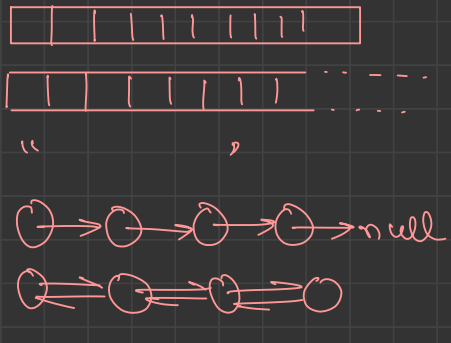
Stacks

★ linear data structures

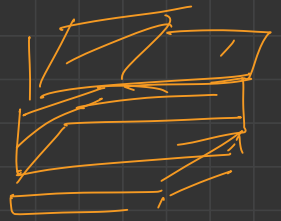
1. Arrays
2. Arraylist
3. Strings
4. Linked list
5. Doubly Linkedlist



Stacks

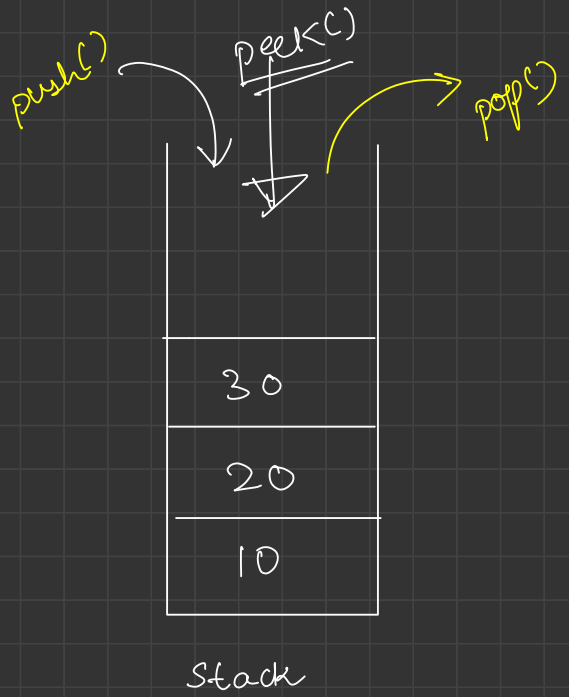
{ last person in, first person out } { LIFO } out- last in first

Stack of CD



Stack of Books

you can only see the topmost data in a Stack.



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

☆ Add element to a stack

- push(int v)
  - st.push(10)
  - st.push(20)
  - st.push(30)
  - st.push(-100)

☆ Remove element from a stack

- pop()

Remove topmost Element

(Returns the value)

- st.pop() → -100

☆ Size of Stack

- st.size() → 3

☆ to view data on toponost position

- peek()
  - st.peek() → 30

```
class Stack
{
    ArrayList<Integer> list;
    int size;
    Stack()
    {
        list = new ArrayList<>();
        size = 0;
    }
                                TC:O(1)
    void push(int v)            SC:O(1)
    {
        list.add(v);
        size++;
    }
```

```
                                TC:O(1)
                                SC:O(1)
    int pop()
    {
        if (size > 0)
        {
            int ele = list.remove(size-1);
            size--;
            return ele;
        }else{
            return -1;
        }
    }
                                TC:O(1)
    int peek()                  SC:O(1)
    {
        if (size > 0)
        {
            return list.get(size-1);
        }else
            return -1;
    }
```

Practical Examples of Stack

- → Recursion
- → Memory Managment
- → undo functionality, implated using Stack.
- → Cache

Q Extra Brackets

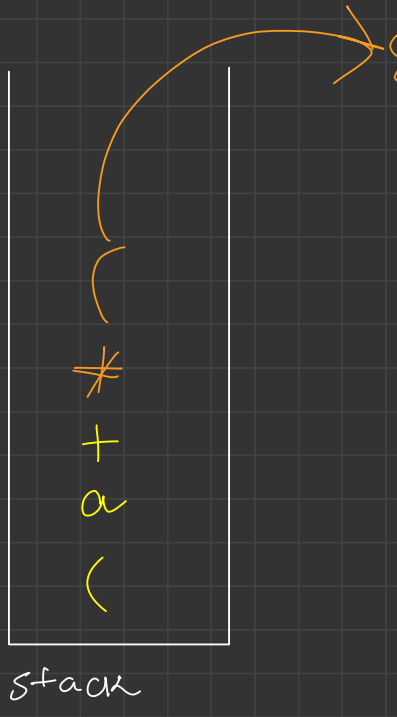
String str = "(a+b)" → No

"((a+b))" → yes

"(a+b)+(c+d+(e*f)())" → yes

"((a)+((b)))" → No

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



last opened Bracket is the first Bracket to be paired with closed Bracket.

```java
public boolean ExtraBrackets(String exp) {
    // Write your code here

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

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

        if (ch != ')') {
            // blindly add to the stack
            st.push(ch);
        } else {
            if (st.peek() == '(') {
                // nothing in between, hence extra pair of bracket
                return true;
            } else {
                // remove people in between you and corresponding opening bracket
                while (st.peek() != '(') {
                    st.pop();
                }

                // as people were removed, hence you were a valid pair
                // remove corresponding open braket
                st.pop();
            }
        }
    }

    return false;
}
```

exp = "((a)+(b)+(c+d))"

TC: O(N)

SC: O(N)

Next Greater Element On Right

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

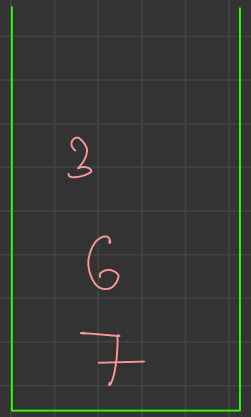
nger[] : {6, 7, 2, 7, -1, 4, 5, 2, 5, -1}

{6, 7, 2, 7, -1, 4, 5, 2, 5, -1}

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

Brute force

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



Stack
(potential nger)

```java
public static long[] nextLargerElement(long[] arr, int n)
{
    //Write code here and print output

    // stack is having potential next greater element on right
    Stack<Long> st = new Stack<>();

    long[] nger = new long[n];

    // going from right to left, because we no nger for rightmost person is -1
    for (int i = n - 1; i >= 0; i--) {
        //are there some potential nger in stack smaller than you, if yes remove them
        // I'm comvering all the range for them
        while (st.size() > 0 && arr[i] >= st.peek()) {
            st.pop();
        }

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

        st.push(arr[i]);
    }

    return nger;
}
```

O(N) operations

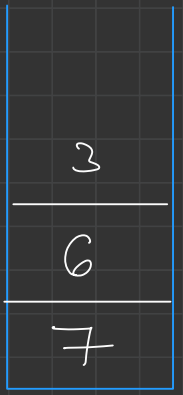
N pop

O(N) lifetime

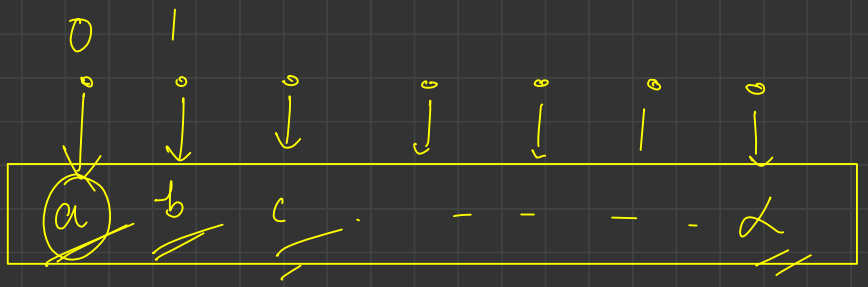
N push

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

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

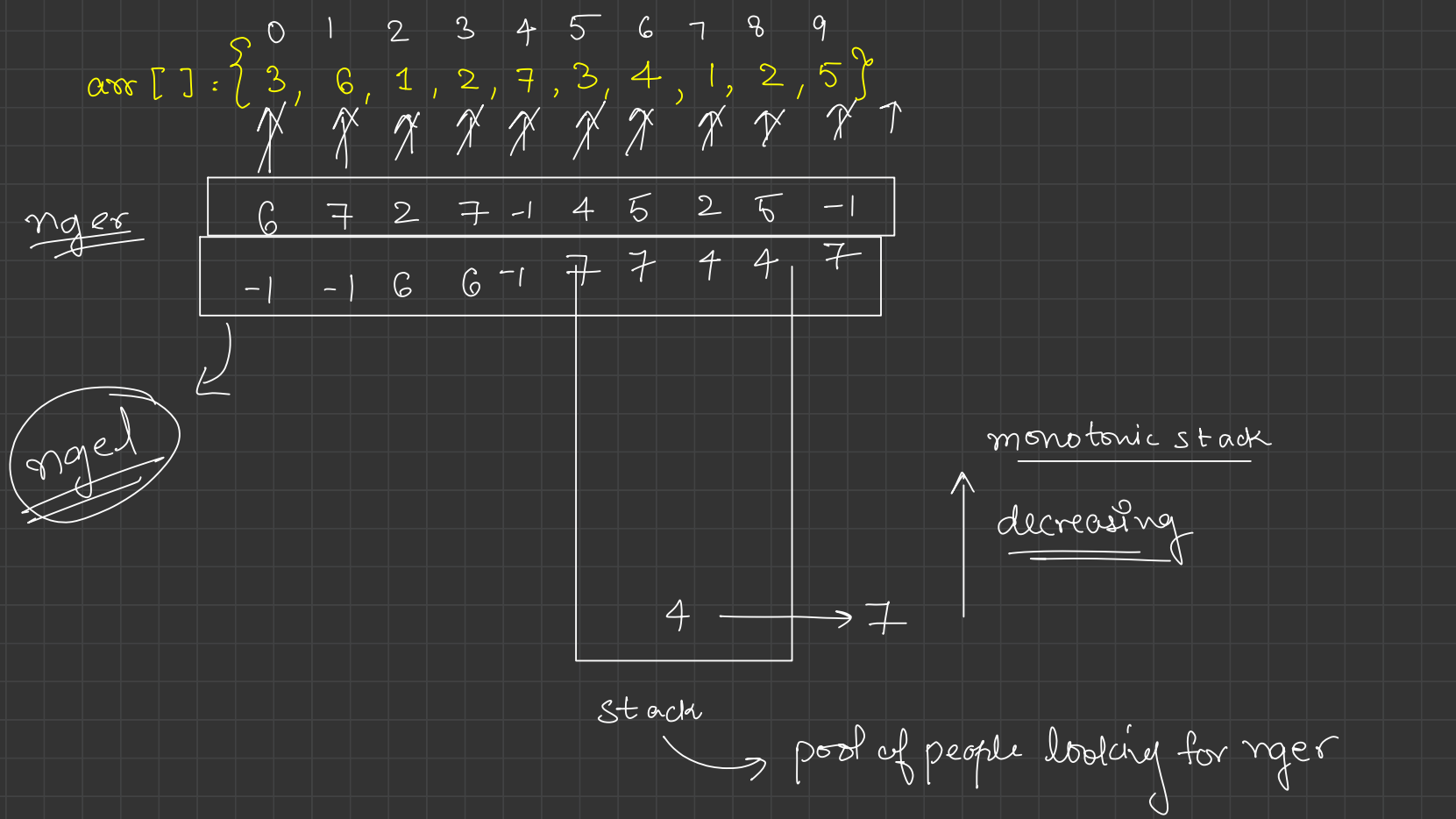


Stack

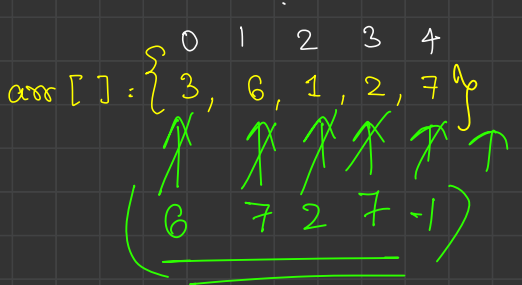


$$\sum_{i} (a+b+c+\cdots+\alpha) = 2N$$

$\hookrightarrow$ $O(N)$


0 1 2 3 4 5 6 7 8 9
arr[]: { 3, 6, 1, 2, 7, 3, 4, 1, 2, 5 }
nger
6 7 2 7 -1 4 5 2 5 -1
-1 -1 6 6 -1 7 7 4 4 7
nged
monotonic stack
decreasing
4 → 7
stack
pool of people looking for nger

```java
public static long[] nextLargerElement(long[] arr, int n)
{
    //Write code here and print output

    // Stack: people looking for next greater element on right
    Stack<Integer> st = new Stack<>();

    long[] nger = new long[n];

    // left to right
    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;
}
```



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