Computer Architecture HW2 Report

Bubble sort & Quick sort By SPIM

EE3 B03901156 Yu Xuan Huang

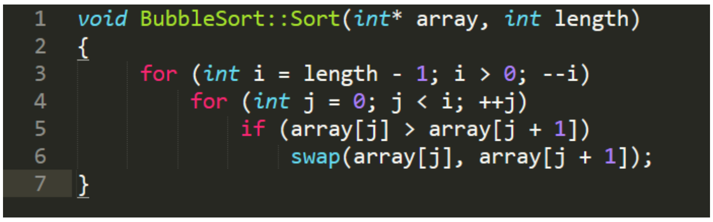
A. Bubble Sort

Design:

First, I divide the whole program into three parts completed by the respective function, including

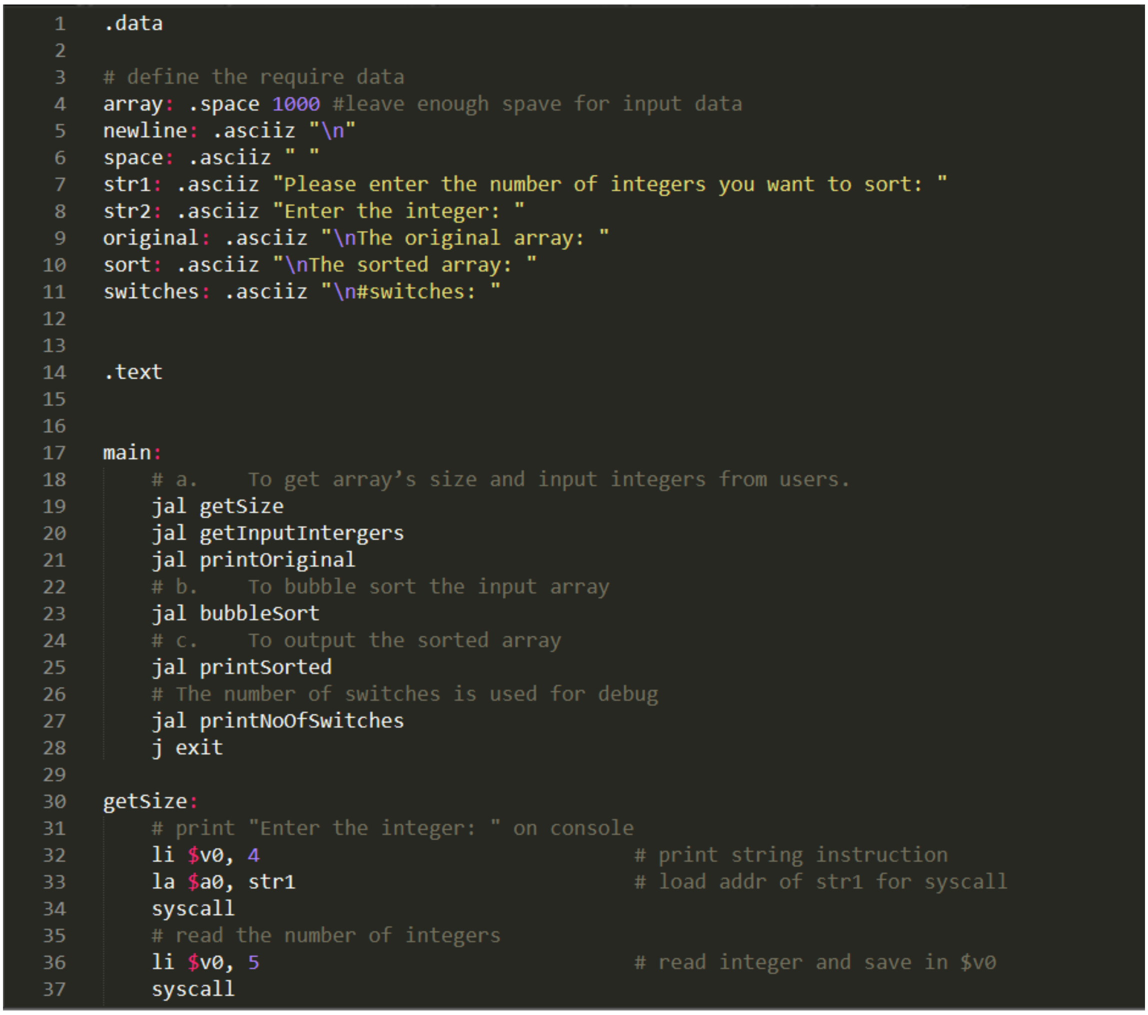
a. To get array's size and input integers from users.
b. To bubble sort the input array.
c. To output the sorted array.

The core of the design is according to the c++ pseudo code on the below:

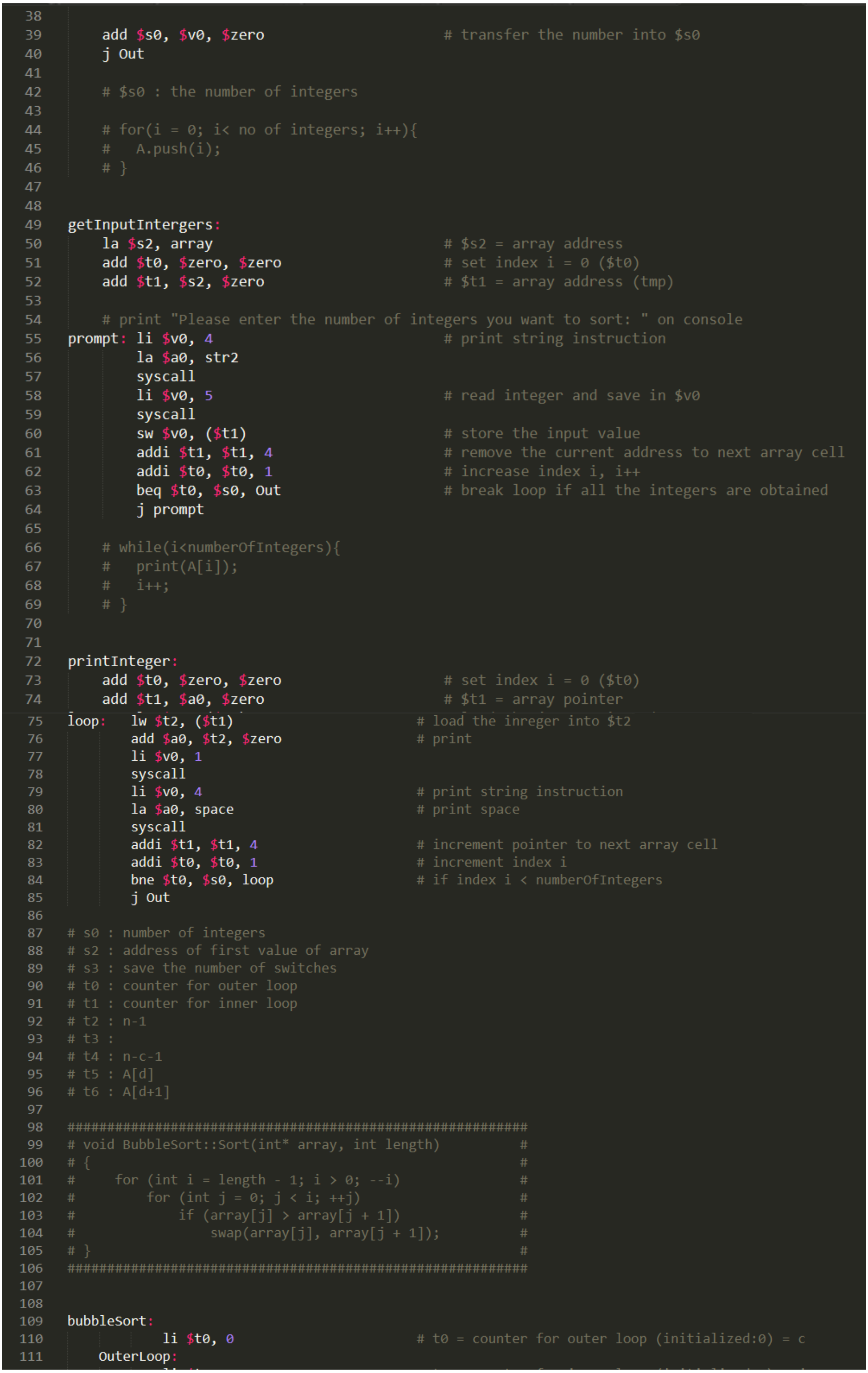


```
void BubbleSort::Sort(int* array, int length)
{
    for (int i = length - 1; i > 0; --i)
        for (int j = 0; j < i; ++j)
            if (array[j] > array[j + 1])
                swap(array[j], array[j + 1]);
}
```

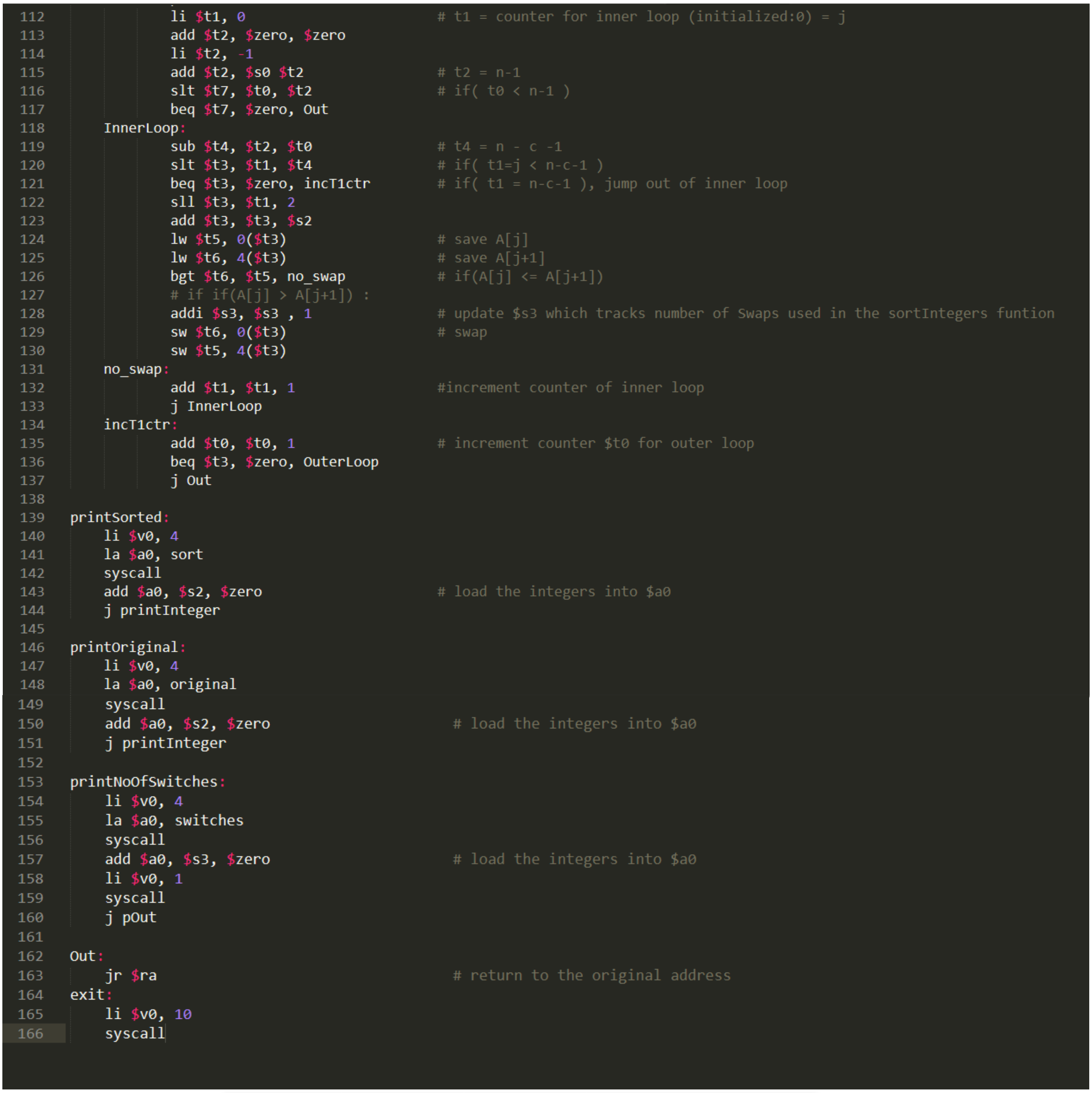
And all the required explanations are illlustrated as comments in the source code on the below:



```
.data

# define the require data
array: .space 1000 #leave enough spave for input data
newline: .asciiz "\n"
space: .asciiz " "
str1: .asciiz "Please enter the number of integers you want to sort: "
str2: .asciiz "Enter the integer: "
original: .asciiz "\nThe original array: "
sort: .asciiz "\nThe sorted array: "
switches: .asciiz "\n#switches: "


.text


main:
    # a.    To get array's size and input integers from users.
    jal getSize
    jal getInputIntergers
    jal printOriginal
    # b.    To bubble sort the input array
    jal bubbleSort
    # c.    To output the sorted array
    jal printSorted
    # The number of switches is used for debug
    jal printNoOfSwitches
    j exit

getSize:
    # print "Enter the integer: " on console
    li $v0, 4                               # print string instruction
    la $a0, str1                            # load addr of str1 for syscall
    syscall
    # read the number of integers
    li $v0, 5                               # read integer and save in $v0
    syscall
```

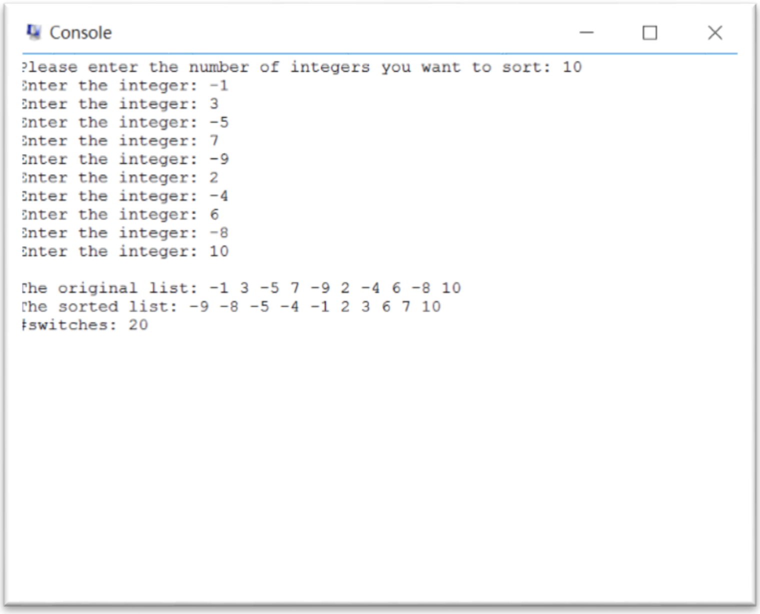


```

    add $s0, $v0, $zero                     # transfer the number into $s0
    j Out

    # $s0 : the number of integers

    # for(i = 0; i< no of integers; i++){
    #   A.push(i);
    # }


getInputIntergers:
    la $s2, array                           # $s2 = array address
    add $t0, $zero, $zero                   # set index i = 0 ($t0)
    add $t1, $s2, $zero                     # $t1 = array address (tmp)

    # print "Please enter the number of integers you want to sort: " on console
prompt: li $v0, 4                           # print string instruction
        la $a0, str2
        syscall
        li $v0, 5                           # read integer and save in $v0
        syscall
        sw $v0, ($t1)                       # store the input value
        addi $t1, $t1, 4                    # remove the current address to next array cell
        addi $t0, $t0, 1                    # increase index i, i++
        beq $t0, $s0, Out                   # break loop if all the integers are obtained
        j prompt

    # while(i<numberOfIntegers){
    #   print(A[i]);
    #   i++;
    # }


printInteger:
    add $t0, $zero, $zero                   # set index i = 0 ($t0)
    add $t1, $a0, $zero                     # $t1 = array pointer
loop:   lw $t2, ($t1)                       # load the inreger into $t2
        add $a0, $t2, $zero                 # print
        li $v0, 1
        syscall
        li $v0, 4                           # print string instruction
        la $a0, space                       # print space
        syscall
        addi $t1, $t1, 4                    # increment pointer to next array cell
        addi $t0, $t0, 1                    # increment index i
        bne $t0, $s0, loop                  # if index i < numberOfIntegers
        j Out

# s0 : number of integers
# s2 : address of first value of array
# s3 : save the number of switches
# t0 : counter for outer loop
# t1 : counter for inner loop
# t2 : n-1
# t3 :
# t4 : n-c-1
# t5 : A[d]
# t6 : A[d+1]

###########################################################
# void BubbleSort::Sort(int* array, int length)           #
# {                                                       #
#     for (int i = length - 1; i > 0; --i)                #
#         for (int j = 0; j < i; ++j)                     #
#             if (array[j] > array[j + 1])                #
#                 swap(array[j], array[j + 1]);           #
# }                                                       #
###########################################################


bubbleSort:
            li $t0, 0                       # t0 = counter for outer loop (initialized:0) = c
    OuterLoop:
```



```
            li $t1, 0                               # t1 = counter for inner loop (initialized:0) = j
            add $t2, $zero, $zero
            li $t2, -1
            add $t2, $s0 $t2                        # t2 = n-1
            slt $t7, $t0, $t2                       # if( t0 < n-1 )
            beq $t7, $zero, Out
    InnerLoop:
            sub $t4, $t2, $t0                       # t4 = n - c -1
            slt $t3, $t1, $t4                       # if( t1=j < n-c-1 )
            beq $t3, $zero, incT1ctr                # if( t1 = n-c-1 ), jump out of inner loop
            sll $t3, $t1, 2
            add $t3, $t3, $s2
            lw $t5, 0($t3)                          # save A[j]
            lw $t6, 4($t3)                          # save A[j+1]
            bgt $t6, $t5, no_swap                   # if(A[j] <= A[j+1])
            # if if(A[j] > A[j+1]) :
            addi $s3, $s3 , 1                       # update $s3 which tracks number of Swaps used in the sortIntegers funtion
            sw $t6, 0($t3)                          # swap
            sw $t5, 4($t3)
    no_swap:
            add $t1, $t1, 1                         #increment counter of inner loop
            j InnerLoop
    incT1ctr:
            add $t0, $t0, 1                         # increment counter $t0 for outer loop
            beq $t3, $zero, OuterLoop
            j Out

printSorted:
    li $v0, 4
    la $a0, sort
    syscall
    add $a0, $s2, $zero                             # load the integers into $a0
    j printInteger

printOriginal:
    li $v0, 4
    la $a0, original
    syscall
    add $a0, $s2, $zero                               # load the integers into $a0
    j printInteger

printNoOfSwitches:
    li $v0, 4
    la $a0, switches
    syscall
    add $a0, $s3, $zero                               # load the integers into $a0
    li $v0, 1
    syscall
    j pOut

Out:
    jr $ra                                            # return to the original address
exit:
    li $v0, 10
    syscall
```

Result:



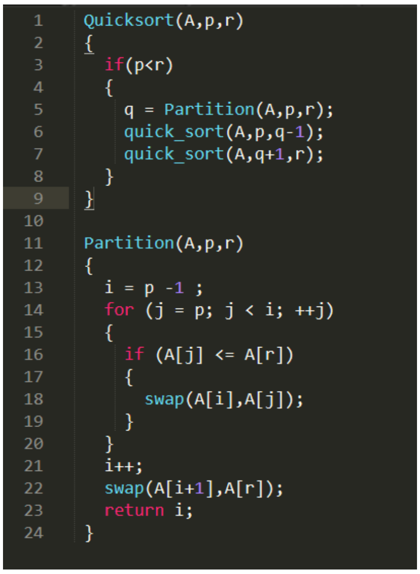
```
?lease enter the number of integers you want to sort: 10
:nter the integer: -1
:nter the integer: 3
:nter the integer: -5
:nter the integer: 7
:nter the integer: -9
:nter the integer: 2
:nter the integer: -4
:nter the integer: 6
:nter the integer: -8
:nter the integer: 10

The original list: -1 3 -5 7 -9 2 -4 6 -8 10
The sorted list: -9 -8 -5 -4 -1 2 3 6 7 10
#switches: 20
```

B. Quick Sort

Design:

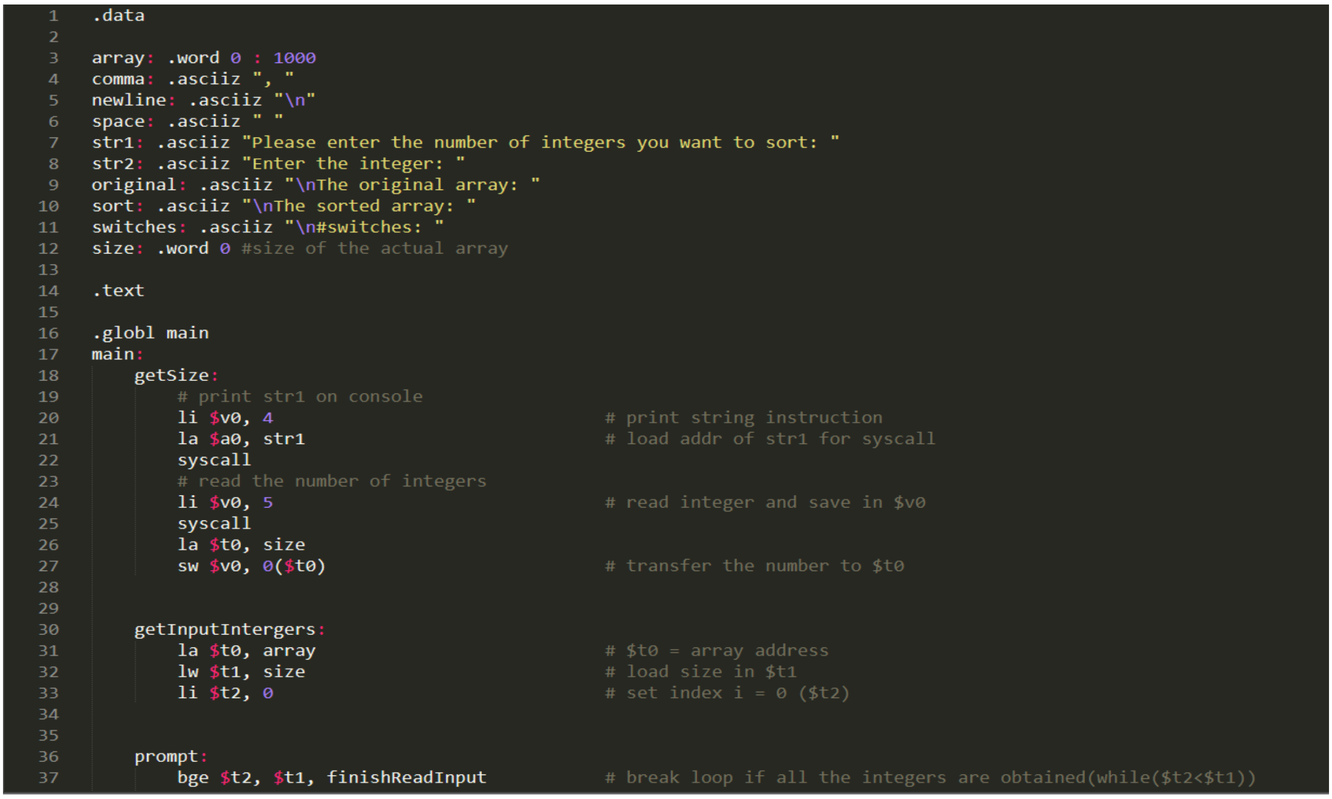
First, I divide the whole program into three parts completed by the respective function, including

a. To get array’s size and input integers from users.
b. To quick sort the input array.
c. To output the sorted array.

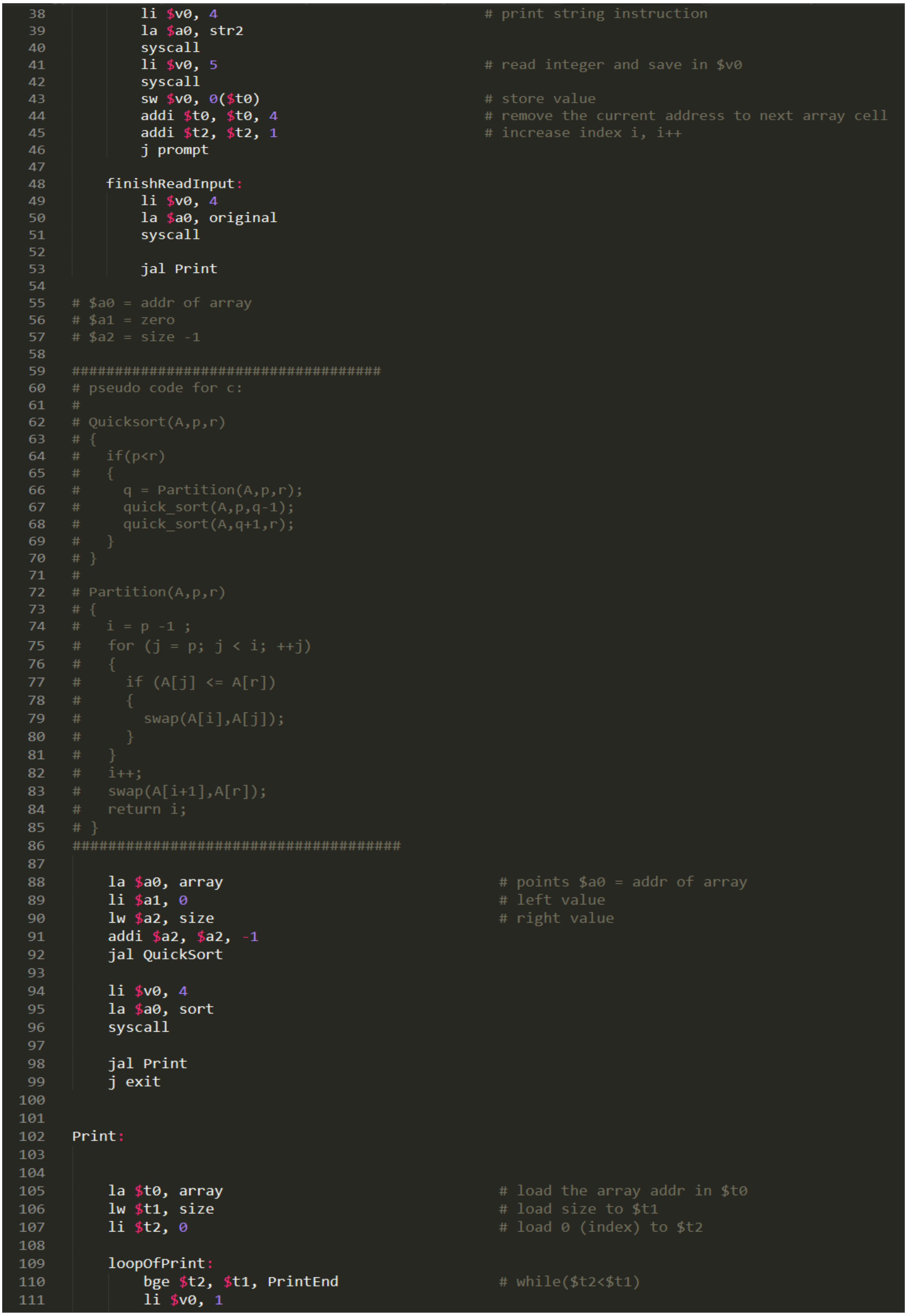


```
Quicksort(A,p,r)
{
  if(p<r)
  {
    q = Partition(A,p,r);
    quick_sort(A,p,q-1);
    quick_sort(A,q+1,r);
  }
}

Partition(A,p,r)
{
  i = p -1 ;
  for (j = p; j < i; ++j)
  {
    if (A[j] <= A[r])
    {
      swap(A[i],A[j]);
    }
  }
  i++;
  swap(A[i+1],A[r]);
  return i;
}
```

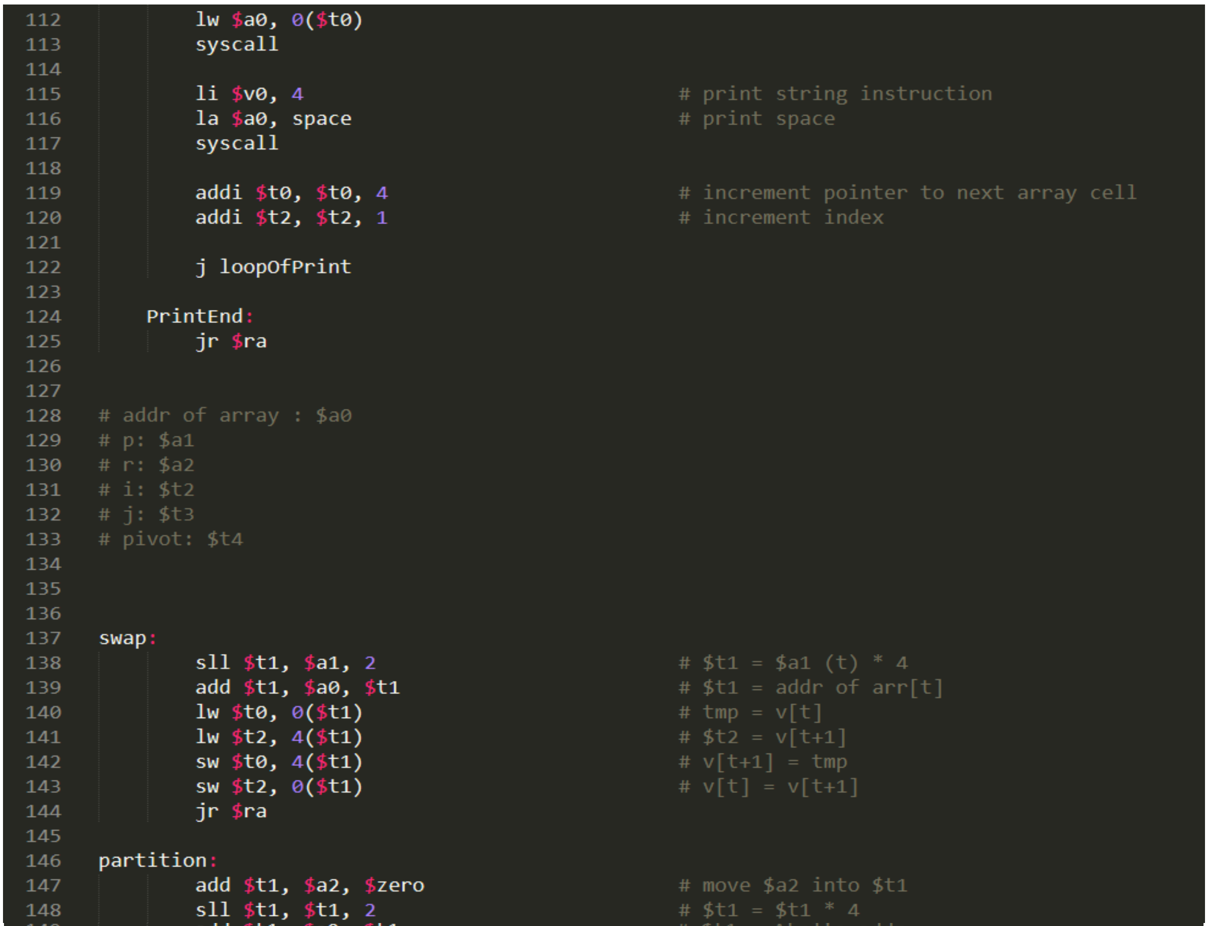
The core of the design is according to the c++   pseudo code on the below:



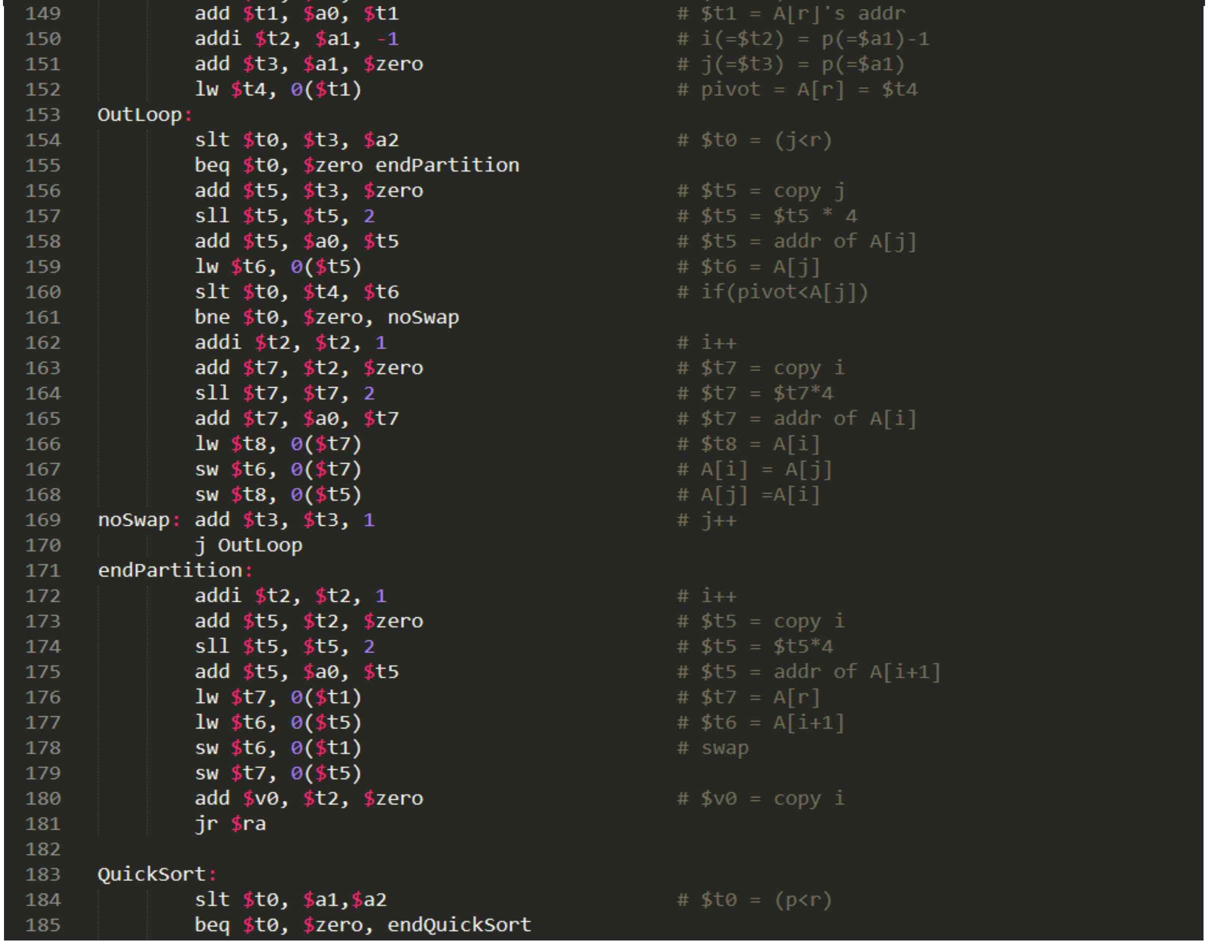
```
.data

array: .word 0 : 1000
comma: .asciiz ", "
newline: .asciiz "\n"
space: .asciiz " "
str1: .asciiz "Please enter the number of integers you want to sort: "
str2: .asciiz "Enter the integer: "
original: .asciiz "\nThe original array: "
sort: .asciiz "\nThe sorted array: "
switches: .asciiz "\n#switches: "
size: .word 0 #size of the actual array

.text

.globl main
main:
    getSize:
        # print str1 on console
        li $v0, 4                               # print string instruction
        la $a0, str1                            # load addr of str1 for syscall
        syscall
        # read the number of integers
        li $v0, 5                               # read integer and save in $v0
        syscall
        la $t0, size
        sw $v0, 0($t0)                          # transfer the number to $t0


    getInputIntergers:
        la $t0, array                           # $t0 = array address
        lw $t1, size                            # load size in $t1
        li $t2, 0                               # set index i = 0 ($t2)


    prompt:
        bge $t2, $t1, finishReadInput           # break loop if all the integers are obtained(while($t2<$t1))
```



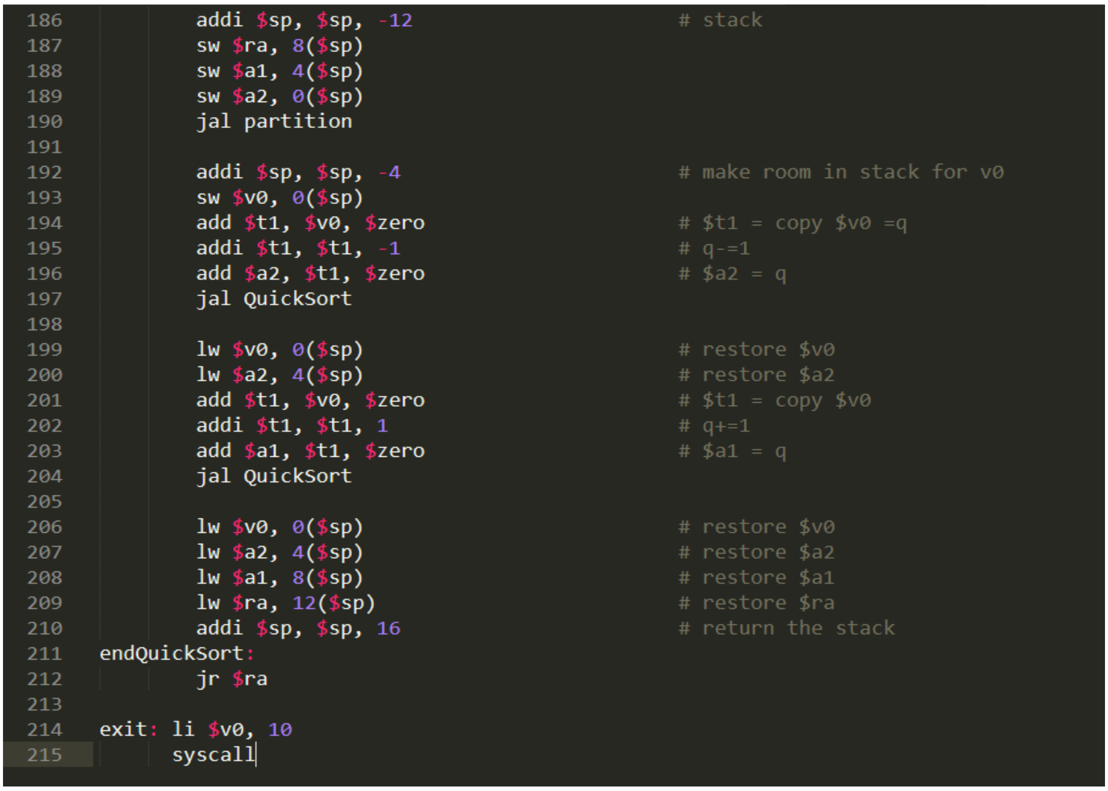
```
        li $v0, 4                               # print string instruction
        la $a0, str2
        syscall
        li $v0, 5                               # read integer and save in $v0
        syscall
        sw $v0, 0($t0)                          # store value
        addi $t0, $t0, 4                        # remove the current address to next array cell
        addi $t2, $t2, 1                        # increase index i, i++
        j prompt

    finishReadInput:
        li $v0, 4
        la $a0, original
        syscall

        jal Print

# $a0 = addr of array
# $a1 = zero
# $a2 = size -1

####################################
# pseudo code for c:
#
# Quicksort(A,p,r)
# {
#   if(p<r)
#   {
#     q = Partition(A,p,r);
#     quick_sort(A,p,q-1);
#     quick_sort(A,q+1,r);
#   }
# }
#
# Partition(A,p,r)
# {
#   i = p -1 ;
#   for (j = p; j < i; ++j)
#   {
#     if (A[j] <= A[r])
#     {
#       swap(A[i],A[j]);
#     }
#   }
#   i++;
#   swap(A[i+1],A[r]);
#   return i;
# }
#####################################

    la $a0, array                               # points $a0 = addr of array
    li $a1, 0                                   # left value
    lw $a2, size                                # right value
    addi $a2, $a2, -1
    jal QuickSort

    li $v0, 4
    la $a0, sort
    syscall

    jal Print
    j exit


Print:


    la $t0, array                               # load the array addr in $t0
    lw $t1, size                                # load size to $t1
    li $t2, 0                                   # load 0 (index) to $t2

    loopOfPrint:
        bge $t2, $t1, PrintEnd                  # while($t2<$t1)
        li $v0, 1
```



```
        lw $a0, 0($t0)
        syscall

        li $v0, 4                               # print string instruction
        la $a0, space                           # print space
        syscall

        addi $t0, $t0, 4                        # increment pointer to next array cell
        addi $t2, $t2, 1                        # increment index

        j loopOfPrint

    PrintEnd:
        jr $ra


# addr of array : $a0
# p: $a1
# r: $a2
# i: $t2
# j: $t3
# pivot: $t4



swap:
        sll $t1, $a1, 2                         # $t1 = $a1 (t) * 4
        add $t1, $a0, $t1                       # $t1 = addr of arr[t]
        lw $t0, 0($t1)                          # tmp = v[t]
        lw $t2, 4($t1)                          # $t2 = v[t+1]
        sw $t0, 4($t1)                          # v[t+1] = tmp
        sw $t2, 0($t1)                          # v[t] = v[t+1]
        jr $ra

partition:
        add $t1, $a2, $zero                     # move $a2 into $t1
        sll $t1, $t1, 2                         # $t1 = $t1 * 4
```

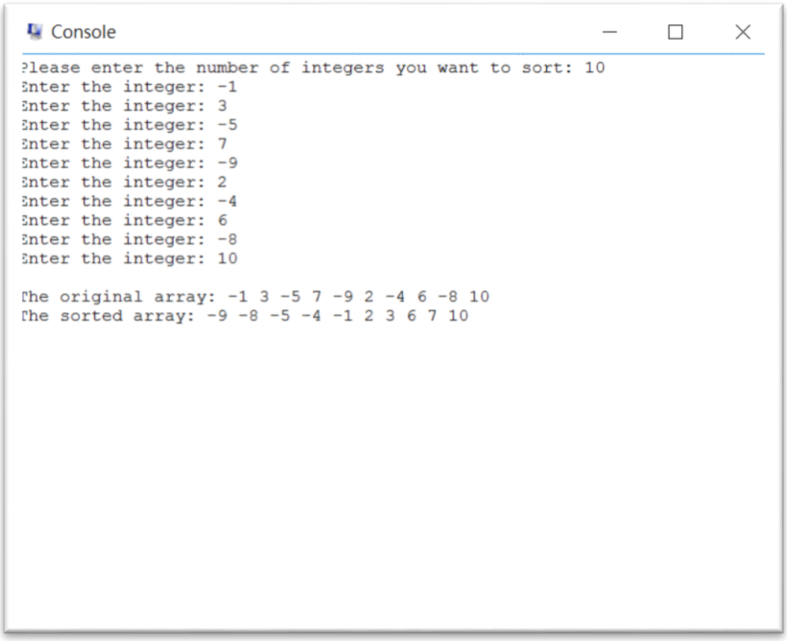


```
        add $t1, $a0, $t1                       # $t1 = A[r]'s addr
        addi $t2, $a1, -1                       # i(=$t2) = p(=$a1)-1
        add $t3, $a1, $zero                     # j(=$t3) = p(=$a1)
        lw $t4, 0($t1)                          # pivot = A[r] = $t4
OutLoop:
        slt $t0, $t3, $a2                       # $t0 = (j<r)
        beq $t0, $zero endPartition
        add $t5, $t3, $zero                     # $t5 = copy j
        sll $t5, $t5, 2                         # $t5 = $t5 * 4
        add $t5, $a0, $t5                       # $t5 = addr of A[j]
        lw $t6, 0($t5)                          # $t6 = A[j]
        slt $t0, $t4, $t6                       # if(pivot<A[j])
        bne $t0, $zero, noSwap
        addi $t2, $t2, 1                        # i++
        add $t7, $t2, $zero                     # $t7 = copy i
        sll $t7, $t7, 2                         # $t7 = $t7*4
        add $t7, $a0, $t7                       # $t7 = addr of A[i]
        lw $t8, 0($t7)                          # $t8 = A[i]
        sw $t6, 0($t7)                          # A[i] = A[j]
        sw $t8, 0($t5)                          # A[j] =A[i]
noSwap: add $t3, $t3, 1                         # j++
        j OutLoop
endPartition:
        addi $t2, $t2, 1                        # i++
        add $t5, $t2, $zero                     # $t5 = copy i
        sll $t5, $t5, 2                         # $t5 = $t5*4
        add $t5, $a0, $t5                       # $t5 = addr of A[i+1]
        lw $t7, 0($t1)                          # $t7 = A[r]
        lw $t6, 0($t5)                          # $t6 = A[i+1]
        sw $t6, 0($t1)                          # swap
        sw $t7, 0($t5)
        add $v0, $t2, $zero                     # $v0 = copy i
        jr $ra

QuickSort:
        slt $t0, $a1,$a2                        # $t0 = (p<r)
        beq $t0, $zero, endQuickSort
```



```
        addi $sp, $sp, -12                      # stack
        sw $ra, 8($sp)
        sw $a1, 4($sp)
        sw $a2, 0($sp)
        jal partition

        addi $sp, $sp, -4                       # make room in stack for v0
        sw $v0, 0($sp)
        add $t1, $v0, $zero                     # $t1 = copy $v0 =q
        addi $t1, $t1, -1                       # q-=1
        add $a2, $t1, $zero                     # $a2 = q
        jal QuickSort

        lw $v0, 0($sp)                          # restore $v0
        lw $a2, 4($sp)                          # restore $a2
        add $t1, $v0, $zero                     # $t1 = copy $v0
        addi $t1, $t1, 1                        # q+=1
        add $a1, $t1, $zero                     # $a1 = q
        jal QuickSort

        lw $v0, 0($sp)                          # restore $v0
        lw $a2, 4($sp)                          # restore $a2
        lw $a1, 8($sp)                          # restore $a1
        lw $ra, 12($sp)                         # restore $ra
        addi $sp, $sp, 16                       # return the stack
endQuickSort:
        jr $ra

exit: li $v0, 10
      syscall
```

Result:



```
Please enter the number of integers you want to sort: 10
Enter the integer: -1
Enter the integer: 3
Enter the integer: -5
Enter the integer: 7
Enter the integer: -9
Enter the integer: 2
Enter the integer: -4
Enter the integer: 6
Enter the integer: -8
Enter the integer: 10

The original array: -1 3 -5 7 -9 2 -4 6 -8 10
The sorted array: -9 -8 -5 -4 -1 2 3 6 7 10
```