121. 买卖股票的最佳时机

```
func maxProfit(prices []int) int {
    n := len(prices)
    max := make([]int, n)
    curMax := 0
    for i := n-1; i >= 0; i-- {
        max[i] = curMax
        if prices[i] > curMax {curMax = prices[i]}
    }
    result := 0
    for i := 0; i < n; i++ {
        if result < max[i]-prices[i] {
            result = max[i]-prices[i]
        }
    }
    return result
}
```

238. 除自身以外数组的乘积

```
func productExceptSelf(nums []int) []int {
    n := len(nums)
    leftProducts := make([]int, n)
    rightProducts := make([]int, n)
    product := 1
    for i := 0; i < n; i++ {
        product *= nums[i]
        leftProducts[i] = product
    }
    product = 1
    for i := n-1; i >= 0; i-- {
        product *= nums[i]
        rightProducts[i] = product
    }
     result := make([]int, n)
     for i := 0; i < n; i++ {
         result[i] = 1
         if i-1 >= 0 {
             result[i] *= leftProducts[i-1]
         }
         if i+1 < n {
             result[i] *= rightProducts[i+1]
         }
     }
     return result
}

func productExceptSelf(nums []int) []int {
    n := len(nums)
    result := make([]int, n)
    leftProduct := 1
    for i := 0; i < n; i++ {
        result[i] = leftProduct
        leftProduct *= nums[i]
    }
    rightProduct := 1
    for i := n-1; i >= 0; i-- {
        result[i] *= rightProduct
        rightProduct *= nums[i]
```

```
    }
    return result
}
```

面试题 05.03. 翻转数位

```
func reverseBits(num int) int {
    if num == 0 {return 1}
    nums := make([]int, 32)
    for i := 0; i < 32; i++ {
        nums[i] = (num&1)
        num >>= 1
    }
    leftOneCounts := make([]int, 32)
    count := 0
    for i := 0; i < 32; i++ {
        leftOneCounts[i] = count
        if nums[i] == 1 {
            count++
        } else {
            count = 0
        }
    }
    rightOneCounts := make([]int, 32)
    count = 0
    for i := 31; i >= 0; i-- {
        rightOneCounts[i] = count
        if nums[i] == 1 {
            count++
        } else {
            count = 0
        }
    }
    ret := 0
    for i := 0; i < 32; i++ {
        if ret < leftOneCounts[i] + rightOneCounts[i] + 1{
            ret = leftOneCounts[i] + rightOneCounts[i] + 1
        }
    }
    return ret
}
```

42. 接雨水

```
func trap(height []int) int {
   n := len(height)
   result := 0
   //遍历每个柱子 n，查找它左边的最高柱子 lh，和有变得最高柱子 rh
   //柱子上能承载的雨水=min(lh,rh)-h
   for i := 1; i < n-1; i++ {
       lh := 0
       for j := 0; j < i; j++ { // 左侧最高 lh
           if height[j] > lh {lh = height[j]}
       }
       rh := 0
       for j := i+1; j < n; j++ { // 右侧最高 rh
           if height[j] > rh {rh = height[j]}
        }
```

```
        carry := int(math.Min(float64(lh), float64(rh))) - height[i]
        if carry < 0 {carry = 0}
        result += carry
    }
    return result
}

//解法 2
func trap(height []int) int {
    n := len(height)
    // 前缀 max
    leftMax := make([]int, n)
    max := 0
    for i := 0; i < n; i++ {
        leftMax[i] = int(math.Max(float64(max), float64(height[i])))
        max = leftMax[i]
    }
    // 后缀 max
    rightMax := make([]int, n)
    max = 0
    for i := n-1; i >= 0; i-- {
        rightMax[i] = int(math.Max(float64(max), float64(height[i])))
        max = rightMax[i]
    }
    // 每个柱子上承载的雨水
    result := 0
    for i := 1; i < n-1; i++ {
        result += int(math.Min(float64(leftMax[i]), float64(rightMax[i]))) - height[i]
    }
    return result
}
```

53. 最大子序和

```
//滑动窗口
func maxSubArray(nums []int) int {
    n := len(nums)
    maxSum := math.MinInt32
    sum := 0
    for i := 0; i < n; i++ {
        if sum < 0 {
            sum = 0
        }
        sum += nums[i]
        if sum > maxSum {
            maxSum = sum
        }
    }
    return maxSum
}

//前后缀
func maxSubArray(nums []int) int {
    if len(nums) == 1 {return nums[0]}
    sum := make([]int, len(nums))
    max := make([]int, len(nums))
    cursum := 0
    for i := 0; i < len(nums); i++ {
```

```
        cursum += nums[i]
        sum[i] = cursum
    }
    curmax := math.MinInt32
    for i := len(sum)-1; i >= 0; i-- {
        if curmax < sum[i] {curmax = sum[i]}
        max[i] = curmax
    }
    result := math.MinInt32
    for i := 0; i < len(nums); i++ {
        if i == 0 && result < max[0] {result = max[0]}
        if i != 0 && result < max[i]-sum[i-1] {result = max[i]-sum[i-1]}
    }
    return result
}
```

191. 位 1 的个数

```
func hammingWeight(num uint32) int {
    oneCount := 0
    var mask uint32 = 1
    for i := 0; i < 32; i++ {
        if num&mask != 0 {
            oneCount++
        }
        mask <<= 1
    }
    return oneCount
}

func hammingWeight(num uint32) int {
    oneCount := 0
    for num != 0 {
        if num&1 == 1 {oneCount++}
        num >>= 1
    }
    return oneCount
}
```

461. 汉明距离

```
func hammingDistance(x int, y int) int {
    r := x ^ y
    mask := 1
    count := 0
    for i := 0; i < 31; i++ {
        if r & mask != 0 {count++}
        mask *= 2
    }
    return count
}
```

面试题 05.06. 整数转换

```
func convertInteger(A int, B int) int {
    C := A ^ B
    count := 0
    for i := 0; i < 32; i++ {
        if (C&1) == 1 {
```

```
            count++
        }
        C >>= 1
    }
    return count
}
```

面试题 05.07. 配对交换

```
func exchangeBits(num int) int {
    ret := 0
    for i := 0; i <= 30; i += 2 {
        a1 := (num&(1<<i))
        b1 := (num&(1<<(i+1)))
        if a1 != 0 {ret |= (1 << (i+1))}
        if b1 != 0 {ret |= (1 << i)}
    }
    return ret
}
```

面试题 05.01. 插入

```
func insertBits(N int, M int, i int, j int) int {
    nbits := make([]int, 32)
    mbits := make([]int, 32)
    mask := 1
    for k := 0; k < 32; k++ {
        if N&mask != 0 {
            nbits[k] = 1
        }
        mask <<= 1
    }
    mask = 1
    for k := 0; k < 32; k++ {
        if (M&mask) != 0 {
            mbits[k] = 1
        }
        mask <<= 1
    }

    for k := i; k <= j; k++ {
        nbits[k] = mbits[k-i]
    }
    mask = 1
    ret := 0
    for k := 0; k < 32; k++ {
        ret += (nbits[k] * mask)
        mask <<= 1
    }
    return ret
}
```

面试题 17.04. 消失的数字

```
func missingNumber(nums []int) int {
    n := len(nums)
    ret := 0
    for i := 0; i <= n; i++ {
        ret ^= i
```

```
    }
    for i := 0; i < n; i++ {
        ret ^= nums[i]
    }
    return ret
}
```

剑指 Offer 56 - I. 数组中数字出现的次数

```
func singleNumbers(nums []int) []int {
    xorRexult := 0
    n := len(nums)
    for i := 0; i < n; i++ {
        xorRexult ^= nums[i]
    }
    tag := 1
    for (xorRexult & tag) == 0 {
        tag = tag << 1
    }
    a := 0
    b := 0
    for i := 0; i < n; i++ {
        if (nums[i] & tag) == 0 {
            a ^= nums[i]
        } else {
            b ^= nums[i]
        }
    }
    return []int{a,b}
}
```

剑指 Offer 56 - II. 数组中数字出现的次数 II

```
func singleNumber(nums []int) int {
    n := len(nums)
    bits := make([]int, 32)
    mask := 1
    for i := 0; i < 32; i++ {
        for j := 0; j < n; j++ {
            if (nums[j]&mask) != 0 {
                bits[i] = (bits[i] + 1) % 3
            }
        }
        mask <<= 1
    }
    result := 0
    mask = 1
    for i := 0; i < 32; i++ {
        if bits[i] == 1 {
            result += mask
        }
        mask <<= 1
    }
    return result
}
```

面试题 16.01. 交换数字

```go
func swapNumbers(numbers []int) []int {
    if numbers[0] == numbers[1] {return numbers}
    numbers[0] ^= numbers[1]
    numbers[1] ^= numbers[0]
    numbers[0] ^= numbers[1]
    return numbers
}
```

231. 2 的幂

```go
func isPowerOfTwo(n int) bool {
    for n != 0 {
        if (n&1) == 1 {
            if (n >> 1) == 0 {
                return true
            } else {
                return false
            }
        }
        n >>= 1
    }
    return false
}
```