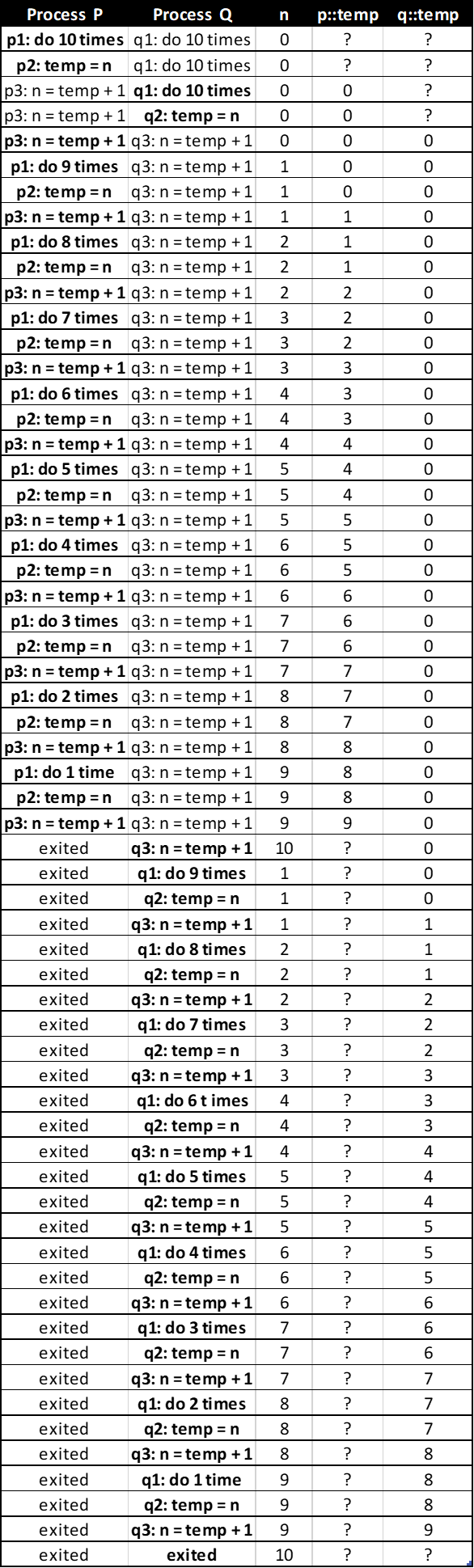
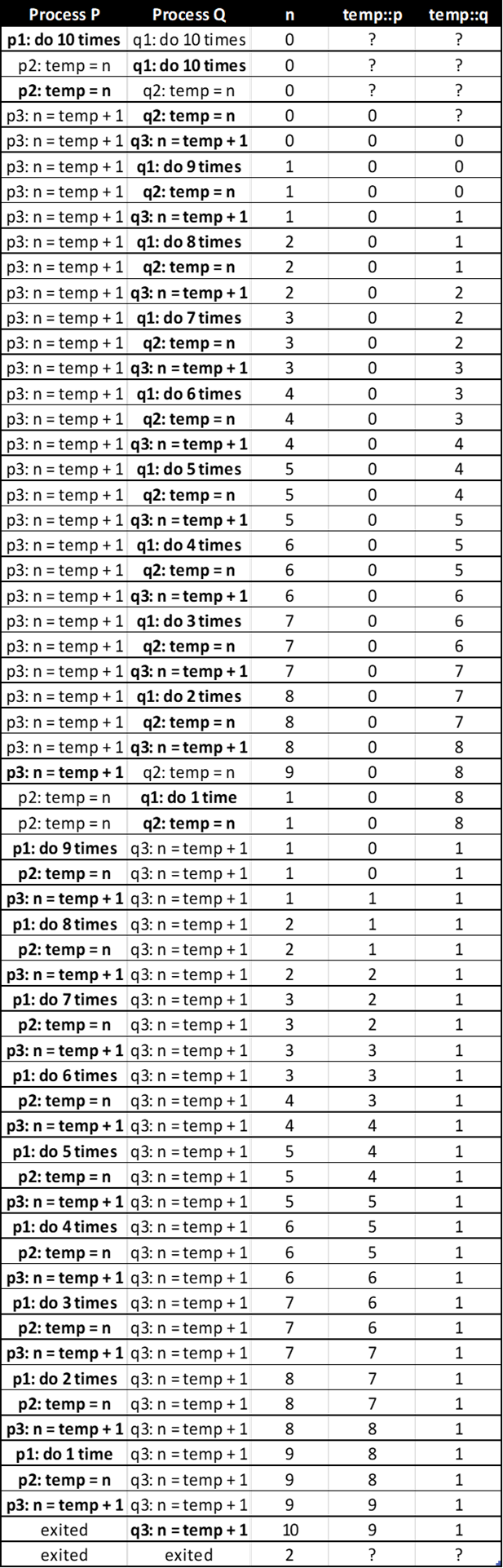
1a. This is the scenario table for 1a. It was straightforward. I just had to store the temp for either process for the other’s entire iteration.



1b. This is the scenario table to get a value of n=2 at the end. This was indeed a tricky one, but after realizing that I could just stall the process of P or Q until the very last 9th iteration, I could retain the minimum value of 0 for n for the longest time.

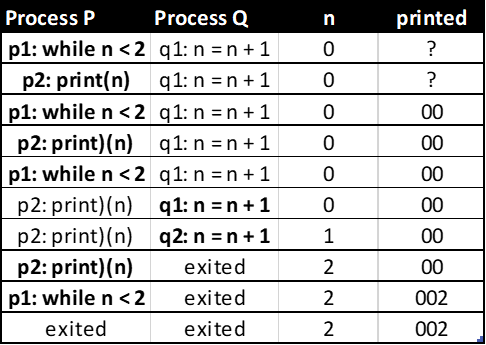
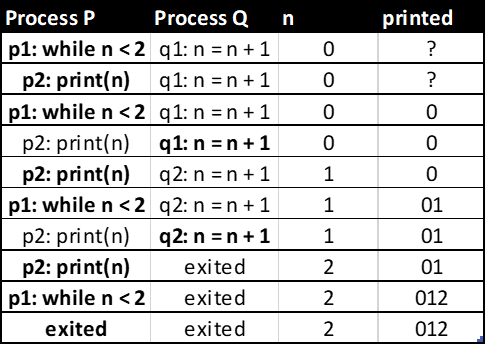


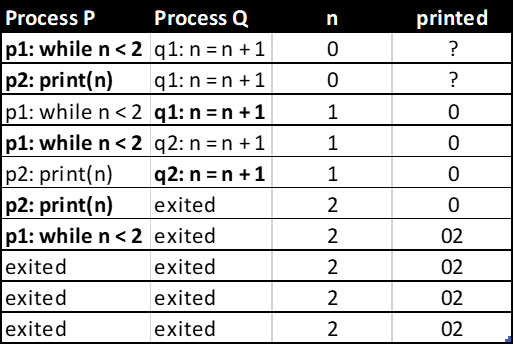
2a. -1, 0, 1

2b. -2, 2, -1, 0, 1

2c. I noticed that from 2a and 2b, the range tends to span between -K to K. Looking back at problem 1, if the 10 were K, then I think that the possible values for K there would be from 0 to 2K since both processes are adding. However, in problem 2, one process is adding and the other is subtracting. So, with all the interleaving and possible interleaves, there will be negative values and positive values. The maximum value that n can reach is K. The minimum value that n can reach is -K. Taking interleaving into account, n can be anywhere between -K to K. Therefore, the possible final value of n is -K to K.

3a.



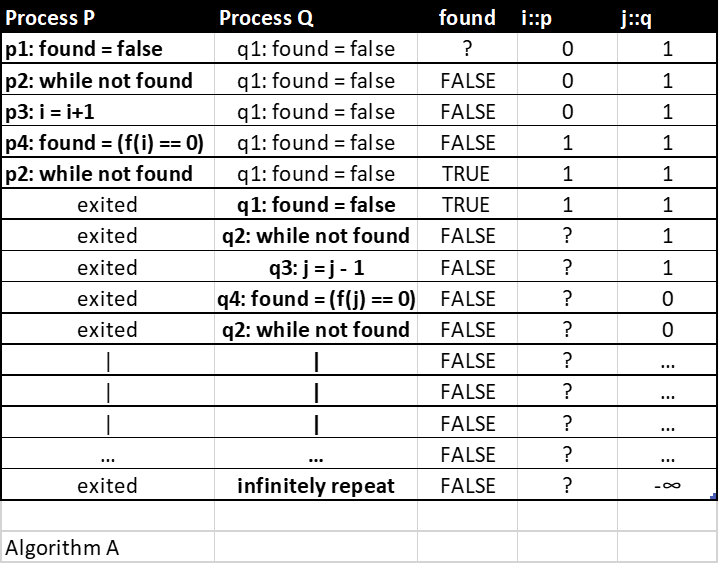


3b. The value 2 does NOT have to appear in the output. The scenario where this would happen is if process Q happens first before process P. In this case, since n is not less than 2, process P would terminate, and there would be no 2 in the printed output.

3c. The value 2 can only appear once in the output. After it appears once, n is already 2, so the while loop will terminate.

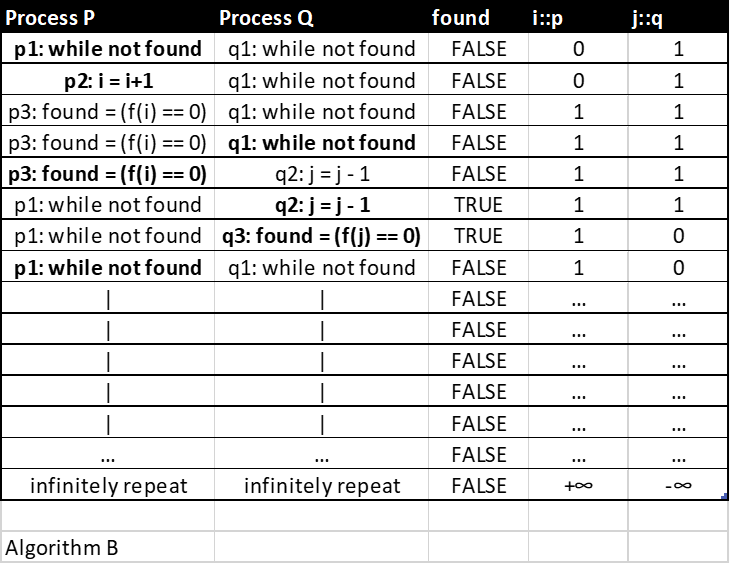
3d. The value 1 can appear an infinitely number of times because q2 can just stall indefinitely, which means the while loop’s condition continuously remains true until q2 is executed.

4a.



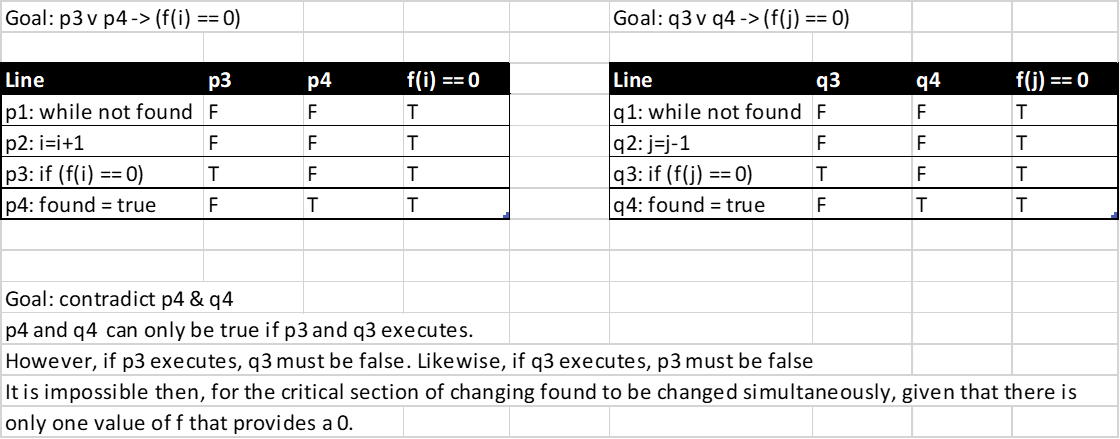
In algorithm A, either process P or process Q could infinitely repeat their while loops after the value has been found. The way this can happen is if one process runs the entirety of its course before the other process has a chance to start. Assume f(1) == 0.

4b.



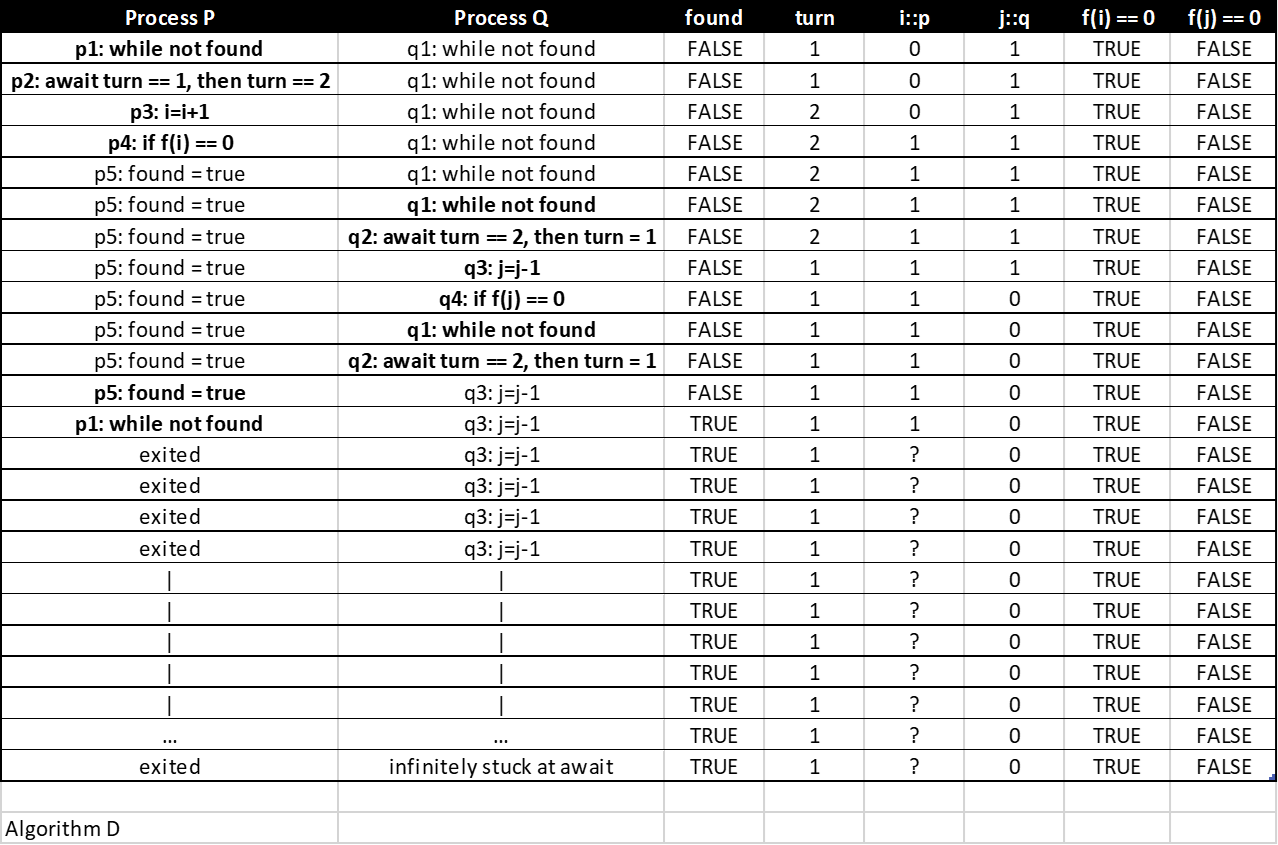
In algorithm B, it also does not work. The found variable can be overwritten once the value has been found, which results in a false output. As a result, both processes can infinitely repeat their loops. Assume f(1) == 0.

4c.



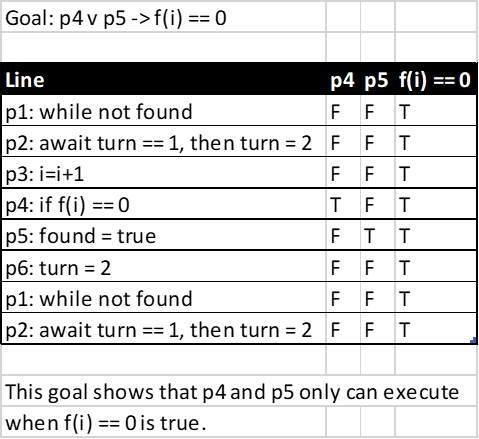
In algorithm C, the shared resource, found, is “protected” by an if statement. Assume that f(1) == 0. Normally, this is still unsafe. However, if we assume that there is only one value in function f which gives the value of 0, this works out as well. It insures that only one of the processes will ever touch the found variable and once found turns true, both processes will terminate. Another thing to note is context switching. If p3 were to execute and f(i) == 0, then any or all lines of process Q would be able to run, and it would not matter. The p4 would still eventually be able to execute, and vice versa if the value was a negative in process Q.

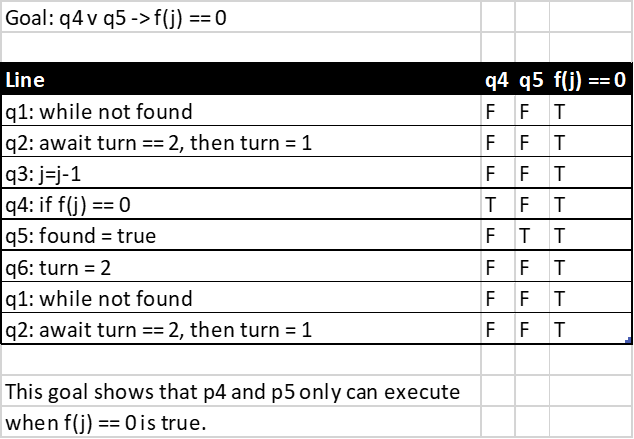
4d.

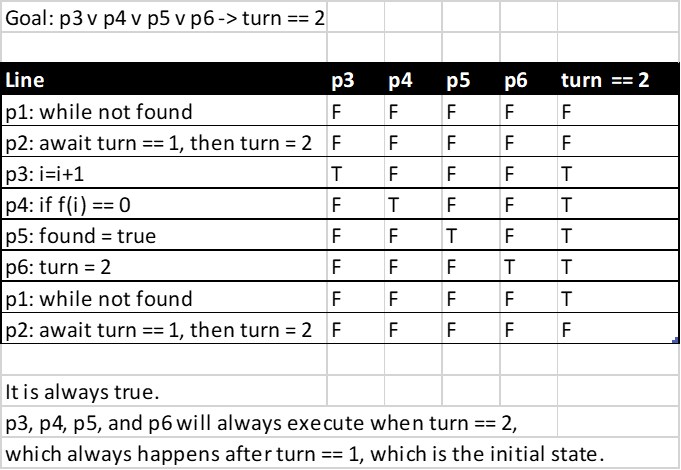


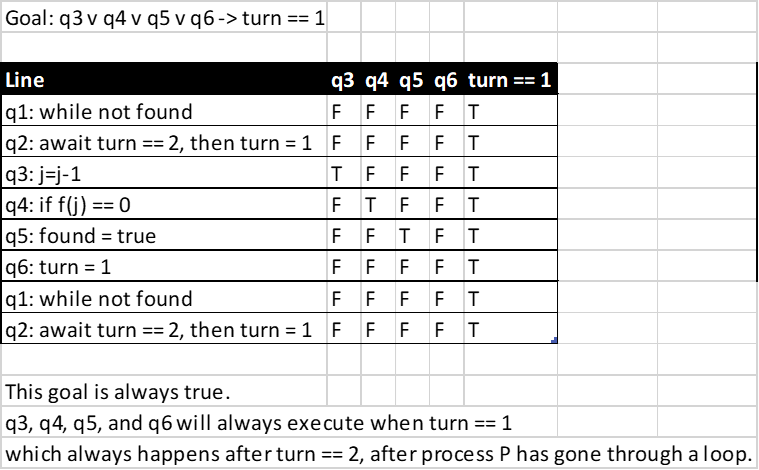
In algorithm D, process P or Q can run into a deadlock if found turns out to be true in one process after the other process has already entered their while loop and waiting for turn to change value. Also, assume that f(1) == 0.

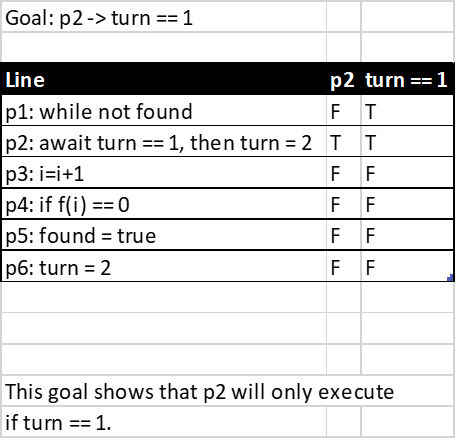
4e.

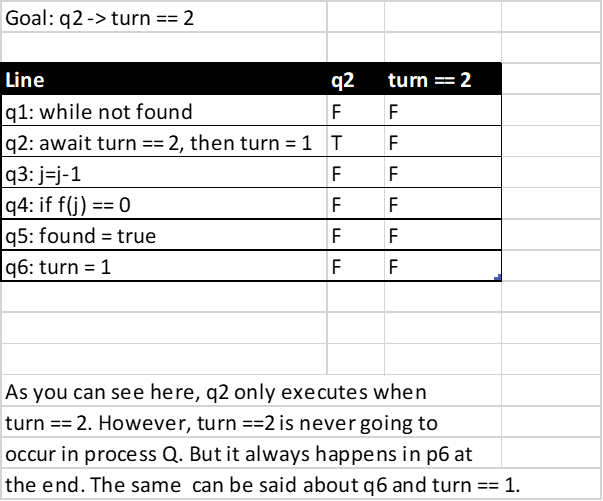












Deadlocking should not be a problem since the turns always are set at the end of the loops before leaving. Context switching should also not matter because with p4 and q4, there is only ever one case where q5 would be ran, and that is when the value is found, which is always only going to be one value.