CS 9868 Internet Algorithmics
Assignment 2: Graduate and Undergraduate Students
Due October 26

Zaid Albirawi
250626065

1. **CountProcessorsRing:**

   Algorithm high-level description: In the first round each process creates a message that contains its own id and sends it to its right neighbor. Each process will also initialize a counter at 1. For every round until the process receives its own message back it will increment the counter. Once the process receives its own message back it will terminate.

   Time complexity: The time complexity of the algorithm is $O(n)$ where n is the number of processes in the ring. This is because each message will need to be forwarded by each process in the ring and each process can only forward one message per round.

   Time complexity: The communication complexity of the algorithm is $O(n^2)$ where n is the number of processes in the ring. This is because each message will need to be forwarded by each process in the ring, therefore, each message is forwarded n times and since there are n messages then the number of messages forwarded over n rounds is n^2.

2. **LeaderElectionLine**:

   Algorithm high-level description: In the first round the leftmost process will send a message with its own ID to its right neighbor. The neighbor will then compare its own ID to the one received. The larger ID will be forwarded to the next right neighbor. Each process will compare its own ID against the one received from their left neighbor. Once the rightmost process receives that ID it will compare it to its own and determine the ID of the leader. Once it determines the ID of the leader it will set its own status to either "leader" or "not leader" and forward the ID of the leader to its left neighbor and terminate. Each process will receive the ID of the leader from their right neighbor, set their status, forward that message, and terminate. Once the leftmost process receives the ID of the leader, it will set its own status and terminate.

Termination Proof: The algorithm only terminates when a process sends a message to its left neighbor or when it receives a message from its right neighbor and it is the leftmost process in the network. The algorithm works by forwarding messages from the leftmost process to the rightmost process in the network. Those messages are processed by each of the processes. Therefore, each process will send a message to their left neighbor except the leftmost process. However, since the second leftmost process sends a message to its left neighbor, then, the leftmost process receives a message from its right neighbor, the second leftmost process, which allows it to terminate.

Correct Output Proof: In the first round, the highest ID in the network is set to be the ID of the leftmost process. This ID is forwarded to the process' right neighbor. The right neighbor will compare its own ID to the received ID and forward the higher ID to the next right neighbor. When a process in the network receives a message from its left neighbor. That message will contain the highest ID of all the processes that received that message before this process. Therefore, since every process in the network receives a message from their left neighbor then when the rightmost process in the network compares the ID received from its neighbors to its own ID, it will determine the highest ID in the network. Once that highest ID in the network is determined the rightmost process will forward a message with the ID of the highest process and each process will be able to determine if they are the leader or not.

Time complexity: The time complexity of the algorithm is $O(n)$ where n is the number of processes in the network. This is because each process forwards two message except for the leftmost and rightmost processes which only forward one each. Therefore, since only one message is forwarded per round then the algorithm needs $2 \times n - 2$ rounds to complete.

Time complexity: The communication complexity of the algorithm is $O(n)$ where n is the number of processes in the network. This is because each process forwards two message except for the leftmost and rightmost processes which only forward one each. Therefore, the algorithm forwards $2 \times n - 2$ messages.

3. **LeaderElectionMesh**:

   Algorithm high-level description: In the first round the leftmost process of every row will send a message with its own ID to its right neighbor. The neighbor will then compare its own ID to the one received. The larger ID will be forwarded to the next right neighbor. Each process will compare its own ID against the one received from their left neighbor. Once the rightmost process of each row receives that ID it will compare it to its own and determine the ID of the highest process in that row. Once this phase is completed all the active processes will suspend except for the top rightmost process. The top rightmost process will now send a message with the highest ID in the first row to its bottom neighbor. The neighbor will then compare the highest ID of its row to the one received. The larger ID will be forwarded to the next bottom neighbor. Each process will compare its row's highest ID against the one received from their upper neighbor. Once the bottom rightmost process receives that ID it will compare it to its row's highest ID and determine the ID of the highest process in the network. Once it determines the ID of the leader it will set its own status to either "leader" or "not leader" and forward the ID of the leader to its left and upper neighbors and terminate. Each process that receives a message from their bottom neighbor will forward that message to their left and upper neighbors. Once the top rightmost process receives a message from its bottom neighbor it will only forward a message to its left neighbor. Lastly, each process will receive the ID of the leader from their right neighbor or bottom neighbor, set their status, forward that message, and terminate. Once the top leftmost process receives the ID of the leader, it will set its own status and terminate.

   Termination Proof: The algorithm only terminates when a process sends a message to its left neighbor or when it receives a message from its right neighbor and it is the leftmost process in the row of the mesh network. The algorithm works by forwarding messages from the leftmost process to the rightmost process in each row of the network and then forwarding messages down and then back up the rightmost column in the network. Those messages are processed by each of the processes. Therefore, each process will send a message to their left neighbor except the leftmost process of each row. However, since the second leftmost process of each row sends a message to its left neighbor, then, the leftmost process of each row receives a message from its right neighbor, the second leftmost process in that row, which allows it to terminate.

Correct Output Proof: In the first round, the highest ID in each row is set to be the ID of the leftmost process of each row. This ID is forwarded to the process' right neighbor. The right neighbor will compare its own ID to the received ID and forward the higher ID to the next right neighbor. When a process in the network receives a message from its left neighbor. That message will contain the highest ID of all the processes that received that message before this process. Therefore, since every process in the row receives a message from their left neighbor then when the rightmost process in the row compares the ID received from its neighbors to its own ID, it will determine the highest ID in the row. Furthermore, since every rightmost process in each row know the highest ID in their row. The highest ID in the network is set to be the ID of the highest ID in the first row. This ID is forwarded to the process' bottom neighbor. The bottom neighbor will compare its row's highest ID to the received ID and forward the higher ID to the next bottom neighbor. When a process in the network receives a message from its upper neighbor. That message will contain the highest ID of all the processes that received that message before this process. Therefore, since every process in the rightmost column of the network receives a message from their upper neighbor then when the bottom rightmost process in the network compares the ID received from its neighbors to its own ID, it will determine the highest ID in the network. Once that highest ID in the network is determined the bottom rightmost process will forward a message with the ID of the highest process and each process will be able to determine if they are the leader or not.

Time complexity: The time complexity of the algorithm is $O(W + L)$ where W is the number of processes in each row of the network and L is the number of processes in each column of the network. This is because each process in each row forwards two message except for the leftmost and rightmost processes of each row which only forward one each; while each process in the last column forwards two message except for the top and bottom processes of that column which only forward one each. Therefore, since L messages are forwarded per round when processing the rows and one message is forwarded per round when processing the rightmost column then the algorithm needs $(2 \times W - 2) + 2 \times L - 2$ rounds to complete.

Time complexity: The time complexity of the algorithm is $O(W \times L)$ where W is the number of processes in each row of the network and L is the number of processes in each column of the network. This is because each process in each row forwards two message except for the leftmost and rightmost processes of each row which only forward one each; while each process in the last column forwards two message except for the top and bottom processes of that column which only forward one each. Therefore, the algorithm forwards $(2 \times W - 2) \times L + (2 \times L - 2)$ messages.