1.假设有一个数组，它的第 i 个元素是一个给定的股票在第 i 天的价格。

设计一个算法来找到最大的利润。你可以完成尽可能多的交易（多次买卖股票）。然而，你不能同时参与多个交易（你必须在再次购买前出售股票）。

```
public class Solution {
    public int MaxProfit(int[] prices) {
        if(prices.Length<=1||prices==null){
            return 0;
        }
        var global = 0;
        var localMin = prices[0];

        for(int i=0;i<prices.Length;i++)
        {
            var local = prices[i] - localMin;
            if(local<0)
            {
                localMin = Math.Min(localMin, prices[i]);
            }
            else{
                global += prices[i] - localMin;
               localMin = prices[i];
            }

        }
         return global;
    }
}
```

189.旋转数组

将包含 n 个元素的数组向右旋转 k 步。

例如，如果 n = 7，k = 3，给定数组 [1,2,3,4,5,6,7] ，向右旋转后的结果为 [5,6,7,1,2,3,4]。

```
public class Solution {
    public void Rotate(int[] nums, int k) {
        k = k%nums.Length;
        Reverse(nums, 0, nums.Length-1);
        Reverse(nums, 0, k-1);
        Reverse(nums, k, nums.Length-1);
    }
    ///翻转方法
       private void Reverse(int[] array, int first, int last)
    {
        var i = first;
        var j = last;
        while (i < j)
        {
            var temp = array[i];
            array[i] = array[j];
            array[j] = temp;
            i++;
            j--;
        }
    }
}
```

217. 存在重复

给定一个整数数组，判断是否存在重复元素。

如果任何值在数组中出现至少两次，函数应该返回 true。如果每个元素都不相同，则返回 false。

```
public class Solution {
    public bool ContainsDuplicate(int[] nums) {
        if(nums.Length<=1) {return false;}
        for(int i=0;i<nums.Length-1;i++)
        {
            for(int j=i+1;j<nums.Length;j++)
            {
```

```
                if(nums[i]==nums[j]){
                    return true;
                }
            }
        }
        return false;
    }
}
```

66.加一
给定一个非负整数组成的非空数组，给整数加一。

可以假设整数不包含任何前导零，除了数字0本身。

最高位数字存放在列表的首位。

```
public class Solution {
    public int[] PlusOne(int[] digits) {
        int i=digits.Length-1;
        for(;i>=0;i--)
        {
            digits[i]=digits[i]+1;
           if(digits[i]==10) digits[i]=0;
            else return digits;
        }
        int [] result=new int[digits.Length+1];
        result[0]=1;
        return result;
    }
}
```

136. 只出现一次的数字
给定一个整数数组，除了某个元素外其余元素均出现两次。请找出这个只出现一次的元素。

```
  public class Solution
    {
        public int SingleNumber(int[] nums)
```

```
    {
        int num = 0;
        for (int i = 0; i < nums.Length; i++)
        {
            num = nums[i] ^ num;
        }
        return num;
    }
  }
```

排序

```
 static void Main(string[] args)
    {
        Console.Write("请输入字符串：");

        string Str = Console.ReadLine();
        //去掉汉字
        Str = Regex.Replace(Str, @"[\u4e00-\u9fa5]", "");

        //去重
        char[] CharStr = Str.ToArray().Distinct().ToArray();
        string Strs = string.Join("", CharStr);
        char[] nums=new Char[10];
        char[] letter = new Char[20];
        //排序
        int aflag = 0;
        int bflag = 0;

        for (int i = 0; i < Strs.Length; i++)
        {
            //如果为数字
            if (Strs[i] >= '0' && Strs[i] <= '9')
            {
                //如果为数字 则放在容器A中
                nums[aflag] = Strs[i];
```

```
                aflag++;
            }
            else {
                //如果为其他字符 则放在容器A中
                letter[bflag] = Strs[i];
                bflag++;
            }
        }
        string snums= string.Join("", nums);
        snums=snums.Trim('\0');
        string bnums = string.Join("", letter);
        bnums=bnums.Trim('\0');
        var a = snums + bnums;
        Console.Write(a);
        Console.ReadLine();
    }
```

LeetCode 27题

给定一个数组和一个值，在这个数组中原地移除指定值和返回移除后新的数组长度。

不要为其他数组分配额外空间，你必须使用 O(1) 的额外内存原地修改这个输入数组。

元素的顺序可以改变。超过返回的新的数组长度以外的数据无论是什么都没关系。

示例:

给定 nums = [3,2,2,3]，val = 3，

你的函数应该返回 长度 = 2，数组的前两个元素是 2。

```
public class Solution {
    public int RemoveElement(int[] nums, int val) {
     int flag =0;
     for (int i=0;i<nums.Length;i++) {
         if(nums[i]!=val)
         {
             nums[flag]=nums[i];
             flag++;
```

```
        }
    }
        return flag;
    }
}
```

反转字符串

```
public class Solution {
    public string ReverseString(string s) {
    if(String.IsNullOrEmpty(s))
    {
        return s;
    }
        StringBuilder sb = new StringBuilder();
        for(int i=s.Length-1;i>=0;i--)
        {
            sb.Append(s[i].ToString());
        }
        return sb.ToString();

}
}
```

给定一个整数数组和一个目标值，找出数组中和为目标值的**两个**数。
你可以假设每个输入只对应一种答案，且同样的元素不能被重复利用。

```
public class Solution {
public int[] TwoSum(int[] nums, int target) {
        //判断为空
        if (nums==null||nums.Length<2)    {
           throw new ArgumentException("Invalid integer array is passed.");
        }
        //定义数组参数
        int[] cren=new int[2];
        int difference = 0;
       Dictionary<int, int> differences = new Dictionary<int, int>();
```

```
    for (int i=0;i<nums.Length;i++)
    {
        int X=nums[i];
        difference=target-X;

        if(differences.TryGetValue(difference, out int firstIndex))
        {
            cren[0]=firstIndex;
            cren[1]=i;
            break;
        }
        else if(!differences.ContainsKey(X))
        {
            differences.Add(X,i);
        }
    }
     return cren;
 }
}
```

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

示例:

输入： `[0,1,0,3,12]`
输出： `[1,3,12,0,0]`
说明:

1. 必须在原数组上操作，不能拷贝额外的数组。
2. 尽量减少操作次数

```
public class Solution {
    public void MoveZeroes(int[] nums) {
        int size=nums.Length;
        int StartFlag=0;
```

```
    int EndFlag=0;
    int NumFlag=0;
    int i=0;
    while(i<size)
    {

        NumFlag=nums[i];
        //此步骤为了找到第一个为0的位置
        if(NumFlag==0)
        {
            StartFlag=i;
            EndFlag= i;
            break;
        }
        ++i;
    }
    if(nums[EndFlag]!=0)
        return;
    // 将当前i的值加1；直接从刚才0元素位置的后一位置开始循环
    ++i;
    while(i<size){
        NumFlag=nums[i];
        if(NumFlag==0){
            EndFlag=i;
        }
        else{
            nums[StartFlag]=NumFlag;
            nums[i]=0;
            ++StartFlag;
            ++EndFlag;
        }
        ++i;
    }
    return;
}
```

}

给定一个**非负整数**组成的**非空**数组，在该数的基础上加一，返回一个新的数组。
最高位数字存放在数组的首位， 数组中每个元素只存储一个数字。
你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:

输入: `[1,2,3]`
输出: `[1,2,4]`
解释: 输入数组表示数字 `123`。

示例 2:

输入: `[4,3,2,1]`
输出: `[4,3,2,2]`
解释: 输入数组表示数字 `4321`。

```
public class Solution {
    public int[] PlusOne(int[] digits) {
        int i=digits.Length-1;
        for(;i>=0;i--)
        {
            digits[i]=digits[i]+1;
           if(digits[i]==10) digits[i]=0;
            else return digits;
        }
         int [] result=new int[digits.Length+1];
        result[0]=1;
        return result;
    }
}
```

两个数组的交集 II

给定两个数组，写一个方法来计算它们的交集。

例如:

给定 *nums1* = `[1, 2, 2, 1]`, *nums2* = `[2, 2]`, 返回 `[2, 2]`.

注意:

- 输出结果中每个元素出现的次数，应与元素在两个数组中出现的次数一致。
- 我们可以不考虑输出结果的顺序。

跟进:

- 如果给定的数组已经排好序呢？你将如何优化你的算法?
- 如果 *nums1* 的大小比 *nums2* 小很多，哪种方法更优?
- 如果*nums2*的元素存储在磁盘上，内存是有限的，你不能一次加载所有的元素到内存中，你该怎么办?

```
public class Solution
{
    public int[] Intersect(int[] nums1, int[] nums2)
        {
        var numAndCount1 = new Dictionary<int, int>();

            foreach(var num in nums1)
            {
                if (!numAndCount1.ContainsKey(num))        numAndCount1[num] =
0;

                numAndCount1[num]++;
            }

            var result = new List<int>();

            foreach(var num in nums2)
            {
                if (numAndCount1.ContainsKey(num) && numAndCount1[num] > 0)
                {
                    result.Add(num);

                    numAndCount1[num]--;
                }
            }
```

```
            return result.ToArray();
    }
}
```

只出现一次的数字

给定一个**非空**整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

示例 1:

输入： `[2,2,1]`

输出： `1`

示例 2:

输入： `[4,1,2,1,2]`

输出： `4`

```
public class Solution {
    public int SingleNumber(int[] nums) {
        int num = 0;
        for(int i = 0; i < nums.Length; i++) {
            num = nums[i]^num;
        }
        return num;
    }
}
```

给定一个数组，它的第 *i* 个元素是一支给定股票第 *i* 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

输入： `[7,1,5,3,6,4]`

输出： `7`

解释： 在第 `2` 天（股票价格 `=` `1`）的时候买入，在第 `3` 天（股票价格 `=` `5`）的时候卖出，这笔交易所能获得利润 `=` `5-1` `=` `4` 。

随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6-3 = 3 。

```
public class Solution {
    public int MaxProfit(int[] prices) {
        if(prices.Length<=1||prices==null){
            return 0;
        }
        var global = 0;
        var localMin = prices[0];

        for(int i=0;i<prices.Length;i++)
        {
            var local = prices[i] - localMin;
            if(local<0)
            {
                localMin = Math.Min(localMin, prices[i]);
            }
            else{
                global += prices[i] - localMin;
               localMin = prices[i];
            }

        }
         return global;
    }
}
```

48. 旋转图像

给定一个 *n* × *n* 的二维矩阵表示一个图像。

将图像顺时针旋转 90 度。

说明：

你必须在**原地**旋转图像，这意味着你需要直接修改输入的二维矩阵。**请不要**使用另一个矩阵来旋转图像。

示例 1:

给定 **matrix** =
```
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],
```

原地旋转输入矩阵，使其变为：
```
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```
示例 2:
给定 **matrix** =
```
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],
```

原地旋转输入矩阵，使其变为：
```
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int r = matrix.size();
        int c = matrix[0].size();
        for(int i=0,j=r-1;i<(r/2);i++,j--)
            swap(matrix[i],matrix[j]);
        for(int i=0;i<r;i++){
            for(int j=0;j<i;j++){
                swap(matrix[i][j],matrix[j][i]);
            }
        }
    }
};
```