数组

1.二分法

前提，有序的数组，并且没有重复元素

```
static void Main(int [] a,int target)
{
  int left = 0;
  int right = a.Length - 1;
  while (left <= right)
  {
    int middle = left + ((left - right) / 2);
    if (a[middle] > target)
    {
      right = middle - 1;
    }
    else if(a[middle]<target)
    {
        left = middle + 1;
    }
    else if (a[middle] == target)
    {
    return middle;//返回下标;
    }
  }
}
```

2.移除元素(双指针操作)

给你一个数组 nums 和一个值 val，你需要 原地 移除所有数值等于 val 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并**原地**修改输入数组。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

示例 1: 给定 nums = [3,2,2,3], val = 3, 函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。 你不需要考虑数组中超出新长度后面的元素。

示例 2: 给定 nums = [0,1,2,2,3,0,4,2], val = 2, 函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。

```
pcublic class Solution {
```

```
    public int RemoveElement(int[] nums, int val) {
        int slow = 0;
        for (int fast = 0; fast < nums.Length; fast++) {
            if (val != nums[fast]) {
                nums[slow] = nums[fast];
                slow++;
            }
        }
        return slow;
    }
}
```

3，有序数组的平方

给你一个按 非递减顺序 排序的整数数组 nums，返回 每个数字的平方 组成的新数组，要求也按 非递减顺序 排序。

示例 1： 输入：nums = [-4,-1,0,3,10] 输出：[0,1,9,16,100] 解释：平方后，数组变为 [16,1,0,9,100]，排序后，数组变为 [0,1,9,16,100]

示例 2： 输入：nums = [-7,-3,2,3,11] 输出：[4,9,9,49,121]

```
public class Solution {
    public int[] SortedSquares(int[] nums) {
        int k = nums.Length - 1;
        int[] result = new int[nums.Length];
        for (int i = 0, j = nums.Length - 1;i <= j;)
        {

            if (nums[i] * nums[i] < nums[j] * nums[j])
            {
                result[k] = nums[j] * nums[j];
              k--;
                j--;
            }
           else
           {
                result[k--] = nums[i] * nums[i];
                i++;
            }
        }
        return result;
    }
}
```

4，长度最小的子数组

给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 ≥ s 的长度最小的 连续 子数组，并返回其长度。如果不存在符合条件的子数组，返回 0。

示例：

输入：s = 7, nums = [2,3,1,2,4,3] 输出：2 解释：子数组 [4,3] 是该条件下的长度最小的子数组。

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int result = INT32_MAX;
        int sum = 0; // 滑动窗口数值之和
        int i = 0; // 滑动窗口起始位置
        int subLength = 0; // 滑动窗口的长度
        for (int j = 0; j < nums.size(); j++) {
            sum += nums[j];
            // 注意这里使用while，每次更新 i（起始位置），并不断比较子序列是否符合条件
            while (sum >= s) {
                subLength = (j - i + 1); // 取子序列的长度
                result = result < subLength ? result : subLength;
                sum -= nums[i++]; // 这里体现出滑动窗口的精髓之处，不断变更i（子序列的起始位置）
            }
        }
        // 如果result没有被赋值的话，就返回0，说明没有符合条件的子序列
        return result == INT32_MAX ? 0 : result;
    }
};

public class Solution {
    public int MinSubArrayLen(int s, int[] nums) {
        int n = nums.Length;
        int ans = int.MaxValue;
        int start = 0,// 滑动窗口起始位置
        end = 0;// 滑动窗口的长度
        int sum = 0;// 滑动窗口数值之和
        while (end < n)  {
            sum += nums[end];
            while (sum >= s)
            {
                ans = Math.Min(ans, end - start + 1);// 取子序列的长度
                sum -= nums[start];
                start++;
            }
            end++;
```

```
        }
        return ans == int.MaxValue ? 0 : ans;
    }
}
```

5，螺旋矩阵

给定一个正整数 n，生成一个包含 1 到 n^2 所有元素，且元素按顺时针顺序螺旋排列的正方形矩阵。

示例:

输入: 3 输出: [[1, 2, 3], [8, 9, 4], [7, 6, 5]]

```
class Solution {
public:
    vector<vector<int>> generateMatrix(int n) {
        vector<vector<int>> res(n, vector<int>(n, 0)); // 使用vector定义一个二维数组
        int startx = 0, starty = 0; // 定义每循环一个圈的起始位置
        int loop = n / 2; // 每个圈循环几次，例如n为奇数3，那么loop = 1 只是循环一圈，矩阵中间的值需要单独处理
        int mid = n / 2; // 矩阵中间的位置，例如：n为3， 中间的位置就是(1，1)，n为5，中间位置为(2, 2)
        int count = 1; // 用来给矩阵中每一个空格赋值
        int offset = 1; // 需要控制每一条边遍历的长度，每次循环右边界收缩一位
        int i,j;
        while (loop --) {
            i = startx;
            j = starty;

            // 下面开始的四个for就是模拟转了一圈
            // 模拟填充上行从左到右(左闭右开)
            for (j = starty; j < n - offset; j++) {
                res[startx][j] = count++;
            }
            // 模拟填充右列从上到下(左闭右开)
            for (i = startx; i < n - offset; i++) {
                res[i][j] = count++;
            }
            // 模拟填充下行从右到左(左闭右开)
            for (; j > starty; j--) {
                res[i][j] = count++;
            }
            // 模拟填充左列从下到上(左闭右开)
            for (; i > startx; i--) {
                res[i][j] = count++;
            }

```

```
        // 第二圈开始的时候，起始位置要各自加1， 例如：第一圈起始位置是(0, 0)，第二圈起始位置是(1, 1)
        startx++;
        starty++;

        // offset 控制每一圈里每一条边遍历的长度
        offset += 1;
    }

    // 如果n为奇数的话，需要单独给矩阵最中间的位置赋值
    if (n % 2) {
        res[mid][mid] = count;
    }
    return res;
  }
};

C#:
public class Solution {
  public int[][] GenerateMatrix(int n) {
    int[][] answer = new int[n][];
    for(int i = 0; i < n; i++)
        answer[i] = new int[n];
    int start = 0;
    int end = n - 1;
    int tmp = 1;
    while(tmp < n * n)
    {
        for(int i = start; i < end; i++) answer[start][i] = tmp++;
        for(int i = start; i < end; i++) answer[i][end] = tmp++;
        for(int i = end; i > start; i--) answer[end][i] = tmp++;
        for(int i = end; i > start; i--) answer[i][start] = tmp++;
        start++;
        end--;
    }
    if(n % 2 == 1) answer[n / 2][n / 2] = tmp;
    return answer;
  }
}
```