***Sorting Algorithm*:-by**

Sanjeev(IMT2022530),Pudi Kireeti(IMT2022059)

```
#your code
addi $t0,$0,0#i=0
addi $t7,$t2,0#address of starting
addi $t5,$t3,0#address of output
addi $t4,$0,0#j=0


subi $s2,$t1,1#s2=n-1
sll $t9,$t1,2#shifting t1 by 2
add $s5,$t9,$t7
jal loop3

loop3:
        slt $t9,$t0,$s2#i<n-1, t9=1 else t9=0
        beq $t9,$0,done# if t9=0 stop
        addi $t6,$0,0#swapped=0
        sub $t8,$s2,$t0#t8=n-1-i
        jal loop2

loop2:
        slt $t9,$t4,$t8#t7=j<n-i-1 then t9=1 else t9=0
        beq $t9,$0,back# if t9=0 stop
        addi $s1,$t8,0#s1=n-i-1
        addi $sp,$sp,-8#making space in stack for 4 numbers
        sw $s1,4($sp)#changed 4 to 8
        addi $s7,$t7,4
        slt $t9,$s7,$s5
```

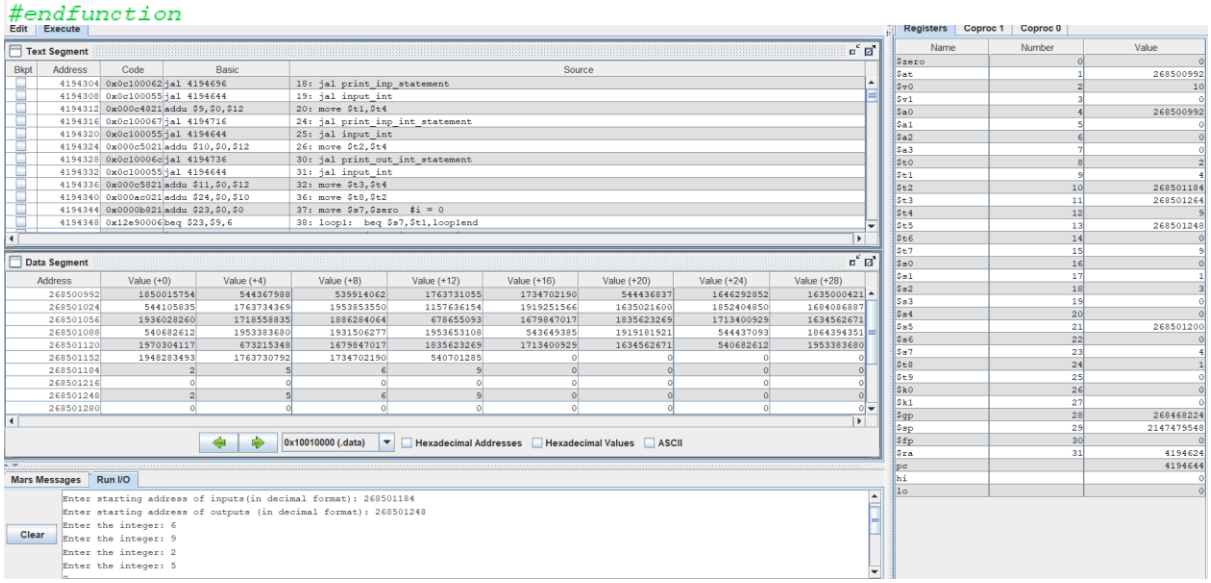
```
loop2:
        slt $t9,$t4,$t8#t7=j<n-i-1 then t9=1 else t9=0
        beq $t9,$0,back# if t9=0 stop
        addi $s1,$t8,0#s1=n-i-1
        addi $sp,$sp,-8#making space in stack for 4 numbers
        sw $s1,4($sp)#changed 4 to 8
        addi $s7,$t7,4
        slt $t9,$s7,$s5
        beq $t9,$0,done
        lw $t9,0($t7) # A[j]
        #addi $s5,$s4,0#s5=A[j]
        lw $t8,4($t7)#as we are dng j+1 s7 also should be changed
        addi $s0,$t4,0#s6=A[j+1]
        sw $s0,0($sp)
        slt $t4,$t8,$t9#if A[j+1]<A[j] then t4=1 else t4=0
        bne $t4,$0,swap#if t4!=0 then go to swap
        lw $t4,0($sp)
        addi $t4,$t4,1#j=j+1
        addi $t7,$t7,4
        #addi $t5,$t5,4
        lw $t8,4($sp)
        addi $sp,$sp,8
        j loop2
back:
        addi $t4,$0,0
        beq $t6,$0,done
```

```
back:
        addi $t4,$0,0
        beq $t6,$0,done
        addi $t0,$t0,1
        addi $t7,$t2,0
        j loop3
swap:
# a method for swapping
        #lw  $s5,0($s7)
        #sw $s5,4($s7)
        sw $t9,4($t7)
        sw $t8,0($t7)
        #sw $t9,4($t7)
        #sw $t8,0($t7)
        #beq $s5,$s5,loop2

        #addi $s7,$s7,4
        addi $t6,$t6,1#making swapped=1
        lw $t4,0($sp)
        lw $t8,4($sp)
        addi $t4,$t4,1
        addi $t7,$t7,4
        #addi $t5,$t5,4
        addi $sp,$sp,8
        #addi $s1,$s1,1#j=j+1
        j loop2
```

```
        lw $t4,0($sp)
        lw $t8,4($sp)
        addi $t4,$t4,1
        addi $t7,$t7,4
        #addi $t5,$t5,4
        addi $sp,$sp,8
        #addi $s1,$s1,1#j=j+1
        j loop2

done:
        lw $t7,0($t2)
        sw $t7,0($t5)
        lw $t7,4($t2)
        sw $t7,4($t5)
        lw $t7,8($t2)
        sw $t7,8($t5)
        lw $t7,12($t2)
        sw $t7,12($t5)
#endfunction
```



The above output is the output for the sorting Algorithm(Bubble Sort) we have written.

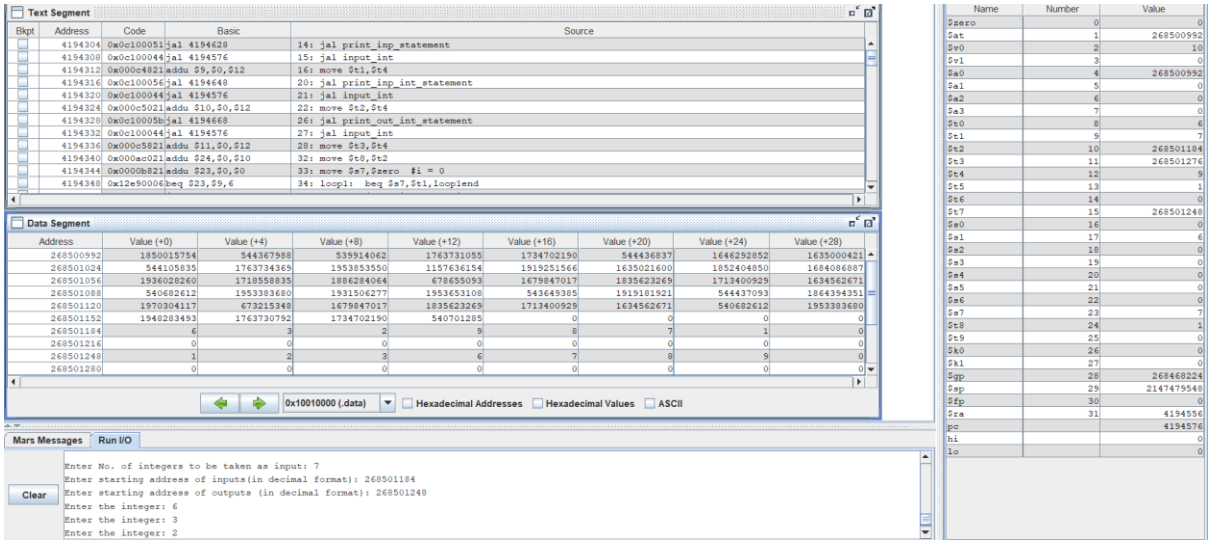
What we have done is, we have 1st initialised all the variables, created space for inputs by shifting it left and adding the input address to it.

1st loop the loop3 if the 1st loop in the program that checks if i<n-1 and initializes a variable swapped to be 0 and we jump to next loop, in the next loop we check the condition j<n-i-1 and proceed for the next following steps where we initialize and stack pointer creating space in it, the we see if j+1> j and the compare A[j] to A[j+1] and if A[j]>A[j+1] the we swap else we got back increment j and do the same again.

The swap function shown is to swap the elements and increment j and address of the input array. At the end we have done and after that is the to show the output up to 4 numbers in the output array.

In this particular program the output array and the input array have the same sorted array.

2nd Code



In the below program we have made slight changes that gives inputs only in the input array and sorted array in the output array.

We have done this by copying these inputs In both input array and output array, we display that in the inputs and in the output array we 1st sort the given array and then display it.

The pictures below show the code and the picture above shows the input and output.

```
#t1 -> n; t2 -> input address; t3 -> output address
addi $s1,$t1,0#s1 is assigned n
addi $t5,$t2,0#t5 is assigned in input address
addi $t7,$t3,0#t7 is given the output address
addi $t0,$0,0#t0=i is assigned 0
copy:
        slt $t9,$t0,$s1#if0<n t9=1 else t9=0
        beq $t9,$0,initial#when t9=0 initial
        lw $t9,0($t5)#storing the 1st element into t9
        sw $t9,0($t7)#giving that element to 0th address of t7
        addi $t5,$t5,4#incrementing input address
        addi $t7,$t7,4#incrementing the output address
        addi $t0,$t0,1#i++
        j copy

initial:
        subi $s1,$s1,1#n=n-1
        addi $t0,$0,0#making t0 0 again
        addi $t4,$0,0#t4=0=j
        addi $t7,$t3,0#t7=t3
        j loop3

loop3:
        slt $t9,$t0,$s1#if i<n-1 t9=1 else t9=0
        beq $t9,$0,exit#if t9=0 directly stop
        sub $t8,$s1,$t0#t8=n-1-i
        slt $t9,$t4,$t8#if j<n-i-1 the t9=1 else t9=0
        slt $t9,$t4,$t8#if j<n-i-1 the t9=1 else t9=0
        bne $t9,$0,loop2#if t9!=0 go to loop2
        addi $t4,$0,0#when the above conditon satisfy make jback to 0
        addi $t0,$t0,1#i++
        addi $t7,$t3,0
        j loop3

loop2:
        lw $t9,0($t7)#t9=A[j]
        lw $t5,4($t7)#t5=A[j+1]
        slt $s2,$t9,$t5#if A[j]<A[j+1] s2=1 else s2=0
        beq $s2,$0,swap #if s2=0 the do swap
        addi $t4,$t4,1#j++
        addi $t7,$t7,4#go to next output address
        j loop3

swap:
        sw $t5,0($t7)#t5 has A[j] and it is shifted to 1st address of the output address
        sw $t9,4($t7)#t9 has A[j+1] and that is shifted to 2nd address of the outpput address
        addi $t4,$t4,1#j++
        addi $t7,$t7,4#increment output address
        j loop3

exit:
```

Machine Language:

```
00100001001100010000000000000000
00100001010011010000000000000000
00100001011011110000000000000000
00100000000010000000000000000000
00000001000100011100100000101010
00010011001000000000000000000110
10001101101110010000000000000000
10101101111110010000000000000000
00100001101011010000000000000100
00100001111011110000000000000100
00100001000010000000000000000001
00001000000100000000000000000100
00100000000000010000000000000001
00000010001000011000100000100010
00100000000010000000000000000000
00100000000011000000000000000000
00100001011011110000000000000000
00001000000100000000000000010010
00000001000100011100100000101010
00010011001000000000000000010011
00000010001010001100000000100010
00000001100110001100100000101010
00010111001000000000000000000100
00100000000011000000000000000000
00100001000010000000000000000001
00100001011011110000000000000000
00001000000100000000000000010010
10001101111111001000000000000000
10001101111011010000000000000100
00000011001011011001000000101010
00010010010000000000000000000011
00100001100011000000000000000001
00100001111011110000000000000100
00001000000100000000000000010010
10101101111011010000000000000000
10101101111111001000000000000100
00100001100011000000000000000001
00100001111011110000000000000100
```

```
00100001000010000000000000000001
00001000000100000000000000000100
00100000000000010000000000000001
00000010001000011000100000100010
00100000000010000000000000000000
00100000000011000000000000000000
00100001011011110000000000000000
00001000000100000000000000010010
00000001000100011100100000101010
00010011001000000000000000010011
00000010001010001100000000100010
00000001100110001100100000101010
00010111001000000000000000000100
00100000000011000000000000000000
00100001000010000000000000000001
00100001011011110000000000000000
00001000000100000000000000010010
10001101111111001000000000000000
10001101111101101000000000000100
00000011001011011001000000101010
00010010010000000000000000000011
00100001100011000000000000000001
00100001111011110000000000000100
00001000000100000000000000010010
10101101111011010000000000000000
10101101111111001000000000000100
00100001100011000000000000000001
00100001111011110000000000000100
00001000000100000000000000010010
```

The above binary text is the binary conversion of the 2nd/ the previous code which is called the machine language.

This is the assembler we have built for our particular sorting algorithm:

```
import re

regis_numb = {
    "$s6": "10110",
    "$t2": "01010",
    "$t5": "01101",
    "$s4": "10100",                                          #defined dictionaries for registers
    "$a1": "00101",
    "$v1": "00011",
    "$v0": "00010",
    "$t4": "01100",
    "$t0": "01000",
    "$a3": "00111",
    "$t6": "01110",
    "$k1": "11011",
    "$t3": "01011",
    "$sp": "11101",
    "$s2": "10010",
    "$t9": "11001",
    "$s5": "10101",
    "$s3": "10011",
    "$0": "00000",
    "$s7": "10111",
    "$t8": "11000",
    "$a0": "00100",
    "$ra": "11111",
    "$s1": "10001",
    "$at": "00001",
    "$t1": "01001",
    "$s0": "10000",
    "loop3": "00000100000000000000010010",
    "$t7": "01111",
    "back": "10000000000000110101",
    "loop2": "0000000000000100",
    "exit": "0000000000010011",
    "swap": "00000100000000000000010010",
    "initial": "0000000000000110",
    "copy": "00000100000000000000000100",
    "swap": "00000100000000000000010010",
    "initial": "0000000000000110",
    "copy": "00000100000000000000000100",
    "swapbeq" : "0000000000000011"

}


J_ins = {'jal': '000011', 'j': '000010'}
I_ins = {'addi': '001000', 'lw': '100011', 'sw': '101011', 'beq': '000100','subi':'000000', 'bne':'000101'}
R_ins = {'sll': '000000', 'slt': '000000', 'sub': '000000', 'add': '000000'}




with open('bubble-sort.asm','r') as file_read:

    for line in file_read.readlines():
        if line.strip():          #for splitting of lines

            line = line.strip()
        # leading tab and trailing newline are removed


        segments = re.split(r'[,\s()]+', line)
                                  # Splitting by commas,spaces and parentheses
        instruction = segments[0]
        if(segments[0]=="addi"):
            print(I_ins[segments[0]] + regis_numb[segments[2]] + regis_numb[segments[1]] + str(bin(int(segments[3]))[2:].zfill(16)))
        if(segments[0]=="bne"):
            print(I_ins[segments[0]] + regis_numb[segments[1]] + regis_numb[segments[2]] + regis_numb[segments[3]])
        if(segments[0]=="beq"):
            if(segments[3]!="swap"):
                print(I_ins[segments[0]] + regis_numb[segments[1]] + regis_numb[segments[2]] + regis_numb[segments[3]])
            else :
                print(I_ins[segments[0]] + regis_numb[segments[1]] + regis_numb[segments[2]] + regis_numb[segments[3]+"beq"])
        if(segments[0]=="lw"):
            print(I_ins[segments[0]] + regis_numb[segments[3]] + regis_numb[segments[1]] + str(bin(int(segments[2]))[2:].zfill(16)))
```

```
segments = re.split(r'[,\s()]+', line)
                        # Splitting by commas,spaces and parentheses
instruction = segments[0]
if(segments[0]=="addi"):
    print(I_ins[segments[0]] + regis_numb[segments[2]] + regis_numb[segments[1]] + str(bin(int(segments[3]))[2:].zfill(16)))
if(segments[0]=="bne"):
    print(I_ins[segments[0]] + regis_numb[segments[1]] + regis_numb[segments[2]] + regis_numb[segments[3]])
if(segments[0]=="beq"):
    if(segments[3]!="swap"):
        print(I_ins[segments[0]] + regis_numb[segments[1]] + regis_numb[segments[2]] + regis_numb[segments[3]])
    else :
        print(I_ins[segments[0]] + regis_numb[segments[1]] + regis_numb[segments[2]] + regis_numb[segments[3]+"beq"])
if(segments[0]=="lw"):
    print(I_ins[segments[0]] + regis_numb[segments[3]] + regis_numb[segments[1]] + str(bin(int(segments[2]))[2:].zfill(16)))
if(segments[0]=="sw"):
    print(I_ins[segments[0]] + regis_numb[segments[3]] + regis_numb[segments[1]] + str(bin(int(segments[2]))[2:].zfill(16)))
if(segments[0]=="subi"):
    print(I_ins[segments[0]] + regis_numb[segments[2]] + regis_numb[segments[1]] + str(bin(int(segments[3]))[2:].zfill(16)))
if(segments[0]=="add"):
    print(R_ins[segments[0]] + regis_numb[segments[2]] + regis_numb[segments[3]] + regis_numb[segments[1]] + '00000' + '100010')
if(segments[0]=="sub"):
    print(R_ins[segments[0]] + regis_numb[segments[2]] + regis_numb[segments[3]] + regis_numb[segments[1]] + '00000' + '100010')
if(segments[0]=="j"):
    print(J_ins[segments[0]] + regis_numb[segments[1]])
if(segments[0]=="slt"):
    print('000000' + '00000' + regis_numb[segments[2]] + regis_numb[segments[1]] + "00000" + '101010')
if(segments[0]=="jal"):
    print(J_ins[segments[0]] + regis_numb[segments[1]])
```

The below binary text is the binary code given when the assembler is run and as we can see the binary text of assembler and the 2nd code of sorting is the same

```
rs\harsh\.vscode\extensions\ms-python.python-2023.16.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '53285' '--' 'C:\Users\harsh\OneDrive\
Desktop\caassembler.py'
00100001001100010000000000000000
00100001010011010000000000000000
00100001011011110000000000000000
00100000000010000000000000000000
00000000000010001100100000101010
00010011001000000000000000000110
10001101101110010000000000000000
10101101111110010000000000000000
00100001101011010000000000000100
00100001111011110000000000000100
00100001000010000000000000000001
00001000000100000000000000000100
00100000000000100000000000000001
00000010001000011000100000100010
00100000000010000000000000000000
00100000000011000000000000000000
00100001011011110000000000000000
00001000000100000000000000010010
00000000000010001100100000101010
00010011001000000000000000010011
00000010001010001100000000100010
00000000000011001100100000101010
00010111001000000000000000000100
00100000000011000000000000000000
00100001000010000000000000000001
00100001011011110000000000000000
00001000000100000000000000010010
10001101111110010000000000000000
10001101111011010000000000000100
00000000000110011001000000101010
00010010010000000000000000000011
00100001100011000000000000000001
00100001111011110000000000000100
00100001111011110000000000000100
00001000000100000000000000010010
10101101111011010000000000000000
10101101111110010000000000000100
00100001100011000000000000000001
00100001111011110000000000000100
00001000000100000000000000010010
PS C:\Users\harsh\OneDrive\Desktop>
```