1.

a)

None of the schedules are conflict equivalent.

-First, S_1 can not be conflict equivalent to any other schedules, since the operations are different. ($R_2(x)$ in S1, $R_2(z)$ in the others).

-S_2 and S_3 is not conflict equivalent. The order of T2 and T3 are different because of the existence of following conflicting actions.

S_2: $R_3(y) \rightarrow W_2(y)$
S_3: $W_2(y) \rightarrow R_3(y)$

-S_2 and S_4 is not conflict equivalent because of the above conflicting actions as well.

-S_3 and S_4 is not conflict equivalent because of the following conflicting actions.

S_3: $R_3(x) \rightarrow W_1(x)$
S_4: $W_1(x) \rightarrow R_3(x)$

b)

-S_3 is conflict serializable to S_3'

$S_3 = R_3(z), W_2(x), W_2(y), R_1(x), R_3(x), R_2(z), R_3(y), W_1(x)$
$S_3' = W_2(x), W_2(y), R_2(z), R_3(z), R_3(x), R_3(y), R_1(x), W_1(x)$
$T_2 \rightarrow T_3 \rightarrow T_1$

-S_4 is conflict serializable to S_4'

$S_4 = R_2(z), W_2(x), R_3(z), W_1(x), W_2(y), R_1(x), R_3(x), R_3(y)$
$S_4 = R_2(z), W_2(x), W_2(y), W_1(x), R_1(x), R_3(z), R_3(x), R_3(y)$
$T_2 \rightarrow T_1 \rightarrow T_3$

c)

S_3 and S_4 are local serializable, but not global serializable, because the transactions of the equivalent serial schedules are in different order.

** If you modify the $R_2(x)$ in S1 into $R_2(z)$,

a) S_1 and S_4 become conflict equivalent.

-S_1 and S_2 is not conflict equivalent because of the following conflicting actions.

S_1: $W_2(x) \rightarrow W_1(x)$
S_2: $W_1(x) \rightarrow R_2(x)$

-S_1 and S_3 is not conflict equivalent because of the following conflicting actions.

S_1: $W_1(x) \rightarrow R_3(x)$
S_3: $R_3(x) \rightarrow W_1(x)$

b) S_1, S_3, S_4 are serilizable.

-S_1 is conflict serializable to S_1'

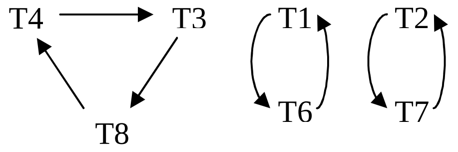
$S_1 = W_2(x), W_1(x), R_3(x), R_1(x), W_2(y), R_3(y), R_3(z), R_2(z)$
$S_1' = W_2(x), W_2(y), R_2(z), W_1(x), R_1(x), R_3(x), R_3(y), R_3(z)$
$T_2 \rightarrow T_1 \rightarrow T_3$

c) S_1 and S_4 are global serializable now.

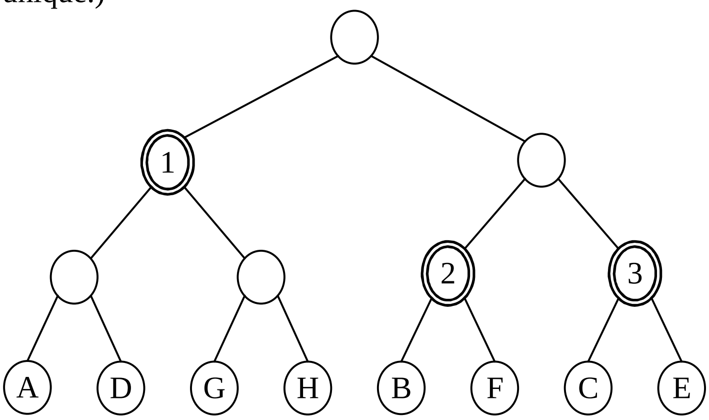
2.
a)

A	B	C	D	E	F	G	H
		W1					
	R7						
R4							
			W3				
						R8	
					R2		
						W3	
				W6			
					R5		
					W7		
		R6					
							W6
W8							
	W2						
			R4				
				W1			

The global WFG is:



b) The 8 nodes can be organized in the following tree structure. (The answer is not unique.)

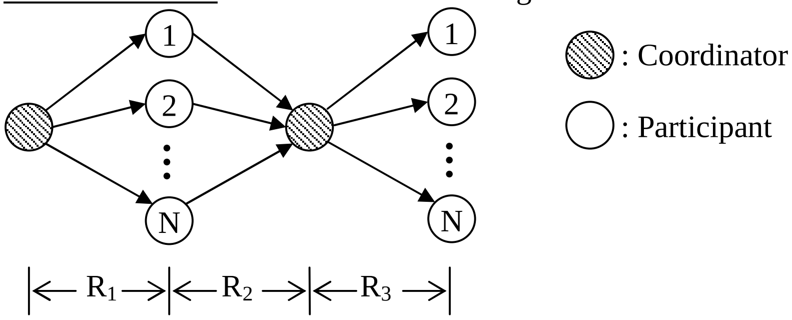


Each node sends its local WFG to the upper level node hierarchically. To minimize the data transmitted, we try to discover deadlocks in the bottom of the tree, as lower as possible. In the above tree, the deadlock caused by T1 and T6 can be detected right at their parent node 2. The deadlock of T2 and T7 can be detected at node 3. And the deadlock of T3, T4 and T8 is discovered at node 1. Since this deadlock involves three sites A, D and G, node 1 is the lowest node that the deadlock can be discovered.

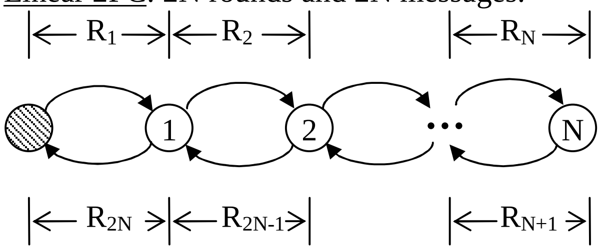
3.

a)

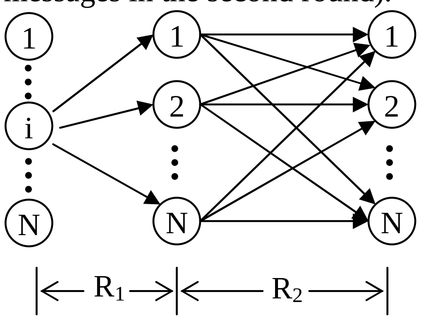
<u>Centralized 2PC:</u> 3 rounds and 3N messages.



<u>Linear 2PC</u>: 2N rounds and 2N messages.

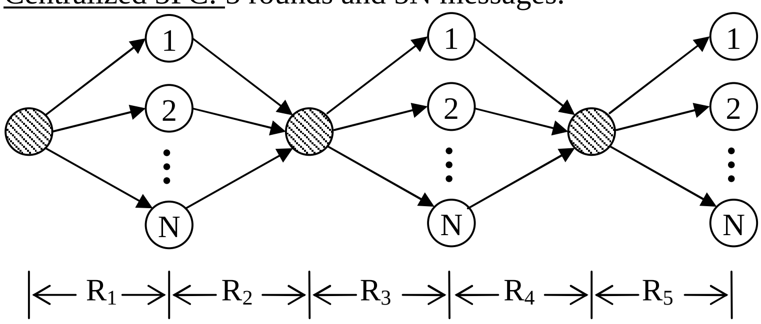


<u>Decentralized 2PC:</u> 2 rounds and N^2+N messages (N messages in the first round and N^2 messages in the second round).



b)

<u>Centralized 3PC:</u> 5 rounds and 5N messages.



c)

Consider the following case: After the coordinator received PREPARED message from all nodes, it starts to out PRECOMMIT to all participates. However, immediately after the coordinator sends out the first PRECOMMIT message, the coordinator fails. The participant who receives that PRECOMMIT message becomes the only node who is in PRECOMMIT state. Unfortunately, it fails too at this moment. After the rest of the nodes discover that the coordinator has failed, election protocol will be run, which is followed by the termination protocol in order to reach the consensus. The new coordinator asks all the other operational participants for their states. Since all alive nodes are in UNCERTAIN state. ABORT decision will be made.

After the failed participant recovered, since PRECOMIT state is not logged, it is now in UNCERTAIN state. Therefore, it will learn the ABORT decision from other nodes. So it will just follow accordingly and abort the transaction.

4.

a)

To minimize communication cost, two possible schemes are:

S1:

Node 1:	$R1 = R[1 \leqslant Y \leqslant 4000]$	16K
Node 2:	$R2 = R[4000 < Y \leqslant 12000]$	16K
	$R3 = R[12000 < Y \leqslant 16000]$	8K
Node 3:	$S1 = S[1 \leqslant Y \leqslant 4000]$	8K
	$S2 = S[4000 < Y \leqslant 12000]$	16K
Node 4:	$S3 = S[12000 < Y \leqslant 16000]$	16K

- Join R1, S1 at Node 1. Node 3 shifts S1 to Node 1. Cost = 8K
- Join R3, S3 at Node 4. Node 2 shifts R3 to Node 4. Cost = 8K
- Join between the remaining partitions R2, S2 and R3, S3 can perform at either Node 2 or Node 3. For example, Node 2 shifts R2, R3 to Node 3. Cost = 16K

Total data transmitted：32K

S2:

Node 1:	$R1 = R[1 \leqslant Y \leqslant 4000]$	16K
Node 2:	$R2 = R[4000 < Y \leqslant 8000]$	8K
	$R3 = R[8000 < Y \leqslant 12000]$	8K
	$R4 = R[12000 < Y \leqslant 16000]$	8K
Node 3:	$S1 = S[1 \leqslant Y \leqslant 4000]$	8K
	$S2 = S[4000 < Y \leqslant 8000]$	8K
	$S3 = S[8000 < Y \leqslant 12000]$	8K
Node 4:	$S4 = S[12000 < Y \leqslant 16000]$	16K

- Join R1, S1 at Node 1. Node 3 shifts S1 to Node 1. Cost = 8K
- Join R2, S2 at Node 2. Node 3 shifts S2 to Node 2. Cost = 8K
- Join R3, S3 at Node 3. Node 2 shifts R3 to Node 3. Cost = 8K
- Join R4, S4 at Node 4. Node 2 shifts R4 to Node 4. Cost = 8K

Total data transmitted：32K

b)

Supposing that the value of Y is uniformly distributed in each bucket, each Y value in [1, 4000] appears in four tuples in $R[1 \leqslant Y \leqslant 4000]$ and in two tuples in $S[1 \leqslant Y \leqslant 4000]$. Therefore, the result of join between $R[1 \leqslant Y \leqslant 4000]$ and $S[1 \leqslant Y \leqslant 4000]$ has tuples.

$$|R[1 \leqslant Y \leqslant 4000] \text{ join } S[1 \leqslant Y \leqslant 4000]| = 4 \times 2 \times 4000 = 32K$$

Similarly, we have

$$|R[4000 < Y \leqslant 12000] \text{ join } S[4000 < Y \leqslant 12000]| = 2 \times 2 \times 8000 = 32K$$

$$|R[12000 < Y \leqslant 16000] \text{ join } S[12000 < Y \leqslant 16000]| = 2 \times 4 \times 4000 = 32K$$

The total number of tuples in the join result is 96K.

To average the join results of nodes, **each node should have 24K tuples after join**. One possible scheme is:

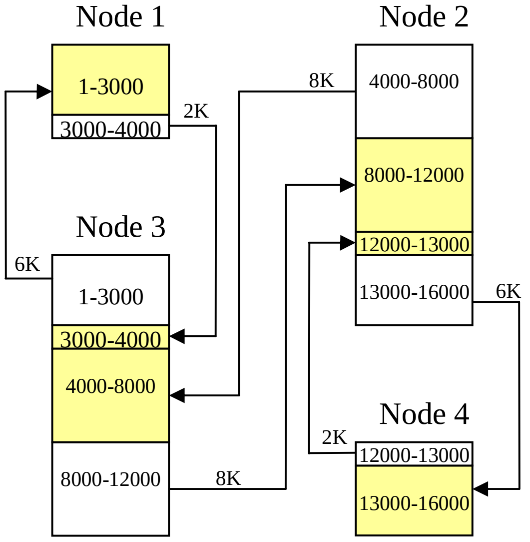
- <u>Node 1</u>:
  Join R[1≤Y≤3000] with S[1≤Y≤3000]
  Estimated results 4 x 2 x 3000 = 24K tuples
  Node 3 shifts S[1≤Y≤3000] to Node 1, Cost = 6K

- <u>Node 2</u>:
  Join R[12000<Y≤13000] with S[12000<Y≤13000]
  Estimated results 2 x 4 x 1000 = 8K tuples
  Node 4 shifts S[12000<Y≤13000] to Node 2, Cost = 4K
  Join R[8000<Y≤12000] with S[8000<Y≤12000]
  Estimated results 2 x 2 x 4000 = 16K tuples
  Node 3 shifts S[8000<Y≤12000] to Node 2, Cost = 8K

- <u>Node 3</u>:
  Join R[3000<Y≤4000] with S[3000<Y≤4000]
  Estimated results 4 x 2 x 1000 = 8K tuples
  Node 1 shifts R[3000<Y≤4000] to Node 3, Cost = 4K
  Join R[4000<Y≤8000] with S[4000<Y≤8000]
  Estimated results 2 x 2 x 4000 = 16K tuples
  Node 2 shifts R[4000<Y≤8000] to Node 3, Cost = 8K

- <u>Node 4</u>:
  Join R[13000<Y≤16000] with S[13000<Y≤16000]
  Estimated results 2 x 4 x 3000 = 24K tuples
  Node 2 shifts R[13000<Y≤16000] to Node 4, Cost = 6K

Total data transmitted：36K



c)For S1 in part a):

- Node 1: $R[1 \leqslant Y \leqslant 4000]$ join with $S[1 \leqslant Y \leqslant 4000]$
  R is the larger relation. So we first map S into the hash table and scan relation R. That is, for each tuple in R, we compare it with the tuples (of S) that have the same hash value. The hash table has 1000 entries and the hash function is random. Therefore, four Y values share the same entry in the hash table.
  - Number of tuples (of S) in each entry: 4x2 = 8
  - Number of tuples of R: 16K
  - **The number of comparisons is: 16K*8 = 128K**
- Node 2: $R[4000 < Y \leqslant 12000]$ join with $S[4000 < Y \leqslant 12000]$
  - Number of tuples in each entry: 8x2 = 16
  - Number of tuples of S: 16K
  - **The number of comparisons is: 16K*16 = 256K**
- Node 3: 0
- Node 4: $R[12000 < Y \leqslant 16000]$ join with $S[12000 < Y \leqslant 16000]$
  - Number of tuples (of R) in each entry: 4x2 = 8
  - Number of tuples of S: 16K
  - **The number of comparisons is: 16K*8 = 128K**

The total number of comparisons is: 512K

For S2 in part a):

- Node 1: $R[1 \leqslant Y \leqslant 4000]$ join with $S[1 \leqslant Y \leqslant 4000]$
  - **The number of comparisons is: 16K*8 = 128K**
- Node 2: $R[4000 < Y \leqslant 8000]$ join with $S[4000 < Y \leqslant 8000]$
  - **The number of comparisons is: 8K*8 = 64K**
- Node 3: $R[8000 < Y \leqslant 12000]$ join with $S[8000 < Y \leqslant 12000]$
  - **The number of comparisons is: 8K*8 = 64K**
- Node 4: $R[12000 < Y \leqslant 16000]$ join with $S[12000 < Y \leqslant 16000]$
  - **The number of comparisons is: 16K*8 = 128K**

The total number of comparisons is: 384K

For the scheme in part b):

- Node 1: $R[1 \leqslant Y \leqslant 3000]$ join with $S[1 \leqslant Y \leqslant 3000]$
  - **The number of comparisons is: 12K*6 = 72K**
  - (Number of tuples of S in each entry: 3x2, Number of tuples of R: 12K)
- Node 2: $R[8000 < Y \leqslant 13000]$ join with $S[8000 < Y \leqslant 13000]$
  - **The number of comparisons is: 12K*5*2 = 120K**
  - (Number of tuples of R in each entry: 5x2, Number of tuples of S: 12K)
- Node 3: $R[3000 < Y \leqslant 8000]$ join with $S[3000 < Y \leqslant 8000]$
  - **The number of comparisons is: 12K*5*2 = 120K**
  - (Number of tuples of S in each entry: 5x2, Number of tuples of R: 12K)
- Node 4: $R[13000 < Y \leqslant 16000]$ join with $S[13000 < Y \leqslant 16000]$
  - **The number of comparisons is: 12K*3*2 = 72K**
  - (Number of tuples of R in each entry: 3x2, Number of tuples of S: 12K)

The total number of comparisons is: 384K