Day2

977. Squares of a Sorted Array

Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

Example 1:

```
Input: nums = [-4, -1, 0, 3, 10]
```

Output: [0,1,9,16,100]

Explanation: After squaring, the array becomes

[16,1,0,9,100].

After sorting, it becomes [0,1,9,16,100].

Example 2:

```
Input: nums = [-7, -3, 2, 3, 11]
```

Output: [4,9,9,49,121]

Constraints:

- 1 <= nums.length <= 10^4
- $-10^4 <= nums[i] <= 10^4$
- nums is sorted in **non-decreasing** order.

Follow up: Squaring each element and sorting the new array is very trivial, could you find an O(n) solution using a different approach?

```
public class Squares_of_a_Sorted_Array_977_E {
```

```
public int[] sortedSquares(int[] nums) {
    int left = 0;
    int right = nums.length - 1;
    int[] result = new int[nums.length];
    int index = result.length - 1;

    while (left <= right){
        if(nums[left] * nums[left] >= nums[right] * nums[right]){
            result[index--] = nums[left] * nums[left];
            left++;
        }else {
            result[index--] = nums[right] * nums[right];
            right--;
        }
    }
    return result;
}
```

209. Minimum Size Subarray Sum

Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example 1:

```
Input: target = 7, nums = [2,3,1,2,4,3]
```

Output: 2

Explanation: The subarray [4,3] has the minimal length

under the problem constraint.

Example 2:

```
Input: target = 4, nums = [1,4,4]
Output: 1
```

Example 3:

```
Input: target = 11, nums = [1,1,1,1,1,1,1]
Output: 0
```

Constraints:

- 1 <= target <= 10⁹
- 1 <= nums.length <= 10^5
- 1 <= nums[i] <= 10⁴

Follow up: If you have figured out the O(n) solution, try coding another solution of which the time complexity is $O(n \log(n))$.

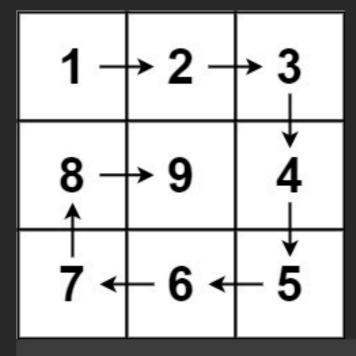
```
class Solution {
   public int minSubArrayLen(int target, int[] nums) {
      int result = Integer.MAX_VALUE;
      int left = 0;
      int subLength = 0;
      int sum = 0;
}
```

```
for (int right = 0; right < nums.length ; right++) {
    sum += nums[right];
    while (sum >= target){
        subLength = right - left + 1;
        sum -= nums[left++];
        result = result > subLength ? subLength : result;
    }
}
return result == Integer.MAX_VALUE ? 0 : result;
}
```

59. Spiral Matrix II

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

Example 1:



```
Input: n = 3
Output: [[1,2,3],[8,9,4],[7,6,5]]
```

Example 2:

```
Input: n = 1
Output: [[1]]
```

```
class Solution {
    public int[][] generateMatrix(int n) {
        int loop = 0;
        int start = 0;
        int[][] arr = new int[n][n];
        int count = 1;
        int i, j;

    while (loop++ < n / 2){
            //左到右</pre>
```

```
for (j = start; j < n - loop; j++) {
        arr[start][j] = count++;
    //右向下
    for (i = start; i < n - loop; i++) {
       arr[i][j] = count++;
    //右到左
    for (; j >= loop ; j--) {
       arr[i][j] = count++;
    //下向上
    for (; i >= loop; i--){
       arr[i][j] = count++;
    start++;
if (n \% 2 == 1){
   arr[start][start] = count;
return arr;
```