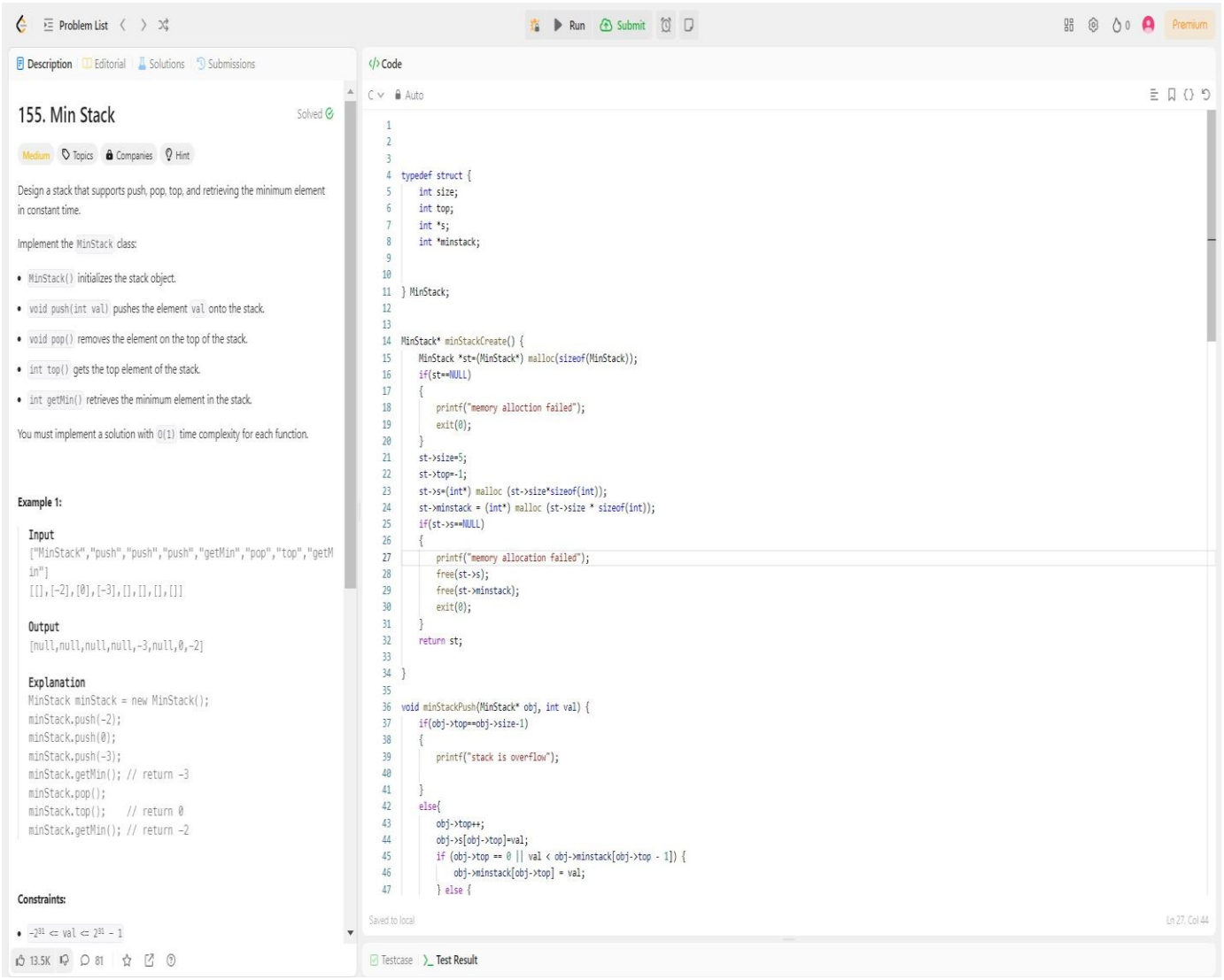
Leetcode minstack



Problem List

Run Submit

Premium

Description | Editorial | Solutions | Submissions

155. Min Stack

Solved

Medium | Topics | Companies | Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Explanation

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();    // return 0
minStack.getMin(); // return -2
```

Constraints:

- $-2^{31} <= val <= 2^{31} - 1$

13.5K | 81

Code

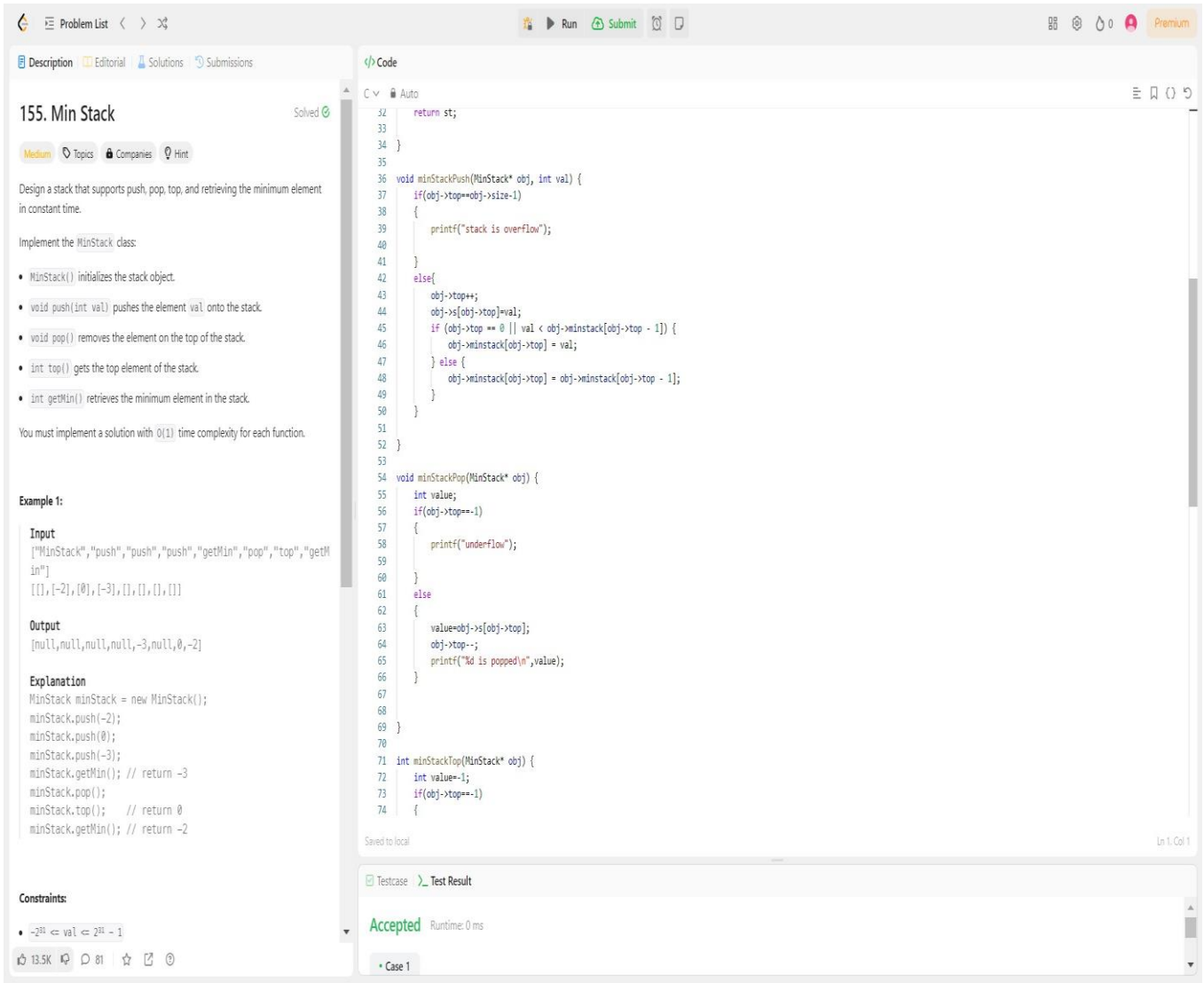
C | Auto

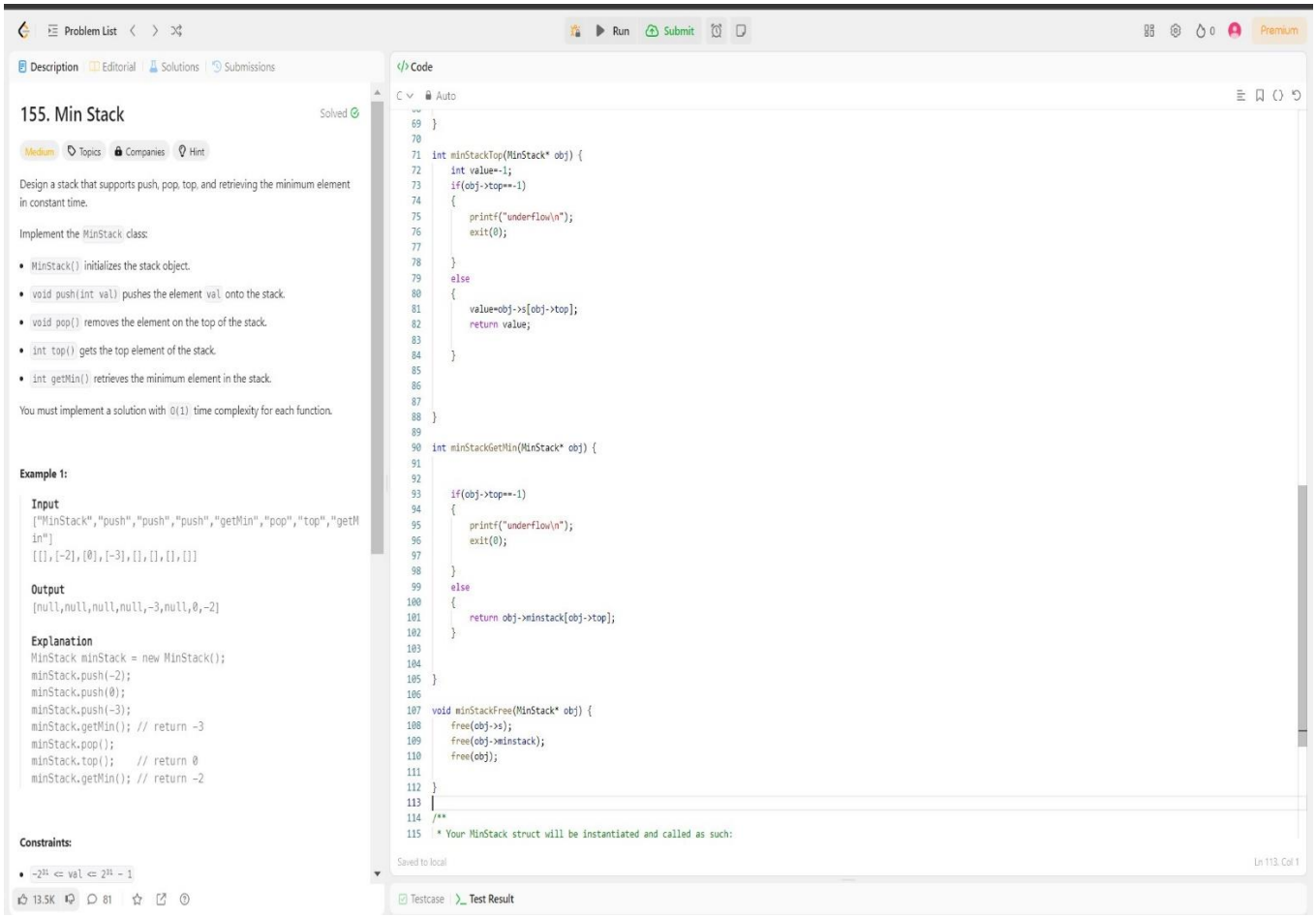
```c

typedef struct {
    int size;
    int top;
    int *s;
    int *minstack;

} MinStack;

MinStack* minStackCreate() {
    MinStack *st=(MinStack*) malloc(sizeof(MinStack));
    if(st==NULL)
    {
        printf("memory alloction failed");
        exit(0);
    }
    st->size=5;
    st->top=-1;
    st->s=(int*) malloc (st->size*sizeof(int));
    st->minstack = (int*) malloc (st->size * sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;

}
void minStackPush(MinStack* obj, int val) {
    if(obj->top==obj->size-1)
    {
        printf("stack is overflow");

    }
    else{
        obj->top++;
        obj->s[obj->top]=val;
        if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
            obj->minstack[obj->top] = val;
        } else {
```

Saved to local

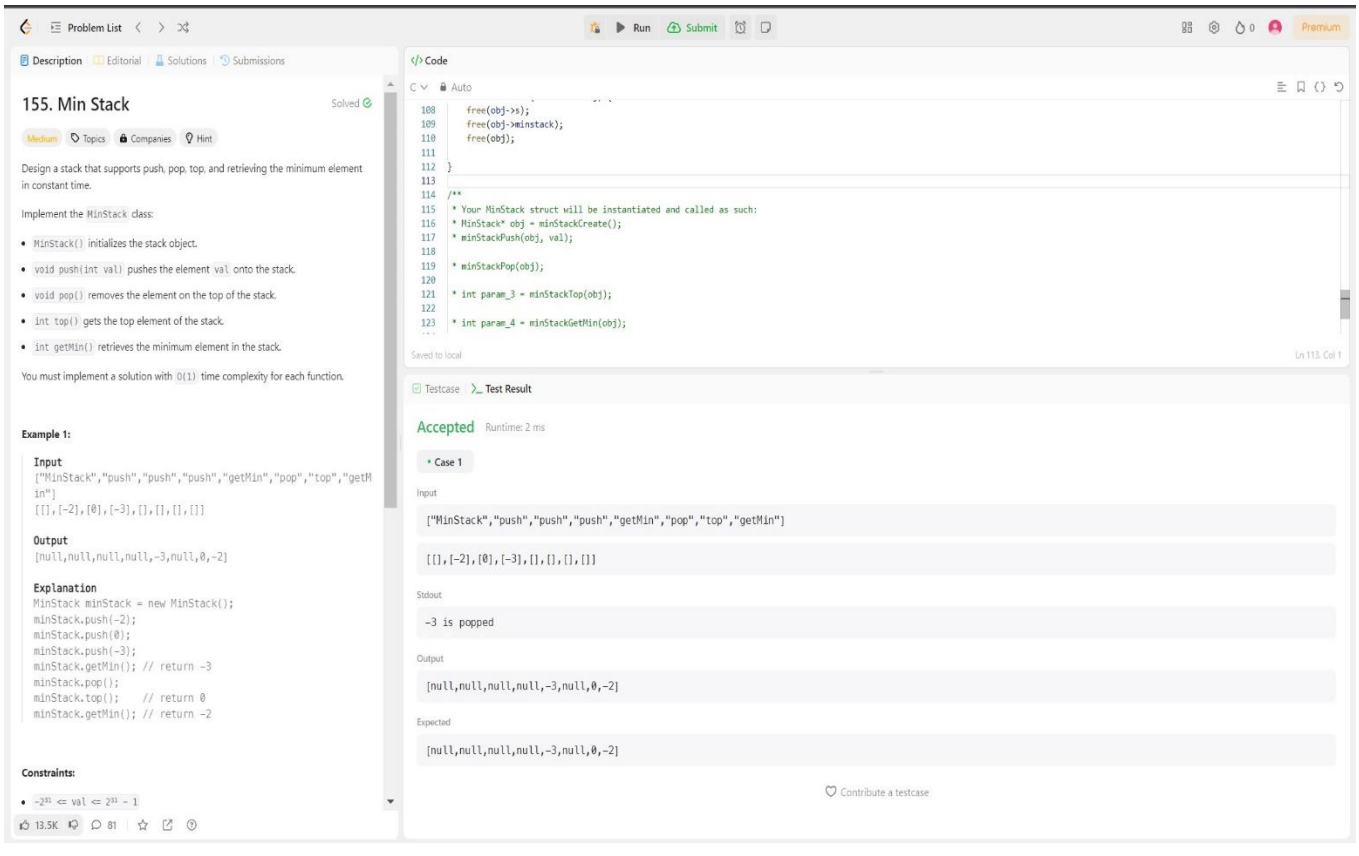
Ln 27, Col 44

Testcase | Test Result


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
Code
155. Min Stack
Solved
Medium
Topics
Companies
Hint
Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
Implement the MinStack class:
MinStack() initializes the stack object.
void push(int val) pushes the element val onto the stack.
void pop() removes the element on the top of the stack.
int top() gets the top element of the stack.
int getMin() retrieves the minimum element in the stack.
You must implement a solution with O(1) time complexity for each function.
Example 1:
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Output
[null,null,null,null,-3,null,0,-2]
Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2
Constraints:
-2^31 <= val <= 2^31 - 1
13.5K
81
C
Auto
return st;
}
void minStackPush(MinStack* obj, int val) {
if(obj->top==obj->size-1)
{
printf("stack is overflow");
}
else{
obj->top++;
obj->s[obj->top]=val;
if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
obj->minstack[obj->top] = val;
} else {
obj->minstack[obj->top] = obj->minstack[obj->top - 1];
}
}
}
void minStackPop(MinStack* obj) {
int value;
if(obj->top==-1)
{
printf("underflow");
}
else
{
value=obj->s[obj->top];
obj->top--;
printf("%d is popped\n",value);
}
}
int minStackTop(MinStack* obj) {
int value=-1;
if(obj->top==-1)
{
Saved to local
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
Code
155. Min Stack
Solved
Medium
Topics
Companies
Hint
Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
Implement the MinStack class:
MinStack() initializes the stack object.
void push(int val) pushes the element val onto the stack.
void pop() removes the element on the top of the stack.
int top() gets the top element of the stack.
int getMin() retrieves the minimum element in the stack.
You must implement a solution with O(1) time complexity for each function.
Example 1:
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Output
[null,null,null,null,-3,null,0,-2]
Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2
Constraints:
-2^31 <= val <= 2^31 - 1
13.5K
81
Auto
Saved to local
Ln 113, Col 1
Testcase
Test Result

Output


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
Code
155. Min Stack
Solved
Medium
Topics
Companies
Hint
Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
Implement the MinStack class:
MinStack() initializes the stack object.
void push(int val) pushes the element val onto the stack.
void pop() removes the element on the top of the stack.
int top() gets the top element of the stack.
int getMin() retrieves the minimum element in the stack.
You must implement a solution with O(1) time complexity for each function.
Example 1:
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Output
[null,null,null,null,-3,null,0,-2]
Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2
Constraints:
-2^31 <= val <= 2^31 - 1
13.5K
81
Auto
Saved to local
Ln 113, Col 1
Testcase
Test Result
Accepted
Runtime: 2 ms
Case 1
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Stdout
-3 is popped
Output
[null,null,null,null,-3,null,0,-2]
Expected
[null,null,null,null,-3,null,0,-2]
Contribute a testcase

Code

```
typedef struct {

    int size;
    int top;
    int *s;
    int *minstack;

} MinStack;


MinStack* minStackCreate() {
    MinStack *st=(MinStack*) malloc(sizeof(MinStack));
    if(st==NULL)
    {
        printf("memory alloction failed");
        exit(0);
    }
    st->size=5;
    st->top=-1;
    st->s=(int*) malloc (st->size*sizeof(int));
    st->minstack = (int*) malloc (st->size * sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;

}

void minStackPush(MinStack* obj, int val) {
    if(obj->top==obj->size-1)
    {
        printf("stack is overflow");

    }
    else{
        obj->top++;
        obj->s[obj->top]=val;
        if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
            obj->minstack[obj->top] = val;
        } else {
            obj->minstack[obj->top] = obj->minstack[obj->top - 1];
        }
    }


}
```

```c
void minStackPop(MinStack* obj) {
    int value;
    if(obj->top==-1)
    {
        printf("underflow");

    }
    else
    {
        value=obj->s[obj->top];
        obj->top--;
        printf("%d is popped\n",value);
    }

}

int minStackTop(MinStack* obj) {
    int value=-1;
    if(obj->top==-1)
    {
        printf("underflow\n");
        exit(0);

    }
    else
    {
        value=obj->s[obj->top];
        return value;

    }

}

int minStackGetMin(MinStack* obj) {
    if(obj->top==-1)
    {
        printf("underflow\n");
        exit(0);

    }
    else
    {
        return obj->minstack[obj->top];
    }

}

void minStackFree(MinStack* obj) {
    free(obj->s);
    free(obj->minstack);
```

```
    free(obj);

}


/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
 * minStackPush(obj, val);

 * minStackPop(obj);

 * int param_3 = minStackTop(obj);

 * int param_4 = minStackGetMin(obj);

 * minStackFree(obj);
*/
```