**函数列表：**

table.insert(table,[ pos,] value)   
table.remove(table[, pos])   
table.concat(table[, sep[, i[, j]]])   
table.sort(table[, comp])

**1. insert 和 remove 只能用于数组元素的插入和移出， 进行插入和移出时，会将后面的元素对齐起来。**

    所以在 for 循环中进行 insert 和 remove 的时候要注意插入和移除时是否漏掉了某些项：

local t = {1,2,3,3,5,3,6}   
        for i,v in ipairs(t) do   
            if v == 3 then   
                table.remove(t,i)   
            end   
        end   
        -- 错误，第四个 3 没有被移除，ipairs 内部会维护一个变量记录遍历的位置，remove 掉第三个数字 3 之后，ipairs 下一个返回的值是 5 而不是 3   
          
        local t = {1,2,3,3,5,3,6}   
        for i=1, #t do   
            if t[i] == 3 then   
                table.remove(t,i)   
                i = i-1   
            end   
        end   
        -- 错误，i=i-1 这段代码没有用，i 的值始终是从 1 到 #t，for 循环里修改 i 的值不起作用   
          
        local t = {1,2,3,3,5,3,6}   
        for i=#t, 1, -1 do   
            if t[i] == 3 then   
                table.remove(t,i)   
            end   
        end   
        -- 正确，从后往前遍历   
          
        local t = {1,2,3,3,5,3,6}   
        local i = 1   
        while t[i] do   
            if t[i] == 3 then   
                table.remove(t,i)   
            else   
                i = i+1   
            end   
        end   
        -- 正确，自己控制 i 的值是否增加 

1. **concat 可以将 table 的数组部分拼接成一个字符串，中间用 seq 分隔。**  
       lua 中字符串的存储方式与 C 不一样，lua 中的每个字符串都是单独的一个拷贝，拼接两个字符串会产生一个新的拷贝，如果拼接操作特别多，就会影响性能：

local beginTime = os.clock()   
        local str = ""   
        for i=1, 30000 do   
            str = str .. i   
        end   
        local endTime = os.clock()   
        print(endTime - beginTime)   
        -- 消耗 0.613 秒，产生了 30000 个字符串拷贝，但只有最后一个是有用的

        local beginTime = os.clock()   
        local t = {}   
        for i=1, 30000 do   
            t[i] = i   
        end   
        local str = table.concat(t, "")   
        local endTime = os.clock()   
        print(endTime - beginTime)   
        -- 消耗 0.024 秒，利用 concat，一次性把字符串拼接出来，只产生了一个字符串拷贝

1. **sort 可以将 table 数组部分的元素进行排序，需要提供 comp 函数，comp(a, b) 如果 a 应该排到 b 前面，则 comp 要返回 true 。**  
       注意，对于 a==b 的情况，一定要返回 false :

local function comp(a,b)   
            return a <= b   
        end   
        table.sort(t,comp)   
        -- 错误，可能出现异常：attempt to compare number with nil   
          
        local function comp(a,b)   
            if a == nil or b == nil then   
                return false   
            end   
            return a <= b   
        end   
        table.sort(t,comp)   
        -- 错误，可能出现异常：invalid order function for sorting   
        -- 也可能不报这个异常，但结果是错误的；   
    之所以 a==b 返回true 会引发这些问题，是因为 table.sort 在实现快速排序时没有做边界检测：   
        for (;;) {   
          while (lua\_rawgeti(L, 1, ++i), sort\_comp(L, -1, -2)) {  // 未检测边界, i 会一直增加   
            if (i>=u) luaL\_error(L, "invalid order function for sorting");   
            lua\_pop(L, 1);   
          }   
          while (lua\_rawgeti(L, 1, --j), sort\_comp(L, -3, -1)) {  // 未检测边界, j 会一直减少   
            if (j<=l) luaL\_error(L, "invalid order function for sorting");   
            lua\_pop(L, 1);   
          }   
          if (j<i) {   
            lua\_pop(L, 3);   
            break;   
          }   
          set2(L, i, j);   
        }

 看以上代码，如果 a==b 时返回 true 且边界上的几个值是相等的话， sort\_comp 就无法阻止 i 继续增长，直到超出边界引发异常 attempt to compare number with nil；即使我们对 a 和 b 进行非空判断，也会因为 i 超过边界而引发异常 invalid order function for sorting   
    快速排序是什么，lua 如何实现快速排序，可以参考 lua 源码中的描述。

原文链接：<http://www.jb51.net/article/64711.htm>