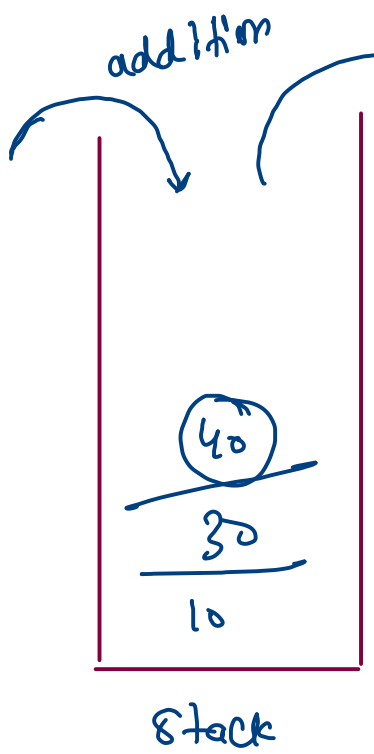
Usage → Stack → LIFO

↳ Last In First Out



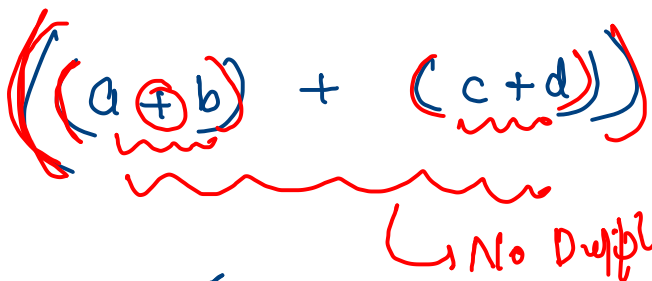
push → 10
push → 20
push → 30
push → 40
peek() → 40
pop()
push (60).
pop() → 60
pop() → 30
pop() → 20
push() → 30

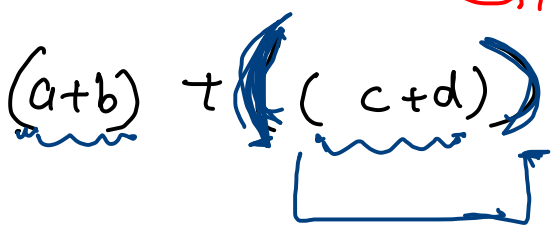
Stack

1. push → add
2. pop → remove
3. peek → get
4. size → size.

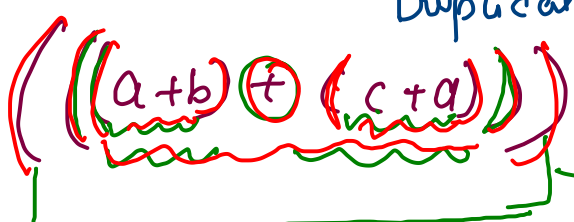
Duplicate Brackets → present → True
absent → false

((a + b) + (c + d)) -> false
(a + b) + ((c + d)) -> true





Duplicate bracket
→ True.



(a+b) + (c+d) ||→

(a+b) + (c+d))] Invalid Expression

(a+b) + ((c+d)) ||→

(((a+b) + (c))) ||→

((a+b) + c] Invalid Expression

(a+b)) + ((c+d)] Invalid Expression

((a + b) + (c + d)) -> false

(a + b) + ((c + d)) -> true

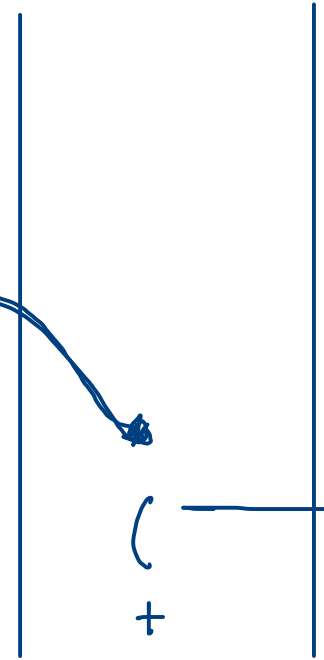
what ?

why?

How?

operator
operand
opening bracket
push

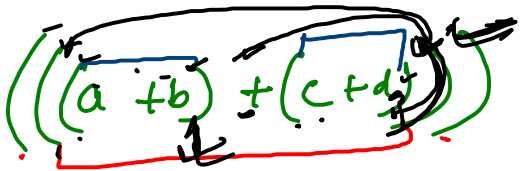
closing bracket

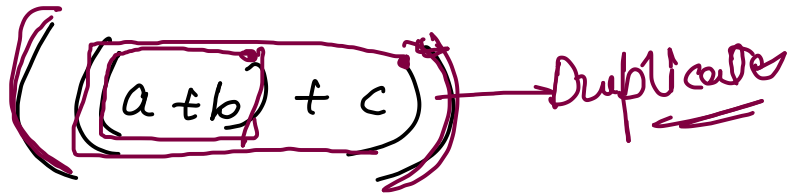


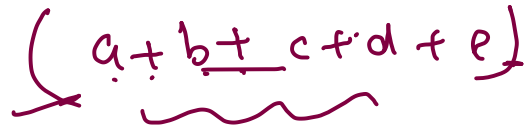
Duplicacy

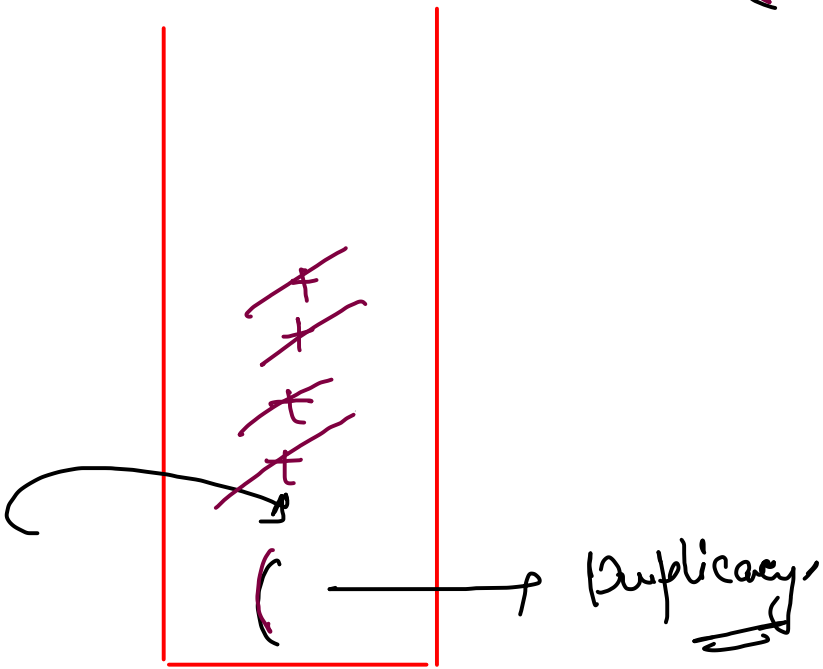
closing bracket → pop until opening bracket is not encounter

again pop → opening brac.


(a +b) + (c+d)


(a +b) + c
Duplicate


a + b + c + d + e


+
+
+
+
(
Duplicacy

Balanced brackets.

[(a+b)+{(c+d)*(e/f)}] TRUE

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

[(a+b)+{(c+d)*(e/f)} → False.

([a+b])]+b

opening → push

Operator, operand → skip

closing → counter opening bracket

st.size()==0 → return false

if yes → pop

otherwise → return false

out loop - stack.size() > 0 → Remaining op. bracket

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

Next Greater Element

allowed complexity - O(n)

Next greater

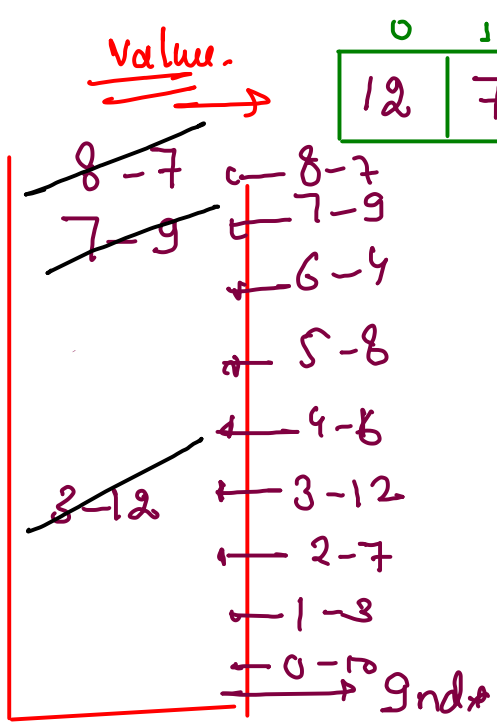
0	1	2	3	4	5	6	7	8
10	3	7	12	6	8	4	9	7

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

What ?

Why ?

How ?



Index _ value.

arr[st.peek()] > arr[i]

push →

else {

pop → until valid

push()

}

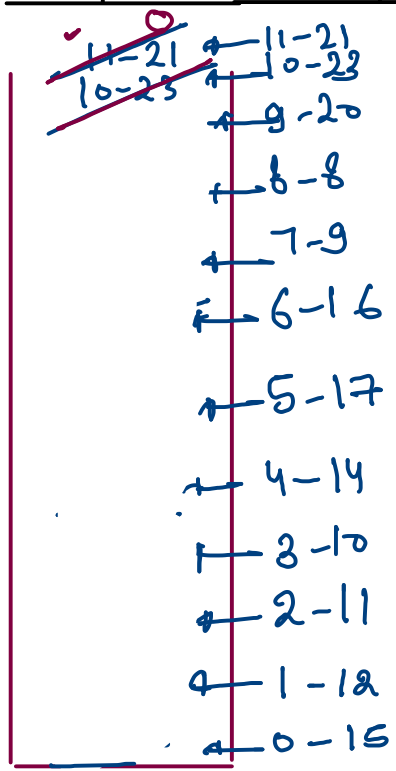
size() > 0

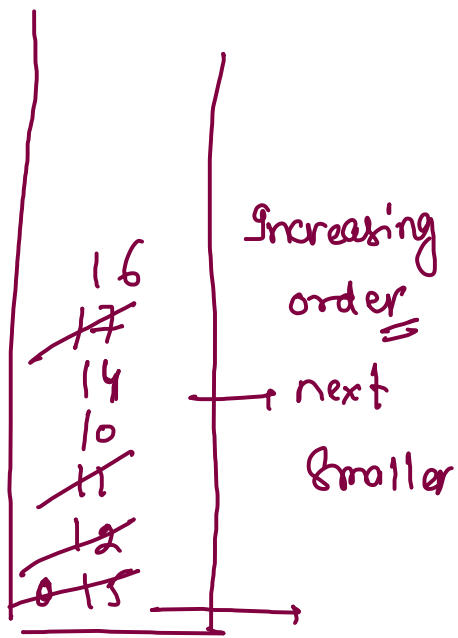
&&

arr[st.peek()] < curr

15	12	11	10	14	17	16	9	8	20	23	21
0	1	2	3	4	5	6	7	8	9	10	11

17	14	14	14	17	20	20	20	20	23	-1	-1



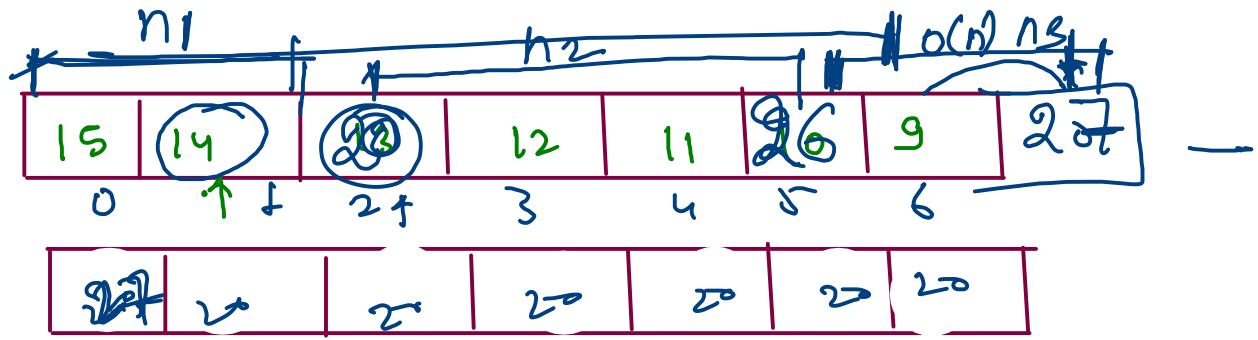


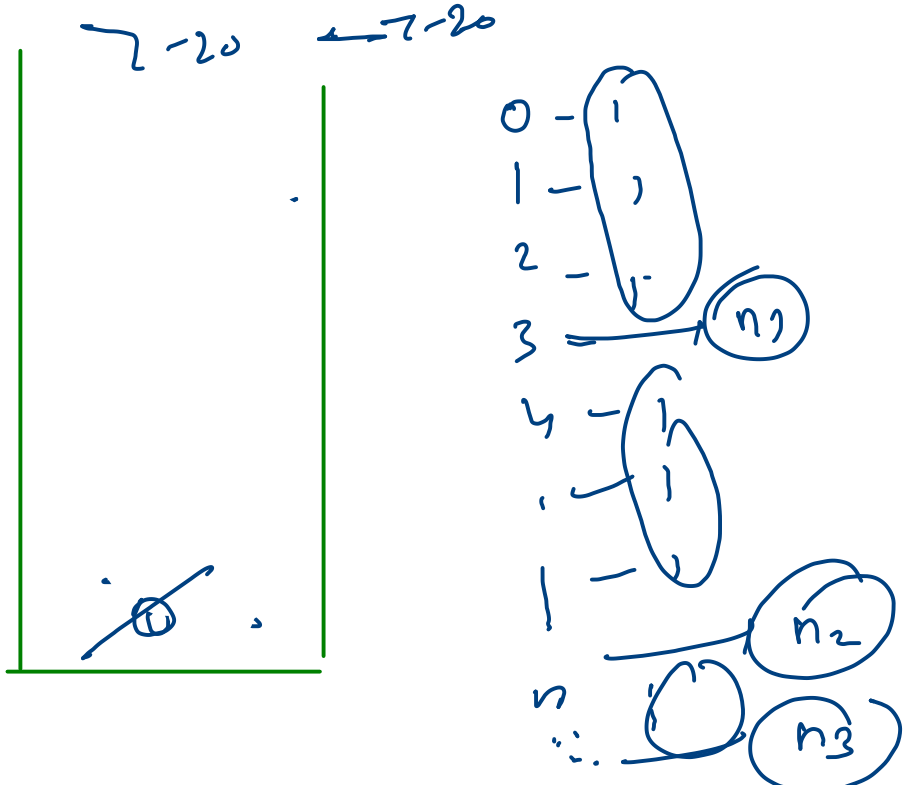
push → O(1)

pop → O(1)

peek → O(1)

size → O(1)

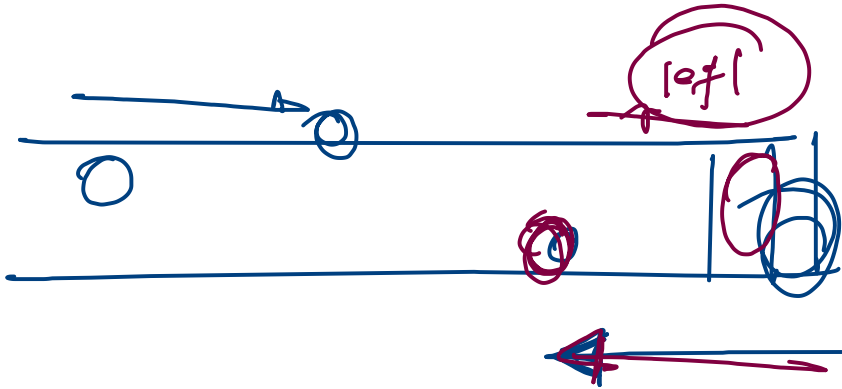


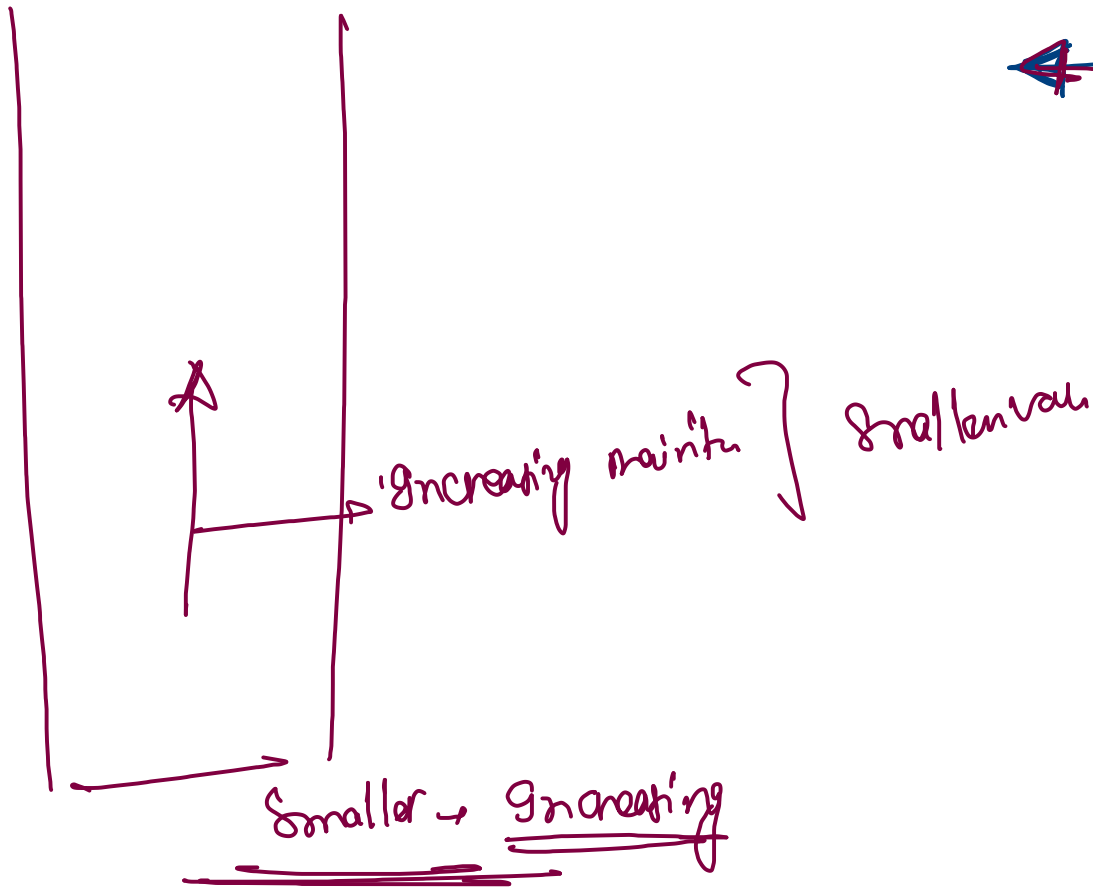


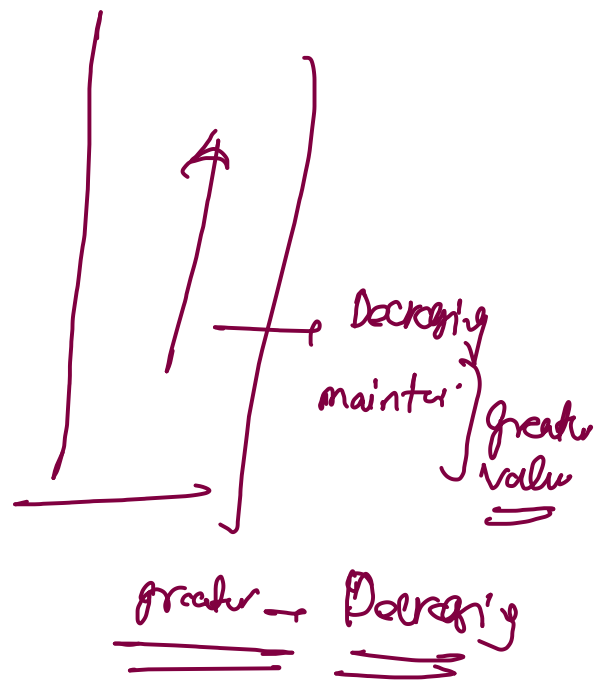

```java
public static int[] ngr(int[] arr) {
    Stack<Integer> st = new Stack<>();
    st.push(0);
    int[] res = new int[arr.length];
    for(int i = 1; i < arr.length; i++) {
        while(st.size() > 0 && arr[st.peek()] < arr[i]) {
            res[st.pop()] = arr[i];
        }
        st.push(i);
    }
    while(st.size() > 0) {
        res[st.pop()] = -1;
    }
    return res;
}
```

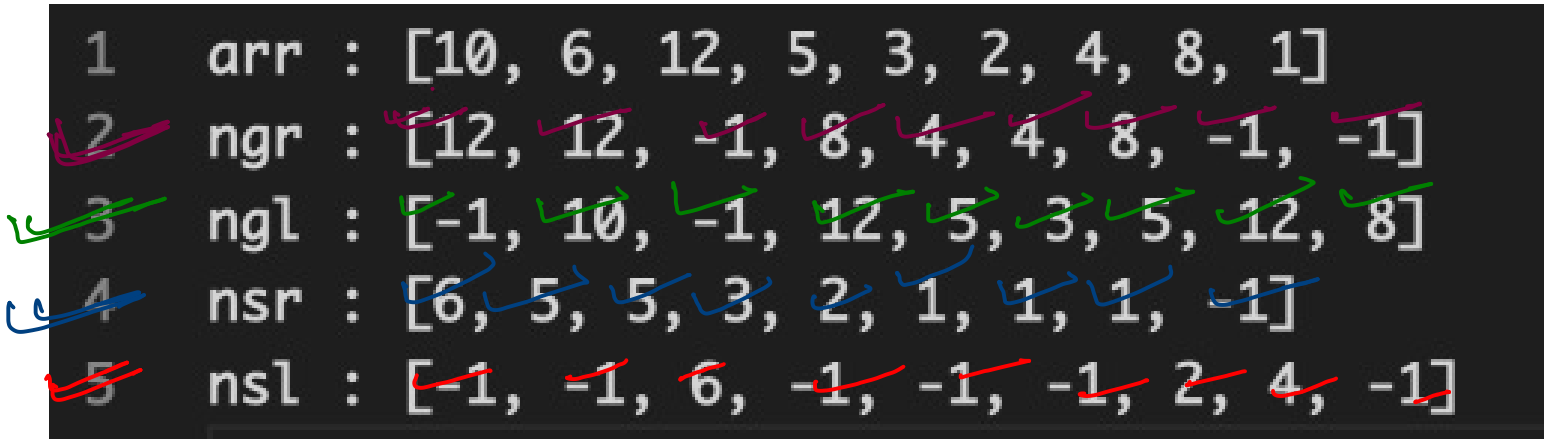
n + n1 + n2 + n3 = O(2n) = O(n)

next greater on left

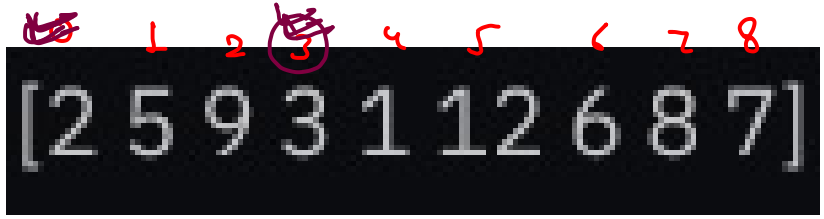







arr : [10, 6, 12, 5, 3, 2, 4, 8, 1]
ngr : [12, 12, -1, 8, 4, 4, 8, -1, -1]
ngl : [-1, 10, -1, 12, 5, 3, 5, 12, 8]
nsr : [6, 5, 5, 3, 2, 1, 1, 1, -1]
nsl : [-1, -1, 6, -1, -1, -1, 2, 4, -1]

next greater on left (index)



nglndx →

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

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