**I. ALGORITHM: Propose a data structure that supports the stack operations PUSH, POP, and a third operation FIND_MIN which returns the smallest element in the stack (does not delete the element).**

For this data structure, we can use two stacks. One stack can store stack elements and the other stack can store the smallest values. We can call the first stack the Master stack and the second stack the Reserve stack. Using this, the PUSH and POP operations allows the top of Reserve stack to always contain the smallest element.

**Algorithms:**

PUSH():

1. Let x be the top element in the Test stack and let y be the top element in the Reserve stack. To start, we push the first element from Master stack into both Test and Reserve. This makes x and y initially equal.
2. Compare the values of x and y.
    a. If x is smaller than y, push x into the Reserve stack.
    b. If x is greater than y, push y into the Reserve stack.
3. Push the next element from Master into Test, but not Reserve. Make the same comparison for the elements from step 2.
4. Repeat the process until there are no longer any elements left in the Master stack.

POP():

1. After step 2 in the PUSH() algorithm:
    a. If x is smaller than y, Pop the top element of the Reserve stack.
        i. This removes the previous smallest element found in the stack and allows the new smallest element to become the top.
    b. If x is greater than y, Pop the top element of the Reserve stack and push the same element again into the top.
2. Pop the top element of the Test stack and return.

FIND_MIN:

Return the top element of the Reserve stack.

**Example:**

Suppose we have the elements 23, 28, 45, 31, and 12 inserted into the Master stack.

1. Place the first element, 28, into both the Test and Reserve stacks:

| Test | Reserve |
|---|---|
| 23 (top) | 23 (top) |

2. Next, we insert the next element from the Master stack, 28, to the top of the Test stack:

| Test | Reserve |
|---|---|
| 28 (top) | 23 (top) |
| 23 | 23 |

After we placed 28 to the Test stack, note how 23 is still the smallest element in the stack, and thus 23 appears again in the top of the Reserve stack.

3. Next, we insert the next element from the Master stack, 45, to the top of the Test stack:

| Test | Reserve |
|---|---|
| 45 (top) | 23 (top) |
| 28 | 23 |
| 23 | 23 |

After we placed 45 to the Test stack, 23 is still the smallest element in the stack, and thus it remains on the top of the Reserve stack.

4. Next, we insert the next element from the Master stack, 31, to the top of the Test stack:

| Test | Reserve |
|---|---|
| 31 (top) | 23 (top) |
| 45 | 23 |
| 28 | 23 |
| 23 | 23 |

After we placed 31 to the Test stack, 23 is still the smallest element in the stack, and thus it remains on the top of the Reserve stack.

5. Next, we insert the final element from the Master stack, 12, to the top of the Test stack:

| Test | Reserve |
|---|---|
| 12 (top) | 12 (top) |
| 31 | 23 |
| 45 | 23 |
| 28 | 23 |
| 23 | 23 |

After we placed 12 to the Test stack, 23 is no longer the smallest element in the stack, and thus 12 is Popped from the Test stack FIND_MIN is called into the Reserve stack. 12 is now placed on the top of the stack.

**II. PROGRAM: See attached .py file.**