a) Are the posts displayed in FIFO, causal, and total order? Justify why.

1. Causal:
No, ya que para que este se cumpla es necesario que las acciones se sigan todas en orden, es decir que cuando se recibe por ejemplo el mensaje 5, se hayan recibido antes los previos. Esto no se cumple en esta versión, ya que vemos que muchas veces se reciben mensajes fuera de orden causal.

2. FIFO:
El orden FIFO tampoco se cumple. Esto lo podemos ver porque los mensajes aparecen desordenados incluso cuando provienen de un mismo proceso.

3. Total:
El orden total no se cumple ya que no todos los procesos reciben los mensajes en el mismo orden. Para serlo el mensaje recibido número 3 (por ejemplo) tendría que ser el mismo en todos los procesos.

En esta versión no se cumple ninguno de los órdenes, esto es porque no hacemos ningún tipo de control ni algoritmo como sí se hace en las siguientes versiones.

b) Are the posts displayed in FIFO, causal, and total order? Justify why
Los mensajes se muestran en orden FIFO, por tanto…
1. Causal:
En esta versión **sí** que se cumple el orden causal. Esto podemos afirmarlo ya que en todos los procesos, los mensajes que recibe de otro, siempre están ordenados de forma causal.
Por ejemplo, en P1 crear un tema y ser respondido primero por P2 y luego por P3, a pesar de que la primera respuesta tenga más latencia, siempre se hará *deliver* de esta primero.

Esto se cumple gracias al sistema de relojes virtuales (***vector clocks***) que hemos implementado en la versión, que se asegura de que los mensajes queden ordenados según cuando suceden, estableciendo así un orden de los mensajes correcto. Estos ayudan a introducir una relación de happened-before a los mensajes.

2. FIFO:
Esta vez los mensajes **sí** que se reciben de forma ordenada. Además teniendo en cuenta que se cumple el orden causal podemos afirmar que este también. Eso es porque un orden causal implica que también hay orden FIFO (ya que dos mensajes cualquiera del mismo proceso están relacionados por happened-before). Ahora, el parámetro jitter no influye en la pérdida de orden FIFO o causal gracias a los vector clocks y la queue ordenada para mantener los mensajes antes de ordenarlos.
3. Total:
No, ya que no hacemos ningún tipo de control total. Esto es porque cada proceso funciona de forma “independiente” y por tanto no todos reciben los mensajes en el mismo orden.
Los mensajes que son concurrentes (no ordenados por happened-before) pueden ser vistos por diferentes procesos en diferentes órdenes -> dos mensajes enviados en el mismo instante

(tiempo de vector clock) no se puede diferenciar quién ha ido antes o después debido a que no tienen relación de causalidad.

c) Are the posts displayed in FIFO, causal, and total order? Justify why.

1. Causal:

No se garantiza. El algoritmo ISIS usado prioriza el acuerdo global, pero no verifica las dependencias causales (como los relojes vectoriales).
No se ha establecido ninguna relación happened-before entre los mensajes. Es decir, es posible que un mensaje de respuesta reciba un número de secuencia acordado anterior al del mensaje original si las propuestas y acuerdos se entrelazan de forma específica, rompiendo el orden causal estricto.

2. FIFO:
No se cumple el orden FIFO, porque la posición de cada mensaje depende del jitter en el request con el cual llega a los procesos. Dos mensajes cualquiera, enviados por el mismo proceso, se entregan en un orden total arbitrario influido por Jitter.

3. Total:
Mecanismo del algoritmo:

- Cada mensaje se asocia con un número de secuencia único acordado por todos los nodos (seq:max/2 y proposal/3)
- Los nodos difunden propuestas, comparan valores y acuerdan el máximo (maxIncrement/2).
- Los mensajes se entregan en orden ascendente de secuencias.

Ejemplo:
Mensajes m1 (secuencia 5), m2 (secuencia 6), m3 (secuencia 7) se entregan en
5 → 6 → 7, independientemente del emisor.

! El algoritmo se basa en que todos los procesos acuerden un número de secuencia único y final para cada mensaje antes de entregarlo. Esto asegura que todos los mensajes entreguen todos los mensajes en el mismo orden global-total.

Sí que se cumple, debido a que todos los procesos reciben el mensaje que sus peers.
Los mensajes se entregan en orden acordado globalmente mediante propuestas y acuerdos de secuencias (seq: max/2 y proposal/3).

Esto se puede demostrar a partir de 2 propiedades:

1) El número de secuencia de un mensaje es único en el sistema e idéntico en todos los procesos. Si un proceso ya ha puesto un #sec S, en los siguientes que proponga será como mínimo a partir de S + 1.

2) Los mensajes se entregan en orden de #sec. Esta propiedad se cumple por la manera que ordena la hold-back queue.

d) Given a multicast system that is both totally-ordered (using an implementation like ours, which guarantees that the sequence number of any message sent is greater than that of any seen by the sender) and FIFOordered, justify whether it is also causally-ordered as a consequence.

Efectivament, sí assoliria ordenació causal. Si ho demostrem pels casos simples de la relació happened-before, els casos generals es deriven a partir d'aquí:

1. p envia m1 i després m2. Com que el multicast satisfà ordenació FIFO, tots els processos lliuraran m1 abans que m2
2. p envia m1, q lliura m1 i llavors envia m2. q proposará per m2 un nombre de seqüència més gran que el nombre acordat per m1, per tant el nombre acordat per m2 segur també que serà més gran, i tothom lliurarà m1 abans que m2

Un sistema multicast que es totalmente ordenado (con números de secuencia crecientes) y FIFO ordenado también es causalmente ordenado. El orden FIFO asegura que los mensajes de un mismo emisor respeten su causalidad, y el orden total con secuencias crecientes garantiza que si m1→m2 (incluso entre emisores), m1 tenga un número menor y se entregue antes que m2.

e) We have a lot of messages in the system. Derive a theoretical quantification of the number of messages needed to deliver a totally-ordered multicast message (from the sending by a worker to the delivery by all the workers) as a function of the number of workers.

Nº mensajes = 1 (send) + N (req) + N (proposal) + N (agreeed) + N (deliver) = 4N + 1 mensajes

f) Compare with the number of messages needed to deliver a multicast message in the basic implementation.

basic: Nº mensajes = 1 (send) + N (multicast) + N (deliver) = 2N + 1 mensajes
2*N (1 send + N-1 multicast + N deliver)

(4N + 1) - (2N + 1) = 4N - 2N + 1 - 1 = 2N → La diferencia de mensajes entre las dos implementaciones es de 2N, requiriendo el multicast total casi el doble de mensajes respecto a la implementación básica.