剑指 Offer 10- I. 斐波那契数列

```
//解法 1 递归 超出时间限制
const mod = 1000000007
func fib(n int) int {
    if n == 0 {return 0}
    if n == 1 {return 1}
    return (fib(n-1)+fib(n-2))%mod
}

//解法 2 递归 + 备忘录
const mod = 1000000007
var memo []int
func fib(n int) int {
    memo = make([]int, n+1)
    return fib_r(n)
}

func fib_r(n int) int{
    if n == 0 {return 0}
    if n == 1 {return 1}
    if memo[n] != 0 {return memo[n]}
    memo[n] = (fib_r(n-1)+fib_r(n-2))%mod
    return memo[n]
}
```

剑指 Offer 10- II. 青蛙跳台阶问题

```
const mod = 1000000007
var memo = make(map[int]int, 0)
func numWays(n int) int {
    if n == 0 {return 1}
    if n == 1 {return 1}
    if v, ok := memo[n]; ok{
        return v
    }
    ret := (numWays(n-1)+numWays(n-2))%mod
    memo[n] = ret
    return ret
}
```

面试题 08.01. 三步问题

```
//解法 1 递归 超时
const mod = 1000000007
func waysToStep(n int) int {
    if n == 1 {return 1}
    if n == 2 {return 2}
    if n == 3 {return 4}
    return ((waysToStep(n-1) + waysToStep(n-2))%mod + waysToStep(n-3))%mod
}

// 解法 2 递归+备忘录
const mod = 1000000007
var memo = make([]int, 1000001)
func waysToStep(n int) int {
    if n == 1 {return 1}
    if n == 2 {return 2}
    if n == 3 {return 4}
    if memo[n] != 0 {return memo[n]}
    memo[n] = ((waysToStep(n-1) + waysToStep(n-2))%mod + waysToStep(n-3))%mod
    return memo[n]
}

//解法 3 非递归实现
func waysToStep(n int) int {
      if n == 1 {return 1}
      if n == 2 {return 2}
      if n == 3 {return 4}
      dp := make([]int, n+1)
      dp[1] = 1
      dp[2] = 2
      dp[3] = 4
      for i := 4; i <= n; i++ {
          dp[i] = ((dp[i-1]+dp[i-2])%1000000007 + dp[i-3])%1000000007
      }
      return dp[n]
}

//解法 4 非递归实现+优化
func waysToStep(n int) int {
     if n == 1 {return 1}
     if n == 2 {return 2}
     if n == 3 {return 4}
     a := 1
     b := 2
     c := 4
     d := 0
     for i := 4; i <= n; i++ {
         d = ((c+b)%1000000007 + a)%1000000007
         a, b, c = b, c, d
     }
     return d
}
```

剑指 Offer 06. 从尾到头打印链表

```
//解法 1 定义全局 result
//可以不必像 java 的解法，定义 result 为 arrayList， 然后赋值给数组。
//因为 java 数组必须是固定长度的。而 go 的 slice 可变长。
var result []int
func reversePrint(head *ListNode) []int {
    //每次都重新初始化，不然 leetcode 里上一个测试用例的值会覆盖下一个测试用例的 result
    result = make([]int, 0)
    reverseTravel(head)
    return result
}

func reverseTravel(head *ListNode) {
    if head == nil {return}
    reverseTravel(head.Next)
    result = append(result, head.Val)
}
```

```
//解法 2
func reversePrint(head *ListNode) []int {
    result := make([]int, 0)
    reverseTravel(head, &result)
    return result
}

func reverseTravel(head *ListNode, result *[]int) {
    if head == nil {return }
    reverseTravel(head.Next, result)
    *result = append(*result, head.Val)
}

//解法 3
func reversePrint(head *ListNode) []int {
    if head == nil {return []int{}}
    subResult := reversePrint(head.Next)
    result := make([]int, len(subResult)+1)
    for i := 0; i < len(subResult); i++ {
        result[i] = subResult[i]
    }
    result[len(result)-1] = head.Val
    return result
}
```

剑指 Offer 24. 反转链表

```
func reverseList(head *ListNode) *ListNode {
    if head == nil {return nil}
    if head.Next == nil {return head}
    newHead := reverseList(head.Next)
    head.Next.Next = head
    head.Next = nil
    return newHead
}
```

剑指 Offer 25. 合并两个排序的链表

```
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
    if l1 == nil {return l2}
    if l2 == nil {return l1}
    if l1.Val < l2.Val {
        subHead := mergeTwoLists(l1.Next, l2)
        l1.Next = subHead
        return l1
    } else {
        subHead := mergeTwoLists(l1, l2.Next)
        l2.Next = subHead
        return l2
    }
}
```

剑指 Offer 16. 数值的整数次方

```
func myPow(x float64, n int) float64 {
    if n >= 0 {
        return rPow(x, n)
    } else {
        return 1/(rPow(x, -1*(n+1))*x)
    }
}

func rPow(x float64, n int) float64{
    if n == 0 {return 1}
    halfPow := rPow(x, n/2)
    if n % 2 == 1 {
        return halfPow * halfPow * x
    } else {
        return halfPow * halfPow
    }
}
```

面试题 08.05. 递归乘法

```
//解法 1
func multiply(a, b int) int {
    if a == 1 {return b}
    halfValue := multiply(a/2, b)
    if a%2 == 1 {
        return halfValue+halfValue+b
    } else {
        return halfValue+halfValue
    }
}

//解法 2
func multiply(a, b int) int {
    n := int(math.Min(float64(a), float64(b)))
    k := int(math.Max(float64(a), float64(b)))
    if n == 1 {return k}
    //n 个 k 相加= (n/2 个 k 相加) +(n/2 个 k 相加)+0(或 k)
    halfValue := multiply(n/2, k)
    if n%2 == 1 {
        return halfValue+halfValue+k
    } else {
        return halfValue+halfValue
    }
}
```