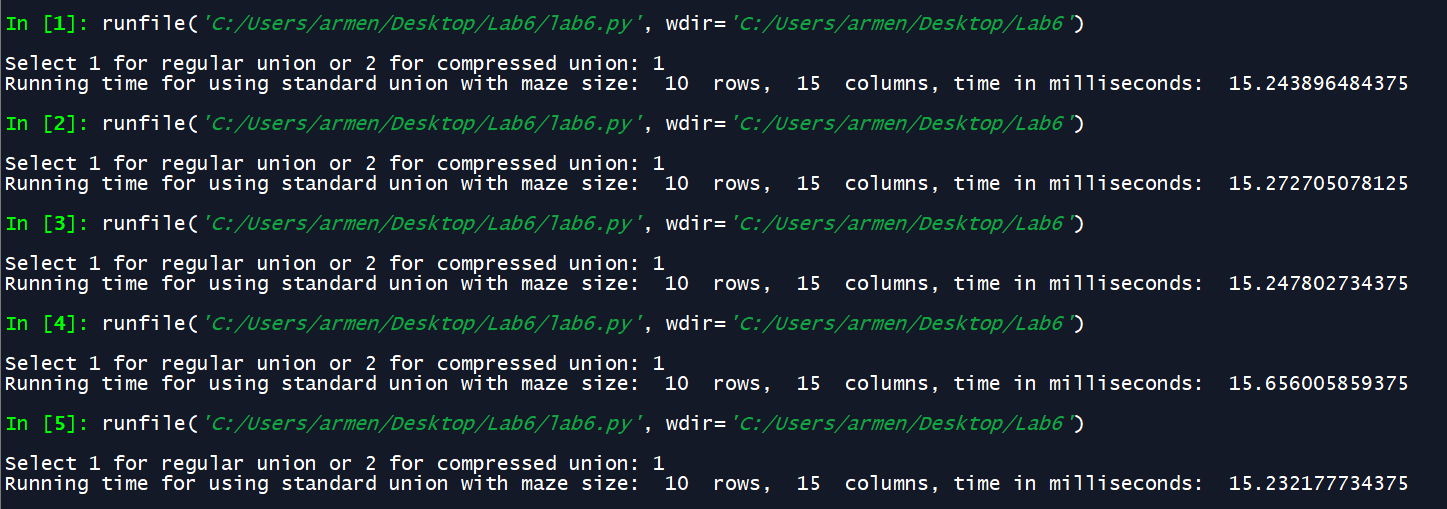
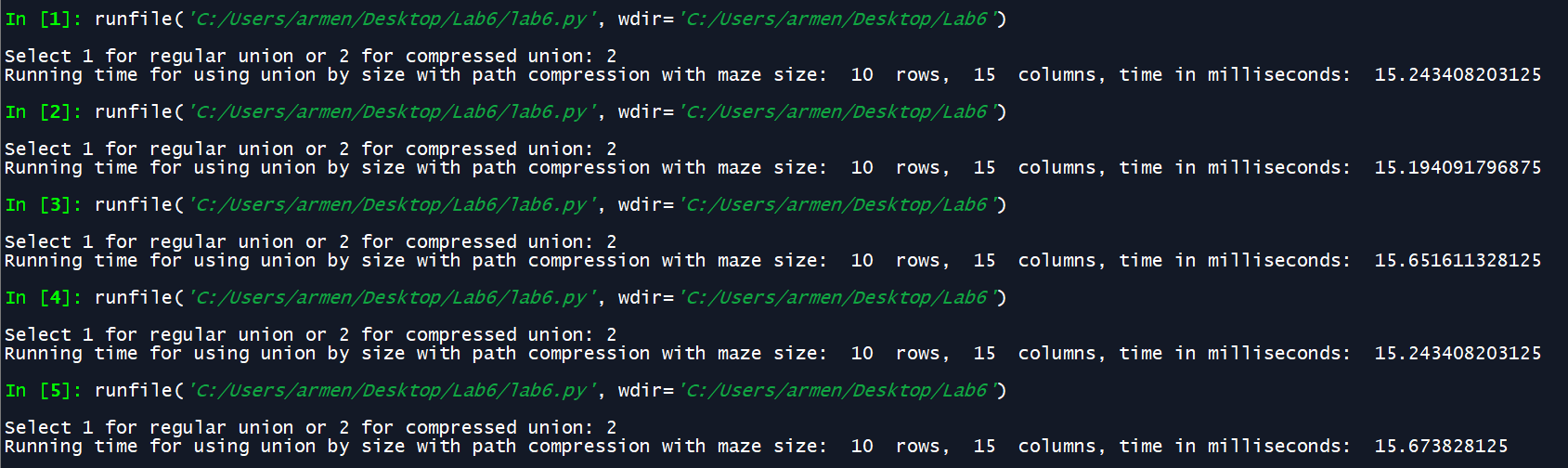
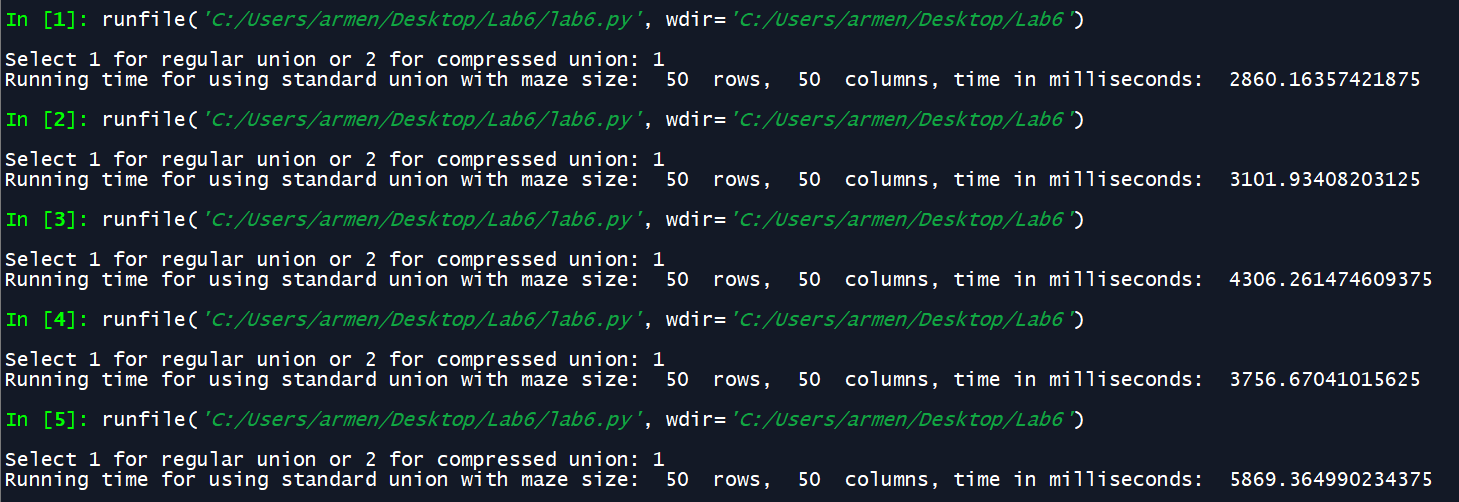
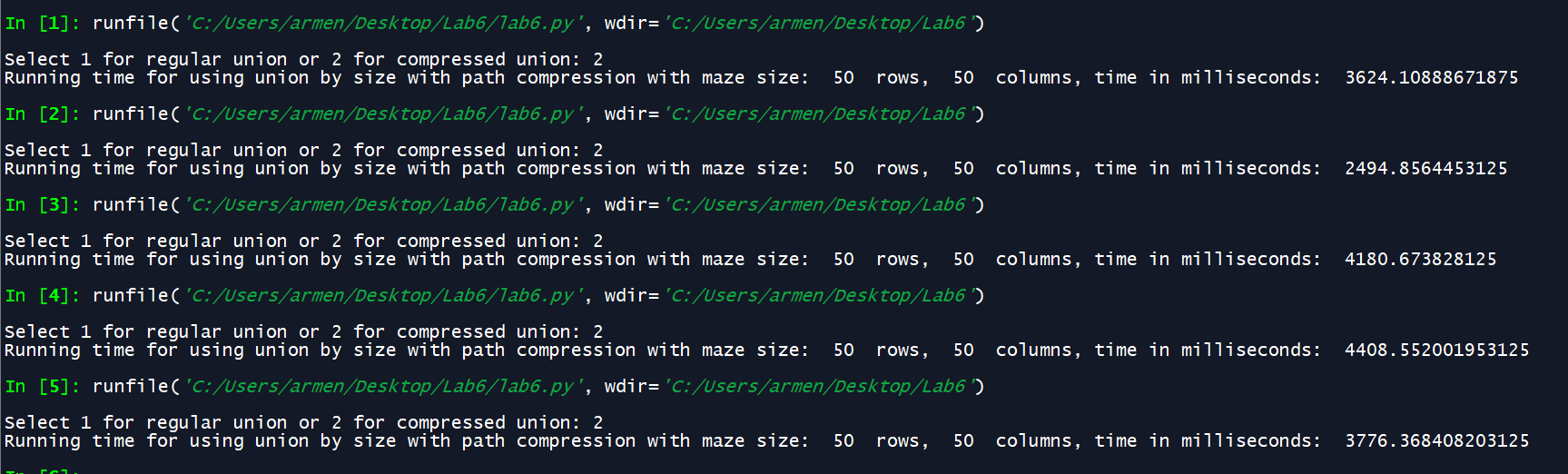
Lab 6 consisted of doing a maze based of cell numbers that removes randomly chosen walls. The chosen walls that will be removed from a cell must not repeat for each cell so that no more than one wall is removed at random. The maze is to have each cell assigned to a different set in a disjoint set forest that will be named M, for Maze. If there is more than one set in M, a wall will be removed at random by union or by compressed union. At the end there will be several comparisons made with rows of size 10, 50, and 75 with columns of size 15, 50, and 75 respectively to determine which union method is fastest.

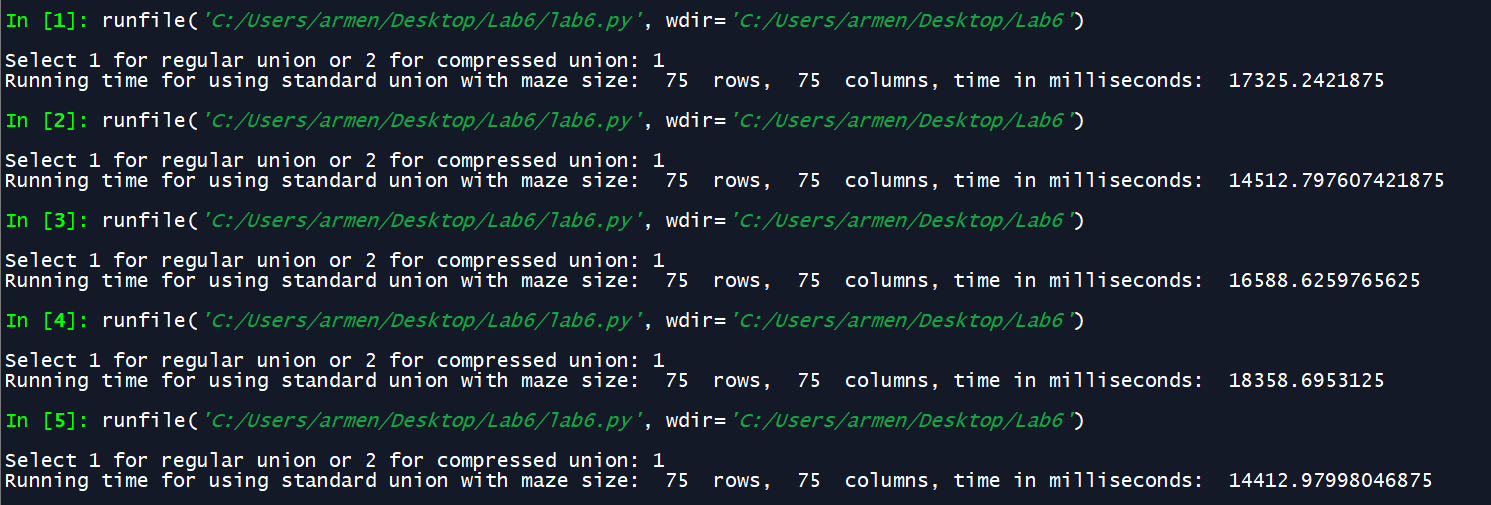
The initial code was provided by the professor which created the initial maze with all cells within it. Since the comparison was to be made with union and compressed union, functions for Disjoint Set Forest, (DSF), where added to the code. The walls for the cells where also provided by the professor so the maze\_rows and maze\_cols were used to create the DSF called M. I decided to add a question for the user to either click 1 for a standard union or 2 for a compressed union so that only one is ran in the console. Following the pseudocode provided in the instructions gave a clear idea on what was needed to remove, at random, one line at most for each cell. A while loop was used to traverse M if the number of sets in M are greater than one. Then a random wall was chosen at random by importing the random functions and assigning a random number from 0 to the length of the wall array called d. If statements were used to separate standard union and compressed union. Depending on which was chosen, another if statement was implemented to call either the method union() or union\_c, the latter is for the compressed union. Union and union\_c had to be modified and added a return True and return False so that if True is returned it doesn’t pop the random wall selected because one wall has already been removed from that specific cell number. When False is returned, that is when the wall from the selected cell number is removed because no walls from that cell have been removed. Finally, when the number of sets reach to only one, the while loop is stopped, and the maze is provided.

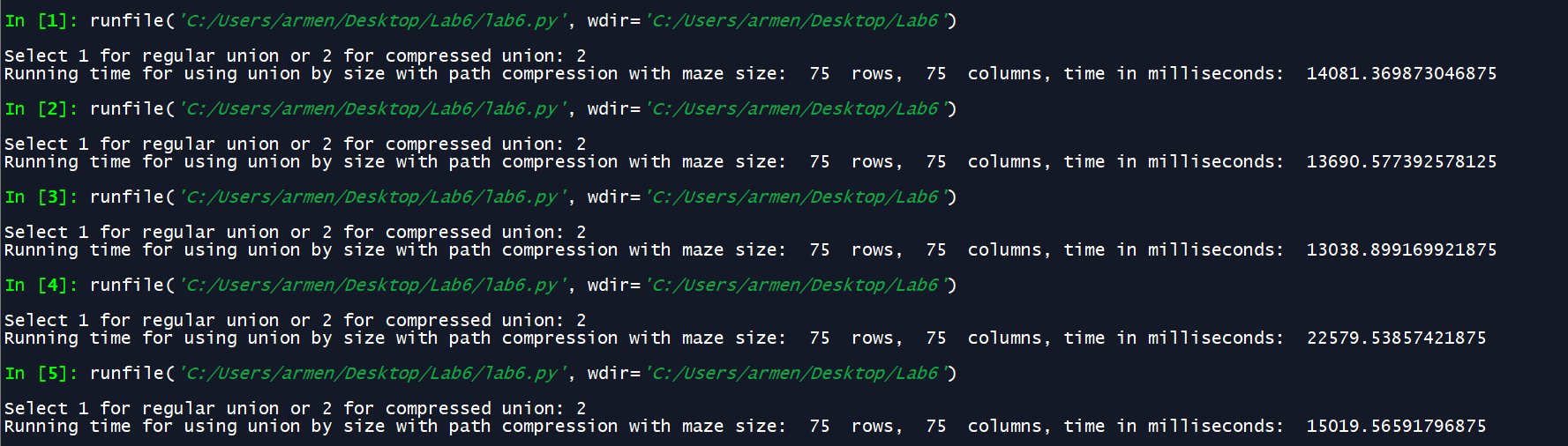
Time was imported to compare the time in milliseconds from the standard union function and the compressed union to build the maze. The following results were produced running each 5 times:











Based on the results, it appears that there isn’t much difference, especially when doing the comparisons of 10 rows and 15 columns. There is a slight edge in speed when doing the comparisons at 50 and at 75 for compressed union. The running time changed drastically when doing 50 by 50 because it took several seconds to complete rather than milliseconds from doing the 10 by 15. Nonetheless, the 75 x75 took the longest which was expected, and the compressed union did show to be a few seconds faster on average but did take quite long on trial 4. Due to such fluctuations, 5 trials were done so that a clear picture is done when comparting.

I did learn a lot on this project, especially when using DSF since I was able to do several comparisons using both the standard union and the compressed union. They are both similar but change when the find function is called. The both find functions return the root of the set sent but one finds the compressed path. For the creation of the maze with the use of DSF, the recursive equation would be T(n)=3T(n-1) + n. This means that the big O notation would be O(n)=3^n. This is due to the recursion used on the find functions found on the union functions. Both the standard union and the compressed union have the same O notation.

Appendix:

