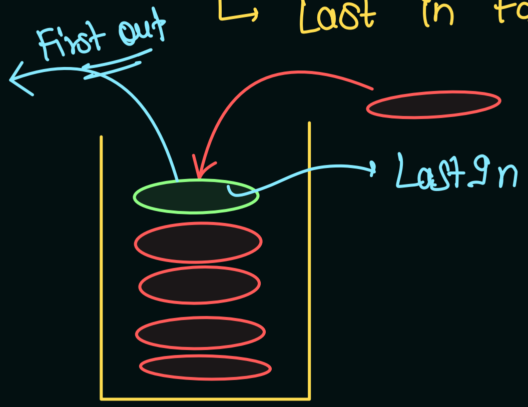
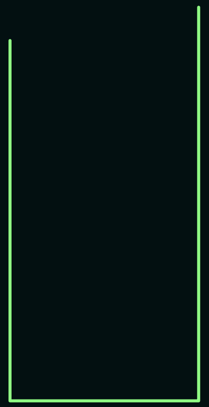
Stack

DS } Org. of Data
- → get
- → set
- ↳ Retrieve

LIFO Semantic } unique behaviour of stack

↳ Last in First out





Integer data

get → peek / top

add → push] add on top of stack

remove → pop();

↳ remove top data and return that value.

Functions of Stack - →

① get] → st.peek() → O(1)

② set] → add → st.push(val); → O(1)

↳ remove → st.pop()! → O(1)

③ size → st.size(); → O(1)

④ isEmpty → st.isEmpty() → boolean
- ↳ True
- ↳ False,] → O(1)

→ Usage of Stack] dependent

- Queue usage

→ Stack Implementation] → generic.

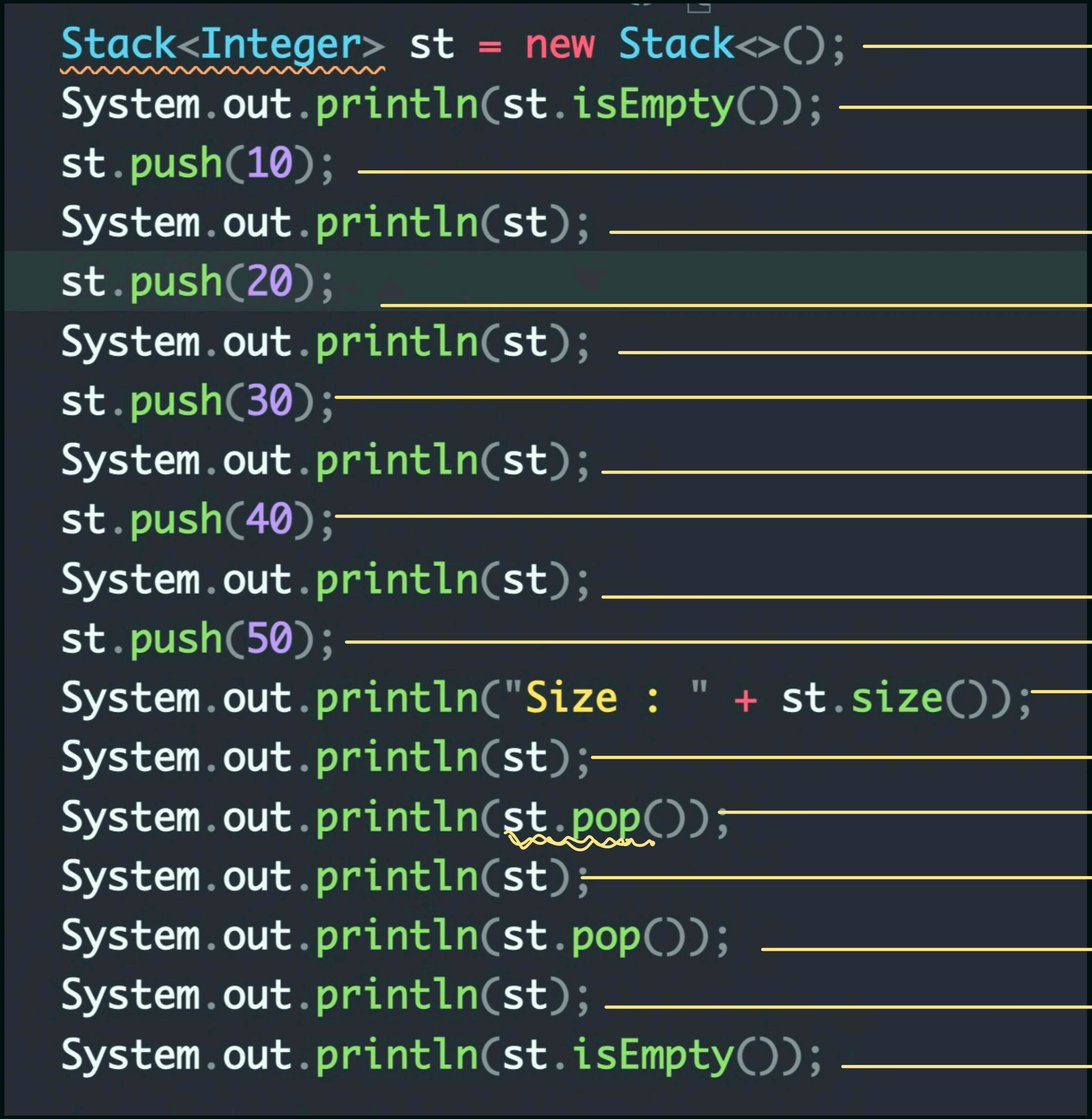
→ Queue Implementation

→ Adapter] → solver

Stack → horizontal   [ ] ↵ add → remove

[10, 20, 30

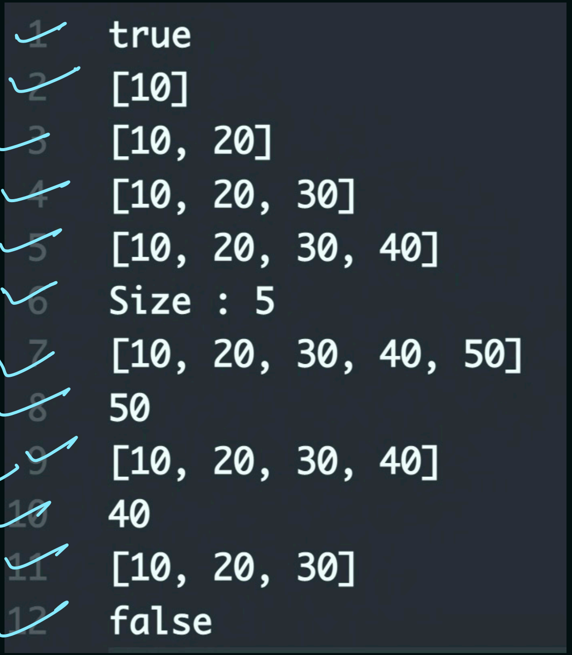
Stack → St



```java
Stack<Integer> st = new Stack<>();
System.out.println(st.isEmpty());
st.push(10);
System.out.println(st);
st.push(20);
System.out.println(st);
st.push(30);
System.out.println(st);
st.push(40);
System.out.println(st);
st.push(50);
System.out.println("Size : " + st.size());
System.out.println(st);
System.out.println(st.pop());
System.out.println(st);
System.out.println(st.pop());
System.out.println(st);
System.out.println(st.isEmpty());
```

→ [ ] ✓
→ True ✓
→ ✓ ✓
→ [10]; ✓
→ ✓ ✓
→ [10, 20]; ✓
→ ✓ ✓
→ [10, 20, 30]; ✓
→ ✓ ✓
→ [10, 20, 30, 40]; ✓
→ ✓ ✓
→ Size : 5 ✓
→ [10, 20, 30, 40, 50] ✓
→ pop → 50 ✓
→ [10, 20, 30, 40]; ✓
→ pop → 40 ✓
→ [10, 20, 30]; ✓
→ False ✓

output is correct



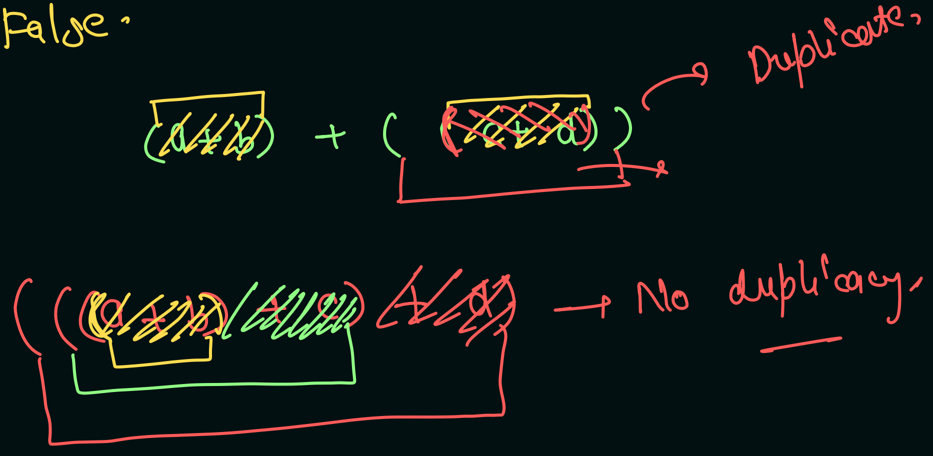
```
✓ true
✓ [10]
✓ [10, 20]
✓ [10, 20, 30]
✓ [10, 20, 30, 40]
✓ Size : 5
✓ [10, 20, 30, 40, 50]
✓ 50
✓ [10, 20, 30, 40]
✓ 40
✓ [10, 20, 30]
✓ false
```

Duplicate brackets:

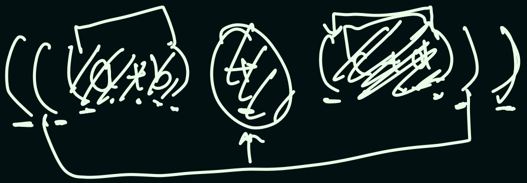
→ Expression is balanced → No. of opening brackets are equal to closing brackets & they are perfectly arranged.

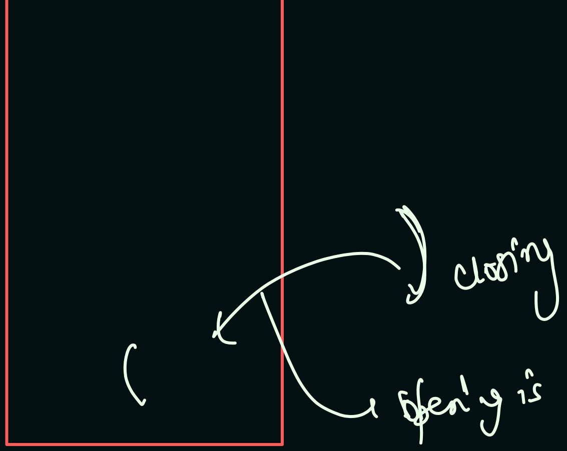
1. (a + b) + ((c+d)) → True
2. (((a+b) + c) + d) → False
3. ✓ (((a+b) + (c+d))) → True
4. (((a+b) + c) + (d +e)) + (f+g) → False.

Example.



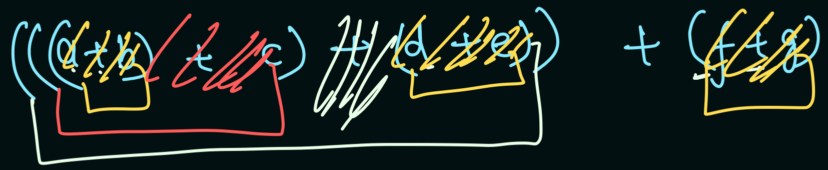
Action
→ closing bracket
otherwis
→ stack push

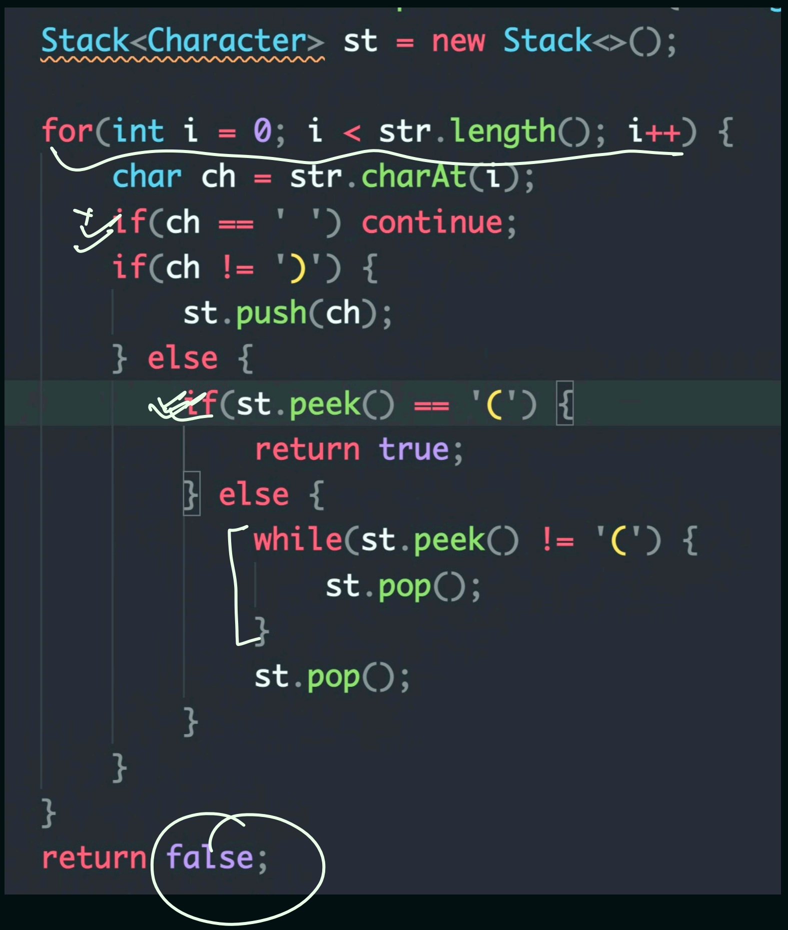




opening is at top → duplicacy is proper

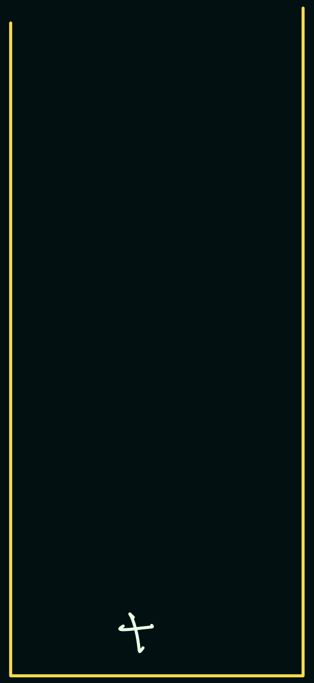
othrn true





```
Stack<Character> st = new Stack<>();

for(int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    if(ch == ' ') continue;
    if(ch != ')') {
        st.push(ch);
    } else {
        if(st.peek() == '(') {
            return true;
        } else {
            while(st.peek() != '(') {
                st.pop();
            }
            st.pop();
        }
    }
}
return false;
```

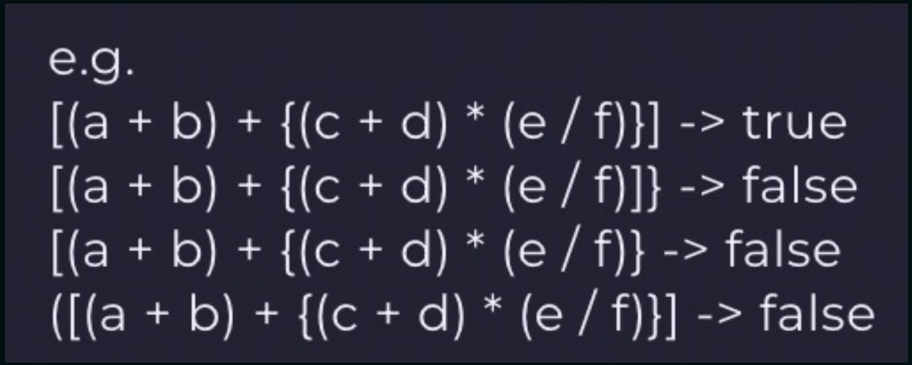


Balanced bracket

hint

- opening bracket
  - push
- closing
  - same pair opening
  - peek() pop

Balanced brackets



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

opening bracket → push

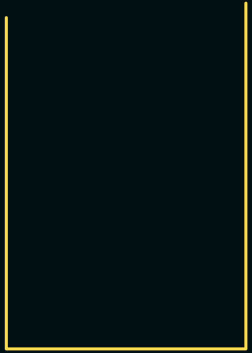
closing bracket → check if corner opening is available or not

op closing

(→)

{ → }

[→]



NOTE!
if at last stk.size()>0
that meas there are more opening bracket then closing bracket

([{a+b) → OR

([a+b] + c)] }

closing → stk.size()==0
More closing bracket

next greater on Right:

~~Right max~~ 14, next greater = 12

0	1	2	3	4	5	6	7	8
10	6	12	5	3	11	14	8	9

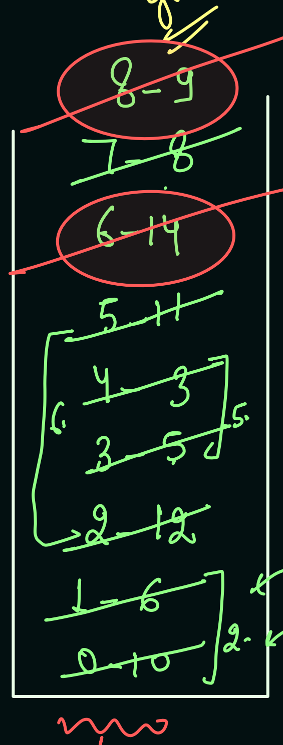
0	1	2	3	4	5	6	7	8
2. 12	2. 12	6. 14	5. 11	5. 11	6. 14	-1	8. 9	-1

Right max

next greater

next greater on Right →

value of = next greater



8-9
7-8
6-14
5-11
4-3
3-5
2-12
1-6
0-10
Index_value

Stack

Maintain → decreasing order of value.

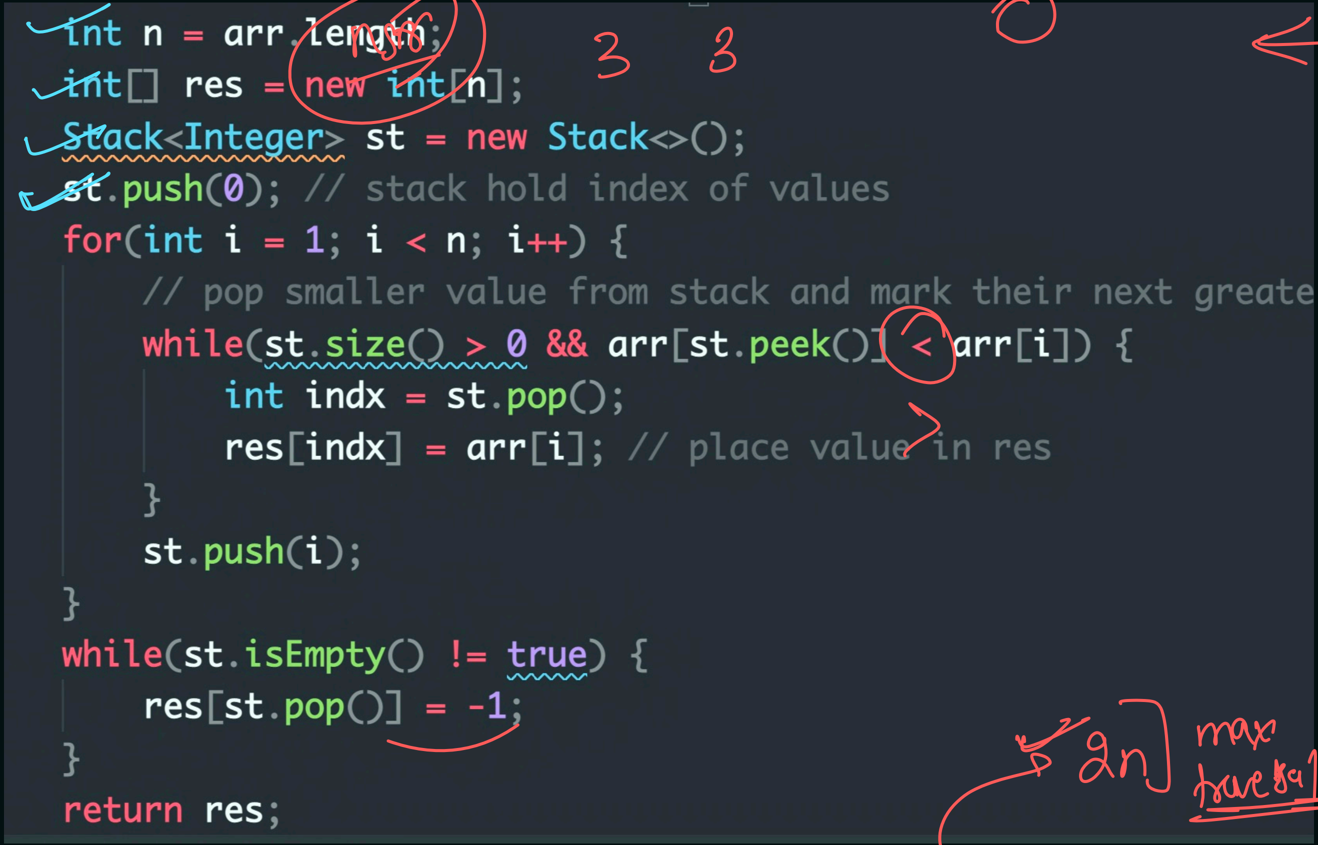
→ pair ≠ → Index carry] ← we can acces value as well array →

0	1	2	3	4	5	6	7	8
2	5	9	3	1	12	6	8	7

n = 9

ngr res →

0	1	2	3	4	5	6	7	8
5	9	12	12	12	-1	8	-1	-1



```
int n = arr.length;
int[] res = new int[n];
Stack<Integer> st = new Stack<>();
st.push(0); // stack hold index of values
for(int i = 1; i < n; i++) {
    // pop smaller value from stack and mark their next greate
    while(st.size() > 0 && arr[st.peek()] < arr[i]) {
        int indx = st.pop();
        res[indx] = arr[i]; // place value in res
    }
    st.push(i);
}
while(st.isEmpty() != true) {
    res[st.pop()] = -1;
}
return res;
```

ngr

3 3

2n] max traversal

complexity → O(n)

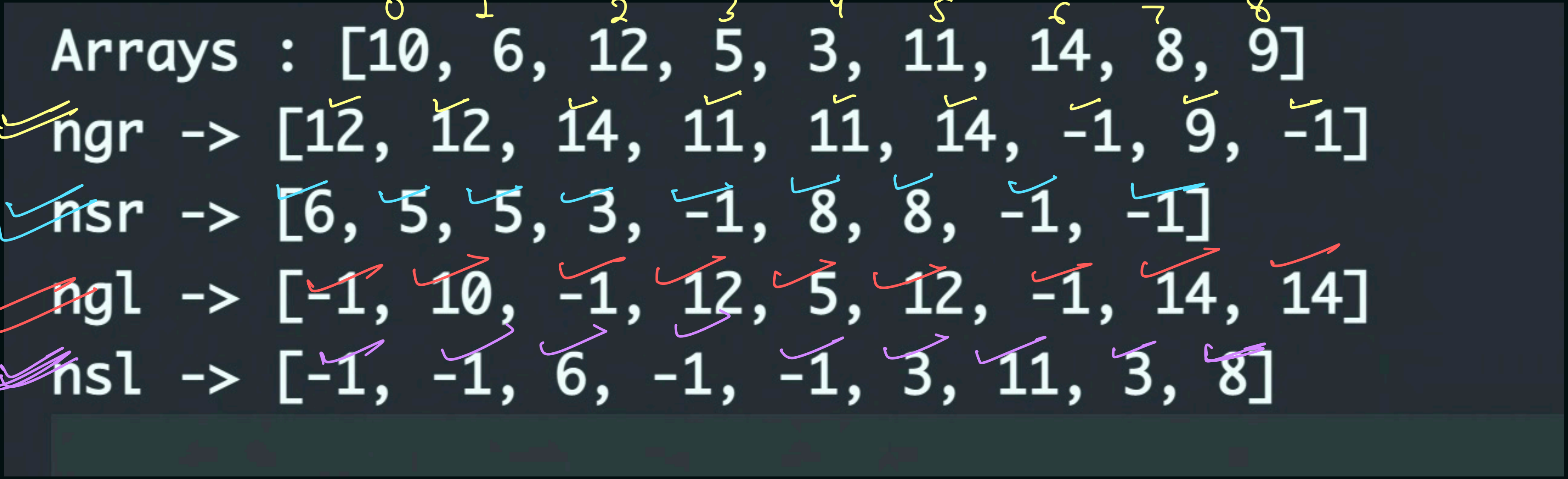
~~8-7~~
~~7-8~~
~~6-6~~] 7.
~~5-12~~
~~4-1~~
~~3-3~~] 5.
~~2-9~~
~~1-5~~] 2.
~~0-2~~] 1.

Stack - st

8-7
7-8
6-6
5-12
4-1
3-3
2-9
1.5
0-2

3-3 3-3
~~2-9~~
~~1-5~~
0-2

next greater / smaller] left
Right

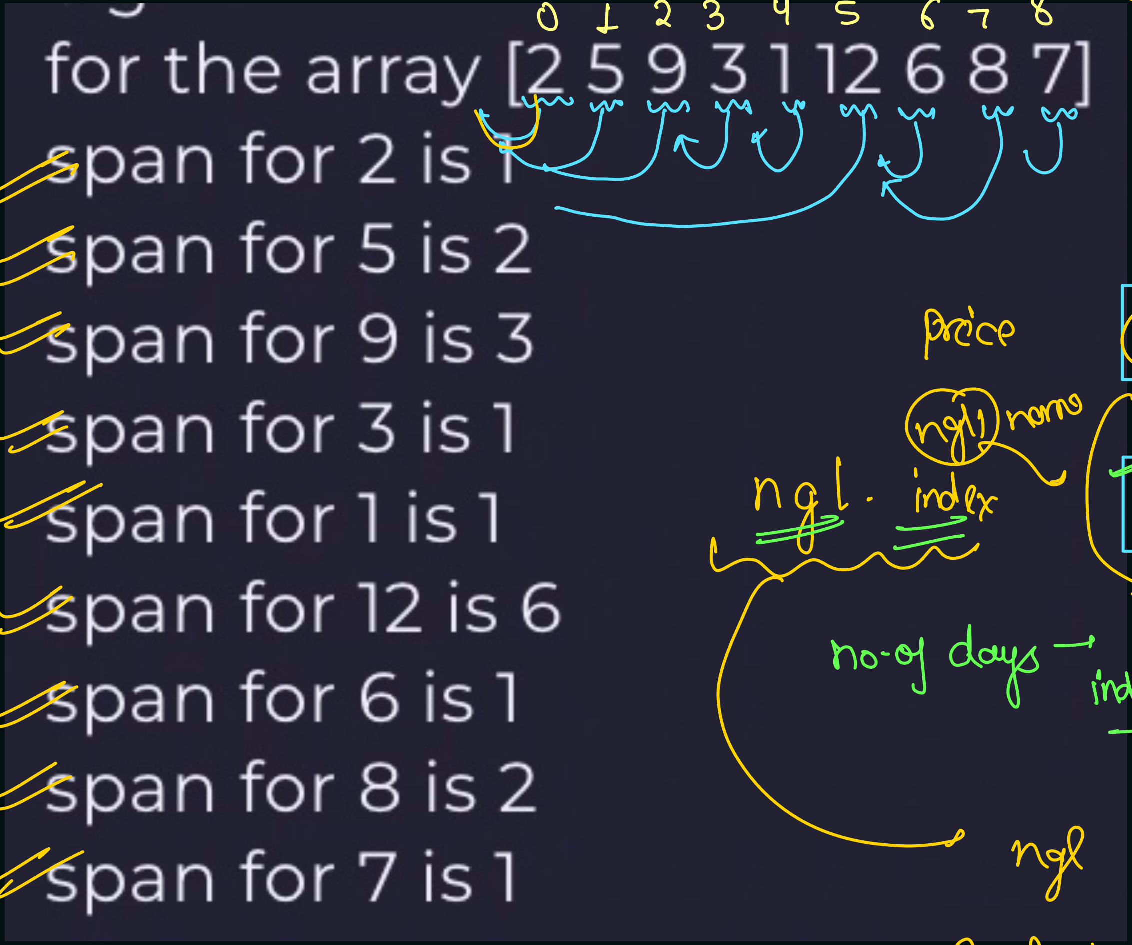


0 1 2 3 4 5 6 7 8

```
Arrays : [10, 6, 12, 5, 3, 11, 14, 8, 9]
ngr -> [12, 12, 14, 11, 11, 14, -1, 9, -1]
nsr -> [6, 5, 5, 3, -1, 8, 8, -1, -1]
ngl -> [-1, 10, -1, 12, 5, 12, -1, 14, 14]
nsl -> [-1, -1, 6, -1, -1, 3, 11, 3, 8]
```

Stock span:

Span → No. of days before today, when stock price was higher than today.



0 1 2 3 4 5 6 7 8

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

→ next greater on left

Price

0	1	2	3	4	5	6	7	8
2	5	9	3	1	12	6	8	7

ngl name

ngl. index

0	1	2	3	4	5	6	7	8
-1	-1	-1	2	3	-1	5	5	7

1 2 3 1 1 6 1 2 1

no. of days → indx - nglindex

ngl not Exist → -1

for final Res

ngl[i] = i - ngl[i];

Daily temperature:

next greater index

	0	1	2	3	4	5	6	7
	[73,	74,	75,	71,	69,	72,	76,	73]
ngr index =	1	2	6	5	5	6	$\boxed{6}$	$\boxed{7}$
res =	1	1	4	2	1	1	0	0

ngr[i] - i

diff

next greater on right index

res = [ngr[i] = ngr[i] - i

if ngr is not available res[i] = i