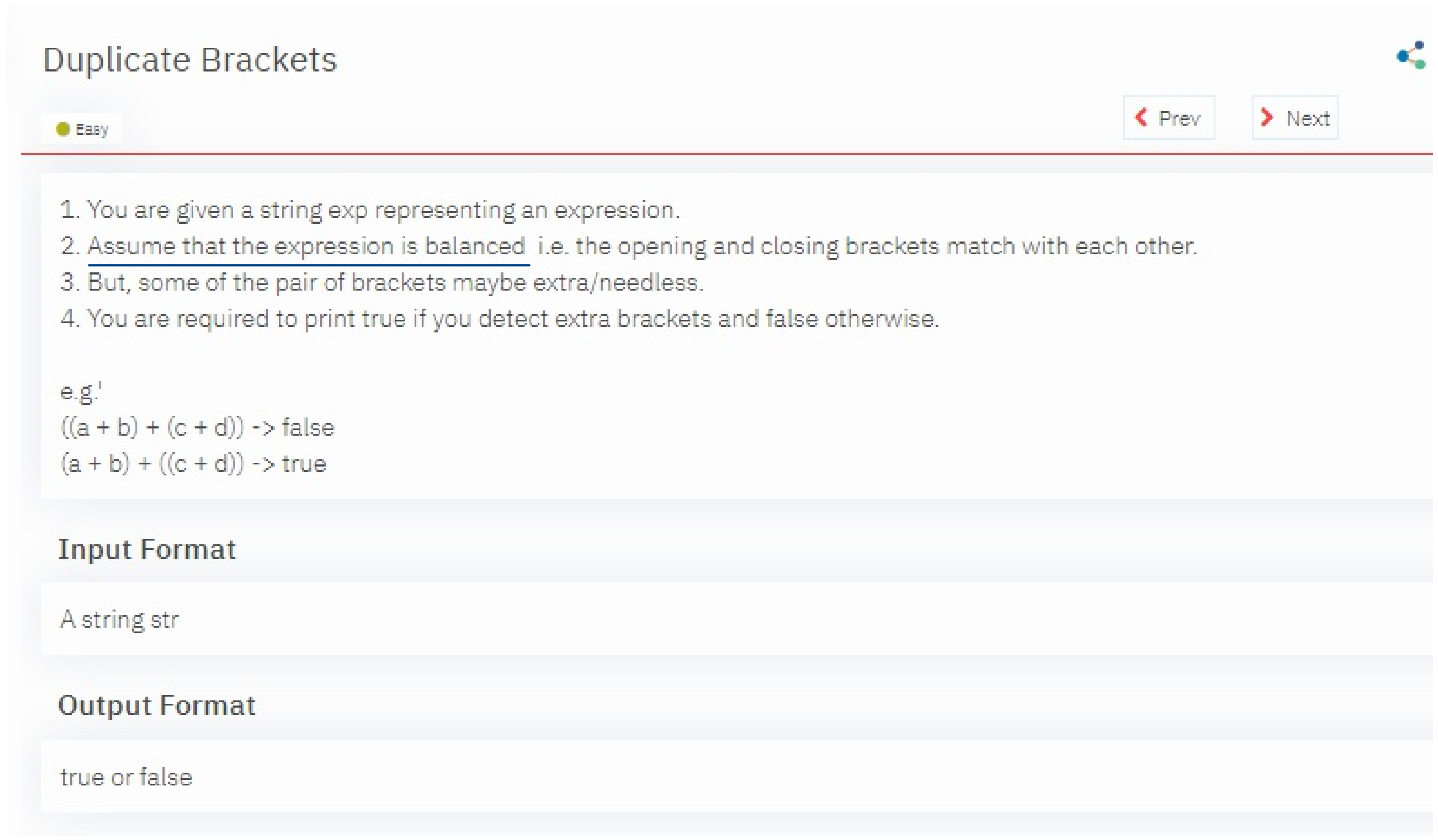


Duplicate Brackets

Easy

Prev Next

1. You are given a string exp representing an expression.
2. Assume that the expression is balanced i.e. the opening and closing brackets match with each other.
3. But, some of the pair of brackets maybe extra/needless.
4. You are required to print true if you detect extra brackets and false otherwise.

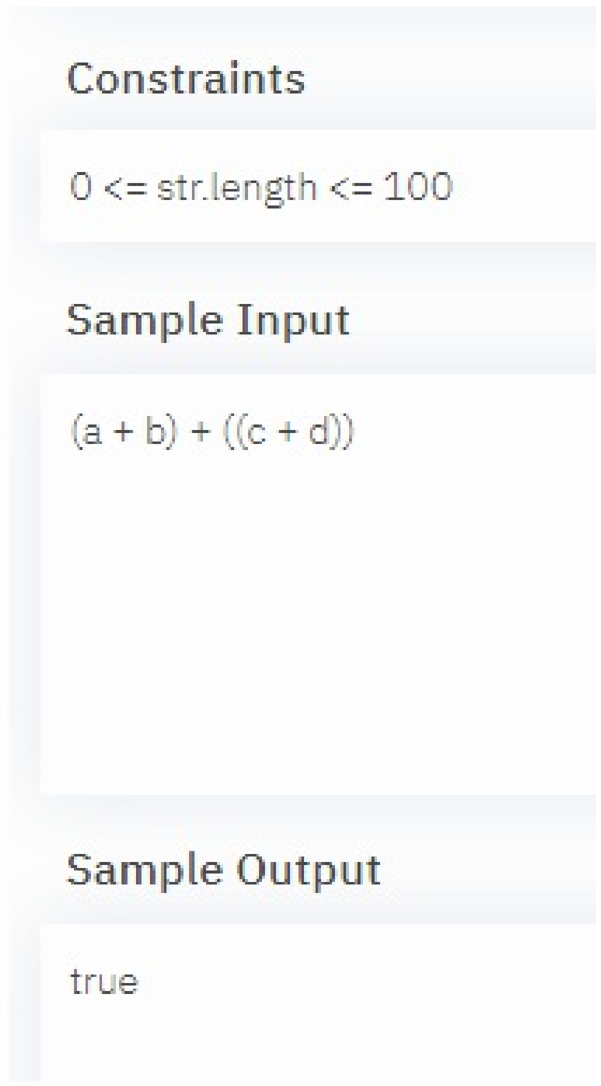
e.g.'
((a + b) + (c + d)) -> false
(a + b) + ((c + d)) -> true

Input Format

A string str

Output Format

true or false



Constraints

0 <= str.length <= 100

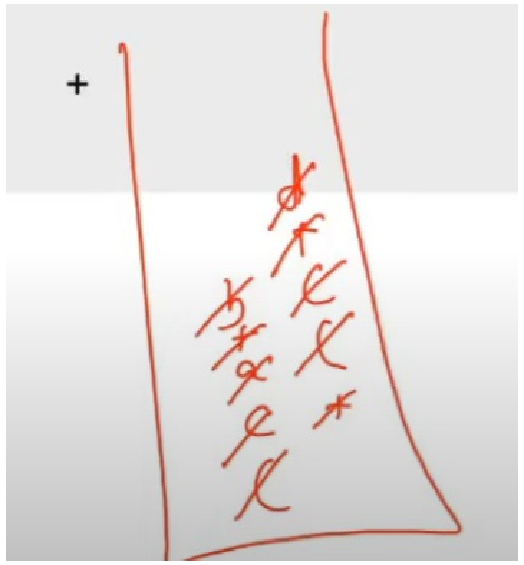
Sample Input

```
(a + b) + ((c + d))
```

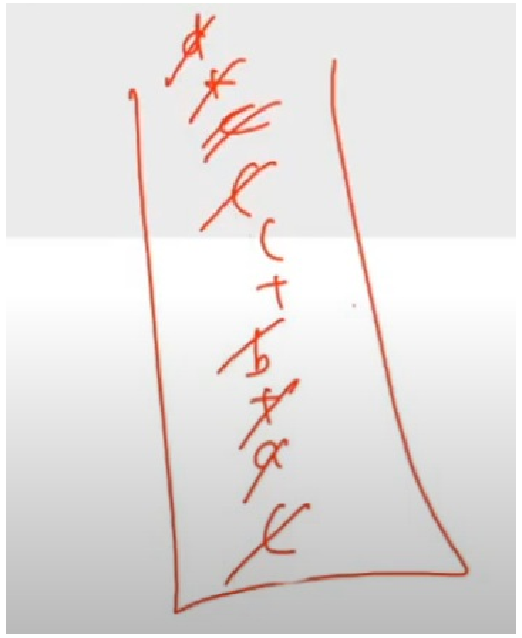
Sample Output

```
true
```

((a + b) + (c + d)) -> false



(a + b) + ((c + d)) -> true



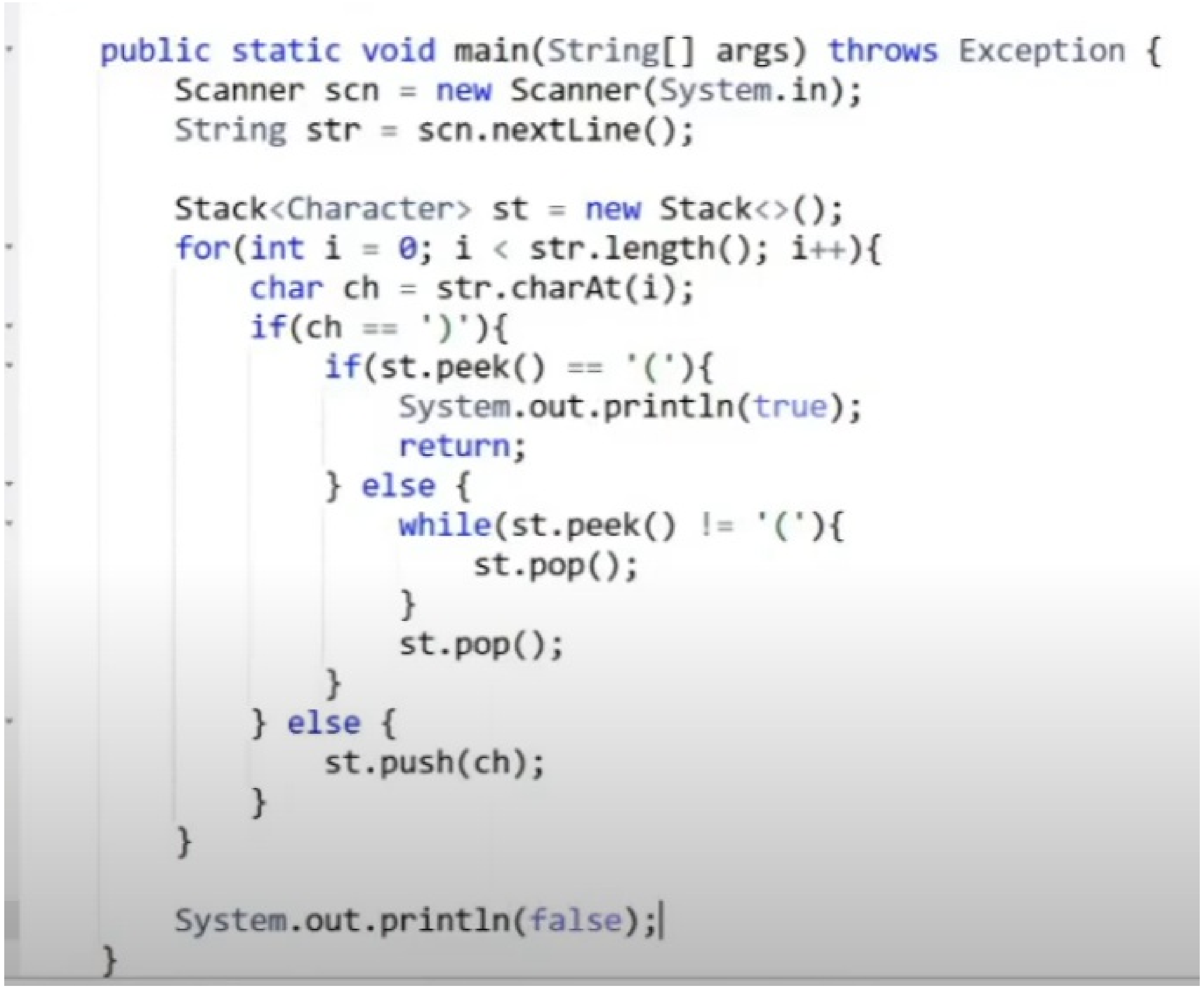
Approach:
push if (, operand, operator
and
if)
then while (peek != openging bracket)
            pop()
pop() - opening bracket as well;

Here when you notice,
for the last closing bracket, you will see the peek element as opening bracket

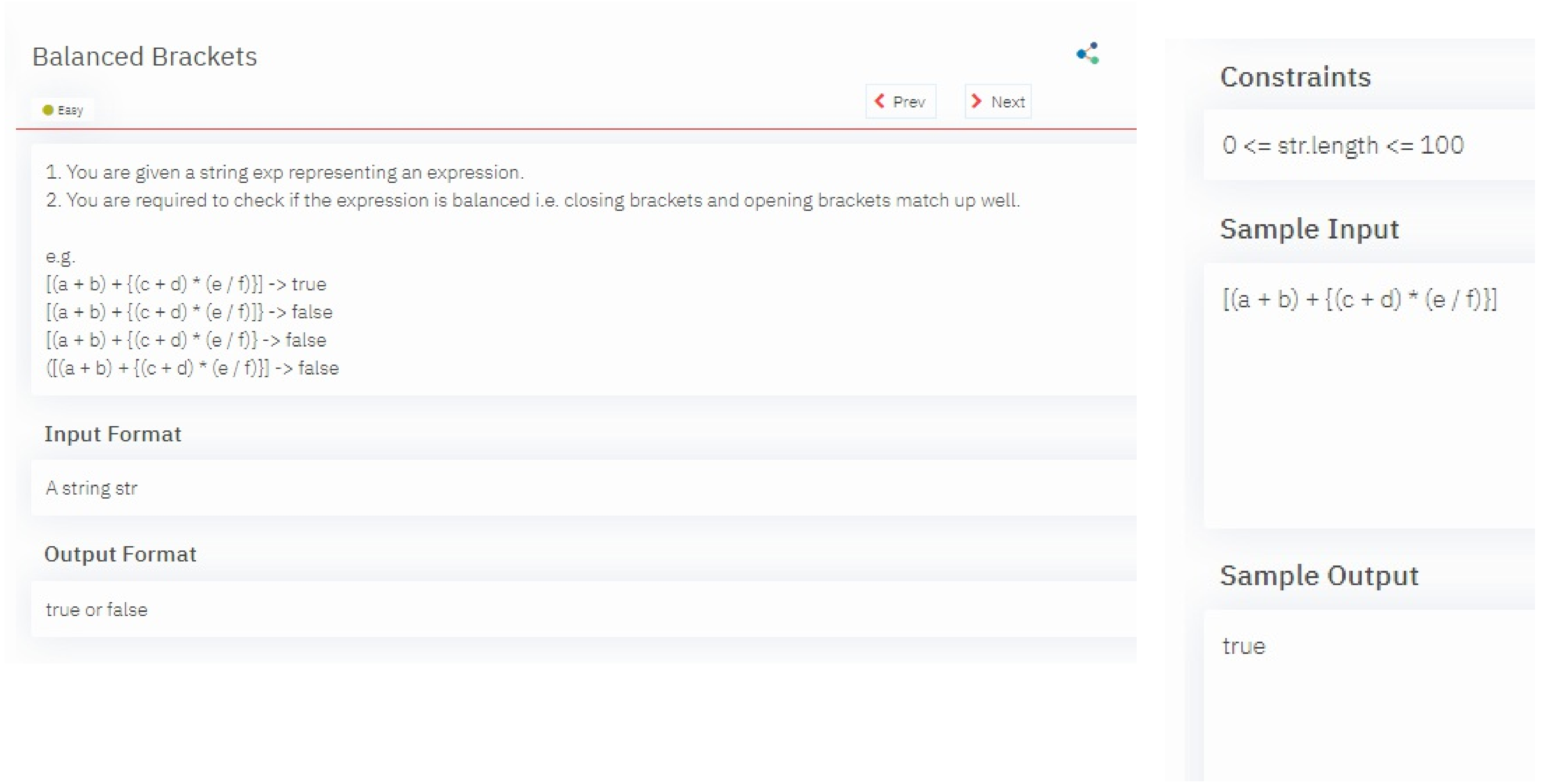
matlab dono bracket k bich me there is no content

i.e it does not contain element
which indicate it is duplicate bracket which is not required

CODE



```java
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();

    Stack<Character> st = new Stack<>();
    for(int i = 0; i < str.length(); i++){
        char ch = str.charAt(i);
        if(ch == ')'){
            if(st.peek() == '('){
                System.out.println(true);
                return;
            } else {
                while(st.peek() != '('){
                    st.pop();
                }
                st.pop();
            }
        } else {
            st.push(ch);
        }
    }

    System.out.println(false);
}
```



Balanced Brackets

1. You are given a string exp representing an expression.
2. You are required to check if the expression is balanced i.e. closing brackets and opening brackets match up well.

e.g.
[(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

Input Format

A string str

Output Format

true or false

Constraints

0 <= str.length <= 100

Sample Input

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

Sample Output

```
true
```

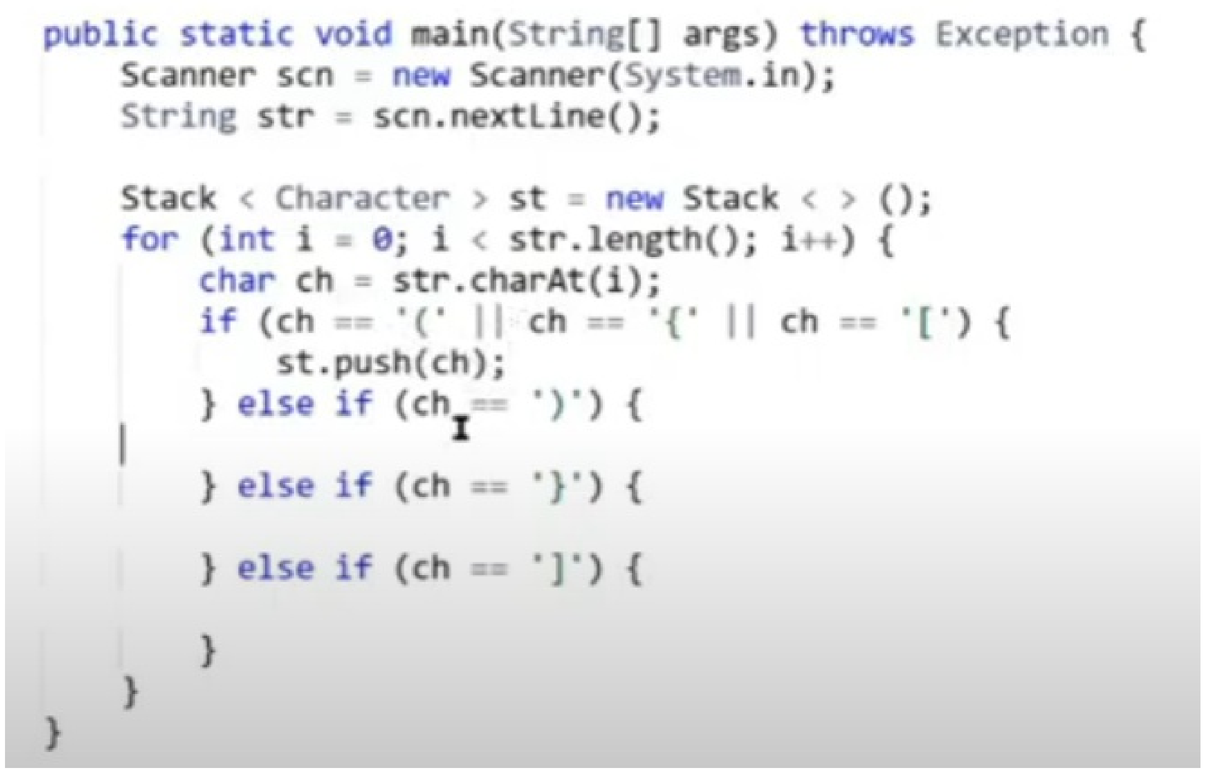
[(a + b) + {(c + d) * (e / f)}] -> true

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

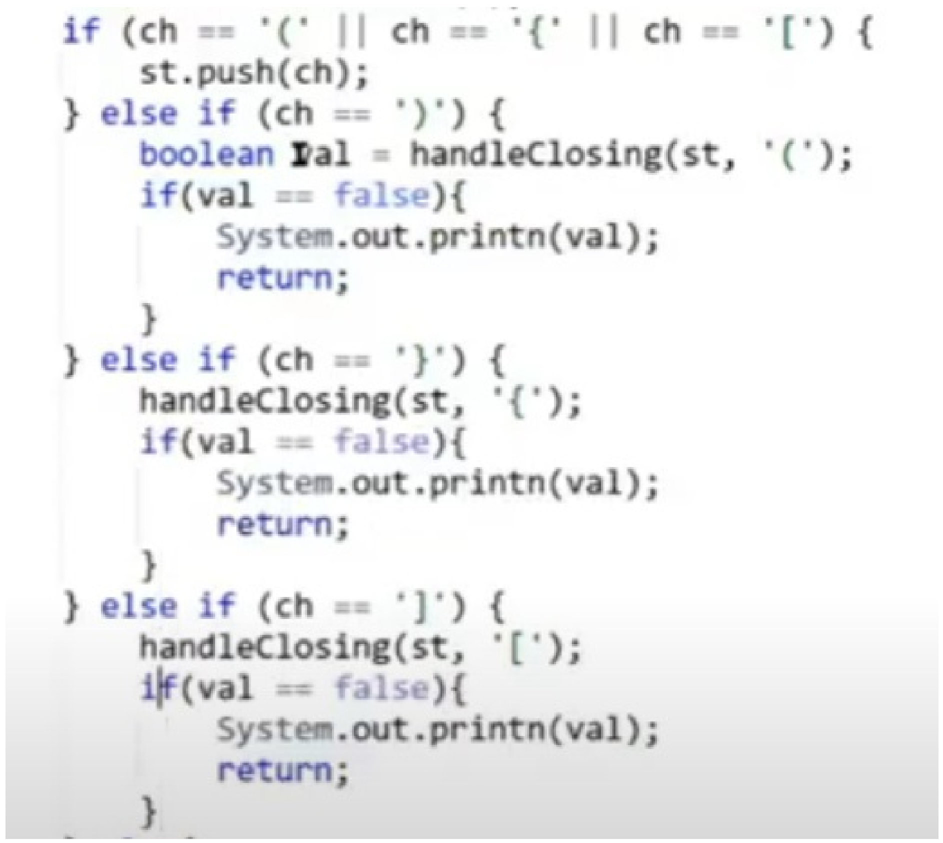
[(a + b) + {(c + d) * (e / f)} -> false

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

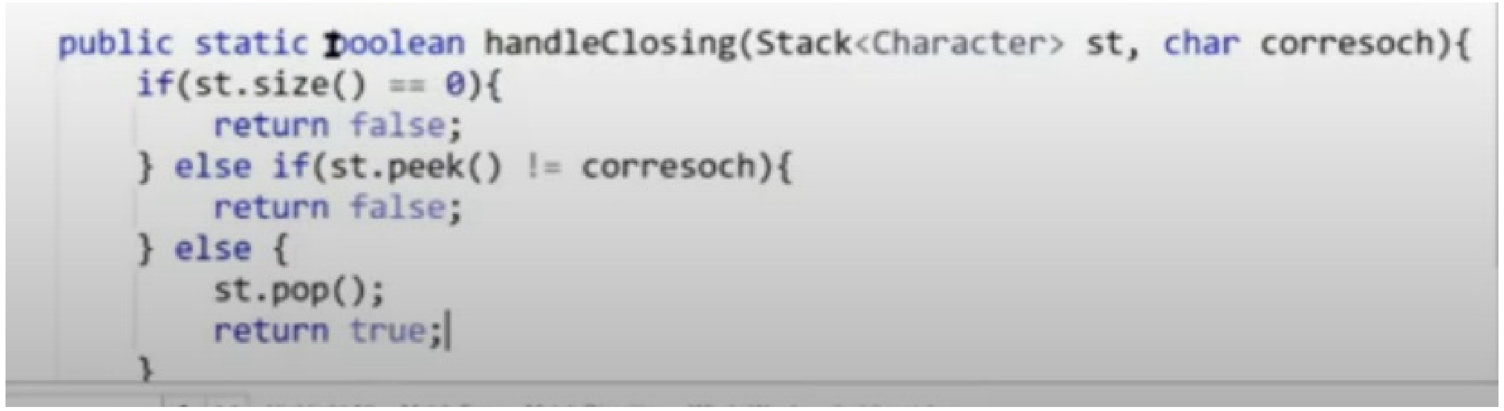
CODE



```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();

    Stack < Character > st = new Stack < > ();
    for (int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        if (ch == '(' || ch == '{' || ch == '[') {
            st.push(ch);
        } else if (ch == ')') {

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

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

        }
    }
}
```

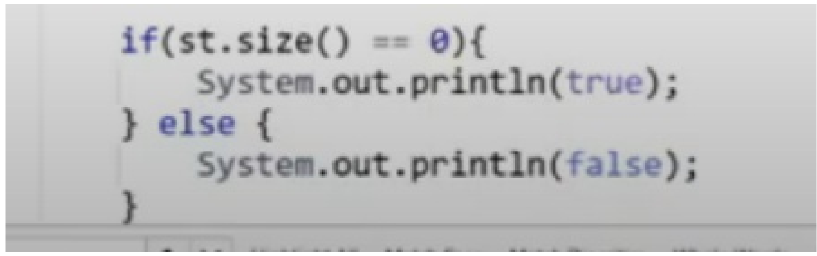


```
if (ch == '(' || ch == '{' || ch == '[') {
    st.push(ch);
} else if (ch == ')') {
    boolean val = handleClosing(st, '(');
    if(val == false){
        System.out.printn(val);
        return;
    }
} else if (ch == '}') {
    handleClosing(st, '{');
    if(val == false){
        System.out.printn(val);
        return;
    }
} else if (ch == ']') {
    handleClosing(st, '[');
    if(val == false){
        System.out.printn(val);
        return;
    }
```

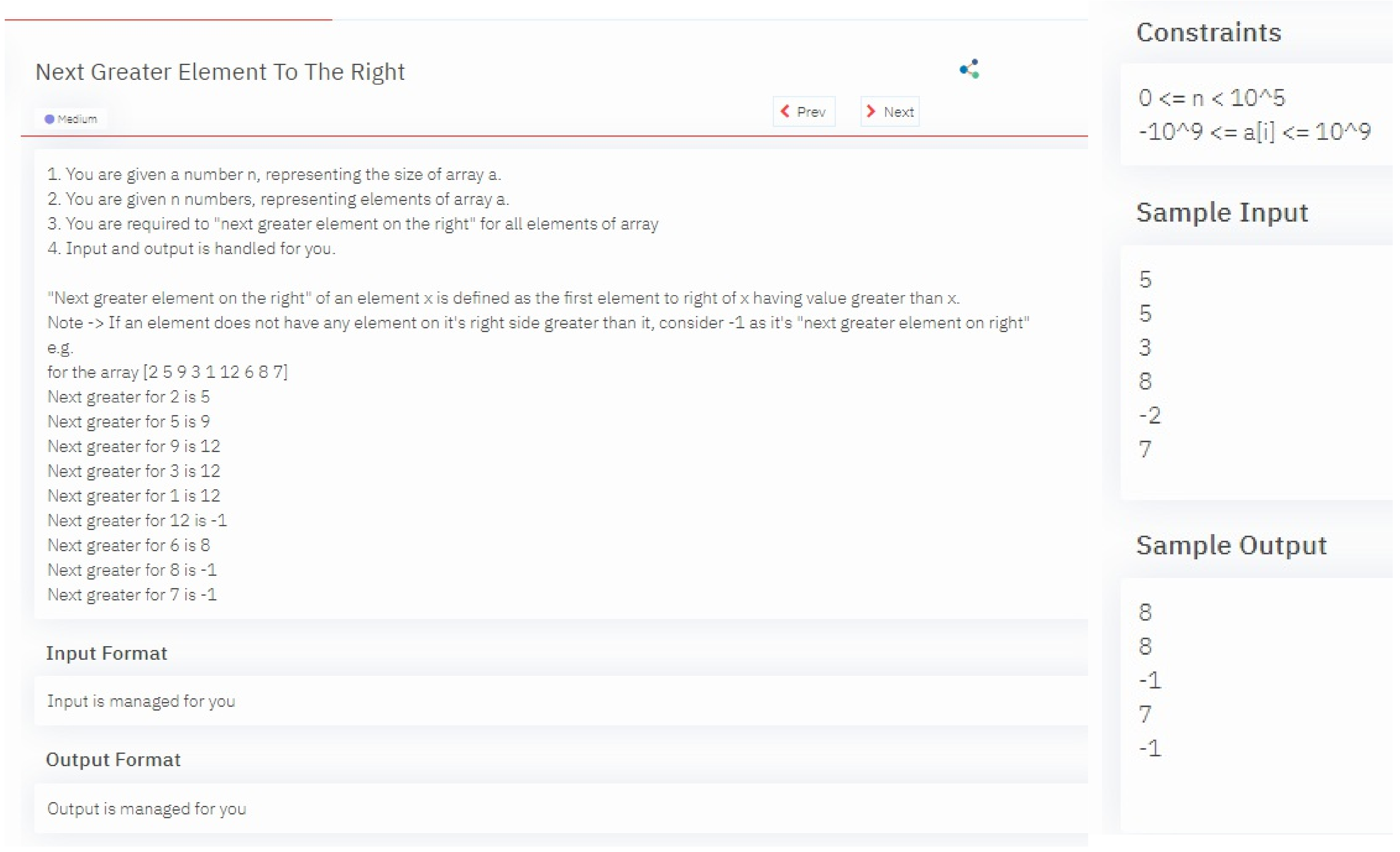


```
public static boolean handleClosing(Stack<Character> st, char corresoch){
    if(st.size() == 0){
        return false;
    } else if(st.peek() != corresoch){
        return false;
    } else {
        st.pop();
        return true;
    }
```

this is outside loop, to handle the only opening bracket case



```
if(st.size() == 0){
    System.out.println(true);
} else {
    System.out.println(false);
}
```


Next Greater Element To The Right

Medium

Prev Next

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing elements of array a.
3. You are required to "next greater element on the right" for all elements of array
4. Input and output is handled for you.

"Next greater element on the right" of an element x is defined as the first element to right of x having value greater than x.
Note -> If an element does not have any element on it's right side greater than it, consider -1 as it's "next greater element on right"
e.g.
for the array [2 5 9 3 1 12 6 8 7]
Next greater for 2 is 5
Next greater for 5 is 9
Next greater for 9 is 12
Next greater for 3 is 12
Next greater for 1 is 12
Next greater for 12 is -1
Next greater for 6 is 8
Next greater for 8 is -1
Next greater for 7 is -1

Input Format

Input is managed for you

Output Format

Output is managed for you

Constraints

0 <= n < 10^5
-10^9 <= a[i] <= 10^9

Sample Input

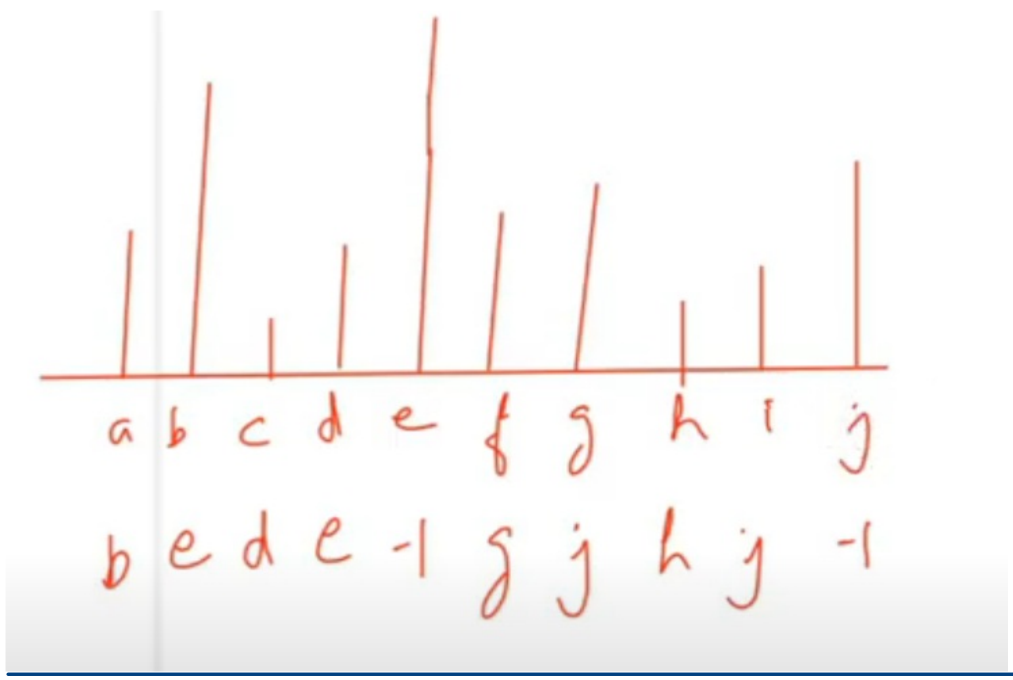
```
5
5
3
8
-2
7
```

Sample Output

```
8
8
-1
7
-1
```

Need to do in O(n)

Next Greater Element To The Right



Note the meaning:

- pop
a answer print karayega
+ push

Note
for next greater element to right, always start form right

Similarly
for next greater element to left, always start form left
for next smaller element to right, always start form right
for next smaller element to left, always start form left

Approach:

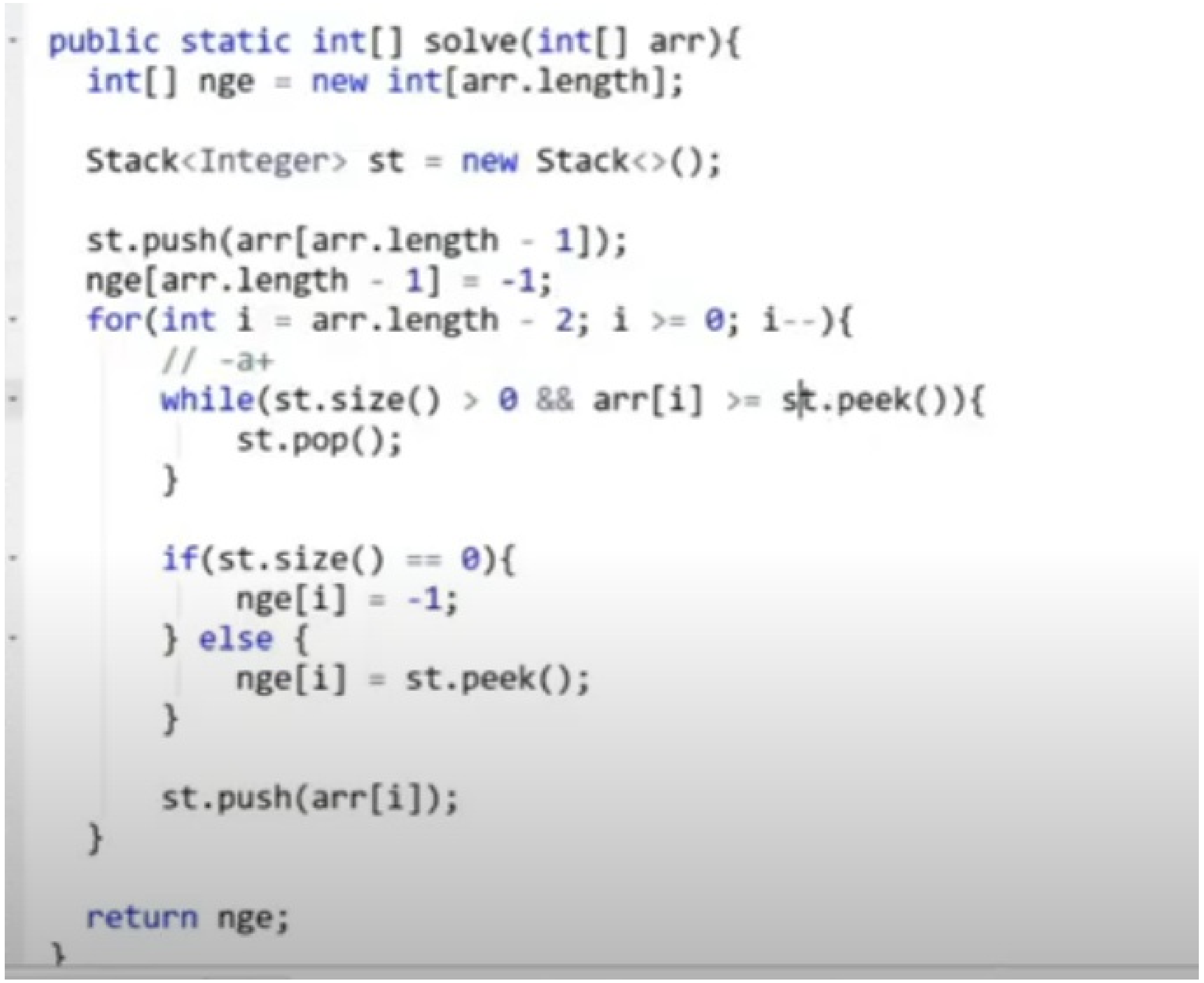
for 1st element there is no greater to right, so ans =-1; ans push element.

consider like this, you will start form right
you always push the element from right but before pushing you need to do some things

This things are:
1. pop the elements if there are small element at peek of stack
.(greater element chaye to tu small element ko pop kara rha he)

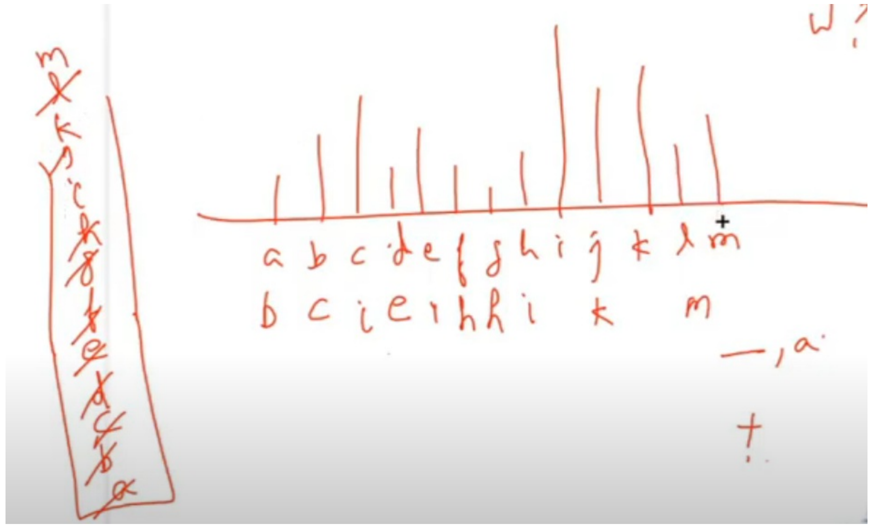
```
    here there are 2 possiblity i.e either you are going to encounter **empty stack** or **greater element**
        if(stack empty) then
            nge[i] = -1  //no greater element
        else
            nge[i]=st.peek()  //greater element
```

2. push current element.



```java
public static int[] solve(int[] arr){
  int[] nge = new int[arr.length];

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

  st.push(arr[arr.length - 1]);
  nge[arr.length - 1] = -1;
  for(int i = arr.length - 2; i >= 0; i--){
      // -a+
      while(st.size() > 0 && arr[i] >= st.peek()){
          st.pop();
      }

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

      st.push(arr[i]);
  }

  return nge;
}
```

Approach: Solving from left



jo bhi element ayega vo choto ko pop karayga
aur jis jis ko pop karaya unka ans khud ban jayega
aur jate jate khud ko push kara dega.

aur last me kuch elements bach jaynege usme hum -1
dal denge

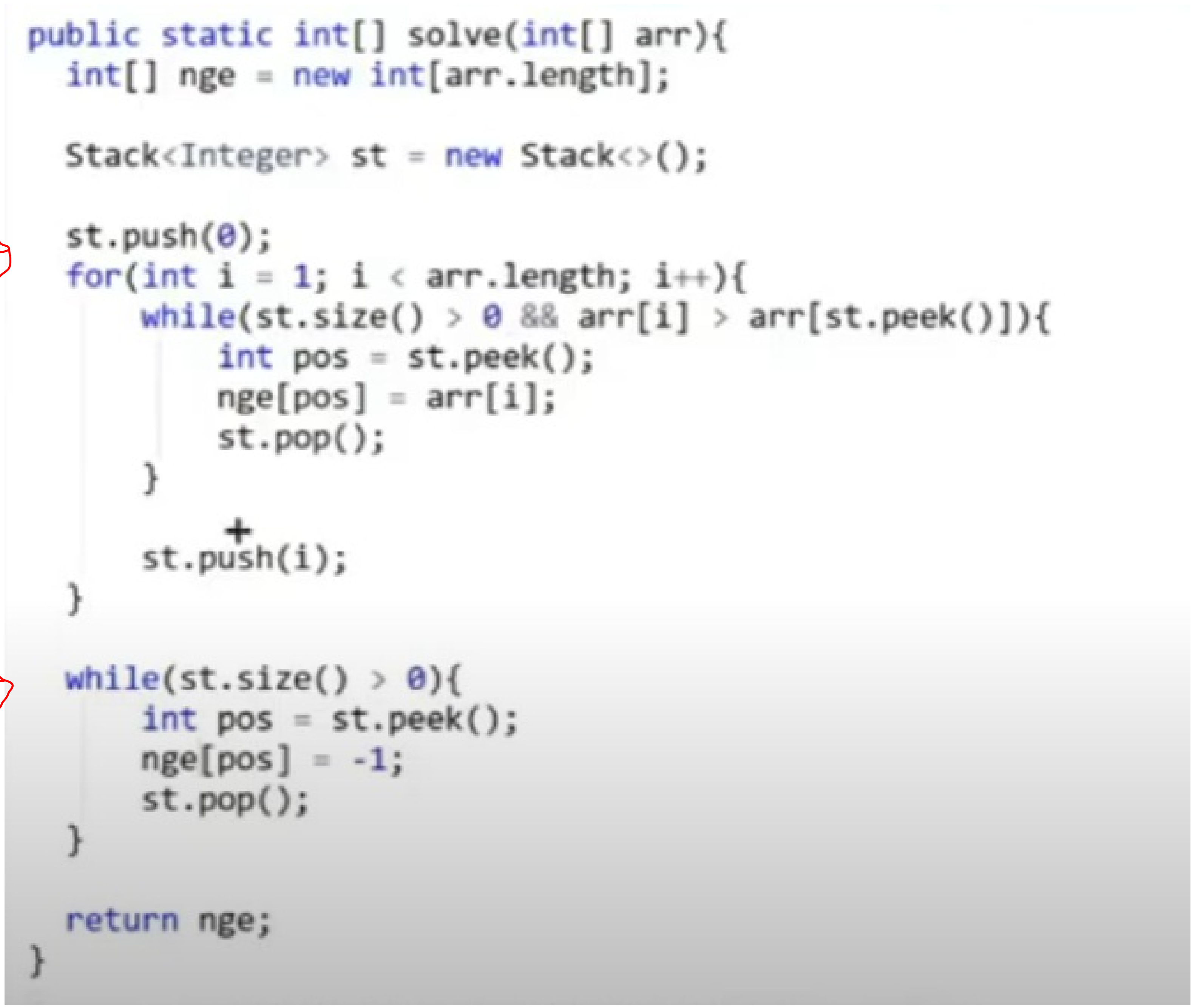
here m,k,i has no greater element to right ,
and are present in stack
so put -1;

Lets say 1st element is in stack i.e 'a'

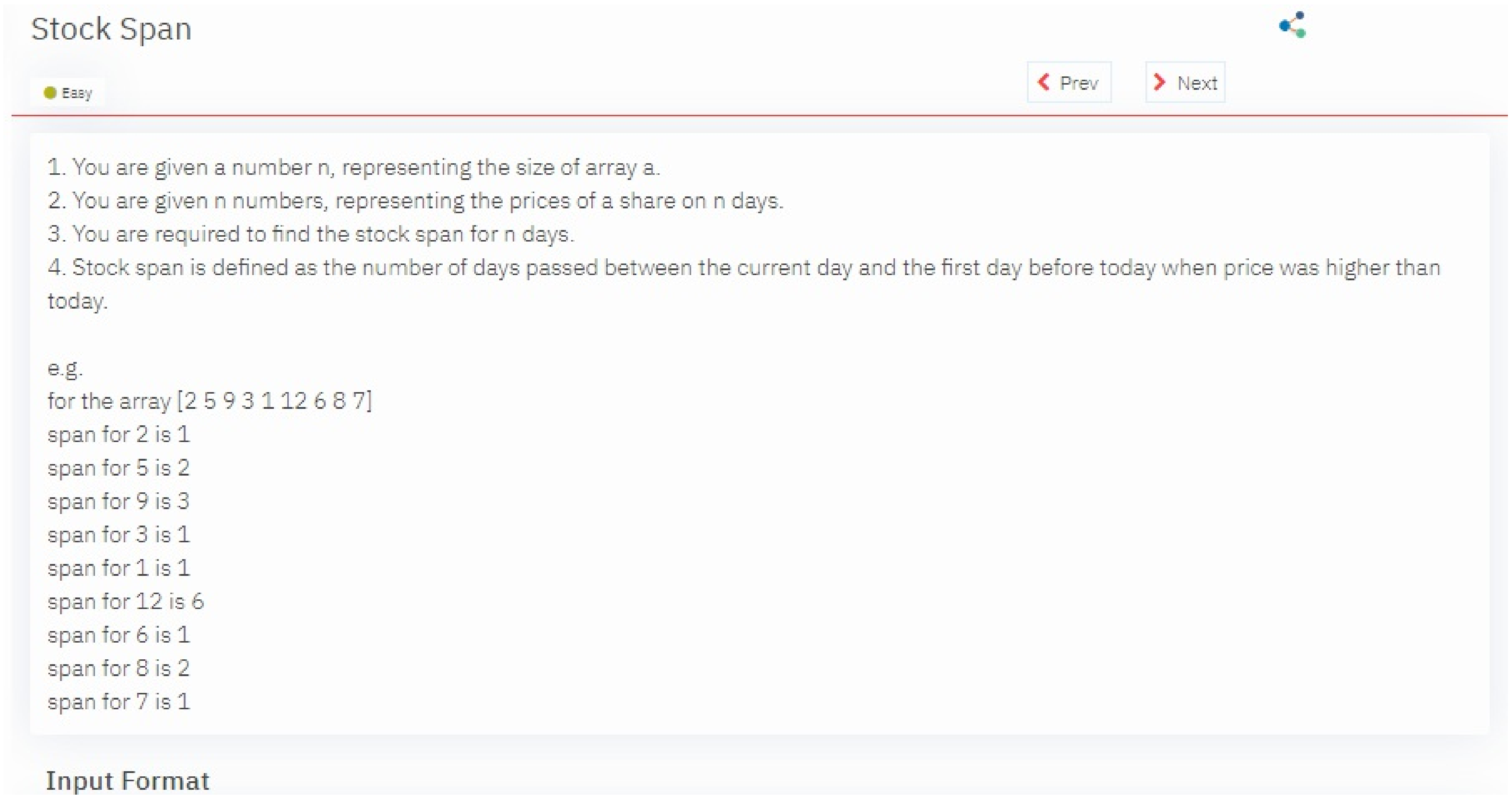
so abhi 'b' aya ,
usne 'a' ko pop kia , bec 'a' chota he from 'b'
aur 'a' ka ans vo khud 'b' ban gaya
aur vo khud 'b' push ho gaya.

index is
pushed

last me kuch
element bach
gaye



```java
public static int[] solve(int[] arr){
  int[] nge = new int[arr.length];

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

  st.push(0);
  for(int i = 1; i < arr.length; i++){
      while(st.size() > 0 && arr[i] > arr[st.peek()]){
          int pos = st.peek();
          nge[pos] = arr[i];
          st.pop();
      }

      st.push(i);
  }

  while(st.size() > 0){
      int pos = st.peek();
      nge[pos] = -1;
      st.pop();
  }

  return nge;
}
```

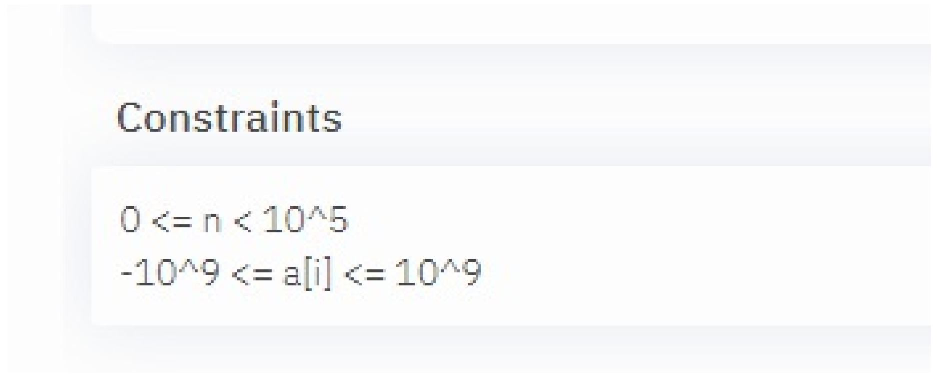


Stock Span

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing the prices of a share on n days.
3. You are required to find the stock span for n days.
4. Stock span is defined as the number of days passed between the current day and the first day before today when price was higher than today.

e.g.
for the array [2 5 9 3 1 12 6 8 7]
span for 2 is 1
span for 5 is 2
span for 9 is 3
span for 3 is 1
span for 1 is 1
span for 12 is 6
span for 6 is 1
span for 8 is 2
span for 7 is 1

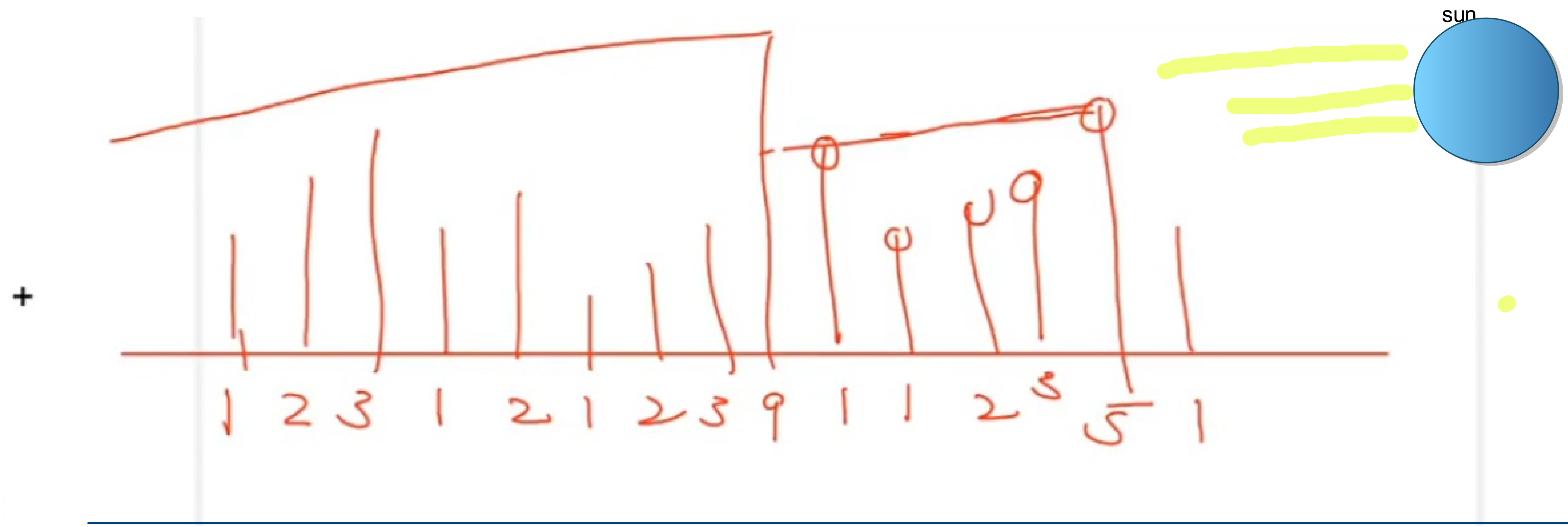
Input Format



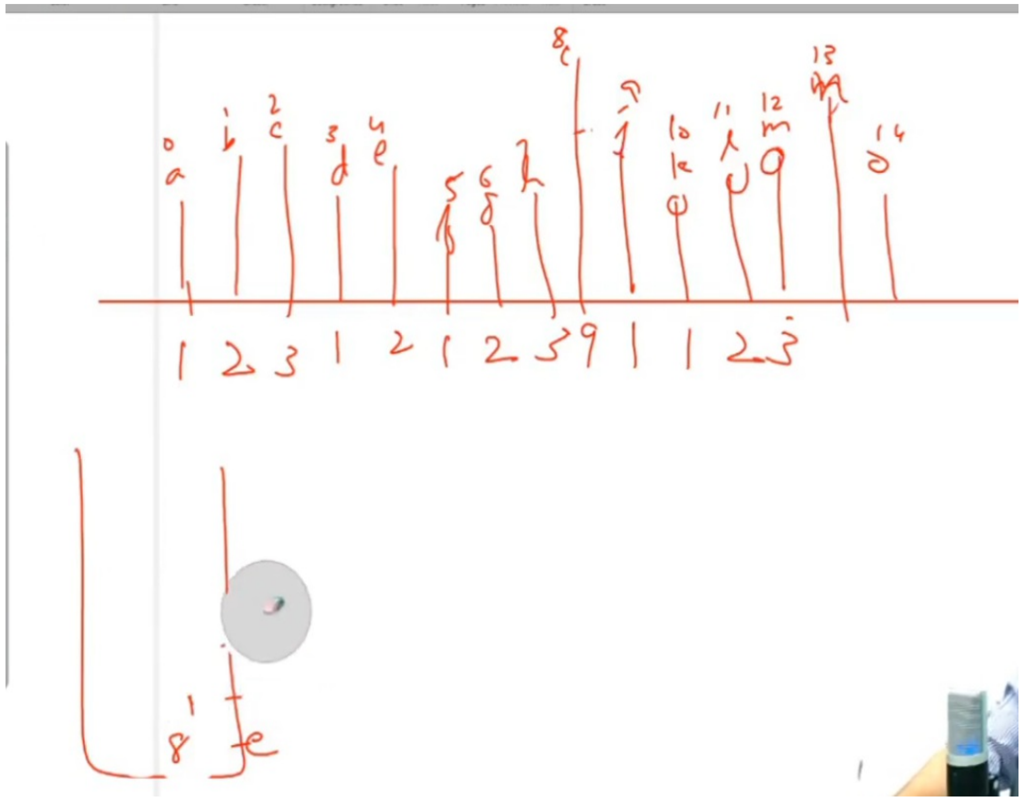
Constraints

0 <= n < 10^5
-10^9 <= a[i] <= 10^9

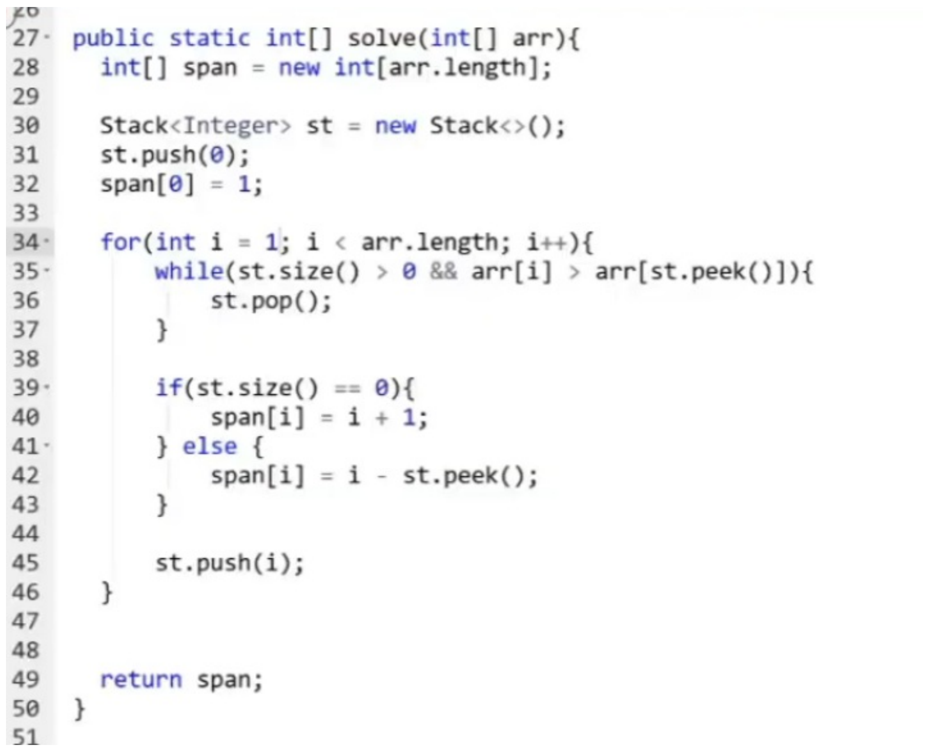
consider there is sun, and
jo greater he uske shadow me jitne log he vo span he, including greater element itself



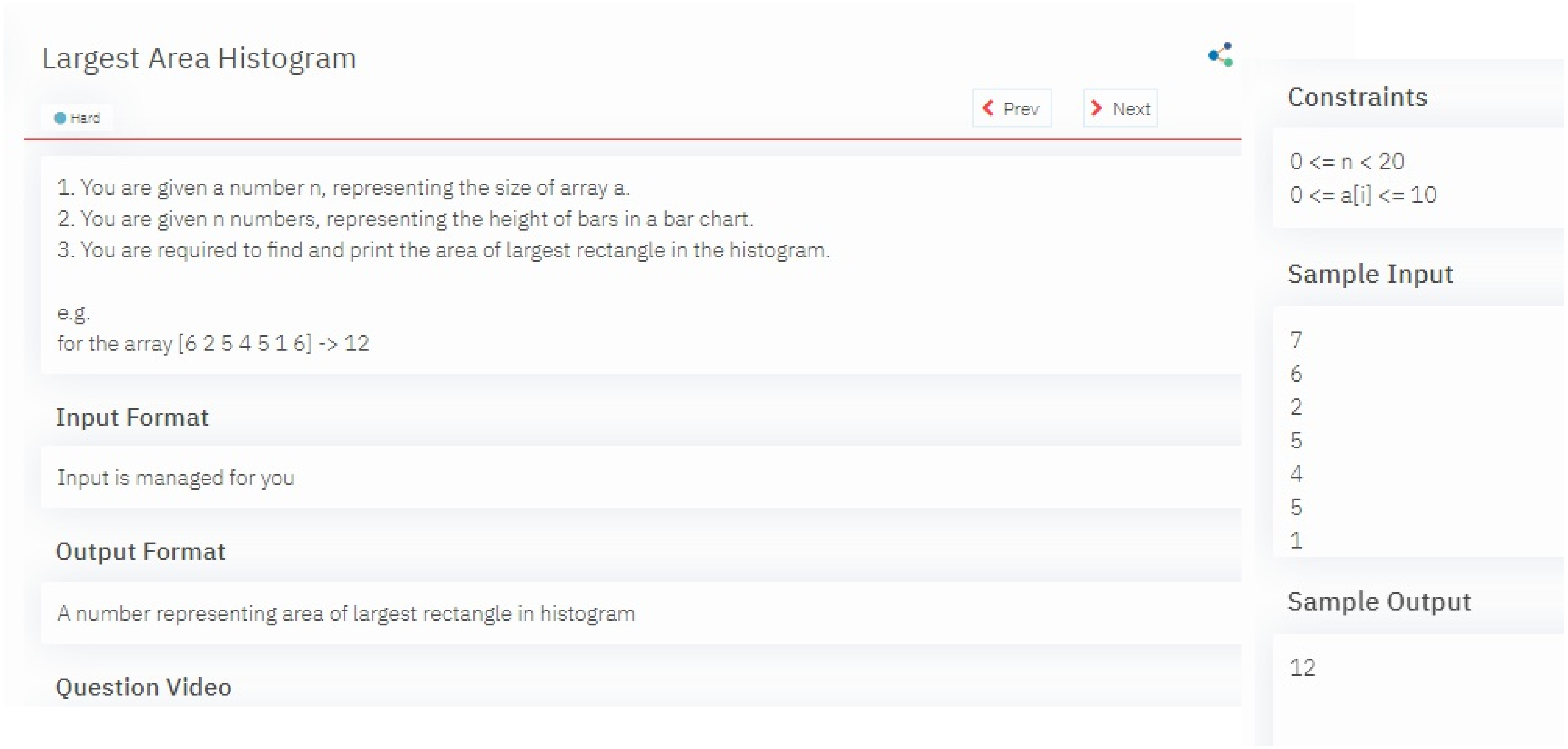
consider index 8 is in process, it will pop all elements withing its span ,
And all the elements are in its span ,so stack will be empty
and its ans = currentindex+1 i.e 8+1=9,
push the current index (8)



for index 4, it wil pop element 3 from its span
and current(index) -st.peek() =4- 2 =2
and push currnet index9;()khud push ho jayega)



```java
public static int[] solve(int[] arr){
  int[] span = new int[arr.length];

  Stack<Integer> st = new Stack<>();
  st.push(0);
  span[0] = 1;

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

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

      st.push(i);
  }


  return span;
}
```



Largest Area Histogram

Hard

Prev Next

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing the height of bars in a bar chart.
3. You are required to find and print the area of largest rectangle in the histogram.

e.g.
for the array [6 2 5 4 5 1 6] -> 12

Input Format

Input is managed for you

Output Format

A number representing area of largest rectangle in histogram

Question Video

Constraints

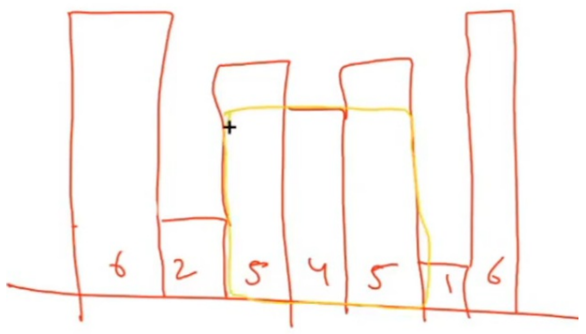
0 <= n < 20
0 <= a[i] <= 10

Sample Input

```
7
6
2
5
4
5
1
```

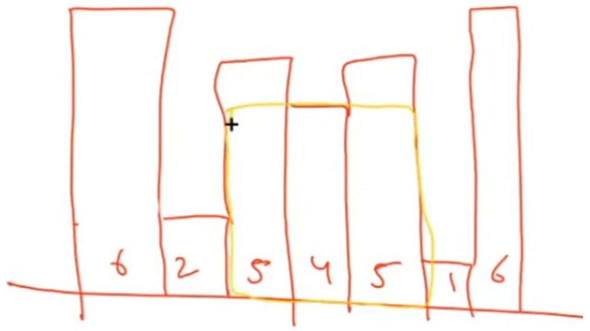
Sample Output

```
12
```



if you height 4, then
you will get area
=4x3=12

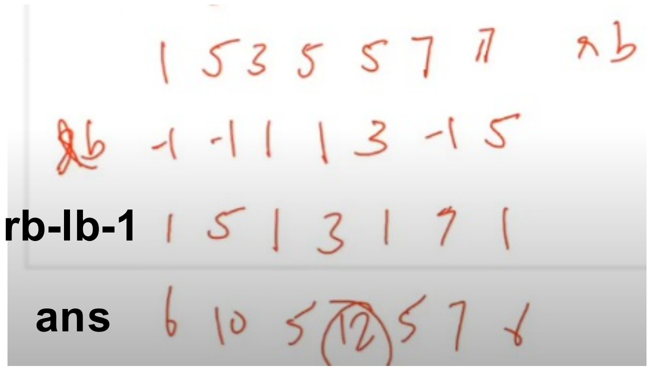
need to do in O(n)



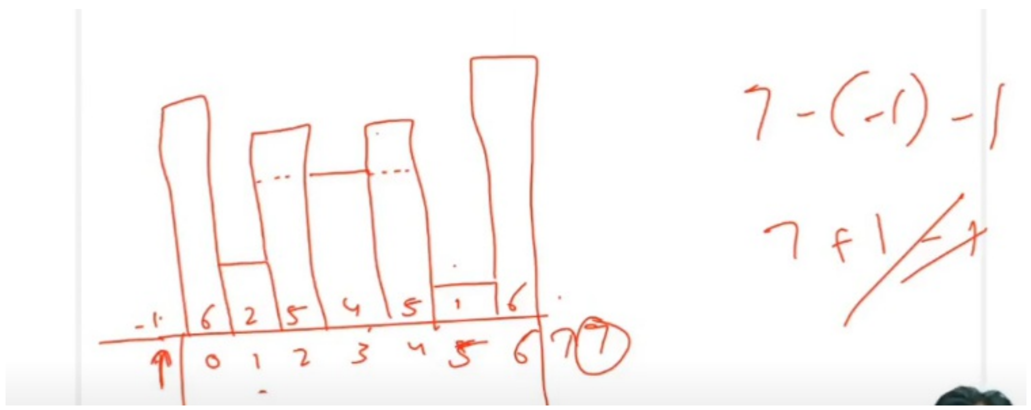
Points to notice:
next smaller element on right is its right boundary
next smaller element on left is its left boundary

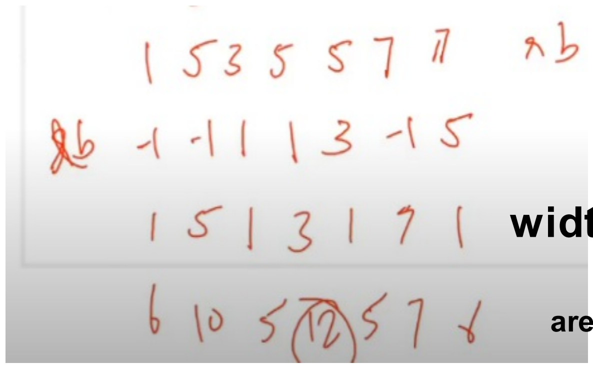
---

for **index3**,
we have **height** =4;
we need to find **width =** index of smaller lement on right - index of smaller element on left -1
width = (5-1) -1 = 3



	1	5	3	5	5	7	7	rb
lb	-1	-1	1	1	3	-1	5	
width rb-lb-1	1	5	1	3	1	7	1	
ans	6	10	5	12	5	7	6	



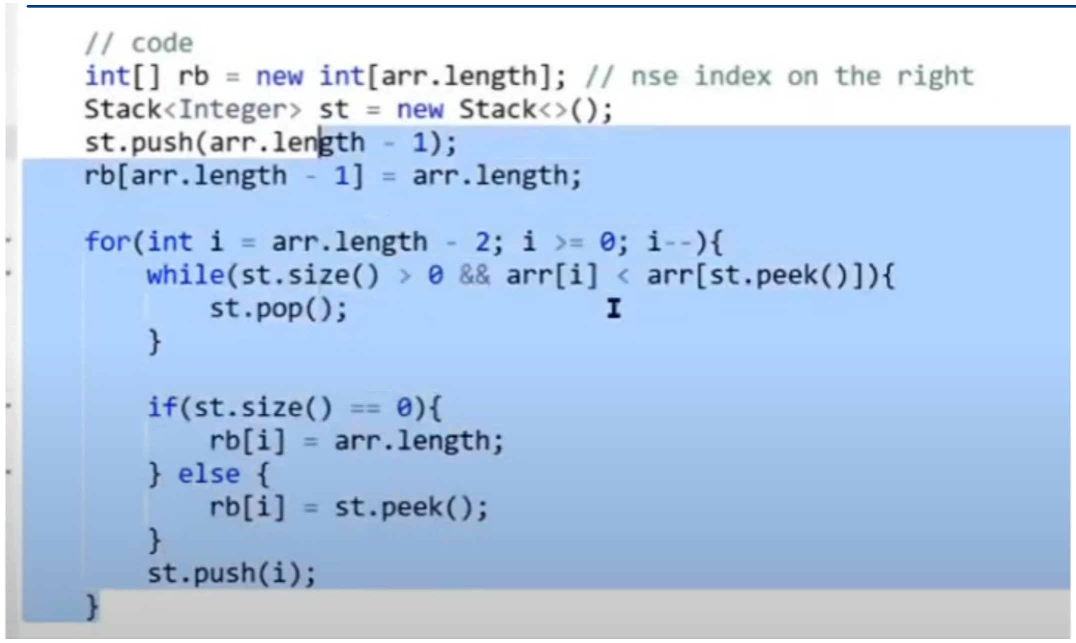


width=rb-lb-1

area=width *height

Next smaller element index on right

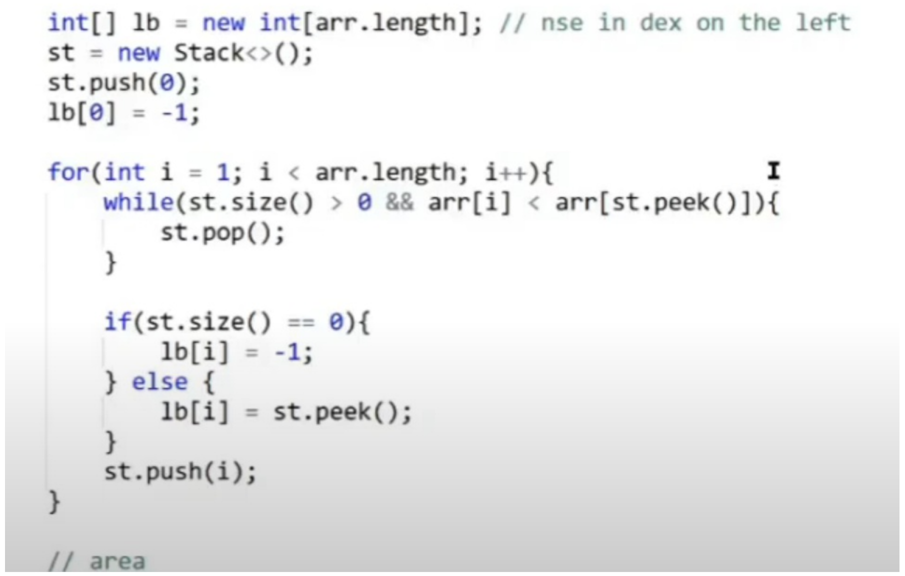
Notice index is getting pushed



```
// code
int[] rb = new int[arr.length]; // nse index on the right
Stack<Integer> st = new Stack<>();
st.push(arr.length - 1);
rb[arr.length - 1] = arr.length;

for(int i = arr.length - 2; i >= 0; i--){
    while(st.size() > 0 && arr[i] < arr[st.peek()]){
        st.pop();
    }

    if(st.size() == 0){
        rb[i] = arr.length;
    } else {
        rb[i] = st.peek();
    }
    st.push(i);
}
```

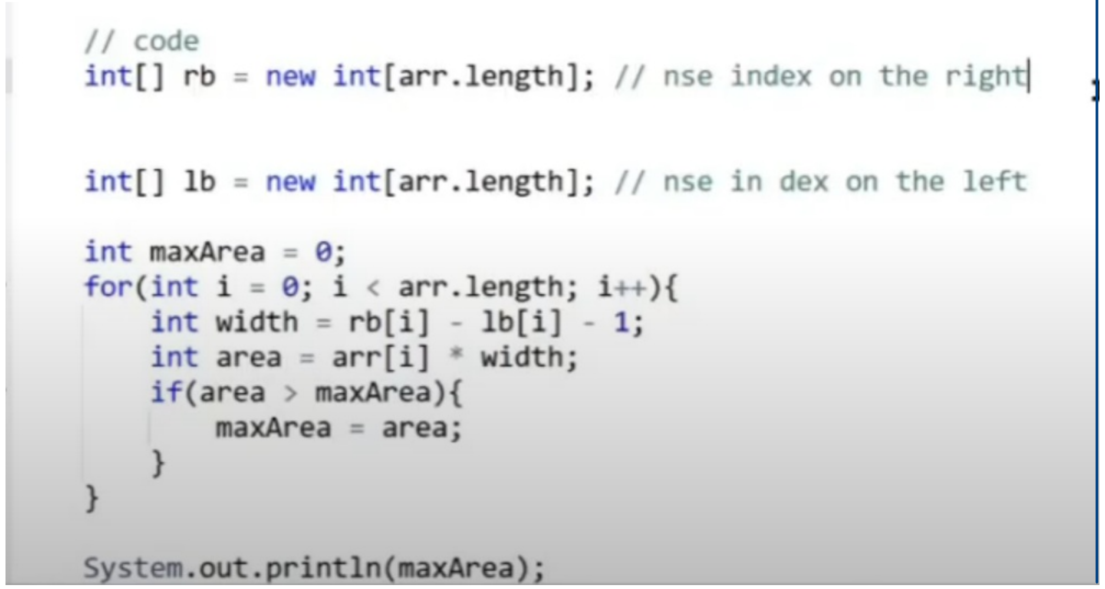
Next smaller element index on left

Notice index is getting pushed

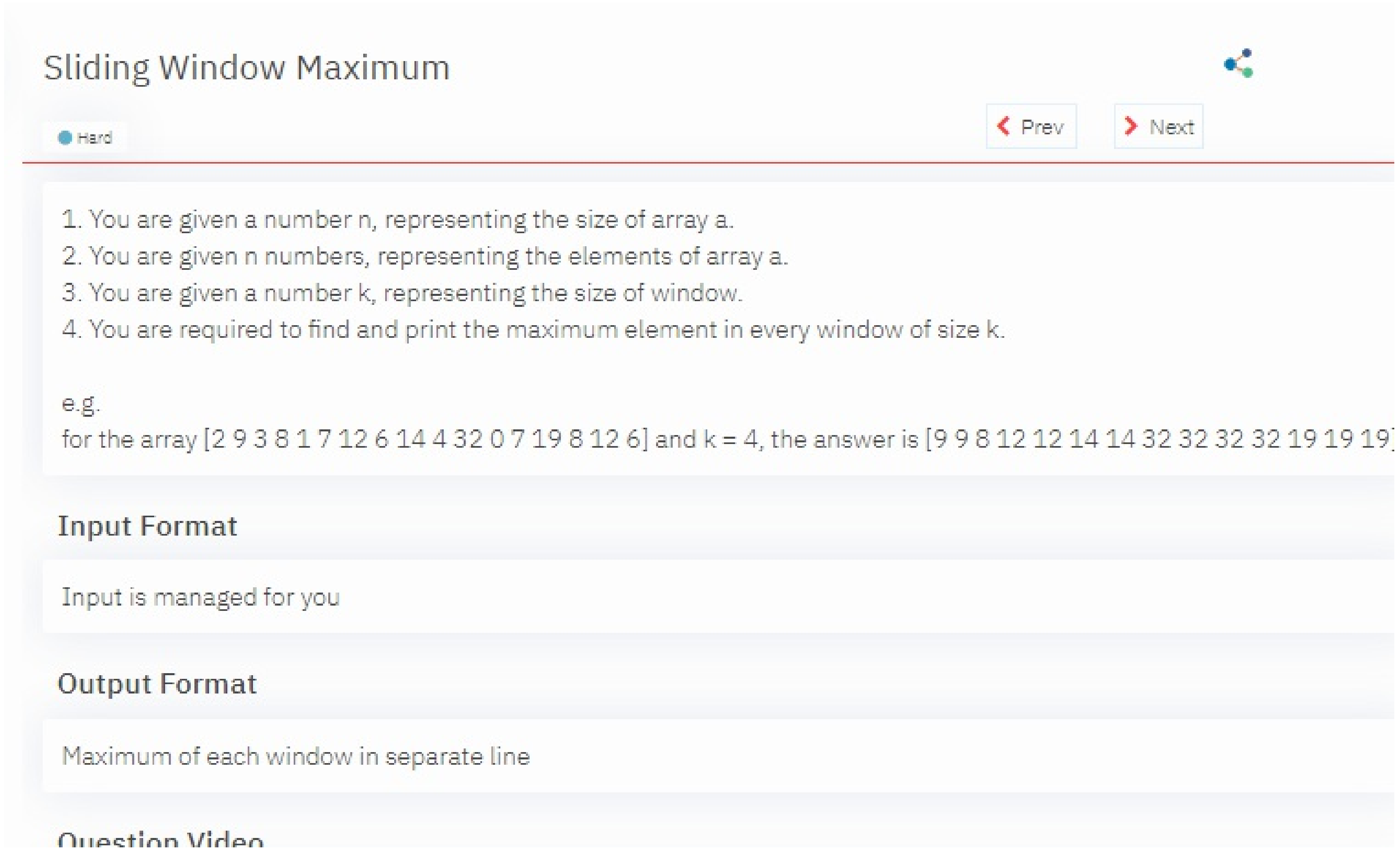


```
int[] lb = new int[arr.length]; // nse in dex on the left
st = new Stack<>();
st.push(0);
lb[0] = -1;

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

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

// area
```

to get max area



```
// code
int[] rb = new int[arr.length]; // nse index on the right


int[] lb = new int[arr.length]; // nse in dex on the left

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

System.out.println(maxArea);
```



Sliding Window Maximum

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing the elements of array a.
3. You are given a number k, representing the size of window.
4. You are required to find and print the maximum element in every window of size k.

e.g.
for the array [2 9 3 8 1 7 12 6 14 4 32 0 7 19 8 12 6] and k = 4, the answer is [9 9 8 12 12 14 14 32 32 32 32 19 19 19]

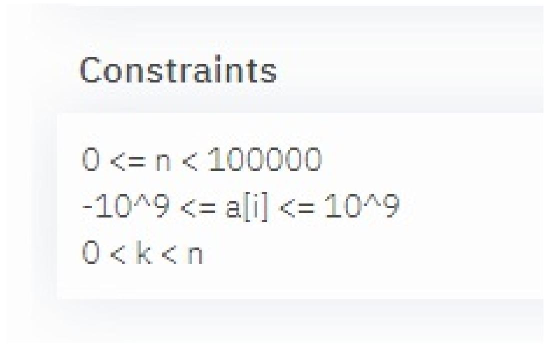
Input Format

Input is managed for you

Output Format

Maximum of each window in separate line

Question Video

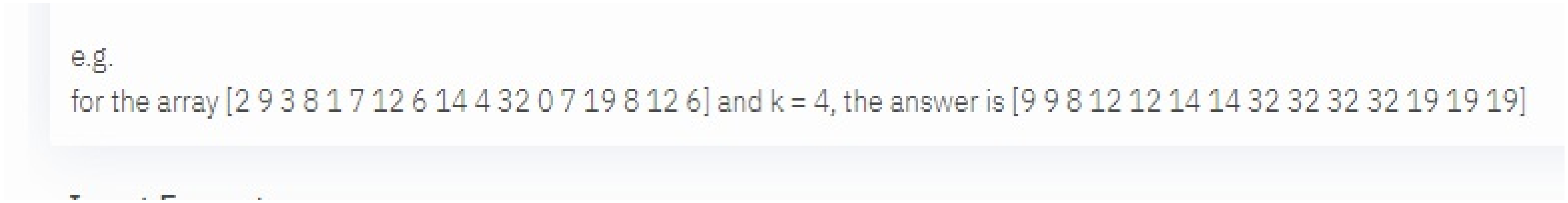


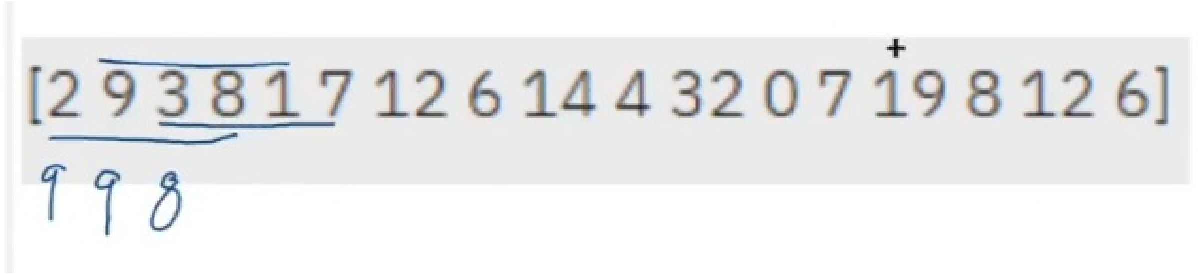
Constraints

0 <= n < 100000
-10^9 <= a[i] <= 10^9
0 < k < n

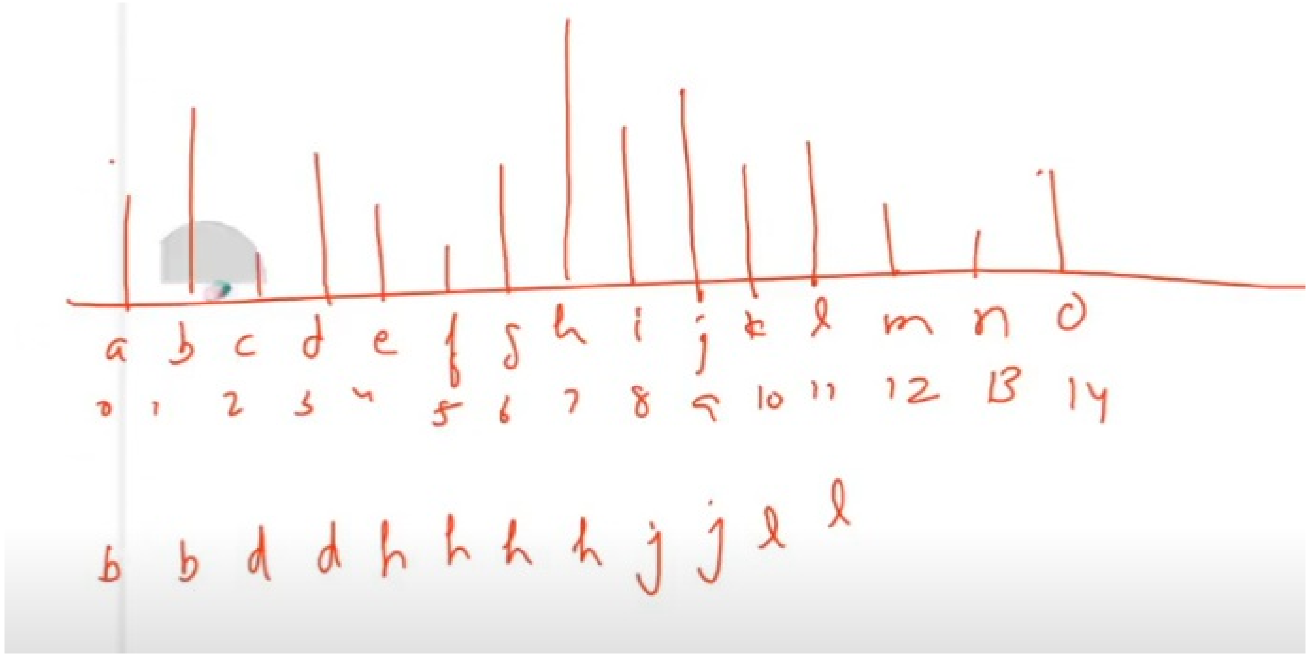
Need to do in O(n)

you can also do in O(Log n) ---> check online

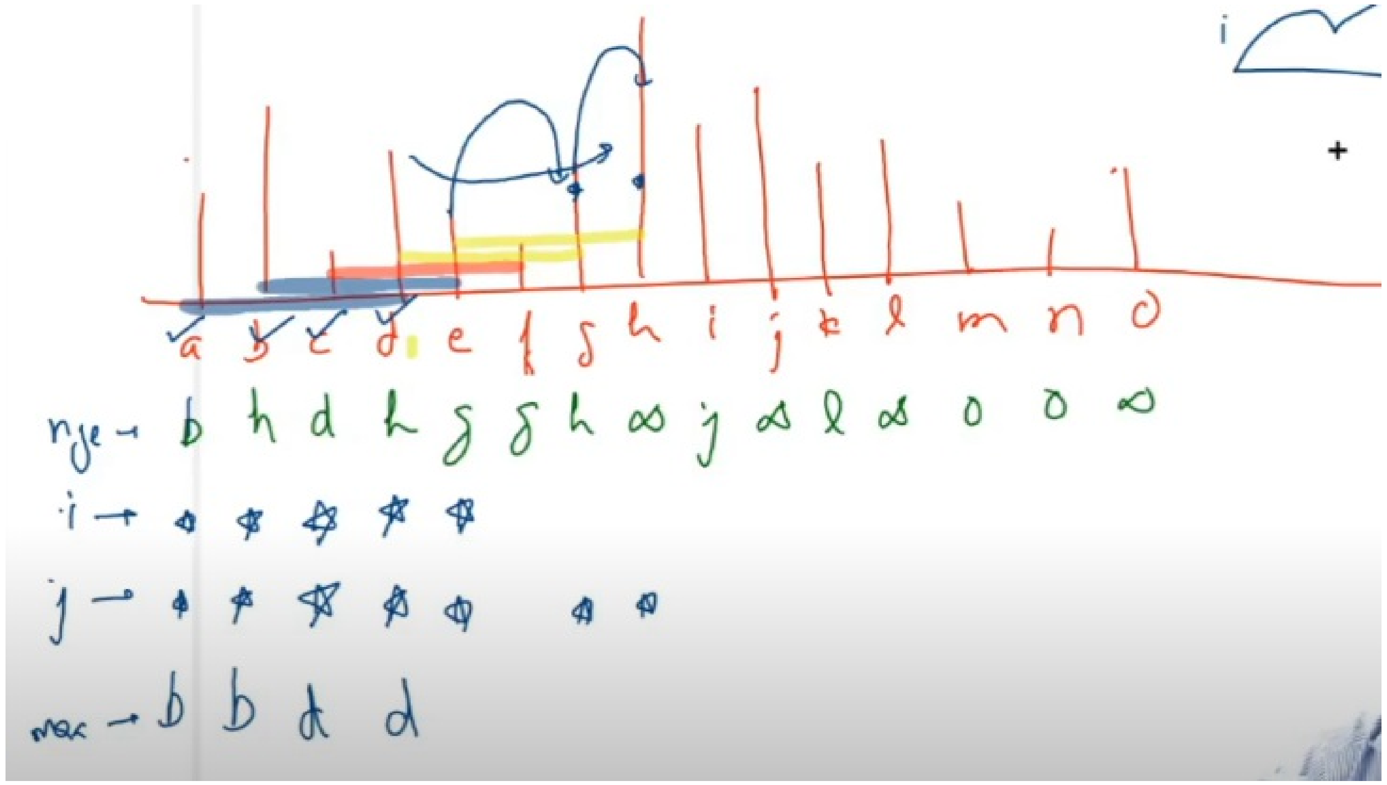

e.g.
for the array [2 9 3 8 1 7 12 6 14 4 32 0 7 19 8 12 6] and k = 4, the answer is [9 9 8 12 12 14 14 32 32 32 32 19 19 19]


[2 9 3 8 1 7 12 6 14 4 32 0 7 19 8 12 6]

K is window size



Approach is dependent on next greater element



i
a b c d e f g h i j k l m n o
nge → b h d h g g h ∞ j ∞ l ∞ o o ∞
i →
j →
max → b b d d

Note
j jumps from next greater to next greater

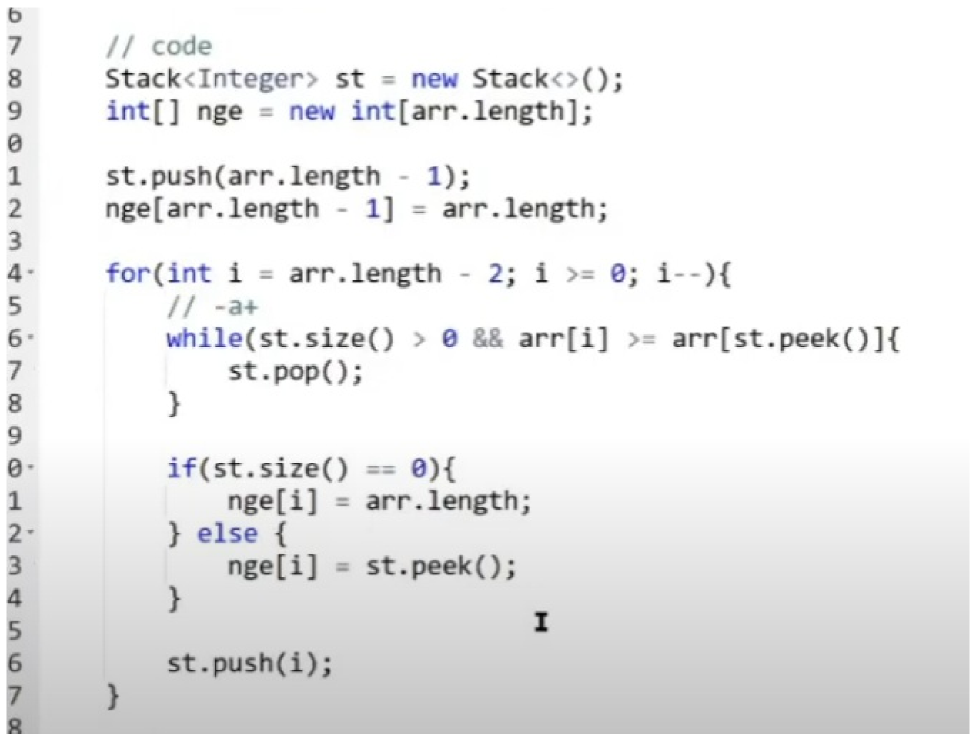
Approach is
for element 'a' and window(a,b,c,d)
nge for element 'a' ,--> b (**inside the window**)
j will jump to b
nge for element b = h (outside the window)
so it will print b **(current position of j)**

for element 'b' and window(b,c,d,e)
nge for element b = h (outside the window)
so it will print current element

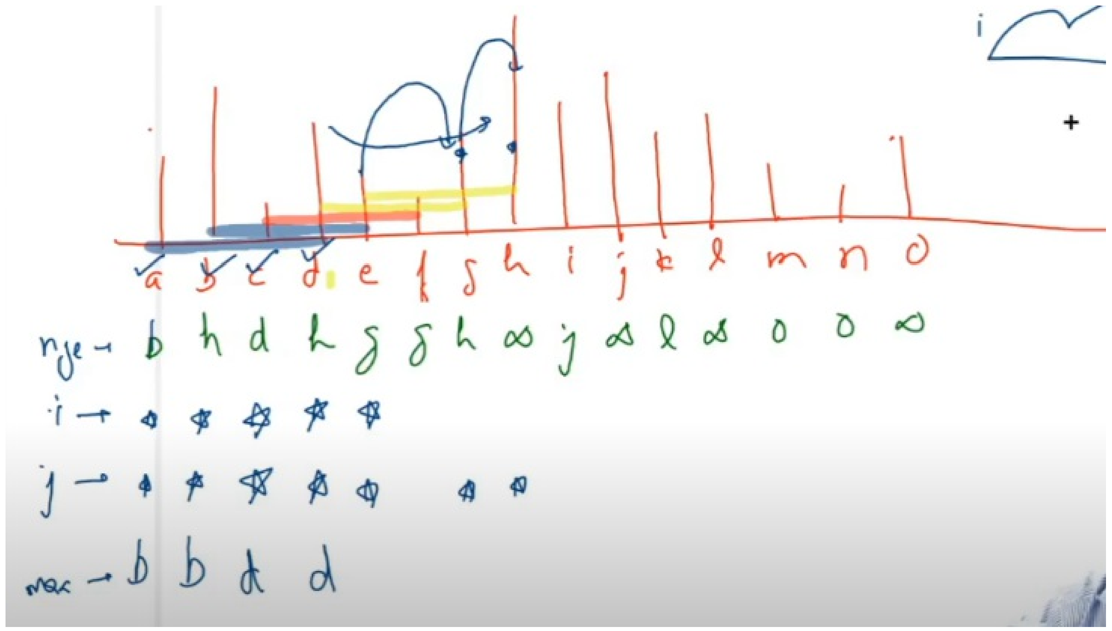
Approach is
for element 'e' and window(e,f,g,h)
nge for element 'e' ,--> g (inside the window)
j will jump to g

nge for element 'g'----> h(**inside the window**)
nge for element 'h' ---> infinity (outside the window)
so ans is element h (current position of j)

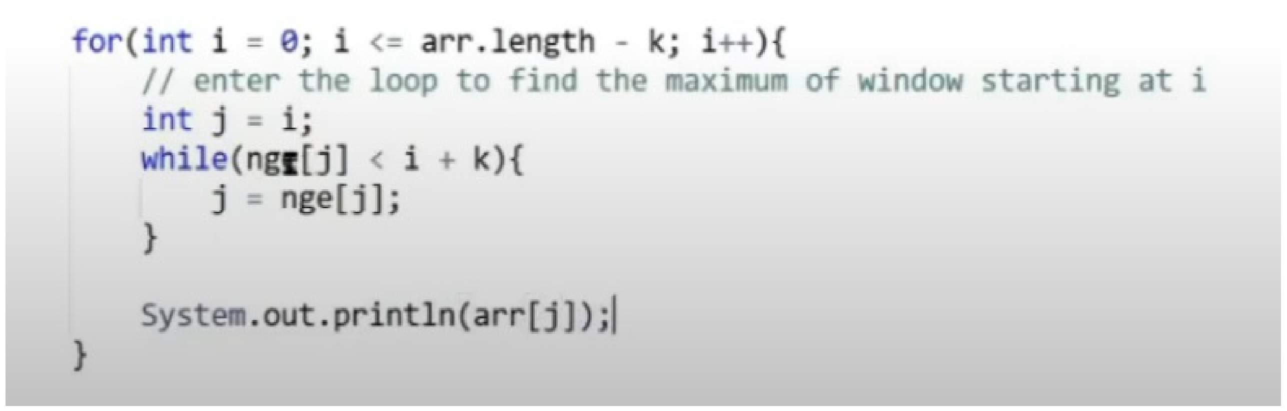
nge - Next greater element



```
    // code
    Stack<Integer> st = new Stack<>();
    int[] nge = new int[arr.length];

    st.push(arr.length - 1);
    nge[arr.length - 1] = arr.length;

    for(int i = arr.length - 2; i >= 0; i--){
        // -a+
        while(st.size() > 0 && arr[i] >= arr[st.peek()]){
            st.pop();
        }

        if(st.size() == 0){
            nge[i] = arr.length;
        } else {
            nge[i] = st.peek();
        }

        st.push(i);
    }
```

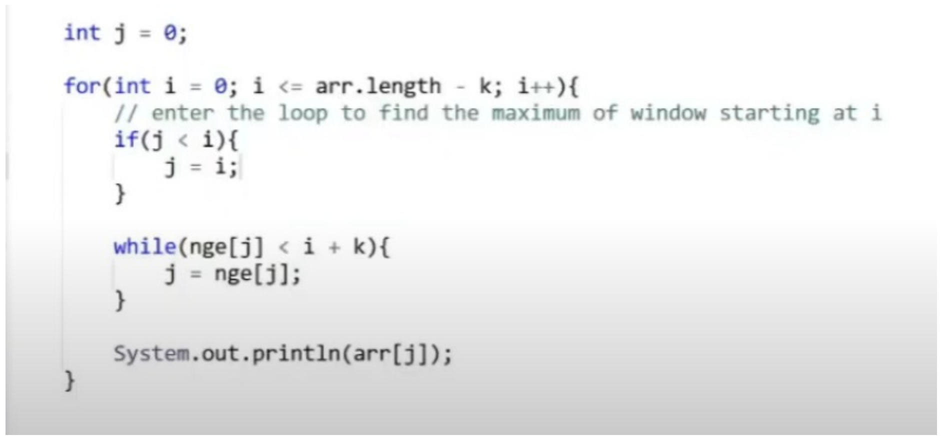

a b c d e f g h i j k l m n o
nge → b h d h g g h ∞ j ∞ l ∞ o o ∞
i →
j →
max → b b d d

To get the max element in window



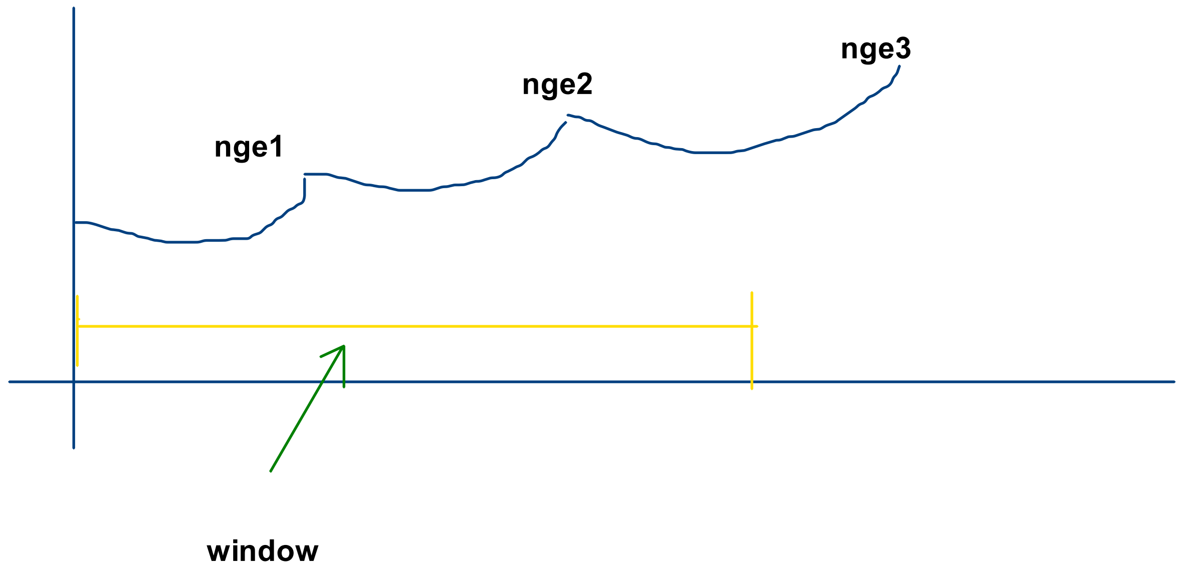
```
for(int i = 0; i <= arr.length - k; i++){
    // enter the loop to find the maximum of window starting at i
    int j = i;
    while(nge[j] < i + k){
        j = nge[j];
    }

    System.out.println(arr[j]);
}
```

performance iimproved for j



```
int j = 0;

for(int i = 0; i <= arr.length - k; i++){
    // enter the loop to find the maximum of window starting at i
    if(j < i){
        j = i;
    }

    while(nge[j] < i + k){
        j = nge[j];
    }

    System.out.println(arr[j]);
}
```

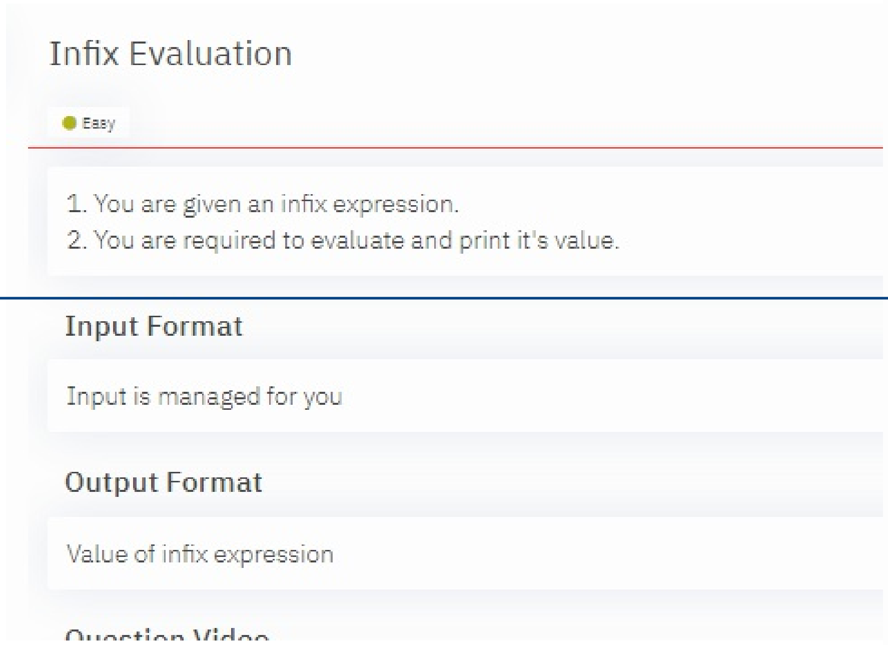
nge - next greater element



Our approach is

nge 3 is out of window
so our ans is nge2

j jumps form nge to nge



Infix Evaluation

Easy

1. You are given an infix expression.
2. You are required to evaluate and print it's value.

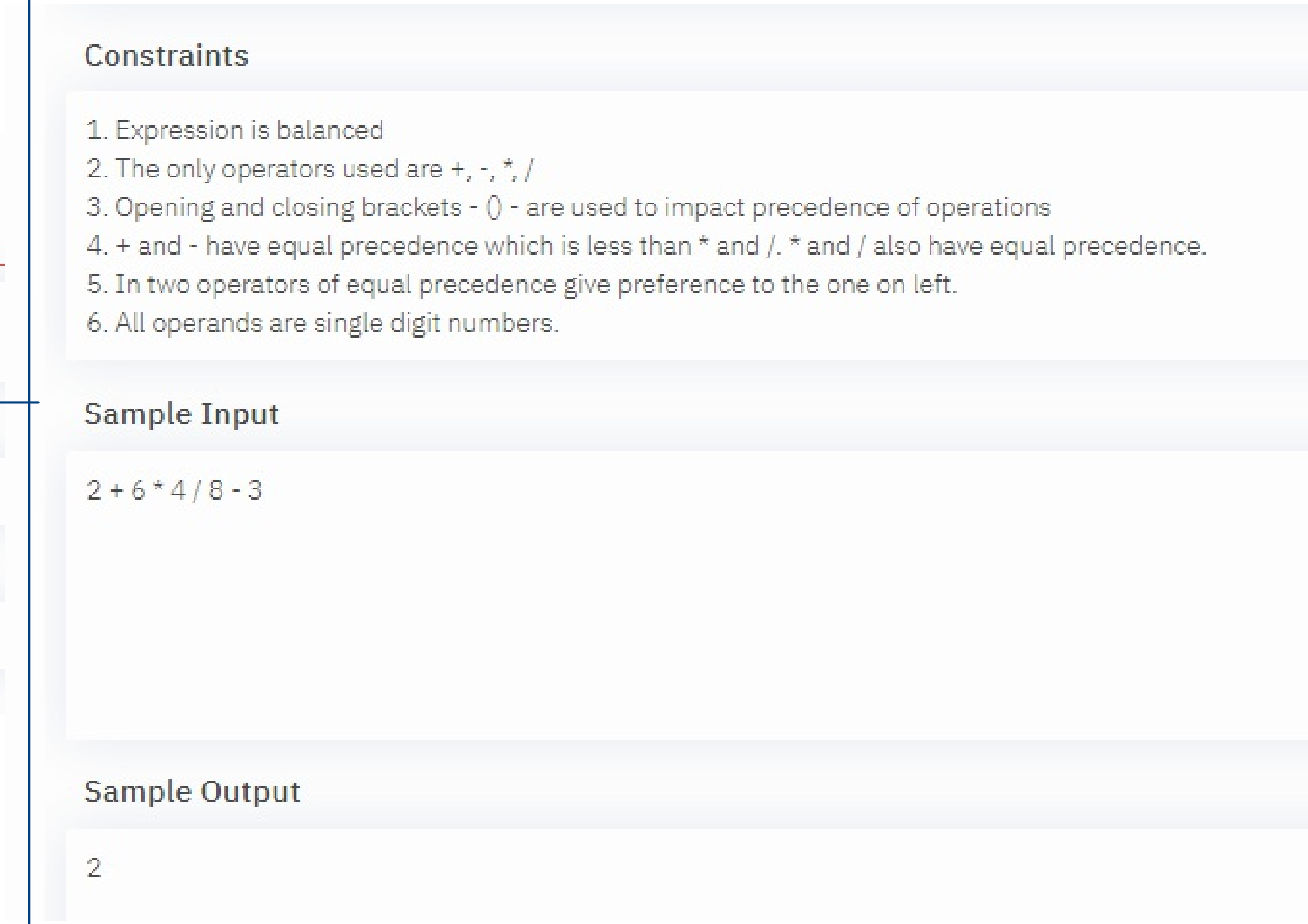
Input Format

Input is managed for you

Output Format

Value of infix expression

Question Video



Constraints

1. Expression is balanced
2. The only operators used are +, -, *, /
3. Opening and closing brackets - () - are used to impact precedence of operations
4. + and - have equal precedence which is less than * and /. * and / also have equal precedence.
5. In two operators of equal precedence give preference to the one on left.
6. All operands are single digit numbers.

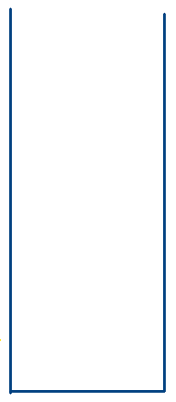
Sample Input

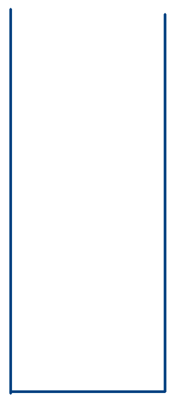
```
2 + 6 * 4 / 8 - 3
```

Sample Output

```
2
```

$2 + (5 - 3 * 6 / 2)$


operand stack


operator stack

Rules to process

1. operand ==> push to operand stack
2. '(' ==> push to operator stack
3. ')' ==> pop till '(' and solve the operator and operand and push result in operand stack
and pop '('

4. operator ==> if operator in stack has precedence >= curr operator ,
then pop (till stack empty or opening bracket or operator in stack has less priority) and solve the operand and push the result in operand stack
and push the current operator

how the operation is performed?

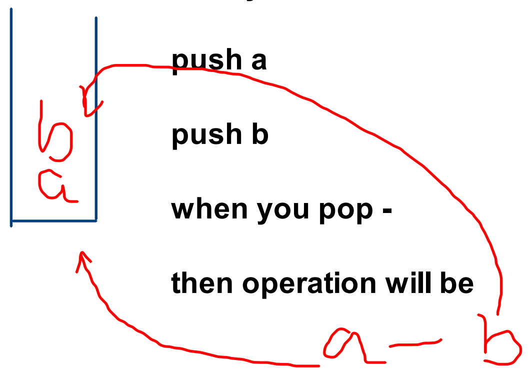
let say we havea-b

push a

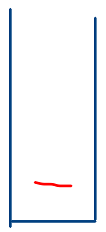
push b

when you pop -

then operation will be



push -



lets say we have a x b - c

push a
push x
push b

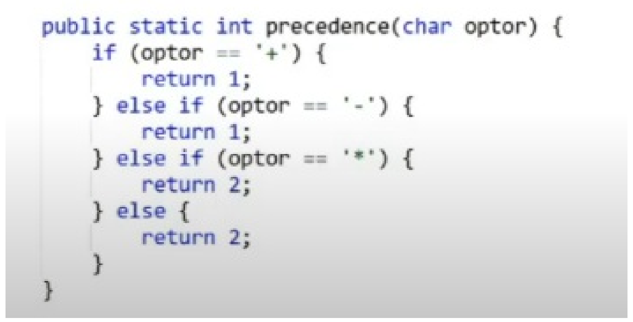
pop x =>(at - , it will check the operator stack,
So top of operator stack(x) have high
precedence than (-)
so pop x and)

push a x b

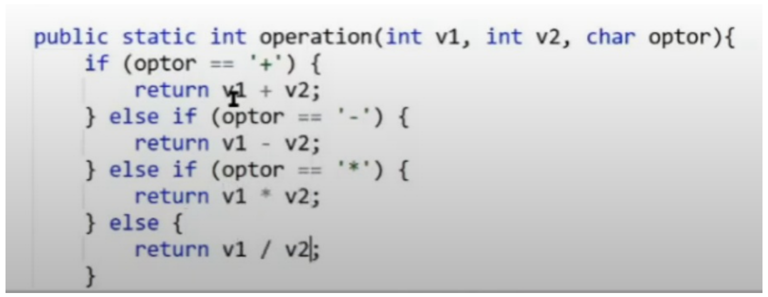
push -

push c

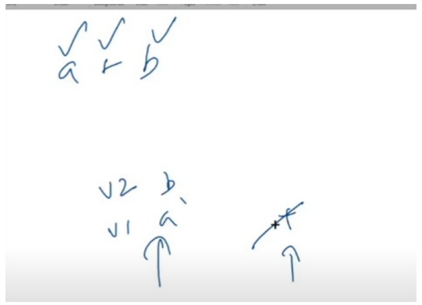
pop - -> here it will a x b -c



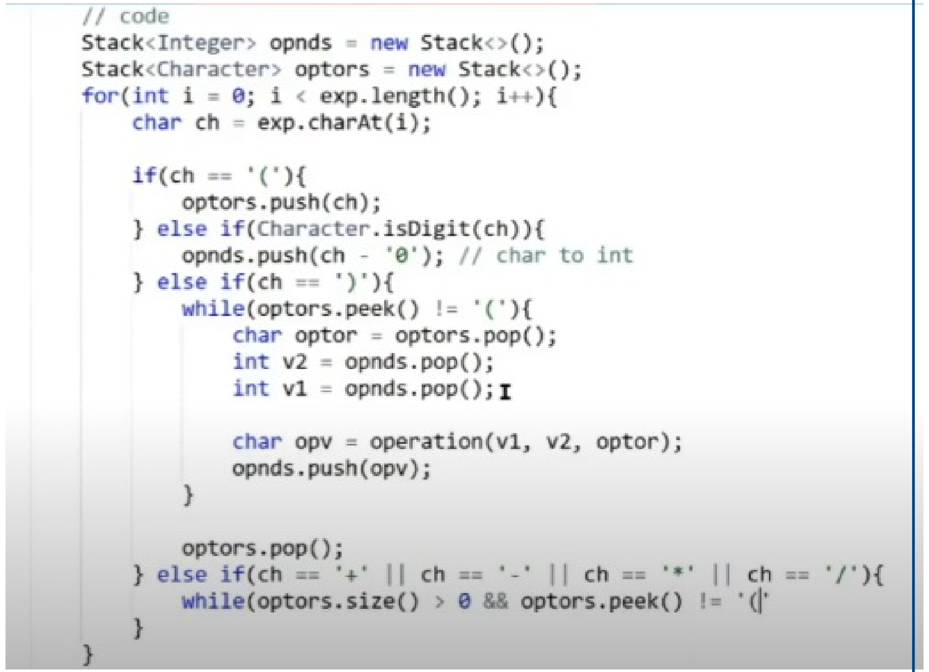
```java
public static int precedence(char optor) {
    if (optor == '+') {
        return 1;
    } else if (optor == '-') {
        return 1;
    } else if (optor == '*') {
        return 2;
    } else {
        return 2;
    }
}
```



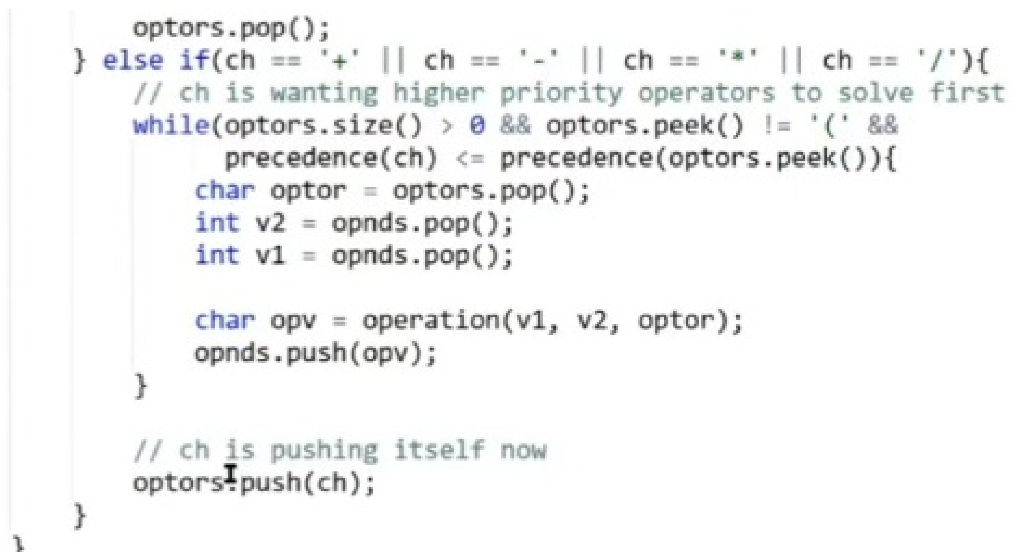
```java
public static int operation(int v1, int v2, char optor){
    if (optor == '+') {
        return v1 + v2;
    } else if (optor == '-') {
        return v1 - v2;
    } else if (optor == '*') {
        return v1 * v2;
    } else {
        return v1 / v2;
    }
```



CODE

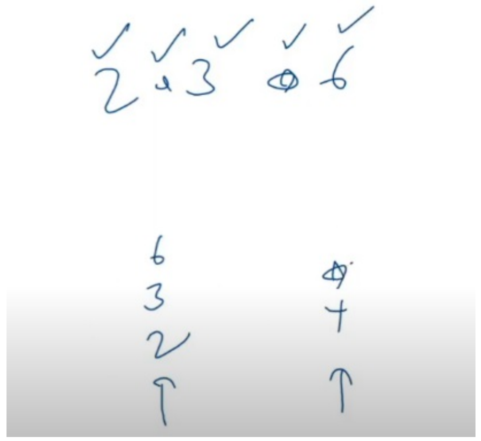


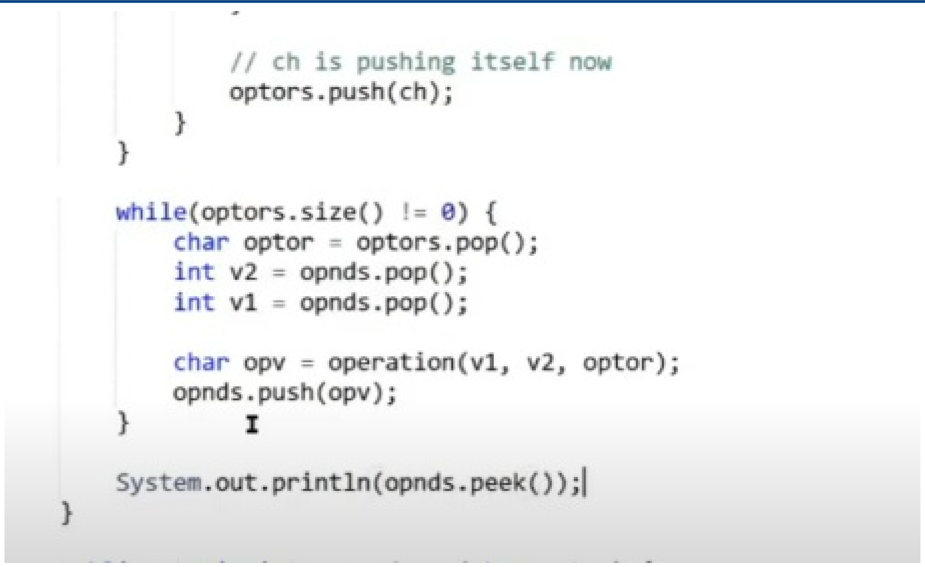
```java
// code
Stack<Integer> opnds = new Stack<>();
Stack<Character> optors = new Stack<>();
for(int i = 0; i < exp.length(); i++){
    char ch = exp.charAt(i);

    if(ch == '('){
        optors.push(ch);
    } else if(Character.isDigit(ch)){
        opnds.push(ch - '0'); // char to int
    } else if(ch == ')'){
        while(optors.peek() != '('){
            char optor = optors.pop();
            int v2 = opnds.pop();
            int v1 = opnds.pop();

            char opv = operation(v1, v2, optor);
            opnds.push(opv);
        }

        optors.pop();
    } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){
        while(optors.size() > 0 && optors.peek() != '('
    }
}
```



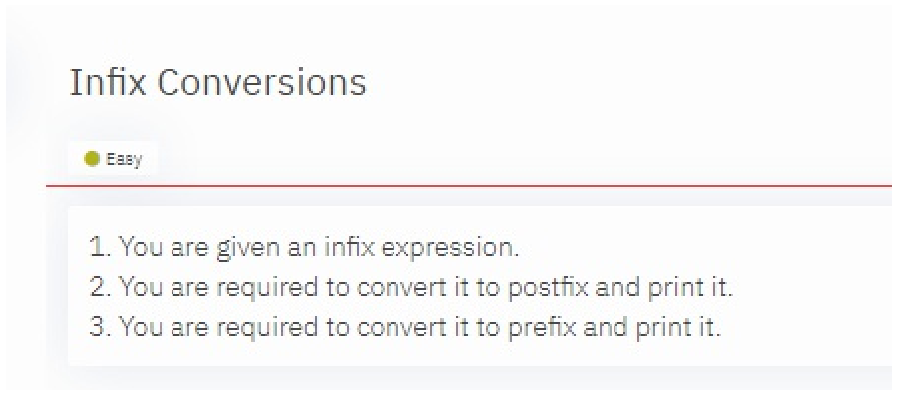
```java
            optors.pop();
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){
            // ch is wanting higher priority operators to solve first
            while(optors.size() > 0 && optors.peek() != '(' &&
                    precedence(ch) <= precedence(optors.peek())){
                char optor = optors.pop();
                int v2 = opnds.pop();
                int v1 = opnds.pop();

                char opv = operation(v1, v2, optor);
                opnds.push(opv);
            }

            // ch is pushing itself now
            optors.push(ch);
        }
}
```

Now to solve the remaining element in stack





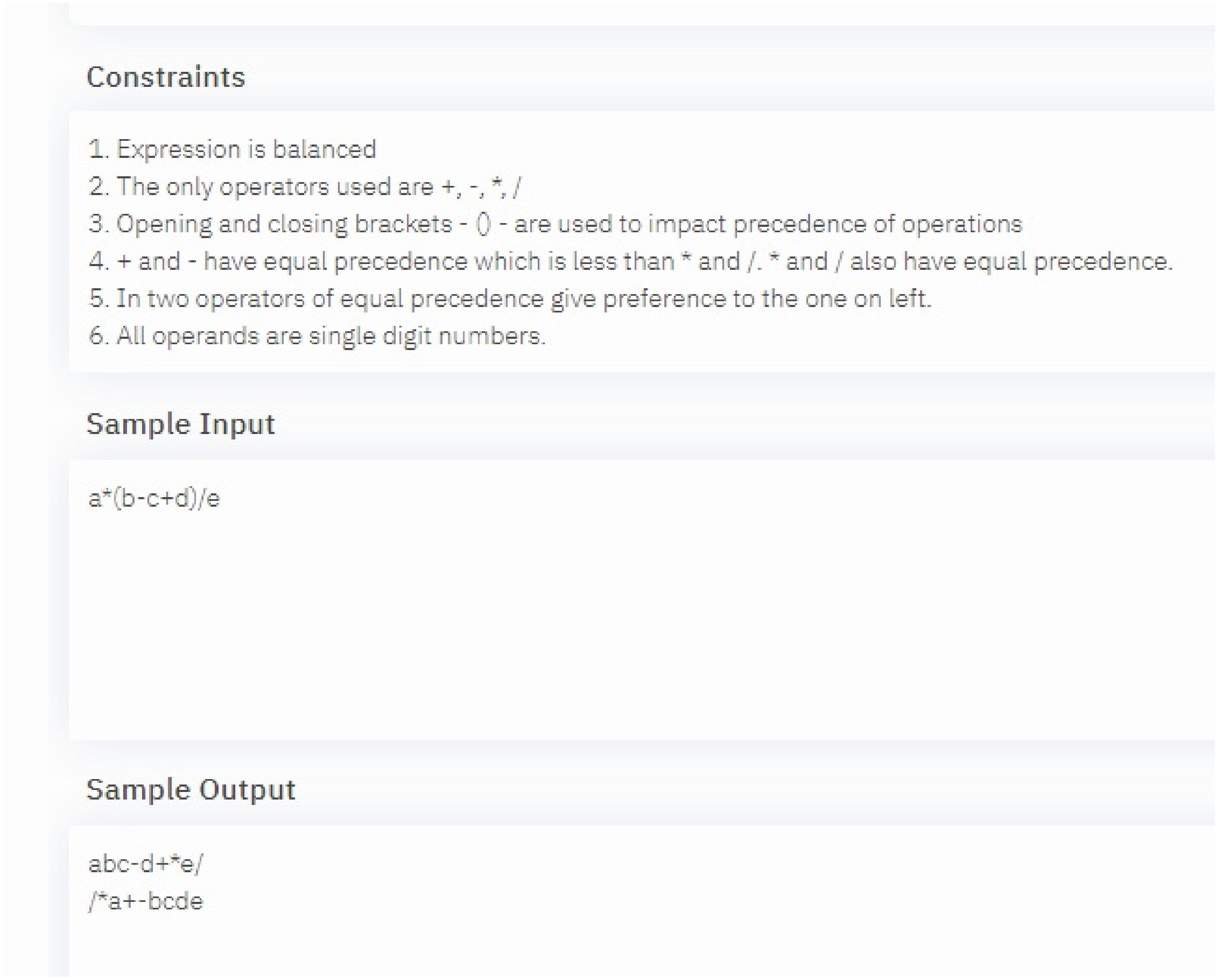
```java
            // ch is pushing itself now
            optors.push(ch);
        }
    }

    while(optors.size() != 0) {
        char optor = optors.pop();
        int v2 = opnds.pop();
        int v1 = opnds.pop();

        char opv = operation(v1, v2, optor);
        opnds.push(opv);
    }

    System.out.println(opnds.peek());
}
```



Infix Conversions

Easy

1. You are given an infix expression.
2. You are required to convert it to postfix and print it.
3. You are required to convert it to prefix and print it.



Constraints

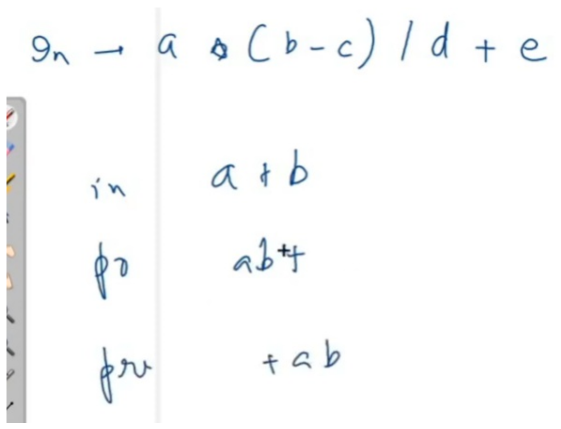
1. Expression is balanced
2. The only operators used are +, -, *, /
3. Opening and closing brackets - () - are used to impact precedence of operations
4. + and - have equal precedence which is less than * and /. * and / also have equal precedence.
5. In two operators of equal precedence give preference to the one on left.
6. All operands are single digit numbers.

Sample Input

```
a*(b-c+d)/e
```

Sample Output

```
abc-d+*e/
/*a+-bcde
```



In → a * (b - c) / d + e

in a + b

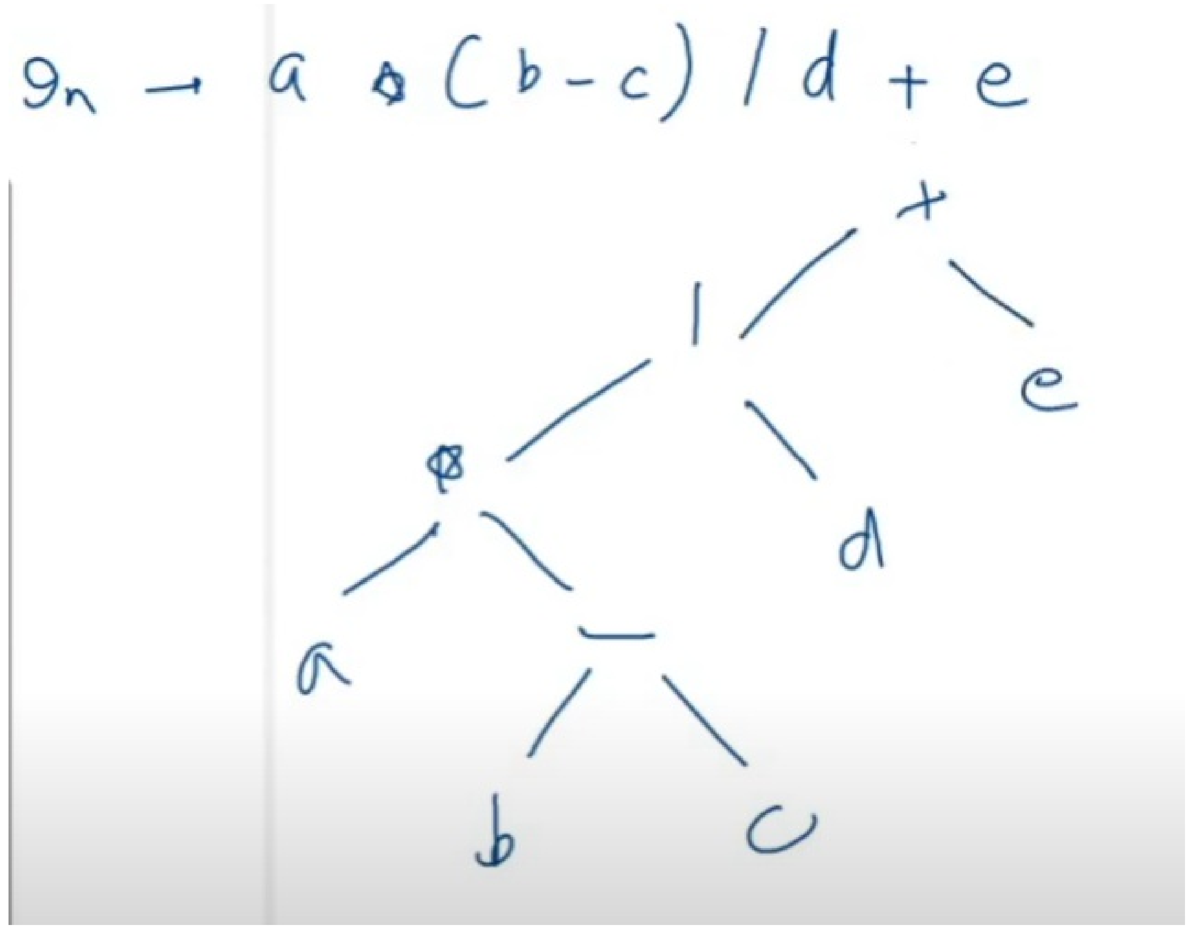
po ab+

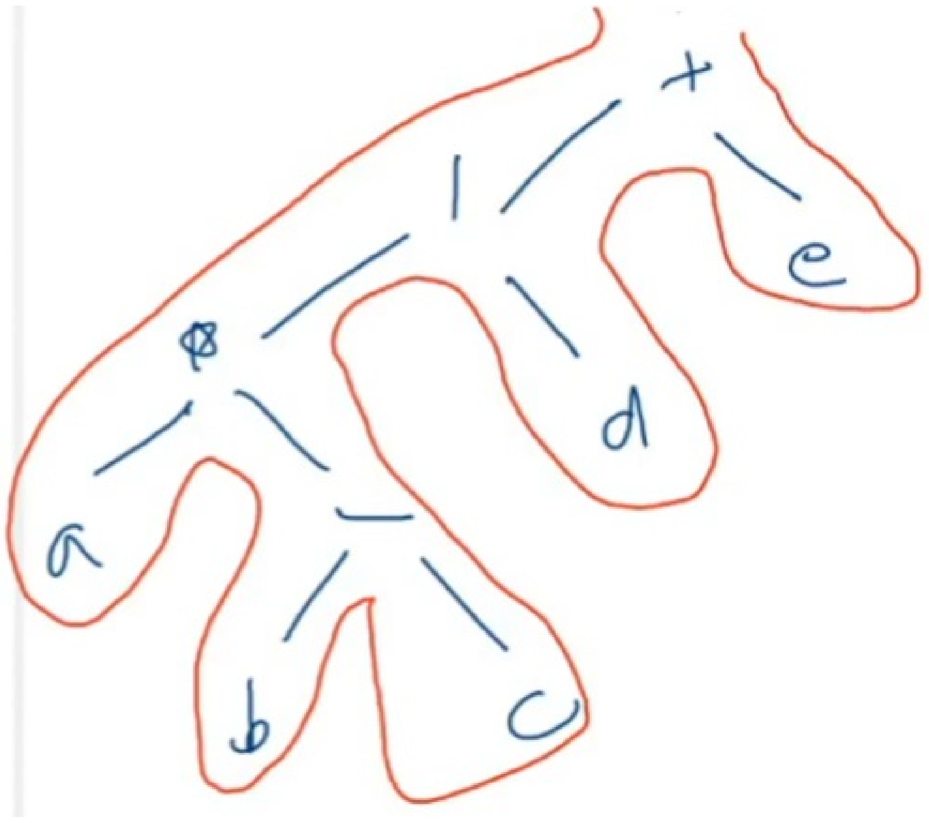
pre +ab

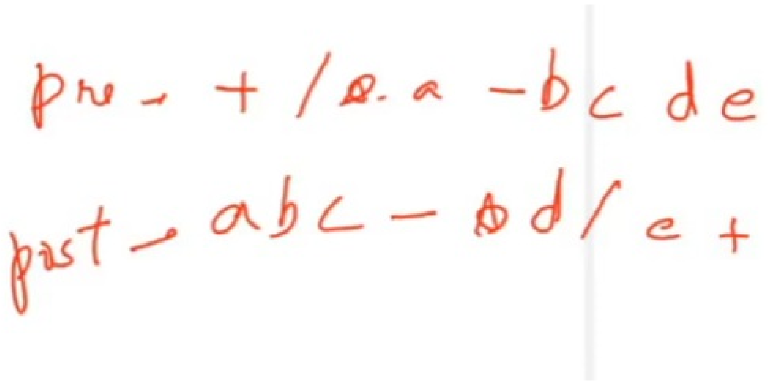
in - a+b

pre - operator then operands (+ab)

post - operands then operators (ab+)

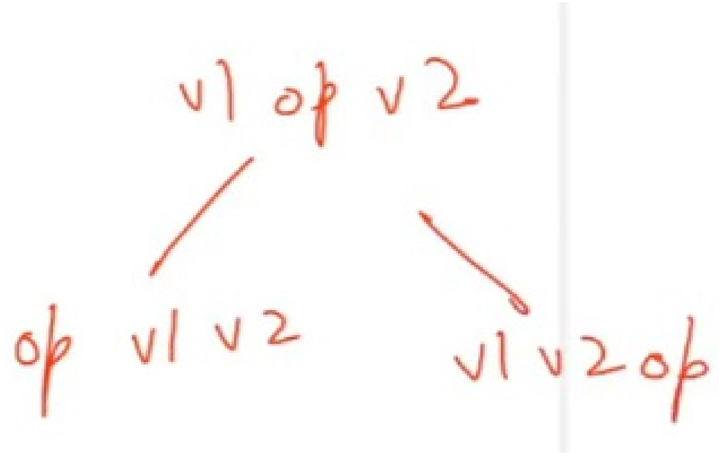


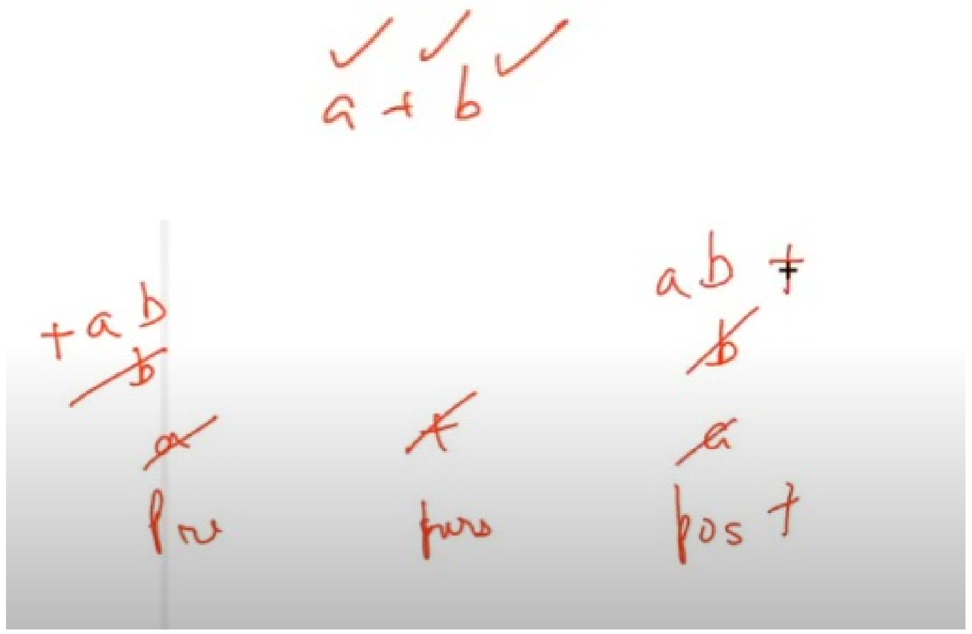




pre - + / * a - b c d e

post - a b c - * d / e +





Rules to process

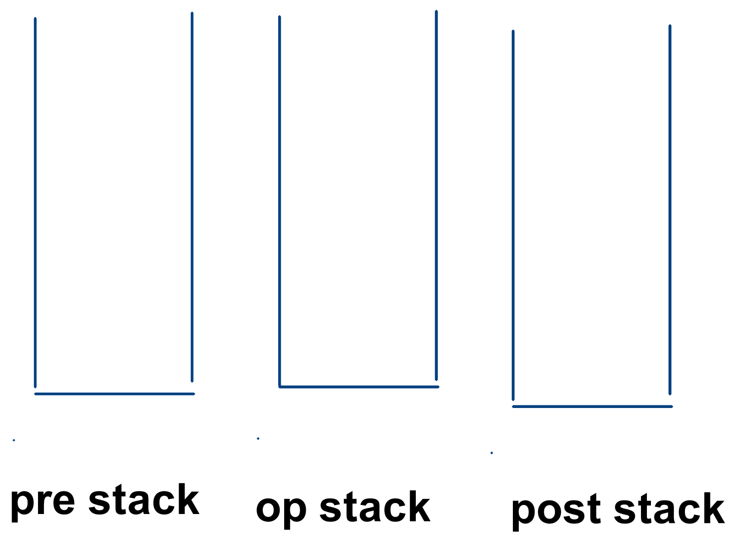
1. operand ==> push to pre/post stack

2. '(' ==> push to operator stack

3. ')' ==> pop till '(' and solve the operator and operand and push result in operand(pre/post) stack and
pop (

4. operator ==> if operator in stack has precedence >= curr operator , then pop (till stack empty or opening bracket or operator in stack has less priority) and

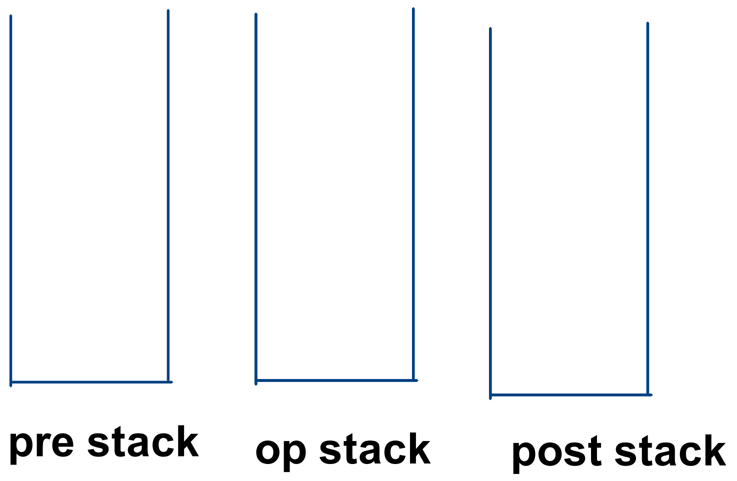
solve the operand with peek operator (HOW? for pre , Op v1 v2 , --- for post, v1 v2 Op) and push the result in operand(pre/post) stack and
push the current operator



$9n \rightarrow a * (b - c) / d + e$

infix expression

Solve yourself



Rules to process

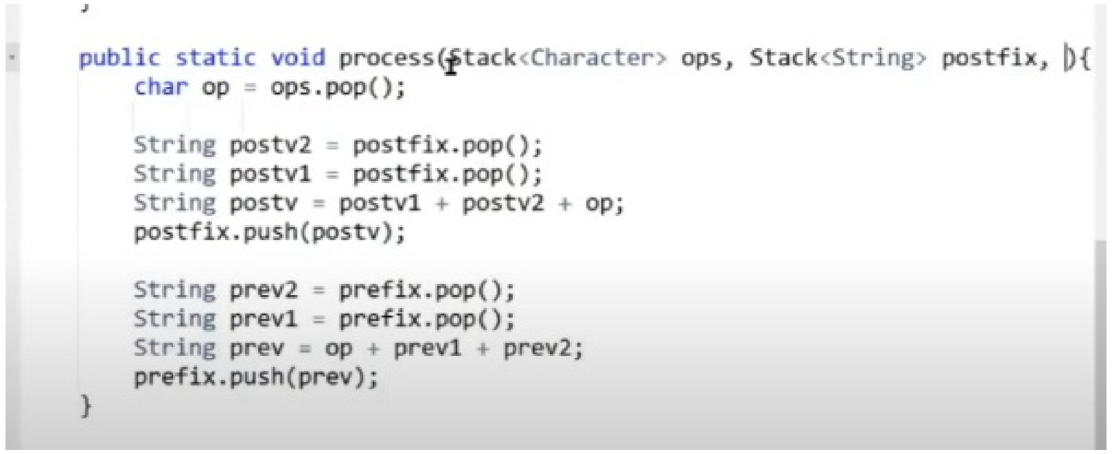
1. operand ==> push to pre/post stack

2. '(' ==> push to operator stack

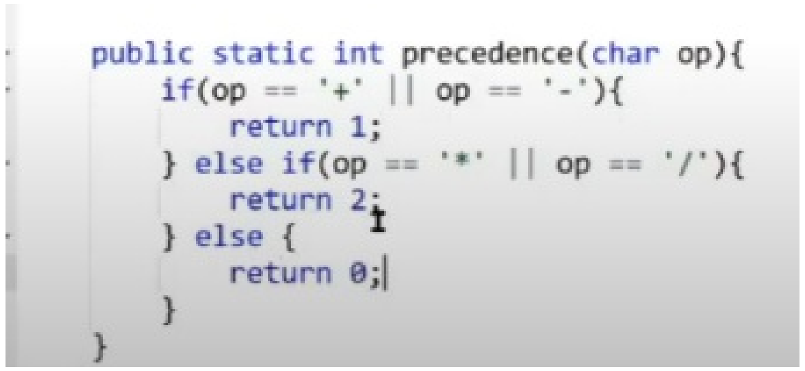
3. ')' ==> pop till '(' and solve the operator and operand and push result in operand(pre/post) stack and
pop (

4. operator ==> if operator in stack has precedence >= curr operator , then pop (till stack empty or opening bracket or operator in stack has less priority) and

solve the operand with peek operator (HOW? for pre , Op v1 v2 , --- for post, v1 v2 Op) and push the result in operand(pre/post) stack and
push the current operator

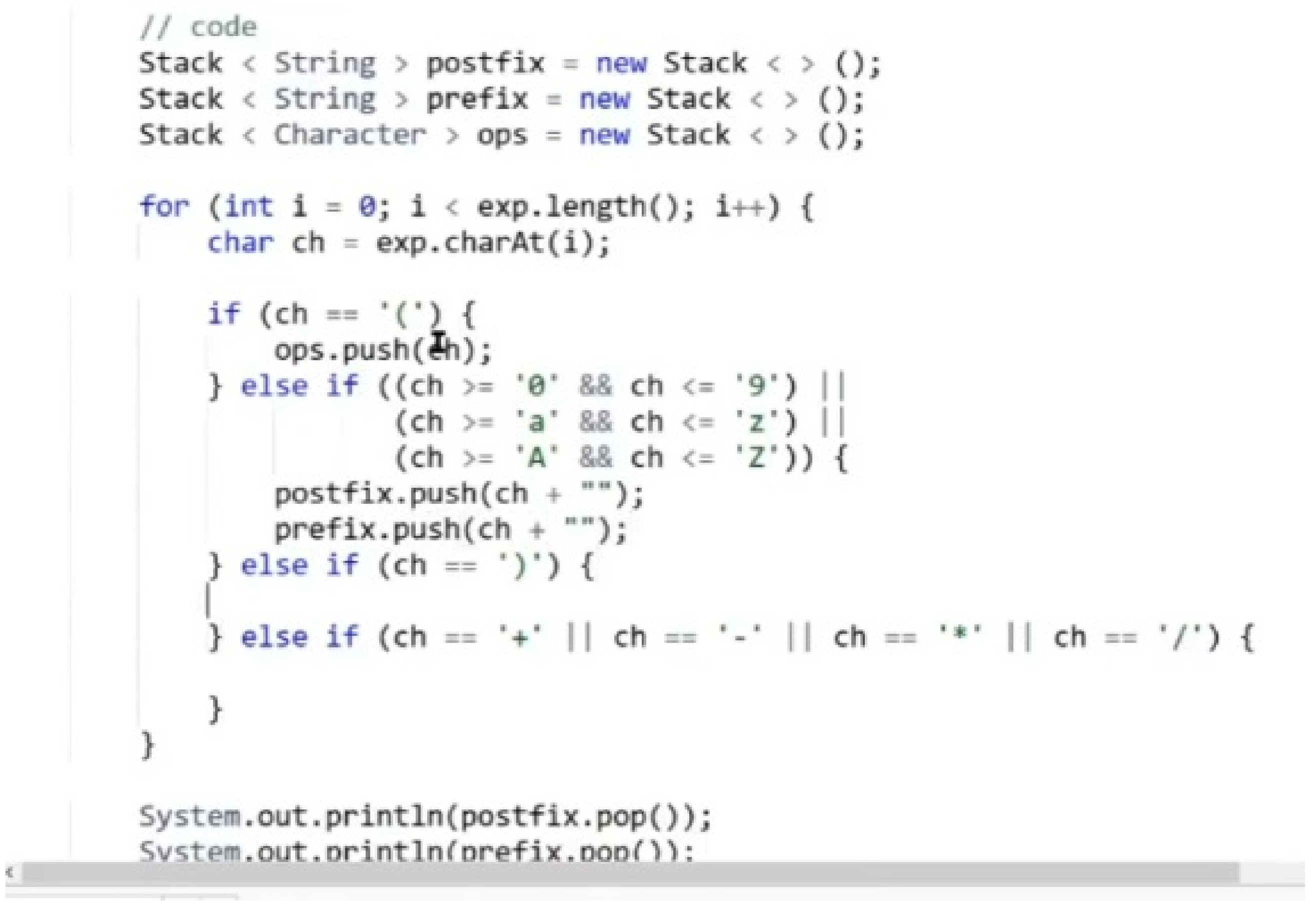


```java
public static void process(Stack<Character> ops, Stack<String> postfix, ){
    char op = ops.pop();

    String postv2 = postfix.pop();
    String postv1 = postfix.pop();
    String postv = postv1 + postv2 + op;
    postfix.push(postv);

    String prev2 = prefix.pop();
    String prev1 = prefix.pop();
    String prev = op + prev1 + prev2;
    prefix.push(prev);
}
```

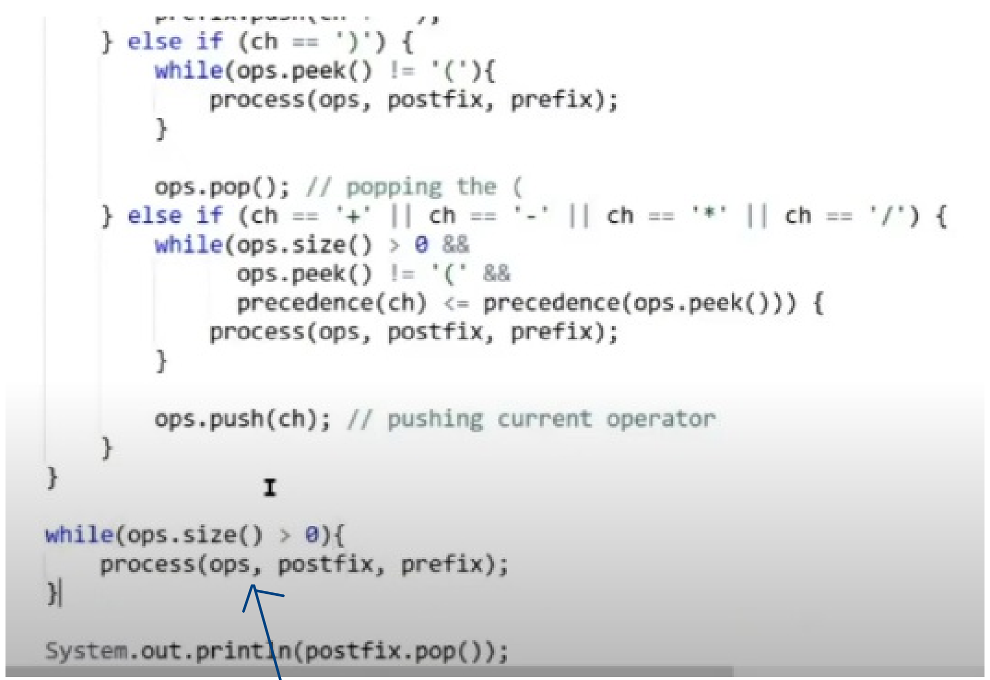


```java
public static int precedence(char op){
    if(op == '+' || op == '-'){
        return 1;
    } else if(op == '*' || op == '/'){
        return 2;
    } else {
        return 0;
    }
}
```

CODE

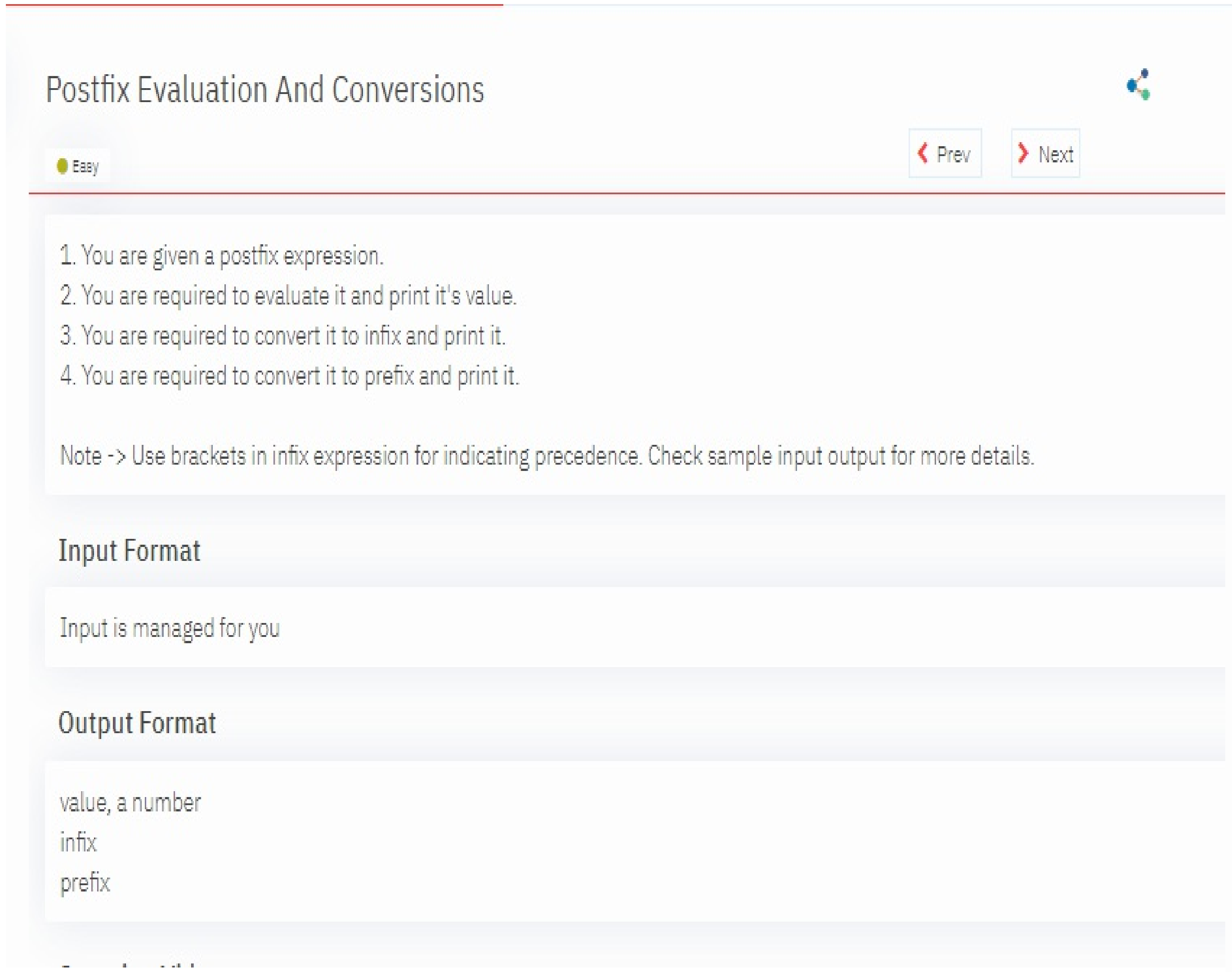


```java
// code
Stack < String > postfix = new Stack < > ();
Stack < String > prefix = new Stack < > ();
Stack < Character > ops = new Stack < > ();

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

    if (ch == '(') {
        ops.push(ch);
    } else if ((ch >= '0' && ch <= '9') ||
               (ch >= 'a' && ch <= 'z') ||
               (ch >= 'A' && ch <= 'Z')) {
        postfix.push(ch + "");
        prefix.push(ch + "");
    } else if (ch == ')') {

    } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {

    }
}

System.out.println(postfix.pop());
System.out.println(prefix.pop());
```



```java
    } else if (ch == ')') {
        while(ops.peek() != '('){
            process(ops, postfix, prefix);
        }

        ops.pop(); // popping the (
    } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
        while(ops.size() > 0 &&
              ops.peek() != '(' &&
              precedence(ch) <= precedence(ops.peek())) {
            process(ops, postfix, prefix);
        }

        ops.push(ch); // pushing current operator
    }
}

while(ops.size() > 0){
    process(ops, postfix, prefix);
}

System.out.println(postfix.pop());
```

solving for remaining operators present in operator stack



Postfix Evaluation And Conversions

1. You are given a postfix expression.
2. You are required to evaluate it and print it's value.
3. You are required to convert it to infix and print it.
4. You are required to convert it to prefix and print it.

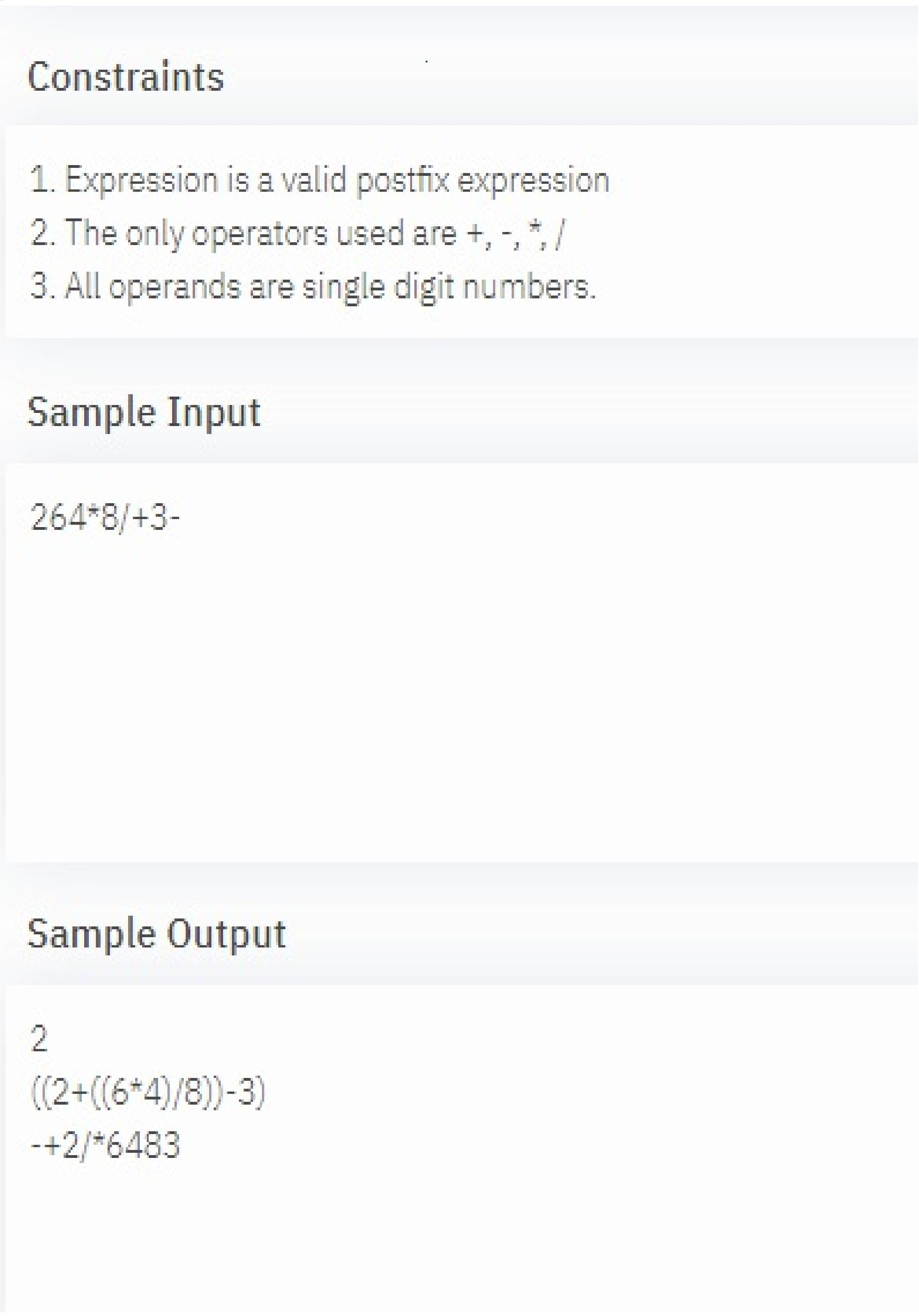
Note -> Use brackets in infix expression for indicating precedence. Check sample input output for more details.

Input Format

Input is managed for you

Output Format

value, a number
infix
prefix



Constraints

1. Expression is a valid postfix expression
2. The only operators used are +, -, *, /
3. All operands are single digit numbers.

Sample Input

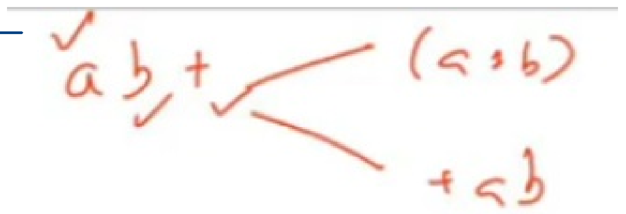
```
264*8/+3-
```

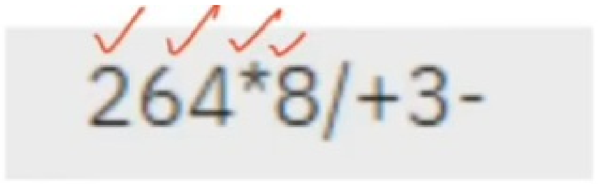
Sample Output

```
2
((2+((6*4)/8))-3)
-+2/*6483
```

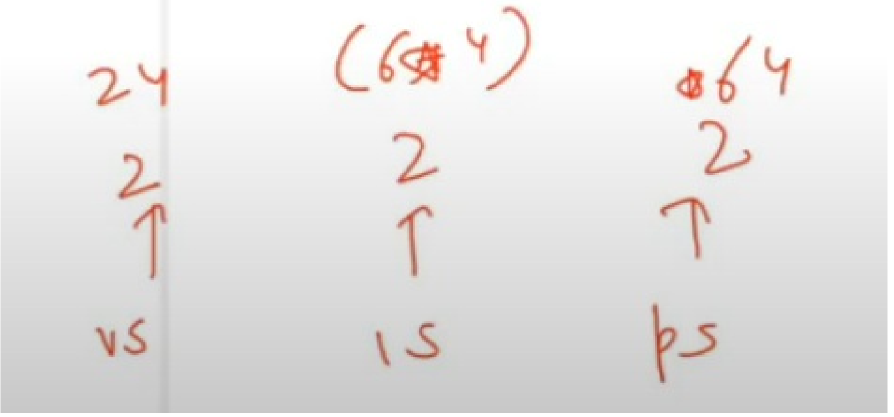
We are doing following things

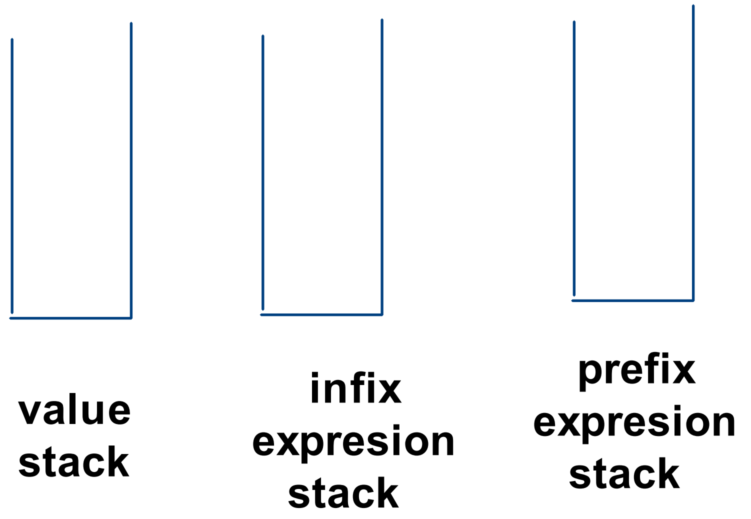
1. Evaluate postfix in value stack
2. Converting postfix to infix (expression skack)
3. Converting postfix to prefix (expression skack)

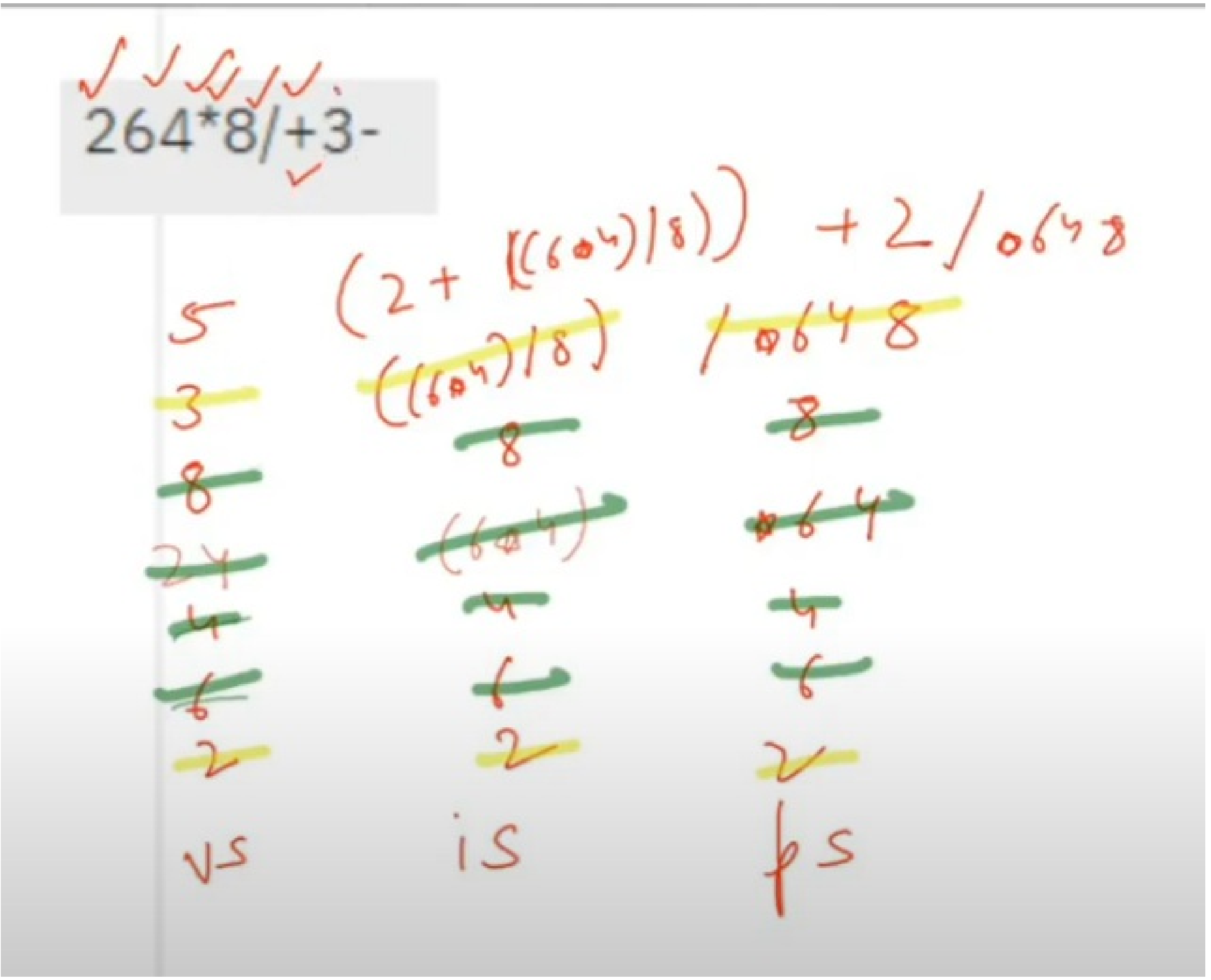



264*8/+3-

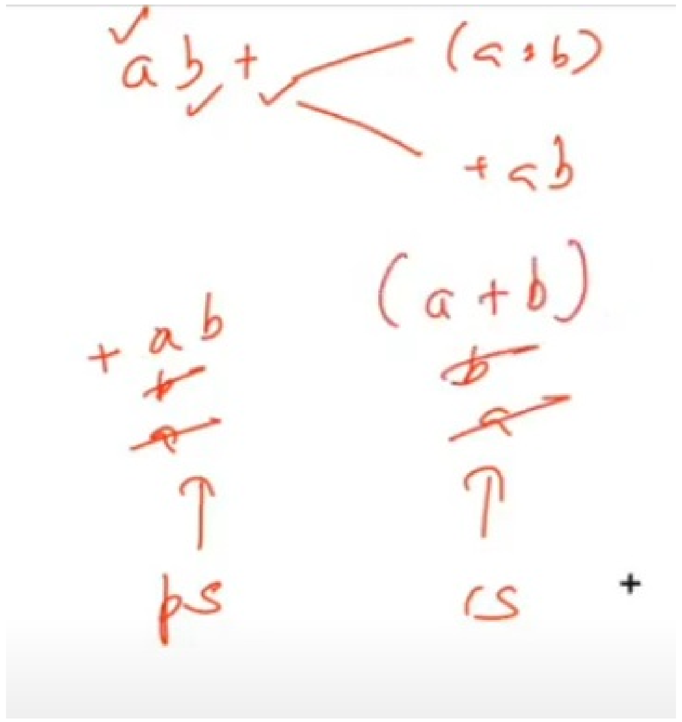
postfix expresion





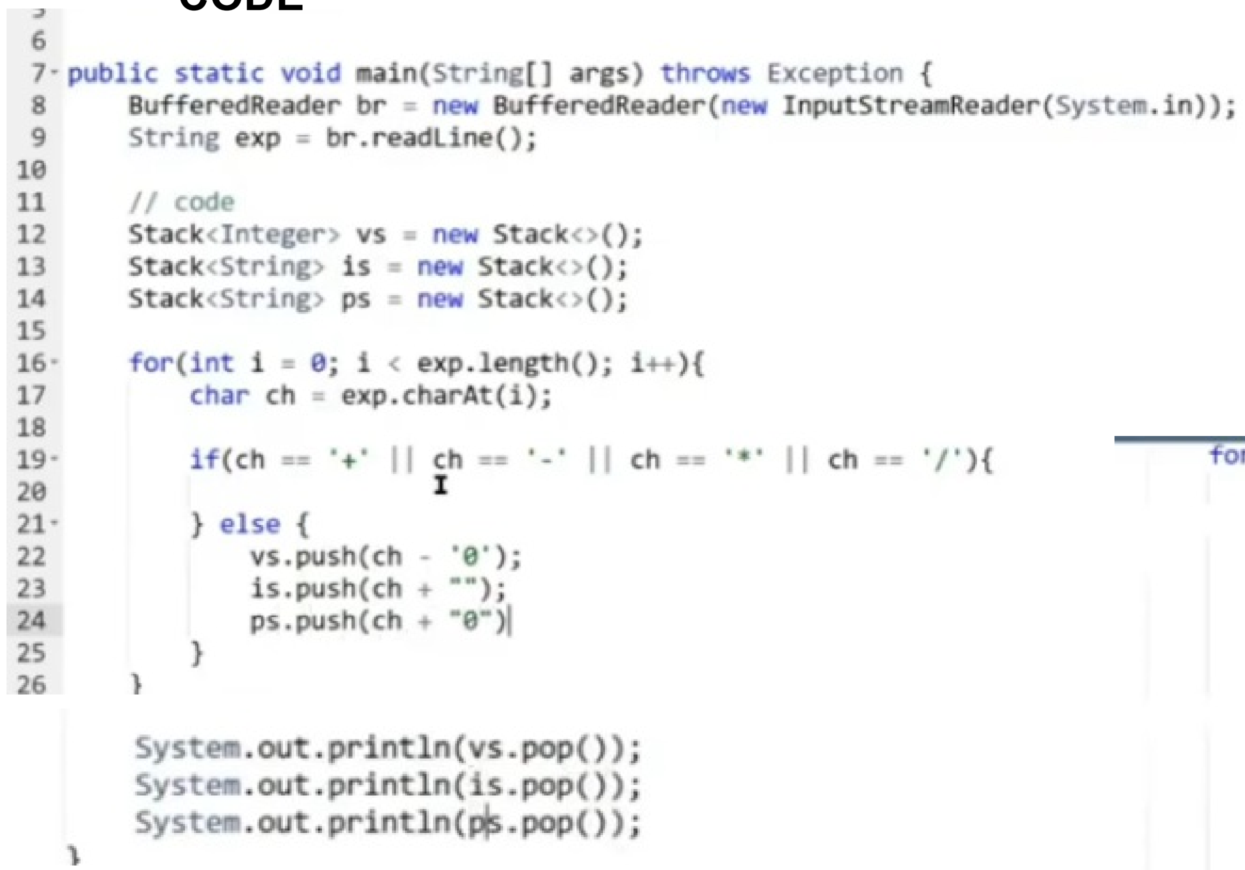


this is how we are going to approach

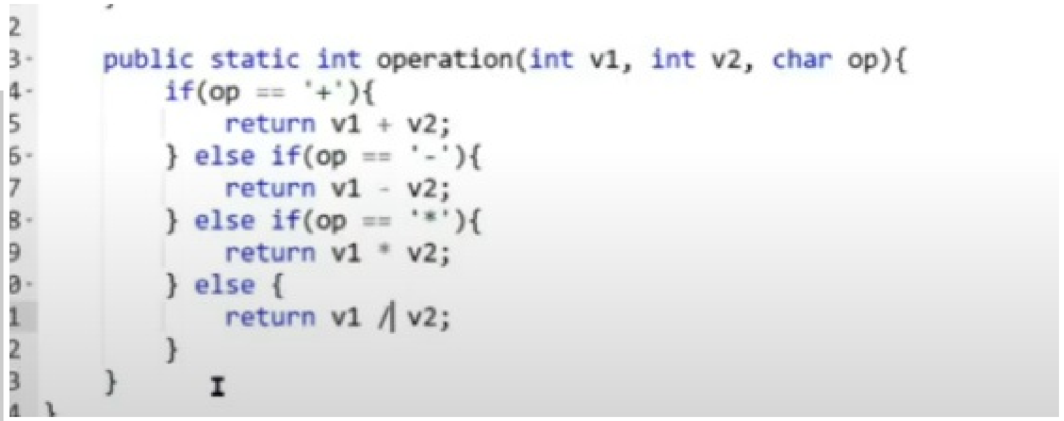


Here
1st pop -value 2
2nd pop- value 1

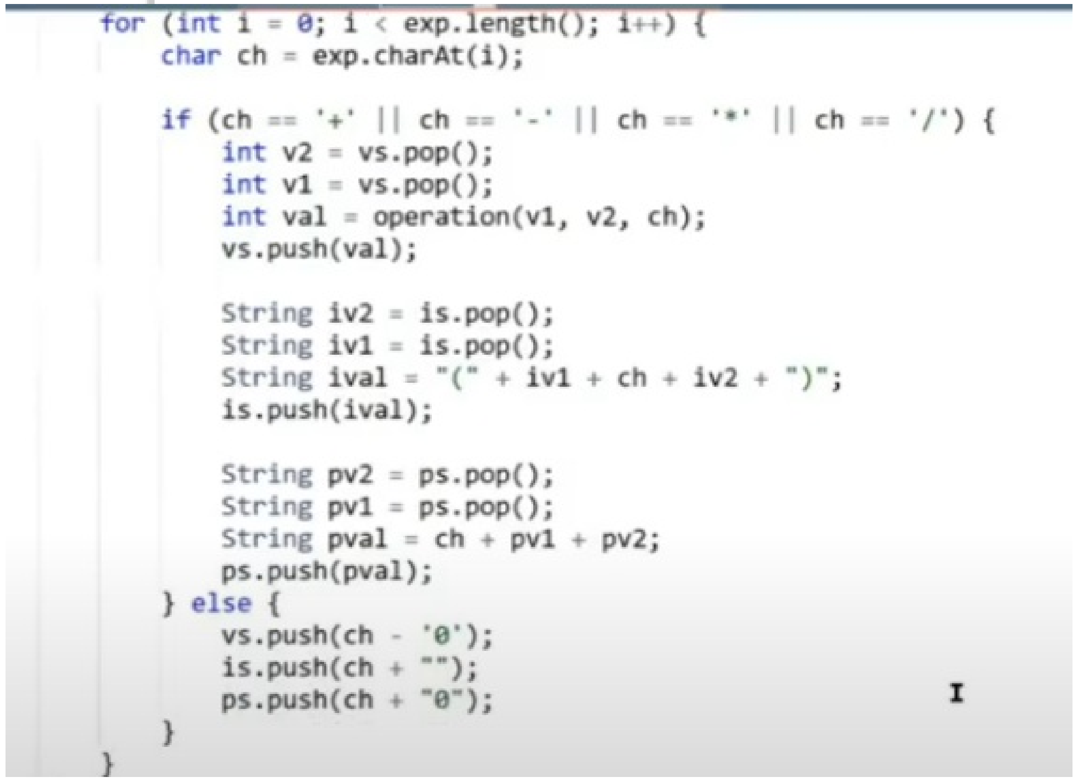
CODE



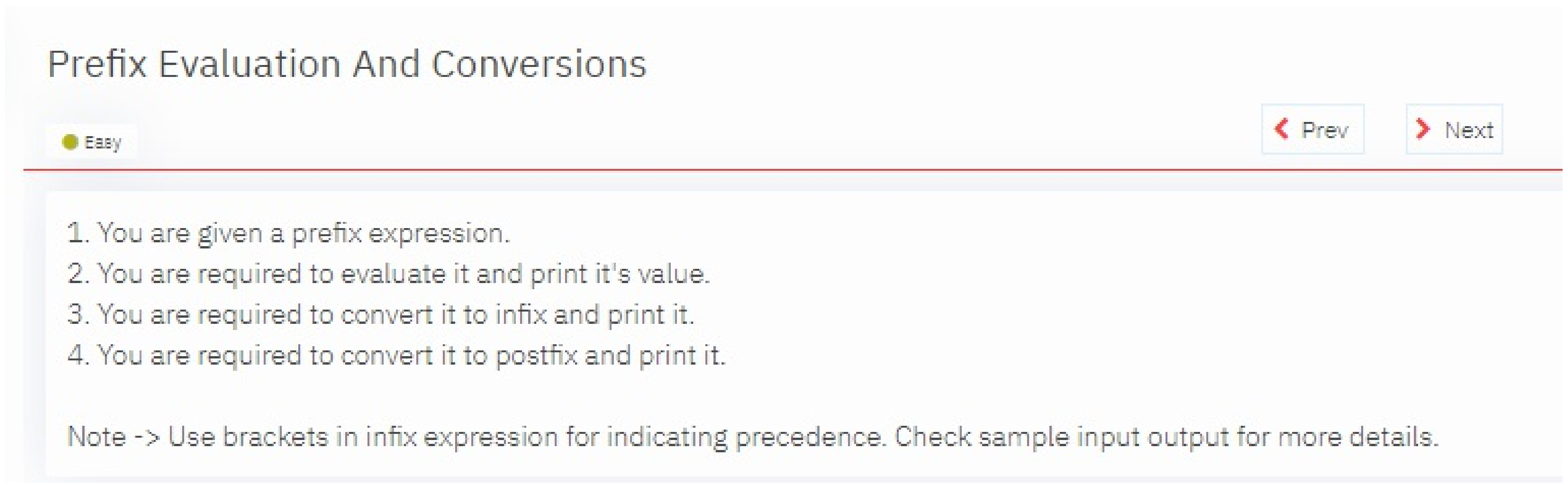
```java
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String exp = br.readLine();

    // code
    Stack<Integer> vs = new Stack<>();
    Stack<String> is = new Stack<>();
    Stack<String> ps = new Stack<>();

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

        if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){

        } else {
            vs.push(ch - '0');
            is.push(ch + "");
            ps.push(ch + "0")
        }
    }

     System.out.println(vs.pop());
     System.out.println(is.pop());
     System.out.println(ps.pop());
}
```



```java
    public static int operation(int v1, int v2, char op){
        if(op == '+'){
            return v1 + v2;
        } else if(op == '-'){
            return v1 - v2;
        } else if(op == '*'){
            return v1 * v2;
        } else {
            return v1 / v2;
        }
    }
}
```



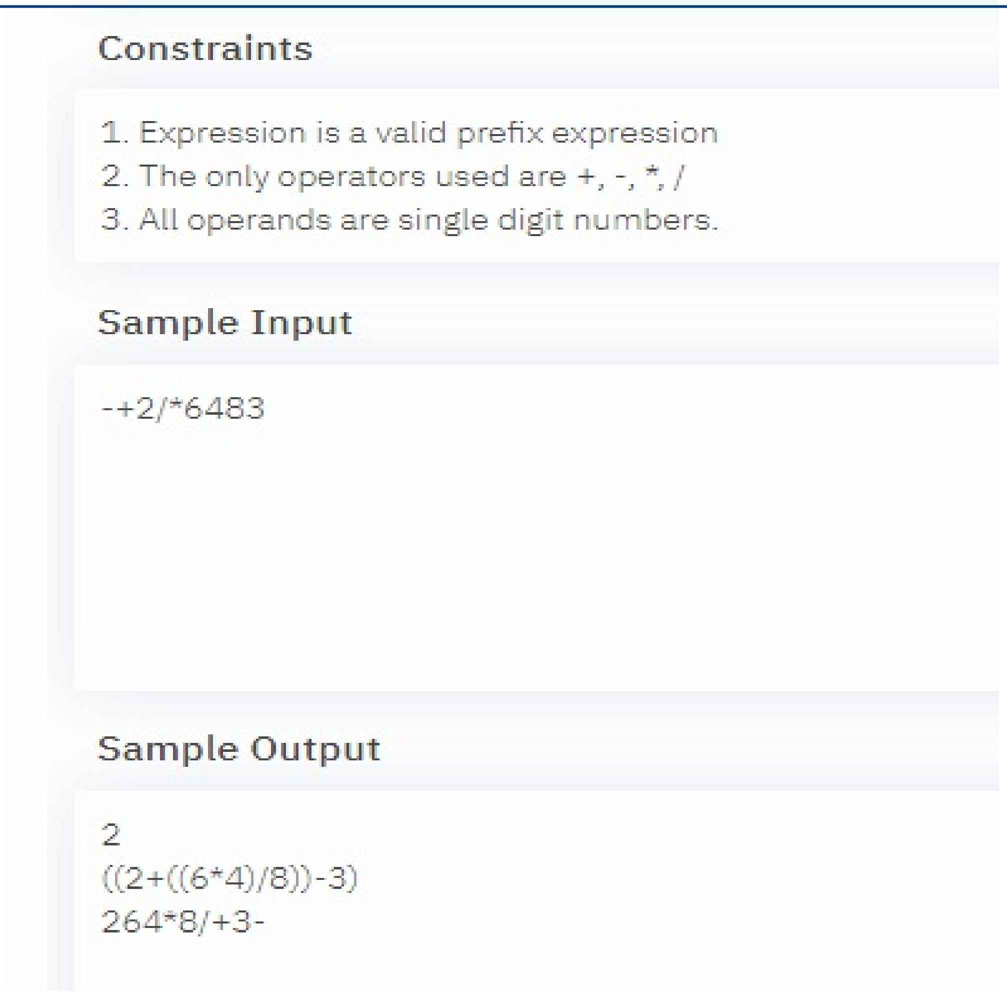
```java
for (int i = 0; i < exp.length(); i++) {
    char ch = exp.charAt(i);

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

        String iv2 = is.pop();
        String iv1 = is.pop();
        String ival = "(" + iv1 + ch + iv2 + ")";
        is.push(ival);

        String pv2 = ps.pop();
        String pv1 = ps.pop();
        String pval = ch + pv1 + pv2;
        ps.push(pval);
    } else {
        vs.push(ch - '0');
        is.push(ch + "");
        ps.push(ch + "0");
    }
}
```



Prefix Evaluation And Conversions

1. You are given a prefix expression.
2. You are required to evaluate it and print it's value.
3. You are required to convert it to infix and print it.
4. You are required to convert it to postfix and print it.

Note -> Use brackets in infix expression for indicating precedence. Check sample input output for more details.



Constraints

1. Expression is a valid prefix expression
2. The only operators used are +, -, *, /
3. All operands are single digit numbers.

Sample Input

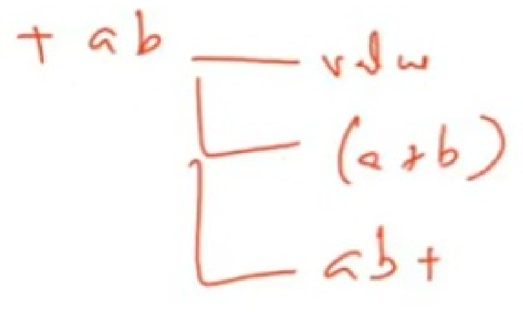
```
-+2/*6483
```

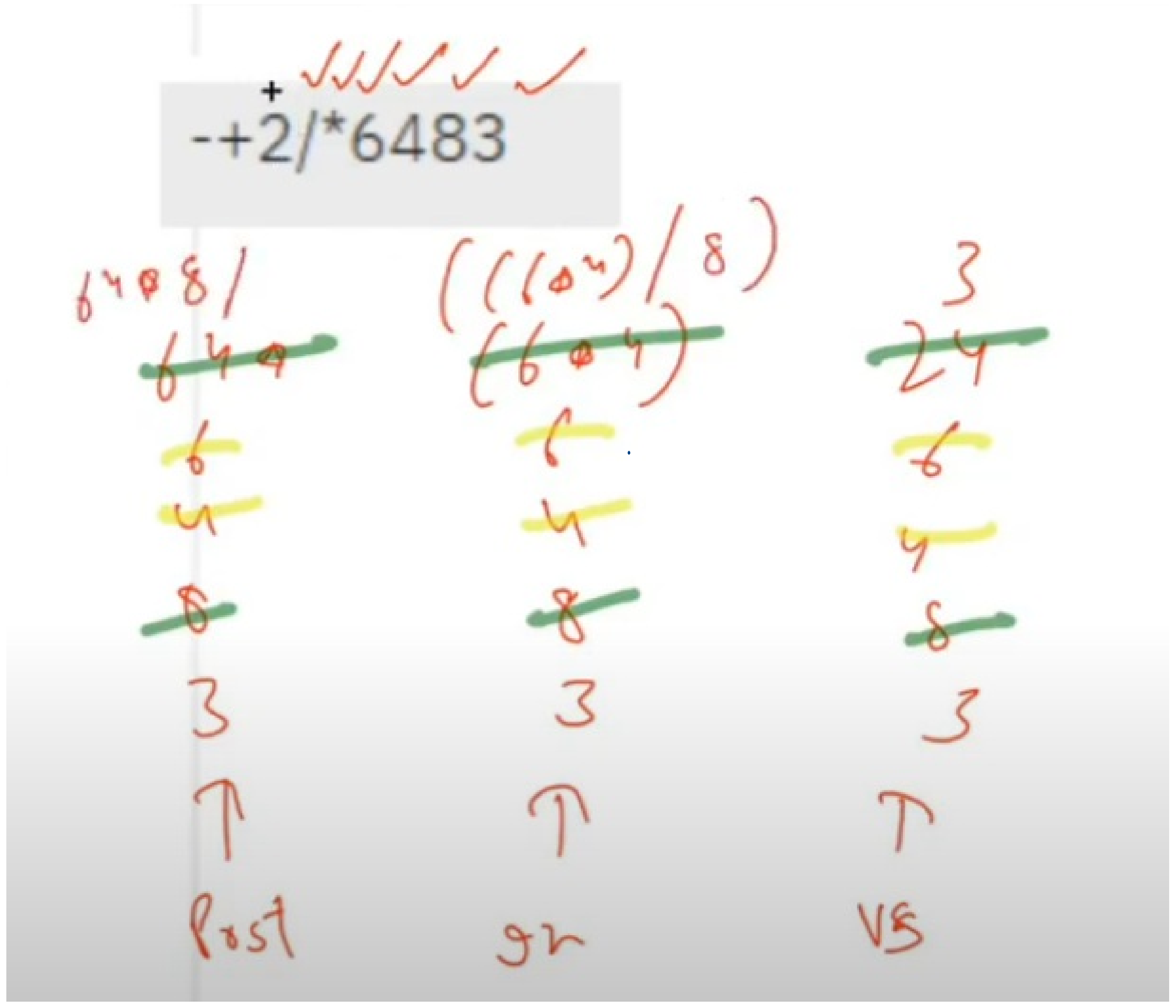
Sample Output

```
2
((2+((6*4)/8))-3)
264*8/+3-
```

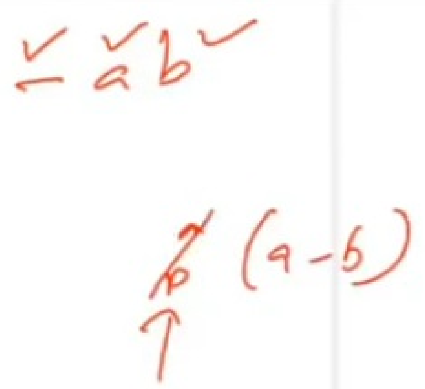
We are doing following things

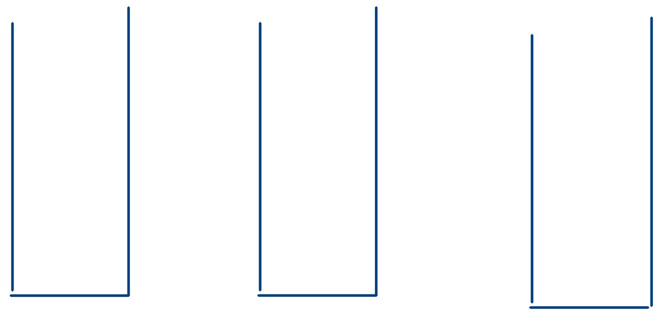
1. Evaluate prefix in value stack
2. Converting prefix to infix (expression skack)
3. Converting prefix to postfix (expression skack)





Here we are evaluating from back
1st pop -value 1
2nd pop- value 2



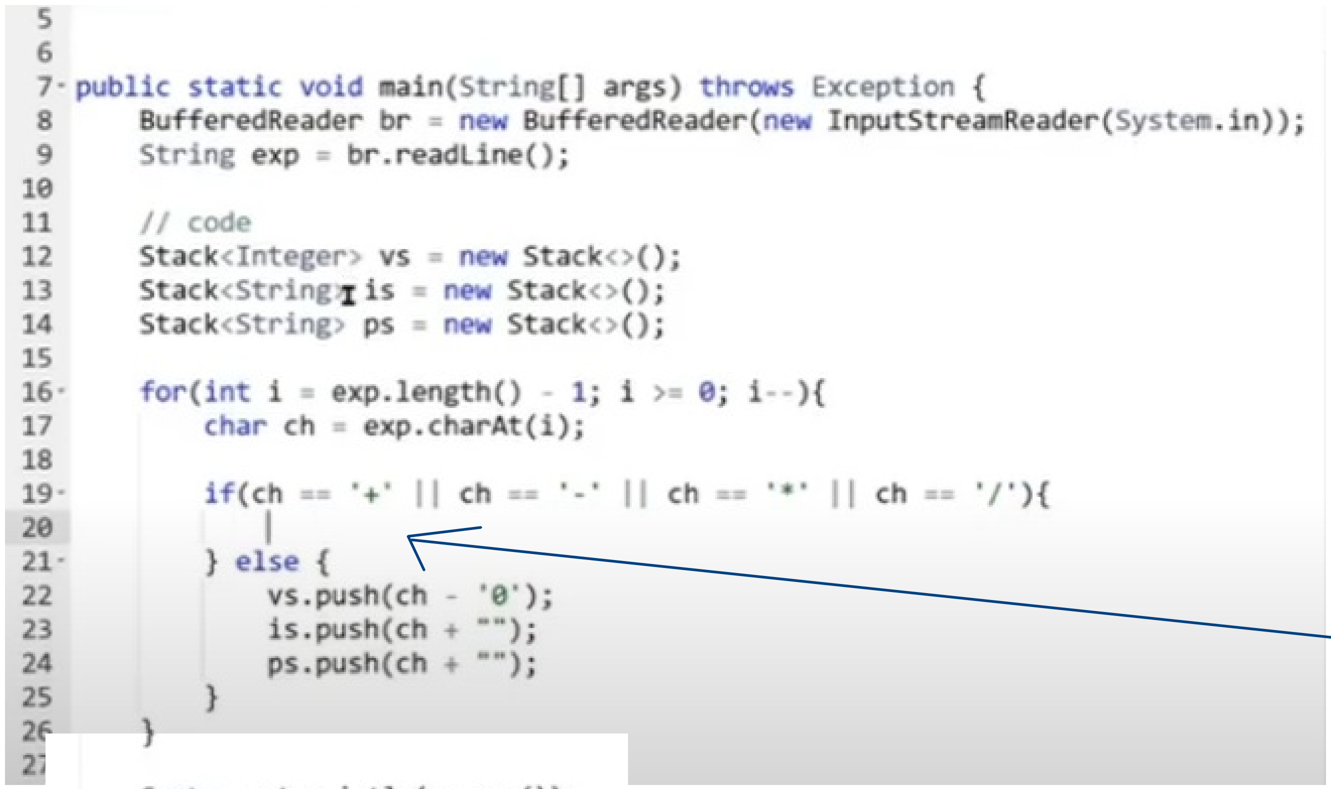


postfix expresion stack

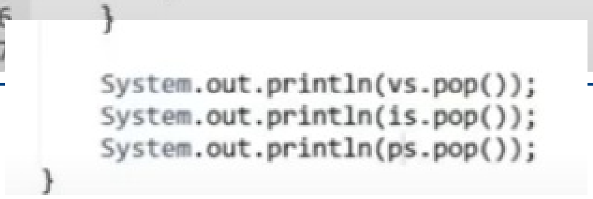
infix expresion stack

value stack

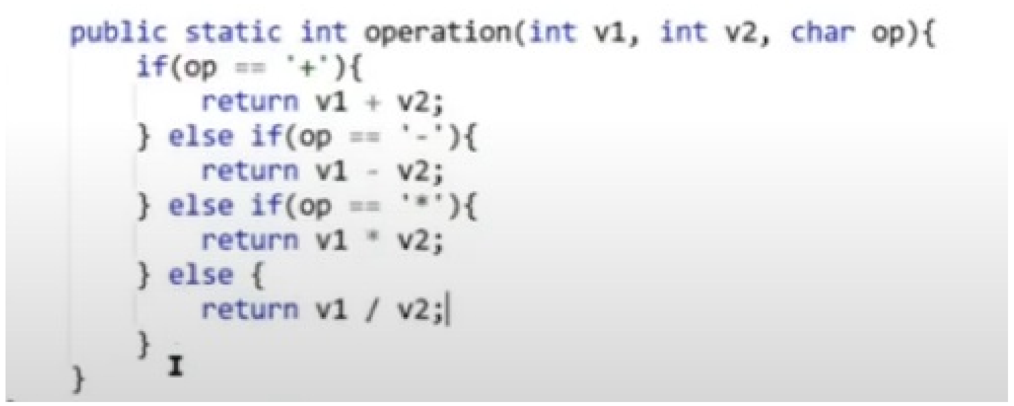
CODE



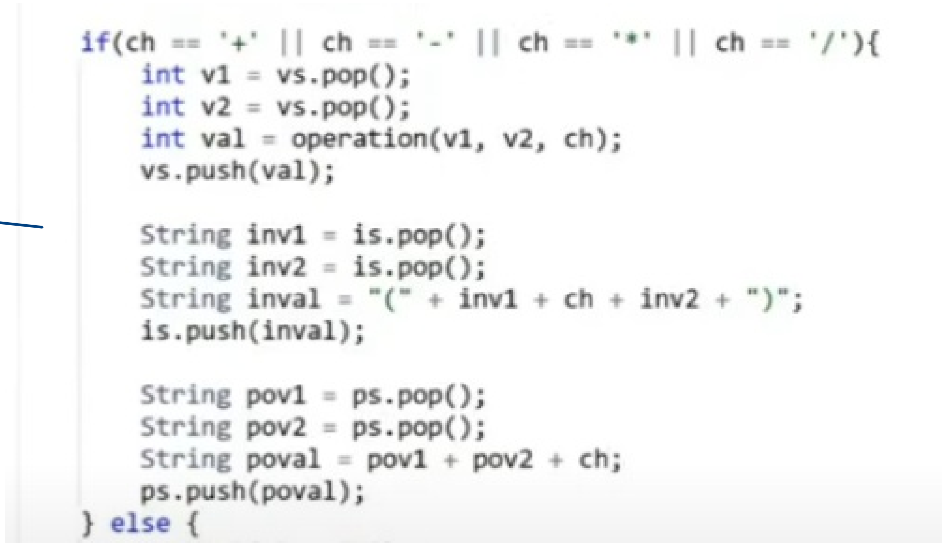
```java
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String exp = br.readLine();

    // code
    Stack<Integer> vs = new Stack<>();
    Stack<String> is = new Stack<>();
    Stack<String> ps = new Stack<>();

    for(int i = exp.length() - 1; i >= 0; i--){
        char ch = exp.charAt(i);

        if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){

        } else {
            vs.push(ch - '0');
            is.push(ch + "");
            ps.push(ch + "");
        }
    }
```



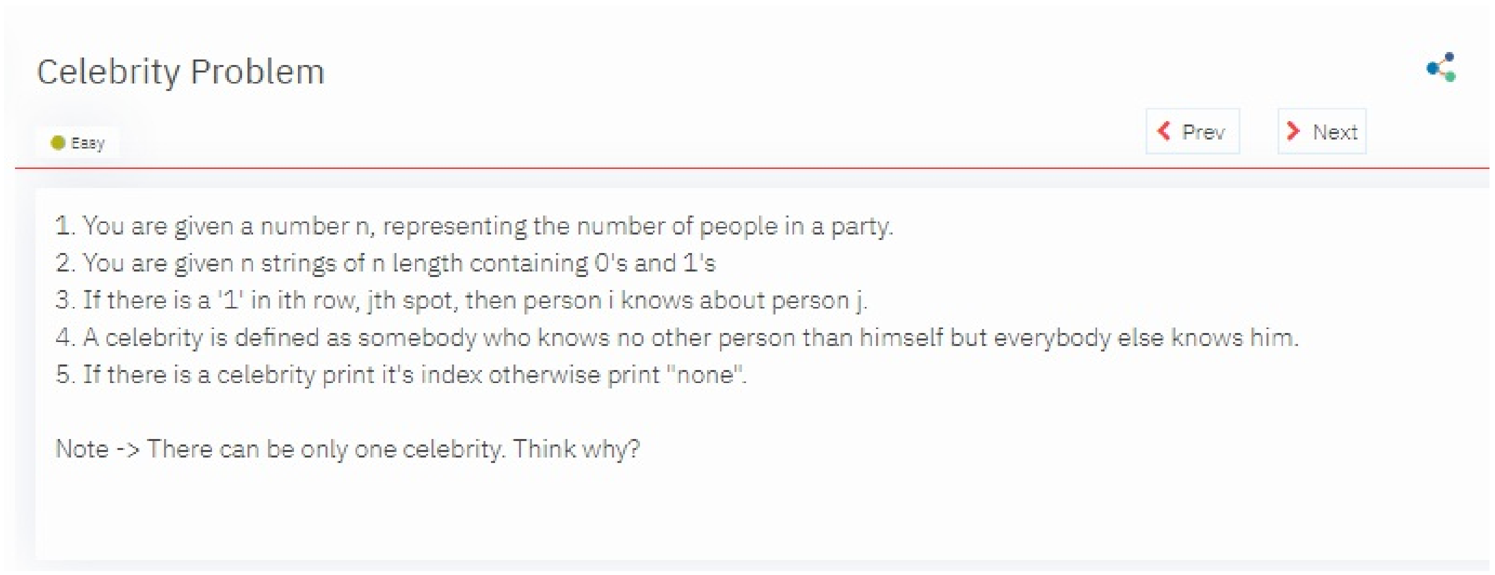
```java
    System.out.println(vs.pop());
    System.out.println(is.pop());
    System.out.println(ps.pop());
}
```



```java
public static int operation(int v1, int v2, char op){
    if(op == '+'){
        return v1 + v2;
    } else if(op == '-'){
        return v1 - v2;
    } else if(op == '*'){
        return v1 * v2;
    } else {
        return v1 / v2;
    }
}
```



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

    String inv1 = is.pop();
    String inv2 = is.pop();
    String inval = "(" + inv1 + ch + inv2 + ")";
    is.push(inval);

    String pov1 = ps.pop();
    String pov2 = ps.pop();
    String poval = pov1 + pov2 + ch;
    ps.push(poval);
} else {
```

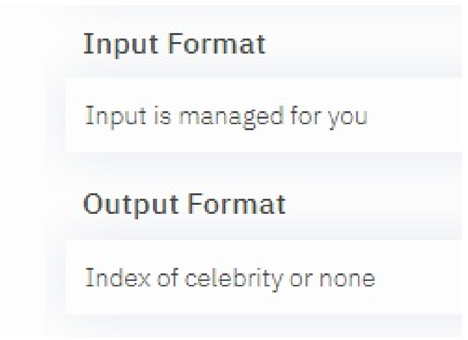

Celebrity Problem

Easy

Prev Next

1. You are given a number n, representing the number of people in a party.
2. You are given n strings of n length containing 0's and 1's
3. If there is a '1' in ith row, jth spot, then person i knows about person j.
4. A celebrity is defined as somebody who knows no other person than himself but everybody else knows him.
5. If there is a celebrity print it's index otherwise print "none".

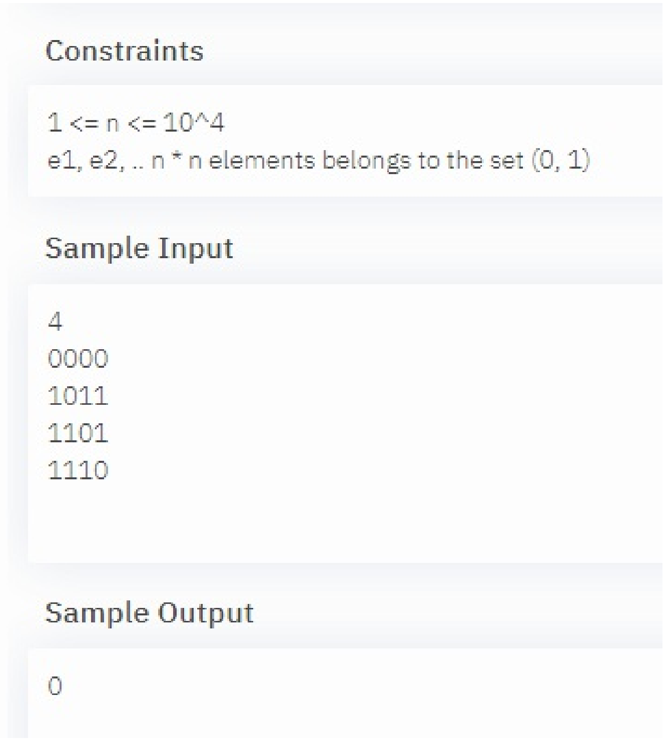
Note -> There can be only one celebrity. Think why?


Input Format

Input is managed for you

Output Format

Index of celebrity or none


Constraints

1 <= n <= 10^4
e1, e2, .. n * n elements belongs to the set (0, 1)

Sample Input

```
4
0000
1011
1101
1110
```

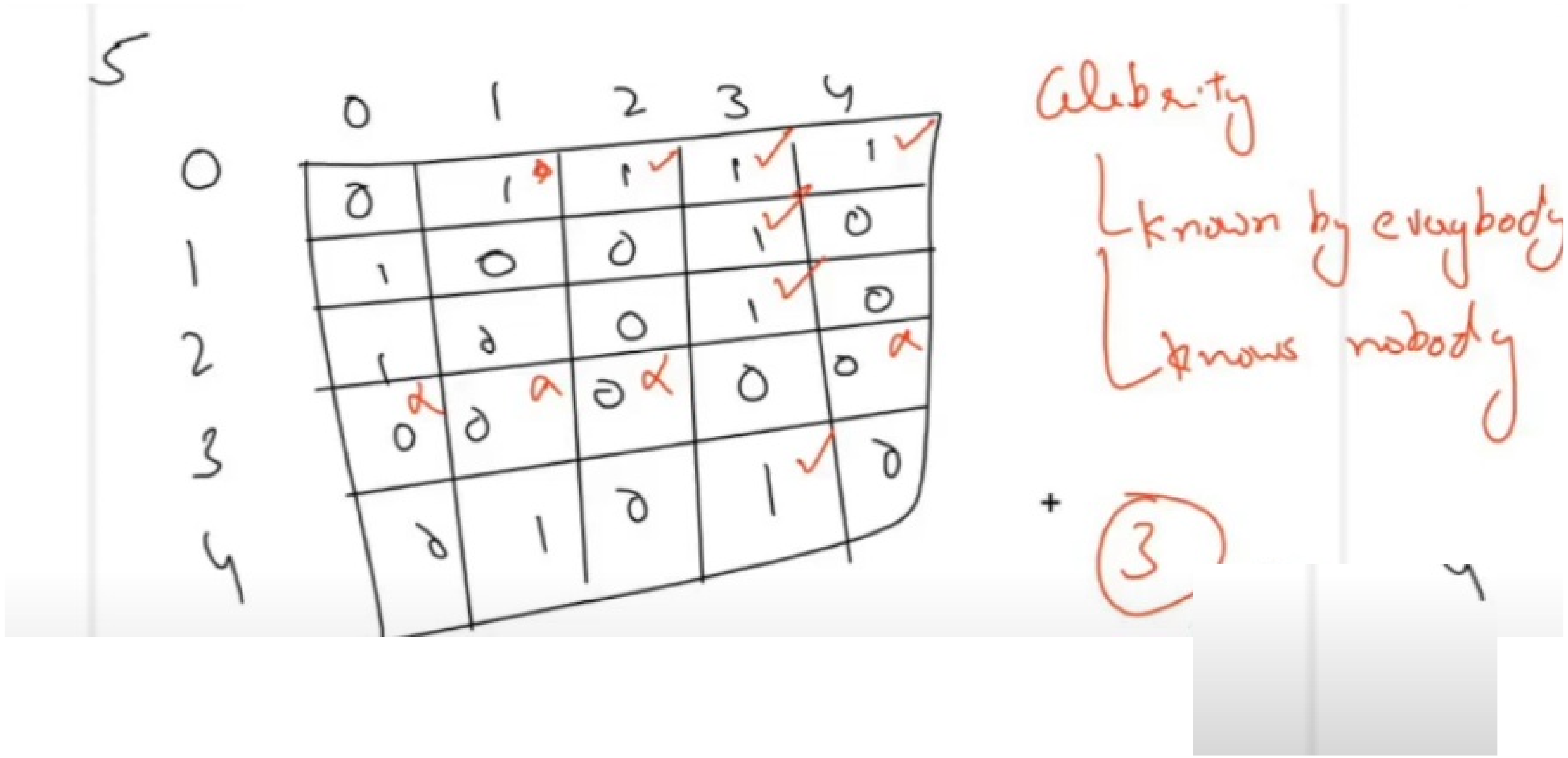
Sample Output

```
0
```

Need to be solved at Time complexity O(n)

Here 3 is celebrity
because, everybody knows 3
but 3 dont know anybody

(i,j)=1 represent - i knows j
(i,j)=0 represent - i dont knows j



5

	0	1	2	3	4
0	0	1	1 ✓	1 ✓	1 ✓
1	1	0	0	1 ✓	0
2	1	0	0	1 ✓	0
3	0 α	0 a	0 α	0	0 α
4	0	1	0	1 ✓	0

Celebrity
known by everybody
knows nobody
3

There can be no 2 celebrity. WHY?

	0	1	2	3
0	×		✓	
1		×	✓	
2	×	×	×	×
3			✓	×

0, 1, 3

. A celebrity is defined as somebody who knows no other person than himself but everybody else knows him.

Every body knows 2 and 2 dont know anybody, therefore 2 is celebrity
If 2 is celebrity then 0,1,3 cannot be celebrity
WHY????
Because In perspective of 0 1 3,
there is only one guy 2
which doesn't know 0,1,3

but according to celebrity statement ,
if 0 want to be celebrity , every body should know 0
if 1 want to be celebrity , every body should know 1
if 3 want to be celebrity , every body should know 3
BUT there is one guy 2 which does not know 0,1,3
This itself makes it imposible to make 0,1,3 as celebrity.

How to approach? --*Elmination approach*(means, A, B, C is not ans then remainig D is ans)

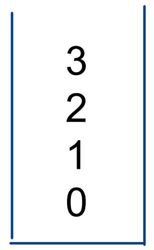
According to celebrity defn -
everybody know him
and he dont know anybody

	0	1	2	3
0	✗	✓	✓	✓
1	✓	✗	✓	✗
2	✗	✗	✗	✗
3	✓	✓	✓	✗

So far now we have checked cell [2,3] , [1,2] , [0,2]
But we are not sure if 2 is celebrity
so 2 is potential candidate.

so check all row 2 (=No)and column 2(=Yes)
then 2 is celibrity else he is not celebrity.

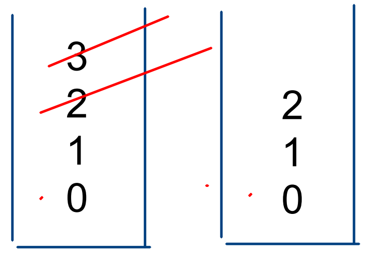
Push 0 1 2 3 on stack



pop 3 and 2 and eliminate one of them and push the remaining

check if 2 knows 3 ? No

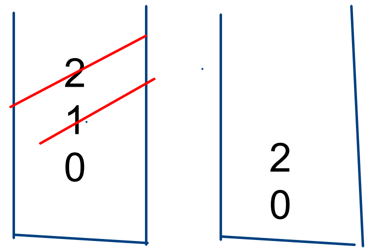
i.e 3 cannot be celebrity because
everybody should know celebrity so
eliminate 3 and push 2



pop 2 and 1 and eliminate one of them and push the remaining

check if 1 knows 2 ? Yes

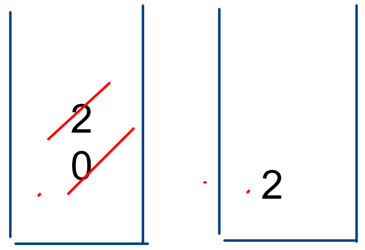
i.e 1 cannot be celebrity because
celebrity doesnt know anybody
so eliminate 1 and push 2

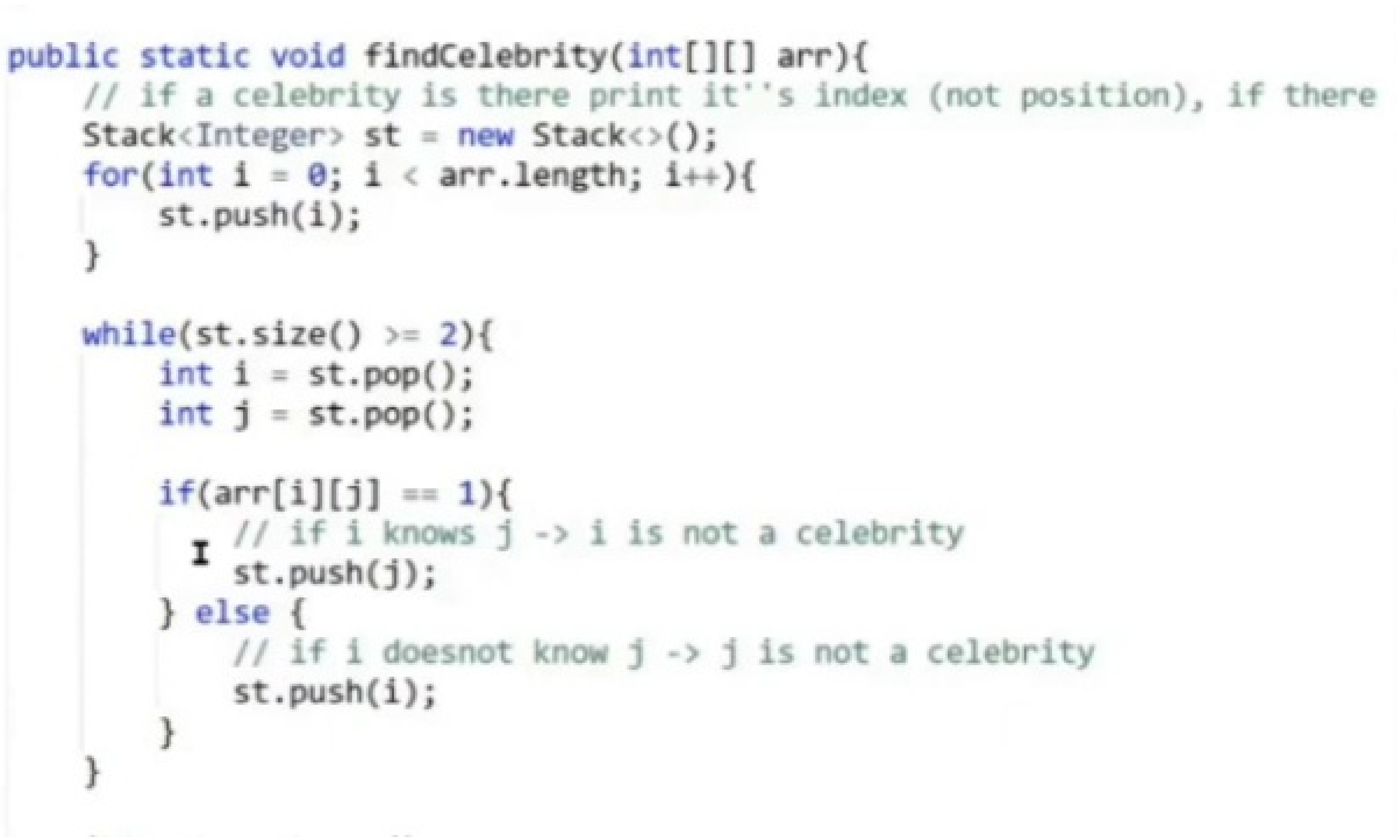


pop 2 and 0 and eliminate one of them and push the remaining

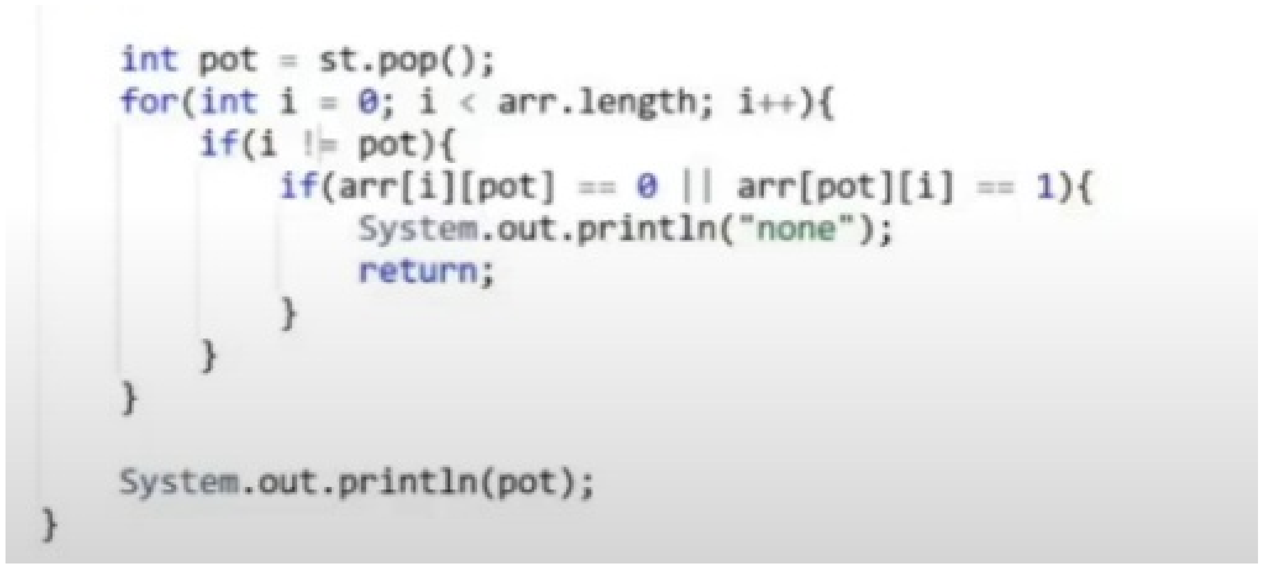
check if 0 knows 2 ? Yes

i.e 0 cannot be celebrity because
celebrity doesnt know anybody
so eliminate 0 and push 2

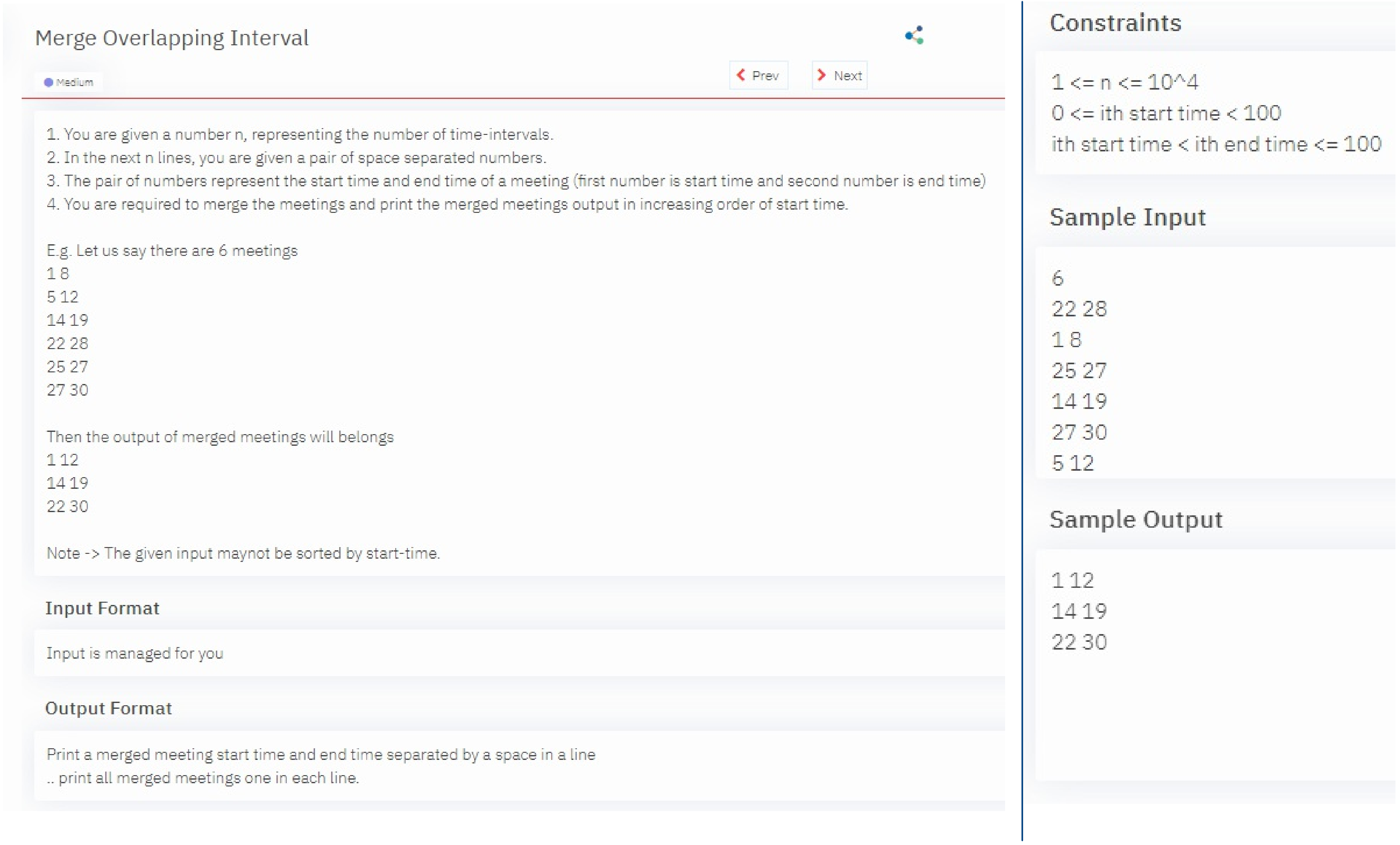




```java
public static void findCelebrity(int[][] arr){
    // if a celebrity is there print it''s index (not position), if there
    Stack<Integer> st = new Stack<>();
    for(int i = 0; i < arr.length; i++){
        st.push(i);
    }

    while(st.size() >= 2){
        int i = st.pop();
        int j = st.pop();

        if(arr[i][j] == 1){
            // if i knows j -> i is not a celebrity
            st.push(j);
        } else {
            // if i doesnot know j -> j is not a celebrity
            st.push(i);
        }
    }
```



```java
    int pot = st.pop();
    for(int i = 0; i < arr.length; i++){
        if(i != pot){
            if(arr[i][pot] == 0 || arr[pot][i] == 1){
                System.out.println("none");
                return;
            }
        }
    }

    System.out.println(pot);
}
```


Merge Overlapping Interval

Medium

Prev Next

1. You are given a number n, representing the number of time-intervals.
2. In the next n lines, you are given a pair of space separated numbers.
3. The pair of numbers represent the start time and end time of a meeting (first number is start time and second number is end time)
4. You are required to merge the meetings and print the merged meetings output in increasing order of start time.

E.g. Let us say there are 6 meetings
```
1 8
5 12
14 19
22 28
25 27
27 30
```

Then the output of merged meetings will belongs
```
1 12
14 19
22 30
```

Note -> The given input maynot be sorted by start-time.

Input Format

Input is managed for you

Output Format

Print a merged meeting start time and end time separated by a space in a line
.. print all merged meetings one in each line.

Constraints

1 <= n <= 10^4
0 <= ith start time < 100
ith start time < ith end time <= 100

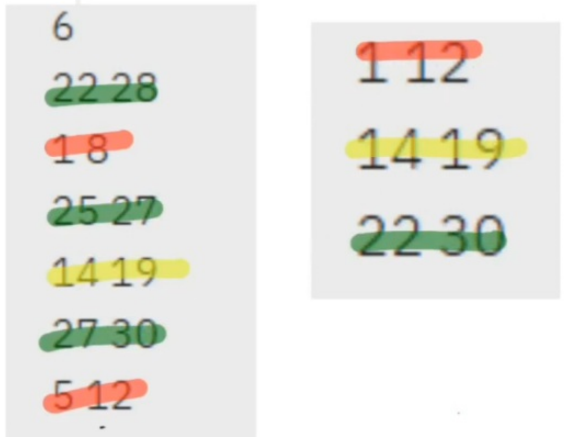
Sample Input

```
6
22 28
1 8
25 27
14 19
27 30
5 12
```

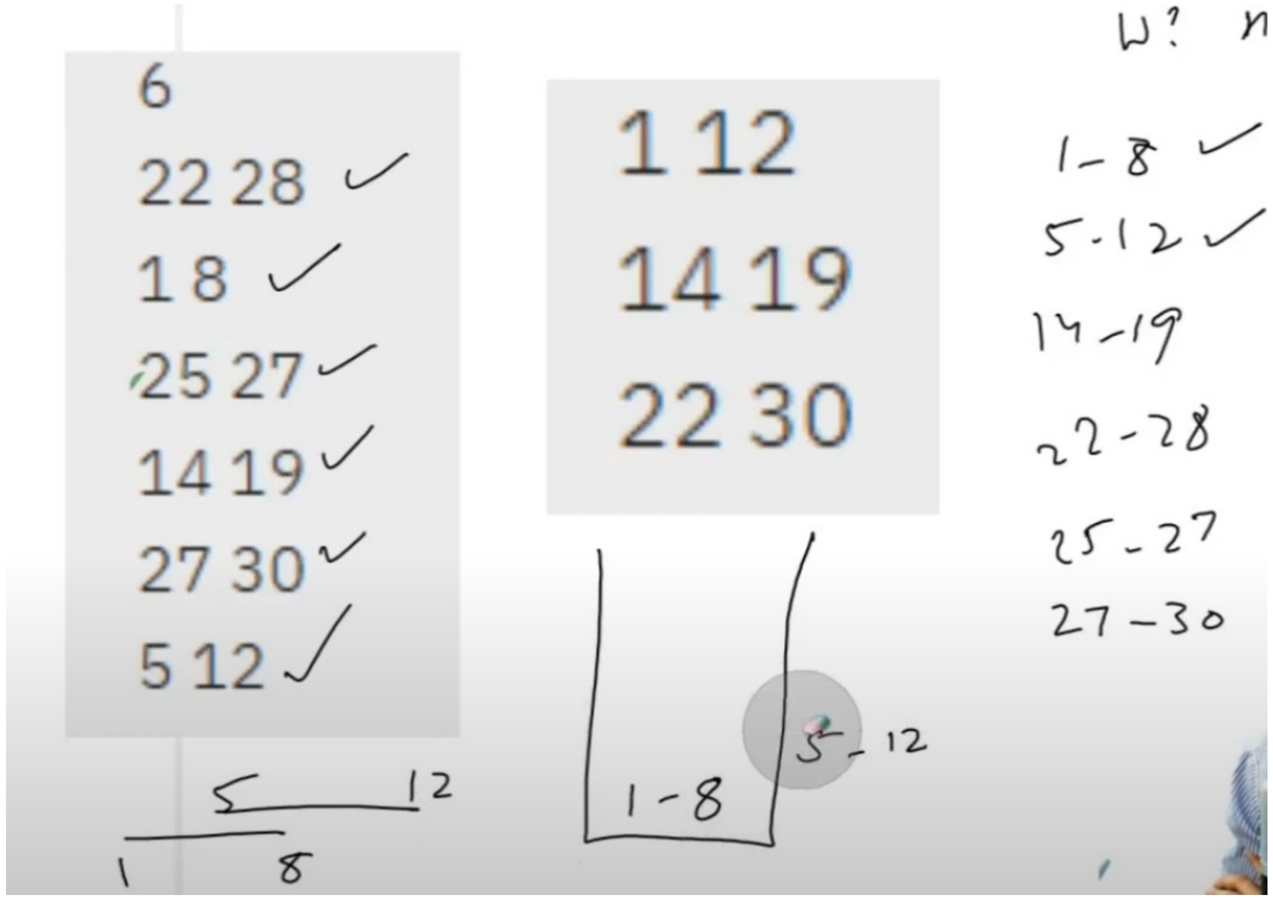
Sample Output

```
1 12
14 19
22 30
```

Need to do in NlogN


6
22 28
1 8
25 27
14 19
27 30
5 12
1 12
14 19
22 30

we need to merge meeting



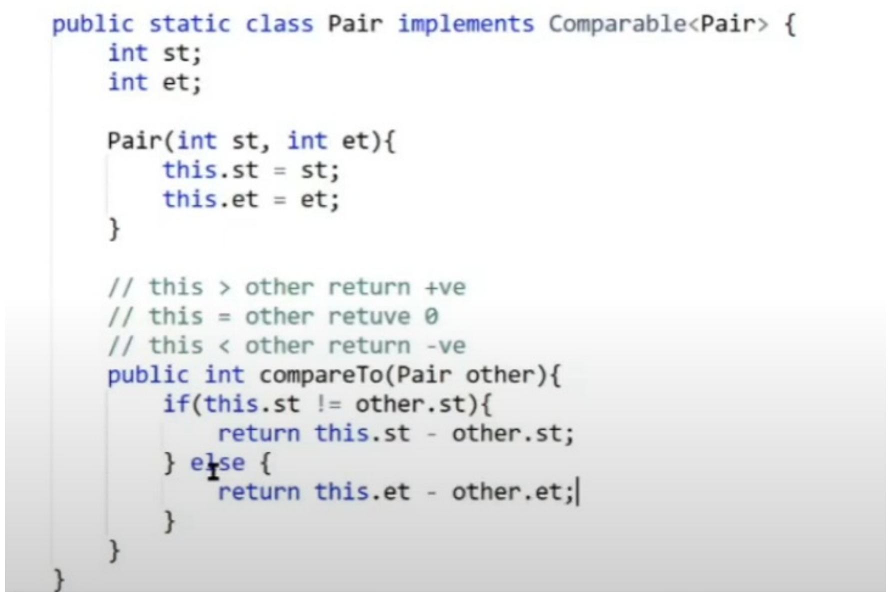
Approach

1.sort by start time. ifsort time is same then sort by end time

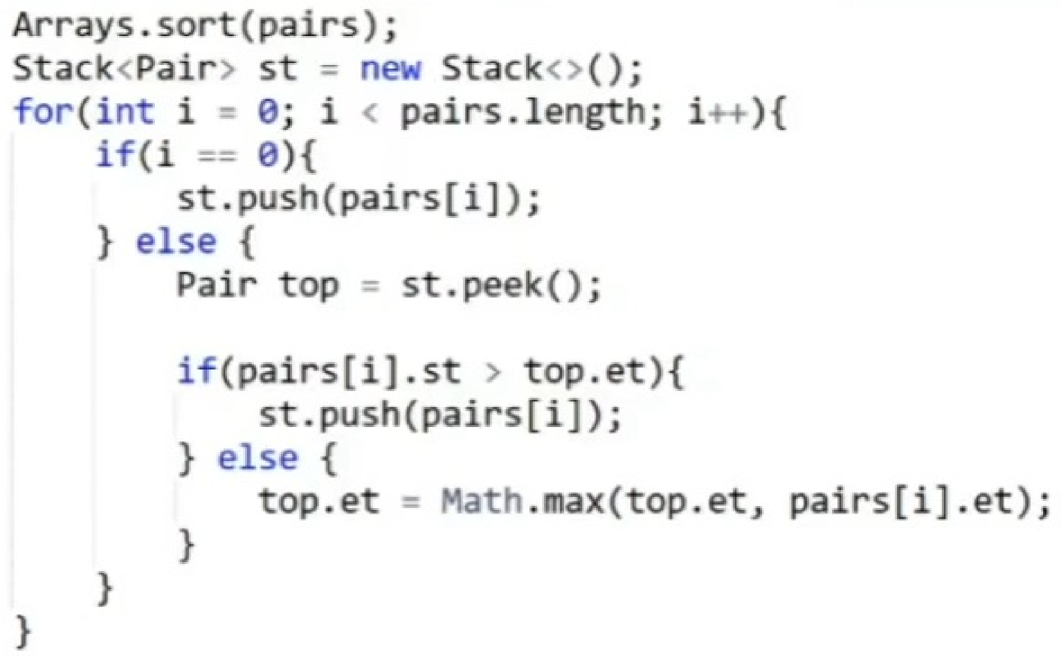
2.push first pair

3. 2nd pair is to be decided whether to be pushed to merged.

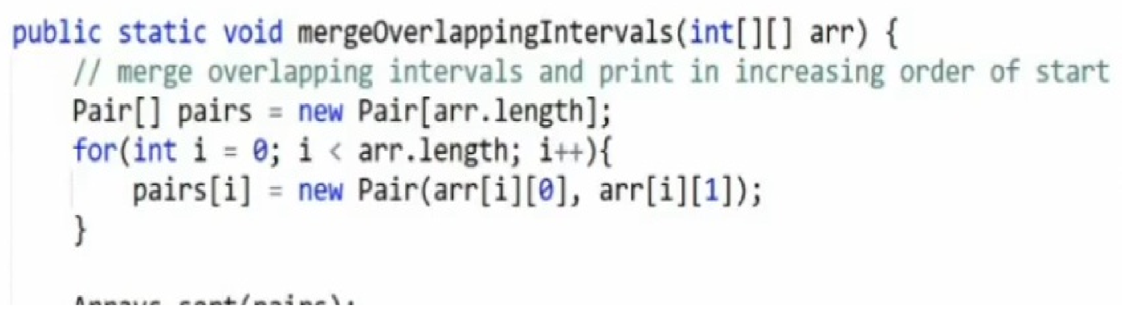
lets say
for (5 -12), we can either push or merge
how to decide?
st time of current pair < top ka end time
so merge
how to merge?
by updating end time of top
But what if
end time of curr element < top.endtime
then we need to take the
max(curr.end_time. , top.end_time)



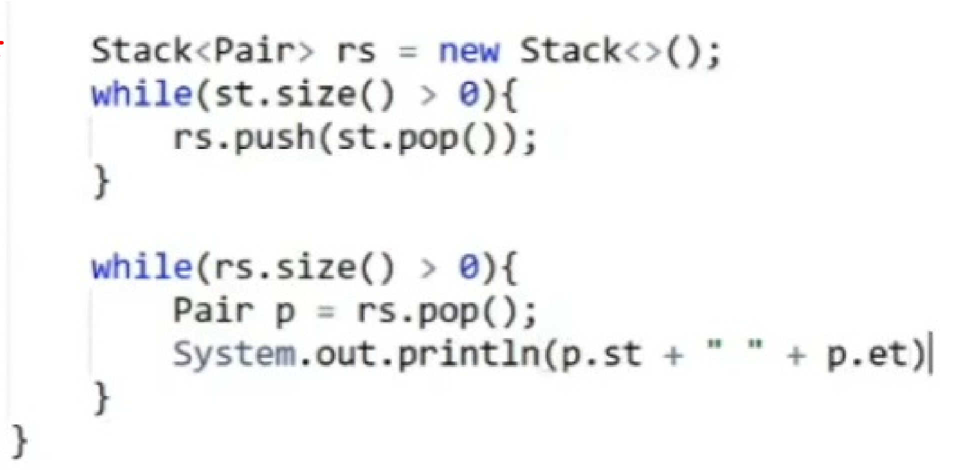
```java
public static class Pair implements Comparable<Pair> {
    int st;
    int et;

    Pair(int st, int et){
        this.st = st;
        this.et = et;
    }

    // this > other return +ve
    // this = other retuve 0
    // this < other return -ve
    public int compareTo(Pair other){
        if(this.st != other.st){
            return this.st - other.st;
        } else {
            return this.et - other.et;
        }
    }
}
```



```java
Arrays.sort(pairs);
Stack<Pair> st = new Stack<>();
for(int i = 0; i < pairs.length; i++){
    if(i == 0){
        st.push(pairs[i]);
    } else {
        Pair top = st.peek();

        if(pairs[i].st > top.et){
            st.push(pairs[i]);
        } else {
            top.et = Math.max(top.et, pairs[i].et);
        }
    }
}
```

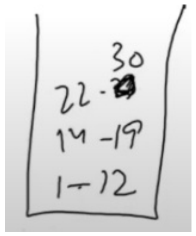


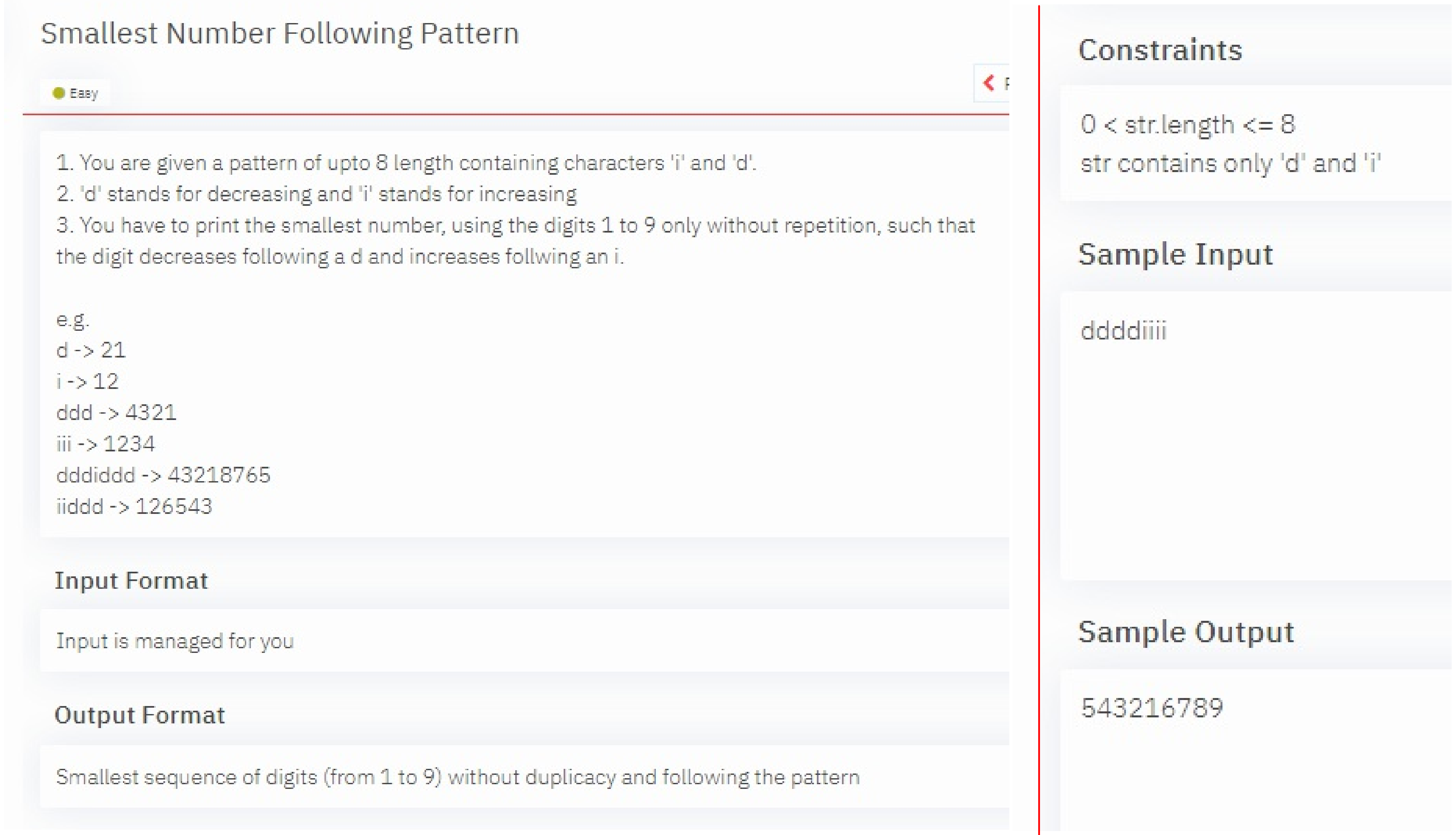
```java
public static void mergeOverlappingIntervals(int[][] arr) {
    // merge overlapping intervals and print in increasing order of start
    Pair[] pairs = new Pair[arr.length];
    for(int i = 0; i < arr.length; i++){
        pairs[i] = new Pair(arr[i][0], arr[i][1]);
    }

    Arrays.sort(pairs);
```



```java
    Stack<Pair> rs = new Stack<>();
    while(st.size() > 0){
        rs.push(st.pop());
    }

    while(rs.size() > 0){
        Pair p = rs.pop();
        System.out.println(p.st + " " + p.et)
    }
}
```

at this point we have, merged meeting in stack but we need element in asc order of start time. So we push element inseparate stack and print them





Smallest Number Following Pattern

Easy

1. You are given a pattern of upto 8 length containing characters 'i' and 'd'.
2. 'd' stands for decreasing and 'i' stands for increasing
3. You have to print the smallest number, using the digits 1 to 9 only without repetition, such that the digit decreases following a d and increases follwing an i.

e.g.
d -> 21
i -> 12
ddd -> 4321
iii -> 1234
dddiddd -> 43218765
iiddd -> 126543

Input Format

Input is managed for you

Output Format

Smallest sequence of digits (from 1 to 9) without duplicacy and following the pattern

Constraints

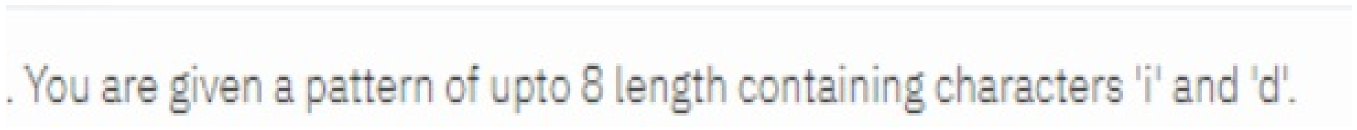
0 < str.length <= 8
str contains only 'd' and 'i'

Sample Input

```
ddddiiii
```

Sample Output

```
543216789
```



. You are given a pattern of upto 8 length containing characters 'i' and 'd'.

Rules are

1. Using the digits 1 to 9 only
2. No repetition
3. You have to print the smallest number
4. digit decreases following a d and increases follwing an i.

Approach:

Always start with smallest problem.