Distributed Systems Assignment Theoretical Part
s1119520

2.1

To prove that $a \to b$ if and only if $V(a) \le V(b)$ we need to show that both $a \to b$ if $V(a) \le V(b)$ and $V(a) \le V(b)$ if $a \to b$ holds.

a)

I will try to prove by contradiction that if $V(a) \le V(b)$ then $a \to b$. So instead of showing that a happened before b, I will try to show that a and b are two concurrent events (from processes p_i and p_j respectively, $p_i \neq p_j$). Note that $V(a) \le V(b)$ is true if and only if k^{th} entry of vector $V(a)$ is equal or less than k^{th} entry of vector $V(b)$ for all $k \in \{1,2,..n\}$ in the vector.
Now, suppose that process p_i vector $V(a)$ at event a has some value k in the i^{th} entry. Then, the only way process p_j vector $V(b)$ at event b can obtain a value for the i^{th} entry that is at least k is through some sequence of messages starting at process p_i and finally reaching process p_j (otherwise process p_j will not be able to update its i^{th} entry of the vector and will not have it being at least k). And here we arrive at contradiction since such a sequence of messages implies that events a and b are not concurrent. And therefore, since we showed that there must be some sequence of messages from event a to event b in order for $V(a) \le V(b)$ to be true that means event a definitely happened before event b.

b)

I will prove second part - if $a \to b$ then $V(a) \le V(b)$ - in two cases:

When events a and b are of the same process p_i then by definition of vector clocks if event a happened before event b the only different entry in vectors $V(a)$ and $V(b)$ is i^{th} entry since it is updated by the process when starting a new task (process increments i^{th} entry's value by one when starting a new task). Therefore, for process p_i, $V(b)[i] = V(a)[i] + k$, where k-1 is the number of events happened between events a and b.

When events a and b are events of different processes p_i and p_j then the only way it is possible to know that a happened before b is by having a sequence of messages between task a for process p_i and task b for process p_j. Therefore, once again by definition of vector clocks event a of process p_i is sending a message with entire $V(a)$ in such situation and event b of process p_j is receiving a message. On receiving a message at event b process p_j updates vector clock by taking max element by element $V(b)[k] = \max(V(b-1)[k], V(a)[k])$, for $k \in \{1,2,..n\}$ and adds 1 to $V(b)[j]$. Therefore, $V(b)$ will definitely be at least $V(a)$. The previous formula doesn't include the case then there are other events between a and b, however that would not change the answer since some event $c, a \to c \to b$ for process p_l would take the max element by element between p_l

previous event vector clock and $V(a)$. In other words, then time goes on no vector clock that is affected by $V(a)$ will be lower than $V(a)$.

2.2

So, we need to show that every process trying to enter its CS must eventually succeed.

Consider a case where a new request is made by process i. It sends a timestamped request (tsi, i) to all other nodes and enters (tsi, i) to its own queue q_i.

Now, for process i to enter CS: (1) the REQUEST (tsi, i) must be at the 'head' of q_i and (2) i must have received REPLY from all processes.

(1) will eventually happen, because if there is some other CS request from process j that arrived earlier than i^{th} request it will be removed from process i queue at the moment when a process i receives RELEASE message from process j. This message will be broadcasted by process j just after it exits from the CS which will definitely happen since no failures in processes exist. Since all requests are eventually removed from the queue i^{th} request will eventually reach the 'head' of the queue.

(2) will also eventually happen since all other processes sends REPLY message immediately after they receive REQUEST (tsi, i) message and since there are no failures in channels or processes they will all definitely REPLY eventually.

Therefore, since both (1) and (2) will eventually happen, process i will at some point enter CS.

2.3

Weighted diameter of this network is 2+2+2+1=7.
The path that realizes that diameter is $A \to C \to E \to G \to H$. There is no cheaper path between A and H.
The diameter of the unweighted network is 4 and there are few corresponding paths. One of them is $A \to C \to E \to G \to I$. There is no faster/cheaper way between A and I.

2.4

b)
I will be keeping all nodes added in MST in array P and edges in Q.

Step 1. Let's start at node 1 by initializing P = {1}, Q = {}.

Step 2. There are 3 possible nodes that can be added (2,4,5). However, the weight between nodes 1 and 2 is the smallest (20), so we add node 2 in to MST: P = {1,2}, Q = {$1 \to 2$}.

Step 3. Now, there are 4 possible nodes that can be added (3,4,5,6). Weight between nodes 2 and 3 is the smallest (7), so we add node 3 in to MST: P = {1,2,3}, Q = {$1 \to 2, 2 \to 3$}.

Step 4. Now, we have 3 possible nodes that can be added (4,5,6). Weight between nodes 2 and 6 is the smallest (9), so we add node 6 in to MST: P = {1,2,3,6}, Q = $\{1 \to 2, 2 \to 3, 2 \to 6\}$.

Step 5. There are 3 possible nodes that can be added at this step – (4,5,9). Weight between nodes 6 and 9 is the smallest (4), so we add node 9 in to MST: P = {1,2,3,6,9}, Q = $\{1 \to 2, 2 \to 3, 2 \to 6, 6 \to 9\}$.

Step 6. Now, there are 3 possible nodes that can be added – (4,5,8). Weight between nodes 6 and 5 is the smallest (11), so we add node 5 in to MST: P = {1,2,3,5,6,9}, Q = $\{1 \to 2, 2 \to 3, 2 \to 6, 6 \to 9, 6 \to 5\}$.

Step 7. At this step we have 3 nodes that can be added next – (4,7,8). Weight between nodes 5 and 7 is the smallest (12), so we add node 7 in to MST: P = {1,2,3,5,6,7,9}, Q = $\{1 \to 2, 2 \to 3, 2 \to 6, 6 \to 9, 6 \to 5, 5 \to 7\}$.

Step 8. There are only two nodes left (4,8) and one of them can be added at this step. Since the weight between nodes 5 and 8 is the smallest (15) we add node 8 in to MST: P = {1,2,3,5,6,7,8,9}, Q = $\{1 \to 2, 2 \to 3, 2 \to 6, 6 \to 9, 6 \to 5, 5 \to 7, 5 \to 8\}$.

Step 9. Only node 4 left not added into the MST. The smallest weight between node 4 and the rest of the MST is through node 5, thus we add the edge $5 \to 4$ and node 4 into MST: P = {1,2,3,4,5,6,7,8,9}, Q = $\{1 \to 2, 2 \to 3, 2 \to 6, 6 \to 9, 6 \to 5, 5 \to 7, 5 \to 8, 5 \to 4\}$. Since no more nodes can be added – the algorithm terminates with the final MST:
Q = $\{1 \to 2, 2 \to 3, 2 \to 6, 6 \to 9, 6 \to 5, 5 \to 7, 5 \to 8, 5 \to 4\}$