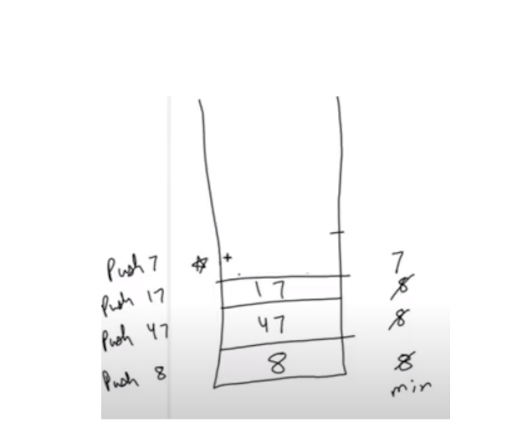
**1. Problem Discussion :**

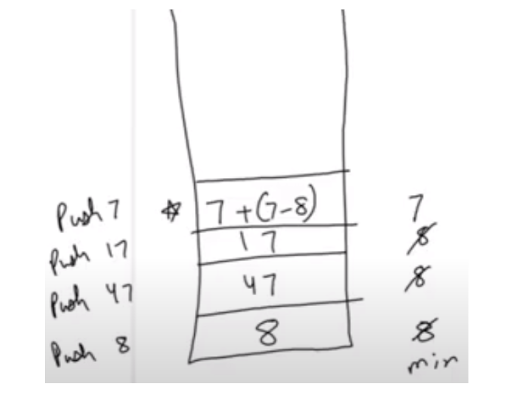
Here, we already are equipped with a Stack data type "data" and an integer "min". Since there is only one variable for calculating min, therefore the space will be constant unlike the previous question which used a "minData" stack for calculating min. The time taken for executing the program should also be constant. We are required to code a function min which should return the smallest element available in the stack and if not available, returns -1 after printing "Stack underflow".

**2. Approach :**

Let's say we are required to push 8, 47 and 17 in the stack initially. This can be easily done by you. Look at the figure 1 and the discussion below it.

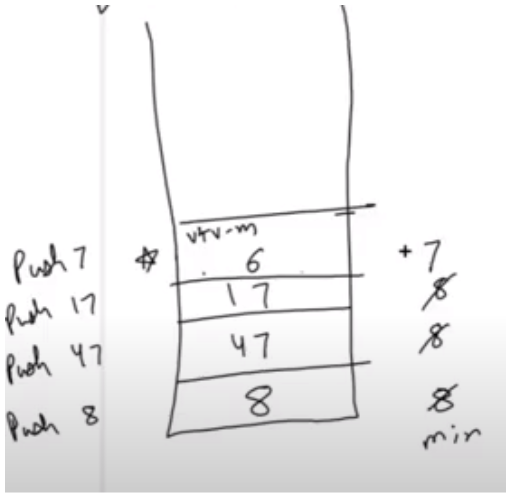
****

Push 8: We push the value 8 and store 8 in "min" . Then 47 and 17 are pushed normally because their value is not less than the value of "min". Next we have to Push 7. Here 7 is less than the value stored in "min" i.e. 7<8 . So instead of pushing the value 7 in the stack we push 7+(7-8) in the stack. This value denotes value + (value - min ). Also we update "min" with our original value 7.

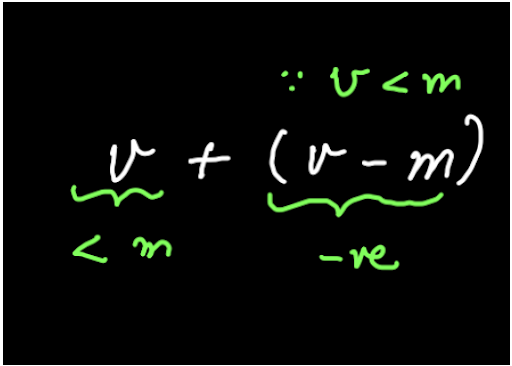
****

Reader, mind you this is only the "WHAT" section. We will soon discuss the "Why" , "How" and "Implications" of this problem. Just have a little patience.

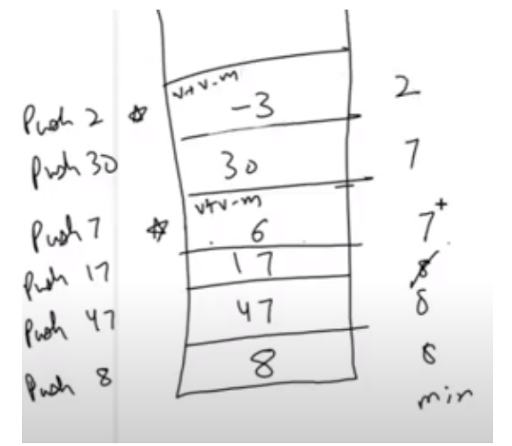
Implications :

****

The implication of storing the value v+v-m in the stack is that for that level, the value at stack is less than the value of min. (where, v=value and m="min") Why do you think it is smaller?

****

Since v< m therefore, (v-m) is negative. Now if v < m, then subtracting something to the value of v must also be less than the min. Hence a value less than 7 i.e. 6will be stored here. Concentrate on the stack as we push 2 into it when we continue the pushing process in Figure 5.

****

Here again for value 2 less than min, we push v+( v-m) in the stack and update the "min" as that value 2. At this level too, the value in stack i.e. -3 is less than the min.

To sum this explanation, we have 3 implications:

1• We store a fake value in the stack 2• Original value is stored in "min". 3• Detection case: This fake value is less than the minimum value for that level.

**3. Code :**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class MinStack {

Stack< Integer> data;

int min;

public MinStack() {

data = new Stack< >();

}

int size() {

return data.size();

}

void push(int val) {

if (size() == 0) {

data.push(val);

min = val;

} else if (val < min) {

data.push(val + val - min);

min = val;

} else {

data.push(val);

}

}

int pop() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

} else

{ if (data.peek() < min) {

int oval = min;

min = 2 \* min - data.pop();

return oval;

} else {

return data.pop();

}

}

}

int top() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

if (data.peek() >= min) {

return data.peek();

} else {

return min;

}

}

}

int min() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

}

else return min;

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

MinStack st = new MinStack();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("push")) {

int val = Integer.parseInt(str.split(" ")[1]);

st.push(val);

} else if (str.startsWith("pop")) {

int val = st.pop();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("top")) {

int val = st.top();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(st.size());

} else if (str.startsWith("min")) {

int val = st.min();

if (val != -1) {

System.out.println(val);

}

}

str = br.readLine();

}

}

}

**4. Code Discussion :**

To understand the implications we simultaneously explain the "Why" of each code snippet.

****

The above code is already available to us. Let's code the other functions using it:

Size :

****

The size( ) function simply returns the size of all the elements i.e. the size of the "data" stack.

Push :

****

If the "data" stack is empty then we push the value and store it in the "min" variable. Else if the value is less than min, then we push value+(value-min) in the stack and update min with the given value as already discussed in the "What" section. Else if the value is greater than "min" it is just pushed in the stack.

Min :

****

If the stack is empty then we return -1 else we simply return the value stored in the "min" variable.

Top :

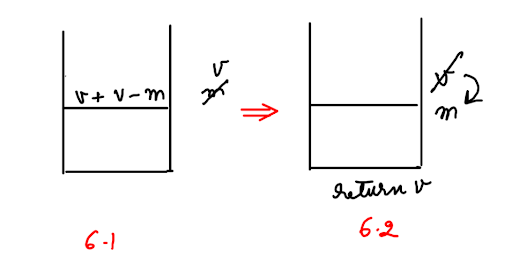
****

If the peek value is greater than or equal to the "min" value then it is directly printed as we made no changes in the original value for that condition. But if the peek value is less than the min value, it means this is the detection case and here the actual value was stored in "min" which is why "min" is returned.

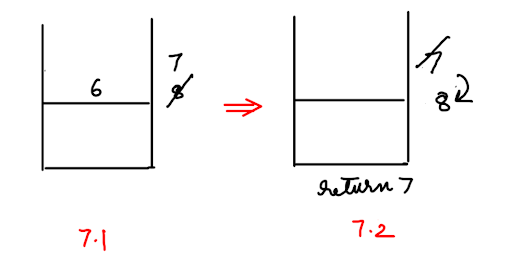
Pop :

****

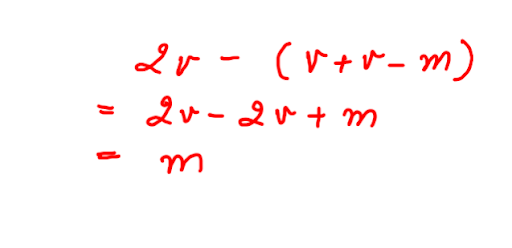
The pop function can be understood as encrypting the decrypted data. If the peek data is greater than or equal to "min" then we pop that value from the stack and return it. But if the peek data is less than "min" then it is our detection case. The value stored in the stack is fake. So now we want to encrypt this fake value back to its original value. Let's see how.

****

In figure 6.1, on decryption the stack stored v+v-m and the min value was changed to v. Now for encryption, in Figure 6.2 we want to return the original value v when stack is popped and want to change the "min" value that stored the given value to return back to the previous "min" value before it. This can be explained for the first detection case of our example (see figure 7). Here first v+v-m =6 was pushed into stack and min=8 was changed to min=7. On encryption, we return 7 when the stack is popped and replace min=7 with the previous value of "min" i.e. min=8.

****

Now look how this encryption from stack value to min takes place:

****

If we subtract the fake value from twice of the original value, we get our previous value of min. For our example, 2\*7 - (7+7 -8) gives us 8 which is the previous "min" value. Hence in the code, we return the latter min =7 when the stack is popped and then change the min value using the above formula back to min=8.

**5. Analysis :**

Time Complexity : O(1)

According to the question, we have written all the functions in constant time.

Space Complexity : O(1)

According to the question, no extra space is used, hence the complexity is constant. We highly suggest you refer to the solution video of this problem to understand the process better.