EE/CS/CPE 3760 Program #3

**Programming Problem**

You are to write a program in MIPS assembly for use in the SPIM simulator. You will implement a “smart” bubble sort in accordance with the following pseudocode:

* Consider an unsorted array X with n elements.
* Scan X[0], X[1], ... X[n-1], comparing adjacent elements (X[0] to X[1], X[1] to X[2], etc.)
  + Swap the adjacent elements if they are out of order (e.g., if X[0] > X[1])
* If no exchanges took place during the scan, the list has been sorted. Otherwise, scan X[0] through X[n-2] in the same manner as before (note: after each scan, we are scanning 1 element less!).
* Continue until no exchanges take place during a pass **OR** n-1 passes have been made.

**Requirements:**

1. Your code must be well-commented to receive full credit. This means any non-trivial line of code should have an accompanying comment.
2. The array to be sorted shall be stored in memory (hardcoded) using an assembler directive. The size of the array shall also be stored in memory (hardcoded) via an assembler directive. Example:   
   Xarray: .word 14, -2, 3, 5, 9  
   Xsize: .word 5
3. You must use a **sub-routine** to implement a comparison function (i.e., to compare adjacent elements to see if a swap is needed). The sub-routine shall take the **addresses** (see Hint 4 below) of the two items to compare as **input parameters**, perform the comparison, and **return** the result of the comparison. A 1 shall be returned if the first element is greater than the second element (i.e., a swap is needed), and a 0 shall be returned otherwise (i.e., no swap needed).
4. You must use a **sub-routine** to implement the swap functionality. The sub-routine shall take the **addresses** of the two items to swap as **input parameters**, and perform the swap. The sub-routine will return nothing.
   1. Note: you may do so, but you are not required to call the swap subroutine from the other subroutine – both can be called from within loops in main. Whichever method you choose, you **must** preserve the old return address across calls.
5. When writing sub-routines, you must follow MIPS register conventions. This means all input parameters **must** be passed using the argument registers (a0-a3) and all return values via the value return registers (v0-v1). In addition, any callee-saved registers ($s registers) used must be saved to the stack and restored before return.
6. Your program must print the sorted list to the screen upon completion.
7. (Optional) Your program may print out the list at the end of each iteration so that the user can see the array as it is being sorted. If you make a subroutine to display the array (you need this for #6 anyway) it will make this simple.
8. Test your program using the following arrays:

15, 5, -3, 8, 5, 1

425, 94, 16

20, 19, 18, 17, 16, 15

**Hints/Tips:**

1. Start by turning the above pseudocode into C/C++
   1. Make sure you understand what loops are needed, what the bounds are, etc.
   2. Look at Malik’s description of bubble sort in the C++ Programming textbook (from CSC1230/2430/2431) if needed (he gives good examples).
   3. Check Wikipedia and youtube – lots of animations and pseudocode for bubble sorts are available.
2. Begin to translate your high-level code by:
   1. Outlining in a text file with comments (e.g., “loop from start to finish”)
   2. Make your comments more detailed to fill in the “extra” assembly code (e.g., comments noting which registers correspond to which variables)
   3. Start writing assembly code to implement each step in your comments
3. Test incrementally, and backup working versions in case you need to revert.
4. To load in the address of a labelled location, use the LA (Load Address) pseudoinstruction. *Example: LA $a0, Xarray //put the addresss of Xarray into $a0*

**Turning in your solution:**

1. Save your code as a .s file, and name it using the following convention: “last\_first\_prog3.s”. For example, “Lincoln\_Abraham\_prog3.s”.
2. Take screenshots of the three test cases, and paste into a Word document. Name the document using the following convention: “last\_first\_prog3.docx”. For example, “Lincoln\_Abraham\_prog3.docx”.
3. Upload your two files to Canvas. Make sure to upload **both files**.