Stacks [Question + LeetCode Questions]

T W T F S S

Page No.:

Date:

YOUVA

① Balanced Brackets. {Check sequence is balanced or Not.}

~~Ex ①~~ → str = "()(())" → True

str = "(()(()" → False.

str = ")()())" → False

str = "(()))())" → False

Rules :-

Ex ① → str = "()(())"

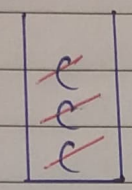
① Opening → push

② Closing

(a) st top → '(' → pop

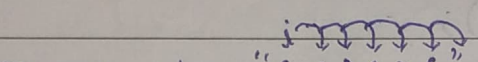
(b) if st is empty - return false.

st

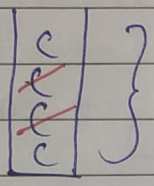


if st. is empty, they are balanced → True.

Ex ② → str = "(()()("



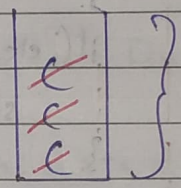
st



if st.size() != 0 return false.

Ex ③ → str = "(()))())"

st



if (st.size() == 0) return false.

⇒ Code :-

```
publis static boolean isBalanced(String str){
    Stack<Character> st = new Stack<>();
    int n = str.length();
    for(int i = 0; i < n; i++){
        Char ch = str.charAt(i);
        if(ch == '('){                          } → Rule ①.
            st.push(ch);                        }
        }
        else{
            if(st.size() == 0)                  } → Ex ③.
                return false;                   }
            if(st.peek() == '(')                } → Rule ②(b).
                st.pop();                       }
        }
    }
    if(st.size() > 0) return false;             } → Ex ②.
    else return true;                           } → Ex ①.
}
```

Follow-up.

② LeetCode Q. No. 20 { Valid Parentheses }

→ Similar to previous question. There is only one difference instead of only normal parenthesis ("(,)") it also include Square, & Curly brackets. ("[]" "{}").

=> Code ::

```
public boolean isValid(String s){
    Stack<Character> st = new stack<Character>();
    int n = s.length();
    for(int i = 0; i<n; i++){
        Char ele = s.charAt(i);
        if(ele == '('){
            st.push(')');
        }
        if(ele == '{'){
            st.push('}');
        }
        if(ele == '['){
            st.push(']');
        }
        else if(st.isEmpty() || st.pop() != ele){
            return false;
        }
    }
    return st.isEmpty();
}
```

③ Remove Consecutive Subsequences. # Using Stack.

Ex① => arr => { 1 2 2 3 10 10 10 4 4 4 5 7 7 2 } Remove this

=> { 1 3 5 2 }

2
~~7~~
5
~~4~~
~~10~~
3
~~2~~
1
st

```
if(arr[i] == arr[i+1]){
    do Nothing;
}
if(st.size()==0 || st.peek()!=arr[i])
    st.push(arr[i]);
else if(arr[i] == st.peek()){
    if(i==n-1 || arr[i]!=arr[i+1]){
        st.pop();
    }
}
```

=> Code :- public static int[] remove(int[] arr){

```
int n = arr.length;
Stack<Integer> st = new Stack<>();
for(int i = 0 ; i < n ; i++){
    if(st.size() == 0 || st.peek() != arr[i]){
        st.push(arr[i]);
    }
    else if(arr[i] == st.peek()){
        if(i == n-1 || arr[i] != arr[i+1]){
            st.pop();
        }
    }
}
```

Follow up.

(24) Leet Code Q. No. 1047 {Remove All Adjacent Duplicates in String}

=> ~~Static~~ Slight difference is there.

Ex (1) -> str = "abbaca"
↳ "ca"

Ex (2) str = "azzxxy"
↳ ay

Have to remove in pairs.

Ex (3) -> str = "abbbabaaa"
↳ "ababa"

Ex (4) str = "aaaaaaaaa"
↳ "a"
only one remain

a
~~a~~
b
a
b
~~b~~
a

str = "abbbabaaa"
i i i i i i i i i

~~for~~ if(st.size() == 0 ~~&&~~)
st.push(i);
if(st.peek() != ~~s.ch~~ s.(i))
st.push(i);
else st.pop();

=> Code :-

```
public string removeDuplicates(String s){
    int n = s.length();
    Stack<Character> st = new Stack<>();
    for(int i = 0 ; i < n ; i++){
        if(st.size() == 0 || st.peek() != s.charAt(i)){
            st.push(s.charAt(i));
        }
        else st.pop();
    }
}
```

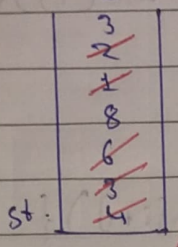
Follow-UP

(5) Next Greater Element

Ex ①

arr = {1, 3, 2, 1, 8, 6, 3, 4}

→ res = {3, 8, 8, 8, -1, -1, 4, -1}



Code :-

```
st.push(arr[n-1]);
for(int i = n-2 ; i >= 0 ; i--){
    while(st.size()>0 && st.peek()<arr[i]){
        st.pop();
    }
    if(st.size()==0)
        res[i] = -1;
    else
        res[i] = st.peek();
    st.push(arr[i]);
}
```

res[n-1] = -1.

---

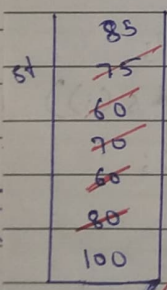
(6) Previous Greater Element : [Same as Greatest Ele]

Ex ①

arr = {100, 80, 60, 70, 60, 75, 85}

→ res = {-1, 100, 80, 80, 70, 80, 100}

⇒ Code :-




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

Follow up

This time we need to keep indexes in our mind.

⑦ Gfg :- Stock Span Problem. {Logic same as Previous Greater Ele}

	0	1	2	3	4	5
arr :-	100	80	60	70	60	75	85
res :-	1	1	1	2	1	4	6

st: 5, 4, 3, 2, 1, 0

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

Same Question on Leetcode Q.No. 901
have to submit it in little different way.

=> Code :-

```
Class StockSpanner {
    Stack<int*[]> st ;
    int ind ;
    public StockSpanner() {
        st = new stack<>() ;
        ind = -1 ;
    }

    public int next (int price){
        ind++ ;
        while (!st.isEmpty() && st.peek()[1] <= price){
            st.pop();
        }
        int ans = ind - (st.isEmpty() ? -1 : st.peek()[0]);
        st.push(new int[]{ ind , price});
        return ans;
    }
}
```

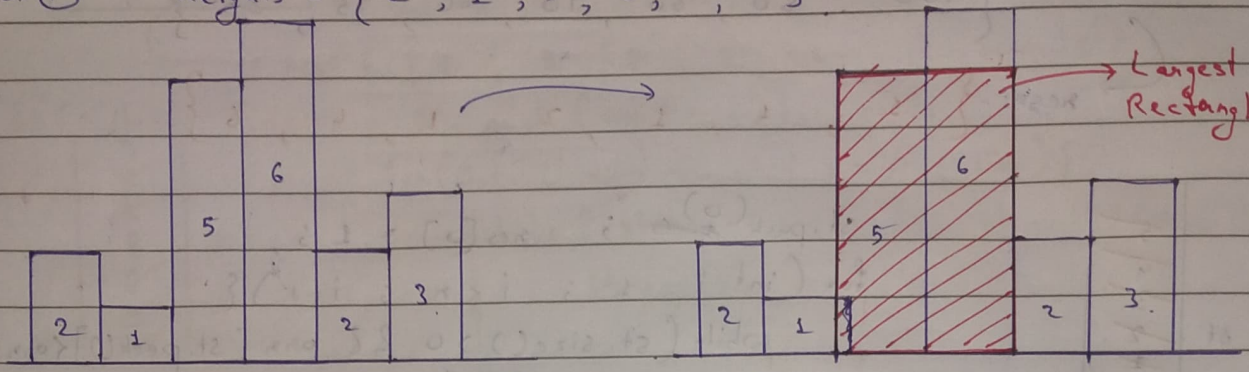
st: 75,5 | 60,4 | 70,3 | 60,2 | 80,1 | 100,0

pair of stack. we use it to store both 2 values & index

(8) Leet Code Q.No. 84. {Largest Rectangle in Histogram}

Page No.: Date: YOUVA

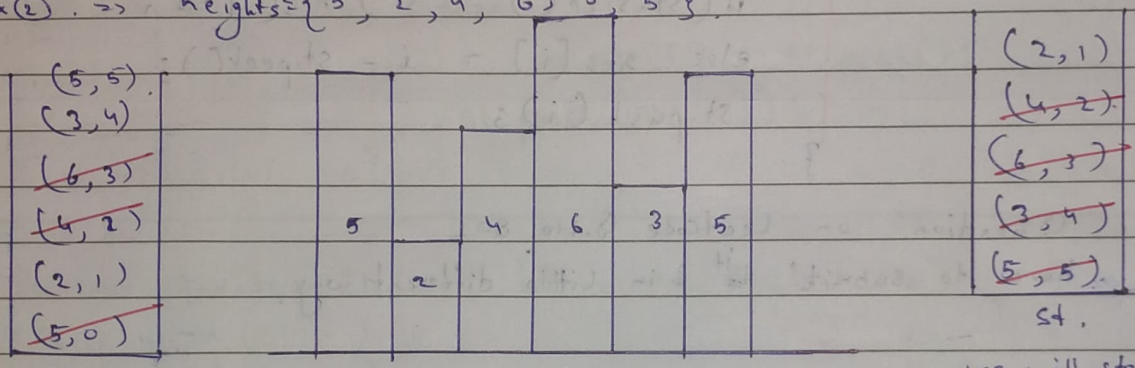
Ex (1) heights = {2, 1, 5, 6, 2, 3} * widths are same



Logic => find Nse & Pse

(Next Smallest Ele) (Previous Smallest ele)

Ex (2) => heights = {5, 2, 4, 6, 3, 5}



St ind => 0 1 2 3 4 5

we will store ~~st~~ ind.

Nse => 1 6 4 4 6 (6) } instead of -1 we will put 'n'. Indexes.

Pse => -1 -1 1 2 1 4

area = height[i] * (nse[i] - pse[i] - 1);

=> Code :- ~~int~~ //Calculate Nse[].

```
st.push(n-1); // index
nse[n-1] = n;
for(int i = n-2 ; i >= 0 ; i--){
    while ( st.size()>0 && heights[st.peek()] >= heights[i]){
        st.pop();
    }
    if (st.size() == 0)
        nse[i] = n;
    else
        nse[i] = st.peek();
    st.push(i);
}
```

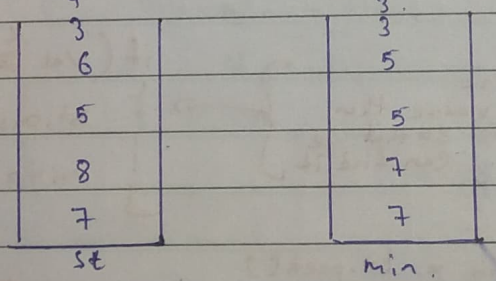
```
// Emptying stack.
while (st.size() > 0){
    st.pop();
}

// Calculate pse[]
st.push(0);
pse[0] = -1;
for (int i = 1 ; i < n ; i++){
    while (st.size() > 0 && heights[st.peek()] >= heights[i]){
        st.pop();
    }
    if (st.size() == 0)
        pse[i] = -1;
    else
        pse[i] = st.peek();
    st.push(i);
}

// Calculate Maxm Area of Rectangle.
int maxc = -1;
for (int i = 0 ; i<n ; i++){
    int area = heights[i] * (nse[i] - pse[i] - 1);
    max = Math.max(max, area);
}
Sout(max);
```

(9) LeetCode Q.No. :- 155 { Min Stack } ⟶ have to ~~Create~~ store push, pop, top, getMin method to store previous min ele to that perticular ele.

Approach - (1).

Ex (1) ⟶

4 7 8 5 6 3 4.



getmin() = O(1)

S.C = O(n).

=> Code :-

```
Class MinStack {
    Stack<Integer> st = new Stack<>();
    Stack<Integer> min = new stack<>();

public void push(int val){
    if(st.size() == 0){
        st.push(val);
        min.push(val);
    }
    else {
        st.push(val);
        if(min.peek() < val) min.push(min.peek());
        else min.push(val);
    }
}
```

S.C => O(n)

We have to to it in O(1).

```
public void pop(){
    st.pop();
    min.pop();
}

public int top() {
    return st.peek();
}

public int getMin(){
    return min.peek();
}
```

=> Approach - (2)

without extra space

* variable → one

push 4.	4
push 3*	-2
push 6	6
push 5*	-2
push 8	8
push 7	7

st.

we will store less value than min so that we can find it.

min = 3

=> pop(){

min - oldmin = st.peek()

oldmin = 2* min - st.peek()

```
if(val < min){
    st.push(val-min);
    min = val;
}
```

st.push(val-min);

val-min < val (Not Always)

2*val - min < val (Always) min = Oldmin (pop()).

st.push(2*val-min);

val + (val - min).

}

=> Code:-

```
Class Minstack {
    Stack<Long> st = new Stack<>();
    long min = -1;
        // Push Method
    public void push(int val){
        long x = (long) val;
        if(st.size() == 0){
            st.push(x);
            min = x;
        }
        else if(x < min){
            st.push(2*x - min);
            min = x;
        }
        else if(x >= min){
            st.push(x);
        }
    }

        // Pop Method.
    public void pop(){
        if(st.size() == 0) return;
        if(st.peek() >= min){
            st.pop();
        }
        else if(st.peek() < min){
            long old = 2*min - st.peek();
            min = old;
            st.pop();
        }
    }
```

~~S.C~~ => S.C => O(1).

```
    // GetMin Method.
    public int getMin(){
        if(st.size() == 0) return -1;
        return (int) min;
    }

    // Peek() Method.
    public int top(){
        if(st.size() == 0) return -1;
        long q = st.peek();
        if(q >= min) return (int)(q);
        if(q < min) return (int) min;
        return 0;
    }
```

(10) gfg { Celebrity Problem }

Potential ← 1 (1), ~~2~~, ~~1~~, ~~0~~

v1 = 2 ; if(M[2][1] == 0)
v2 = 1 ; M T by st.push(v2);
v1 = 1
v2 = 0.

else if(M[1][2] == 0) st.push(v2);

n = 3

M[][] ⇒

	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

(1) → is Celebrity.

~~⇒ Celebrity is the Person~~

~~⇒ Celebrity is~~

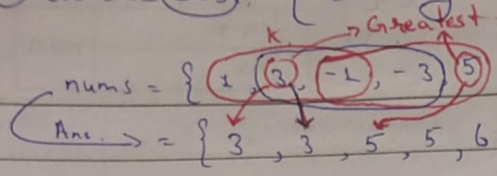
Who is ~~is~~ Celebrity ?

→ A person who don't know any one & every one knows him.

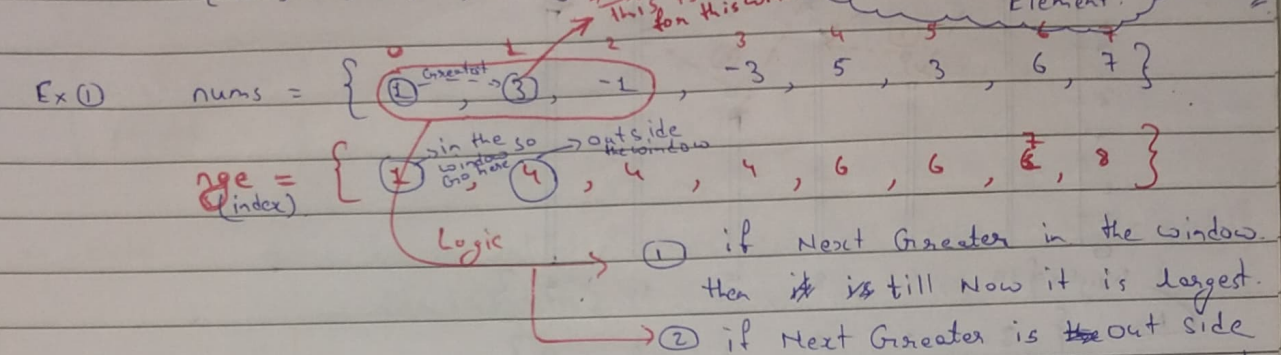
⇒ Code :-

```
public static void main(String[] args){
    int[][] M = {{0,1,0},{0,0,0},{0,1,0}};
    int n = M.length;

    Stack<Integer> st = new Stack<>();
    for(int i=0 ; i<n ; i++){
        st.push(i);
    }

    while(st.size() > 1){
        int v1 = st.pop();
        int v2 = st.pop();
        if(M[v1][v2] == 0){ // Shayad v1 celeb ho, v2 to celeb nahi hai.
            st.push(v1);
        }
        else if(M[v2][v1] == 0) st.push(v2);
    }
    if(st.size() == 0) Sout("-1");
    else{
        int potential = st.pop();
        for(int j=0 ; j<n ; j++){
            if(M[potential][j] == 1) Sout("-1");
        }
        for(int i=0 ; i<n ; i++){
            if(i == potential) continue;
            if(M[i][potential] == 0) Sout("-1");
        }
        Sout(potential);
    }
}
```

(11) LeetCode (239). { Sliding Window Maximum }



Hint:- find Next Greater Element.



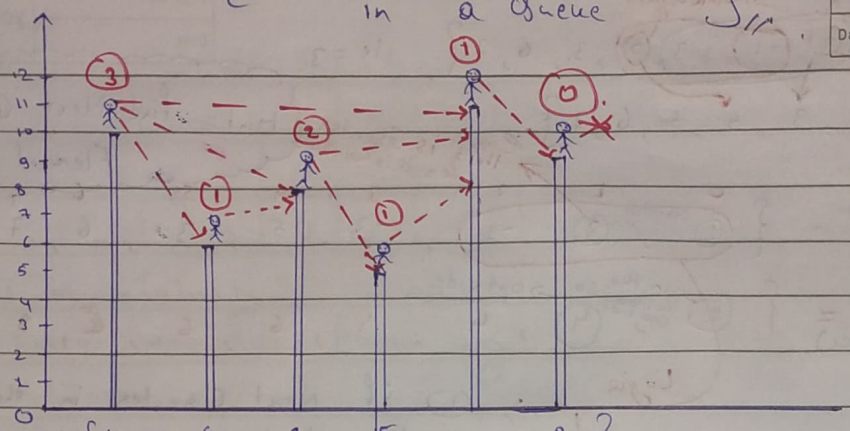
⇒ Code :-

```
public int[] maxSlidingWindow(int[] nums, int k){
    int n = nums.length;
    int[] ans = new int[n-k+1];
    int z = 0;
    Stack<Integer> st = new Stack<>();
    int[] nge = new int[n];
    st.push(n-1);
    nge[n-1] = n;
    for(int i = n-2 ; i >= 0 ; i--){
        while(st.size() > 0 && nums[i] > nums[st.peek()])
            st.pop();
        if(st.size() == 0) nge[i] = n;
        else nge[i] = st.peek();
        st.push(i);
    }
    int j = 0;
    for(int i = 0 ; i < n-k+1 ; i++){
        if(j >= i+k) j = i;
        int max = nums[j];
        while(j < i+k){
            max = nums[j];
            j = nge[j];
        }
        ans[z++] = max;
    }
    return ans;
}
```

Nge Code

Logic Part.

(12) LeetCode (1944) { Number of Visible People in a Queue } //.



=> Code :-

```
public int[] canSeePersonCount(int[] heights) {
    int n = heights.length;
    int[] res = new int[n];
    Stack<Integer> st = new Stack<>();
    for (int i = 0; i < n; i++) {
        while (!st.isEmpty() && heights[st.peek()] <= heights[i]) {
            res[st.peek()]++;
            st.pop();
        }
        if (!st.isEmpty()) {
            res[st.peek()]++;
        }
        st.push(i);
    }
    return res;
}
```

h[st.peek()] <= h[i]	
✗ 10 <= 6	(res++)
✓ 6 <= 8	(res++ pop.)
✗ 10 <= 8	
✗ 8 <= 5	
✓ 5 <= 11	
✓ 8 <= 11	
✓ 10 <= 11	
✗ 11 <= 9	

=> Dry Run :-

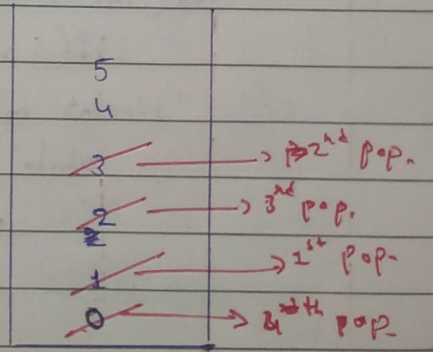
heights = { 10 , 6 , 8 , 5 , 11 , 9 }
(indices 0 1 2 3 4 5)

res = { ~~0~~ ③, ~~0~~ ①, ~~0~~ ②, ~~0~~ ①, ~~0~~ ①, 0 }

i = ~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, 6

i == n → Break.

res = { 3 , 1 , 2 , 1 , 1 , 0 }.



(13) LeetCode (946) {Validate Stack Sequence}

Given :-

Ex ① { pushed = {1, 2, 3, 4, 5}
popped = {4, 5, 3, 2, 1} } → True.

Ex ② { pushed = {1, 2, 3, 4, 5}
popped = {4, 3, 5, 1, 2} } → False.

=> Code :-

```
public boolean validateStackSequence(int[] pushed, int[] popped){
    Stack<Integer> st = new Stack<>();

    int j = 0;

    for(int val : pushed){
        st.push(val);

        while(!st.isEmpty() && st.peek() == popped[j]){
            st.pop();
            j++;
        }
    }
    return st.isEmpty();
}
```

st.peek() == popped[j]

- ✗ 1 == 4
- ✗ 2 == 4
- ✗ 3 == 4
- ✓ 4 == 4
- ✗ 3 == 5
- ✓ 5 == 5
- ✓ 3 == 3
- ✓ 2 == 2
- ✓ 1 == 1

=> Dry Run :- ①

pushed = {1, 2, 3, 4, 5}
popped = {4, 5, 3, 2, 1}

j = 0 1 2 3 4 5.

5 → 2nd pop (j++)
4 → 1st pop (j++)
3 → 2nd pop (j++)
2 → 4th pop (j++)
1 → 5th pop (j++).

st

=> st is empty → return True → Valid Stack Sequence

② pushed = {1, 2, 3, 4, 5}
popped = {4, 3, 5, 1, 2}

j = 0 1 2 3

st.peek() == popped[j]

- ✗ 1 == 4
- ✗ 2 == 4
- ✗ 3 == 4
- ✓ 4 == 4
- ✓ 3 == 3
- ✗ 2 == 5
- ✓ 5 == 5
- ✗ 2 == 1

5 → 3rd pop.
3 → 1st pop.
2 → 2nd pop.

st

=> st is Not empty → Return. (False)

Not Valid Stack Sequence.

(14) LeetCode (394) { Decode String }.

Ex (1) s = "3[a]2[bc]"
Ans → "aaabcbc".

Ex (2) s = "3[a2[c]]"
Ans → "accaccacc".

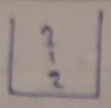
Ex (3) s = "2[abc]3[cd]ef"
Ans → "abcabccdcdcdef".

⇒ Code :-

```
public String decodeString(String s){
    Stack<Character> st = new Stack<>();
    for(char ch : s.toCharArray()){
        if(ch != ']')  st.push(ch);
        else{
            // Get the substring.
            StringBuilder sb = new StringBuilder();
            while(st.peek() != '['){
                sb.append(st.pop());
            }
            // Remove '['
            st.pop();
            // Now Get The Number.
            int k = 0;
            int base = 1;
            while(!st.isEmpty() && Character.isDigit(st.peek())){
                k = (st.pop() - '0') * base + k;
                base = base * 10;
            }
            while(k-- > 0){
                for(int i = sb.length()-1; i >= 0; i--){
                    st.push(sb.charAt(i));
                }
            }
        }
    }
    char[] result = new char[st.size()];
    for(int i = st.size()-1; i >= 0; i--){
        result[i] = st.pop();
    }
    return new String(result);
}
```

(213)

Dry Run :-



k = 0
base = 1.

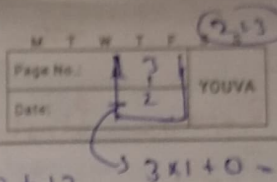
k = (st.pop() - '0') * base + k.
= (3 - '0') * 1 + 0
k = 3

base = 10.

k = (1 - '0') * 10 + 3
= 13

base = 100.

k = (2 - '0') * 100 + 13
= 213.



Ex ① S = "3[a]2[bc]"

⇒ ch ==] → Loop break.

// Get substring

Sb = "a"

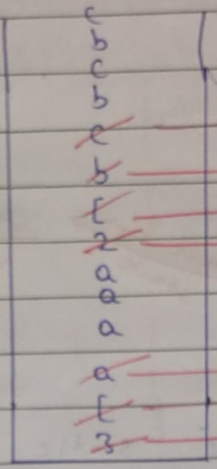
⇒ st.peek() == '[' → Loop break.

// Remove '['.

// Get Number

k = 3 - '0' * 1 + 0

k = 3.



result = {a, a, a, b, c, b, c}

return new String(result);.

// Get Sub. string

Sb = "cb".

k = 2 - '0' * 1 + 0

k = 2.

---

(15). ~~Leet Code~~ (234). {Palindrome Linked List}. Need to Check LL is Palindrome (True) or Not (False)

Ex ① (1) → (2) → (2) → (1).

Ans → True.

Ex ② (1) → (2).

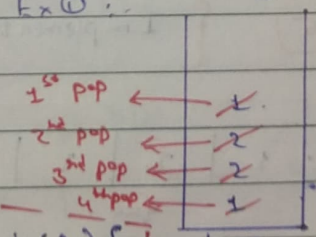
Ans → False.

⇒ Code :- public boolean isPalindrome(ListNode head){

```
Stack<ListNode> reverse = new Stack<ListNode>();
ListNode dummy = head;
while(dummy != null){
    reverse.push(dummy);
    dummy = dummy.next;
}
//Match List with reverse stack.
while(head != null && !reverse.isEmpty()){
    if(reverse.pop().val != head.val)
        return false;
    else{
        head = head.next;
    }
}
return true;
}
```

⇒ Dry Run :-

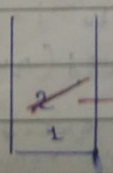
Ex ① :



pop().val != head.val
K 1 != 1
K 2 != 2
K 2 != 2
K 1 != 1

Reverse. st.isEmpty.
Return True.

Ex ①.



2 != 1 → Return False.