**Algorithm:**

minElement keep track to minimum element stack is holding.

(1) PUSH(x): Insert x at the top of stack.

If stack is empty, insert x into the stack and make minElement equal to x.

If stack is not empty, compare x with minElement.

Cases:

1. If (x>=minElement) simply insert x

2. If (x<minElement) insert (2 \* x -minElement) into the stack and minElement = x.

**EXPLANATION:**

The above statement "2\*n-min Element" is used because of the following things:

* When POP action is performed to keep track of the minElement, we use Push (2\*x-minElement) in our answer.
* W use (2\*x-minElement) instead of push(x) so as to retrieve the previous element which is minimum using current minElement and its value stored instead.

**We have here the illustration:**

**Push(x):**

If x<minElement, push (2\*x-minElement) into stack and make minElement = x

**POP ( ):** If x< minElement, updateMinElement = 2\* minElement -x.

**Here is the Push Operation:**

|  |  |  |
| --- | --- | --- |
| Push (x) | Stack | minElement |
| 3 | 3 | 3 |
| 5 | 5 3 | 3 |
| 2 | 1 5 3 | 2 |
| 1 | 0 1 5 3 | 1 |

When we use push (2), 2 < minElement, minElement = 2, stack = (2 \* 2 - 3) = 1 as previously minElement = 3.

Insert (1),

1 < minElement (2).

Stack = (2 \* 1 - 2) = 0

Push (0) in stack, update minElement = 1

**Now, POP ( ).**

|  |  |  |
| --- | --- | --- |
| Stack | Element remove | minElement |
| 0 1 5 3 | - | 1 |
| 1 5 3 | 0 | 2 |
| 5 3 | 1 | 3 |
| 3 | 5 | 3 |

Number Removed (0), 0 < minElement,

number removed is 0, minElement = (2 \* 1 - 0) = 2

Number removed 1, 1 < minElement

New minElement = (2 \* 2 - 1) = 3

Number removed 5, 5 > minElement.

minElement is not removed or updated. minElement is still 3.

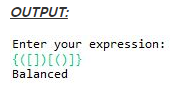
Thus, this helps to preserve the elements and its minimum.

**NOTE: The stack doesn't hold actual value of an element.**

**Program:**

**The programing language used here is Java(attached as a java file)**

**Outpot:**



**And :**



