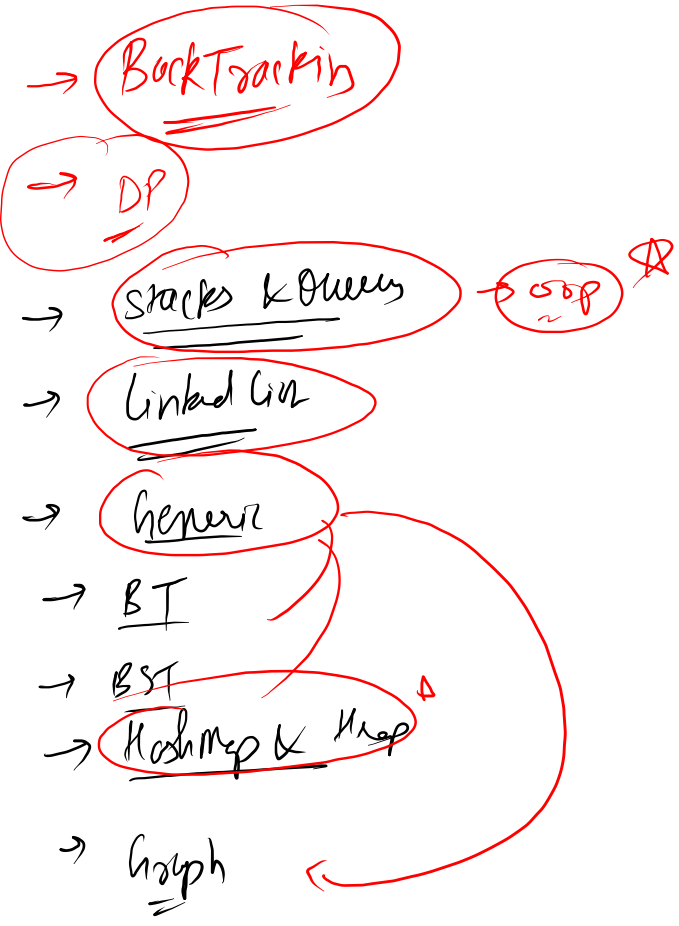

BackTracking
DP
Stacks & Queues
OOP
Linked List
Generics
BT
BST
HashMap & Map
Graph

Stacks → LIFO

Declaration

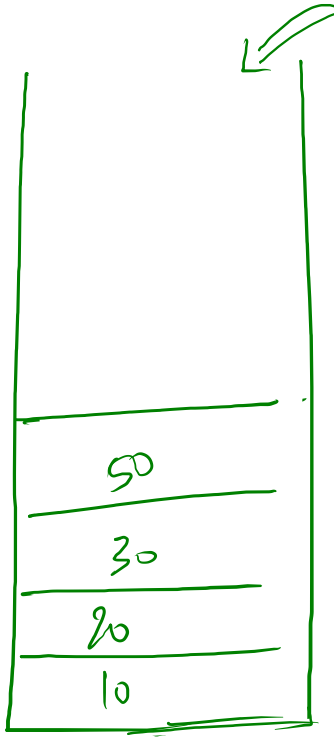
Stack < > st = new Stack<>();

Implementation
- Array
- LL

Operations → pop, push, peek, size

→ Core Sides

NO direct ACCESS



push 10 ✓
push 20 ✓
peek ✓ 20
push 30 ✓
push 40 ✓
pop → 40
peek → 30
push 50 →
peek ✓ 50
size → 4

Duplicate Brackets

↳ Needless Brackets

((a+b)) → True

0 1 2 3 4 5 6

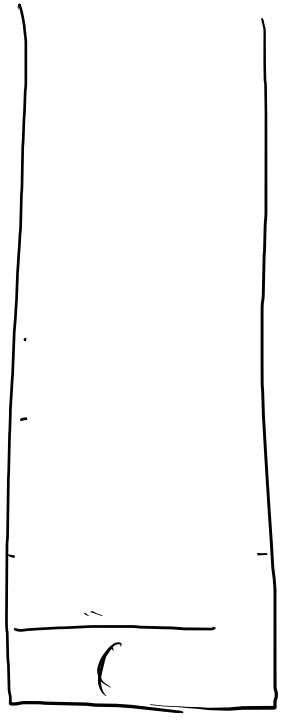
(a+b) → false

() → True

(a) → false

a+((b+c)) → true

(a+(b+c)) → false



ch = ')'

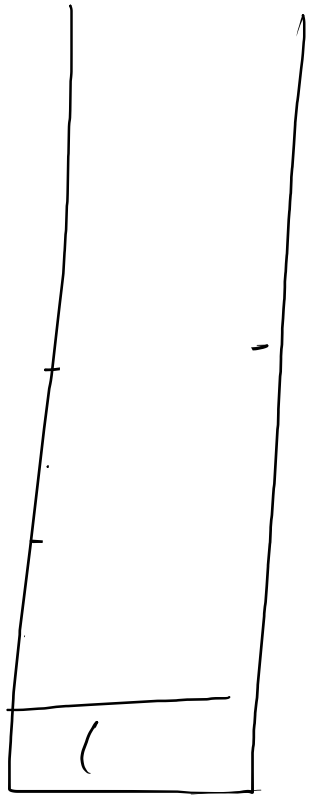
True

```
Stack<Character> st = new Stack<>();
for( idx=0 ; idx < str.length() ; idx){
    ch ← str.charAt(idx);
    if(ch == ')'){
    }else{
        push
    }
}
```

((a+b))

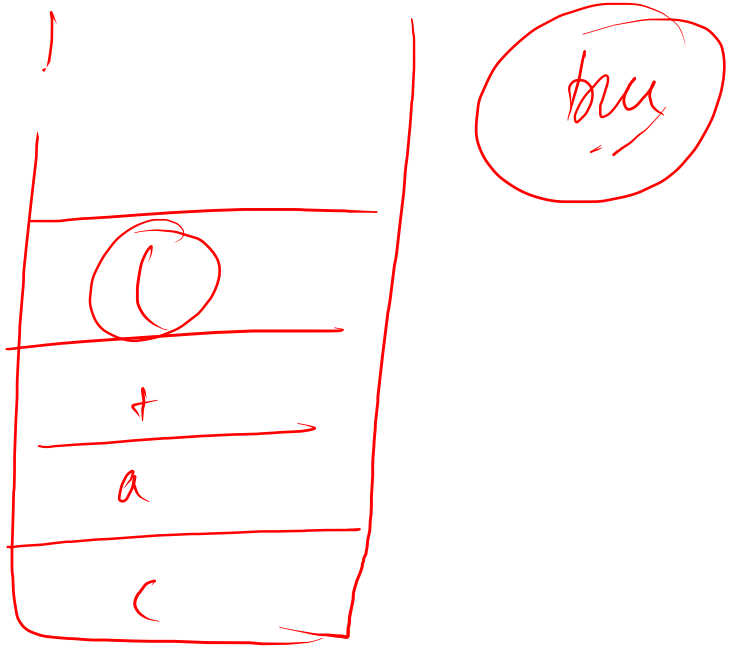
0 1 2 3 4 5 6

true




```
String exp = scn.nextLine();

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

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

    if(ch == ')'){
        if(st.peek() == '('){
            System.out.println("true");
            return;
        }

        while(st.peek() != '('){
            st.pop();
        }
        st.pop();
    }else{
        st.push(ch);
    }
}

System.out.println("false");
```

(a + (b + c))



Balanced Brackets

(), {}, []

1. No. of O.B. = no. of C.B.
2. Mismatch

(), {}, []

Exp → meaning

→ ✗

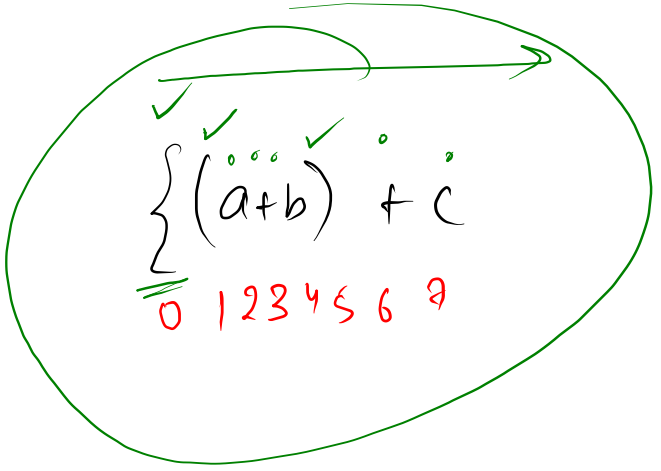
[{(a+b) + c} + d] ⇒ True

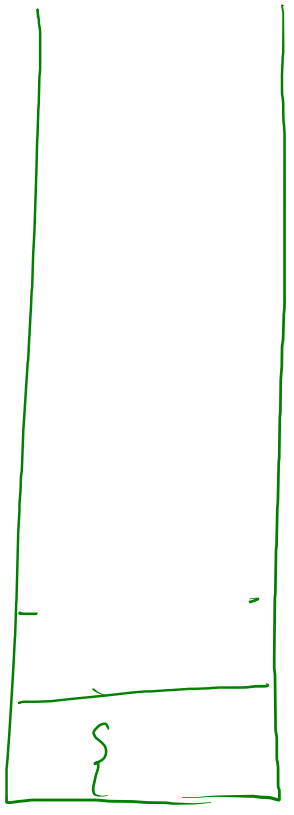
((_)} ⇒ false

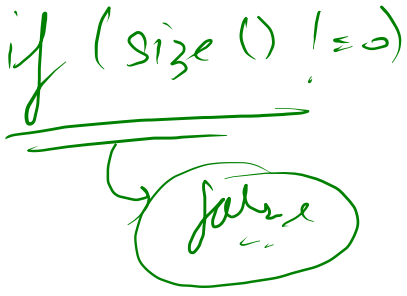
[{(a+b) + c} + d ⇒ False

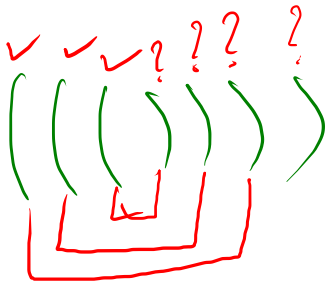
{(a+b)}] ⇒ false

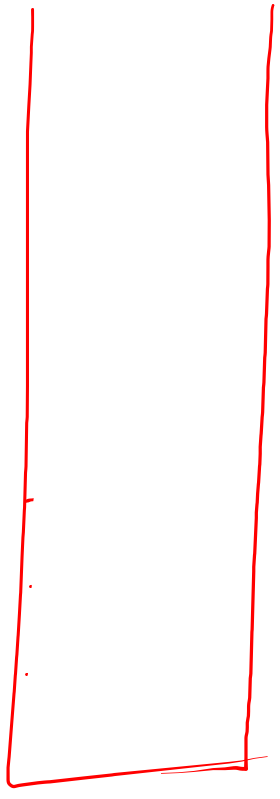
{(a+b}) ⇒ false


{(a+b) + c
0 1 2 3 4 5 6 7

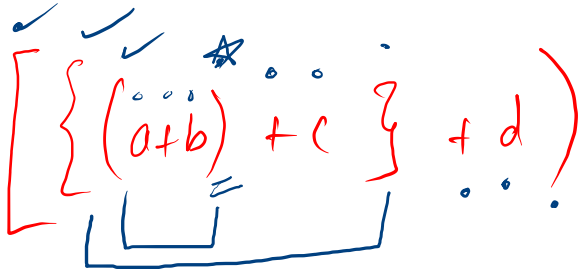

{

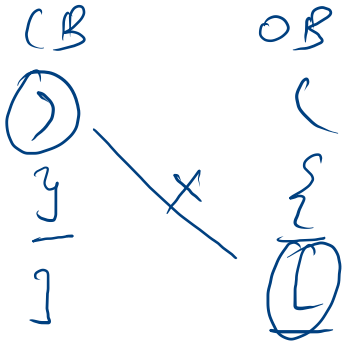

if (size() != 0)
false

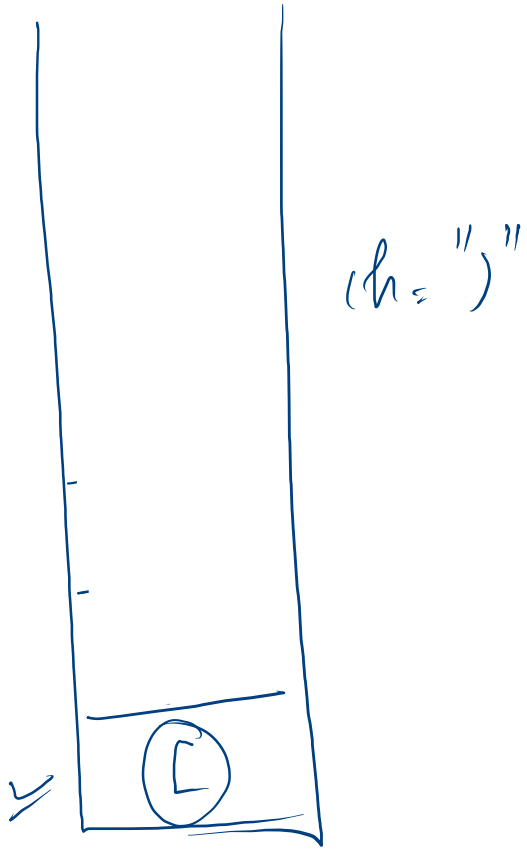




ch = ')'

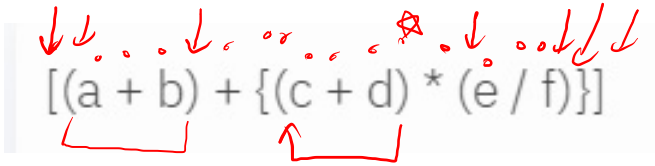


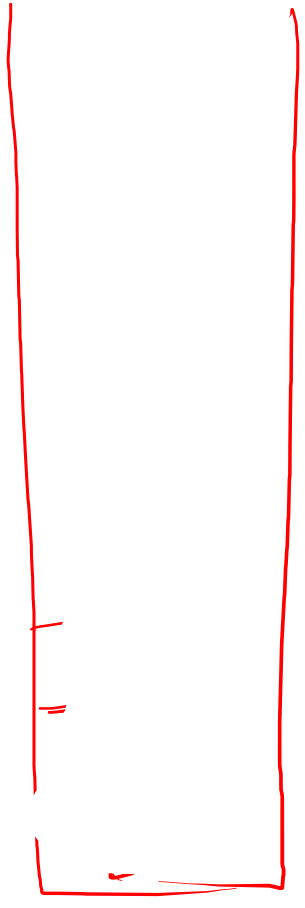


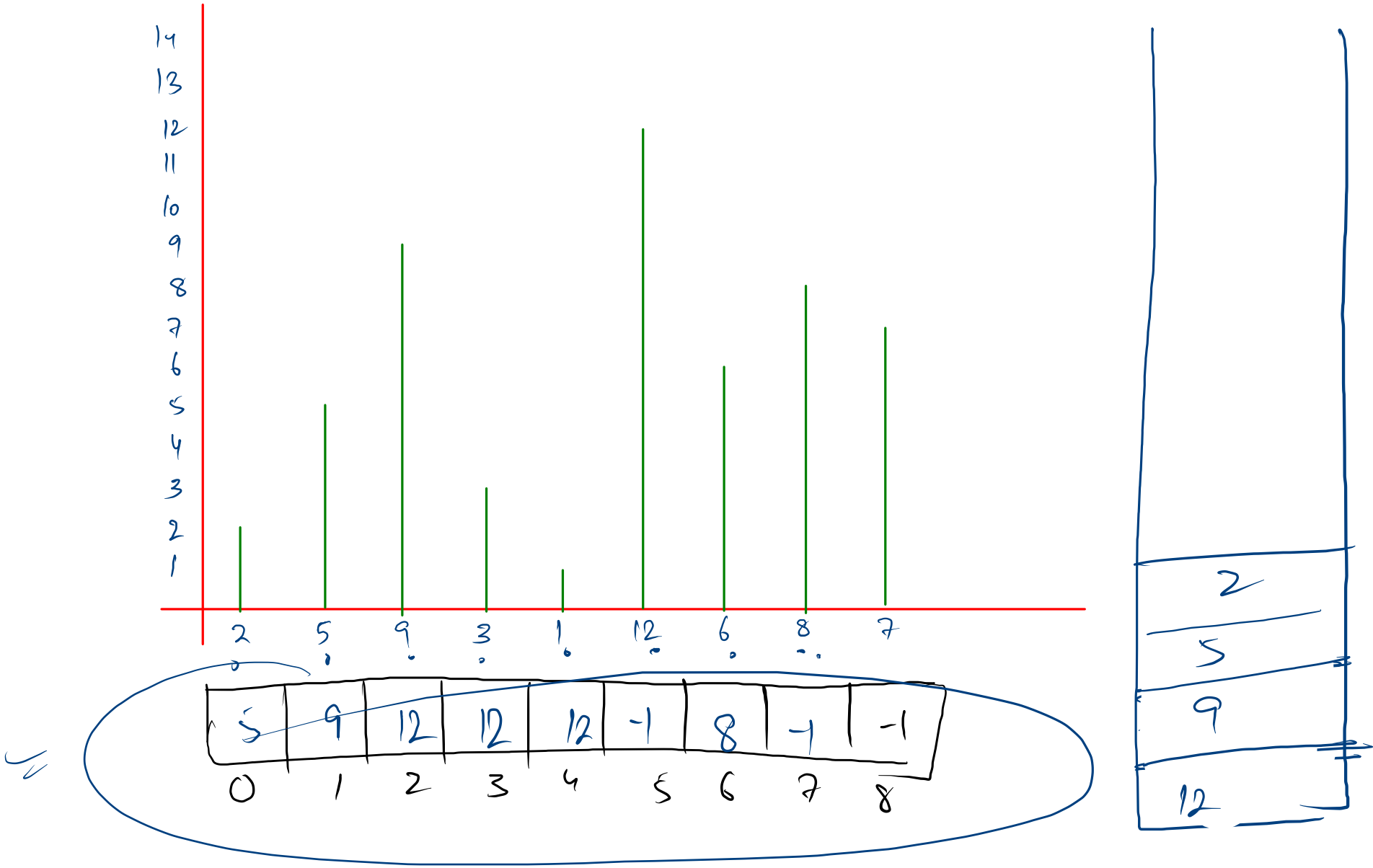


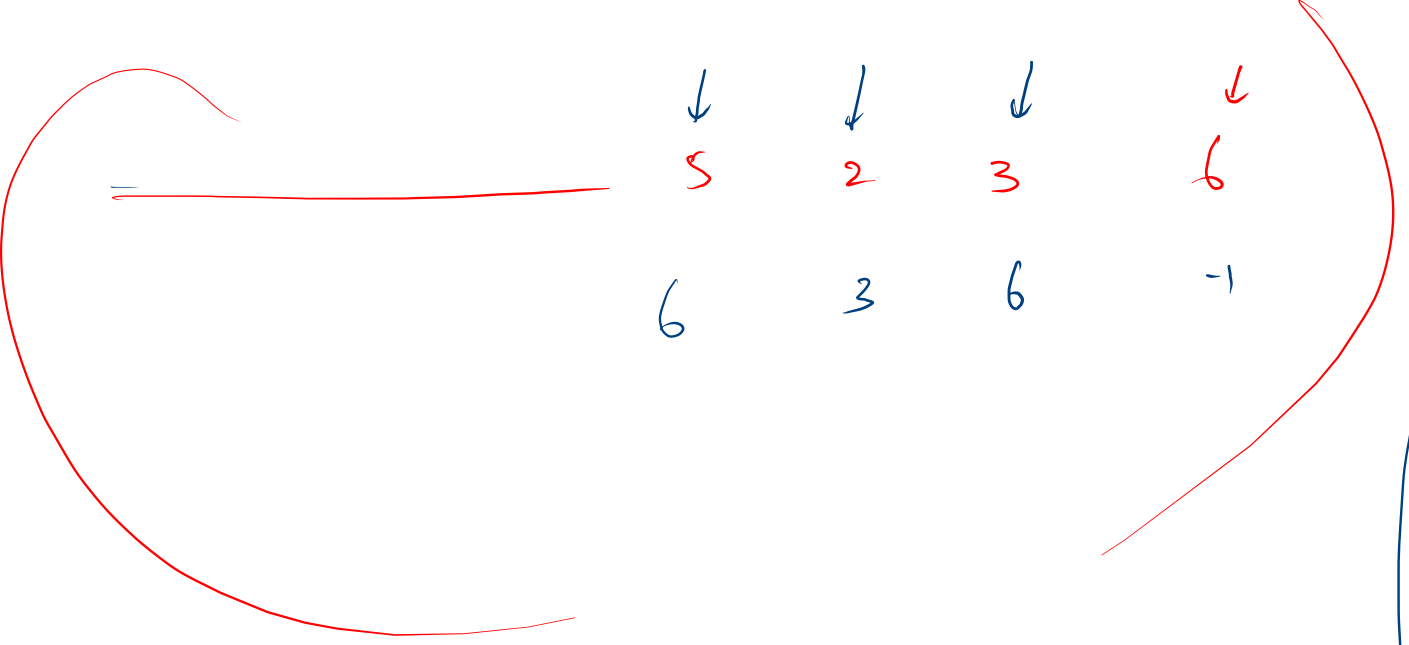

```
for(int idx = 0 ; idx < exp.length() ; idx++){
    char ch = exp.charAt(idx);

    if(ch == '(' || ch == '{' || ch == '['){
        st.push(ch);
    }else if(ch == '}' || ch == ')' || ch == ']'){
        if(st.size() == 0){
            return false; // nOB < nCB
        }
    }
}

if(st.size() != 0){
    return false; // nOB > nCB
}
```

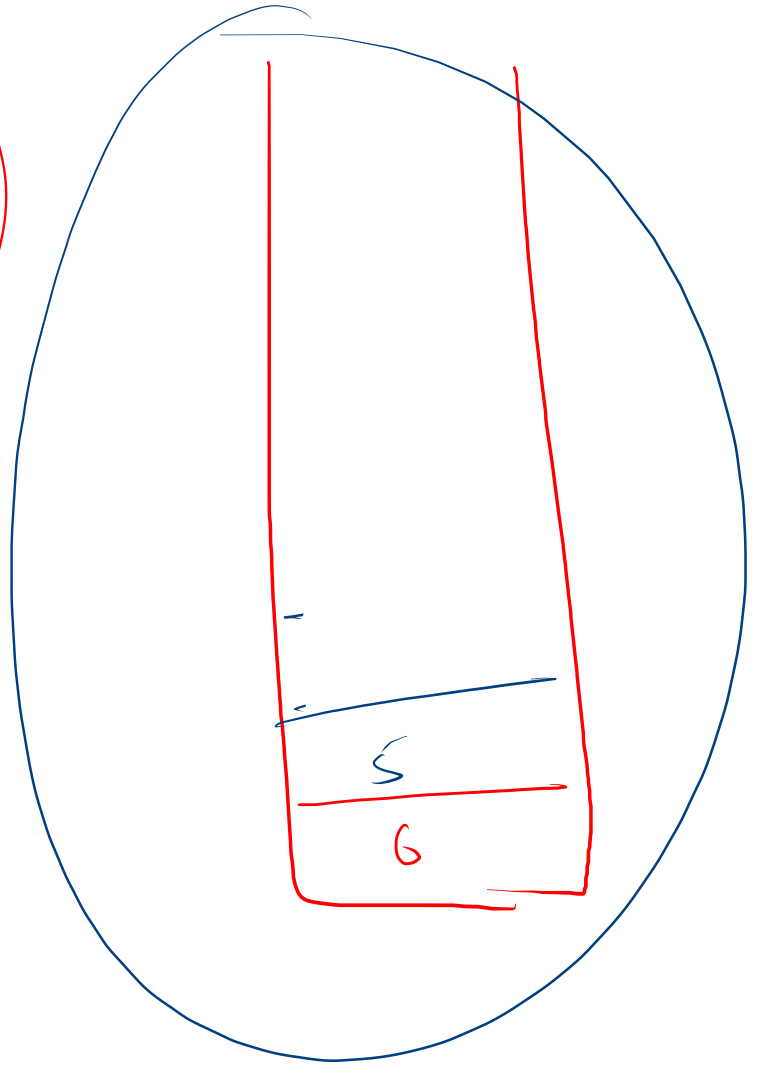
```java
public static boolean isBalanced(String exp){
    Stack<Character> st = new Stack<>();

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

        if(ch == '(' || ch == '{' || ch == '['){
            st.push(ch);
        }else if(ch == '}' || ch == ')' || ch == ']'){
            if(st.size() == 0){
                return false; // nOB < nCB
            }

            if(ch == ')' && st.peek() != '('){
                return false;
            }

            if(ch == '}' && st.peek() != '{'){
                return false;
            }

            if(ch == ']' && st.peek() != '['){
                return false;
            }

            st.pop(); // opening bracket
        }
    }

    if(st.size() != 0){
        return false; // nOB > nCB
    }

    return true;
}
```

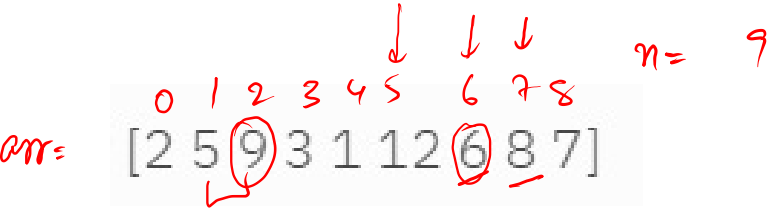


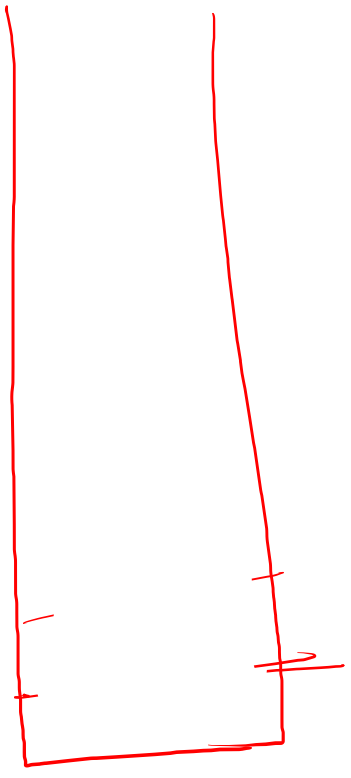



14
13
12
11
10
9
8
7
6
5
4
3
2
1
2 5 9 3 1 12 6 8 7
5 9 12 12 12 -1 8 -1 -1
0 1 2 3 4 5 6 7 8
2
5
9
12


5 2 3 6
6 3 6 -1


5
6

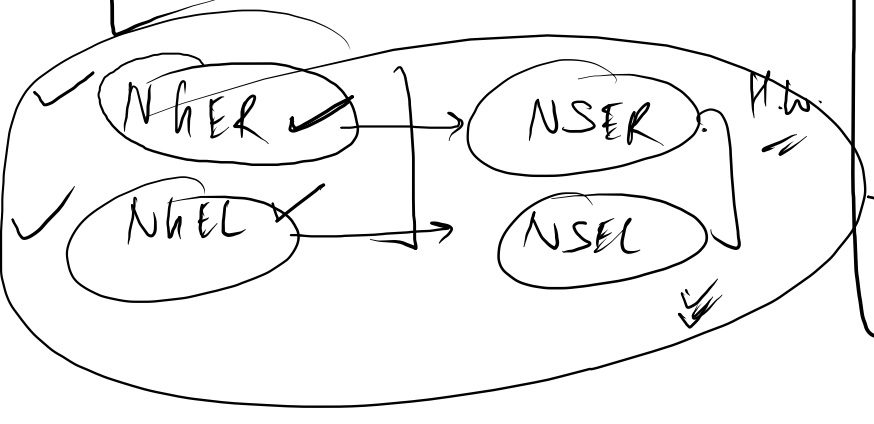


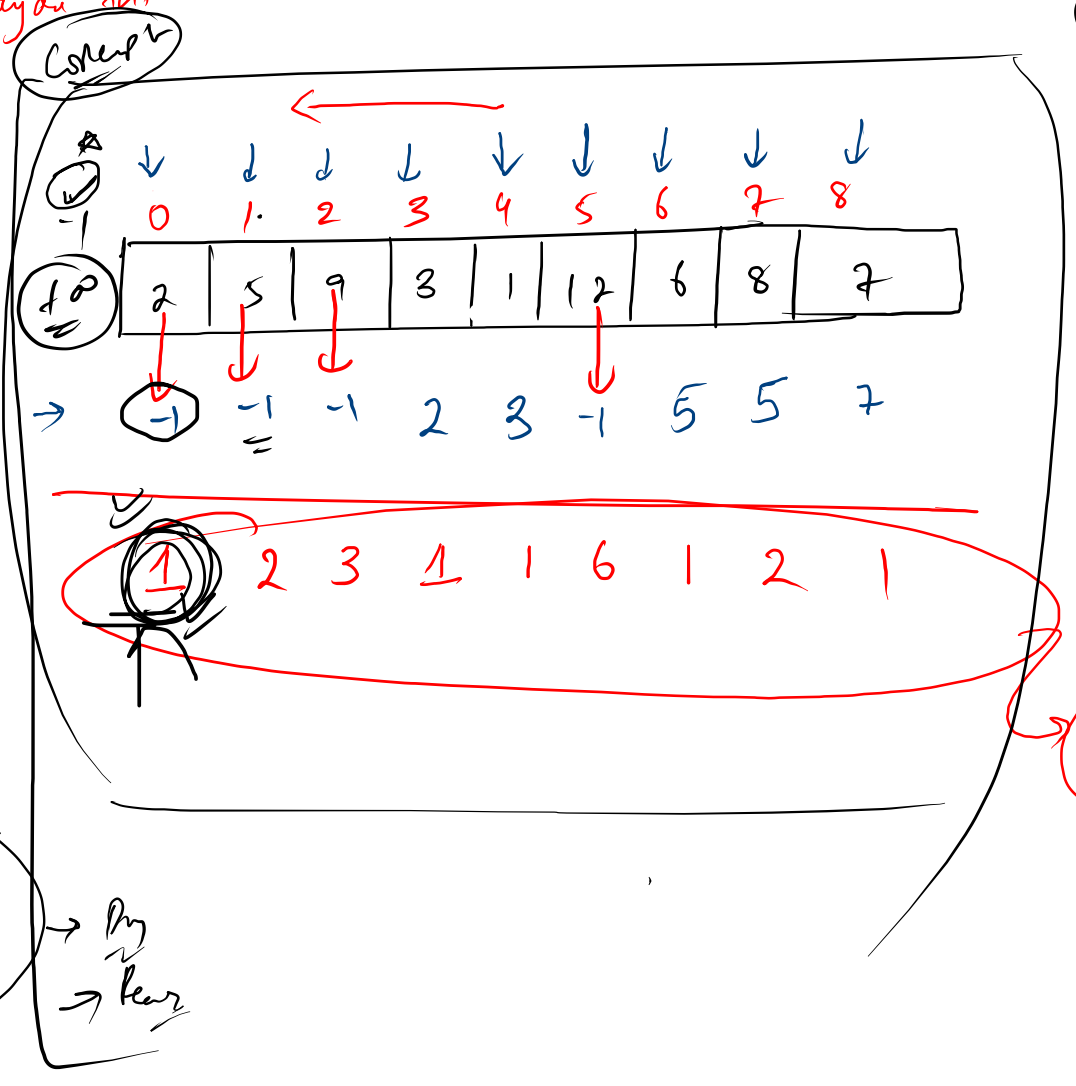


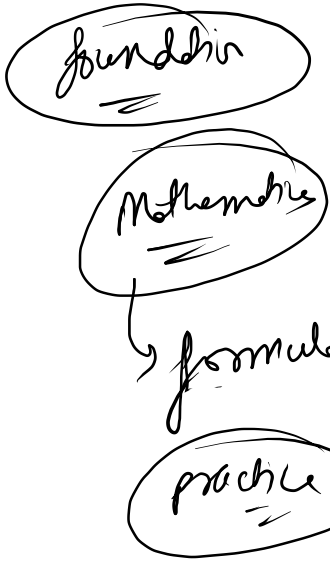
```java
public static int[] solve(int[] arr){
  int n = arr.length;
  Stack<Integer> st = new Stack<>();
  st.push(arr[n-1]);
  int res[] = new int[n];
  res[n-1] = -1;

  for(int idx = n-2 ; idx >= 0 ; idx--){
   while(st.size() > 0 && arr[idx] >= st.peek()){
       st.pop();
   }

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

   st.push(arr[idx]);
  }
  return res;
```

Kitne din pehle humari value jayda thi

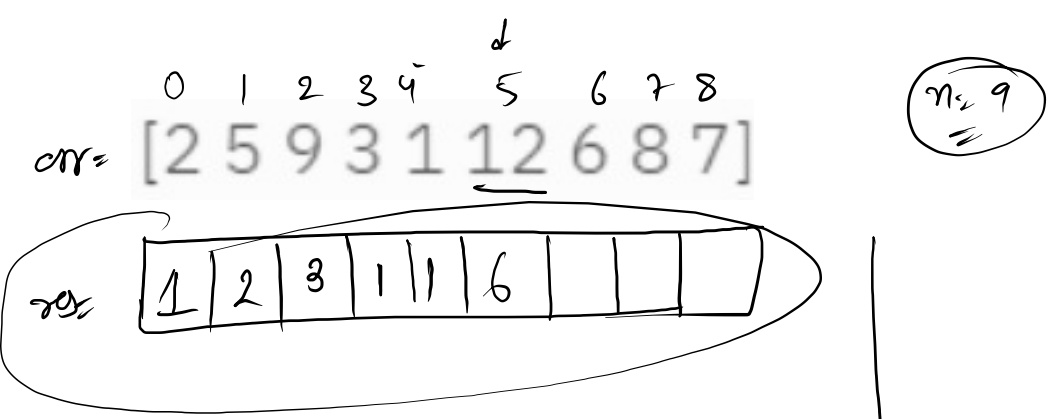
for the array [2 5 9 3 1 12 6 8 7]
span for 2 is 1
span for 5 is 2
span for 9 is 3
span for 3 is 1
span for 1 is 1
span for 12 is 6
span for 6 is 1
span for 8 is 2
span for 7 is 1

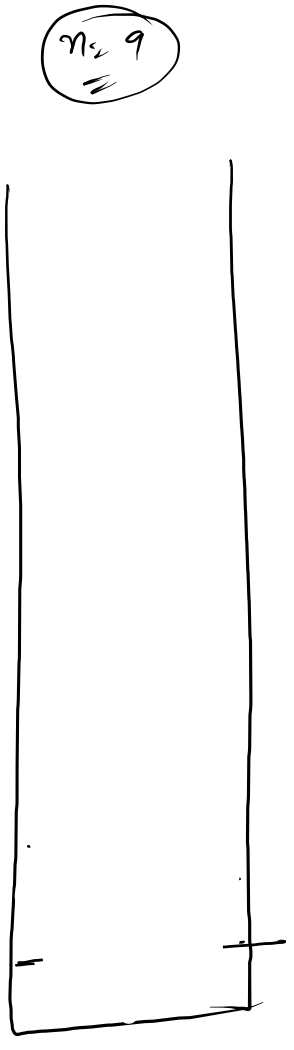




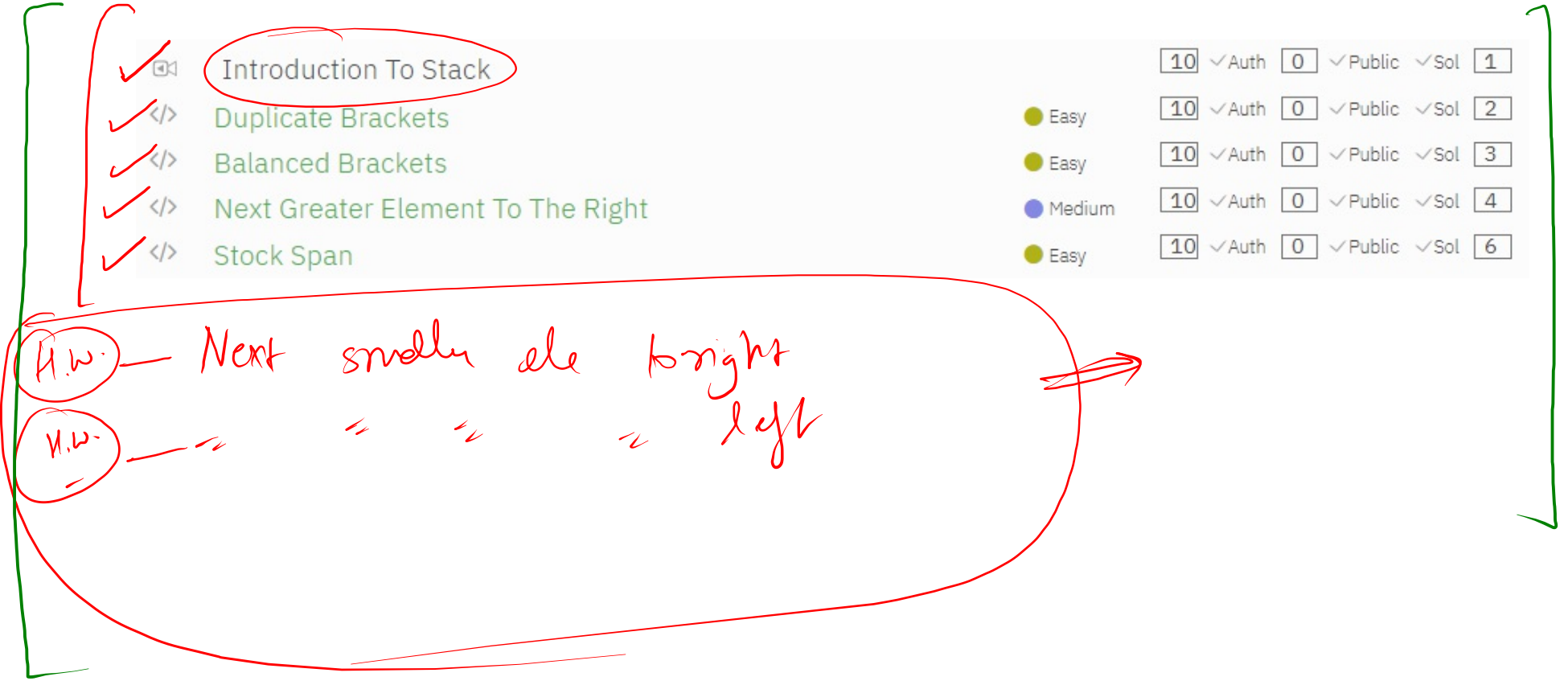


Stock Span






```java
public static int[] solve(int[] arr){
    int n = arr.length;
    Stack<Integer> st = new Stack<>();
    st.push(0);
    int res[] = new int[n];
    res[0] = 1;

    for(int idx = 1; idx < n ; idx++){
        while(st.size() > 0 && arr[idx] >= arr[st.peek()]){
            st.pop();
        }

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

        st.push(idx);
    }
    return res;
}
```


Introduction To Stack
10 Auth 0 Public Sol 1
Duplicate Brackets
Easy
10 Auth 0 Public Sol 2
Balanced Brackets
Easy
10 Auth 0 Public Sol 3
Next Greater Element To The Right
Medium
10 Auth 0 Public Sol 4
Stock Span
Easy
10 Auth 0 Public Sol 6
H.W. — Next smaller ele to right
H.W. — " " " " left