6.033 Computer System Engineering

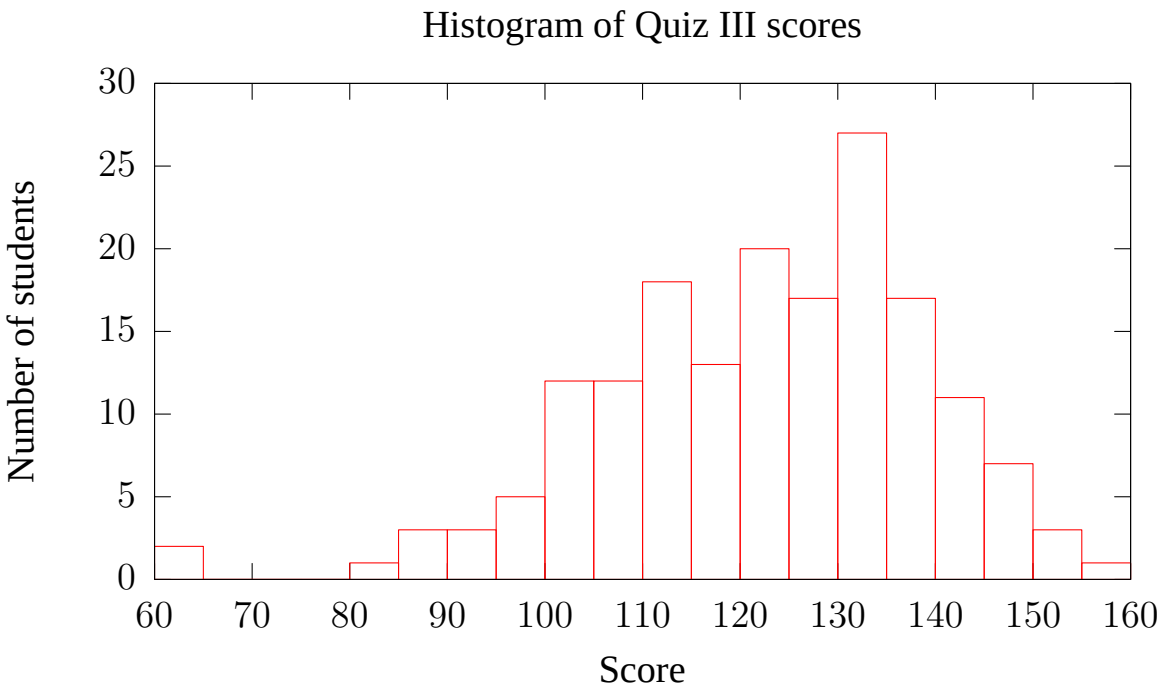
Spring 2009

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2006

Quiz III Solutions



$N = 172$

MEDIAN $= 123/160 = 76.9\%$

MEAN $= 121.6/160 = 76.0\%$

STDDEV $= 16.81/160 = 10.5\%$

I Reading Questions

1. [16 points]: Both ARIES (reading # 14) and the Log-Structured File System (LFS, reading #13) use a log to recover in the event of a system crash. Which of the following statements about these systems are true?

(Circle True or False for each choice.)

A. True / False The REDO phase of recovery in ARIES begins at the location of the last checkpoint.

FALSE. *Aries REDO begins at the location determined in the analysis phase.*

B. True / False The Roll Forward phase of recovery in LFS begins at the location of the last checkpoint.

TRUE.

C. True / False The state of the system in ARIES after recovery has completed reflects the effects of any transaction that committed prior to the crash.

TRUE.

D. True / False The state of the system in LFS after recovery has completed reflects the effects of all invocations of WRITE that returned prior to the crash.

FALSE. *LFS does not force the log to disk after every write, so some writes may disappear after a crash.*

E. True / False The portion of the log used during recovery can be truncated (that is, discarded) after successful completion of recovery in ARIES.

TRUE.

F. True / False The portion of the log used during recovery can be truncated (that is, discarded) after successful completion of recovery in LFS.

FALSE. *The log cannot be truncated in LFS because it is the only copy of the data.*

G. True / False In ARIES, logging ensures the atomicity of updates to database records in the face of system crashes.

TRUE.

H. True / False In LFS, logging ensures the atomicity of updates to files in the face of system crashes.

FALSE. *LFS does not ensure atomicity of writes; for example, a buffer written may span multiple segments.*

2. [8 points]: In the "Reflections on Trusting Trust" paper (reading # 17), which of the following statements about the Trojan horse that Ken Thompson inserted into the C compiler are true?

(Circle True or False for each choice.)

A. True / False The Trojan horse could not be detected by looking at the source code of the compiler.

TRUE.

B. True / False The Trojan horse could not be detected by disassembling the compiler binary.

FALSE. *The Trojan horse would be visible in the assembly of the compiler.*

C. True / False The C compiler with the code for the Trojan horse inserted could not be compiled by an unmodified version of the compiler.

FALSE. *An unmodified C compiler works just fine.*

D. True / False The technique used to insert the Trojan horse in the C compiler could not have been used for a Java compiler.

FALSE. *Java is equally susceptible to this attack.*

3. [8 points]: Which of the following techniques are likely to reduce the susceptibility of a program to a buffer overrun exploit?

(Circle True or False for each choice.)

A. True / False Writing a program in Java instead of C.

TRUE. *Java is much less vulnerable to buffer overruns because it has array bounds checking.*

B. True / False Using C, but never allocating arrays on the stack.

TRUE. *Having no arrays on the stack makes overwriting the stack much harder.*

C. True / False Disallowing execution of code stored on the stack.

TRUE. *Most buffer overrun vulnerabilities rely on being able to write executable code into the stack.*

D. True / False Making the start (bottom) of the stack begin at a random location in memory.

TRUE. *Starting the stack at a random location makes it difficult for an attacker to determine what return address to jump to.*

Name:

4. [8 points]: Which of the following is a true statement about the Witty paper (reading #20)?
(Circle True or False for each choice.)

A. True / False In many cases, a single Witty packet reveals enough information to recover the state of the pseudo random number generator (PRNG) used by the infected machine that sent the packet.

TRUE.

B. True / False The authors were able to detect Patient Zero because it picked the IP address of its destinations differently from other Witty infectees.

TRUE.

C. True / False The use of a hit-list helped Witty to spread quickly.

TRUE.

D. True / False All traffic received by a network telescope is worm traffic.

FALSE. *The network telescope receives traffic from a variety of sources.*

5. [4 points]: The RAID paper ("A case for redundant arrays of inexpensive disks (RAID)" by Patterson et al., Proceedings of the ACM SIGMOD Conference, 1988) defines a failure in an array as follows: the array is said to experience a failure if *any* disk in the array fails. Given this definition of failure, the Mean Time to Failure (MTTF) of an array, compared to the MTTF of a single disk in the array, is:

(Circle the BEST answer)

A. smaller

This is the correct answer. The MTTF of an array goes down because the probability that any one *of the several disks in the array fails is greater than the probability that a single disk will fail.*

B. equal

C. larger

Name:

6. [6 points]: The RAID paper introduces a number of RAID "levels". Which of the following is an advantage of RAID level 5 over RAID level 4?

(Circle True or False for each choice.)

A. True / False RAID level 5 uses fewer disks than RAID level 4, so it has a lower redundancy overhead.

FALSE. *RAID 5 uses the same number of disks as RAID 4.*

B. True / False RAID level 5 removes a bottleneck for small, random writes.

TRUE.

C. True / False RAID level 5 removes a bottleneck for large, sequential reads.

FALSE. *RAID 4 and RAID 5 perform comparably for large sequential reads.*

7. [8 points]: The Unison paper ("How to Build a File Synchronizer" by Trevonr Jim, Benjamin Pierce and Jerome Vouillon) states the following invariant for file synchronizers: At every moment during a run of the file synchronizer, every user file has either its original contents, or its correct final contents. Which of the following statements is true about the Unison file synchronizer?

(Circle True or False for each choice.)

A. True / False Unison does not attempt to maintain this invariant; its safety guarantees are based on a different invariant.

FALSE. *Unison attempts to maintain this invariant, though it does not always succeed.*

B. True / False Unison maintains this invariant in all cases.

FALSE. *Unison does not always succeed in maintaining this invariant because many file systems do not provide support for an atomic file replacement operation.*

C. True / False Unison uses file creation time to determine the most recent version of a file.

FALSE. *Unison doesn't identify the most recent version. It detects updates using cryptographic fingerprints of the file contents.*

D. True / False In order to compare two versions of a file, Unison sends the content of one version over the network.

FALSE. *Unison exchanges lists of changed files, including their fingerprints, between the two systems. The fingerprints are used to determine if the two versions are the same.*

8. [10 points]: Which of the following statements are true, given Professor Abelson's lecture?
(Circle True or False for each choice.)

A. True / False There are no restrictions on speech in the US law.

FALSE. *Though the First Amendment limits the power of Congress to make restrictions on speech, it is not absolute. For example, obscene speech can be prohibited.*

B. True / False The current version of the communication decency act has achieved its original goal (stopping the distribution of offensive material).

FALSE. *The "display" provision of the CDA, which was intended to prevent minors from harmful material on the Internet by restricting indecent speech, was ruled unconstitutional by the Supreme Court in* Reno v. ACLU.

C. True / False The "Good Samaritan" provision makes providers responsible for the content they distribute.

FALSE. *The "Good Samaritan" provision does precisely the opposite; it ensures that content distributors such as ISPs cannot be held responsible for content they did not originate.*

D. True / False Prof. Abelson libeled Prof. Kaashoek when he called him names.

FALSE. *Libel consists only of* written *defamation;* slander *is the equivalent for spoken communication.*

E. True / False The legal system suffers from the second-system effect.

FALSE. *The second-system effect refers to the tendency of system designers to make the second version of a successful system unnecessarily complex by adding too many new features. This doesn't apply to the legal system, because there is no clear second version of the system.*

II Reliable block store

Alice is interested in using the RECOVERABLE_PUT and RECOVERABLE_GET primitives to write and read sectors of a magnetic disk in a recoverable way. The primitives are intended to mask failures when the system fails (e.g., due to power failure) while RECOVERABLE_PUT is running. She uses the implementation that Prof. Liskov presented in lecture. In this implementation each recoverable item, x, is stored at two disk locations, $x.D0$ and $x.D1$, which are updated and read as follows:

```
    // Write the bits in data at item x
procedure RECOVERABLE_PUT(x, data)
 1. flag ← careful_get(x.D0, buffer);     // read into a temporary buffer
 2. if flag = ok  then {
 3.    CAREFUL_PUT(x.D1, data);
 4.    CAREFUL_PUT(x.D0, data);
 5. } else {
 6.    CAREFUL_PUT(x.D0, data);
 7.    CAREFUL_PUT(x.D1, data);
 8. }


    // Read the bits of item x and return them in data
procedure RECOVERABLE_GET(x, data)
 1. flag ← CAREFUL_GET(x.D0, data);
 2. if flag = ok  then
 3.    return;
 4. CAREFUL_GET(x.D1, data);
```

The CAREFUL_GET and CAREFUL_PUT procedures are also as specified in lecture and in the class notes (see page 9-86, figure 9-34). There should be no need to look up the pseudocode, the property that is relevant for the questions is that CAREFUL_PUT computes a checksum of the data stored, and CAREFUL_GET verifies the checksum. In this way CAREFUL_GET can detect cases when the original data is damaged by a system failure during CAREFUL_PUT.

In the following questions you may assume that the only failure to be considered is a fail-stop failure of the system during the execution of RECOVERABLE_GET and RECOVERABLE_PUT. After a fail-stop failure the system restarts.

9. [6 points]: Indicate which of the following is true (or false) for RECOVERABLE_PUT and RECOVERABLE_GET:

(Circle True or False for each choice.)

A. True / False This code obeys the rule "never overwrite the only copy"

TRUE. *As was shown in lecture, if only one copy of the data is uncorrupted,* RECOVERABLE_PUT *will write to the other copy first. This preserves the invariant that at least one copy of the data is always valid.*

B. True / False RECOVERABLE_PUT and RECOVERABLE_GET ensure that if just one of the two copies is good (i.e., CAREFUL_GET will succeed for one of the two copies), the caller of RECOVERABLE_GET will see it.

TRUE.

C. True / False RECOVERABLE_PUT and RECOVERABLE_GET ensure that the caller will always see the result of the last RECOVERABLE_PUT that wrote at least one copy to disk (i.e., the first CAREFUL_PUT completed successfully).

FALSE. *Consider the following case: initially, $x.D0$ is uncorrupted, and the system fails after completing the first* CAREFUL_PUT *(line 3). Then $x.D0$ will contain the old data and $x.D1$ the new data. However,* RECOVERABLE_GET *will return the old data from $x.D0$.*

10. [4 points]: Suppose that when RECOVERABLE_PUT starts running, the copy at $x.D0$ is good. Give the number of the statement whose completion is the commit point in RECOVERABLE_PUT.

LINE #4. *After the new data is written to $x.D0$,* RECOVERABLE_GET *will return the new data. Line #3 is not the commit point because of the case described in Problem 9.C.*

11. [4 points]: Suppose that when RECOVERABLE_PUT starts running, the copy at $x.D0$ is bad. Give the number of the statement whose completion is the commit point in RECOVERABLE_PUT.

LINE #6. *As soon as the new data is written to $x.D0$,* RECOVERABLE_GET *will return it, even though $x.D1$ still contains the old data.*

Name:

Consider the following chart showing possible statuses that the system could be in prior running RECOVERABLE_PUT:

	S1	**S2**	**S3**
$x.D0$	old	old	bad
$x.D1$	old	bad	old

For example, when the system is in status S2, $x.D0$ contains an old value and $x.D1$ contains a bad value (i.e., CAREFUL_GET will return an error).

12. [10 points]: Assume that RECOVERABLE_PUT is attempting to store a new value into item x and the system fails. Which of the following statements are true?

(Circle True or False for each choice.)

A. True / False ($x.D0$=new, $x.D1$=new) is a potential outcome of RECOVERABLE_PUT, starting in any of the three states.

TRUE. *This can happen if the system fails after successfully completing both* RECOVERABLE_PUT *operations.*

B. True / False Starting in state S1, a possible outcome is ($x.D0$=bad, $x.D1$=old).

FALSE. *If $x.D0$ is valid, then* RECOVERABLE_PUT *will overwrite $x.D1$ before touching $x.D0$.*

C. True / False Starting in state S2, a possible outcome is ($x.D0$=bad, $x.D1$=new).

TRUE. *This can happen if* RECOVERABLE_PUT *fails in the middle of the* CAREFUL_PUT *on line #4.*

D. True / False Starting in state S3, a possible outcome is ($x.D0$=old, $x.D1$=new).

FALSE. *$x.D0$ initially contains a bad value, and* RECOVERABLE_PUT *would not replace it with an old value.*

E. True / False Starting in state S1, a possible outcome is ($x.D0$=old, $x.D1$=new).

TRUE. *This can happen if* RECOVERABLE_PUT *fails after completing the* CAREFUL_PUT *on line #3, but before beginning the* CAREFUL_PUT *on line #4.*

Name:

Ben Bitdiddle proposes a simpler version of RECOVERABLE_PUT. This would be used with the RECOVERABLE_GET routine shown above.

procedure SIMPLE_PUT($x, data$)
  CAREFUL_PUT($x.D0, data$)
  CAREFUL_PUT($x.D1, data$)

13. [4 points]: Will the system work correctly if Ben replaces RECOVERABLE_PUT with SIMPLE_PUT? (Explain briefly).

NO. *The system will not work properly. It is possible for the system to first fail during the second* CAREFUL_PUT *(to $x.D1$), leaving $x.D0$ as the only good copy. If, on the next call to* RECOVERABLE_PUT, *the system fails during the first* CAREFUL_PUT *(to $x.D0$), both copies will be corrupted.*

14. [6 points]: Now consider other failures than system failures during RECOVERABLE_PUT. Indicate which of the following statements is true or false:

(Circle True or False for each choice.)

A. True / False Suppose $x.D0$ and $x.D1$ are stored on different disks. Then RECOVERABLE_PUT and RECOVERABLE_GET also mask a single head crash (i.e., the disk head hits the surface of a spinning platter), assuming no other failures.

TRUE. *The head crash can damage at most one disk, leaving all of the data on the other disk available to* RECOVERABLE_GET.

B. True / False Suppose $x.D0$ and $x.D1$ are stored as different sectors on the same track. Then RECOVERABLE_PUT and RECOVERABLE_GET also mask a single head crash, assuming no other failures.

FALSE. *The head crash can wipe out both copies of the data, leaving no good copies available to* RECOVERABLE_GET.

C. True / False Suppose that as part of the failure, the operating system misbehaves and overwrites $data$ (the in-memory copy of the data being written to disk by RECOVERABLE_PUT). Nevertheless, RECOVERABLE_PUT and RECOVERABLE_GET mask this failure, assuming no other failures.

FALSE. *The same damaged version of the data will be written to both $x.D1$ and $x.D0$, leaving no good copy available to* RECOVERABLE_GET.

Name:

Now we consider how to handle decay failures. The approach is to periodically correct them by running a SALVAGE routine. This routine checks each replicated item periodically and if one of the two copies is bad, it overwrites that copy with the good copy. The code for SALVAGE is on page 9-89 in the class notes (figure 9-35), but you don't really need to look at it to answer the question.

In the following you may assume that there is a decay interval D such that at least one copy of a duplicated sector will be good D seconds after the last execution of RECOVERABLE_PUT or SALVAGE on that duplicated sector. You may further assume that the system recovers from a failure in less than F seconds, where $F << D$, and that systems failures happen so infrequently that at most one can happen in a period of D seconds.

15. [8 points]: Which of the following guarantees that the approach handles decay failures.
(Circle True or False for each choice.)

A. True / False SALVAGE runs only in a background process that cycles through the disk with the guarantee that each replicated sector is salvaged every P seconds, where P is less than D - F.

TRUE. SALVAGE *will run frequently enough so that at most one sector can be corrupted in the time between checks.*

B. True / False SALVAGE runs only as part of running RECOVERABLE_PUT (i.e., first we salvage, then we overwrite).

FALSE. *Calls to* RECOVERABLE_PUT *can occur arbitrarily infrequently, leaving plenty of time for both copies of the sector to become corrupted.*

C. True / False SALVAGE runs only as part of running RECOVERABLE_PUT and RECOVERABLE_GET (i.e., in either case we first salvage, and then we read or write).

FALSE. *Same reasoning as above.*

D. True / False SALVAGE runs only as part of recovering from a fail-stop failure: at this point all duplicated sectors are salvaged.

FALSE. *Fail-stop failures can also occur arbitrarily infrequently, allowing both copies to become corrupted in the meantime.*

III Security

Ben makes the block store described in the previous section available as an Internet service. He modifies the block store to offer two remote procedure calls RECOVERABLE_PUT and RECOVERABLE_GET, which a program running on a client machine can invoke and which invoke the corresponding procedures on the server.

Alice and Bob use this service to share blocks, but they want to verify the integrity of the blocks they write. Ben writes two client procedures for them. These procedures make use of two recoverable blocks for each shared block; the first, *x.info*, stores the data, while the second, *x.sig*, stores the signature. (In the code, the notation n.PROCEDURE means invoke remote procedure call PROCEDURE on the server named n.)

procedure INTEGRITY_PUT$(x, data, k_{priv})$
  $signature \leftarrow sign(data, k_{priv})$;
  $server$.RECOVERABLE_PUT(*x.info*, $data$);
  $server$.RECOVERABLE_PUT($x.sig$, $signature$);

procedure INTEGRITY_GET$(x, data, k_{pub})$
  $server$.RECOVERABLE_GET(*x.info*, $data$);
  $server$.RECOVERABLE_GET($x.sig$, sig);
  if VERIFY$(data, sig, k_{pub})$ = ACCEPT **then return** OK;
  else return ERROR

Bob calls INTEGRITY_PUT with his private key and Alice calls INTEGRITY_GET with Bob's public key. SIGN and VERIFY are as described in the class notes.

Name:

16. [14 points]: What assumptions must Alice make to believe that when INTEGRITY_GET returns a block (that is, it doesn't return REJECT), it is indeed a block put by Bob?

(Circle True or False for each choice.)

A. True / False Bob didn't disclose his private key to anyone.

TRUE. *If Bob disclosed his private key to someone else, they could use it to sign data that Alice will then accept as coming from Bob.*

B. True / False An attacker doesn't impersonate the server $server$.

FALSE. *Even if an attacker is impersonating the server, he or she will be unable to generate a signature for which* VERIFY *will return* ACCEPT.

C. True / False Ben implemented VERIFY and SIGN correctly.

TRUE. *If* VERIFY *is implemented incorrectly (for example, if it always returns* ACCEPT*), it may incorrectly accept data that is not properly signed.*

D. True / False The block server didn't experience any decay failures.

FALSE. *If the block decays, the signature will not match (with high probability), and Alice will reject the data.*

E. True / False The attacker doesn't have the ability to store another signature into $x.sig$.

FALSE. *If the attacker puts another value into x.sig, it will not verify and Alice will reject the data.*

F. True / False Bob's public key **speaks for** Bob.

TRUE. *Alice verifies using Bob's public key, but she must assume that the public key she has is indeed the one that belongs to Bob. She must have met Bob and received his public key, or she must have received a certificate from some trustworthy source attesting that the key is Bob's.*

G. True / False The RPC system sends the remote procedure calls over an encrypted, authenticated communication channel between clients and *server*.

FALSE. *All data can be sent in the clear; verifying the signature on the message is sufficient to ensure the message's authenticity.*

Name:

IV Transactions

This question explores transactions using the following set of transactions:

- T1: begin(); write(x); read(y); write(z); commit();
- T2: begin(); read(x); write(z); commit();
- T3: begin(); read(z); write(y); commit();

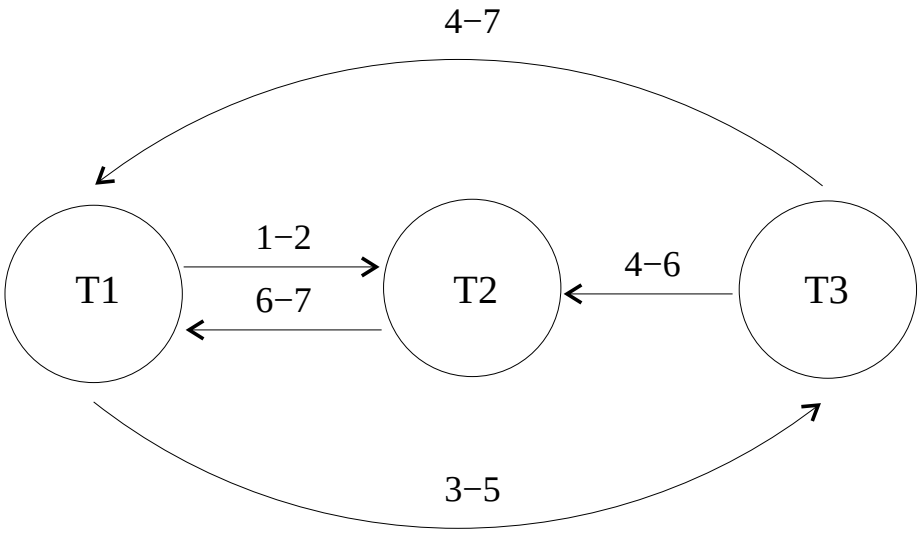
The transactions read and write the records x, y, and z, which are stored on a disk. Consider the following two schedules in which the reads and writes might happen:

- schedule 1:
  (1) write(x)$_{T1}$; (2) read(x)$_{T2}$; (3) read(y)$_{T1}$; (4) read(z)$_{T3}$; (5) write(y)$_{T3}$; (6) write(z)$_{T2}$; (7) write(z)$_{T1}$
- schedule 2:
  (1) read(z)$_{T3}$; (2) read(x)$_{T2}$; (3) write(x)$_{T1}$; (4) write(y)$_{T3}$; (5) read(y)$_{T1}$; (6) write(z)$_{T2}$; (7) write(z)$_{T1}$.

"(n) read(r)$_T$" should be read as at step n transaction T performs a read of r.

We are concerned with whether these schedules are serializable. The way to determine this is to compute the action graph. Recall that an *action graph* contains a node for each transaction in the schedule, and an arrow (directed edge) from T_i to T_j if T_i and T_j both used some record, r, in conflicting modes (e.g., write/write and write/read), and T_i used r before T_j used r.

17. [6 points]: For schedule 1 above, use the table below to indicate whether there is an arrow in the action graph from the first transaction to the second, and if there is, give a pair of steps that caused that arrow to be in the graph. We have started you off by showing one arrow in the graph. (For your convenience, you might wish to contruct the graph before filling out the table.)



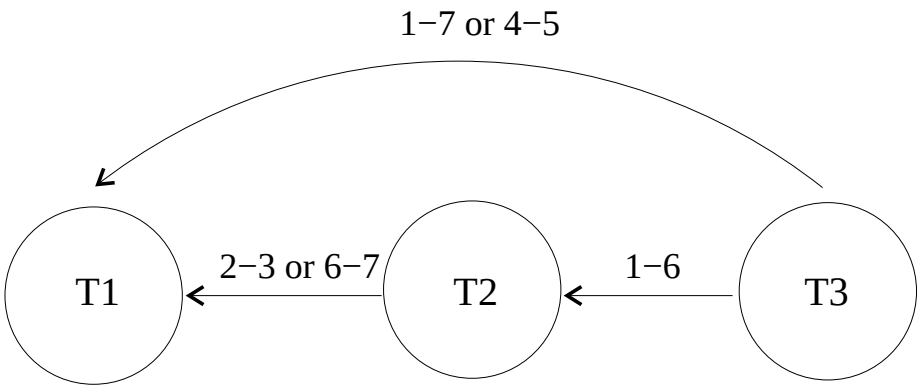
Edge	Yes/No	Reason
$T1 \rightarrow T2$	Y	Steps 1 and 2
$T1 \rightarrow T3$	Y	Steps 3 and 5
$T2 \rightarrow T1$	Y	Steps 6 and 7
$T2 \rightarrow T3$	N	
$T3 \rightarrow T1$	Y	Steps 4 and 7
$T3 \rightarrow T2$	Y	Steps 4 and 6

18. [4 points]: Is schedule 1 serializable? If no, explain briefly why not; if yes give a serial schedule for it.

NO. *This schedule is not serializable because there is a cycle in the action graph.*

Name:

19. [6 points]: Now fill out the table for schedule 2. This time we haven't provided you with any of the arrows in the graph. (Again, for your convenience, you might wish to contruct the graph before filling out the table.)



Edge	Yes/No	Reason
$T1 \rightarrow T2$	N	
$T1 \rightarrow T3$	N	
$T2 \rightarrow T1$	Y	Steps 2 and 3 (also steps 6 and 7)
$T2 \rightarrow T3$	N	
$T3 \rightarrow T1$	Y	Steps 1 and 7 (also steps 4 and 5)
$T3 \rightarrow T2$	Y	Steps 1 and 6

20. [4 points]: Is schedule 2 serializable? If no explain why not; if yes give a serial schedule for it.

YES. *This schedule is serializable because there is no cycle in the action graph. The serial schedule is* $T3$, $T2$, $T1$.

Name:

21. [4 points]: Could schedule 2 have been produced by two-phase locking, assuming that a transaction acquires a lock on an object as the first part of the step in which it first uses the object? For example, for step 3 of schedule 2, transaction T1 will acquire a write lock on x, and then write x. Answer true or false and give a brief (one sentence) explanation of your answer.

NO. *Schedule 2 could not have been produced by two-phase locking because two-phase locking does not allow a transaction to acquire a lock after it has released any locks. In schedule 2 this rule is violated. For example,* $T2$ *must acquire the lock on* x *in step 2, and release it before step 3 (since* $T1$ *needs to acquire this lock in step 3), but* $T2$ *needs to acquire the lock on* z *in step 6.*

Name:

Consider the following log that reflects schedule 2 above:

```
 1. begin T1
 2. begin T2
 3. begin T3
 4. update T1 x 1 2
 5. update T3 y 1 2
 6. update T2 z 1 2
 7. commit T3
 8. commit T2
 9. update T1 z 2 3
10. commit T1
```

You may assume that this log is correct. The notation in an update record indicates the old state just before this particular modification happened, and the new state just after it happened. For example, entry 4 indicates that the old state of x was 1 and the new state was 2. The system uses a correct redo/undo recovery procedure from the class notes.

22. [4 points]: Suppose the machine fails after record 7 of the log has made it to disk but before record 8 is on the disk. What states do x, y, and z have after recovery is complete?

- state of x: **1**
- state of y: **2**
- state of z: **1**

If record 7, but nothing after it, made it to disk, then the only transaction with a commit record logged is T3. So after recovery, only T3 will be part of the system state. T3 changes y from 1 to 2, but x and z keep their old values because they were only modified by T1 and T2, which did not commit.

23. [4 points]: Suppose the machine fails after record 9 of the log has made it to disk but before record 10 is on the disk. What states do x, y, and z have after recovery is complete?

- state of x: **1**
- state of y: **2**
- state of z: **2**

This time, record 9 makes it to disk, but not record 10, so T2 and T3 committed. Thus y and z have updated values, but since T1 never committed, no *change to x will be reflected in the system state after recovery and T1's change to z will also not be reflected in the system state.*

Name:

Suppose the database consists of a collection of integer objects stored on disk. Each write operation increments the object being modified by one, using a write-ahead logging protocol. Periodically, the system flushes objects from an in-memory cache to disk, whether the modifications to those objects have been committed or not.

To save space in the log, Ben suggests that update records just indicate the operation. For example, entry 4 would be:

```
update T1 x increment(1)
```

This record says that transaction $T1$ increments x by 1. When the transaction manager sees this entry during recovery, it performs the specified operation: increment x by 1. Ben makes no other changes to the recovery protocol.

24. [4 points]: Assuming all objects are initialized to 0, does Ben's plan work correctly? Why or why not. (Explain briefly.)

NO. *Ben's plan does not work correctly. The reason is that this type of log entry does not permit idempotent operations. In more detail: suppose transaction T1 increments x and commits. Since x might or might not have been flushed to disk from the cache, the transaction manager may or may not need to redo the update to x. However, it has no way to tell whether it needs to do so, because the update record does not contain the previous state of x. Similarly, if a transaction does not commit, the transaction manager does not know whether it needs to undo the increment operation or not.*

End of Quiz III—Enjoy the summer!!

Name: