integer firstUnsortedIndex, testIndex, elementToInsert;
for firstUnsortedIndex = 1 to (length-1) do
    testIndex = firstUnsortedIndex-1;
    elementToInsert = numbers[firstUnsortedIndex];
    while (testIndex >=0) AND (numbers[testIndex] > elementToInsert ) do
        numbers[ testIndex + 1 ] = numbers[ testIndex ];
        testIndex = testIndex - 1;
    end while
    numbers[ testIndex + 1 ] = elementToInsert;
end for

2. Write MIPS Assembly Language code for the above insertion sort algorithm

.data
numbers: .word 20, 30, 10, 40, 50, 60, 30, 25, 10, 5
length: .word 10

.text
.globl main
main:
    # Use $2 to hold firstUnsortedIndex
    # Use $3 to hold testIndex
    # Use $4 to hold elementToInsert
    # Use $5 to hold value of numbers[ .. ]
    # Use $6 to calculate the address of numbers[ ... ] in
    # Use $7 to hold the value of (length-1)
    # Use $8 to hold the base/start address of the numbers array
for_init:  li $2, 1
lw $7, length
sub $7, $7, 1
la $8, numbers
for_loop:  bgt $2, $7, end_for
    sub $3, $2, 1
    mul $6, $2, 4 # address of numbers[i]= base addr of numbers + i*(element size)
    add $6, $8, $6
    lw $4, 0($6)
while:      blt $3, 0, end_while
    mul $6, $3, 4 # address of numbers[i]= base addr of numbers + i*(element size)
    add $6, $8, $6
    lw $5, 0($6)
    ble $5, $4, end_while
    sw $5, 4($6)
    sub $3, $3, 1
    j while
end_while:  mul $6, $3, 4 # address of numbers[i]= base addr of numbers + i*(element size)
    add $6, $8, $6
    sw $4, 4($6)
    addi $2, $2, 1
    j for_loop
end_for:
    li $v0, 10 # system call to exit
    syscall
integer firstUnsortedIndex, testIndex, elementToInsert;

for firstUnsortedIndex = 1 to (length-1) do
testIndex = firstUnsortedIndex-1;
elementToInsert = numbers[firstUnsortedIndex];

while (testIndex >=0) AND (numbers[testIndex] > elementToInsert ) do
    numbers[ testIndex + 1 ] = numbers[ testIndex ];
testIndex = testIndex - 1;
end while

numbers[ testIndex + 1 ] = elementToInsert;
end for

MIPS Assembly Language code for the above insertion sort algorithm BY “WALKING POINTERS”
.data
numbers: .word 20, 30, 10, 40, 50, 60, 30, 25, 10, 5
length: .word 10

.text
.globl main
main:
    # Use $2 to hold address of numbers[firstUnsortedIndex]
    # Use $3 to hold address of numbers[testIndex]
    # Use $4 to hold elementToInsert
    # Use $5 to hold value of numbers[test]
    # Use $6 to calculate the address of numbers[test] in
    # Use $7 to hold the address of numbers[length-1]
    # Use $8 to hold the base/staring address of the numbers array
for_init:   la $2, numbers
            addi $2, $2, 4
            la $7, length
            sub $7, $7, 4
            la $8, numbers
for_loop:   bgt $2, $7, end_for
            sub $3, $2, 4
            lw $4, 0($2)
while:      blt $3, $8, end_while
            lw $5, 0($3)
            ble $5, $4, end_while
            sw $5, 4($3)
            sub $3, $3, 4
            j while
end_while:
            sw $4, 4($3)
            addi $2, $2, 4
            j for_loop
end_for:
    li $v0, 10 # system call to exit
    syscall