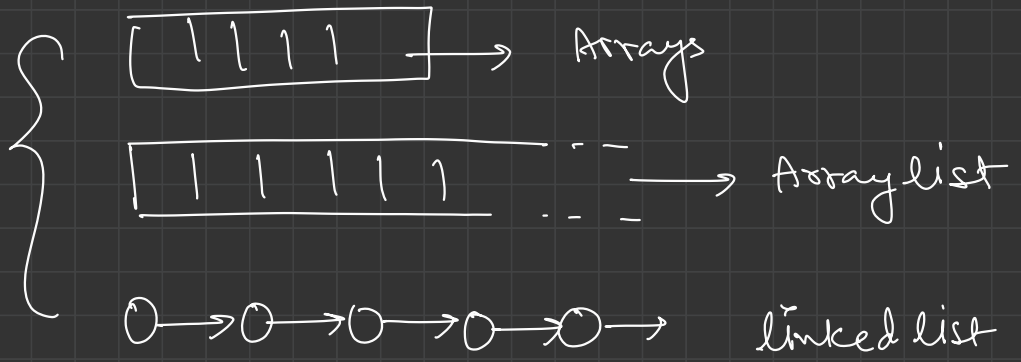
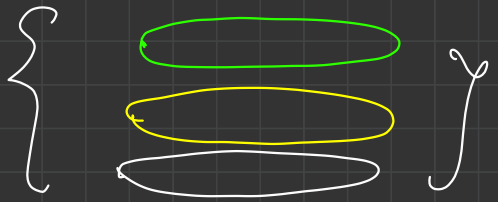
Stacks → linear data structure



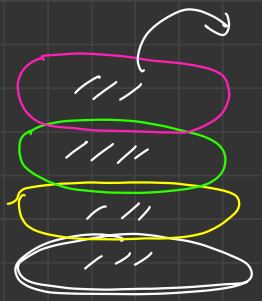
(1) Stack is a linear DS



dishes are stacked together!

Stack of plates

Stack Operations

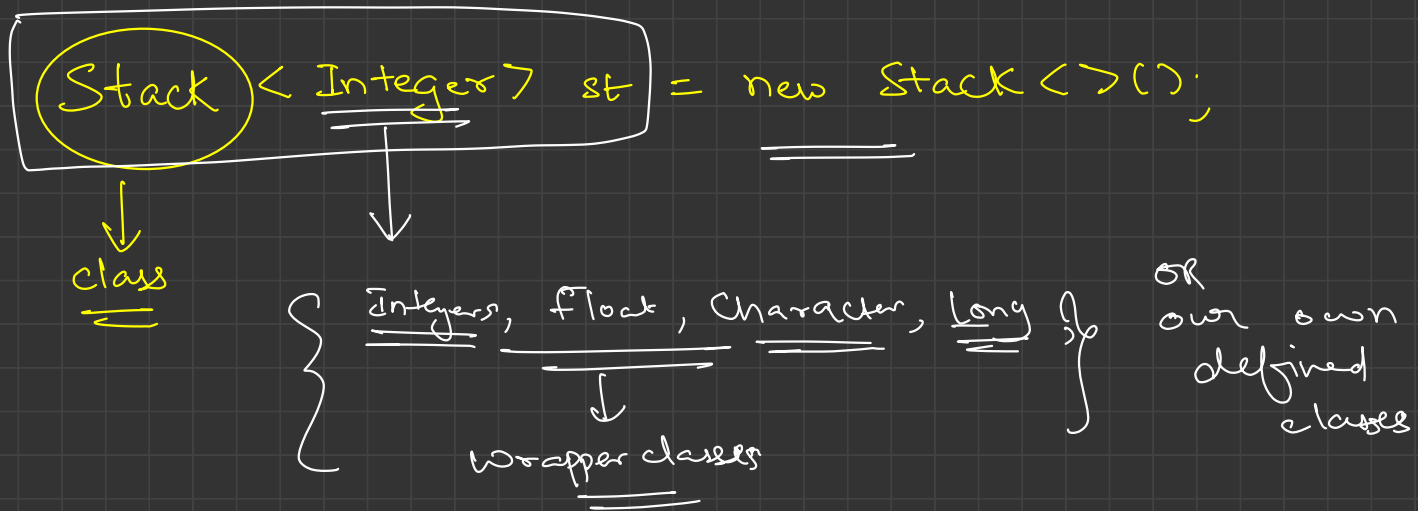


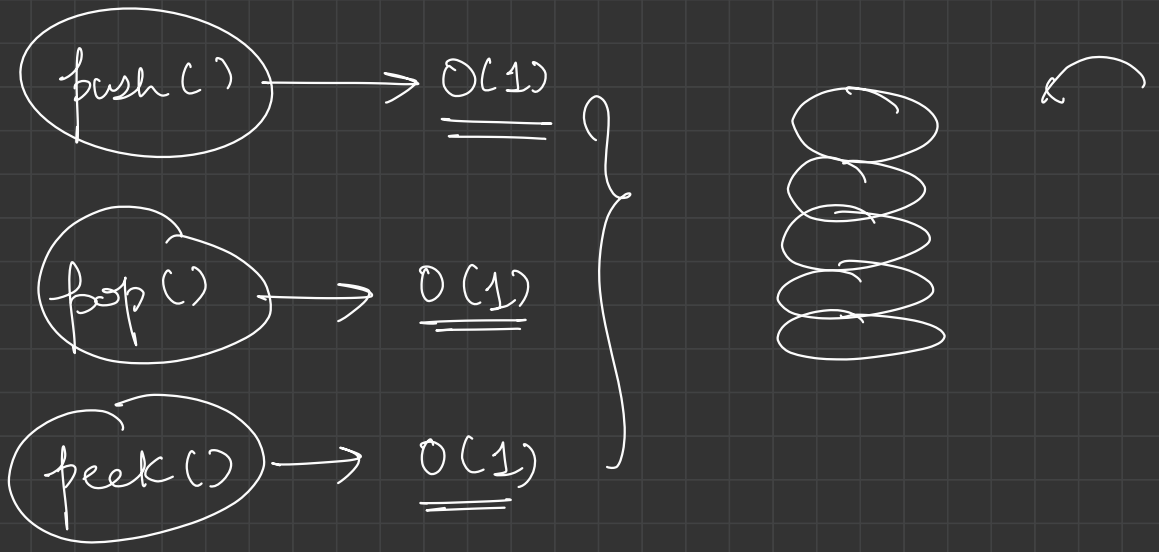
1. addition of new data in a stack is done on top
2. deletion of data from a stack is done from top
3. data on top is only readable

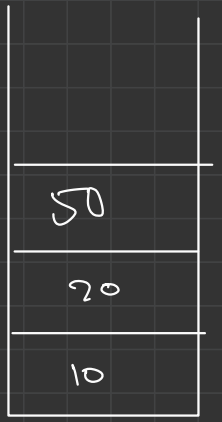
① push() : add data in the stack (top of the stack)

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

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


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


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



St.peek() → 30

St.pop()

St.peek() → 20

St.push(50)

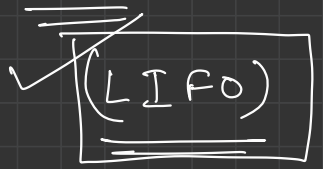
St.peek() → 50

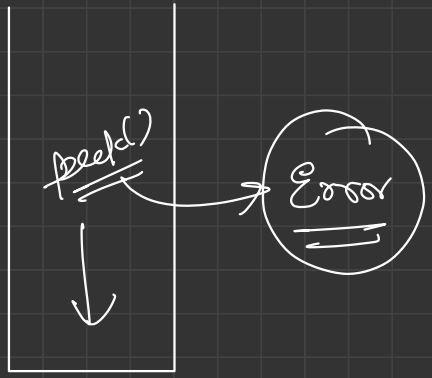
St.size() → 3

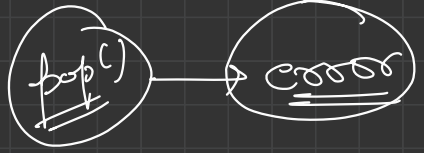
① size()

→ Size of the Stack!

Stack follows, last In, But first out.







Stack underflow

St

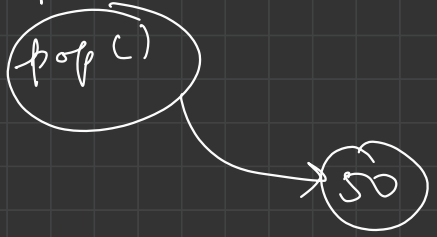
push (20)

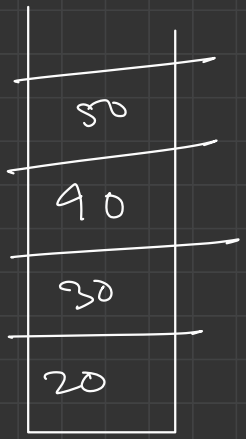
push (30)

push (40)

push (50)

pop ()





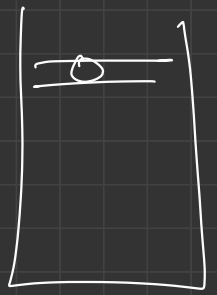
(last person In, first out)

ArrayList

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

Stack

1. push() on top
2. pop() from top
3. peek() See on top()
4. Size of Stack



linked list

add last()

add first()

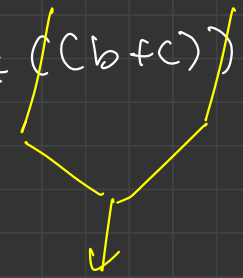
remove last()

remove first()

Size of linked list()

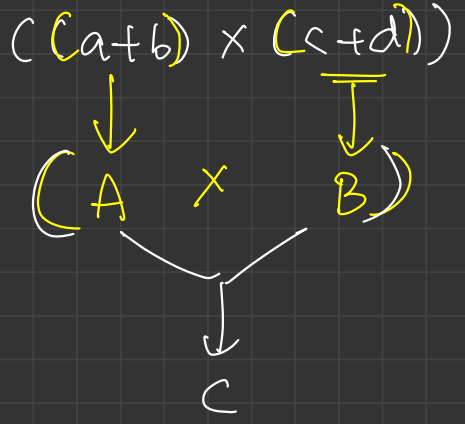
Q $(a+(b\times c)-(d\times f))$ ⟶ false (No extra brackets)

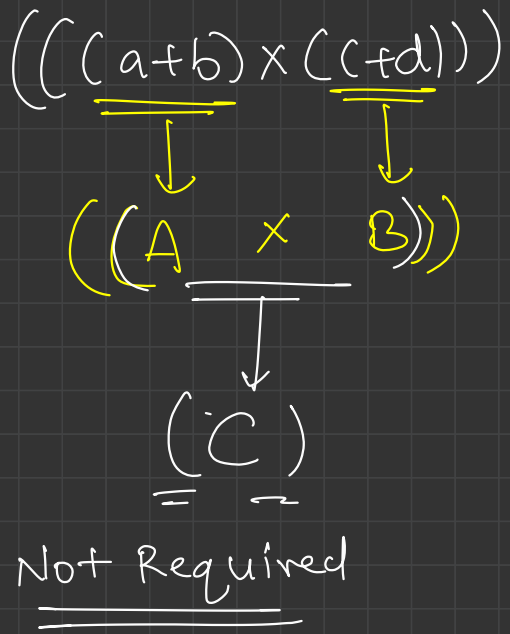
$(a+((b+c))-(d\times f))$ ⟶ true



extra bracket

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


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

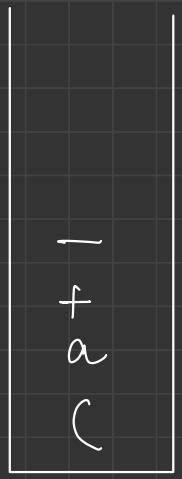

(((a+b) x (c+d)))
((A x B))
(C)
Not Required

(a + (b x c) − (d x f))

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

(+) → '(' or operator char.

(−) → ')'



Last person opened has to be closed first.

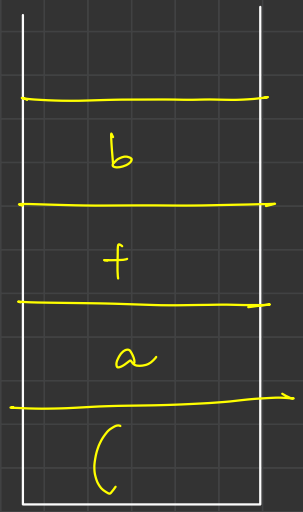
(LIFO)

search for corresponding '(' bracket in stack

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

means, we have an exp.

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



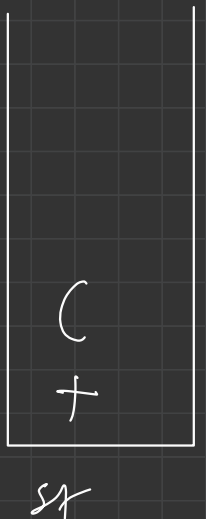
St

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

broadcast:

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

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

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

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

    return false;
}
```

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



a+b+(d)

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

(5)

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

whileC() → N times in its life span

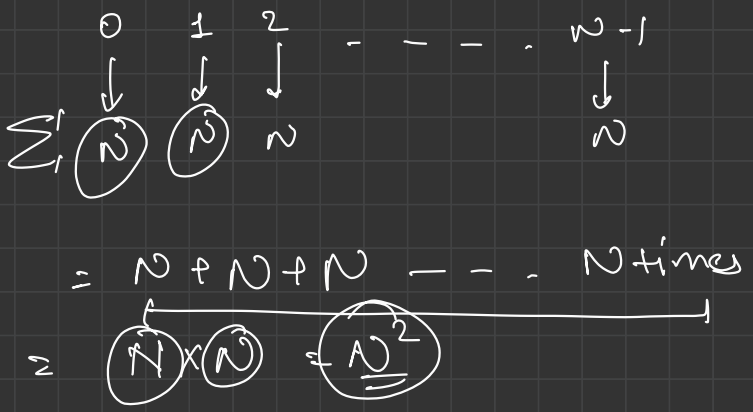
O(2N)

i = 0, 1, 2, 3, 4

N times

O(N²)

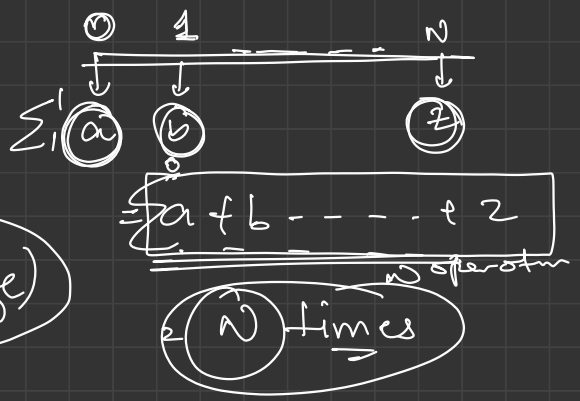
```
for (i -> 0 -> n)
{
    while (n)
}
```

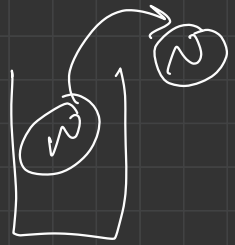


O(N)

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

(N time in life)





Next Greater Element on Right

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

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

Solve? (1) Bruteforce

```
int[] nger = new - - -
for(int i=0 ; n)
{
   for(int j= i+1 ; n)
      if(arr[j] > arr[i])
      {
         nger[i] = arr[j]
         break;
```

}

} TC: $O(N^2)$

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

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

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

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

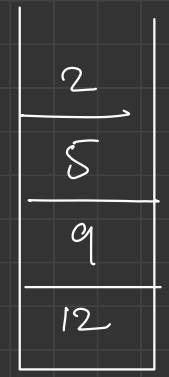
nger[] : [5, 9, 12, 12, 12, -1, 8, -1, -1]

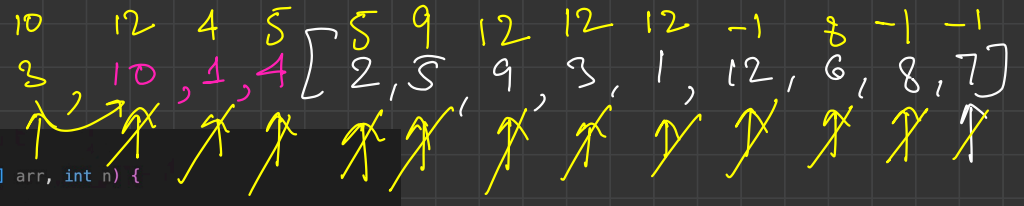
Approach 1

(1) traverse array from last to start.

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

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

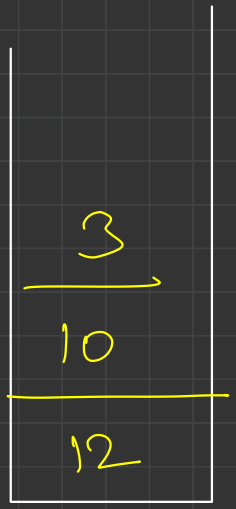





```java
class Solution {
    public static long[] nextLargerElement(long[] arr, int n) {
        // Write code here and print output
        long[] nger = new long[n];
        Stack<Long> st = new Stack<>();

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

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

            st.push(arr[i]);
        }

        return nger;
    }
}
```

N times

N time
life time



St

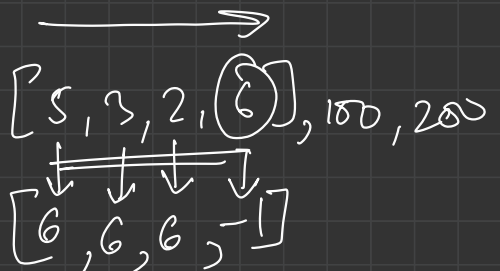
Time Complexity : O(N)

Space Complexity : O(N)

↳ Stack Space

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

Approach 2



push ⟶ when topmost element of stack is greater than you

pop() ⟶ when some greater than me ones encountered,

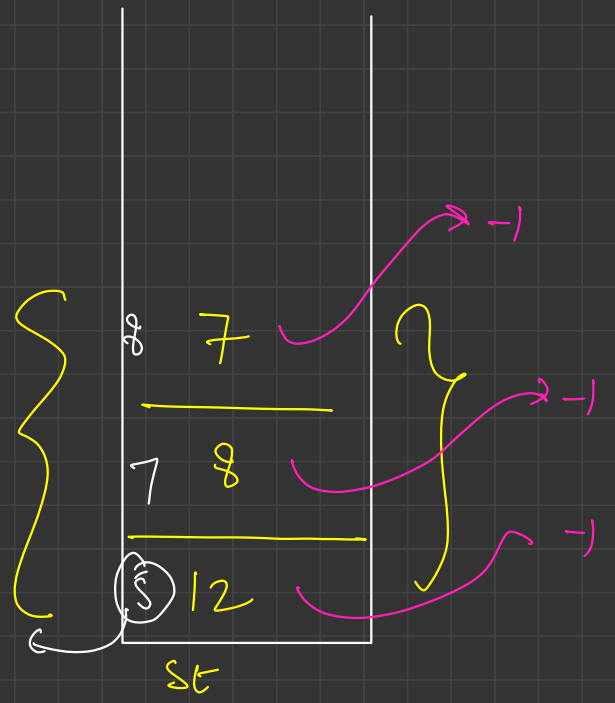
arr[] : [2, 5, 9, 3, 1, 12, 6, 8, 7]

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

0 1 2 3 4 5 6 7 8

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

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



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

    for (int i = 0; i < n; i++) {

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

        st.push(i);
    }

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

    return nger;
}
}
```

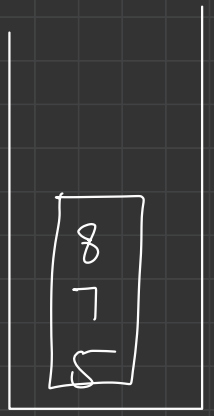
N times

N times
Lifetime

0 1 2 3 4 5 6 7 8

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

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



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