面试题 10.01. 合并排序的数组

```go
func merge(A []int, m int, B []int, n int) {
    k := m+n-1
    i := m-1
    j := n-1
    for i >= 0 && j >= 0 {
        if A[i] >= B[j] {
            A[k] = A[i]
            k--
            i--
        } else {
            A[k] = B[j]
            k--
            j--
        }
    }
    for i >= 0 {
        A[k] = A[i]
        k--
        i--
    }
    for j >= 0 {
        A[k] = B[j]
        k--
        j--
    }
}
```

21. 合并两个有序链表

```go
func mergeTwoLists(list1 *ListNode, list2 *ListNode) *ListNode {
    if list1 == nil {return list2}
    if list2 == nil {return list1}
    p1 := list1
    p2 := list2
    head := &ListNode{} // 虚拟头节点
    tail := head
    for p1 != nil && p2 != nil {
        if p1.Val <= p2.Val {
            tail.Next = p1
            tail = p1
            p1 = p1.Next
        } else {
            tail.Next = p2
            tail = p2
            p2 = p2.Next
        }
    }
    //如果 p1 还没处理完， 就把剩下的直接接到 tail 后面
    if p1 != nil {tail.Next = p1}
    if p2 != nil {tail.Next = p2}
    return head.Next
}
```

242. 有效的字母异位词

```go
func isAnagram(s string, t string) bool {
    if len(s) != len(t) {
        return false
    }
    str1 := []byte(s)
    str2 := []byte(t)

    sort.Slice(str1, func(i, j int) bool {
        return str1[i] < str1[j]
    })
    sort.Slice(str2, func(i, j int) bool {
        return str2[i] < str2[j]
    })
    for i := 0; i < len(str1); i++ {
        if str1[i] != str2[i] {return false}
    }
    return true
}
```

1502. 判断能否形成等差数列

```go
func canMakeArithmeticProgression(arr []int) bool {
    sort.Ints(arr)
    diff := arr[1] - arr[0]
    for i := 2; i < len(arr); i++ {
        if arr[i] - arr[i-1] != diff {return false}
    }
    return true
}
```

252. 会议室

```go
func canAttendMeetings(intervals [][]int) bool{
    sort.Slice(intervals, func(i, j int) bool {
        return intervals[i][0] < intervals[j][0]
    })
    for i := 1; i < len(intervals); i++ {
        if intervals[i][0] < intervals[i-1][1] {return false}
    }
    return true
}
```

56. 合并区间

```go
func merge(intervals [][]int) [][]int {
    sort.Slice(intervals, func(i, j int) bool {
        return intervals[i][0] < intervals[j][0]
    })
    result := make([][]int, 0)
    curLeft := intervals[0][0]
    curRight := intervals[0][1]
    for i := 1; i < len(intervals); i++ {
        if intervals[i][0] <= curRight {
            if intervals[i][1] > curRight {
                curRight = intervals[i][1]
            }
        } else {
            result = append(result, []int{curLeft, curRight})
            curLeft = intervals[i][0]
            curRight = intervals[i][1]
        }
    }
    result = append(result, []int{curLeft, curRight})
    return result
}
```

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

```go
func exchange(nums []int) []int {
    i := 0
    j := len(nums)-1
    for i < j {
        if nums[i] % 2 == 1 {
            i++
            continue
        }
        if nums[j] % 2 == 0 {
            j--
            continue
        }
        nums[i], nums[j] = nums[j], nums[i]
        i++
        j--
    }
    return nums
}
```

75. 颜色分类

```go
func sortColors(nums []int) {
    p := 0
    q := len(nums)-1
    for p < q {
        if nums[p] != 2 {
            p++
            continue
        }
        if nums[q] == 2 {
            q--
            continue
        }
        nums[p], nums[q] = nums[q], nums[p]
        p++
        q--
    }
    i := 0
    j := p
    if nums[j] == 2 {j--}
    for i < j {
        if nums[i] == 0 {
            i++
            continue
        }
        if nums[j] == 1 {
            j--
            continue
        }
        nums[i], nums[j] = nums[j], nums[i]
        i++
        j--
    }
}
```

147. 对链表进行插入排序

```
func insertionSortList(head *ListNode) *ListNode {
    if head == nil {return nil}
    //存储已经排序好的节点
    newHead := &ListNode{Val:  math.MaxInt32, Next: nil}
    //遍历节点
    p := head
    for p != nil {
        tmp := p.Next
        //寻找 p 节点插入的位置，插入到哪个节点后面
        q := newHead //初始化值
        for q.Next != nil && q.Next.Val <= p.Val { //循环结束条件
            q = q.Next
        }
        //将 p 节点插入
        p.Next = q.Next
        q.Next = p
        p = tmp
    }
    return newHead.Next
}
```

148. 排序链表

```
//解法 1  递归解法
func mergeSortList(head *ListNode) *ListNode {
    if head == nil {return nil}
    if head.Next == nil {return head}
    midNode := findMidNode(head)
    nextNode := midNode.Next
    midNode.Next = nil
    leftHead := mergeSortList(head)
    rightHead := mergeSortList(nextNode)
    return mergeList(leftHead, rightHead)
}

func findMidNode(head *ListNode) *ListNode{
    slow := head
    fast := head
    for fast.Next != nil && fast.Next.Next != nil {
        fast = fast.Next.Next
        slow = slow.Next
    }
    return slow
}

func mergeList(headA, headB *ListNode) *ListNode{
    newHead := &ListNode{}
    tail := newHead
    pa := headA
    pb := headB
    for pa != nil && pb != nil {
        if pa.Val <= pb.Val {
            tail.Next = pa
            tail = tail.Next
            pa = pa.Next
        } else {
            tail.Next = pb
            tail = tail.Next
            pb = pb.Next
        }
    }
    if pa != nil {tail.Next = pa}
    if pb != nil {tail.Next = pb}
    return newHead.Next
}

//解法 2 非递归解法
func sortList(head *ListNode) *ListNode {
    n := len(head)
    step := 1
    for step < n {
        newHead := &ListNode{}
        tail := newHead
        p := head
        for p != nil {
            //[p, q]
            q := p
            count := 1
            for q != nil && count < step {
                q = q.Next
                count++
            }
            if q == nil || q.Next == nil { //这一轮合并结束了
                tail.Next = p
                break
            }
            // [q+1, r]
            r := q.Next
            count = 1
            for r != nil && count < step {
                r = r.Next
                count++
            }
            //保存下一个 step 的起点
            var tmp *ListNode = nil
            if r != nil {
                tmp = r.Next
            }
            //merge [p, q] [q+1, r]
            tail.Next, tail = merge(p, q, r)
            p = tmp
        }
        head = newHead.Next
        step *= 2
    }
    return head
}

func len(head *ListNode) int {
    if head == nil {return 0}
    n := 1
```

```go
    p := head
    for p != nil {
        n++
        p = p.Next
    }
    return n
}

func merge(p , q, r *ListNode) (*ListNode, *ListNode){
    newHead := &ListNode{}
    tail := newHead
    pa := p
    pb := q.Next
    q.Next = nil
    if r != nil {
        r.Next = nil
    }
    for pa != nil && pb != nil {
        if pa.Val <= pb.Val {
            tail.Next = pa
            tail = tail.Next
            pa = pa.Next
        } else {
            tail.Next = pb
            tail = tail.Next
            pb = pb.Next
        }
    }
    if pa != nil {
        tail.Next = pa
        tail = q
    }
    if pb != nil {
        tail.Next = pb
        tail = r
    }
    return newHead.Next, tail
}

//链表冒泡排序 (交换节点,有点费劲)
func bubbleSortList(head *ListNode) *ListNode {
    if head == nil {return nil}
    if head.Next == nil {return head}
    p := head
    n := 0
    for p != nil {
        p = p.Next
        n++
    }
    for i := 0; i < n; i++ {
        q := head
        var pre *ListNode = nil
        for j := 0; j < n-i-1 ; j++ {
            //交换 q 和 q.next: 先删除 q.next, 并记录为 tmp; 再把 tmp 插入到 pre 和 q 之间
            if q.Val > q.Next.Val {
                //交换结束后, q 已经位于 q.next; 因此不必另加一句 q = q.Next
                if pre == nil { //q 是头节点
                    pre = q.Next
                    tmp := q.Next
                    q.Next = q.Next.Next
                    tmp.Next = q
                    head = tmp
                } else {
                    tmpPre := pre
                    //q 和 q.next 交换，交换完后，q.next 会成为 q 的 pre，因此这里提前记录下。
                    pre = q.Next
                    tmp := q.Next
                    q.Next = q.Next.Next
                    tmp.Next = q
                    tmpPre.Next = tmp
                }
            } else {
                pre = q
                q = q.Next
            }
        }
    }
    return head
}

//链表冒泡排序 (交换值, 建议就用交换值吧)
func bubbleSortList(head *ListNode) *ListNode {
    if head == nil {return nil}
    if head.Next == nil {return head}
    p := head
    n := 0
    for p != nil {
        p = p.Next
        n++
    }
    for i := 0; i < n; i++ {
        q := head
        for j := 0; j < n-i-1 ; j++ {
            if q.Val > q.Next.Val {
                tmp := q.Val
                q.Val = q.Next.Val
                q.Next.Val = tmp
            }
            q = q.Next
        }
    }
    return head
}

//快排, 参考 zhengge 数组快排
func linkQuickSort(head *ListNode) *ListNode{
    tail := head
    for tail.Next != nil{
        tail = tail.Next
    }
    linkSubQuickSort(head, tail)
    return head
}

func linkSubQuickSort(p, r *ListNode){
    if p == r {return }
    preq ,q := linkPartition(p, r)
    if q != r {
         linkSubQuickSort(q.Next, r)
    }
```

```go
    if q != p {
         linkSubQuickSort(p, preq)
    }
    return
}

func linkPartition(p, r *ListNode) (*ListNode, *ListNode){
    var pre *ListNode = nil
    i := p
    j := p
    for j != r {
        if j.Val < r.Val {
            tmp := j.Val
            j.Val = i.Val
            i.Val = tmp
            pre = i
            i = i.Next
        }
        j = j.Next
    }
    tmp := i.Val
    i.Val = r.Val
    r.Val = tmp
    return pre ,i
}
```

215. 数组中的第 K 个最大元素

```go
func findKthLargest(nums []int, k int) int {
    if len(nums) < k {return -1}
    return quickSort(nums, 0, len(nums)-1, k)
}

func quickSort(nums []int, p, r, k int) int{
    if p > r {return -1}
    q := partition(nums, p, r)
    if q-p+1 == k {
        return nums[q]
    } else if q-p+1 < k {
        return quickSort(nums, q+1, r, k-(q-p+1))
    } else {
        return quickSort(nums, p, q-1, k)
    }
}

func partition(nums []int, p, r int) int{
    i := p-1
    j := p
    for j < r {
        if nums[j] > nums[r] {
            nums[j], nums[i+1] = nums[i+1], nums[j]
            i++
        }
        j++
    }
    nums[i+1], nums[r] = nums[r], nums[i+1]
    return i+1
}

//解法 2 双指针, 更简单, 建议基础不太好的记这个解法
```

```go
func findKthLargest(nums []int, k int) int {
    if k <= 0 || len(nums) < k {return -1}
    return quickSortForKth(nums, 0, len(nums)-1, k)
}

func quickSortForKth(nums []int, p, r, k int) int{
    if p > r {return -1}
    q := partition(nums, p, r)
    if q-p+1 == k {return nums[q]}
    if q-p+1 < k {
        return quickSortForKth(nums, q+1, r, k-(q-p+1) )
    }else {
        return quickSortForKth(nums, p, q-1, k)
    }
}

//从大到小
func partition(a []int, p, q int) int{
    r := q
    i := p
    j := q-1
    for i <= j && i <= q && j >= p{
        if a[i] >= a[r] {i++;continue}
        if a[j] < a[r] {j--;continue}
        if a[i] < a[r] && a[j] >= a[r] {
            a[i], a[j] = a[j], a[i]
            i++
            j--
        }
    }
    a[i], a[r] = a[r], a[i]
    return i
}
```

面试题 17.14. 最小 K 个数

```go
var result []int
var count = 0
func smallestK(arr []int, k int) []int {
    if k == 0 && len(arr) < k {return []int{}}
    result = make([]int, k)
   //重新初始化，不然 leetcode 里上一个测试用例的结果当作下一个测试用例的初始值,可能是 leetcode 的
bug
    count = 0
    quickSort(arr, 0, len(arr)-1, k)
    return result
}

func quickSort(nums []int, p, r, k int) {
    if p > r {return}
    q := partition(nums, p, r)
    if q-p+1 == k {
        for i := p; i <= q; i++ {
            result[count] = nums[i]
            count++
        }
    } else if q-p+1 < k {
        for i := p; i <= q; i++ {
            result[count] = nums[i]
            count++
        }
        quickSort(nums, q+1, r, k-(q-p+1))
    } else {
        quickSort(nums, p, q-1, k)
    }
}

func partition(nums []int, p, r int) int{
    i := p-1
    j := p
    for j < r {
        if nums[j] < nums[r] {
            nums[j], nums[i+1] = nums[i+1], nums[j]
            i++
        }
        j++
    }
    nums[i+1], nums[r] = nums[r], nums[i+1]
    return i+1
}
```

剑指 Offer 51. 数组中的逆序对

```
var reverseCount = 0
func reversePairs(nums []int) int {
    //每次都重新初始化，不然 leetcode 里上一个测试用例的结果会覆盖下一个测试用例的 reverseCount
    reverseCount = 0
    mergeSort(nums, 0, len(nums)-1)
    return reverseCount
}

func mergeSort(nums []int, p, r int) {
    if p >= r {return}
    q := (p+r)/2
    mergeSort(nums, p, q)
    mergeSort(nums, q+1, r)
    merge(nums, p, q, r)
}

func merge(nums []int, p, q, r int) int{
    tmp := make([]int, r-p+1)
    i := p
    j := q+1
    k := 0
    for i <= q && j <= r {
        if nums[j] < nums[i] {
            reverseCount += (q-i+1)
            tmp[k] = nums[j]
            k++
            j++
        } else {
            tmp[k] = nums[i]
            k++
            i++
        }
    }
    for j <= r {
        tmp[k] = nums[j]
        k++
        j++
    }
    for i <= q {
        tmp[k] = nums[i]
        k++
        i++
    }
    for i := 0; i < r-p+1; i++ {
        nums[i+p] = tmp[i]
    }
    return reverseCount
}
```