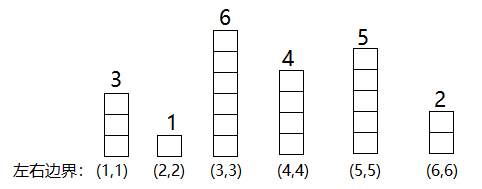
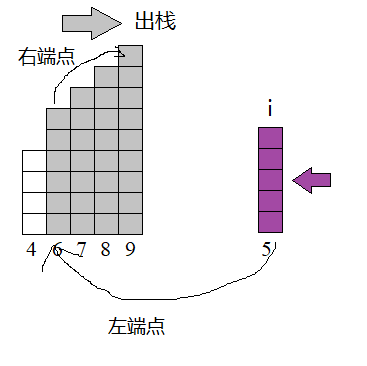
只需一次遍历，维护一个单调递增的栈。

初始时，每个数的左右边界均为自身，例如样例3 1 6 4 5 2中每个数的左右边界如图所示。



假设依次入栈并维护单调栈如图所示：



那么对于要出栈的栈顶top元素来说，以此元素为最低点，则左边是top-1的右边一个,因为左边比它大的都弹出去了，留下来的左边第一个是比它小的元素。右边是i-1的位置，因为一旦top要弹出，说明右边比它大的情况结束了，进来了一个小的值。

当前栈中每一个元素的右端点至少都是当前位置，左端点则都是之前已记录位置。

踢掉元素进行更新的时候要注意将踢掉元素的左端点的值继承下来，因为踢掉它们意味着它们比当前元素要大，所以当前元素的左端点必然能延伸到它们能延伸到的位置。

由上述过程可看出，当每个元素被踢出来时，其左端点值和右端点值都最终确定了，因而可以计算以它为最小值的这一段对应的答案。

此外，还要注意在最后补上一个最小元素（-1），这是为了保证所有的元素最终都能被踢出来。

以样例来说

3 1 6 4 5 2

3进栈，此时堆栈为3(1，？)

1进栈前踢掉3，此时3的右边界肯定是它自己，即3 [1,1]

此时堆栈为1(1，？)，因为3的位置肯定是1的左边界

6进栈，此时堆栈为1(1，？), 6(3，？)，因为左边的1肯定比6小，所以6的左边界为自身位置

4进栈前踢掉6[3,3]， 因为4比6小，所以6自身位置即为右边界

此时堆栈里为1(1，？), 4(3，？)

5进栈，此时堆栈为 1(1，？), 4(3，？), 5(5，？)

2进栈前踢掉5[5,5]，踢掉4[3,5]，为什么4的右边界是5呢，因为踢掉它们意味着它们比当前元素要大，所以当前元素的左端点必然能延伸到它们能延伸到的位置。

因为是递增栈，所以右边界是5的位置

此时堆栈为1(1，？), 2(6，？)

为了清空堆栈，-1进栈，踢掉2[6,6]，踢掉1[1,6]，为什么1的右边界是6呢，因为踢掉它们意味着它们比当前元素要大，所以当前元素的左端点必然能延伸到它们能延伸到的位置。

-1(7)