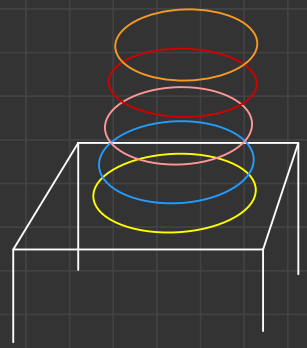
Stacks.

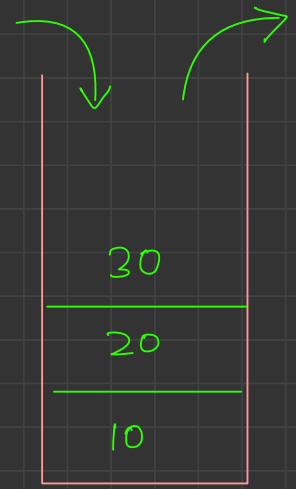
→ linear data structures

1. Arrays
2. Array list
3. Strings
4. Linked list
5. Doubly linked

Stacks. { last in, first-out } { LIFO }



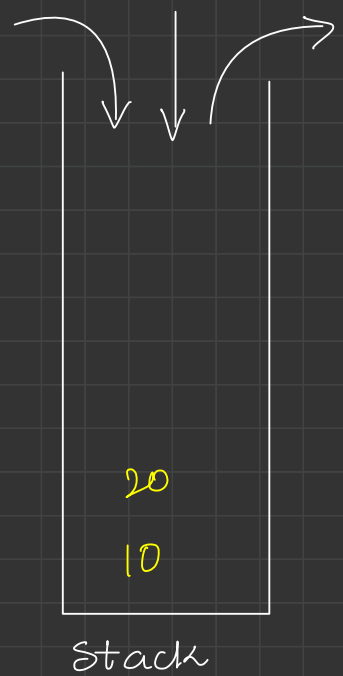
these disks are stacked up



Stacks

last disked was the first to be used.

Stacks → linear data structure
→ follows LIFO




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

→ push(v)
↳ add data to the stack

```
st.push(10);
st.push(20);
st.push(30);
```

→ size
↳ return size of stack

→ pop()
↳ removes the topmost data.

st.pop() ⇝ 30

st.peek() ⇝ 20

→ peek()
↳ enables us to view the topmost ele.

Practical Examples of stacks

- → Recursion
- → Memory Management
- → undo and redo functionality is implemented using stacks.

Q Extra Brackets.

String str = "(a+b)" → No

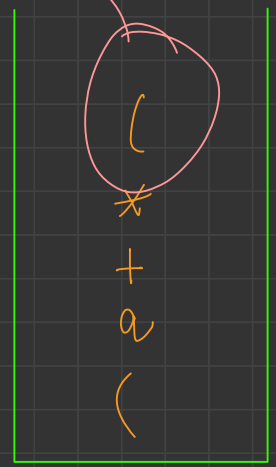
= "((a+b))" → Yes

= "(a+b) + ((*d)) + (m+n+(e/f)()) " → Yes

= "((a)+(b))" → No

str = "(a + (b * d + f - (m) + n - o) * (z)())"

Extra Pair!



Stack

Next Greater Element on Right

arr[] = {3, 6, 1, 2, 7, 3, 4, 1, 2, 5}

nger[] = {6, 7, 2, 7, -1, 4, 5, 2, 5, -1}

Bruteforce

TC: $O(N^2)$

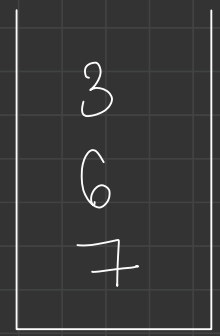
SC: $O(1)$

TC: $O(N)$?

6	7	2	7	-1	4	5	2	5	-1

nger[]

arr[] = { 3, 6, 1, 2, 7, 3, 4, 1, 2, 5 }



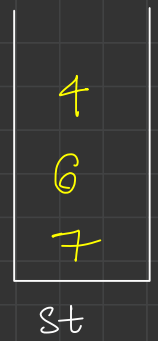
Stack

potential nger

```java
public static long[] nextLargerElement(long[] arr, int n)
{
    // stack holds potential nger
    Stack<Long> st = new Stack<>();

    long[] nger = new long[n];

    for (int i = n - 1; i >= 0; i--) {
        long ele = arr[i];

        while (st.size() > 0 && st.peek() <= ele) {
            st.pop();
        }

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

        st.push(ele);
    }

    return nger;
```

arr[] = { 4, 3, 6, 5, 6, 7, 2, 1, 5 }

nger[] = { 6 6 7 6 7 -1 5 5 -1 }



push $\longrightarrow$ (N) times

pop $\longrightarrow$ (N) times

total time complexity

$$\sum \; a + b + c + \quad + \quad + \quad + \; \alpha = N$$

$$\boxed{\begin{array}{l} TC: O(N) \\ SC: O(N) \end{array}}$$

	0	1	2	3	4	5	6	7	8	9
arr[] =	3,	6,	1,	2,	7,	3,	4,	1,	2,	5
	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~
	6	7	2	7	-1	4	5	2	5	-1

Approach 2

st

↳ pool of people looking for nger

```java
public static long[] nextLargerElement(long[] arr, int n)
{
    // stack: people looking for nger
    Stack<Integer> st = new Stack<>();

    long[] nger = new long[n];

    for (int i = 0; i < n; i++) {
        long ele = arr[i];

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

        st.push(i);
    }

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

    return nger;
}
```

nge[ ] ← [-1 -1 6 6 -1 7 7 4 4 7]

ngr ← [6 7 2 7 -1 4 5 2 5 -1]

	0	1	2	3	4	5	6	7	8	9
arr[] =	3	6	1	2	7	3	4	1	2	5

Stack

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

1. stock span problem
2. largest area histogram