CIS11 Course Project Part 1: Documenting the Project

Introduction

1.1 Purpose

The Bubble Sort Algorithm's overall objective is to iteratively compare adjacent pairs of elements in an array, and swap them if they are not in order.

1.2 Intended Audience and Users

The algorithm is catered to users who seek to sort minute sets of numbers using a simple sorting technique. It should not be used by users who need to sort extremely large sets of data in a short amount of time.

1.3 Product Scope

The Program's overall intention is to short small lists of numbers numerically.

1.4 Reference

Source Documents for the Program Requirements and Specification

Sample Input: User is prompted to input 8 numbers, ranging from 0 - 100

11	8	2	17	6	4	3	21

Sample Output after Bubble Sort: Display sorted values in ascending order in console.

2	3	4	6	8	11	17	21

2. Overall Description

2.1 Product Perspective

The primary program simply asks the user to input 8 numbers ranging from 0-100, and from there performs the BubbleSort program in LC-3

The data type is restricted to numbers only, and it’s unlikely that other data formats will be supported in the future.

2.2 Product Functions

The overall description of functionality:

The main programs task is catered for ease of use on the users side. Simply input 8 numbers, and a sorted list is returned. The algorithm working behind the scenes utilizes a loop that will execute a swap conditional as a counter iterates through all the numbers inputted by the user. Once the swap conditional flag is no longer valid, or the iterative loop ends, the program can return the list.

Technical functionality

Technical Functions are as listed:
Initialization: Basic preparatory steps like declaring program start address, registers, labels, etc.

Input Loop: In order to properly obtain the user input, a loop will have to run, prompting the user to enter numbers until the 8th number is given, and then it will move to sort.

Bubble Sort: The algorithm will contain iterative loops, conditional statements, and stack management protocols. An outer loop will be placed to reset the flag as it iterates through the given numbers and sorts them. Once the outer loop no longer is valid (the sorted flag is true), the loop ends. Inside, is the sorting process, which will need a stack operation to store and restore values of the register that need to be preserved and then popped back into registers after sorting.

Display data: In order for the program to make feasible sense to the user running it, there will need to be a UI output that not only prompts for the numbers, but also returns the sorted list of numbers after it's been sorted.

2.3 User Classes and Characteristics

Users involved in the development process are students with technical backgrounds. Each worked on a task like documentation, programming, and git management. All social/business aspects are managed by the whole team as a group, and not by one individual.

2.4 Operating Environment

The operating system used can vary depending on what the simulator requires. Typically, Windows, Mac, and Linux are all supported by the simulator, so running the application will be seamless. The program itself is developed on LC-3 Editor, and should be run in the LC-3 Console. Both of which should be at the latest versions respectively.

2.5 Design and Implementation Constraints

The application is limited to only receiving numerical data types. It will not properly function if fed something like a string or non-integer value. It's also limited to small lists of numbers (8 specifically), making it inconvenient for those seeking to sort greater sets of numbers.

2.6 Assumptions and Dependencies

The application itself is dependent on the LC-3 Simulator. It can be run on an LC-3 simulator on a browser, and there is no specific browser required, it simply has to be a LC-3 simulator.

3. External Interface Requirements

3.1 User Interfaces

The user interface will not require manipulation of the console from the user side. There will be a menu programmed in, which will prompt the user for the input, and will display an ASCII output.

3.2 Hardware Interfaces

Ideally, the application should be run on a computer. Running the LC-3 console and simulator is seamlessly done through the actual application, or through a web simulator.

3.3 Software Interfaces

To run the program, the .obj file will NEED to be run on the LC-3 Simulator. Otherwise, the other option is running it on a website simulator.

3.4 Communications Interface

There is no specific browser needed to run the program, even if the web simulator is being used to run it. Wifi will only be needed to obtain the program from the github repo, and if the user decides to run the program on a web simulator. Otherwise, the LC-3 simulator will run the program without a wifi connection.

4. Detailed Description of Functional requirements

4.1 Type of Requirement (summarize from Section 2.2)

The functions needed to successfully run the Bubble Sort program such that it's a pleasant experience for the user, while also performing its intended function are as such:

A display menu will need to be implemented, prompting users for numerical input 8 times to create a list of numbers to sort. There cannot be more than 8 numbers, and each of those numbers cannot exceed 100, or be lower than 0. The bubble sort algorithm itself will utilize inner and outer loops, conditional statements, and stack management to fetch and store sorted numbers. Data is stored internally, within the application.

4.2 Performance requirements

The nature of bubble sort is that it's a relatively slow program, running at O(n^2). This, along with the fact that the LC-3 program will only accept 8 numbers, makes the performance of the program

mediocre at best, and suited for small sets of numbers. The sorting manner itself is very trivial and easy to understand, which also makes it a confident algorithm, suitable for a quick sorting if need be.

4.3 Flowchart OR Pseudocode.

```
Pseudocode progression may be as such:

LD REGISTER TO HOLD ARRAY LENGTH

SWAP FUNCTION:

      LD REGISTER VALS TO BE SWAPPED

      STR VALS TO SWAP

      RETURN FROM SUBROUTINE (WHEN NEEDED)

START PROGRAM AT ORIG X3000

      LD ADDRESS OF ARRAY

      LD LOOP COUNTER

      LOOP FOR INPUT:

            DISPLAY INPUT PROMPT

            STORE NUMBER IN MEMORY

            INCREMENT UNTIL LOOP COUNTER 0 (OR 8 TBD)

            BRnzp UNTIL ALL DESIRED NUMBERS ARE ENTERED

      BUBBLE SORT ALGO:

            OUTER LOOP:

                  LD OUTER LOOP COUNTER

                  LD ARRAY ADDRESS

                  LD INNER LOOP COUNTER

                  LD SWAP FLAG

                  INNER LOOP:

                        IMPLEMENT SWAP FUNCTION FROM EARLIER
```

```
                    SET SWAP FLAG TO INDICATE SWAP

               FALSE SWAP:

                    IF NO SWAP NEEDED FROM INNER LOOP, JMP
HERE

               BRp FOR INNER LOOP

               BRnzp FOR OUTER LOOP



MAIN FUNCTION:

     LD NUMBERS

     LD LOOP COUNTER

     DISPLAY LOOP UNTIL ALL NUMBERS ARE DISPLAYED

     HALT

     FILL NUMBERS ARRAY WITH ADDRESSES

     END
```