For the problems given, develop MIPS programs that satisfy the specifications in the problem statement. Remember: To turn in this homework, email your NotePad files of the programs as attachments to the TA. Note: these are more advanced problems, and deserve careful consideration and focus.

1. (CLO 5—Assy Lang.) Develop a program that arranges the series of decimal numbers shown in ascending order from smallest to largest, then prints them out. If you wish, you can copy and paste the program in the space to your left so that you will have a printed record of it.

When the numbers are printed, put a carriage return/line feed between each, for neatness. Note 1: You will not need any counters, as the data provided has a final zero value, so that all you have to do is test for 0 and then quit. Note that the 0 is NOT to be printed out—it is not a value to be compared but merely an “edge” to tell you that the list is over.

Note 2: This must be a recursive program.

Comments: This program is a good example of the use of recursion to “keep one’s place” in the string of numbers, when a given number must be “backed up” and inserted into its rightful place. Using recursive code, the “jr” section, or “procedure uncall,” gets the sort back to the next number in the sequence to be compared and placed. This program is very similar in concept to the alphabetization program from Lecture 17.
2. (CLO 5—Assy Lang.) Compose a program to initialize an array with a series of numbers. This program will be very similar to one that we did in class, but with a twist or two. The array is to be 20 X 20 words, so you will need to use the “.space” command to initialize the array size. It should be 400 words (20 X 20), but remember, using the “.space” directive, you have to reserve bytes.

In this case, you are going to load each row of the array with slightly different numbers. You will be loading rows 0-19, and columns 0-19, or 20 X 20. Remember that the first row is 0! For even rows (including 0, that is, 0, 2, 4, 6, etc.), load the numbers 1-20 consecutively in the 20 columns. For odd rows (1, 3, 5, 7, etc.) load the numbers 20-1 consecutively, that is, in reverse order.

Thus, you will load even rows with the sequence 1-20, and odd rows with the sequence 20-1.

As in problem one, you can print out and paste a copy of the program in the space at the right for your records.

```assembly
.text
main: la $t9,array # load array address
li $t0,0 # initializing row counter
row: beq $t0,20,done # done yet?
    li $t1,0 # initializing column counter
    mul $t4,$t0,80 # initializing row address
    add $t9,$t9,$t4 # "
    div $t2,$t0,2 # odd or even row?
    mfhi $t2 # "
    bnez $t2,orow # "
    erow: li $t5,1 # initialize even-row number
        sw $t5,0($t9) # store element
        addi $t1,$t1,1 # increment element value
        j eloop # do it again
    urow: addi $t0,$t0,1 # increment row counter
        la $t9,array # re-initialize array address
        j row # next row
    orow: li $t5,20 # initialize even-row number
        sw $t5,0($t9) # store element
        addi $t1,$t1,1 # increment column counter
        beq $t1,20,uloop # stored 20 columns?
        addi $t9,$t9,4 # no, calc. add. or next pos.
        addi $t5,$t5,1 # increment element value
        j oloop # do it again
        done: li $v0,10 # done; QUIT!
        syscall
.data
array: .space 1600
```

Comments: This program will take a little time, but it is relatively simple if you take the time to review the array program shown in class. Note that, as in the classroom problem, you will need nested loops to do the program most easily.

However, unlike the classroom example, you are NOT storing a simple value. The value you store must be incremented every time a column element is stored.

Further, adjacent rows have elements that “go in the opposite direction.” Even rows count UP, odd rows count DOWN.

You may have found (as the example to the right shows) that having an “even loop” and an “odd loop” simplifies bookkeeping in the inner loop storage routine. That is, setting up TWO (2) inner loops helps.
3. (CLO 5—Assy Lang.)
Compose a program to arrange the characters in “Hello, world!\n” in numerical order, smallest value to largest, and print out the result. That will of course mean that the string of characters looks more or less like gibberish, but the characters will be in proper numerical order, smallest to largest.

Note that this requires a recursive program to be done properly. As for the other two problems in this set, you can paste a copy of the program to the right for your records, if you wish.

Comments: This program is almost identical to the recursive loop program that was shown in class in which a set of ASCII small letters were alphabetized. In this case, you simply put ALL characters in numerical order, rather than just lower-case letters. The result looks a bit odd, but the characters are in strict order, smallest numerical value to largest. If you paid attention in class and took a little time to study the class example, this problem was VERY easy!