DAY 135.

DSA

DATE

STACKS AND QUEUES

CONTINUED.....

Date → 11 Feb, 2022.

Day → Friday

Stacks

1. Stack एक linear data structure है जिसमे सभी operations (element adding & removal) एक particular order के according होते हैं।

The order can be LIFO (Last IN First Out).
or FILO (First IN Last Out).

Stack is a linear data structure which follows a FI (First IN Last Out) or LIFO (Last IN First Out) for various operations.

→ What are these linear data structure?

Data Structure where the elements are arranged sequentially or linearly is called a Linear Data Structure. In a linear data structure, we can travel all the elements in a single pass. (मतलब memory में एक साथ store होते हैं).

Array
Arraylist
String

} These are the linear data structure.

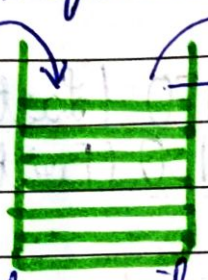
DATE

2. Stack memory में एक साथ सारे element store करता है जैसे कि array भी memory में एक साथ store करता है सभी अपने elements को

Stack mainly 3 ही operation allow करता है (element को add करना, remove करना, get करना).

Stack में ये तीनों operation एक ही end पर कर सकते हैं

add के लिए जो function है → push()
remove के लिए जो function है → pop()
get के लिए जो function है → peek()



In Stack, the Insertion & deletion happens on the same end.

Stack Overflow can occur when there is no more heap memory available to be allocated to our stack

* push() → Adds an item in the stack. If the stack is full, then this condition is said Stack Overflow.

In Underflow Trying to remove an element from empty stack can give us a run time error

* pop() → removes an item from the stack. The items are popped (removed) in an reversed order in which they are pushed.
If the stack is empty, then it is said to be underflow condition

* peek() → returns the top element of the stack

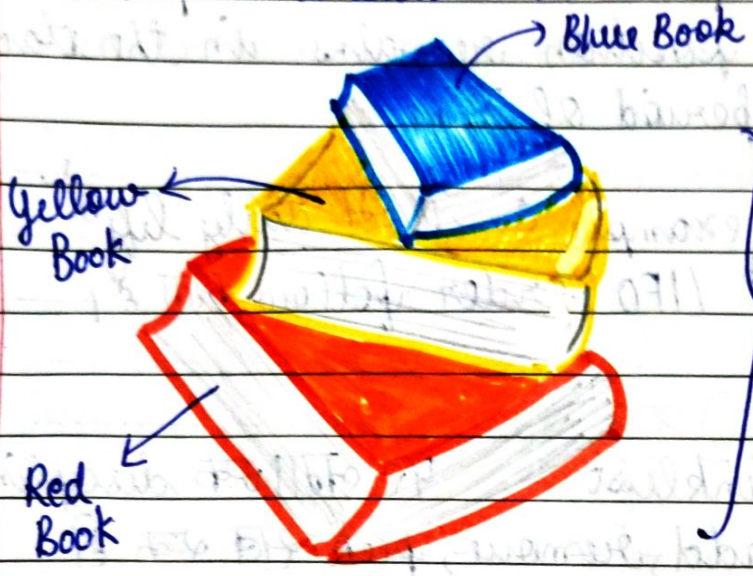
* isEmpty() → returns true if the stack is empty, else return false.

* The Time Complexities of operations on stack →
push(), pop(), isEmpty() & peek() all take O(1) time.
We do not run any loop in any of these operation.

Understanding FILO (First In Last Out)

Real Life Example (Eg 1)

→ वैसे तो हमारे पास बहुत सारे real life example है Stack के, लेकिन FILO का example समझने के लिए हम books को consider करेंगे



→ अगर हमें top वाली blue book लेनी है तो हमे कुछ नहीं करना होगा, We can simply pick it from the top.

→ लेकिन अगर हमें सबसे नीचे रखी (bottom) Red Book को लेना है, तो हमे पहले ऊपर रखी Yellow और Blue Book को निकालना होगा, तब ही हम नीचे रखी Red Book को ले सकते हैं।

→ और अगर हमें एक book को और add करना है lets say green book तो हम उसे Blue Book के ऊपर (top पर ही) add कर सकते हैं।

This type of nature is known as LIFO (Last IN First Out) as you can first get the last item added in the stack.
Also, you will get the first item inserted in the stack at last (as it becomes the bottom-most element), & this order is known as the FILO (First IN Last Out) order.

Real Life Example 2

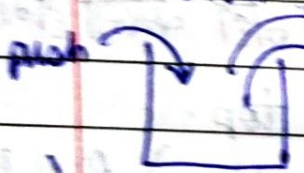
DATE

We can consider example of plates stacked over one another - in a canteen
The plate which is at the top is 1st first one to be removed i.e the plate which has been placed at the bottommost position remains in the stack for the longest period of time.

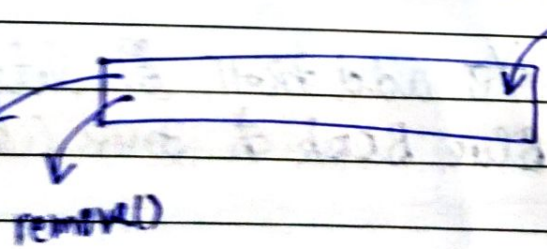
तो, ये भी एक example है जो हम daily life में देखते हैं जो कि LIFO order follow करता है।

Queue और

* Stack basically list पर लगा एक discipline है जिसमें add, remove, peek सब एक ही end पर होंगे (Top पर ही add होगा, remove होंगे, peek भी top से ही होगा)



Queue में Discipline होता है add होगा end में और front से remove होगा।



होगा तो ऐसा ही होगा वरना नहीं होगा। इसलिए बोलते Stack और Queue disciplined arraylist है

DATE

Application/Usage of Stack

1. Redo - Undo feature in many places like editors, photoshop.
2. Forward and backward feature in Web browsers.
3. Used in many algorithms
   - Towers of Hanoi,
   - Tree Traversal (Pre Order, In Order, Post Order)
4. Graph algorithm like DFS (Depth First Search), Topological Sorting, Connected Component, etc.

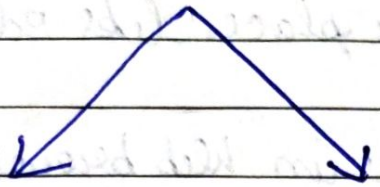
5. Backtracking is one of the algorithm designing techniques.
   Some example of Backtracking are Knight-Tour Problem, N-Queen Problem, Find Path through Maze.
   In all these problems we move into someway, if that way is not efficient then we come back to previous state and go into some another path.
   To get back from current state, we need to store the previous state for that purpose, we need a stack.

6) Infix to Prefix / Postfix Conversion

Implementation of Stack

DATE

* The Implementation of a Stack can be done in 2 ways



Using array Using linked list.

Implementing Stack using Arrays.

Pros
1) Easy to implement
2) Memory is saved as pointers involved are not (ई element की next element की address save नहीं करनी पड़ती तो space की use होती है)

Cons
3) It is not dynamic. (array की size increase या decrease नहीं कर सकते).

It does not grow or shrink depending on need at runtime.

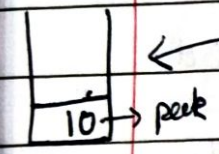
Implementing Stack using Linked List

Pros
1) The linked list implementation of stack can grow & shrink according to the needs at runtime

Cons
2) It requires extra memory due to involvement of pointers. (हर element (node) की next element (node) की address save करनी पड़ती है)

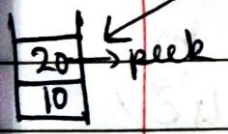
Example-1

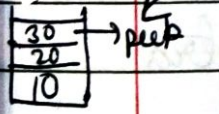
```
import java.util.*
public class Stack
{
  public static void main (String[] args)
  {
    Scanner scan = new Scanner(System.in);
    Stack stack = new Stack <>();
    stack.push(10);
    System.out.println(stack.peek());

    stack.push(20);
    System.out.println(stack.peek());
    stack.push(30);
    stack.push(50);
    stack.push(60);
    while(stack.isEmpty() == false)
    {
      System.out.println(stack.peek());
      stack.pop();
    }
  }
}
```



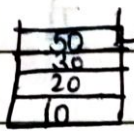
हमने एक stack बनाया

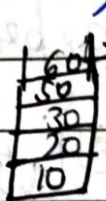
10





20





जब तक stack empty न हो जाए तब तक ये लूप चलेगा

Ques. 1

DUPLICATE BRACKETS

DATE

हमे एक math expression given होगा

हमे एक stringexpression given होगा that represents an expression.

* हम assume करते है कि expression एक balanced expression है,
  (मतलब हर opening bracket के लिए एक closing bracket given है).
* अब हमे ये detect करना होगा कि क्या इस expression मे कोई needless/ waste brackets है?

हमे true print करना होगा if we detect extra brackets and we need to print false otherwise

eg → ((a+b) + (c+d))

There is no waste / redundant / needless bracket found.

So, we need to print false in this case

c+d belong to Pair 2.

Pair1 Pair2

DATE

(a+b) + ((c+d)) → true

This pair of Bracket is useless This is Pair3

हर Pair के अंदर atleast एक character होना चाहिए that doesn't belong to any other pair

So, there is no character that belong to Pair 3. So Pair 3 of brackets is useless.

So, if there is no character that specifically belong to the Pair, it means the pair is useless.

Approach

1. हम stack data structure का use करेंगे to check. if there is a character (atleast 1 character) between each pair of bracket.

2. सबसे पहले हम एक empty stack को initialize करेंगे।

```
Stack stack = new Stack<>();
```

3 फिर हम पूरी Expression String के हर character पर traverse करेंगे (using the (i)).

(i) → this i is the index of character in the String

a) अगर character at ith index

str.charAt(i) ≠ ')'

→ Push the character into Stack.

अगर character एक closing bracket नहीं है तो उस character को stack में डालो.

b) लेकिन अगर str.charAt(i) == ')'

↓

अगर character at ith index पर closing bracket है

→ check if the character at peek(top) of stack is '(' or not

↳ opening bracket

अगर stack.peek() == '(' तो return true

→ If the character at peek of stack is not "(" तो इसका मतलब है कि there is atleast 1 character b/w pairs of bracket.

Pop all the characters from stack until you get '(' opening bracket at top.
Then pop that opening bracket too.

→ If all the characters are traversed and no duplicate bracket is found, then return false.

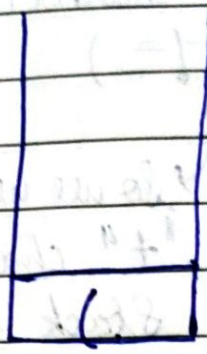
↳ अगर हम सभी characters पर traverse कर चुके हैं और अब stack empty है मतलब हमें कोई Duplicate Bracket नहीं मिला तो हम simply false return करवेंगे।

Eg 1 ((a)+b)
0 1 2 3 4 5 6

str.charAt(i) != ")"

DATE

1) → i = 0

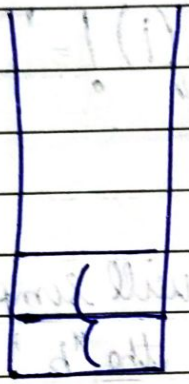


"(" ≠ ")"

↳ तो हम simply इस character को stack में push कर देंगे।

2) i = 1

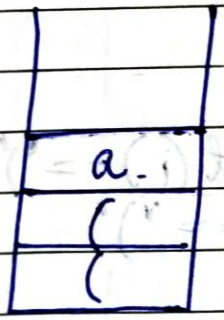
str.charAt(i) != ")"



"(" ≠ ")"

↳ तो हम simply इस character को stack में push कर देंगे।

3) i = 2

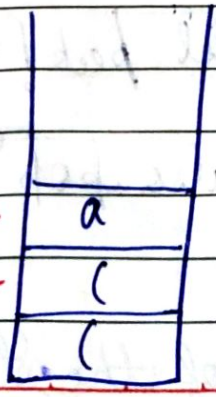


str.charAt(i) != "a"

"(" ≠ a

↳ तो हम simply इस character को stack में push कर देंगे।

4) i = 3



remove ←

remove ←

str.charAt(i) == ")"

")" == ")"

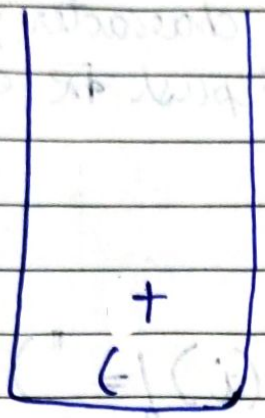
→ So, we will check if stack.peek == "("

stack.peek = a ⇒ a ≠ "("

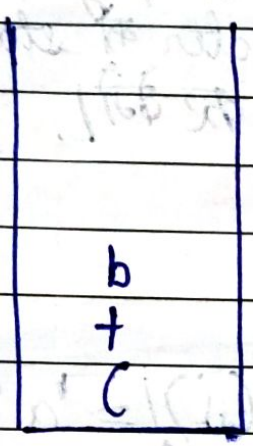
we will remove element from stack unless we get "(" & then remove "("

4. i=4. str.charAt(i) != ")"
+ !=).

↳ So we will simply push "+" character into the Stack.



5. i=5 str.charAt(i) != ")"
b != ")"



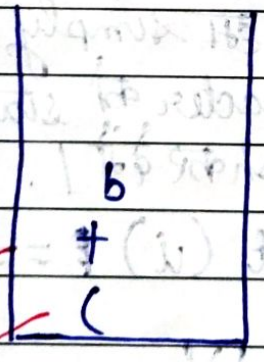
↳ So we will simply push the "b" character into Stack.

6. i=6. str.charAt(i) == ")"
")" == ")"

Stack.peek() != "("

So Pop until peek != "("

and also pop "("



remove
remove
remove.

All characters traversed of the String and no duplicate pair found, so we can return false.

Eg-2 ((a + b))
0 1 2 3 4 5 6

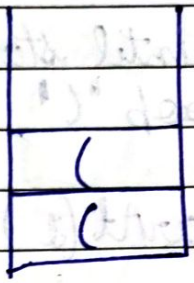
DATE

1 i = 0.

str.charAt(i) != ")"
"(" != ")"

So we can simply push "(" character into the Stack.

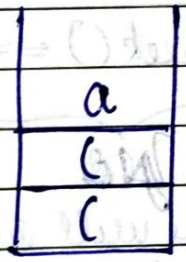
2. i = 1.



str.charAt(i) != ")"
"(" != ")"

So, we can simply push "(" character into the stack.

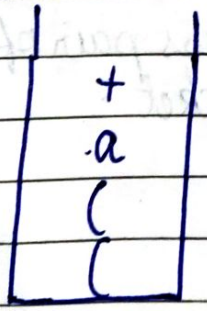
3. i = 2.



str.charAt(i) != ")"
"a" != ")"

So, we can push "a" into the Stack

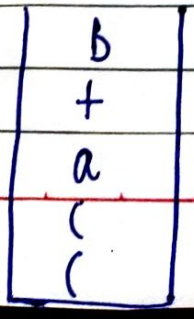
4. i = 3



str.charAt(i) != ")"
"+" != ")"

So, we can just push "+" into the Stack.

5. i = 4.



str.charAt(i) != ")"
"b" != ")"

So, we can just push "b" into stack.

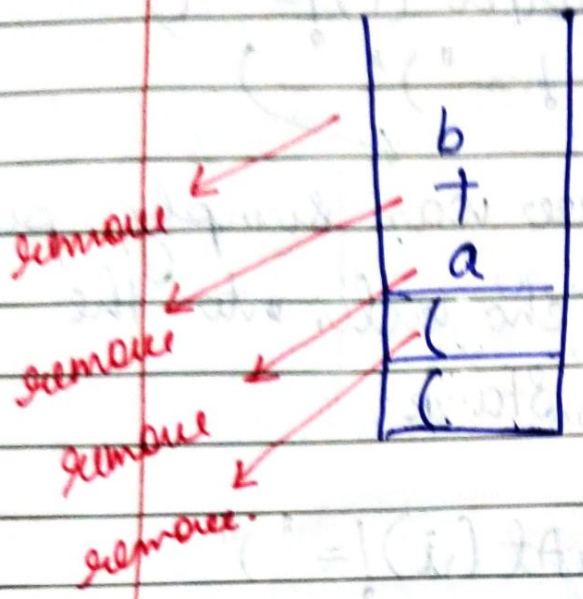
PAGE

DATE

6. i = 5

str.charAt(i) == ")"

")" == ")"



So, we will first check is stack.peek() == "("

NO.

So pop until stack.peek == "(" and then pop "("

7. i = 6.

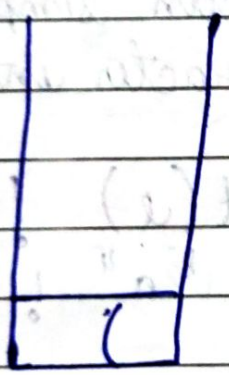
str.charAt(i) == ")"

")" == ")"

So, will first check is stack.peek() == "("

Yes.

we will simply return true Because this is a needless pair of Bracket.



<code>

```java
public class Main
{
  public static void main (String [] args)
  {
    Scanner scan = new Scanner(System.in);
    String str = scan.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.size()>0 && st.peek()!='(')
          {
            st.pop();
          }
          st.pop();
        }
      }
      else {
        st.push(ch);
      }
    }
  }
  System.out.println(false);
```

- `String str = scan.nextLine();` → string input ली
- `new Stack<>();` → एक stack बनाया
- `for(...)` → सभी string के character पर loop लगाया
- `if(ch==')')` → अगर ith index वाला character closing bracket है और stack के peek पर opening bracket है तो ~~true~~ print करदो because there is a needless bracket present
- `else` → या अगर stack के peek पर opening bracket नहीं है तो तब तक pop करो जब तक opening Bracket ना मिल जाए
- `st.pop();` → और फिर opening Bracket को भी pop करदो
- `else { st.push(ch); }` → और अगर ith character opening Bracket नहीं है तो उसे Stack में push करदो।

The time complexities is O(n). Even though we have a while loop inside for loop.

a while loop is nested inside a for loop.

Overall अगर देखा जाए तो

हम n character को stack में push कर रहे हैं → O(n)

और n characters को ही Stack से remove कर रहे हैं (Pair by Pair) → O(n)

So Overall T.C = O(n) + O(n)

$\approx 2O(n)$

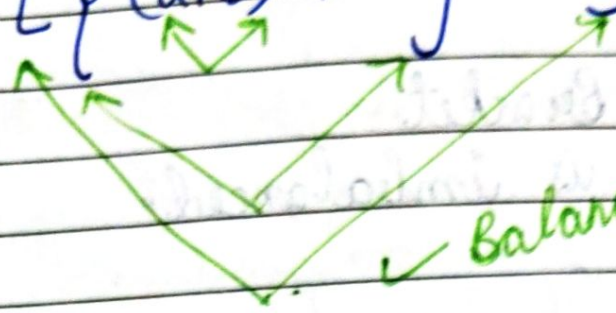
T.C $\approx O(n)$

Ques 2

Balanced Brackets

DATE

eg. 1) [{ (a + b) + c } + d]

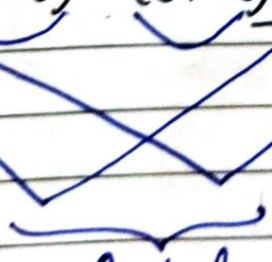


हर 1 opening Bracket के लिए 1 closing Bracket होना चाहिए

* An expression will be having unbalanced brackets if the order of Brackets do not match, or if the opening Brackets are more than the closing Bracket or if the closing Brackets are more than the opening Brackets

अगर ऐसे expression में unbalanced brackets होंगे, then we need to print true else we need to print false.

eg 2). [{ (a + b) + (c * d)] * e } / f



अगर { curly bracket में square bracket open hua तो
] square Bracet के अंदर ही
} curly Bracket close होना चाहिए

eg 3) $(a+b)]$ → not Balanced, extra closing Bracit

DATE

इस closing Bracet ']' के लिए कोई corresponding opening bracket है ही नहीं।

∴ Extra Closing Bracket.
∴ The expression is imbalanced.

eg.4 $[(a+b)$ → not balanced, extra opening bracket.

इस opening bracket '[' के corresponding कोई closing Bracet ']' नहीं है।

∴ Extra opening Bracet.

So the expression is imbalanced.

We should only focus when we get an opening/closing bracket. We can just ignore other characters

DATE

Approach

eg → [{(a+b)+c}]

① सबसे पहले एक stack बनालो

② पूरी string पर traverse करो loop लगाके

'[' or '{' or '('

अगर इनमे से कोई opening Bracket है character तो उस character को stack मे push करदो

③ ']' or '}' or ')'

और अगर इनमे से कोई closing Bracket है

तो → a) पहले check करो कि stack.peek पर इस character का corresponding opening Bracket है या नहीं

']' के लिए '['
'}' के लिए '{'
')' के लिए '('

अगर corresponding opening Bracket मिलजाए stack की peek पर तो उस opening bracket को stack से pop करदो

Do this unless we don't traverse on each character of string if we don't see any imbalanced bracket

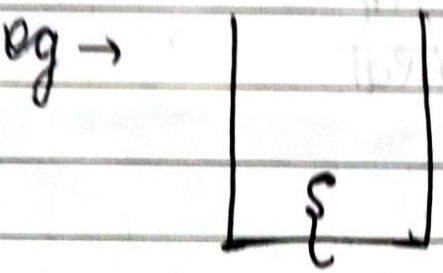
b) अगर हमे corresponding opening bracket stack के top पर नहीं मिला, it means the expression is imbalanced So simply return false.

④ अगर stack empty है, और हमे एक closing bracket as our next character मिला in string तो इसका मतलब है कि हमारे पास extra closing bracket है, it means the expression is unbalanced so return false.

eg → a+b)

DATE

(5) और हमने पूरी String को traverse कर लिया,
But still our string is not empty,
it still has an opening Bracket, it means there is an extra opening bracket in our expression.
It means, the expression is imbalanced.
So, we can simply return false.



Eg → { { a + b }

we have traversed the string completely

The pointer is not pointing to any index of character string

But still the stack is not empty yet.

Imbalanced.

(6) और Stack भी empty है और हमने पूरी String भी traverse कर ली है तो इसका means the expression is balanced.
So, we can simply return true

Time Complexity

↳ The time complexity of this algorithm is $O(n)$ as we are traversing a string of length n.

We push & pop only the brackets in Stack, so there are some constant K brackets & push() & pop() has time complexity of $O(1)$ so $k * O(1)$.
$k(O(1)) \cong O(1)$ ↓ bracket

Space Complexity

↳ The space complexity is $O(n)$.
We are continuously pushing & popping element from stack.

फिर भी $O(n)$ space?

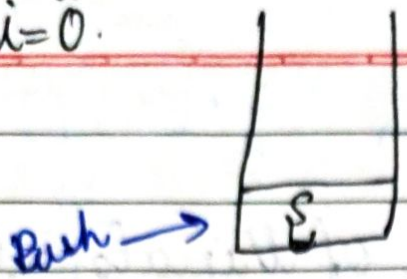
Yes, because the maximum size of stack can be equal to the length of String.
(Expression में सारे characters ही opening Brackets भी हो सकते हैं और पूरी String traverse करने के बाद हमें पता लगेगा कि Stack is not empty yet, so the Brackets are not Balanced.

Eg-1). ① {[(a+b)]}

i=0

{

It is an opening Bracket so, we can just push it into the stack

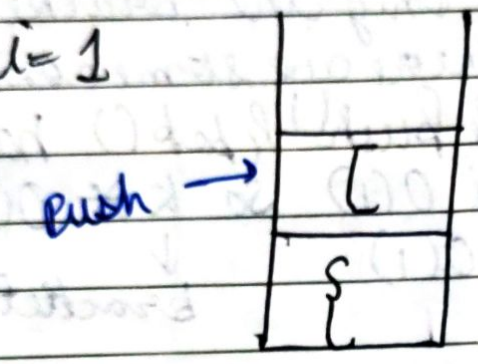


② {[(a+b)]}

i=1

[

↳ It is an opening Bracket. So we can just push it

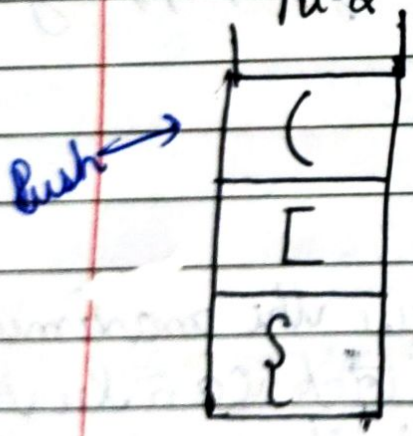


③ {[(a+b)]}

i=2

(

↳ It is an opening Bracket so we can simply push into the stack.



④ {[(a+b)]}

i=3

a

↳ It is neither an opening bracket, nor closing bracket so, we can simply ignore it

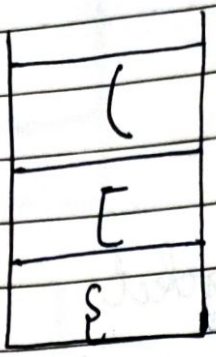
DATE

⑤ {[(a+b)]}

i=4

+

↳ Ignore (Because it is neither a opening Bracket nor a closing Bracket)



⑥ {[(a+b)]}

i=5

b

↳ Ignore (Neither an opening Bracket, nor a closing Bracket)

⑦ {[(a+b)]}

i=6

)

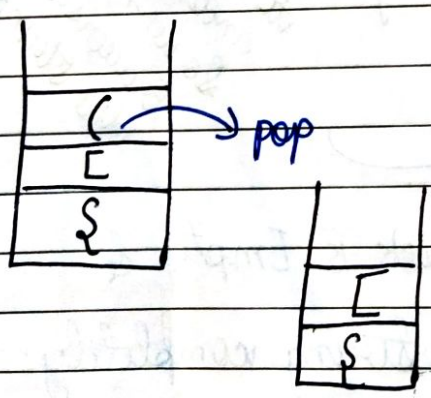
↳ It is an closing Bracket.

So now check if the Stack.peek() == (

↙ ↘

Yes corresponding bracket of)

So, pop it from the stack



⑧ {[(a+b)]}

i=7

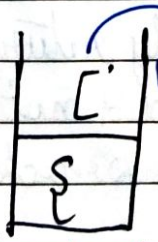
']'

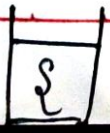
↳ It is a closing Bracket

Check whether the stack.peek() is corresponding to ']' or not

stack.peek() == '[' so pop it from stack

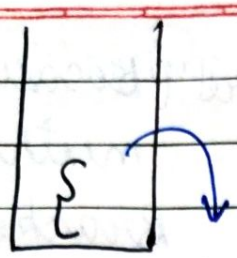
Yes.





(9) {[(a+b)]}

DATE



i = 8

'}'

pop

↳ It is a closing Bracket.

So we will check if the peek of stack is the corresponding Bracket of '}' or not

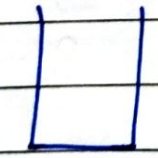
stack.peek() == '{'

↳ Yes, it is the corresponding Bracket to '}'

So, we can simply pop it out from the stack.

{[(a+b)]}

The pointer is now not pointing to any index so we have traversed the string



Now, The Stack is Empty & we have traversed the string completely.

So, we can simply return true because we did not find any imbalanced Bracket

Eg 2). (a+(b+c)+d.

① i=0 (

↳ It is an opening Bracket So push it into the Stack.

② i=1. a

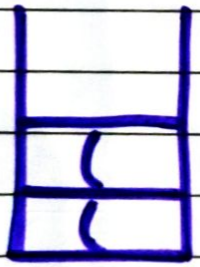
↳ Ignore.

③ i=2. +

↳ Ignore.

④ i=3 (

↳ It is an opening bracket so push it into the Stack



⑤ i=4 b

↳ Ignore.

⑥ i=5 +

↳ Ignore.

⑦ i=7)

↳ Ignore.

(8) $i = 8$.

↳ It is a closing bracket.

So, we can check if stack.peek == '('

↳ yes

This is the corresponding bracket of ')'

pop.

So we need to pop the bracket at peek of stack. (()

(9)

$i = 9$ $(a+(b+c)+d$.

$i = 9$.

\+

↳ Ignore.

(10). $i = 10$.

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

$i = 10$.

d

↳ Ignore

(11) Now, we have traversed the string completely

But the stack is yet not empty

So, we can see that there was no corresponding closing Bracket from this opening Bracket

So, The Expression was not Balanced

because the number of opening brackets were more than the number of closing brackets

So, the Expression was imbalanced.

So, we can simply return false.

How to code?

* first of all, declare the stack

1. Iterate through the String

(एक for loop लगाके पूरी string पर traverse करना होगा)

→ Push any opening bracket whether '(' or '[' or '{' whenever you encounter it

जब भी हमें string में traverse करते हुए (/[/{ (opening Bracket मिले) on ith index of String, we will simply push the opening Bracket into the Stack.

→ जब भी हमें ')' or ']' or '}' कोई closing Bracket मिले on ith index of String while iterating on String

→ See if this closing bracket matches with the corresponding opening Bracket on top of Stack.

If it matches, pop the opening bracket & print false & break.

2. If at any point, you encounter a closing bracket and there is no element to pop i.e Stack is empty, print false, and break

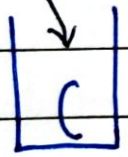
eg → a+b)

It means that the number of closing bracket is more as compared to number of opening Bracket

Here, no. of opening Brackets were 0
no. of closing Brackets were 1.

3. उपर दी हुई String पर traverse कर चुके हैं and there was no mismatch found but the Stack is still not empty. It means the number of opening brackets is more than the number of Closing Brackets

eg → ((a+b) + (c+d) + g



So, print false & break.

4. If you have iterated the entire string and there was no mismatch found (उपर की तीनों unbalanced Bracket वाली condition). और Stack भी empty है, It means that the Bracket were matching & the number of opening and closing bracket was equal. So, print true.

```
import java.util.*;
public class BalancedBrackets
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        String str = scan.nextLine();
        Stack<Character> stack = new Stack<>();
        for(int i=0; i<str.length(); i++)
        {
            char ch = str.charAt(i);
            if(ch=='(' || ch=='{' || ch=='[')
            {
                stack.push(ch);
            }
            else if(ch==')' || ch=='}' || ch==']')
            {
                if(stack.isEmpty()==false &&
                   ch==')')
                {
                    if(stack.peek()!='(')
                    {
                        System.out.println("false");
                        return;
                    }
                    else{stack.pop();}
                }
            }
        }
    }
}
```

DATE

PAGE

→ अगर stack empty नहीं है

अगर ch अगर] है

DATE

```
else if (stack.isEmpty==false && ch=="]")
{
  if(stack.peek()!='[')
  {
    System.out.println(false);
    return;
  }
  else
  {
    stack.pop();
  }
}
```

अगर stack के top पर '[' नहीं है तो false print करदो और return करदो।

stack से से '[' को remove करदो as it is the top of stack

```
else if (stack.isEmpty==false && ch=='}')
  {
    if (stack.peek()!='{')
    {
      System.out.println(false);
      return;
    }
    else
    {
      stack.pop();
    }
  }
```

अगर stack empty और stack के ch अगर } है

↳ अगर stack के top पर अगर { नहीं है

तो false print करदो & return

अगर } है stack का top pop करदो

हमने पूरी string पर traverse नहीं किया

```
else if (stack.isEmpty == true)
  {
    System.out.println(false);
    return;
  }
} → The for loop ends here
System.out.println();
} → main function ends here
} → The class BalancedBrackets ends here.
```

so we can ↓

✓ ch as our }|)|] we have और अगर stack empty हो गई (opening bracket नहीं है). It means extra }|)|] closing bracket है।

* Tree में T.C पता करने के लिए उसके code को नहीं देखना चाहिए

DATE

Tree के code में TC Euler Tree को देखने पर पता चला चलती है

अगर Pre Order/Post Order/In Order में कोई काम नहीं हो रहा O(1).

∴ $O(n)$ = T.C of Tree

अगर Pre Order/Post Order/In Order में O(n) काम हो रहा है तो

$T.C = O(n^2)$

हर node के children पर अगर काम हो रहा है

$O(c) = 2n-1$

children

$O(2n-1) = T.C.$

* Stack में T.C पता करने के लिए Stack को देखना चाहिए code Dry run करते हुए

अगर हर element ~~code~~ stack में एक बार add होकर एक ही बार pop होता है तो T.C होगी O(n)

Ques - 3

Next Greater Element to the Right

DATE

हमे एक array given होगा

2	5	9	3	1	12	6	8	7

हमे array के हर number के लिए

उसकी Right Side में पहला बड़ा element बताना है।

अगर Right Side में कोई बड़ा element है ही नहीं, तो ... -1.

2 5 9 3 1 12 6 8 7

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

हम एक array given होगा

1. हम एक answer array बनायेंगे जिसमे हम हर element का Next Greater Element To Right store करेंगे और हम एक Stack बनायेंगे
2. हम given array पर reverse order में traverse करेंगे for loop लगा के

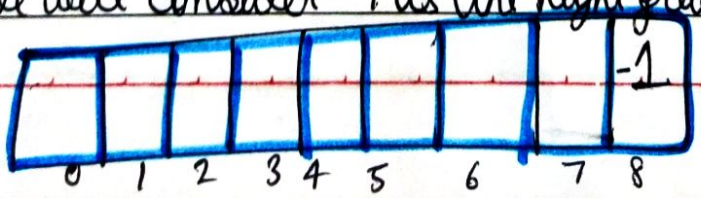
for eg → 2 5 9 3 1 12 6 8 7

i = 8.

(Size of array) - 1

→ Push the last element into Stack

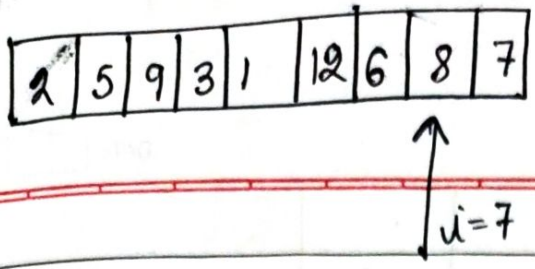
अब (last element) 7 के Right में कोई greater element नहीं है तो we will consider -1 as the Right greater element for last element.



PAGE

3.

2	5	9	3	1	12	6	8	7



DATE

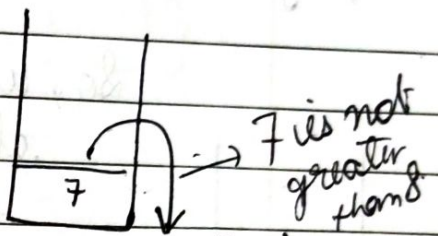
} not greater than 8.

Now we move to the last second element

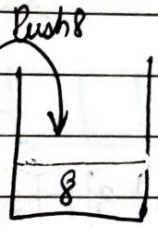
अब हम Stack में से element pop करायेंगे जब तक 8 (with element) से बड़ा कोई element न मिले, And then Push 8 into the stack

So $7 < 8$ तो 7 को Pop करदो

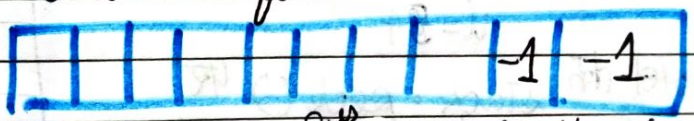
2	5	9	3	1	12	6	8	7



So stack is empty
और हमें 8 से बड़ा कोई element नहीं मिला. So, this means 8 से Right में कोई greater element है ही नहीं.

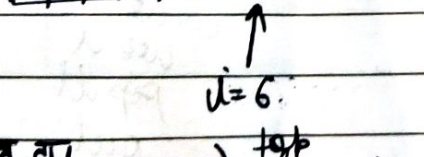


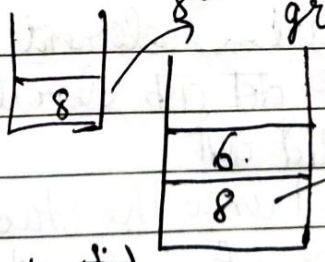
So, we will consider -1 for 8 too.



4.

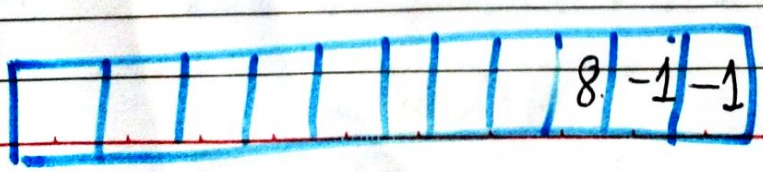
2	5	9	3	1	12	6	8	7





* अब हम Stack में top element check करेंगे, जब तक Stack में greater element न मिल जाए, तब तक stack में से element को pop कराओ. Put 6 in the stack.

So, : The greater element to Right for 6 is 8.



PAGE

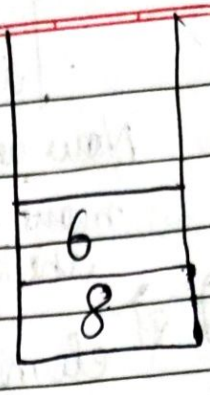
5.

2	5	9	3	1	12	6	8	7

i=5

DATE

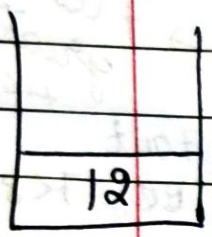
जब तक 12 से बड़ा कोई element न मिल जाए, तब तक pop all elements out from the stack.



Not greater than 12

So, we can pop 6, 8 from stack; Now, stack becomes empty, and now put 12 in the Stack

So, we can see that we have popped all elements out of stack & we did not find any element greater than 12. So -1 is considered as Greater element to Right for 12.

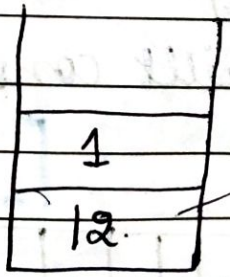


						-1	8	-1	-1

6

2	5	9	3	1	12	6	8	7

i = 4



12 is greater than 1

So we won't pop it out.

जब तक stack.peek() पर कोई 1 से greater element न मिल जाए तब तक elements को pop कराते चलो and put 1 into the Stack

12 is greater than 1.

So 12 is the greater element to Right for 1

				12	-1	8	-1	-1

7. | 2 | 5 | 9 | 3 | 1 | 12 | 6 | 8 | 7 |

DATE

i=3

1	1 lesser than 3
12	12 greater than 3.

जब तक 3 से greater element Stack की peek न हो तब तक element pop कराओ

So pop 1 out of the Stack & 12 is greater than 3

So, Push 3 into the Stack

12 is greater than 3 so 12 is the greater element to right for 3

3.	
12	12

			12	12	-1	8	-1	-1

8. | 2 | 5 | 9 | 3 | 1 | 12 | 6 | 8 | 7 |

i=2

जब तक 9 से greater element Stack की peek न हो तब तक Stack से element pop कराओ & push 9 into Stack

				12	12	12	-1	8	-1	-1

push 2 into stack

9
12

3	Smaller than 9 so pop 3 out
12	So 12 is greater element to Right for 9.

9. | 2 | 5 | 9 | 3 | 1 | 12 | 6 | 8 | 7 |

i=1. जब तक 5 से greater element Stack की peek न हो, तब तक Stack से element pop कराओ and push 5 into Stack

So 9 is greaterelement to Right for 5.

9
12

9 is greater element to right for 5.

			9	12	12	12	-1	8	-1	-1

5
9
12

Put 5 in Stack.

10

2	5	9	3	1	12	6	8	7

DATE

i=0 जब तक 2 से greater element न मिल जाए तब तक Stack में से element pop कराओ

So, 5 is greater element to Right for 2.

5 is greater than 2

So,

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

This is our answer array.

Abbroach

DATE

1. सबसे पहले हमें एक answer array of same size as of gerien array, एक Stack बनाना है.
2. हर gerien array पर reverse order में traverse करेंगे and push the last element of gerienarray in the Stack.
3. अब last element के right में कोई element नहीं है, तो last element के लिए हम -1 को greater element to the right consider करेंगे, — — —
4. अब second last element पर आते हैं (& similarly on rest of elements), we check our stack, if the top most element of Stack is smaller than present element (second last element) If it is smaller than the present element, then we pop out the peek of Stack until we get an element which is greater than present element. Whenever we get the greater element, we store it in answer array at index of present element & push present elements into the Stack.
5. अब हम Stack में से element remove करते हैं, और पूरा Stack empty हो गया है & we did not get any element greater than present element (it means we did not have any element greater than present element at right) so, we store -1 corresponding to present elements index in answer array. & push present the present element into Stack.

<code>

DATE

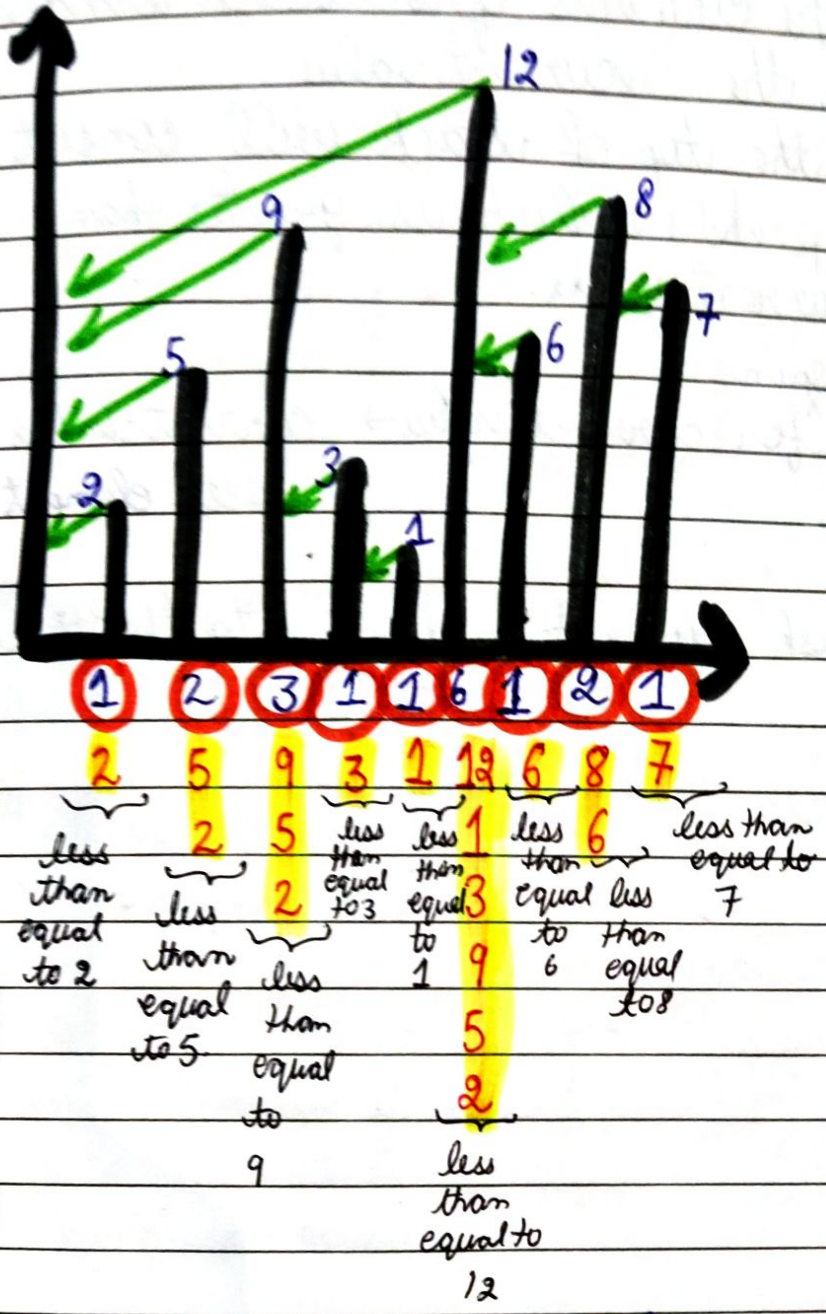
```
import java.util.*;
public class Main
{
  public static void display (int[] a)
  {
    for (int i=0; i<a.length; i++)
    { System.out.println (a[i]);
    }
  }

public static void main (String args[])
{ BufferReader br = new BufferReader (new
          InputStreamReader (System.in));
  int n= Integer.parseInt ( br.readLine ());
  int a[] = new int [n];
  for(int i = 0 ; i<n; i++).
  { a[i]= Integer.parseInt( br.readline());
  }
  int ans[] = solve (a);
  display (ans);
}
```

```
public static int[] solve(int[] arr)
{
    int ans[] = new int[arr.length];
    Stack<Integer> stack = new Stack<>();

    ans[arr.length-1] = -1;
    stack.push(arr[arr.length-1]);

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

        if(stack.isEmpty()==true)
        {
            ans[i] = -1;
        }
        else{
            ans[i] = stack.peek();
        }

        stack.push(arr[i]);
    }
    return ans;
}
```

Class-4

STOCK SPAN

DATE



हमें एक n size का array दिया है, हमें n numbers given हैं जो एक share का price बताते हैं on n different days.

* हमें n days के लिए Stock span बताना होगा।

Stock Span is defined as the number of days passed between the current day and the first day before today when the price was higher than today.

DATE

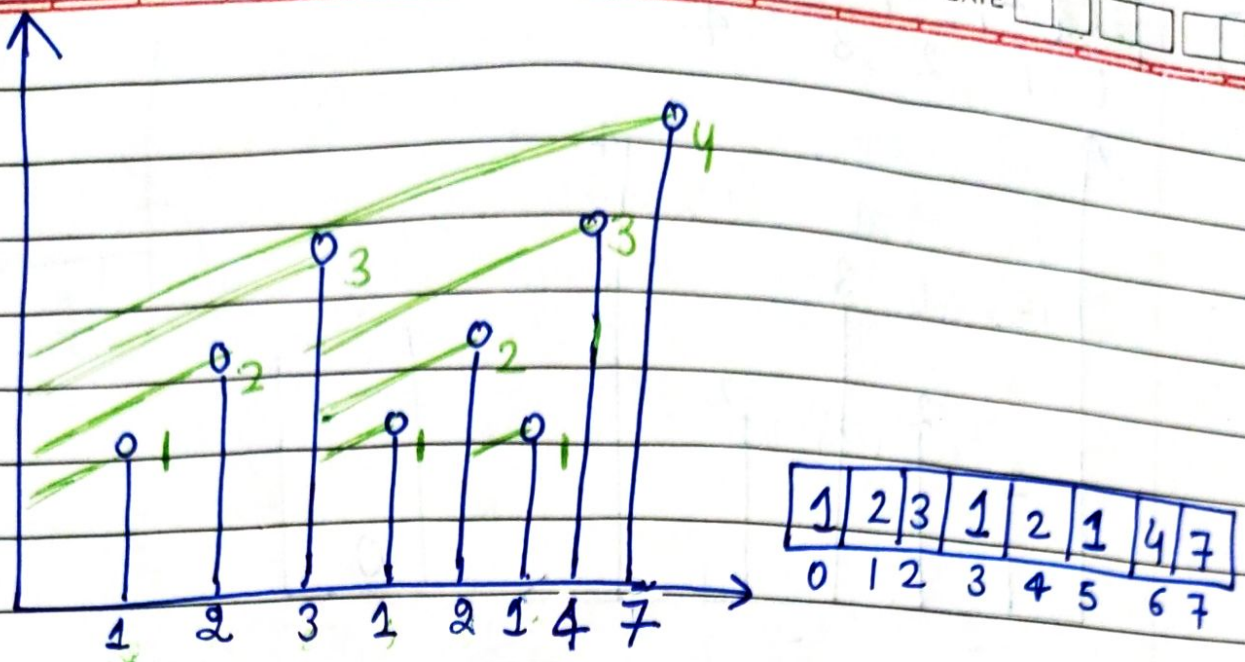
Approach

1. Pop all the ^index of elements from stack which is smaller than the current value.
2. Now the top of stack will consist the peek() which is greater than the current value.

Span.

Stock ^ for current value → current index - peek index

3. Now Push current index into the Stack.



* हम एक Stack बनाएँगे जिसमें हम array के elements के corresponding index store कराएँगे।

Way 1 → हम elements store नहीं कर रहे, हम सिर्फ index ही क्यों store कर रहे हैं?

↓

The stack will only contain the index because it will help us to calculate the answer much more easily.

Way 2 → हम Pair class के object में element और index डालके, उसे भी Stack में store कर सकते हैं

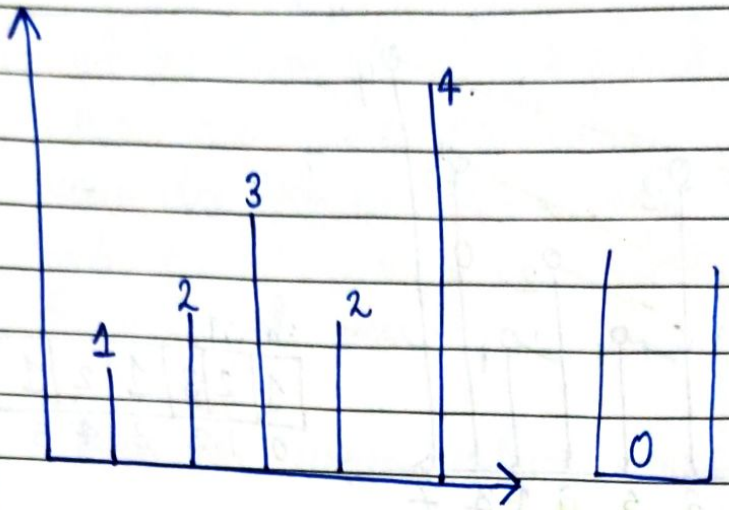
और stack के top पर index है j और current index i है तो current element (day) का span होगा (i-j)+1

We will maintain a stack.
The span of first element is always 1.
Because there is no element before it.
Also, we push its index into span

DRY RUN * We create an ans array for storing its stock span corresponding its each element. DATE

① | 1 | 2 | 3 | 2 | 4 |

0 1 2 3 4.

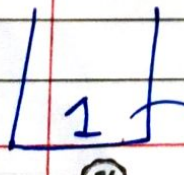


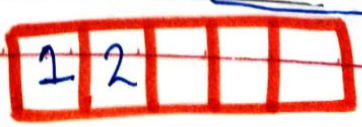
We put index 0 for first element in Stack.
For first element, Stock Span will be 1 because there is no element on the left smaller than 1.

② i = 1 अब एक for loop चलेगा from index 1 to last index of array to traverse of each index.
current index वाला element अगर stack peek से छोटा है तो pop कराओ element जब तक peek () index वाला element current index वाले element से ज्यादा न हो।
अगर stack empty हो जाए तो current index stack में डालदो. stack span will be current index + 1.

1 < 2
arr[0] < arr[1] So pop it out. If stack become empty ans[1] = i+1;
& Put 1 in Stack
↳ Index

ans[i] = 2

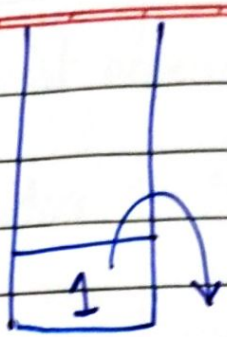




PAGE

③ $i = 2$

DATE



arr[1] < arr[2].

So pop 1 out of Stack

Now stack becomes empty.

So stack span will be index + 1.

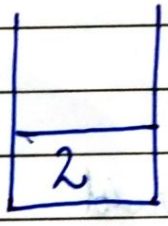
ans[2] → current index + 1

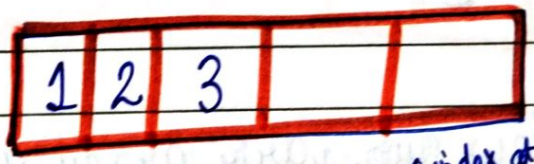
$i = 2$

ans[2] = 2 + 1

ans[2] = 3

* Put i into Stack.





④ $i = 3$

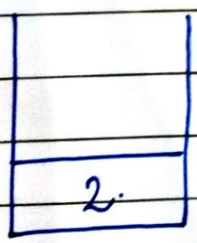
arr[2] > arr[3]
3 > 2.

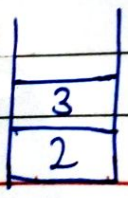
→ Peek index (index of the peek of stack).
→ current index

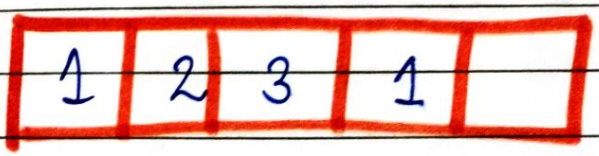
So Stock Span will be 3 - 2 = 1

ans[3] = 1

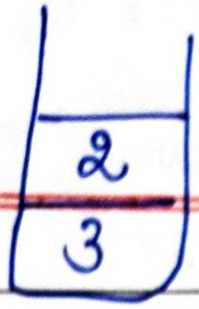
Now Push 3 into Stack.







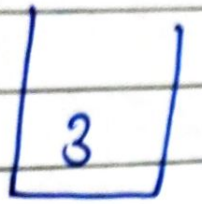
⑤ i=4



arr[4] is greater than arr[2]

So pop 2 out

DATE



arr[4] > arr[3]

So pop 3 out.

Stack become empty

So, Stack Span = i+1.

ans[i] = i+1

ans[i] = 5

1	2	3	1	5

Now our arr array is traversed completely & we got our ans array.

Approach

DATE

* first element के लिए 1 डाला stack में.
* index 2 से last index तक for loop चलेगा.
* current index i वाला element (i) अगर peek index (j) से बड़ा है तो तब तक peek index element current index element से बड़ा न मिले, तब तक stack से index pop करते चलो।

→ अगर greater element वाला index stack में मिलगया तो current element का stock span होगा → ( ) & push it in the stack ↳ peek index - current index

→ अगर pop करते हुए stack empty हो गया है (It means इसे कोई greater Element index stack में नहीं मिला)

तो current element का Stock Span होगा

↳ i + 1

↓

current index.

& Push i in the Stack.

<code>

DATE

```
import java.util.*;
public class Main
{
  public static void main (String [] args)
  {
    BufferReader br = new BufferReader
        (new InputStreamReader (System.in));

    int n = Integer.parseInt (br.readLine ());
    int a[] = new int [n];
    for (int i=0; i<n; i++)
      {
        a[i] = Integer.parseInt (br.readline ());
      }

    int [] span = solve (a);
    display (span);
  }

public static (void display (int [] a)
  {
    for (int i=0; i<a.length(); i++)
      {
        System.out.println (a [i]);
      }

  }
```

→ solve function ∈ answer array ∈ ll

→ display function ∈ to print the answer array

</code>

```
public static int[] solve(int[]arr)
{
    int ans[] = new int[arr.length];
    ans[0]=1;
    Stack<Integer> stack = new Stack<>();
    stack.push(0);
    for(int i= 1; i<= arr.length-1 ; i++)
    {
        while (stack.isEmpty() == false &&
        && arr[stack.peek()] < arr[i])
        {stack.pop();
        }
        if (stack.isEmpty() == true)
        {
            ans[i]= i+1;
            stack.push(i);
        }
        else
        {
            int j = stack.peek();
            ans[i]=i-j;
            stack.push(i);
        }
    }
    return ans;
}
}
```

हर element के stack span को save करने के लिए हम answer (ans) array बनाएंगे

first element का stock span 1 ही होगा

एक stack बनाएंगे

first element का index 0 stack में push करेंगे

arr के 1st index से last index पर loop चलाएंगे

जब तक stack empty नहीं है और peek index वाला element i index वाले element से छोटा है तो pop

अगर stack empty है तो stock span होगा i+1 और stack में i push करो

अगर stack empty नहीं है और current index वाले element से greater element मिल गया at arr[peek()] तो stock span होगा i - peek और stack में i push करेंगे

ans array full हो चुका है तो उसे return कर दो