[n];
        Stack<Integer> st = new Stack<>();
        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;
    }
}
```

SC O(N)

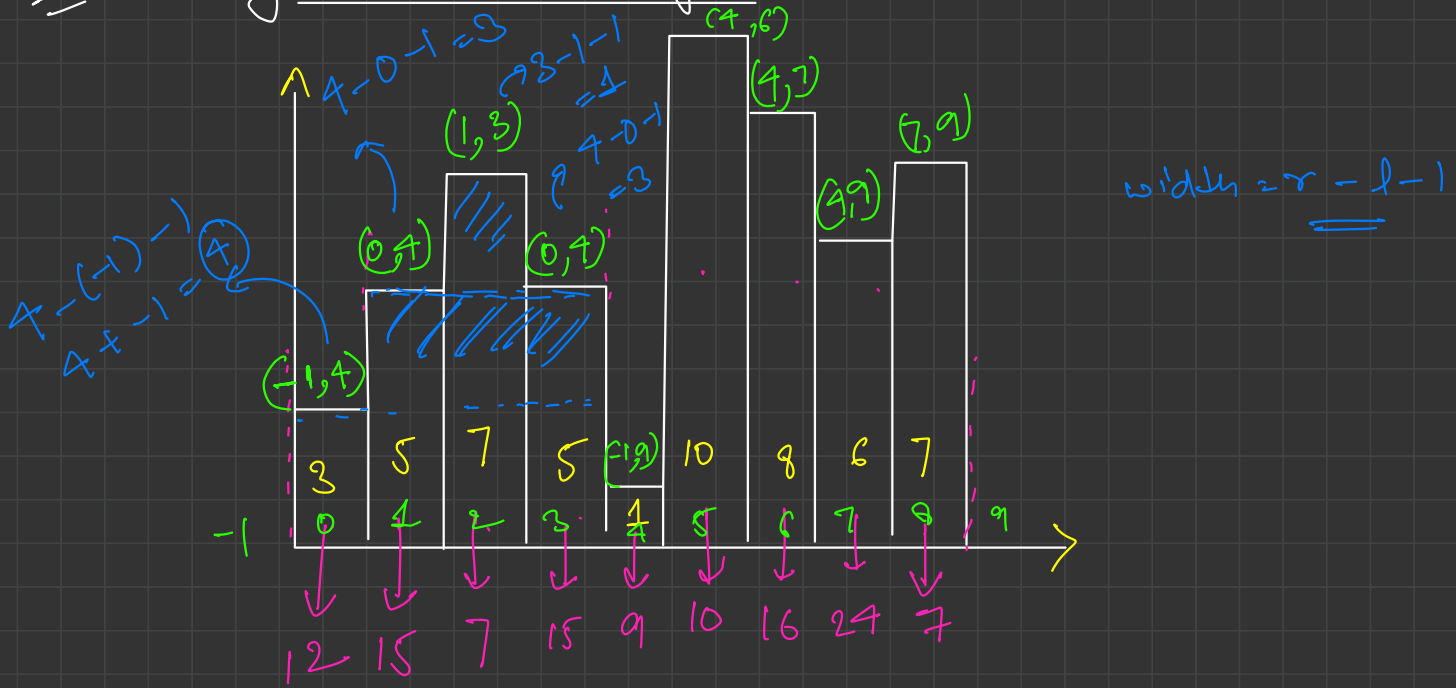
O(N)

O(N)

TC: O(N)

SC: O(N)

Q largest Area Histogram



Steps

① get next smaller element by idx on left

② get next smaller ele by idx on right

③ width = r − l − 1 area = height × (width)

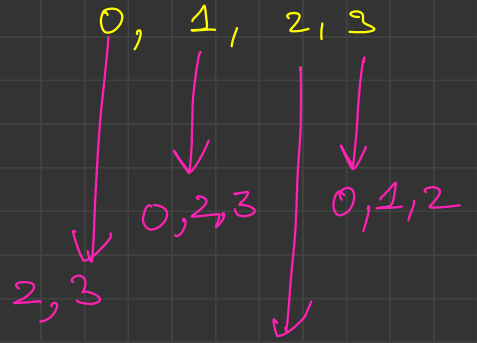
maxArea

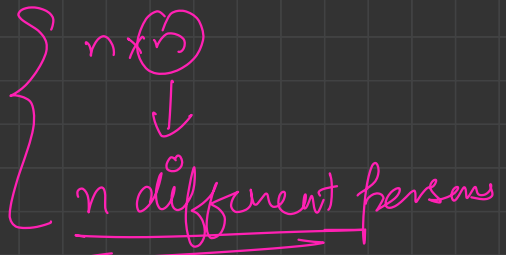
Celebrity Problem

	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)

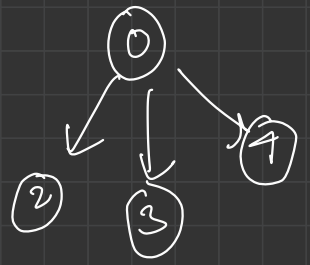
if i knows j arr[i][j] = 1

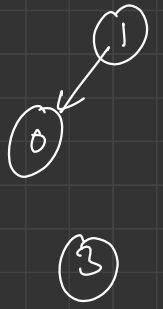
if i doesn't knows j arr[i][j] = 0

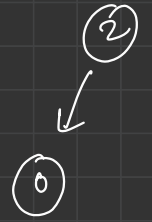


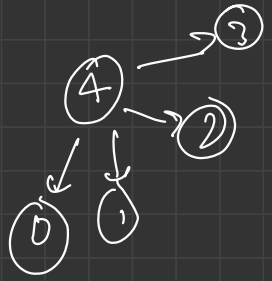


	0	1	2	3	4
0	0	0	1	1	1
1	1	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	1	1	1	1	0



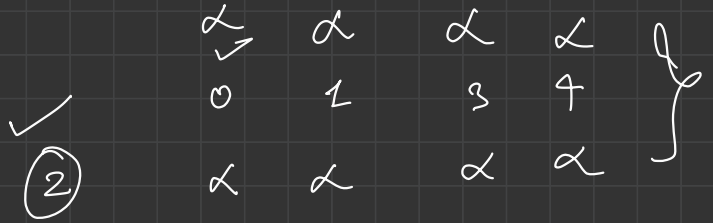






(1) He should not know anyone

(2) He should be known by everyone



Only One call at Max possible!

Brute force !TC: $O(N^2)$

elemination technique

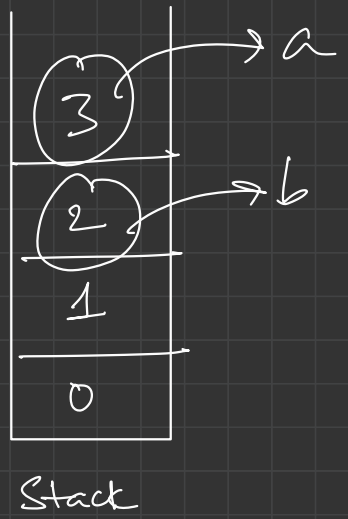
if arr[a][b] = 1 {a knows b}

b can be my celeb

if arr[a][b] = 0 {a doesn't know b}

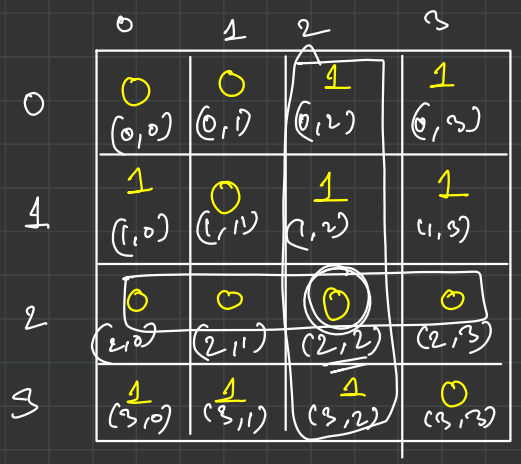
a can be my celeb

	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)



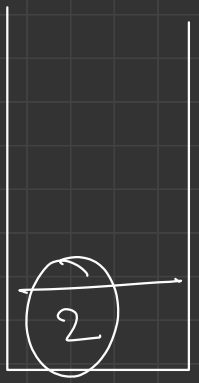
Step 1 get top 2 elements of stack

```
If arr[a][b] : 1 , st.push(b) ;
else st.push(a);
```

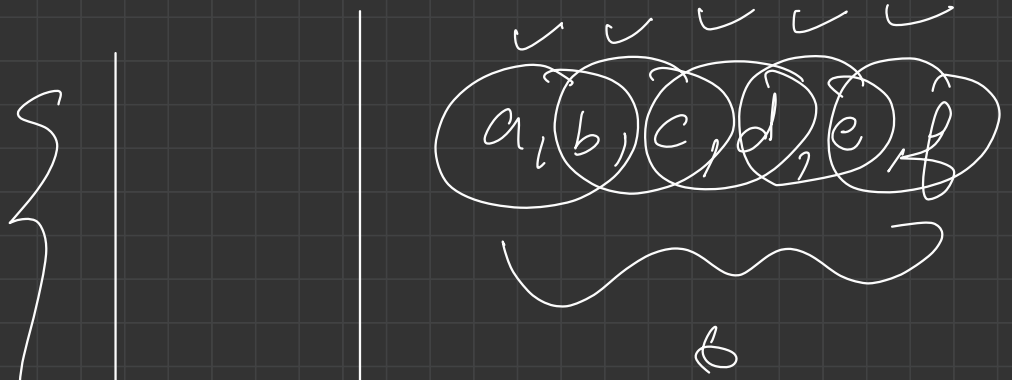



```java
class Solution {
    int findCelebrity(int M[][], int n) {
        // step 1: add everyone to the stack
        Stack<Integer> possibleCelebs = new Stack<>();
        for (int i = 0; i < n; i++) {
            possibleCelebs.push(i);
        }

        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 knows 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;



celeb = 2



TC: $O(N) + O(N) + O(N) = O(3N) = O(N)$

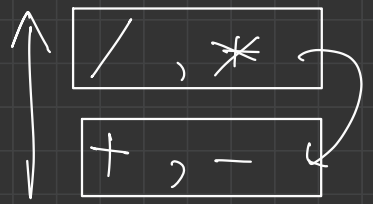
SC: $O(N)$

Evaluate Infix Expressions

Rule BODMAS

① first Brackets are evaluated

② precedence:



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

① 2 * 4 / 2 + 6

② 8 / 2 + 6

③ 4 + 6

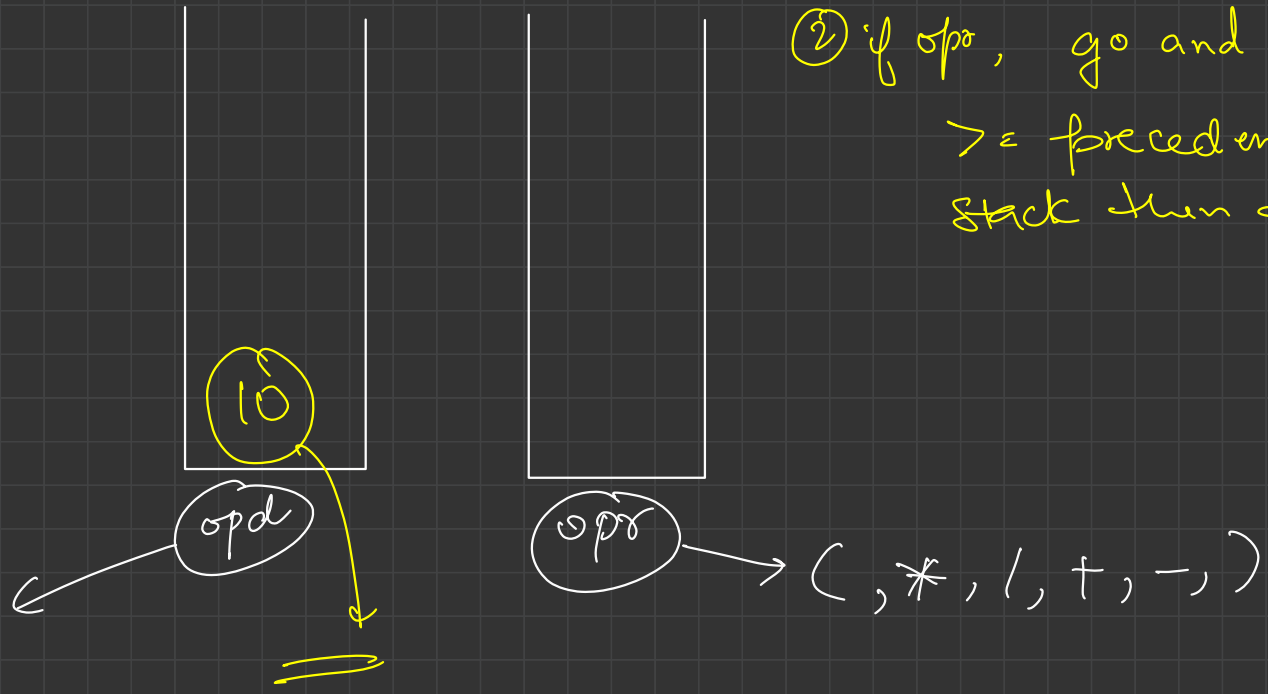
④ 10

/ — 2 + → 1

* — 2 - → 1

2 * 4 / (4 / 2) + 6

a/b



① if opd, opd.push(cn)

② if opr, go and solve >= precedence from stack then add me.

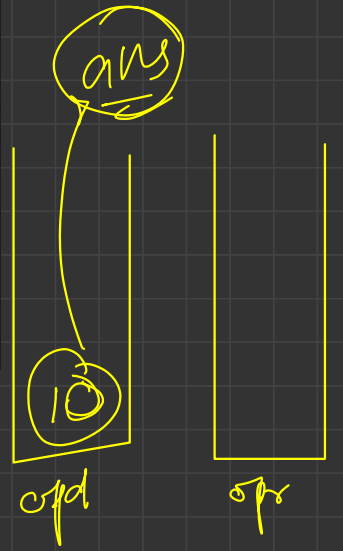
$'9' - '0' = ?$

```java
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;
    }
}
```

```java
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);
            }

            // remove the '(' bracket
            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);
        }
    }
```

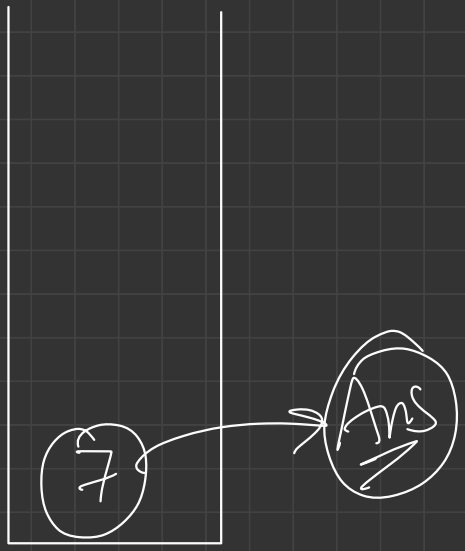
```java
    while (opr.size() > 0) {
        int v2 = opd.pop();
        int v1 = opd.pop();

        char o = opr.pop();

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

        opd.push(ans);
    }

    return opd.peek();
}
```

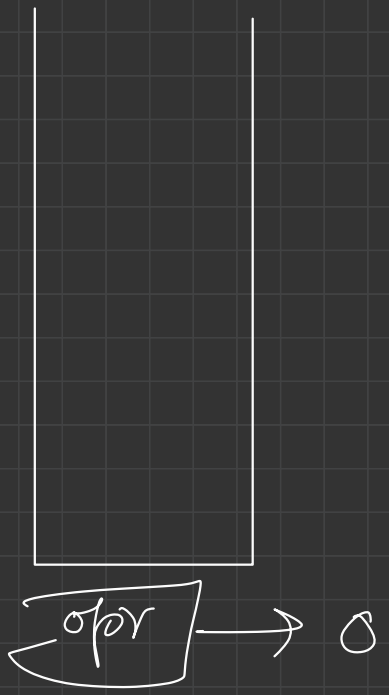
2 * 4 / (4 / 2) + 6



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

v1 * v2 = 8





Convert Infix to postfix and prefix.

2 * 4 / (4 / 2) + 6 ⟶ Infix

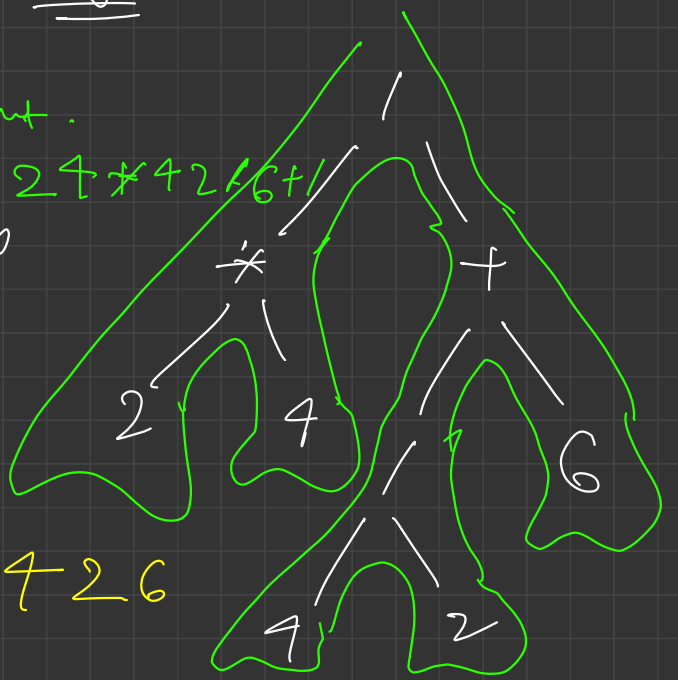
left → right → root.

postfix opd + opr

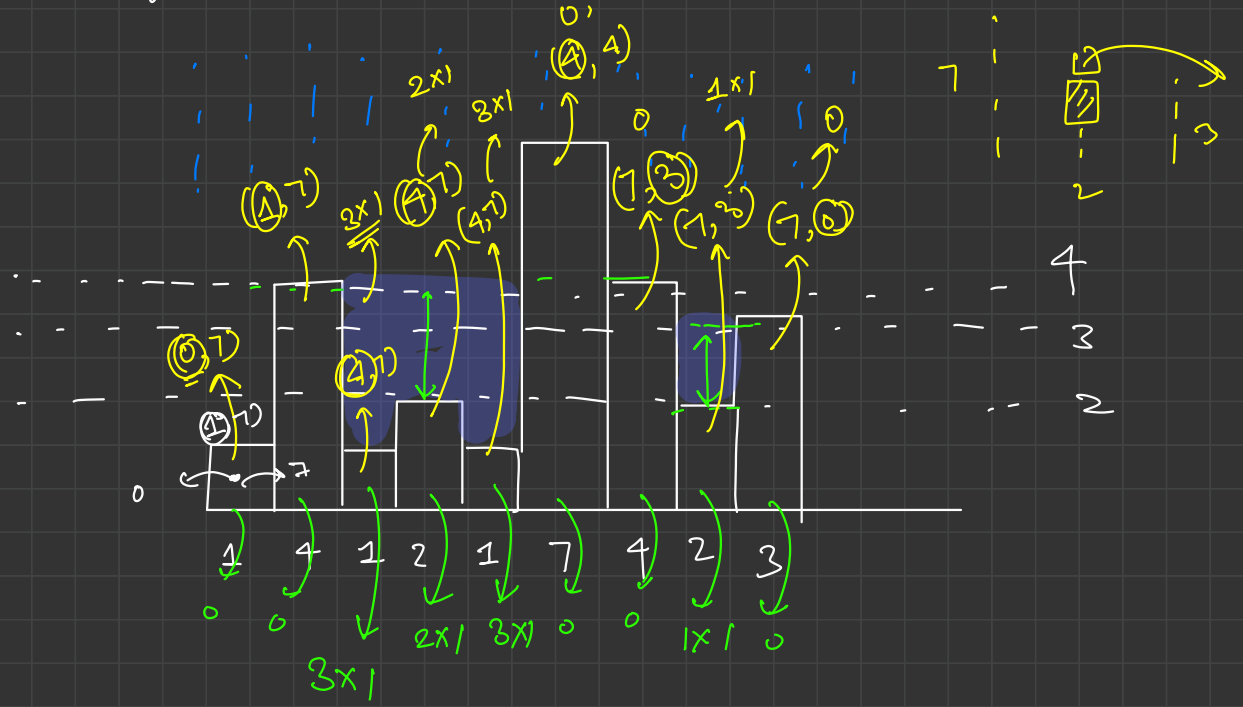
prefix opr + opd

root → left → right

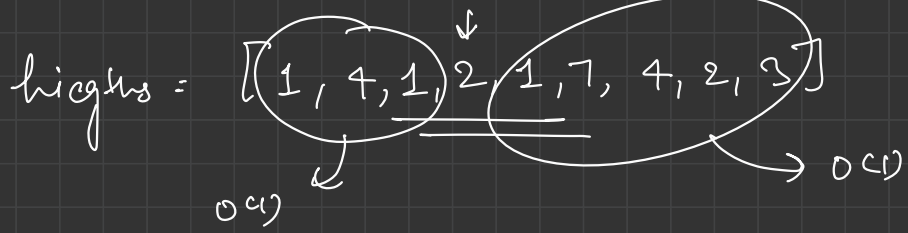
/ * 2 4 + / 4 2 6



Trapping Rain Water



$0+0+3+2+3+0+0+1+0 = 9$ units of water



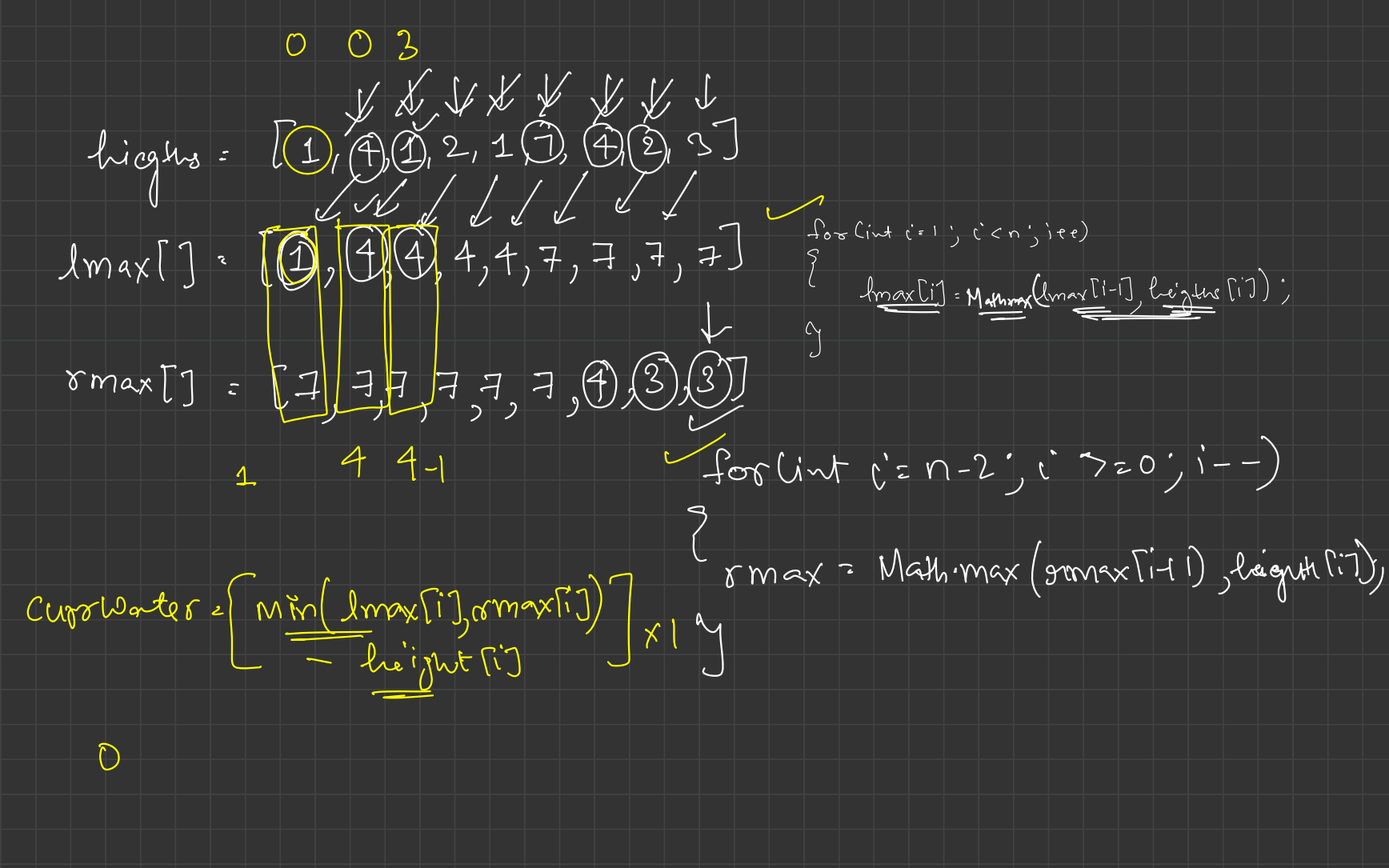
maxWater
+= (min(lm, rm) − h) × 1

O(N) → work for Each Element

Brute force Method !

TC : O(N × N) = O(N²)

SC : O(1)


0 0 3
heights = [1, 4, 1, 2, 1, 7, 4, 2, 3]
lmax[] = [1, 4, 4, 4, 4, 7, 7, 7, 7]
rmax[] = [7, 7, 7, 7, 7, 7, 4, 3, 3]
1 4 4-1
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]);
}
CupsWater = [Min(lmax[i], rmax[i]) - height[i]] x1
0

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

TC : O(3N) = O(N)

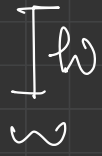
SC : O(N)

lmax[] : [1, 4, 4, 4, 4, 7, 7, 7, 7]

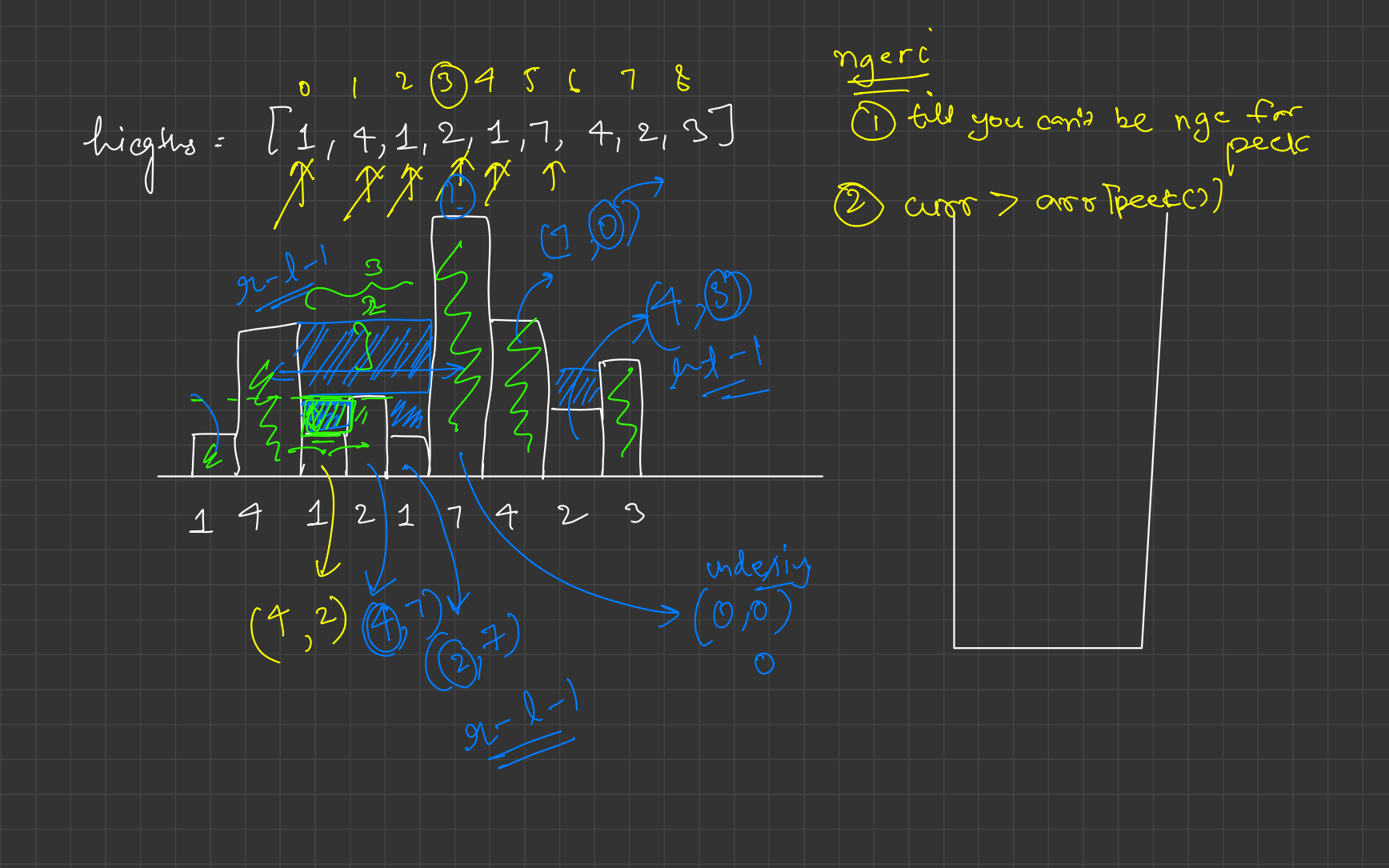
rmax[] : [7, 7, 7, 7, 7, 7, 4, 3, 3]

maxHeight 1

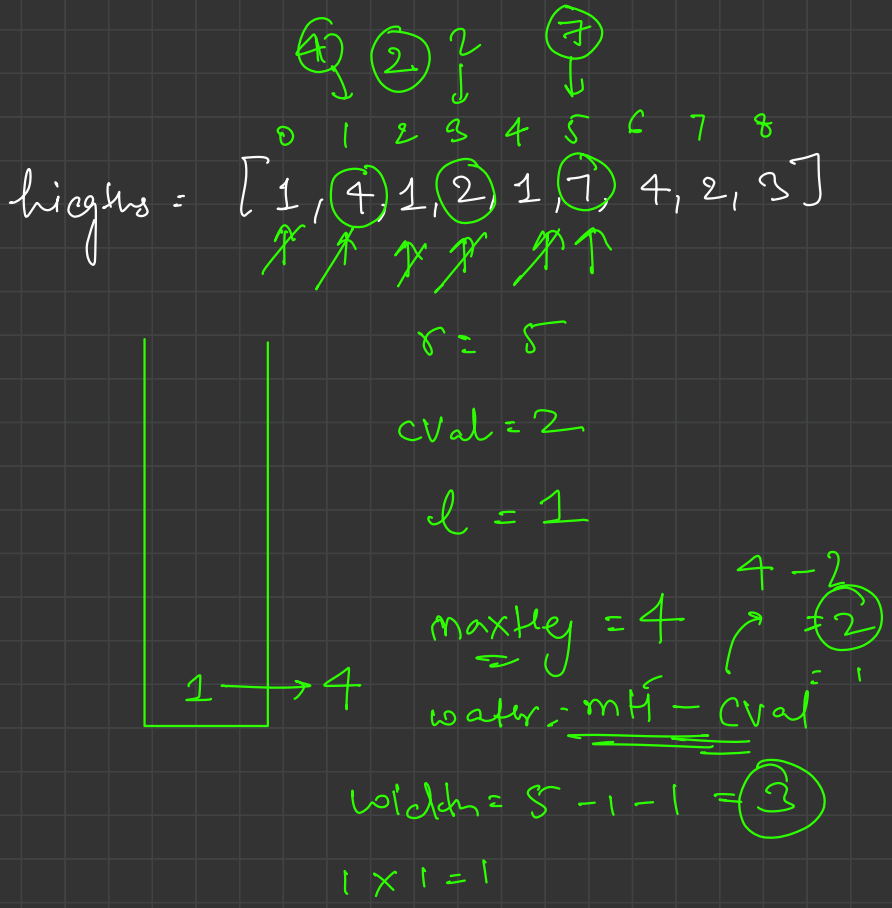
water height = 1 - 1



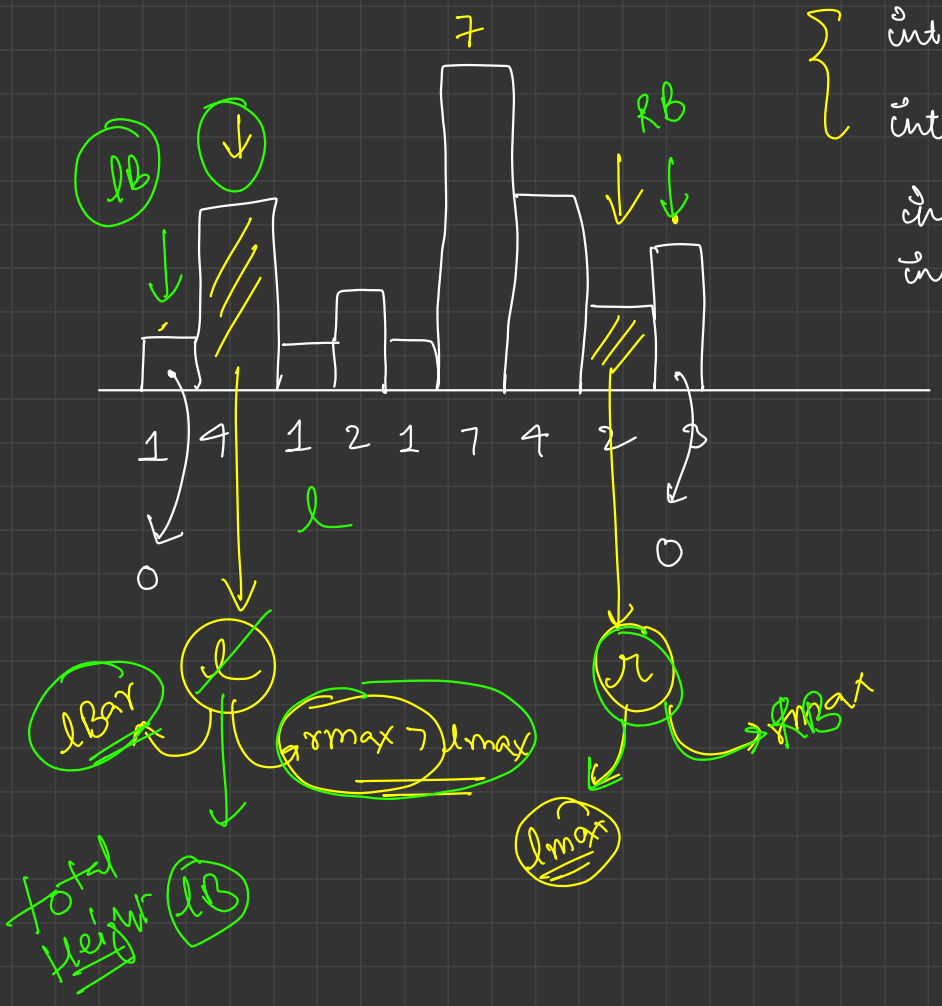
area = height × 1


0 1 2 3 4 5 6 7 8
heights = [1, 4, 1, 2, 1, 7, 4, 2, 3]
1 4 1 2 1 7 4 2 3
(4, 2)
(0, 0)
0

```java
// 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);
}
```



{TC: O(N)
SC: O(N)}



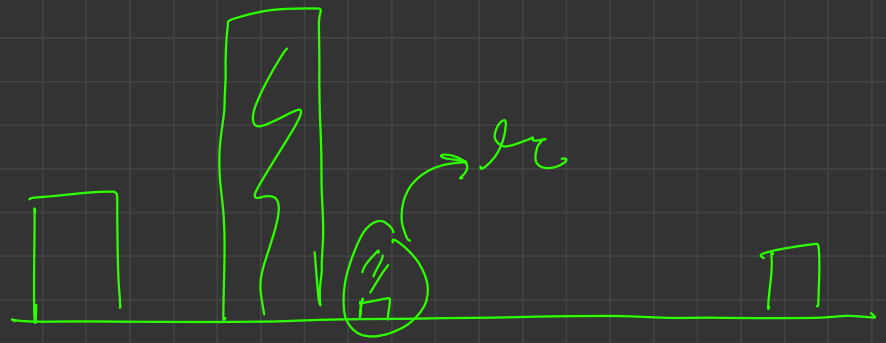

```
int leftBar = arr[0];   1
int rightBarr = arr[n-1]; 3
int l = 1;
int r = n-2;
```

```
if (leftBar < RightBarr)
{
    if (lB < arr[l])
        lB = arr[l]
    else
        water += (lB - arr[l])
    l++;
}
else
{
    if (RB < arr[r])
        RB = arr[r];
    else
        water += (RB - arr[r])
    r--;
```

$lB < RB$

TH

lB

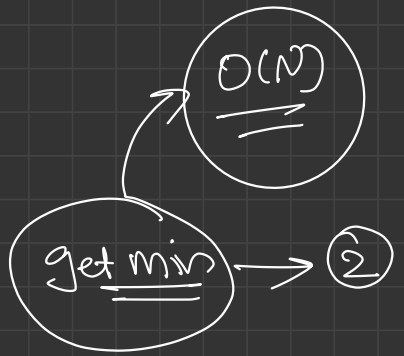


$RB < dB$

Height : RB

Minimum Stack

2, 3, 4, 1, 10, 5

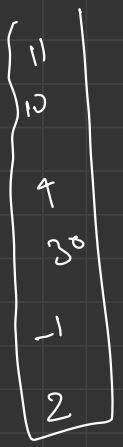


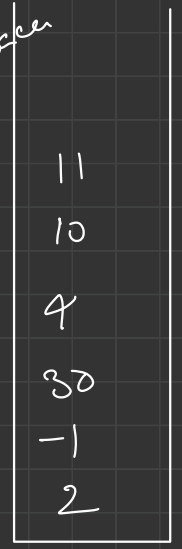
```java
// TC: O(N), SC: O(N)
int getMin() {
    // Write Code here
    if (mainStack.size() == 0) {
        return -1;
    }

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

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

    return minVal;
}
```

O(N)

O(N) space

O(N)





mainStack

temp

minVal = MaxInt // 10 // -1

(2) -1, 4, 3, 4, 5

i)

push(2)

getMin()

push(-1)

pop()

push(4)

getmin()

push(3)

push(4)

getmin()

push(5)

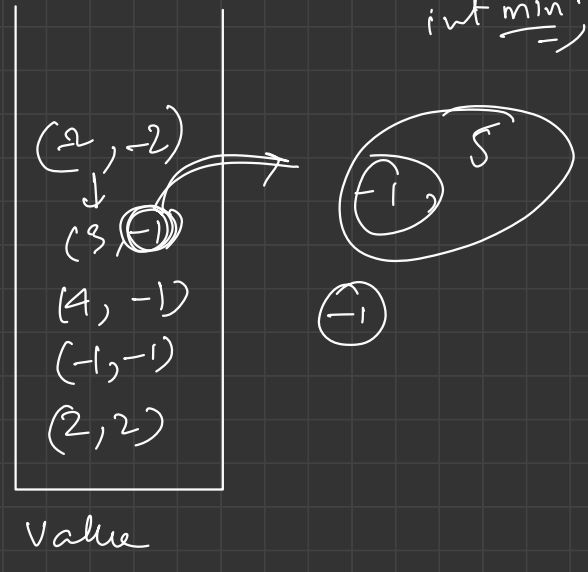
getMin() ⟶ O(1) time Complexity ?

2, -1, 4, 3, 4, 5

class Pair {
  int val;
  int min;
}

push(-2)

push(2)
push(-1)
push(4)
push(3)
getMin() ⟶ st.peek().min
O(1)
= -1



→ push(4), getMin() → 4

→ push(3), getMin() → 3

→ push(7) → getMin() → 3

→ push(5), getMin() → 3

→ push(4), getMin() → 3

→ push(1), getMin() → 1

→ pop()

{ prevMin = 2 × minVal − st.peek() }

2 × 1 − (−1) : >

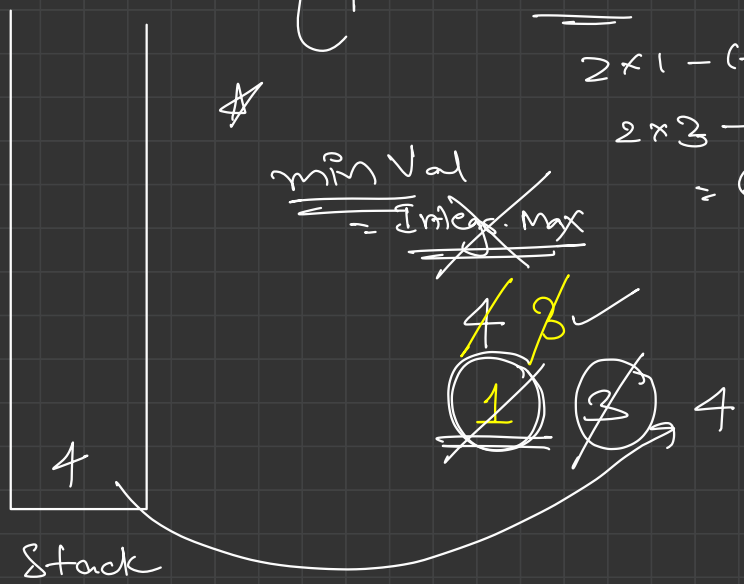
2 × 3 − (2)

: 6 − 2

= 4

minVal

= [illegible] Max



2 × 3 − 4

: 2

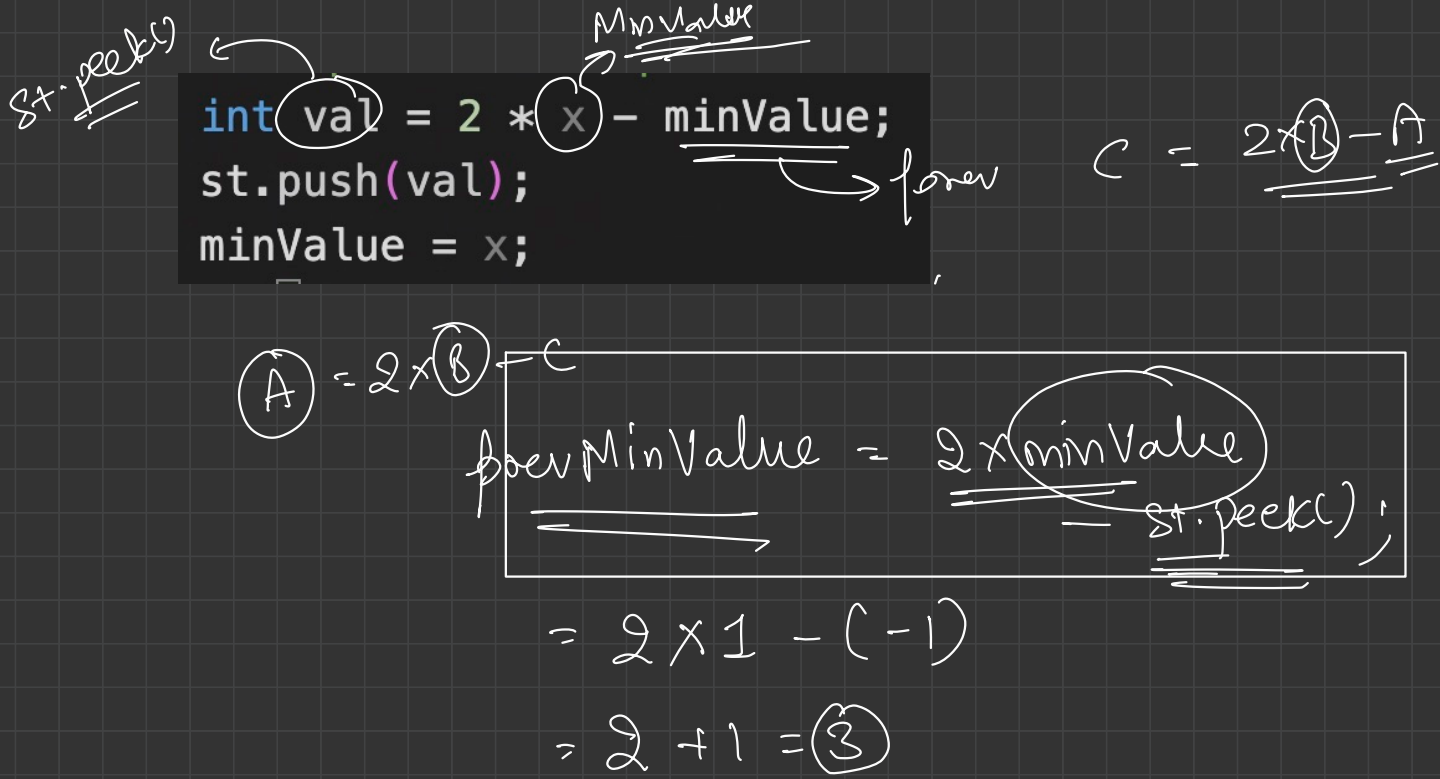
minVal — (x)

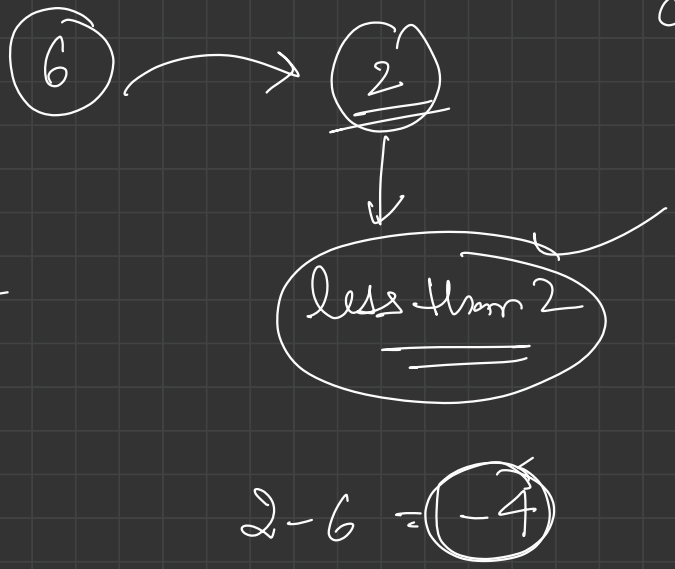
⊖ → —

★ Whenever you are updating minVal

then push (2 × x − minVal) ≠ val

newNum = x


st.peek()
minValue
int val = 2 * x - minValue;
st.push(val);
minValue = x;
prev
C = 2×B − A
A = 2×B − C
prevMinValue = 2×minValue − st.peek();
= 2×1 − (−1)
= 2 + 1 = 3



curr min = 2

2x2-6 = -2

Min

2x2 = 4 - 6

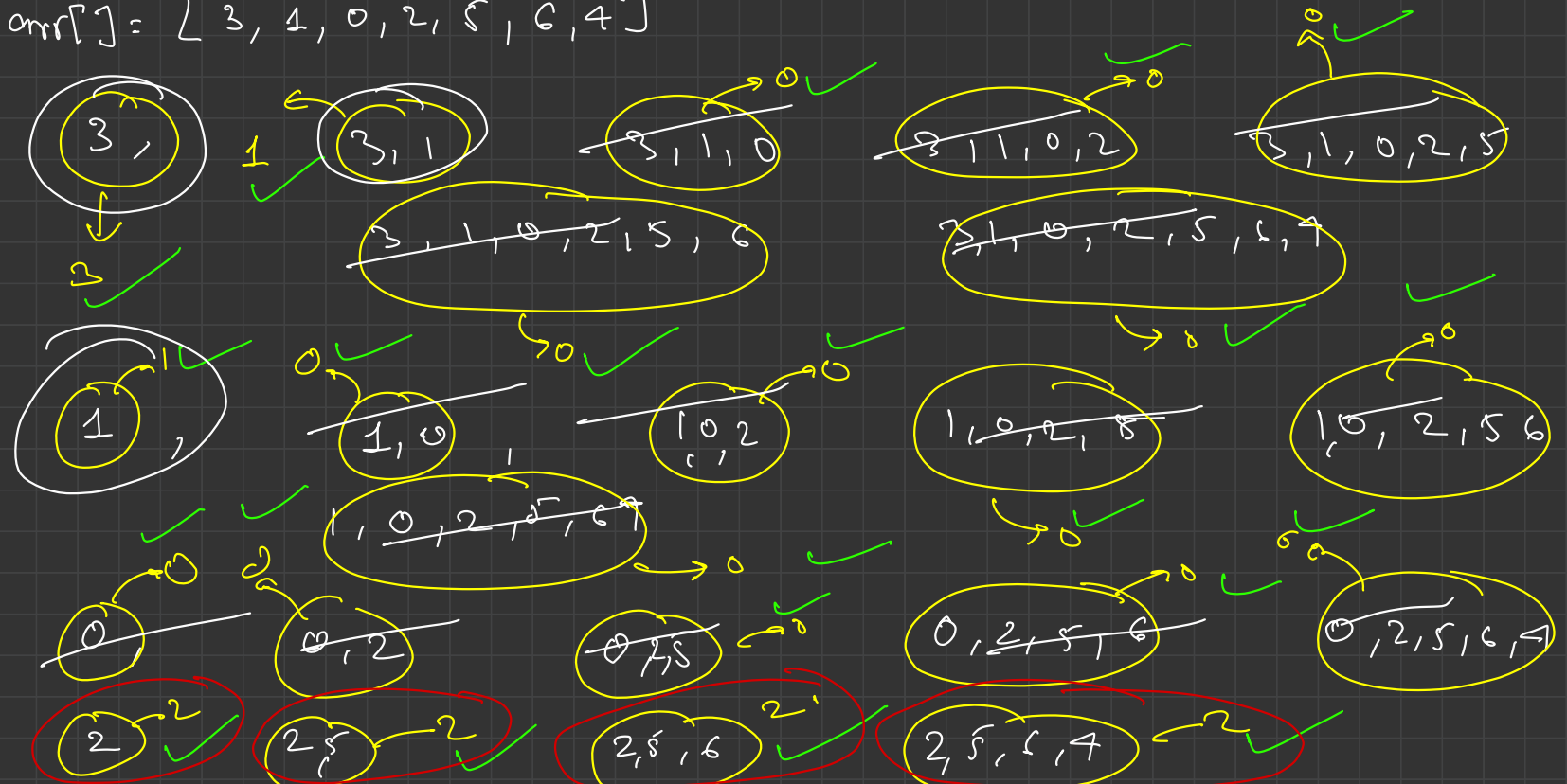
a + b - c = d

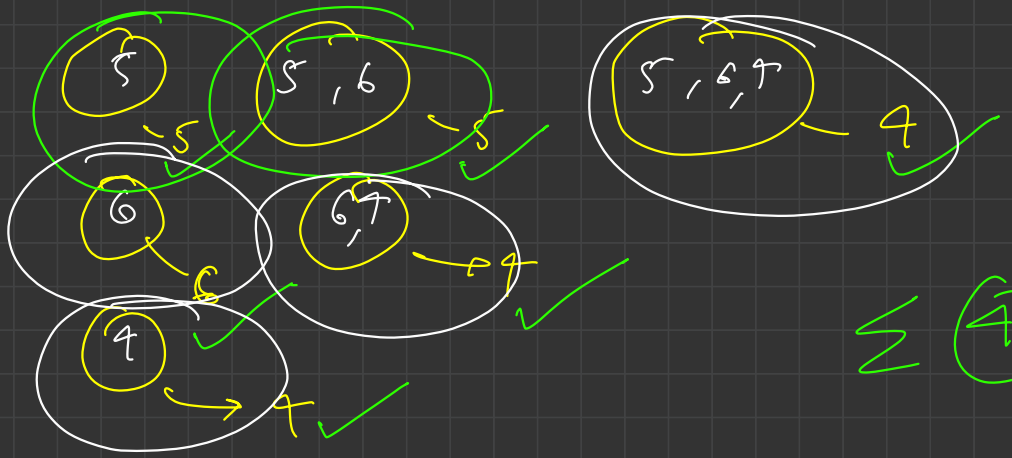
b < c

d < a

Sum of Subarrays Minimum

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





Brute force :

TC : $N \times N^2 = O(N^3)$

Smarter Brute force

TC : $O(N^2)$

{ TC: O(N) SC: O(N) }

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

{ 2 2,5 2,5,6 2,5,6,4 } ⟶ 2

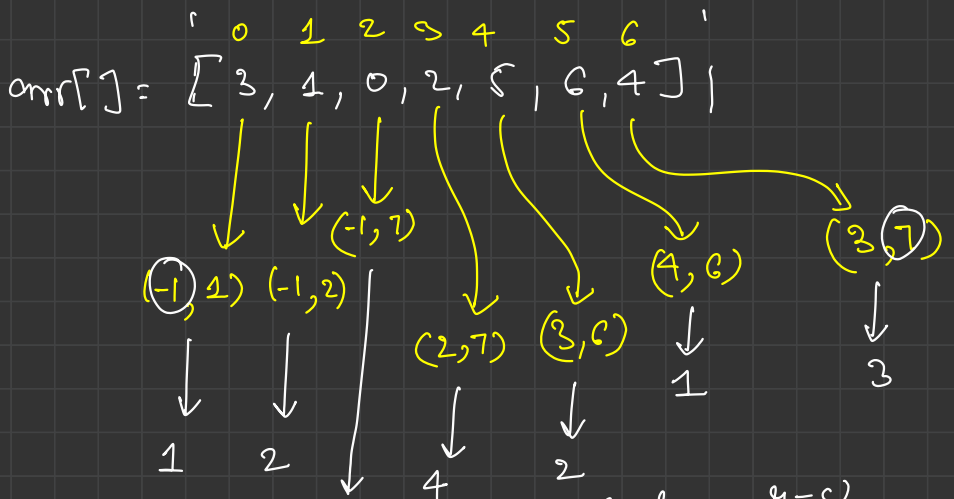
{ 5 5,6 } ⟶ 5

{ 4 6,4 5,6,4 } ⟶ 4

{ 6 } ⟶ 6

{ 3,1 } ⟶ 1

{ 3 } ⟶ 3



total Number of Subarrays with curr as min

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

$(c - l)$

$2 - (-1) = 3$

$7 - 2 = 5$

$(r - c)$

$\longrightarrow 3 \times 5 = 15$

Sum min of sum = x x curr val