CS204(2025)
Stack & Queue
Instructions for Implementation

- Write the program in C or C++ using standard input/output.
- Follow the input/output format strictly.
- Ensure your code:
  - Handles **edge cases** correctly.
  - Meets the **time and space complexity constraints**.
- **Do not use STL containers like `vector`, `set`, or `map` in C++.** Use raw arrays unless explicitly allowed.
- You are encouraged to write clean, modular, and well-documented code.

Q1. Min stack

Problem Statement:

Alice wants to design a stack that supports push, pop, top and retrieving minimum element in constant time. Help her to do so!!

- **push(int val)** pushes the element **val** onto the stack.
- **pop()** removes the element on the top of the stack.
- **top()** gets the top element of the stack.
- **getMin()** retrieves the minimum element in the stack.

Every function must be implemented in O(1) time.

Input:

- The first line contains a number “n” which denotes the number of operations to perform on the stack.
- Next “n” lines contain either one or two integers denoting the command number and value (if any).
  Command number:
  1→push()
  2 →pop()
  3 →top()
  4 →getMin()

Output:

- Print the output for commands top(), getMin()
- If there is no top element or no minimum element, then print “null”.

Constraints

$0 < n <= 1000$

-100<= values <= 100

Sample Test Cases

1. **Sample Input:**
```
7
1 -2
1 0
1 -3
4
2
3
4
```
Sample Output:
```
-3
0
-2
```

2. **Sample Input:**
```
4
1 5
3
4
2
```
Sample Output:
```
5
5
```

Q2. Infix to Postfix

Problem Statement:

Given infix expression containing brackets ‘(‘ and operations [+, - , /, *] and single digit numbers, return postfix expression. Output must contain only operators and numbers
standard precedence: * / + - left-to-right associativity.

Input

String consisting of brackets(,)’, operators, and numbers

Output

Print the Postfix expression in a single line

Sample Test cases

1. **Sample Input**
   2+3*4
   Sample Output
   234*+
2. **Sample Input**
   (2+3)*4
   Sample Output
   23+4*

Q3. Postfix Evaluation

Problem Statement:

Given a Postfix expression containing operations [+, -, *, /] and single-digit numbers, return the result after evaluation of the postfix expression, and return the maximum size of the stack used during evaluation of the postfix expression

Input

String consisting of operators and numbers

Output

First line:

Print the evaluated value of the postfix expression. If the postfix expression is not valid, then print “Invalid”.

Second line:

Print the maximum stack size.

Sample Test Cases

1. **Sample Input:**
   234*+
   Sample Output:
   14
   3
2. **Sample Input**
   2+
   Sample Output:
   Invalid

Q4. Implement a queue using stacks.

Problem Statement:

You will be given operations of enqueue and dequeue. But you have to use a stack behind the scenes to stimulate the queue. Also, think about the complexities of enqueue and dequeue if stacks are used.
(Hint: You can use multiple stacks)
Input Format:

- First Line: Integer ‘n’ (number of operations given for the queue)
- Next ‘n’ lines may contain either of the following formats:
  - 1 m
  - 2

  Where for ‘1’ you have to enqueue the given ‘m’ integer
  For ‘2’, you have to dequeue and print the front of the queue. If empty print -1

Output Format:

- For every operation of type ‘2’, you must print dequeue as well as print the front of queue on a new line

Constraints;

- 1 <= n <= 100
- 0 <= m <= 10^9

Sample Input 1:
```
3
1 10
1 0
2
```
Sample Output 1:
```
10
```

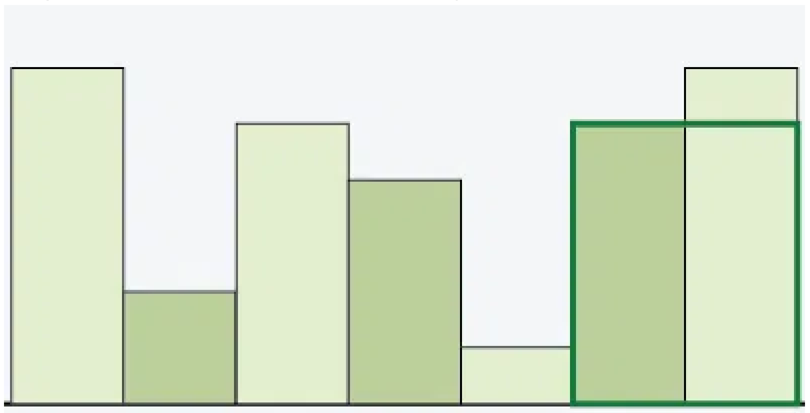
Sample Input 2:
```
6
1 6023
1 3141
2
1 2181
1 100
2
```

Sample Output 2:
```
6023
3141
```

Q5. Max Area

Problem Statement:

Picture a city street with a row of buildings, each of a different height. Assume the width of each building is 1. You need to find the biggest possible rectangular area for a billboard that can be attached to a continuous block of these buildings. The shortest building on that block would limit the billboard's height. What is the maximum area you could achieve?



E.g.) One of the feasible solutions for selecting a billboard is given above

Input Format:

- First Line: Integer ‘n’ (number of buildings next to each other)
- Next line contains ‘n’ integers, each separated by space, where each integer represents the height of a building

Output Format:

- Print a single integer followed by a new line

Constraints;

- 1 <= n <= 10^5
- 1 <= height of building <= 10^9

Sample Input 1:
```
8
4 1 5 3 3 2 4 1
```

Sample Output 1:
```
10
```

Sample Input 2:
```
6
3 5 1 7 5 9
```

Sample Output 2:
```
15
```

Q6. Alice wants to design a data structure that supports operations:

Stores given element

Get the minimum element in O(1)

Remove the minimum element in O(logN)

Help her!!!

Input Format:

- First Line: Integer n (number of operations)
- Next n lines contain either of the following formats:
  - 1 m
  - 2
  - 3

  Where for ‘1’ you have to store the given ‘m’ integer

  For ‘2’, you have to print the minimum element among the ones stored until now in O(1). If empty print -1

  For ‘3’, you have to remove the minimum element among the ones stored until now in O(logN)

Output Format:

- For every operation of type ‘2’, you have to print the minimum element

Constraints:

- 1 <= n <= 100
- 1 <= m <= 100000

Sample Input 1:

```
3
1 100
1 20
2
```

Sample Output 1:

```
20
```

Sample Input 2:

```
4
1 100
1 21
3
2
```

Sample Output 2:

```
100
```

Q7. The median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is **the first of the middle values**.

- For example, for arr = [2,3,4], the median is 3.
- For example, for arr = [2,3], the median is (2,3) = 2.

  You have to build a data structure that performs two operations:
  1) Stores ‘m’ numbers when asked
  2) Gives the median of the numbers stored until now in the data structure in O(1)

Input Format:
The first line contains ‘n’, which gives the number of operations.
In Next ‘n’ lines, you will be given 2 types of operations in the following format:

- 1 m
  x_1 x_2 x_3 … x_m
  Where ‘1’ represents operations of storing given ‘m’ numbers
  Next line contains ‘m’ integers denoted by x_1,x_2, …, x_m
- 2
  Where ‘2’ represents the operation to print the median of the stored numbers

Output Format:

- For every operation of type ‘2’, you have to print the median element. **The answer will always be an integer.**

Constraints:

- 1 <= n <= 100
- -10^9 <= number <= 10^9
- You can assume that there will be at least one operation of type ‘1’ before ‘2’

Sample Input 1:
```
4
1 4
10 -1 4 2
2
1 3
1 3 5
2
```
Sample Output 1:
```
2
3
```
Sample Input 2:
```
2
1 1
1
2
```
Sample Output 2:
```
1
```

Q8. A bracket is considered to be any one of the following characters: `(`, `)`, `{`, `}`, `[`, or `]`.

Two brackets are considered to be a matched pair if an opening bracket (i.e., `(`, `[`, or `{`) occurs to the left of a closing bracket (i.e., `)`, `]`, or `}`) of the same type. There are three types of matched pairs of brackets: `[]`, `{}`, and `()`.

A matching pair of brackets is not balanced if the set of brackets it encloses is not matched. For example, `{[(])}` is not balanced because the contents in between `{` and `}` are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, `(`, and the pair of parentheses encloses a single, unbalanced closing square bracket, `]`.

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

- It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return `YES`. Otherwise, return `NO`.

<u>Input Format:</u>
The first line contains 'n', which gives the string length.
Next Line contains n characters where each character can be any of `(`, `)`, `{`, `}`, `[`, or `]`
<u>Output Format:</u>
- If the bracket sequence is balanced, then print "Yes". Else print "No".

Constraints:
- 2 <= n <= 100

Sample Input 1:
4
{][}
Sample Output 1:
No

Sample Input 2:
2
{}
Sample Output 2:
Yes