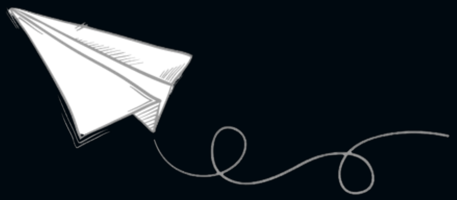
TCS NQT

Programming Problems with implementations



Array & Strings-3

Lecture 06

By- Aditya sir

About Aditya Jain sir

1. Appeared for GATE during BTech and secured AIR 60 in GATE in very first attempt - City topper
2. Represented college as the first Google DSC Ambassador.
3. The only student from the batch to secure an internship at Amazon. (9+ CGPA)
4. Had offer from IIT Bombay and IISc Bangalore to join the Masters program
5. Joined IIT Bombay for my 2 year Masters program, specialization in Data Science
6. Published multiple research papers in well known conferences along with the team
7. Received the prestigious excellence in Research award from IIT Bombay for my Masters thesis
8. Completed my Masters with an overall GPA of 9.36/10
9. Joined Dream11 as a Data Scientist
10. Have mentored working professions in field of Data Science and Analytics 10K+
11. Have been mentoring GATE aspirants to secure a great rank in limited time
12. Have got around 27.5K followers on Linkedin where I share my insights and guide students and professionals.

Telegram

@ADITYASIR_PW

Telegram Link for Aditya Jain sir:

https://t.me/AdityaSir_PW

[NAT]

#Q. Maximum Subarray Sum – Kadane's Algorithm

Problem Statement: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

Examples

Example 1:

Input: arr[] = {2, 3, -8, 7, -1, 2, 3}

Output: 11

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

Example 2:

Input: arr[] = {-2, -4}

Output: -2

Explanation: The subarray {-2} has the largest sum -2.

Example 3:

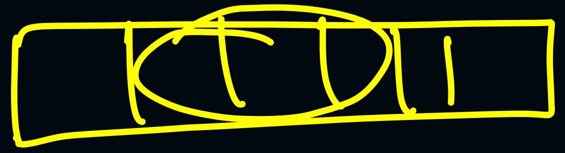
Input: arr[] = {5, 4, 1, 7, 8}

Output: 25

Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

V. Imp

Interview



eg1 :- $A = [2, 3, \boxed{-8}, \underbrace{7, -1, 2, 3}]$

~~5~~ -3 7 $7+5-1 = \boxed{11}$

Subarray
↓
contiguous

Ans :- 11 $\{7, -1, 2, 3\}$

$\underline{\{2, 3, 7, 2, 3\}}$

eg2 :- $\{-2, -4\}$

$$\Rightarrow \left.\begin{aligned} \{-2\} &\longrightarrow \underline{-2} \\ \{-4\} &\longrightarrow \underline{-4} \\ \{-2, -4\} &\longrightarrow -4-2 = \underline{\underline{-6}} \end{aligned}\right\} \text{max}$$

<u>eg3</u> :- $A = [5, 4, 1, 7, 8]$

$\{5\}$ $\{5, 4\}$

$\{4\}$ $\{5, 4, 1\}$

$\{1\}$ $\{5, 4, 1, 7\}$

$\vdots$

Ans: $\underbrace{5+4+1}+7+8$

$= 10 + 15 = \boxed{25}$

Appl :- Brute Force → All possible Subarrays :-

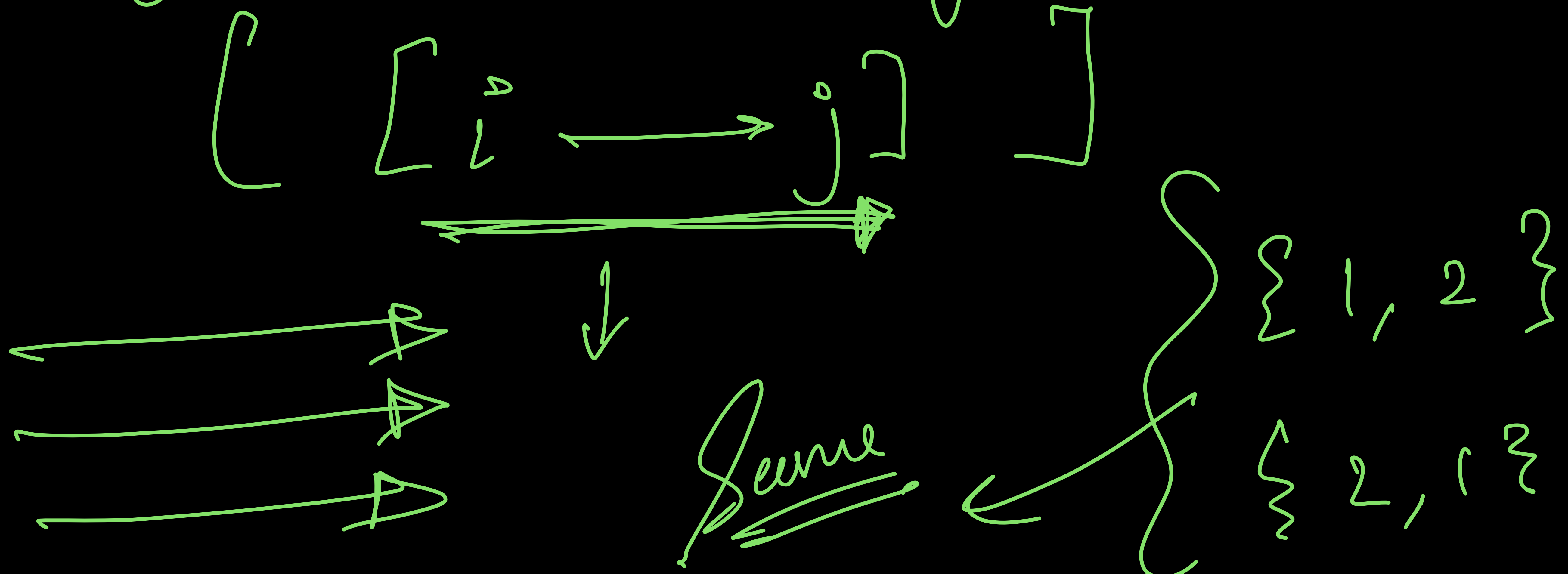
logic → find all possible Subarrays
and calc their Sums.

and store it
and keep max of it

[0, 1, 2, 3, 4 .. n]

i → start of subarray.

j → end point of subarray.



```
ans = INT_MIN;
for ( i = 0 ⟶ n-1)  ⟶ O(n)
{
    sum = 0                      i ⟶ j
    for ( j = i ⟶ n-1)  ⟶ O(n)
    {
        sum += a(j)
    }
                                   O(n^2)
    ans = max (ans, sum)
}
```

Appr 2 :- Kadane's Algo .

include

eg :- A = [$\boxed{2}$, ③ , −8 , 7 , −1 , 2 , 3]

curr_sum

✓ ✗

```
ans = a(0)

curr_sum = a(0)

for i = 1 ⟶ n
{
    curr_sum = max( curr_sum + ai ,
                    ai );

    ans = max ( ans , curr_sum );
}
```

$A = [2, 3, -8, 7, -1, 2, 3]$

fresh start

$2+3$

ans $= 5 - 8 = -3$ ✗

cur_sum $= \cancel{-3}\ 7$

cur_sum

$= \max(-3+7, 7)$

ans $= \cancel{5}\ 7$

$(4, 7)$

$(7, 6) = (7, 8) = (8, 11)$

$(-1, 6) = (6+2, 2) = (8+3, 3)$

$= 11$

$\boxed{11}$

[NAT]

#Q. Find Smallest Missing Positive Number

Problem Statement: Given an unsorted array arr[] containing both positive and negative elements, the task is to find the smallest positive number missing from the array.

Note: You can modify the original array.

Examples

Example 1:

Input: arr[] = {2, -3, 4, 1, 1, 7}

Output: 3

Explanation: The positive numbers in sorted order are {1, 2, 4, 7}. The smallest missing positive number is 3.

Example 2:

Input: arr[] = {5, 3, 2, 5, 1}

Output: 4

Explanation: The positive numbers in sorted order are {1, 2, 3, 5, 5}. The smallest missing positive number is 4.

Example 3:

Input: arr[] = {-8, 0, -1, -4, -3}

Output: 1

Explanation: There are no positive numbers in the array, so the smallest missing positive number is 1.

[MCQ]

#Q. Longest Common Subsequence (LCS)

Problem Statement: Given two strings, s1 and s2, the task is to find the length of the Longest Common Subsequence. If there is no common subsequence, return 0. A subsequence is a string generated from the original string by deleting 0 or more characters, without changing the relative order of the remaining characters.

For example, sub sequences of “ABC” are “”, “A”, “B”, “C”, “AB”, “AC”, “BC” and “ABC”. In general, a string of length n has 2n sub sequences.

Examples

Example 1:

Input: s1 = "ABC", s2 = "ACD"

Output: 2

Explanation: The longest subsequence common to both strings is "AC", which has a length of 2.

Example 2:

- Input: s1 = "AGGTAB", s2 = "GXTXAYB"
- Output: 4

Explanation: The longest common subsequence is "GTAB", which has a length of 4.

Example 3:

- Input: s1 = "ABC", s2 = "CBA"
- Output: 1

Explanation: The longest common sub sequences of length 1 are "A", "B", and "C". Hence, the result is 1.

[NAT]

#Q. Remove All Occurrences of an Element in an Array

Problem Statement: Given an integer array arr[] and an integer ele, the task is to remove all occurrences of ele from arr[] in-place and return the count of elements that are not equal to ele.

Additionally, if there are k elements not equal to ele, then arr[] should be modified such that the first k elements contain those not equal to ele, while the remaining elements can be anything.

Note: The order of the first k elements may change.

Examples

Example 1:

Input: arr[] = [3, 2, 2, 3], ele = 3

Output: 2

Explanation:

- There are 2 elements (2, 2) not equal to 3.
- Modify arr[] such that the first 2 elements contain 2.

Example 2:

Input: arr[] = [0, 1, 3, 0, 2, 2, 4, 2], ele = 2

Output: 5

Explanation:

- There are 5 elements (0, 1, 3, 0, 4) not equal to 2.
- Modify arr[] so that the first 5 elements contain these numbers.

Modified arr[]: [0, 1, 3, 0, 4, _, _, _]

[NAT]

#Q. 0/1 Knapsack Problem

Problem Statement: Given n items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

Input: W = 4, profit[] = [1, 2, 3], weight[] = [4, 5, 1] Output: 3 Explanation: There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible profit is 3. Note that we cannot put both the items with weight 4 and 1 together as the capacity of the bag is 4.

Input: W = 3, profit[] = [1, 2, 3], weight[] = [4, 5, 6] Output: 0

[NAT]

#Q. Move all zeros to end of array

Problem Statement : Given an array of integers arr[], the task is to move all the zeros to the end of the array while maintaining the relative order of all non-zero elements.

Examples:

Input: arr[] = [1, 2, 0, 4, 3, 0, 5, 0] Output: arr[] = [1, 2, 4, 3, 5, 0, 0, 0]
Explanation: There are three 0s that are moved to the end.

- Input: arr[] = [10, 20, 30] Output: arr[] = [10, 20, 30] Explanation: No change in array as there are no 0s.
- Input: arr[] = [0, 0] Output: arr[] = [0, 0] Explanation: No change in array as there are all 0s.

[NAT]

#Q. Second Largest Element in an Array

Problem Statement: Given an array of positive integers arr[] of size n, the task is to find the second largest distinct element in the array.

Note: If the second largest element does not exist, return -1.

Examples:

- Input: arr[] = [12, 35, 1, 10, 34, 1] Output: 34 Explanation: The largest element of the array is 35 and the second largest element is 34.
- Input: arr[] = [10, 5, 10] Output: 5 Explanation: The largest element of the array is 10 and the second largest element is 5.
- Input: arr[] = [10, 10, 10] Output: -1 Explanation: The largest element of the array is 10, but there is no second largest element.

[NAT]

#Q. Subset Sum Problem

Problem Statement: Given an array arr[] of non-negative integers and a value sum, the task is to check if there is a subset of the given array whose sum is equal to the given sum.

Examples:

- Input: arr[] = {3, 34, 4, 12, 5, 2}, sum = 9 Output: True Explanation: There is a subset (4, 5) with sum 9.
- Input: arr[] = {3, 34, 4, 12, 5, 2}, sum = 30 Output:False Explanation: There is no subset that adds up to 30.

[NAT]

#Q. Rearrange Array Elements by Sign

Problem Statement: Given an array arr[] of size n, the task is to rearrange it in an alternate positive and negative manner without changing the relative order of positive and negative numbers. In case of extra positive/negative numbers, they appear at the end of the array.

Note: The rearranged array should start with a positive number, and 0 (zero) should be considered as a positive number.

Examples:

- Input: arr[] = [1, 2, 3, -4, -1, 4] Output: arr[] = [1, -4, 2, -1, 3, 4]
- Input: arr[] = [-5, -2, 5, 2, 4, 7, 1, 8, 0, -8] Output: arr[] = [5, -5, 2, -2, 4, -8, 7, 1, 8, 0]

"Aditya Jain Sir PW"

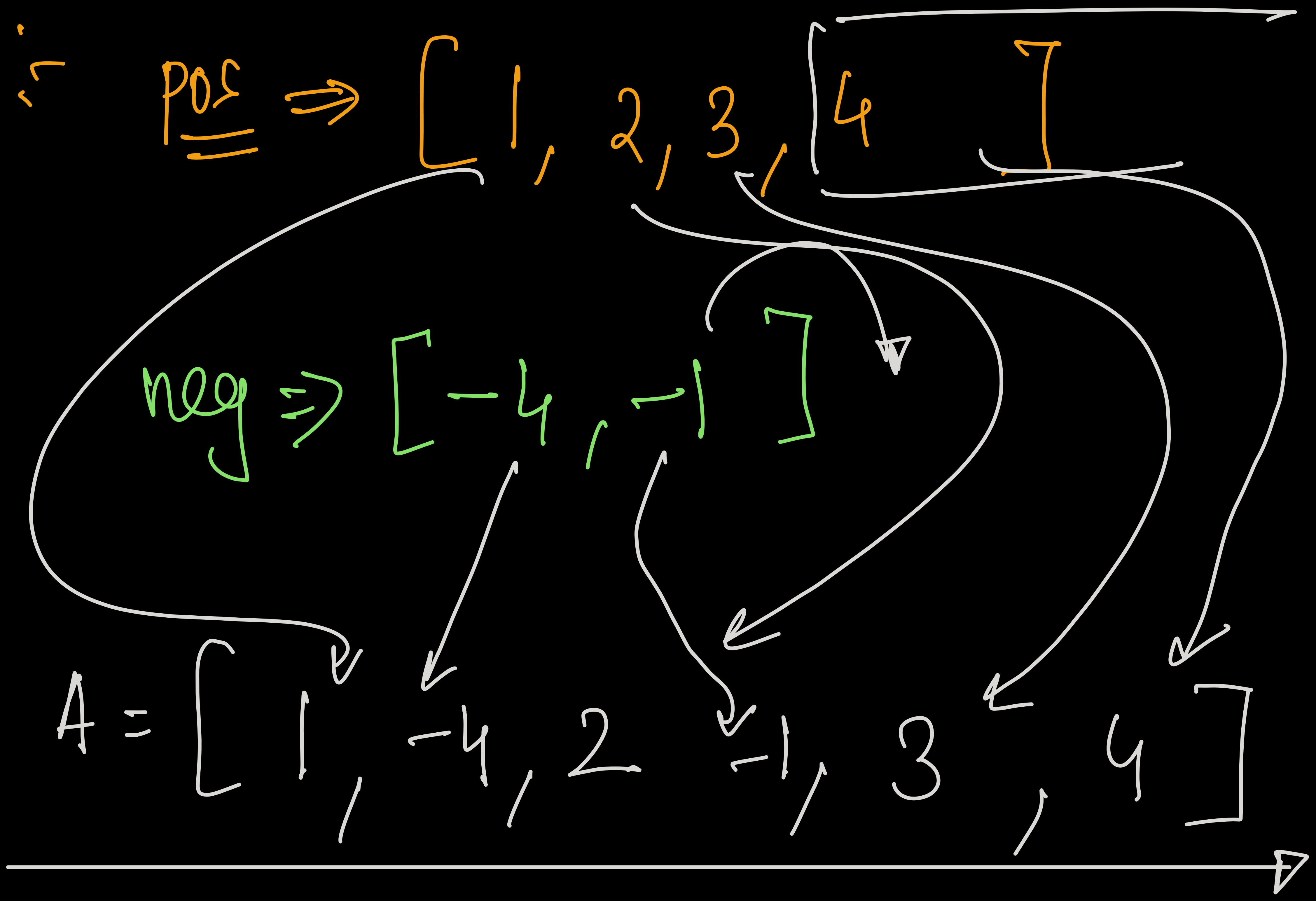
eg1 :- [1, 2, 3, -4, -1, 4]

$$\left[\begin{array}{cccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ + & + & + & - & - & + \end{array}\right]$$

$$\boxed{\text{alternate } \underline{+\text{ve}} \text{ \& } \underline{-\text{ve}}}$$

$$\underrightarrow{[\,1 \quad -4 \quad 2 \quad -1 \quad 3 \quad 4\,]} \quad ✓$$

Appr :-



pos ⇒ [1, 2, 3, 4]

neg ⇒ [−4 , −1]

A = [1, −4, 2, −1, 3, 4]

Logic code :-

0 → true

```
vector<int> pos, neg;

for (i=0 → n-1)
{
    if (a[i] < 0)
    {
        neg.pb(a[i]);
    }
    else {
        pos.pb(a[i]);
    }
}
```

0 1 2 3 →

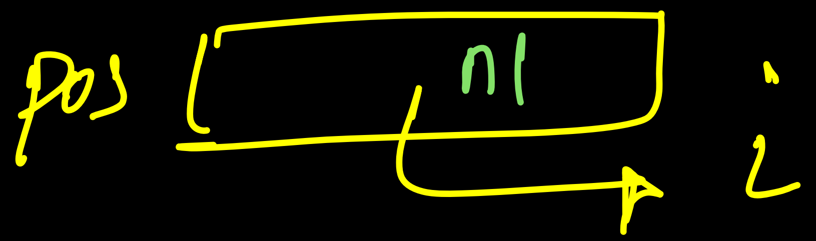
pos → [ ]

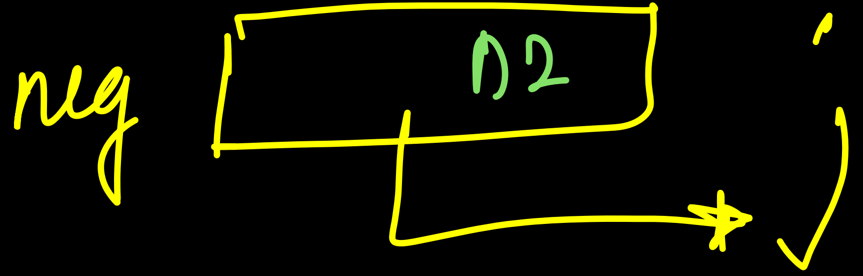
neg → [ ]

```
int i=0, j=0, ind=0

while (i<n1 and j<n2)
{
    if ( ind%2==0)
    {
        a[ind] = pos[i];
        i++;
    }
    else
    {
        a[ind]= neg[j];
    } j++
    ind++
}
```

0
2
4
6





```
while (i < n1)
{
    a[ind] = pos[i]
    ind++
    i++
}
while (j<n2)
{
    a[ind] = neg[j]
    ind++
}   j++
```

$curr\_sum + \cancel{a_i} > \cancel{a_i}$

$curr\_sum > 0$

$[\quad\quad] \; a_i$

> 0 true

$\boxed{[\quad\quad] \; \text{-ve}} \; a_i$ ✓

fresh start.

[NAT]

#Q. Missing ranges of numbers

Problem Statement: You have an inclusive interval [lower, upper] and a sorted array of unique integers arr[], all of which lie within this interval. A number x is considered missing if x is in the range [lower, upper] but not present in arr. Your task is to return the smallest set of sorted ranges that includes all missing numbers, ensuring no element from arr is within any range, and every missing number is covered exactly once.

Imp

Examples

- Input: arr[] = [14, 15, 20, 30, 31, 45], lower = 10, upper = 50 Output: [[10, 13], [16, 19], [21, 29], [32, 44], [46, 50]] Explanation: These ranges represent all missing numbers between 10 and 50 not present in the array
- Input: arr[] = [-48, -10, -6, -4, 0, 4, 17], lower = -54, upper = 17 Output: [[-54, -49], [-47, -11], [-9, -7], [-5, -5], [-3, -1], [1, 3], [5,16]] Explanation: These ranges represent all missing numbers between -54 and 17 not present in the array.

Topic : Problem Statement

V. Imp Interview Que.

Problem Statement: Valid Parentheses in an Expression

Given a string s representing an expression containing various types of brackets: {}, (), and [], the task is to determine whether the brackets in the expression are balanced or not. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

Examples:

Example 1:

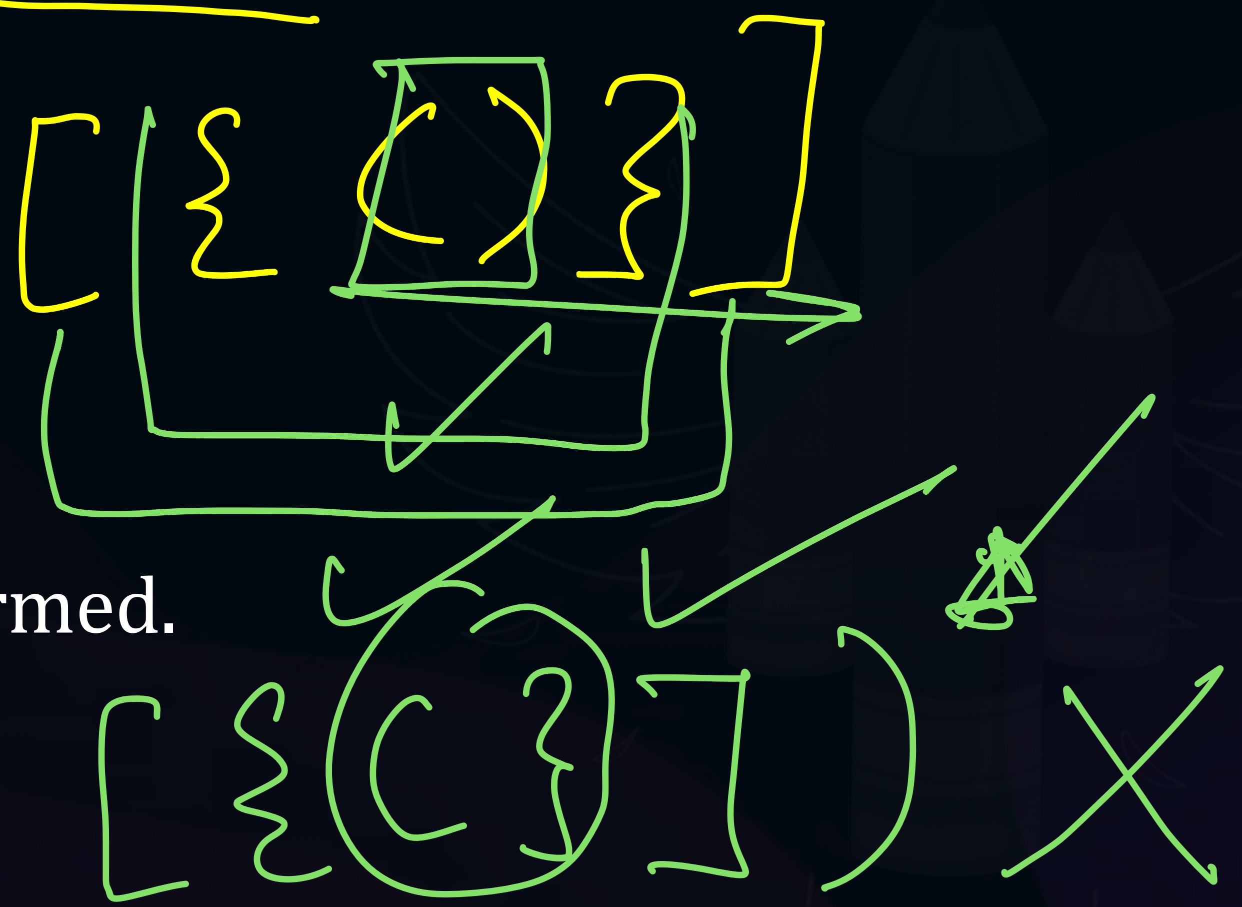
Input:

s = "[{()}]"

Output:

true

Explanation: All the brackets are well-formed.



Example 2:

Input:

s = "[()()]{}"

Output:

true

Explanation: All the brackets are well-formed.

Example 3:

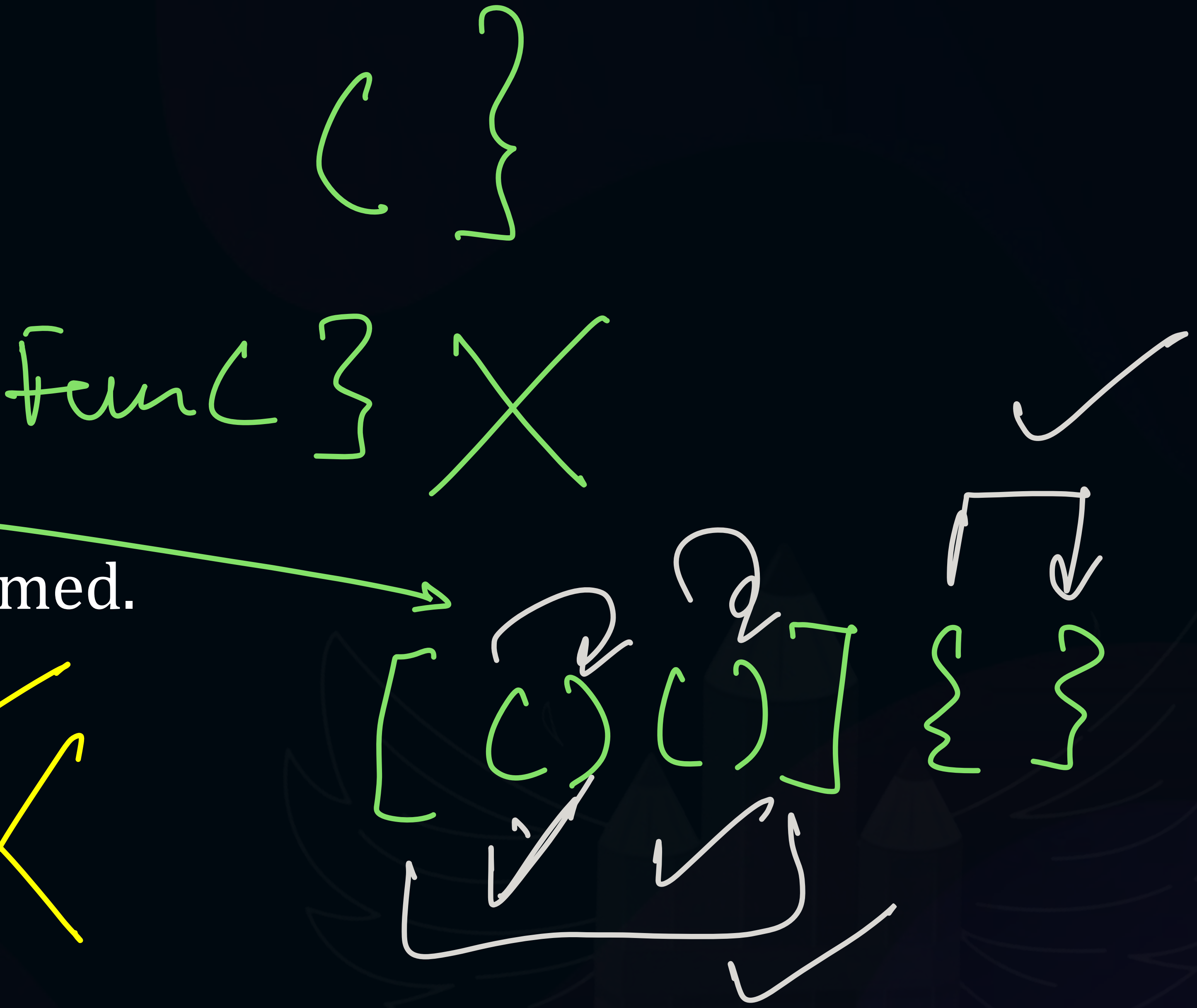
Input:

s = "([]"

Output:

false

Explanation: The expression is not balanced as there is a missing) at the end.



Example 4:

Input:

s = "([{]})"

Output:

false

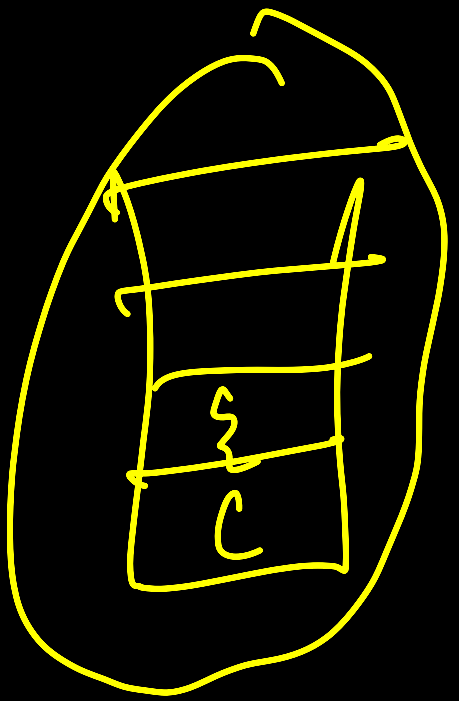
Explanation: The expression is not balanced because there is a closing] before the closing }.

$([\{] \})$

Stack → push( ), pop( )

Cond1 :- when any open bracket comes.

$\longrightarrow$ push( )



if (s(i) == '(' or '{' or '[')

{

st.push (s[i])

}

$\longrightarrow$ else if (!st.empty( ) and

1) $\longrightarrow$ (s(i) = ')' and st.top( ) = '('

OR

2) s(i) == '}' and st.top( ) = '{')

OR

3)

```
((s(i) : ']' and st.top : '[')

    →        st.pop();

else
{
         an = "No";
         break;

}
```

Topic : Problem Statement

Problem Statement: Reverse a String using Stack

Given a string str, reverse it using a stack.

Examples:

Example 1:

Input:

str = "abc"

Output:

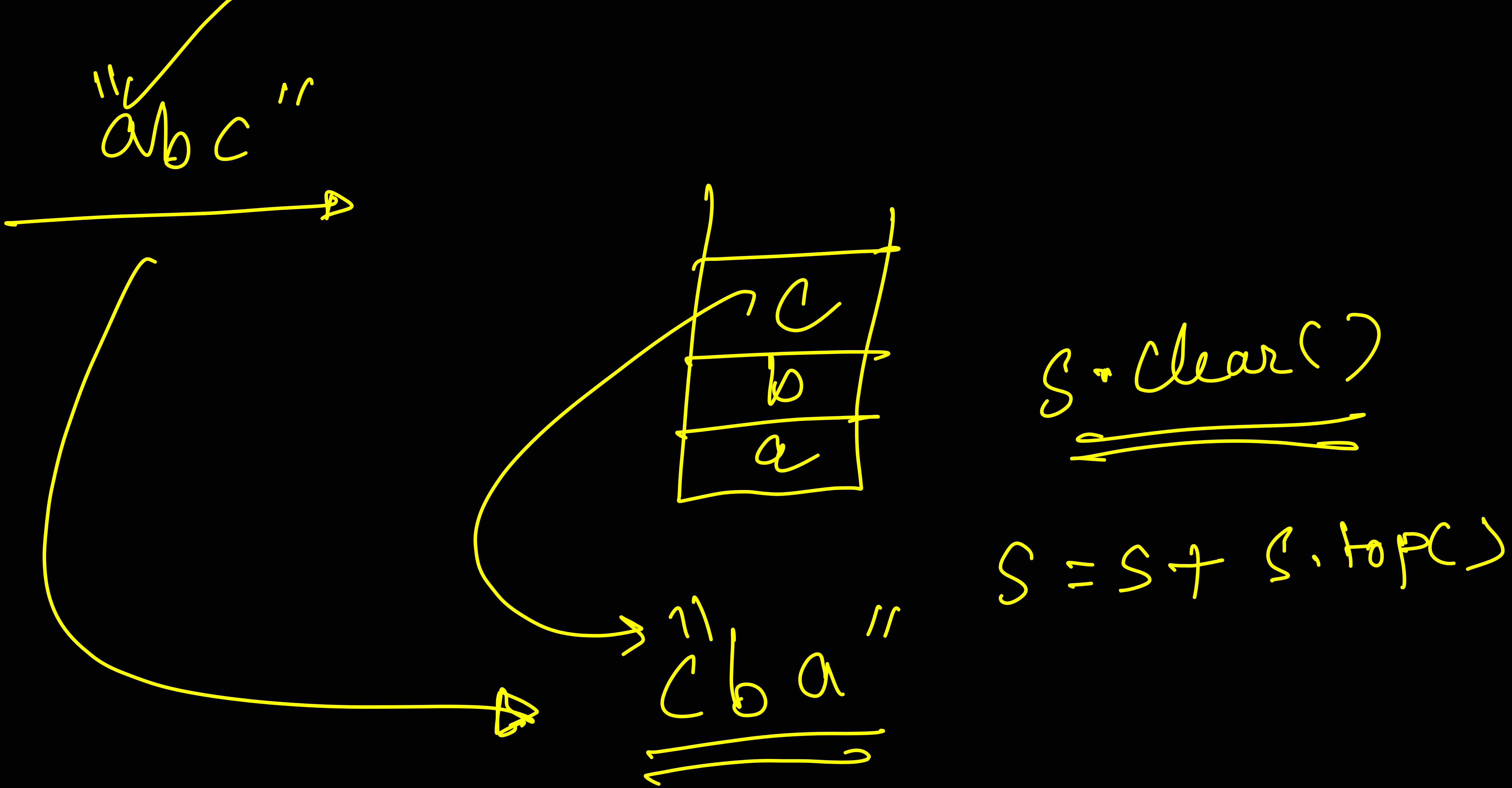
"cba"

3Q)

Soln :-

$\text{"abc"} \longrightarrow \text{"cba"}$

$\text{Stack} \longrightarrow \boxed{\text{LIFO}}$


"abc"
c
b
a
"cba"
s.clear()
s = s + s.top()

```
{ for( char c : s)
     {
            st.push( c);

     }

     s.clear()
     while( !st.empty())
        {
            s = s + st.top();
            st.pop();
        }
}
```

Topic : Problem Statement

Problem Statement: Check for Balanced Parentheses

Given a string s containing different types of brackets: {}, (), and [], the task is to determine whether the brackets are properly balanced. A well-balanced expression ensures that every opening bracket has a corresponding and correctly ordered closing bracket.

Examples:

Example 1:

Input:

s = "[{()}]"

Output:

true

Explanation: All brackets are correctly paired and properly nested.

Example 2:

Input:

s = "[()()]{}"

Output:

true

Explanation: The brackets are well-formed and correctly structured.

Example 3:

Input:

s = "([]"

Output:

false

Explanation: The expression is unbalanced due to a missing closing).

Example 4:

Input:

s = "([{]})"

Output:

false

Explanation: The expression is incorrect as] appears before } in an invalid sequence.

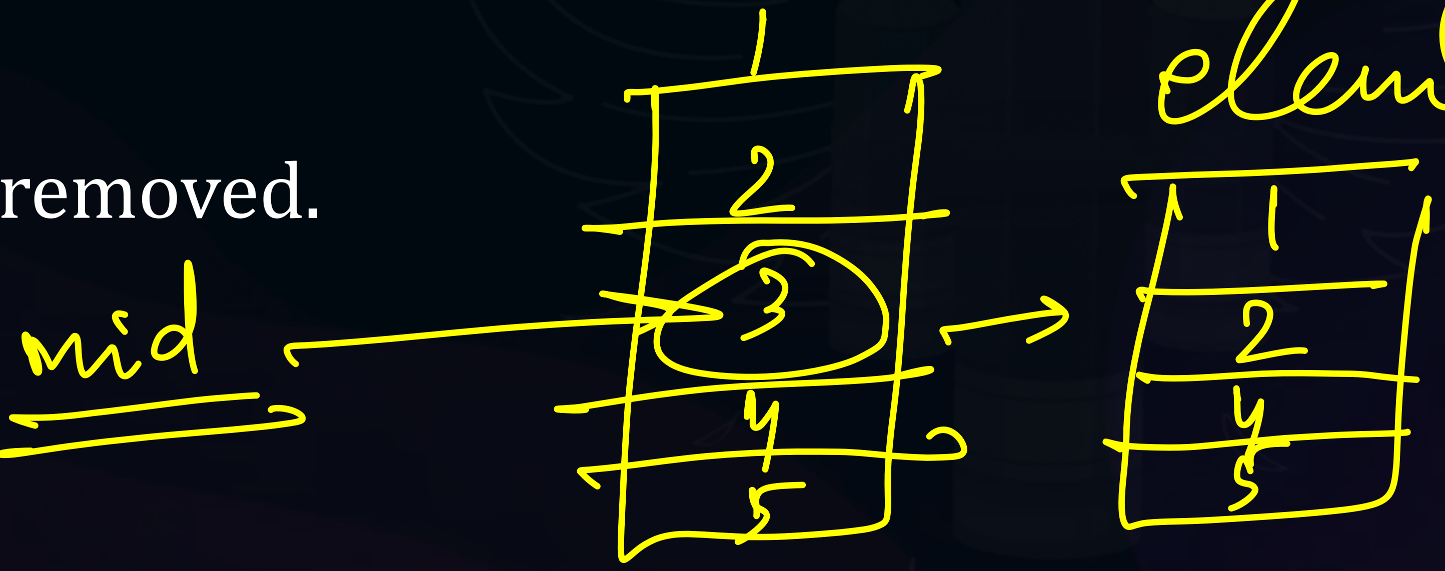
Topic : Problem Statement

Problem Statement: Delete Middle Element of a Stack
Given a stack with standard operations (push(), pop(), and empty()), the task is to remove the middle element from the stack without using any additional data structure.

stack

assume given stack has odd no. of elems.

Examples:
Example 1:
Input:
Stack (Top to Bottom): [1, 2, 3, 4, 5]
Output:
Stack (Top to Bottom): [1, 2, 4, 5]
Explanation: The middle element 3 is removed.

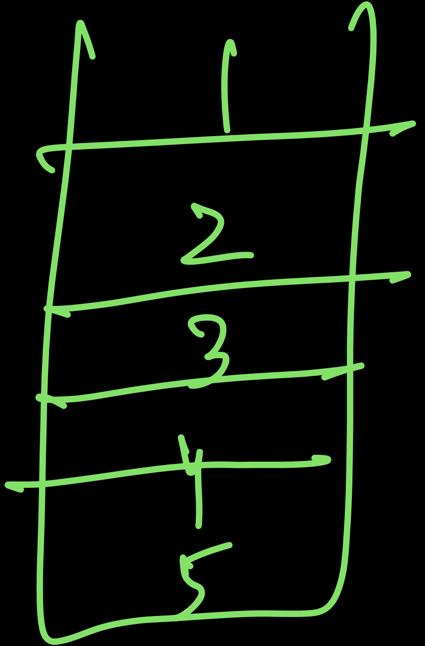


Stack < int > s1 $\longrightarrow$ given

Stack < int > s2

$n = s1.size()$

5

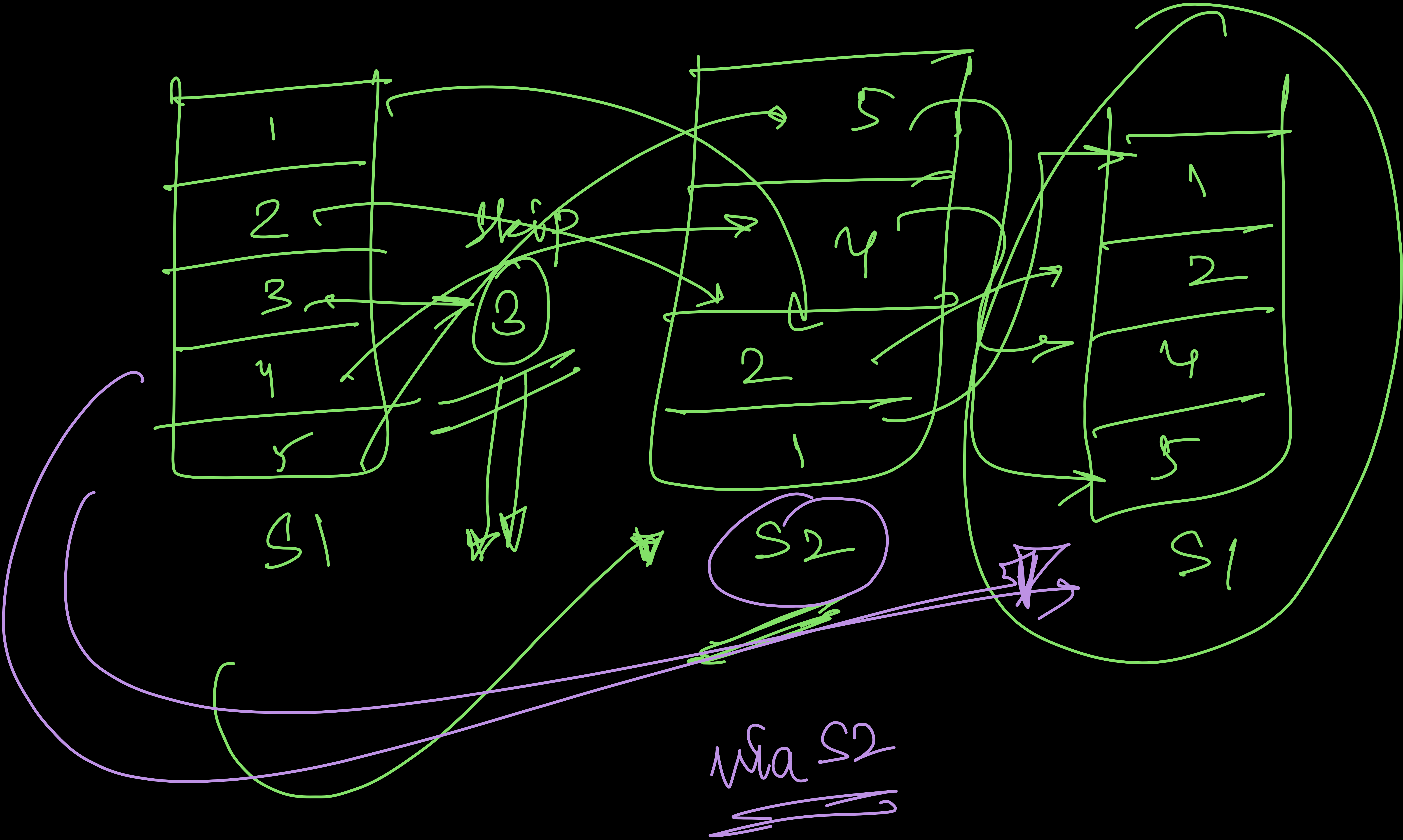


mid = $n/2$

$5/2 = \boxed{2}$

```
for (i = 0 ——> n-1)
{
    if (i != mid)
    {
        val = s1.top()
        s2.push(val)
    }
    s1.pop();
}
```

counting


1
2
3
4
5
S1
skip
3
5
4
2
1
S2
1
2
4
5
S1
via S2

THANK - YOU

" Aditya Jain Sir PW "