Project 1A: Sorting Algorithms

Consequent

This is my reflection on the efforts I put forth and learnings I experienced during Project 1A. In Project 1A, we were given a scaffolding code in PLP and were asked to write three sorting algorithms. Two of them were bubble sort and insertion sort. The third one was a wild card (our choice).

My group was a 2 member team, so we decided to pick one algorithm each and if one of us completes first, he can start on the third one. I took up the bubble sort algorithm. The challenge was to either code the algorithms in least number of lines or to make them more efficient by minimizing the total instruction cycles for their execution. I chose to go down the efficiency path.

I first worked out the algorithm in pseudo code. Then I tried to find ways to optimize the logic. One tweak in the logic that, in my opinion, critically improved the performance was to use a flag. I used the flag to track if, in any given iteration of the loop, bubbling (swapping) of values happened or not. If no bubbling happened, then the control would simply break out of the loop and return the list obtained so far and this was the sorted list. This logic worked because, if there is no swapping or bubbling, then it implies that all the numbers are in their respective positions, i.e., list is sorted.

I then started implementing it in PLP. I felt this was fairly easy as we had already done some coding in PLP by this time in Activity 1. I did find one thing that helped in further reducing the required instruction cycles for the execution of bubble sort algorithm. It was to replace as many “nop” operations with other operations as possible. This made sure that independent operations were executed in the “nop” cycles that basically are stalling cycles.

Optimizing the algorithm at logic level (pseudo code) as well as implementation level (PLP) ensured that my code executed in least number of instruction cycles when compared to other submissions from our class.

Next I had a discussion with my team mate and got to know that he was still working on the insertion sort code. He did not need help on it but just some more time. So we decided that I will start on the next algorithm and he can join when he is done with his task.

For the wild card, I chose to implement selection sort. This is another in-place sorting algorithm. Again my aim was to make it efficient. So, I approached this problem in a similar manner by first working on the pseudo code and then implementing it in PLP.

The idea of flagging that worked with bubble sort, didn’t really work with selection sort. I am not sure if it can work but my approach failed. So I chose not to waste more time in optimizing the logic in Pseudo code and started implementing the logic in PLP. Here, I again applied my learning that “nop” operations can be replaced with other independent operations.

This algorithm was third best in the class in efficiency and I think I could have improved on it by working a bit more on the pseudo code itself. But it is not possible to make it as good as the best one as it was Heap sort and, as we all know, heap sort’s time complexity is much better than that of selection sort.

At the end of this all, I was happy with what I managed to complete and learn. I also feel that my coding on PLP has improved through this activity.