

## Project: Sort of Fun

**Due: Wednesday November 1 – by 4:10 PM (Before Class)**

**Submission: Two Files Uploaded to Carmen**

**This quiz is individual work. You can consult class materials, you can research algorithms (books, www etc.) but you are not allowed to collaborate with others.**

For this project you need to employ diverse skills such as loading data into the MCU memory, using graphing tools in CCS, visualizing Q format numbers in CCS, and writing subroutines for a given contract. We will cover these CCS tools in class on Wednesday October 25. If you are not able to attend the class, I highly recommend that you check out all the materials for Lecture 18 before you start working on this project.

### Motivation:

One could argue that sorting is one of the most fundamental operations in computing and that no coding class is complete without a sort. There is a place for sorting in many real-life applications too.

You know how to compute the average of a given array of numbers. If you were interested in the median instead of the average – well, you would have to start by sorting the array. That is exactly what you will do for this project: Sort the elements of a given array from smallest to largest.

There are many sorting algorithms: bubble sort, cocktail shaker sort (no kidding), heap sort, insertion sort, merge sort, quick sort, selection sort and even spaghetti sort – all with different computational complexity and other characteristics. And yes, spaghetti sort is not a good one.

For this project, you will implement **selection sort**. Selection sort is not the most efficient in terms of computational complexity. If you are sorting an array of  $n$  elements, the runtime will grow with the square of  $n$ . That means, sorting twice as many elements will take four times longer. But what selection sort lacks in speed, it makes up in simplicity. It is one of the simplest sort algorithms to implement, hence well suited for our class.

First familiarize yourself with how the selection sort works. You can consult any resource. Khan Academy might be a good starting point:

<https://www.khanacademy.org/computing/computer-science/algorithms/sorting-algorithms/a/sorting>

Once you have a good understanding of how selection sort works, you can start coding. Let the fun begin.

## Part 1: Coding Task

You will implement three subroutines called `sort`, `select`, and `swap` whose contracts are given in the file “project.asm” that is available on Carmen. Download this file and add your code to it as indicated. All your subroutines have to follow their contracts very closely, modifying core registers they are not allowed to modify may break the rest of the code.

Here is what these subroutines do:

The subroutine `swap` is the easiest to implement. This subroutine takes the addresses of two words in memory and swaps the content. The addresses are input in registers `R8` and `R9`. These addresses are not modified but the content at these addresses is swapped. For example, if `R8=0x1C00` and `R9=0x1C02` the subroutine will swap the word stored at `0x1C00` with the word stored at `0x1C02`. This subroutine still needs to work correctly when both addresses are the same.

There are many ways of doing swaps and you might have already swapped before (it was required for the challenge levels in Quiz 4). You are free to choose your favorite swap. I am partial to swapping over the stack: push, move, pop and you are done in three simple lines.

The subroutine `select` simply returns **the address of the smallest element** of a given array (of 16-bit signed numbers). Its input are the starting address of the array and the number of elements in the array. We have discussed the algorithms to find the minimum and maximum value. You have already written code to find the maximum value in an array for Midterm 1. This time you will find the address of the minimum element. If the smallest value is occurring multiple times, the subroutine can return the address of any of the occurrences.

The subroutine `sort` will reorder the elements in a given array (of signed 16-bit numbers). It takes as input the starting address of the array and the number of elements in the array. You will implement **selection sort**: start at the beginning of the given array, find the address of the smallest element, and swap it with the first element. You have your first element sorted.

Then, start at the second element of the array, find the smallest among the remaining elements (including the second element) and swap this with the second element. At this point you have the first two elements sorted. Repeat this with the unsorted elements: find the next smallest element, swap it with the third element etc.

The subroutine `sort` will call both `select` and `swap`, so you want to make sure that both subroutines work correctly before bringing all together. Once you are done with your implementation, have a look at the resulting code. Make sure that your code is spaghetti free. Ask yourself if you could possibly simplify it, remove any unnecessary lines, if any, etc. And most importantly, make sure to test them under different scenarios.

Remember, good code is correct, spaghetti-free, has good comments that makes it easy to follow. These practices will make both your and our lives easier (we = me and the TAs).

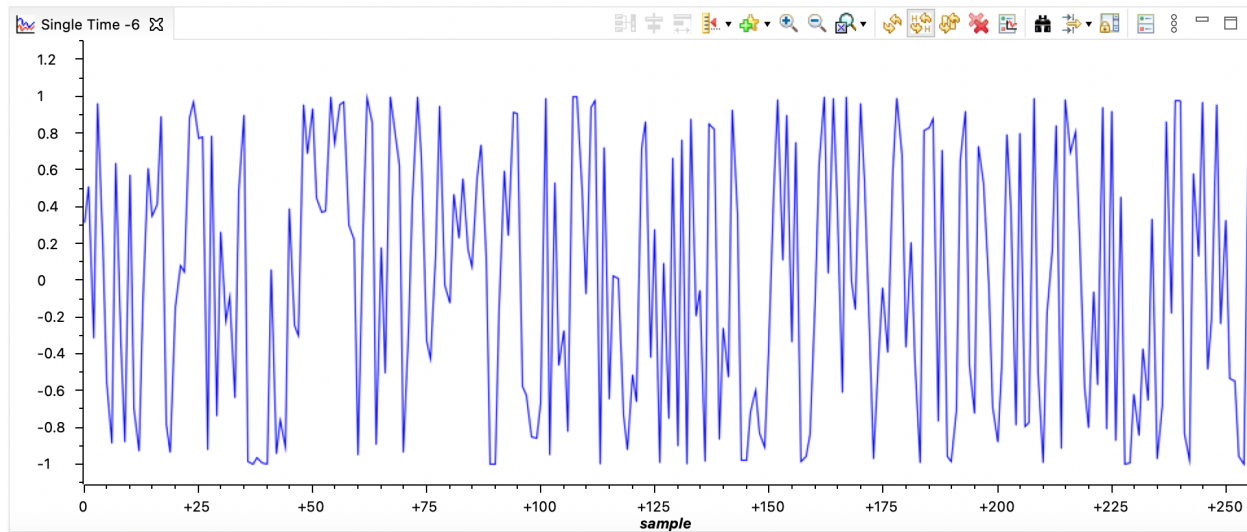
**Please format your assembly code like assembly code – see lecture notes for more detail.**

## Part 2: Importing data into MCU memory and graphing

Download the data file “project\_Q8.dat” from Carmen. You can open and view the contents of this file in CCS. Check the header line and the range of values. The data file contains **256 data points**. These are 16-bit signed numbers with Q value 8 in CCS, i.e., Q(8.8) format.

Use the Load Memory tool in CCS to place the data from “project\_Q6.dat” starting at the label `array` that has been defined in RAM. For more information, see the instructions given in Lecture 18. To confirm that you have correctly loaded the data, view it in the memory browser as 16-bit Signed Int with Q value 8. All values in the array should be between 1 and -1.

Graph the unsorted array. Then run your code to sort the array and graph the sorted version. Make sure to use correct parameters, including the correct Q value, so that both graphs show the correct range – all values should be between -1 and 1. The unsorted array should look like this:





You will submit screenshots of both the unsorted and sorted arrays. Please crop/resize all your screenshots so that they are easy to read.

## Part 3: Preparing two files for submission – please follow all instructions

Once you have written and tested your subroutines `select`, `swap` and `sort`, you are ready to run the code and prepare your submission.

After assembling and uploading the code to the MCU, make sure to load the data file into memory as outlined in Part 2. Confirm that the array has been populated correctly, graph it, and take your screenshot.

Run your code to sort the array. Make sure to execute the code fully before you graph your sorted array and take another screenshot. To do so, you press the green triangle button , wait a few seconds, and press pause . You can confirm that your program has completed execution if it is in the infinite loop.

You will submit two files:

### A single PDF file with two parts

#### 1. The graphs of the original and the sorted arrays

For both the graphs choose “**16-Bit Signed Int**” and Q-value 8.

#### 2. The screenshots of the three subroutines

Add three separate, legible screenshots of your subroutines `sort`, `select`, and `swap` that show all instructions in the subroutines. Make sure to include line numbers in your screenshot to facilitate easy feedback.

You do not need to submit a screenshot of the main program or other parts in `project.asm`. Please switch to Light Mode for easy readability. Remember, a good screenshot shows all the code uninterrupted, is properly sized, and easy to read.

To take a partial screenshot in Windows, press the **Windows Button and Shift and S**. This will bring up a crosshair which allows you to take the screenshot of the selected rectangular area. In macOS, press **command and control and shift and 4** to bring up the crosshair.

### Text File: Your source code

Save the final version of your source code as txt file so it can be read in Carmen, and make sure that both files reflect your name, e.g., `firstname_lastname.txt` or `name_number.txt`. Easiest way to do this is to use `File>Save As` in CCS.

I will randomly select and run source code files throughout the semester. If your source code file does not produce your claimed results (here the graphs) you will receive zero points for the assignment – end of story. Make sure to submit the correct source code in the correct format – no word or PDF files for source codes.

## Part 4: Submission

Make sure that your PDF file contains

1. Graphs of the array before and after sorting with correct length and Q-value.
2. Good screenshots of your subroutines `sort`, `select`, and `swap`.

Make sure that you are submitting the correct source code in the correct format (text file).

**Submit your PDF and text file to Carmen before class on Wednesday November 1, 2023.**

Happy Halloween 🎃