# 剑指 Offer 30 包含min函数的栈

> Label：栈
>
> 定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)。

- 添加 list

```
class MinStack {
    List<Integer> list;
    Stack<Integer> stack;
    public MinStack() {
        list = new ArrayList<>();
        stack = new Stack<>();
    }
    public void push(int x) {
        this.stack.push(x);
        this.list.add(x);
    }
    public void pop() {
        int a = this.stack.pop();
        this.list.remove(list.indexOf(a));
    }
    public int top() {return this.stack.peek();}
    public int min() {
        Collections.sort(list);
        return list.get(0);
    }
}
```

- 辅助单调栈

```
class MinStack {
    Stack<Integer> A, B;
    public MinStack() {
        A = new Stack<>();
        B = new Stack<>();
    }
    public void push(int x) {
        A.add(x);
        if(B.empty() || B.peek() >= x) // 保存当前最小就行
            B.add(x);
    }
    public void pop() {
        if(A.pop().equals(B.peek())) // 最小值出栈，B才用动
            B.pop();
    }
    public int top() {return A.peek();}
    public int min() {return B.peek();}
}
```

- 倒序链表 每个节点都保存 min

```
class MinStack {
    private Node head;
    public MinStack() {}
    public void push(int x) {
        if (head == null) {head = new Node(x, x, null);}
        else {head = new Node (x, Math.min(x, head.min), head);}
    }
    public void pop() {head = head.next;}
    public int top() {return head.val;}
    public int min() {return head.min;}
    class Node {
        public int val;
        public int min;
        public Node next;
        Node(int val, int min, Node next) {
            this.val = val;
            this.min = min;
            this.next = next;
        }
    }
}
```

- 辅助栈

```
class MinStack {
        Stack<Integer> stack;
        Stack<Integer> minMin;  // 一直都保存当前对应最小值
    public MinStack() {
        stack = new Stack<>();
        minMin = new Stack<>();
        minMin.push(Integer.MAX_VALUE);// 先预存一个最大值
    }
    public void push(int x) {
        stack.push(x);
        minMin.push(Math.min(minMin.peek(), x)); // 类似链表，每次都保存最小节点
    }
    public void pop() {
        minMin.pop();
        stack.pop();
    }
    public int top() {return stack.peek();}
    public int min() {return minMin.peek();}
}
```