1. (100 points) Problem 22-3 (Euler tour of a strongly connected, directed graph) on p.623, both parts (a) and (b).

Solution referenced from Elad Shahar.

**a)** Show that G has an Euler tour if and only if in-degree(v) = out-degree(v) for each vertex v ∈ V

We start by breaking this ”if and only if” problem into its two distinct parts.

**Part 1**: Euler tour exists ⇒ in-degree(v) = out-degree(v), ∀v ∈ V

Hint from Prof.:Decompose Euler tour into a set of edge-disjoint simple cycles, then use a construction proof (pick any vertex v such that in-degree = out-degree, then build a cycle from it. Keep in mind that G is strongly connected).

We start by noting that a Euler tour can be decomposed into a set of edge-disjoint simple cycles, that when combined form the tour. We now examine each ”sub-cycle.” We note that since they are simple edge-disjoint cycles, each vertex v in the cycle has one edge coming into it and one edge leading out of it. Therefore, in-degree(v)=out-degree(v) for each of the cycles. Now we observe the entire graph, and note that since an Euler tour exists, each simple cycle must be connected together, where each cycle has an edge coming in and an edge going out. We can therefore say that for each vertex v in the graph, in-degree(v)= out-degree(v).

**Part 2**: in-degree(v) = out-degree(v), ∀v ∈ V ⇒ Euler tour exists

Hint from Prof.: Use a constructive proof.

We choose to do this one via construction. We begin by figuring out how to build cycles. We start by examining some vertex v that has at least one edge leaving. We follow that edge to get to another vertex. Assuming we get to a vertex other than v, we can leave that vertex (since it has one edge going in, it must have some unvisited edge going out). We continue this pattern of following edges out until we end up back in v (which we are guaranteed to eventually end up in, since the graph is strongly connected, and v has at least one edge coming in to match its leaving edge). We start the proof by picking any vertex v and using the technique just mentioned to build a cycle that starts and ends at v. One we have observed this cycle, we search for any unvisited edges. We then go to a visited node w connected to said edge, and follow it
until we end up back in $w$. Again, since the graph is strongly connected and in-degree equals out-degree for every vertex, we are guaranteed that this will happen. We now simply add this new cycle into our existing cycle to make a longer cycle. When we examine every edge, we are in the situation that every edge has been visited exactly once. That is, we have found an Euler tour through the graph.

b) Describe an $O(E)$-time algorithm to find an Euler tour of $G$ if one exists. (Hint: Merge edge-disjoint cycles.) Hint from Prof.: Use (a) to show that degree examination decides if a Euler tour exists. Then use the constructive proof from (a) in the algorithm.

For the sake of the algorithm, we assume (similar to the book’s algorithms) that the graph is implemented as an adjacency list. We also assume that the algorithm is given a "copy" of the graph that can be deconstructed in the process of building the Euler tour.

**Pseudo-code**: BUILD-EULER-TOUR(G):

```plaintext
1 tour ← NEW-LIST()
2 vList ← NEW-DOUBLEVAL-LIST(G)
3 INSERT-INTO-DOUBLEVAL-LIST(vList, ANY-VERTEX(G), -1)
4 while NOT-EMPTY (vList): do
5   REMOVE-FROM-DOUBLEVAL-LIST(vList, v, ptr)
6   cycle ← NEW-LIST()
7   while out-degree(v) > 0 do
8     u ← ANY-UNVISITED-ADJACENT-VERTEX(V, v)
9     MARK-VISITED-ADJACENT(V, v, u)
10    out-degree(v) = out-degree(v) -1
11   ADD-TO-CYCLE(cycle, v)
12   if out-degree(v) > 0 then
13     INSERT-INTO-DOUBLEVAL-LIST(vList, v, LOCATION(v))
14   else v ← u
15   if ptr ≠ -1 then
16     tour ← cycle
17   else
18     ADD-CYCLE-TO-TOUR(tour, cycle, PREDECESSOR(ptr))
19 return tour
```

"As Advertised" Correctness

We take the approach that we did for Part 2 above. We start with a vertex $v$, and build a cycle from it. We then find additional cycles and add them into our main one, until all edges have been visited.

We start off [line 1] by initializing tour (the tour we are building) to a new list that we will use later. We then [line 2] initialize vList (our list of vertices to visit) to a list
that holds value pairs. We start if off [line 3] with any vertex in the graph, and the marker -1 to represent the fact it’s the first iteration. In the future, this second value will hold the location of the vertex in the graph.

Our while loop [line 4] keeps going as long as we have elements in our vList—that is, we have elements we still need to visit. The first line of our loop [line 5] removes the first element from the list, then immediately [line 6] starts a new cycle. The while loop [line 7 through 18] build more cycles and add them to the main one as we did in (a) above. [line 12] checks to see if there is another edge that we won’t visit in this cycle, and if so adds that vertex to vList so we examine it later. [line 15-16], which initializes the tour if we are on our first run through (based on the -1 marker). If it’s not our first run through [line 17 and 18], we add the new cycle to our existing tour right before the vertex where it branches off.

**Running Time Analysis**

Since every edge is visited at most once and at least once (exactly once), and all additional work done is constant, the algorithm runs in $\Theta(E)$ time.