Anthony Bisgood

CSC 252

ASM5 Report

Brief Overview

For this project I decided to make a sorting stopwatch. My code first takes in the number of inputs the user wants to sort, so if you want to sort 50 numbers you simply type in 50 and hit enter. It then asks you to input all the numbers you wish to sort. For this part just type in a number you wish to add to the sorting algorithm and press enter for each number. Once the number of numbers you have inputted equals the number of inputs you wanted to sort, it adds all the numbers to an array. Next, we run the bubble sort. For each step of the bubble sort, whenever 2 numbers swap places, it will print out the sort number that swap took place, and the array with that sort having taken place. After the bubble sort is run, it checks to see if the array is truly sorted, if it is, we will print that the sort is correct, or if it isn't, we'll print that the sort is wrong. After that we print out the final sorted array as well as the amount of time the sort took.

InDepth Explanation

When taking the initial user input, I used the syscall 5, which takes a user inputted integer and stores it into v0.

To take the rest of the numbers we wanted to sort, I ran the user input call through a loop, ending when the number of calls asking for a user input equaled the initial number asking for how many numbers we wanted to sort. This took place in the method "takeInput:". After the user inputted each number, I added that number to an array of space 400. Which means that we can sort a max of 100 numbers.

After taking all the numbers in we now run the sorting algorithm, but before that I took in the current system time using the syscall 30. This call returns a high and low order, but because sorting is relatively fast, we only needed to save the low order number.

After taking the system time I ran the bubble sort. Every time there was a sort, I printed out what number sort that was, as well as the current array of numbers. While this will affect the final time taken, I felt like this was a cool feature. To print the array, I made a "printSortedArray" method that would loop through the array and print each number.

After the sort ran, I used syscall 30 again to get the end time of the sort. Using the beginning and end times if we simply subtract, we can get the total amount of time the sort took in milliseconds.

Next, I ran through the array one more time to make sure all the numbers were in their correct places and returned the result of that. This was done by using a simple while loop with the length of the array.

Finally, I printed out the final sorted array and the total amount of time taken.

Closing

I thought this project would be cool for a number of reasons, firstly we can directly see what types of inputs would affect the runtime of a sorting algorithm, and compare their best- and worst-case times. Seeing the array change after each sort is also a fun concept to visualize. In addition, this program is easily expandable and later on I could add more sorting algorithms and visualize how they work and compare their runtimes.