1: Proving my HW 2 solution

Analysis:
The main idea is that for the reported value of any processor's bit - a majority consensus is easy to achieve regardless of whether it has its communication compromised or not. Once this is known, the algorithm must just guarantee triviality as well and it will be complete. This can be done by having multiple such consensuses and a consensus of these consensuses of some sorts. The algorithm presented below articulates this formally.

Algorithm:

Send your id to all processors ▷ Round 1
Send all the ids you received to all processors ▷ Round 2
Let x_0 = majority of the 3 bits that the other processors claimed was the bit of P_0
Let x_1 = majority of the 3 bits that the other processors claimed was the bit of P_1
Let x_2 = majority of the 3 bits that the other processors claimed was the bit of P_2
Output the majority of $x_0, x_1 and x_2$

Running time:
This algorithm terminates with an output in 2 rounds.

Correctness:
Henceforth the term byzantine processor is used to refer to the processor that is affected by the adversary and suffers a byzantine fault.

Assume P_0 is the byzantine processor, it can send any bit it wants to the other 3 processors but two of them must receive the same bit as a bit can take only two values (Pigeon-hole principle) Therefore, x_0 for P_0 = this bit (the one received by 2 processors) as it will get back the 3 messages it sent for its own bit but also since P_0 is the byzantine processor, $P_1, P_2 and P_3$ must relay information perfectly, they too must have the same 3 messages. (This is because of the passing of received bits in round 2).
Assume now P_0 is not the byzantine processor, $P_1, P_2 and P_3$ must receive the same bit but one of them may not convey this value accurately to another, however since P_0 and the other processor will faultlessly convey the correct bit the adversary's sabotage will not affect any processor from obtaining the correct bit.
Hence all processors will always agree on x_0 but by symmetry, it is easy to verify that this holds true in a similar manner for x_1 and x_2.
Therefore now that it is shown that the processors will come to a consensus, all that remains to show is that this algorithm doesn't break the trivial condition as well i.e. some processor must have had the output bit as its input. Note here using the previous explanation that when P_0 is not the byzantine processor, x_0 = the true bit of P_0. Since only one processor can be byzantine, at least 2/3 of $x_0, x_1 and x_2$ must be the true bit of their respective processors. There are now two cases: these 2 bits are the same in which case this will be the majority and thus the output for all processors - triviality is not broken; or they aren't the same in which case triviality is maintained regardless of which bit is output.

To conclude, since this algorithm shows that the processors come to a consensus while not breaking the trivial condition, it is a correct algorithm.

2: t+1 election in a t-resilient byzantine asynchronous system

Analysis:
It is interesting to note that processors do know from whom they receive a message therefore an adversary capable of creating Byzantine faults cannot have a processor 'pretend 'to be another processor. As a result even in the byzantine asynchronous model in the first round, the adversary can at most delete a message, any incorrect behavior in this round must be a deletion as it is meaningless to claim to be any other processor as the receiving processor would be able to trivially verify this (as it knows who is sending a message). Therefore in the first round the asynchronous byzantine model reduces to the asynchronous omission model and thus admits a solution to t+1 election t-resiliently in the trivial manner - by simply using the solution for t+1 election from the omission model (as it outputs in 1 round, each processor using the simplex to accurately know which value it must output).