南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

南京理工大学计算机科学与工程学院

程序设计基础（II）课程实践报告

选　　题：整数排序——快速排序

班　　级：9191069501

学生姓名：王子辉

学　　号：9191160D0236

指导教师：刘冬梅

一、 设计思想

快速排序是一种时间复杂度平均在 O($nlogn$)、最坏情况在 O(n^2)规模的高效的排序算法，其基本设计思想在于：对于输入的待排序数组 A[1…n]，每次以 A[low…high]的首元素 A[low]为基准，通过一趟排序将 A[low…high]分割成独立的两部分，其中一部分的所有数据比另一部分的所有数据要小，从而将对 A[1…n]排序转变为对 A[low…mid-1]和 A[mid+1…high]这两个数组分别进行排序的子问题。以此类推，直至子数组分无可分。若快速排序的任务是将输入数组 A[1…n]按由大到小的规则排序，则当且仅当输入数组本身就为顺序时，时间复杂度为最坏情况 O(n^2)。

二、 代码分析

1、伪代码

```
#  划分
# Cross References:
# a0 = &array
# a1 = low
# a2 = high
# t0 = left
# t3 = *(array + a1)
# t4 = *(array + a2)
Split (a0, a1, a2):
    t0 = 1;    #  标志左划分还是右划分
    while (a1 != a2)
    {
        if (t0 == 1)     #  左划分
        {
            while (a1 != a2 && Mem[a0 + a1] <= Mem[a0 + a1])
            {
                a2--;
            }
            t1 = a1 << 2;
            t2 = a2 << 2;
            t3 = Mem[a0 + t1];
            t4 = Mem[a0 + t2];
            Mem[a0 + t1] = t4;
            Mem[a0 + t2] = t3;
            t0 = 0;
```

```
            }
            else        # 右划分
            {
                while (a1 != a2 && Mem[a0 + a1] <= Mem[a0 + a1])
                {
                    a1++;
                }
                t3 = Mem[a0 + a1];
                t4 = Mem[a0 + a2];
                Mem[a0 + a1] = t4;
                Mem[a0 + a2] = t3;
                t0 = 1;
            }
        }
        return a1;

# 快速排序
# Cross References:
# a0 = &array
# a1 = low
# a2 = high
# v0 = mid    that is    Split (a0, a1, a2)
# t0 = mid - 1
# t1 = mid + 1
QuickSort (a0, a1, a2):
        if (a1 >= a2)          # 递归边界
        {
            return;
        }
        v0 = Split (a0, a1, a2);
        t0 = v0 - 1;
        t1 = v0 + 1;
        QuickSort (a0, a1, t0);
        QuickSort (a0, t1, a2);

# 输出数组
# Cross References:
# a0 = &array
# a1 = low
# a2 = high
PrintArray (a0, a1, a2):
        while (a1 <= a2)
        {
            cout << Mem[a0 + a1];
```

```
            a1 ++;
        }
        return;

    main:
        t0 = &array;
        t1 = 5;
        while (t1 != 0)
        {
            cin >> v0;
            Mem[t0] = v0;
            t0 += 4;
            t1 --;
        }
        a0 = &array;
        a1 = 0;
        a2 = 4;
        QuickSort (a0, a1, a2);
        PrintArray (a0, a1, a2);
        return 0;
```

2、MIPS 源码

```
            .data
result:     .asciiz "Sorted array:"
            .align 2
array:      .word 40
            .globl main
            .text
main:
            # 输入数组元素
            la    $t0, array
            li    $t1, 5
input:      beqz      $t1, end_input
            li    $v0, 5
            syscall
            sw    $v0, ($t0)
            addi  $t0, $t0, 4
            addi  $t1, $t1, -1
            b     input

            # 输入参数
end_input:
            la    $a0, array            # a0：数组首地址
```

```
    li      $a1, 0              # a1：low
    li      $a2, 4              # a2：high

    # 调用 QuickSort
    addiu     $sp, $sp, -24
    sw    $a0, 0($sp)
    sw    $a1, 4($sp)
    sw    $a2, 8($sp)
    sw    $ra, 20($sp)
    jal    QuickSort
    lw    $ra, 20($sp)
    lw    $a0, 0($sp)
    lw    $a1, 4($sp)
    lw    $a2, 8($sp)
    addiu     $sp, $sp, 24

    # 调用 PrintArray
    addiu     $sp, $sp, -16
    sw    $a0, 0($sp)
    sw    $a1, 4($sp)
    sw    $a2, 8($sp)
    sw    $ra, 12($sp)
    jal    PrintArray
    lw    $ra, 12($sp)
    addiu     $sp, $sp, 16

    # 结束
    li      $v0, 10
    syscall


QuickSort:
    # 加载参数
    lw    $a0, 0($sp)
    lw    $a1, 4($sp)
    lw    $a2, 8($sp)

    # 递归边界
    bge $a1, $a2, qs_ret

    # 调用 Split 进行划分
    addiu     $sp, $sp, -20
    sw    $a0, 0($sp)
    sw    $a1, 4($sp)
```

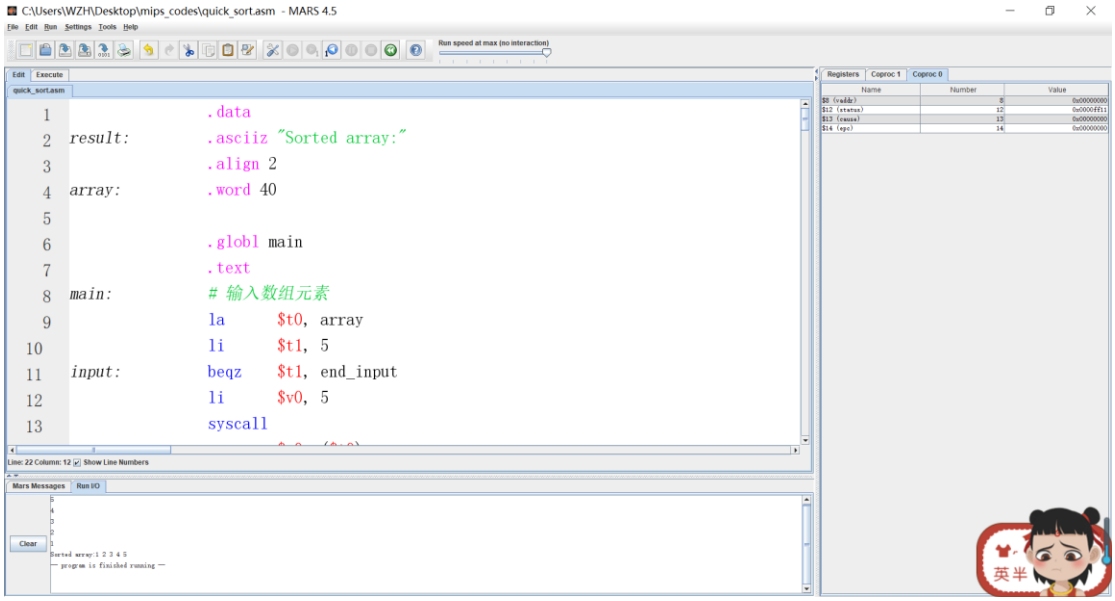
```
        sw   $a2, 8($sp)
        sw   $ra, 16($sp)
        jal   Split
        lw   $ra, 16($sp)
        lw   $v0, 12($sp)
        addiu     $sp, $sp, 20

        #  堆栈中存储 mid - 1 和 mid + 1
        addi $t0, $v0, -1
        addi $t1, $v0, 1
        sw   $t0, 12($sp)          # mid - 1
        sw   $t1, 16($sp)          # mid + 1

        #  调用 QuickSort
        lw   $a0, 0($sp)           # a0：数组首地址
        lw   $a1, 4($sp)           # a1：low
        lw   $a2, 12($sp)          # a2：mid - 1
        addiu     $sp, $sp, -24
        sw   $a0, 0($sp)
        sw   $a1, 4($sp)
        sw   $a2, 8($sp)
        sw   $ra, 20($sp)
        jal   QuickSort
        lw   $ra, 20($sp)
        addiu     $sp, $sp, 24

        #  调用 QuickSort
        lw   $a0, 0($sp)           # a0：数组首地址
        lw   $a1, 16($sp)          # a1：mid + 1
        lw   $a2, 8($sp)           # a2：high
        addiu     $sp, $sp, -24
        sw   $a0, 0($sp)
        sw   $a1, 4($sp)
        sw   $a2, 8($sp)
        sw   $ra, 20($sp)
        jal   QuickSort
        lw   $ra, 20($sp)
        addiu     $sp, $sp, 24

qs_ret:   jr     $ra


Split:
        lw   $a0, 0($sp)
```

```
        lw   $a1, 4($sp)
        lw   $a2, 8($sp)
        li   $t0, 1                 #  标志左划分还是右划分
loop:   beq $a1, $a2, ret
        beqz    $t0, right
left:   beq $a1, $a2, out_left      # while 语句条件判断
        sll  $t1, $a1, 2
        sll  $t2, $a2, 2
        add $t1, $t1, $a0           # t1 = &array + i
        add $t2, $t2, $a0           # t2 = &array + j
        lw   $t3, ($t1)             # t3 = array[i]   *(t1)
        lw   $t4, ($t2)             # t4 = array[j]   *(t2)
        bgt  $t3, $t4, out_left     # while 语句条件判断
        addi$a2, $a2, -1            # j--
        b    left
out_left: sll  $t2, $a2, 2          #  更新 j
        add $t2, $t2, $a0
        lw   $t4, ($t2)             # t4 = array[j]
        sw   $t3, ($t2)             # &array + j = array[i]
        sw   $t4, ($t1)             # &array + i = array[j]
        li   $t0, 0
        b    loop
right:  beq $a1, $a2, out_right     # while 语句条件判断
        sll  $t1, $a1, 2
        sll  $t2, $a2, 2
        add $t1, $t1, $a0           # t1 = &array + i
        add $t2, $t2, $a0           # t2 = &array + j
        lw   $t3, ($t1)             # t3 = array[i]   *(t1)
        lw   $t4, ($t2)             # t4 = array[j]   *(t2)
        bgt  $t3, $t4, out_right    # while 语句条件判断
        addi$a1, $a1, 1             # i++
        b    right
out_right:
        sll  $t1, $a1, 2            #  更新 i
        add $t1, $t1, $a0
        lw   $t3, ($t1)             # t3 = array[i]
        sw   $t3, ($t2)             # &array + j = array[i]
        sw   $t4, ($t1)             # &array + i = array[j]
        li   $t0, 1
        b    loop
ret:    move    $v0, $a1
        sw   $v0, 12($sp)
        jr   $ra
```

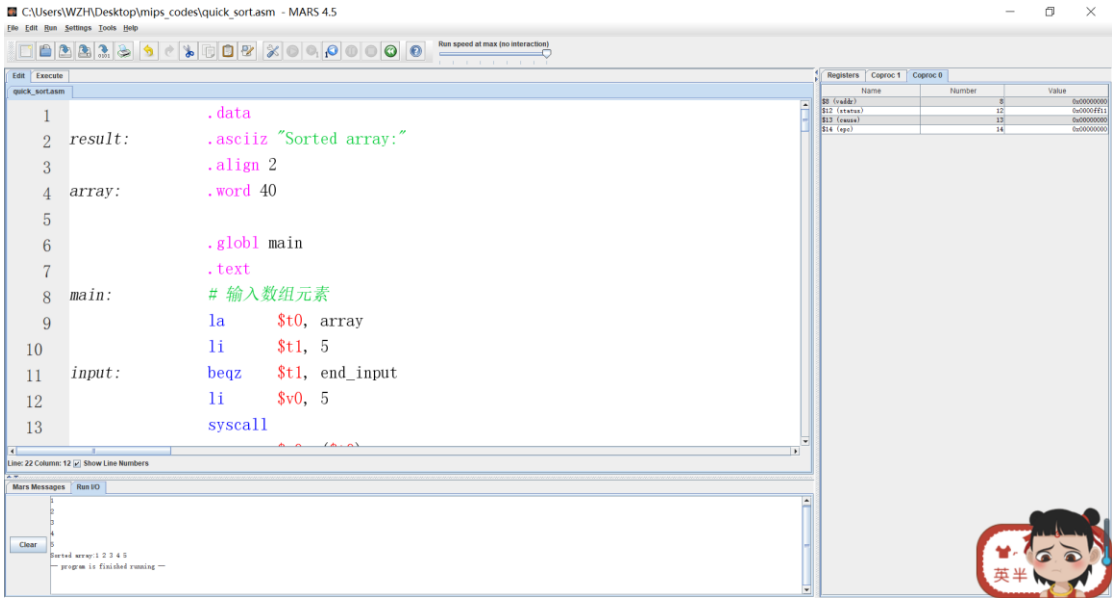
```
PrintArray:
        lw    $a0, 0($sp)
        lw    $a1, 4($sp)
        lw    $a2, 8($sp)
        addiu     $sp, $sp, -4
        sw    $a0, 0($sp)
        # 打印字符串
        la    $a0, result
        li    $v0, 4
        syscall
        lw    $a0, 0($sp)
        addiu     $sp, $sp, 4
p_loop: bgt   $a1, $a2, p_ret
        lw    $t0, ($a0)
        addiu     $sp, $sp, -4
        sw    $a0, 0($sp)
        move      $a0, $t0
        li    $v0, 1
        syscall
        # 输出空格
        li    $a0, 32
        li    $v0, 11
        syscall
        lw    $a0, 0($sp)
        addiu     $sp, $sp, 4
        addi$a0, $a0, 4
        addi$a1, $a1, 1
        b     p_loop
p_ret:  jr    $ra
```

三、运行结果截图

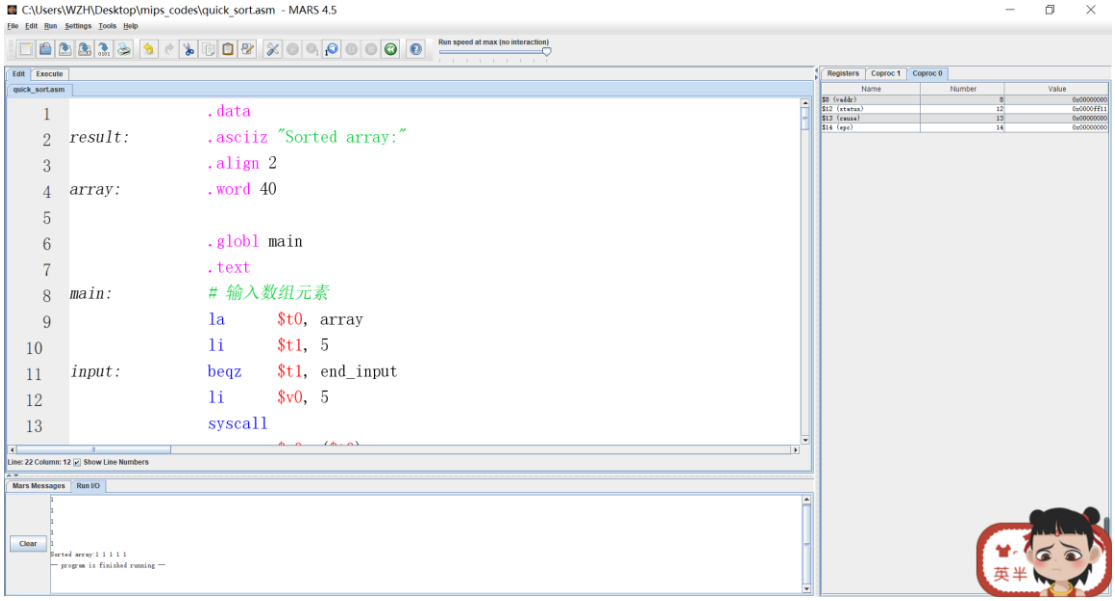
如图输入为{5, 4, 3, 2, 1}， 输出为{1, 2, 3, 4, 5}



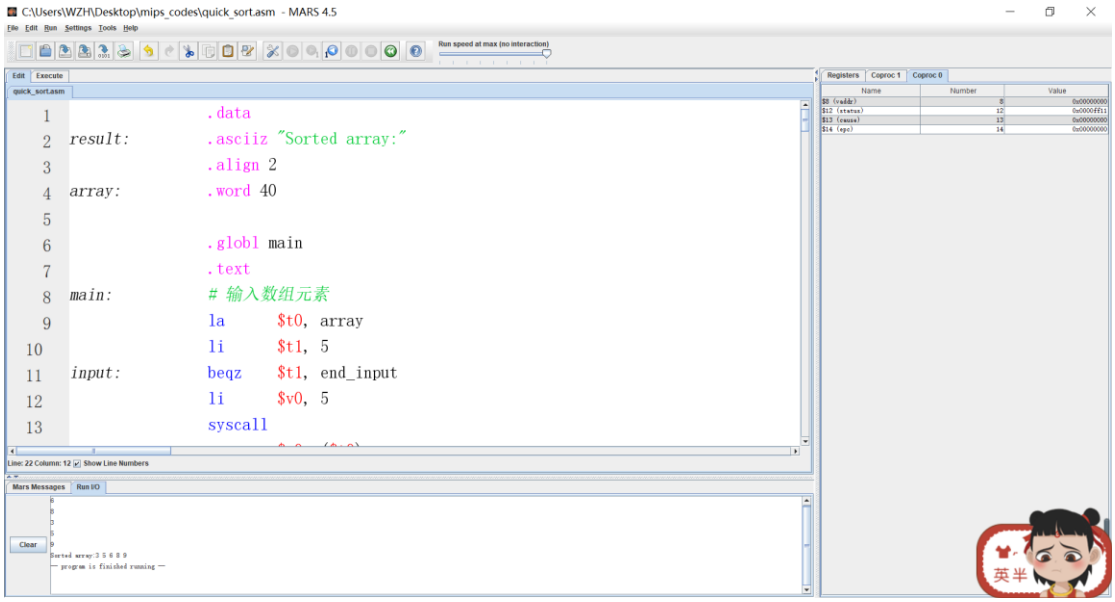
如图输入为{1, 2, 3, 4, 5}， 输出为{1, 2, 3, 4, 5}



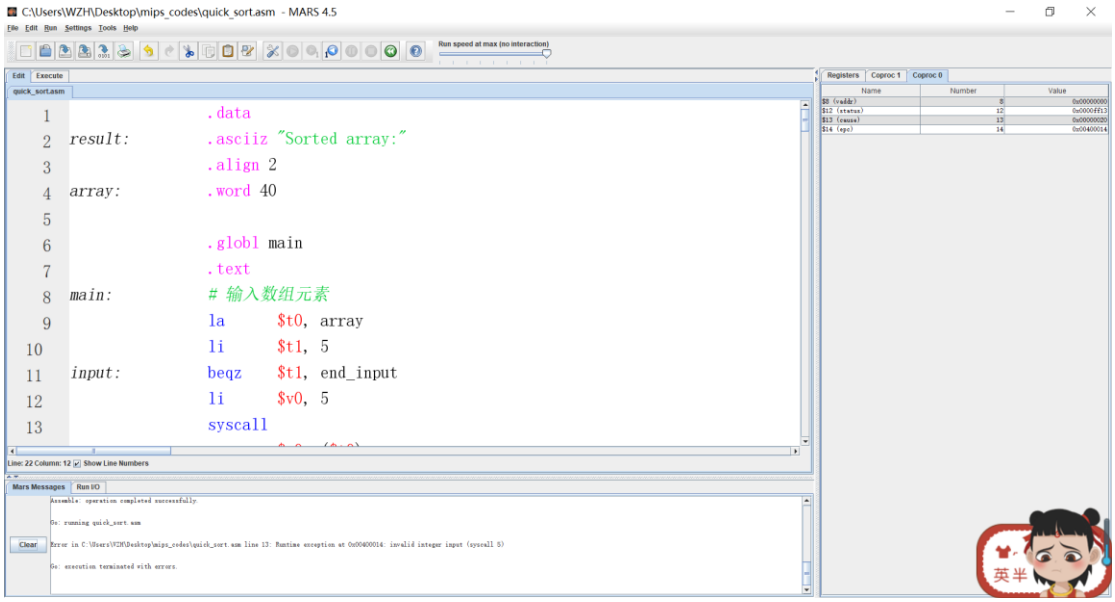
如图输入为{1, 1, 1, 1, 1}，输出为{1, 1, 1, 1, 1}



如图输入为{6, 8, 3, 5, 9}，输出为{3, 5, 6, 8, 9}



如图输入字符 a，模拟器抛出异常



四、心得体会

由于转专业的缘故，再加上去年的课程冲突，我只好在大三下学期才选上大一的课。以前总听原计科院的同学说汇编很困难，想拿高分不容易，这六周课上完，作为已经学完数据结构、算法设计和组成原理的我竟深以为然。

前几次课还可以，上的是网课，靠在床上半迷糊半清醒地听课，虽然效率不高，但毕竟还是学过组成原理的人了，什么程序计数器 PC、指令类型、指令执行过程、指令流水线以及分支延迟这些还是熟悉的。就记得上学期上组原时，老师生怕大家忘记了曾经学过的汇编指令的内容，于是把组原里关于 MIPS 和 RISC-V 的内容花了两节课的时间缕了一遍，因此这学期上 MIPS 的时候还留有印象。到了后面几周，真正要上手写代码了，尽管我在刷题以及各种课设中已经写了一两万行的代码，在 Mars 模拟器面前，我还是一脸懵。根本找不回前阵子做编译器时，一看到代码就文思泉涌的感觉。于是决定先吃透上课讲的代码，非常典型又具有代表性，然后先照葫芦画瓢般吃透程序模板再上手写代码。这几次作业下来就有这样的感觉，把上课精讲过的代码仔细研读一遍，写课后作业时都少 de 很多 bug。

在我认识的人当中，汇编大作业最后交的很多都是年历，于是这次决定挑战一下自己，实现一个快速排序算法。说实话，debug 的过程很痛苦，但是最后出正确结果也是发自内心的喜悦，和做完编译器大作业的成就感相当。也是感慨，如果我一开始就学过汇编，那我在上数据结构、算法设计和组成原理时，一定会有新的感悟和体会。对于数据结构这方面，我记得的就是在讲到 DFS 那一章时，老师说过，其实递归本质上就是系统栈的弹入和弹出，因此 DFS 虽然提供了递归和非递归两种实现，但两者本质上是没有区别的。本人有幸在听过这种说法后，真真切切地体验了一把系统栈的调用；透过 QuickSort 函数中的 return，我看到了其递归调用的本质，于是对递归算法又有了新的认识和理解。

以前总觉得，既然是面向工作，我们学这些古早的、底层的语言有何意义？从背弃指针面向对象的 Java，到“人生苦短，我用 Python”，再到“世界上最好

的语言是 PHP”，高级语言逐渐成为了代码工程中的主角。像 MIPS 这种底层语言，就连个 IDE 都没有，还得靠 Java 环境下的模拟器运行，如果不是以前摸索很久才给电脑下了个 JDK，可以运行 Mars，现在只能靠文本文档在 QtSpim 里面摸索。但这几次作业以及课程实践下来，才真的体会到了 MIPS 作为底层语言的美。它可能没有那么简洁、美观、实用，甚至都不能被称作是一种“语言”，但它用一次次的寄存器操作告诉你，再好看的代码，也得是由 MIPS、RISC-V 等提供的一条条指令构成的；学它的目的并不是说以后可以用精简指令集做项目，那不可能，关键是要让我们理解系统的真谛，了解电脑是怎么理解我们写进 IDE 里的一段程序，从而做到真正的“人机交互”。

最后还是很感谢这门课，以及我的任课教师刘老师。虽然我目前在智能专业，研究生做的应该也是模式识别的相关方向，以后可能与软硬件系统这方面不会有太多的交集，但是它确实开阔了我的眼界，让我对以往学过的算法又有了新的认识。知其然而又知其所以然，这才是一个合格的程序员该有的素养。