1.
   a) None of the schedules are conflict equivalent.

   -First, S\textsubscript{1} cannot be conflict equivalent to any other schedules, since the operations are different. (R\textsubscript{2}(x) in S\textsubscript{1}, R\textsubscript{2}(z) in the others).

   -S\textsubscript{2} and S\textsubscript{3} is not conflict equivalent. The order of T\textsubscript{2} and T\textsubscript{3} are different because of the existence of following conflicting actions.
     \[
     \begin{align*}
     S_2: R_3(y) &\rightarrow W_2(y) \\
     S_3: W_2(y) &\rightarrow R_3(y)
     \end{align*}
     \]

   -S\textsubscript{2} and S\textsubscript{4} is not conflict equivalent because of the above conflicting actions as well.

   -S\textsubscript{3} and S\textsubscript{4} is not conflict equivalent because of the following conflicting actions.
     \[
     \begin{align*}
     S_3: R_3(x) &\rightarrow W_1(x) \\
     S_4: W_1(x) &\rightarrow R_3(x)
     \end{align*}
     \]

   b) -S\textsubscript{3} is conflict serializable to S\textsubscript{3}'
      \[
      \begin{align*}
      S_3 = & R_3(z), W_2(x), W_2(y), R_1(x), R_3(x), R_2(z), R_1(y), W_1(x) \\
      S_3' = & W_2(x), W_2(y), R_2(z), R_3(z), R_3(x), R_3(y), R_1(x), W_1(x) \\
      & T_2 \rightarrow T_3 \rightarrow T_1
      \end{align*}
      \]

   -S\textsubscript{4} is conflict serializable to S\textsubscript{4}'
     \[
     \begin{align*}
     S_4 = & R_2(z), W_2(x), R_3(z), W_1(x), W_2(y), R_1(x), R_3(x), R_3(y) \\
     S_4' = & R_2(z), W_2(x), W_2(y), W_1(x), R_1(x), R_3(z), R_3(x), R_3(y) \\
     & T_2 \rightarrow T_1 \rightarrow T_3
     \end{align*}
     \]

   c) S\textsubscript{3} and S\textsubscript{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\textsubscript{2}(x) in S\textsubscript{1} into R\textsubscript{2}(z),
   a) S\textsubscript{1} and S\textsubscript{4} become conflict equivalent.

   -S\textsubscript{1} and S\textsubscript{2} is not conflict equivalent because of the following conflicting actions.
     \[
     \begin{align*}
     S_1: & W_2(x) \rightarrow W_1(x) \\
     S_2: & W_1(x) \rightarrow R_2(x)
     \end{align*}
     \]

   -S\textsubscript{1} and S\textsubscript{3} is not conflict equivalent because of the following conflicting actions.
     \[
     \begin{align*}
     S_1: & W_1(x) \rightarrow R_3(x) \\
     S_3: & R_3(x) \rightarrow W_1(x)
     \end{align*}
     \]

   b) S\textsubscript{1}, S\textsubscript{3}, S\textsubscript{4} are serializable.

   -S\textsubscript{1} is conflict serializable to S\textsubscript{1}'
     \[
     \begin{align*}
     S_1 = & W_2(x), W_1(x), R_3(x), R_1(x), W_2(y), R_3(y), R_2(z) \\
     S_1' = & W_2(x), W_2(y), R_2(z), W_1(x), R_1(x), R_3(x), R_3(y), R_3(z) \\
     & T_2 \rightarrow T_1 \rightarrow T_3
     \end{align*}
     \]

   c) S\textsubscript{1} and S\textsubscript{4} are global serializable now.
The global WFG is:

```
T4  T3
  
T8  T6
  
T1  T2
```

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

```
  1
 / \
A   D
 / \
G   H
 / \
B   F
 / \
C   E
```

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) Centralized 2PC: 3 rounds and 3N messages.

b) Centralized 3PC: 5 rounds and 5N messages.

---

Linear 2PC: 2N rounds and 2N messages.

Decentralized 2PC: 2 rounds and $N^2+N$ messages (N messages in the first round and $N^2$ messages in the second round).
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 PRECOMMIT 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 ≤ Y ≤ 4000] 16K
Node 2: R2 = R[4000 < Y ≤ 12000] 16K
Node 3: S1 = S[1 ≤ Y ≤ 4000] 8K
Node 4: S3 = S[12000 < Y ≤ 16000] 16K

- Join R1, S1 at Node 1. Node 3 shifts S1 to Node 1. 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 ≤ Y ≤ 4000] 16K
Node 2: R2 = R[4000 < Y ≤ 8000] 8K
Node 3: S1 = S[1 ≤ Y ≤ 4000] 8K
Node 4: S4 = S[12000 < Y ≤ 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

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 ≤ Y ≤ 4000] and in two tuples in S[1 ≤ Y ≤ 4000]. Therefore, the result of join between R[1 ≤ Y ≤ 4000] and S[1 ≤ Y ≤ 4000] has tuples.

|R[1 ≤ Y ≤ 4000] join S[1 ≤ Y ≤ 4000]| = 4 x 2 x 4000 = 32K

Similarly, we have

|R[4000 < Y ≤ 12000] join S[4000 < Y ≤ 12000]| = 2 x 2 x 8000 = 32K
|R[12000 < Y ≤ 16000] join S[12000 < Y ≤ 16000]| = 2 x 4 x 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:

- **Node 1:**
  Join $R[1 \leq Y \leq 3000]$ with $S[1 \leq Y \leq 3000]$
  Estimated results $4 \times 2 \times 3000 = 24K$ tuples
  Node 3 shifts $S[1 \leq Y \leq 3000]$ to Node 1, Cost = 6K

- **Node 2:**
  Join $R[12000 < Y \leq 13000]$ with $S[12000 < Y \leq 13000]$
  Estimated results $2 \times 4 \times 1000 = 8K$ tuples
  Node 4 shifts $S[12000 < Y \leq 13000]$ to Node 2, Cost = 4K
  Join $R[8000 < Y \leq 12000]$ with $S[8000 < Y \leq 12000]$
  Estimated results $2 \times 2 \times 4000 = 16K$ tuples
  Node 3 shifts $S[8000 < Y \leq 12000]$ to Node 2, Cost = 8K

- **Node 3:**
  Join $R[3000 < Y \leq 4000]$ with $S[3000 < Y \leq 4000]$
  Estimated results $4 \times 2 \times 1000 = 8K$ tuples
  Node 1 shifts $R[3000 < Y \leq 4000]$ to Node 3, Cost = 4K
  Join $R[4000 < Y \leq 8000]$ with $S[4000 < Y \leq 8000]$
  Estimated results $2 \times 2 \times 4000 = 16K$ tuples
  Node 2 shifts $R[4000 < Y \leq 8000]$ to Node 3, Cost = 8K

- **Node 4:**
  Join $R[13000 < Y \leq 16000]$ with $S[13000 < Y \leq 16000]$
  Estimated results $2 \times 4 \times 3000 = 24K$ tuples
  Node 2 shifts $R[13000 < Y \leq 16000]$ to Node 4, Cost = 6K

Total data transmitted: 36K
c) For S1 in part a):

- **Node 1**: $R[1 \leq Y \leq 4000]$ join with $S[1 \leq Y \leq 4000]$
  
  $R$ 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: $4 \times 2 = 8$
  
  Number of tuples of $R$: 16K
  
  **The number of comparisons is**: $16K \times 8 = 128K$

- **Node 2**: $R[4000 < Y \leq 12000]$ join with $S[4000 < Y \leq 12000]$
  
  Number of tuples in each entry: $8 \times 2 = 16$
  
  Number of tuples of $S$: 16K
  
  **The number of comparisons is**: $16K \times 16 = 256K$

- **Node 3**: 0

- **Node 4**: $R[12000 < Y \leq 16000]$ join with $S[12000 < Y \leq 16000]$
  
  Number of tuples (of $R$) in each entry: $4 \times 2 = 8$
  
  Number of tuples of $S$: 16K
  
  **The number of comparisons is**: $16K \times 8 = 128K$

The total number of comparisons is: 512K

For S2 in part a):

- **Node 1**: $R[1 \leq Y \leq 4000]$ join with $S[1 \leq Y \leq 4000]$
  
  **The number of comparisons is**: $16K \times 8 = 128K$

- **Node 2**: $R[4000 < Y \leq 12000]$ join with $S[4000 < Y \leq 12000]$
  
  **The number of comparisons is**: $8K \times 8 = 64K$

- **Node 3**: 0

- **Node 4**: $R[12000 < Y \leq 16000]$ join with $S[12000 < Y \leq 16000]$
  
  **The number of comparisons is**: $16K \times 8 = 128K$

The total number of comparisons is: 384K

For the scheme in part b):

- **Node 1**: $R[1 \leq Y \leq 3000]$ join with $S[1 \leq Y \leq 3000]$
  
  **The number of comparisons is**: $12K \times 6 = 72K$
  
  (Number of tuples of $S$ in each entry: $3 \times 2$, Number of tuples of $R$: 12K)

- **Node 2**: $R[8000 < Y \leq 13000]$ join with $S[8000 < Y \leq 13000]$
  
  **The number of comparisons is**: $12K \times 5 \times 2 = 120K$
  
  (Number of tuples of $R$ in each entry: $5 \times 2$, Number of tuples of $S$: 12K)

- **Node 3**: 0

- **Node 4**: $R[13000 < Y \leq 16000]$ join with $S[13000 < Y \leq 16000]$
  
  **The number of comparisons is**: $12K \times 3 \times 2 = 72K$
  
  (Number of tuples of $R$ in each entry: $3 \times 2$, Number of tuples of $S$: 12K)

The total number of comparisons is: 384K