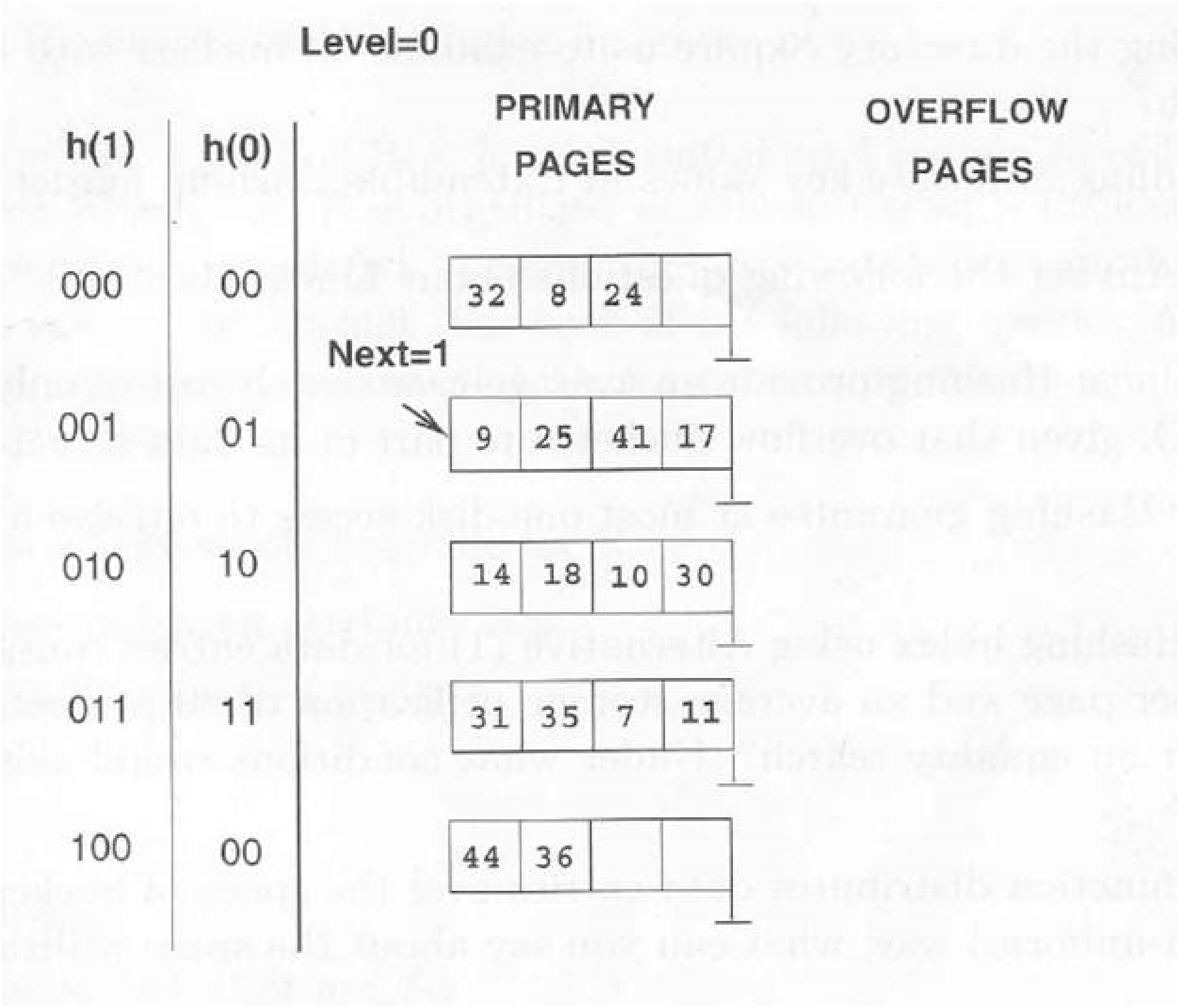
Homework 2 solutions

---

1. Consider the Linear Hast hashing index shown in the following figure. Assume that we split whenever an overflow page is created. Answer the following questions about this index:

1) Show the index after inserting an entry with hash value 12.
2) Show the original index after inserting an entry with hash value 29.
3) Show the original index after deleting the entries with hash values 36 and 44. (Assume that the full deletion algorithm is used).
4) Find a list of entries whose insertion into the original index would lead to a bucket with two overflow pages. Use as few entries as possible to accomplish this. What is the maximum number of entries that can be inserted into this bucket before a split occurs that reduces the length of this overflow chain?


Level=0
h(1)
h(0)
PRIMARY PAGES
OVERFLOW PAGES
000
00
32 8 24
Next=1
001
01
9 25 41 17
010
10
14 18 10 30
011
11
31 35 7 11
100
00
44 36

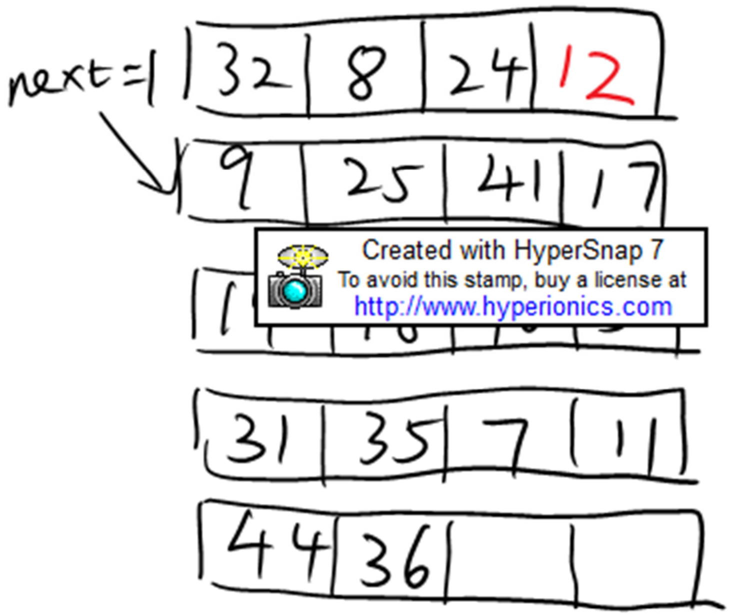
1)

level = 0

h(1)	h(0)
000	00
001	01
010	10
011	11
100	00

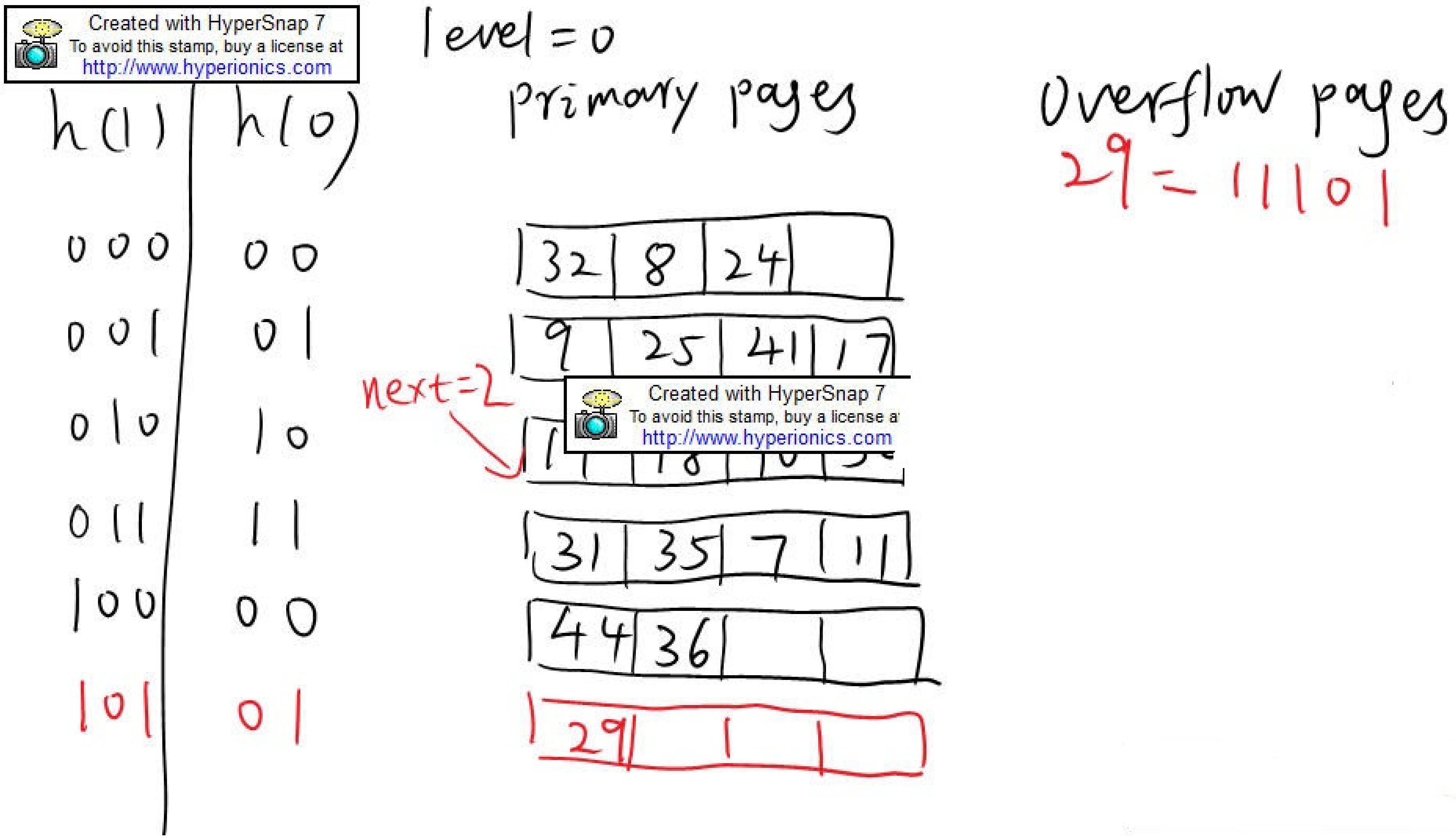
primary pages

Overflow pages



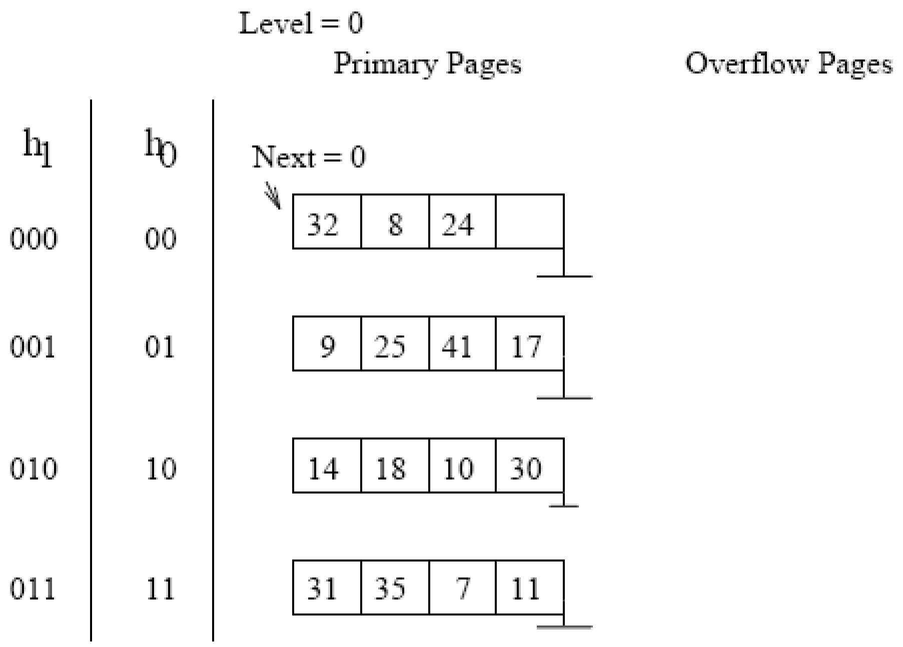
12 = 11$\underline{\underline{00}}$

2)



level = 0
primary pages
Overflow pages
29 = 11101
h(1)
h(0)
000 00
001 01
010 10
011 11
100 00
101 01
32 | 8 | 24
9 | 25 | 41 | 17
next=2
18
31 | 35 | 7 | 11
44 | 36
29

3)



4) **The following constitutes the minimum list of entries to cause two overflow pages in the index :**

63, 127, 255, 511, 1023

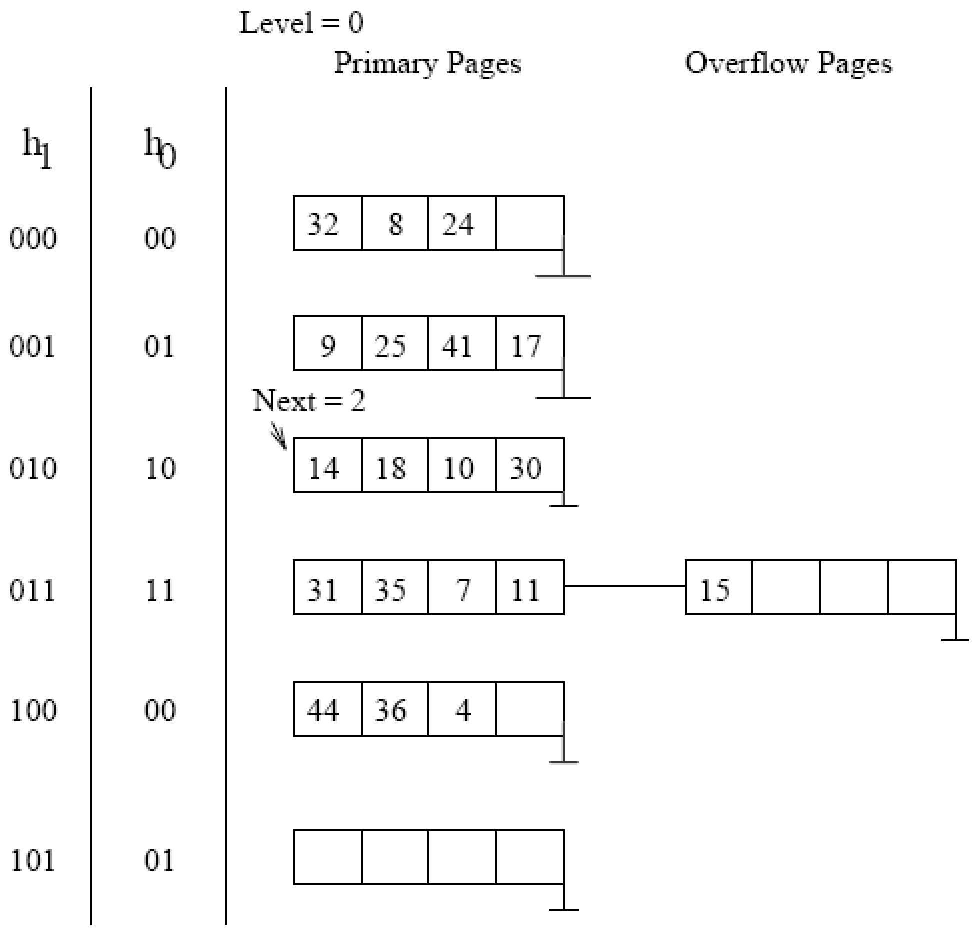
The first insertion causes a split and causes an update of Next to 2. The insertion of 1023 causes a subsequent split and Next is updated to 3 which points to this bucket.

This overflow chain will not be redistributed until three more insertions (a total of 8 entries) are made.

2. Consider the data entries in the Linear Hasing index shown in the following figure. Assume that a bucket split occurs whenever an overflow page is created.
1) What is the maximum number of data entries that can be inserted (given the best possible distribution of keys) before you have to split a bucket? Explain very briefly.

The maximum number of entries that can be inserted without causing a split is 6 because there is space for a total of 6 records in all the pages. A split is caused whenever an entry is inserted into a full page.

2) show the index after inserting a single record whose insertion causes a bucket split.



3) a. What is the minimum number of record insertions that will cause a split of all four buckets? Explain very briefly.

b. What is the value of Next after making these insertions?

c. What can you say about the number of pages in the fourth bucket showon after this series of record insertions?

$$\lceil \log_{19}(20{,}000/20) \rceil + 1 = 4$$

1) a) a) Consider the list of insertions 63, 41, 73, 137 followed by 4 more entries which go into the same bucket, say 18, 34, 66, 130 which go into the 3rd bucket. The insertion of 63 causes the first bucket to be split. Insertion of 41, 73 causes the second bucket split leaving a full second bucket. Inserting 137 into it causes third bucket-split. At this point at least 4 more entries are required. to split the fourth bucket. A minimum of 8 entries are required to cause the 4 splits.
   b) Since all four buckets would have been split, that particular round comes to an end and the next round begins. So Next = 0 again.
2) There can be either one data page or two data pages in the fourth bucket after these insertions. If the 4 more elements inserted into the 2nd bucket after 3rd bucket-splitting, then 4th bucket has 1 data page. If the new 4 more elements inserted into the 4th bucket after 3rd bucket spliting and all of them have 011 as its last three bits, then 4th bucket has 2 data pages. Otherwise, if not all have 011 as its last three bits, then the 4th bucket has 1 data page.

---

3. Consider the following classes of schedules: serializable, conflict-serializable, view-serializable, recoverable, avoids-cascading-aborts, and strict. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly. The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1). T1:R(A), T1:W(A), T1:R(B), T2:R(B), T3:W(B), T1:W(A), T2:R(B)

T1	T2	T3
R(A)		
W(A)		
R(B)		
	R(B)	
		W(B)
W(A)		
	R(B)	

It is NOT serializable, NOT conflict-serializable, NOT view-serializable;

It is NOT avoid cascading aborts, not strict;

We can not decide whether it’s recoverable or not, since the abort/commit sequence of these transactions are not specified.

2). T1:R(A), T2:W(A), T1:W(A), T2:Abort, T1:Commit

T1	T2
R(A)	
	W(A)
W(A)	

	Abort
Commit	

It is not serializable, not conflict-serializable, and not view-serializable;

It is recoverable and avoid cascading aborts;

It is not strict, because T2 writes A, A was overwritten by T1 before T2 aborts.

3). T1:R(A), T2:R(B), T4:R(C), T3:W(A), T4:W(C), T4:Abort, T2:R(A), T2:Commit, T1:R(B), T1:Commit, T3:Commit

T1	T2	T3	T4
R(A)			
	R(B)		
			R(C)
		W(A)	
			W(C)
			ABORT
	R(A)		
	Commit		
R(B)			
Commit			
		Commit	

It is serializable, conflict-serializable, and view-serializable,

It is NOT recoverable, because T2 reads A modified by T3 and T2 commits before T3 does.

Therefore, it is NOT avoid cascading aborts, NOT strict.

4). T1:R(A), T2:R(A), T3:R(B), T1:W(A), T2:W(A), T3:W(B)

T1	T2	T3
R(A)		
	R(A)	
		R(B)
W(A)		
	W(A)	
		W(B)

Not serializable, not view-serializable, not conflict-serializable

It is recoverable and avoid cascading aborts; not strict.

5). T1:W(A), T3: R(C), T2:R(B), T1:R(B), T1:Commit, T3: W(C), T3:Commit, T2:R(A), T2:Commit

T1	T2	T3
W(A)		
		R(C)
	R(B)	
R(B)		

Commit		
		W(C)
		Commit
	R(A)	
	Commit	

It is serializable, conflict-serializable, and view-serializable;

It is recoverable, avoid cascading aborts, and strict;

6). T1:W(A), T3:R(B),T2:R(A), T1:W(A), T3:Abort, T2:Commit, T1:Abort

T1	T2	T3
W(A)		
		R(B)
	R(A)	
W(A)		
		Abort
	Commit	
Abort		

It is NOT serializable, NOT view-serializable, and NOT conflict-serializable;

It is not recoverable, therefore not avoid cascading aborts, not strict.

7). T1:R(A), T3:W(A), T3:Commit, T1:W(A), T1:Commit, T2:R(A), T2:Commit

T1	T2	T3
R(A)		
		W(A)
		COMMIT
W(A)		
COMMIT		
	R(A)	
	COMMIT	

It is NOT serializable and NOT view-serializable, NOT conflict-serializable;

It is recoverable, avoid cascading aborts and strict.

8). T1:W(A), T3:R(B),T2:R(A), T3:W(B), T1:W(A), T2:Abort, T1:Commit,T3:Commit

T1	T2	T3
W(A)		
		R(B)
	R(A)	
		W(B)
W(A)		
	Abort	
Commit		

		Commit

It is not serializable, not view-serializable, not conflict-serializable;

It is not recoverable, therefore not avoid cascading aborts, not strict.

9). T1: R(A), T3:W(A), T3:Commit, T2:W(A), T2:Commit, T1:R(A), T1:W(A), T1:Commit

T1	T2	T3
R(A)		
		W(A)
		Commit
	W(A)	
	Commit	
R(A)		
W(A)		
Commit		

It is not serializable, not view-serializable, not conflict-serializable;

It's recoverable, ACA and strict

10). T1:W(A), T3:W(B), T2:R(A), T3:Abort, T1:W(A), T2:Commit, T1:Commit

T1	T2	T3
W(A)		

		W(B)
	R(A)	
		Abort
W(A)		
	Commit	
Commit		

It is not serializable, not view-serializable, not conflict-serializable;

It is not recoverable, therefore not avoid cascading aborts, not strict.

11). T1:R(A), T2:W(A), T1:W(A), T3:R(A), T1:Commit, T2:Commit, T3:Commit

T1	T2	T3
R(A)		
	W(A)	
W(A)		
		R(A)
Commit		
	Commit	
		Commit

It is NOT serializable and NOT view-serializable, NOT conflict-serializable;

It is recoverable, but not avoid cascading aborts, not strict.

12). T1:R(A), T2:W(B), T2:W(A), T1:W(A), T2:Commit, T1:Commit

T1	T2
R(A)	
	W(B)
	W(A)
W(A)	
	Commit
Commit	

It is NOT serializable, NOT view-serializable, not conflict-serializable;

It is recoverable and avoid cascading aborts;

It is not strict.

---

4. Consider the following sequence of actions, listed in the order they are submitted to the DBMS:

Sequence S1: T1:R(B), T2:R(C), T2:W(B), T3:R(A), T3:W(C), T1:W(A), T1:Commit, T2:Commit, T3:Commit

Sequence S2: T1: R(B), T1: W(C), T~~3~~2:R(A), T3(RA),T1:Commit, T2: R(B), T2:W(B), T2:Commit, T3: W(C), T3:W(A), T3: Commit

Assume that the timestamp of transaction Ti is i. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action of an unblocked transaction. For each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.

S1

T1	T2	T3
R(B)		
	R(C)	
	W(B)	
		R(A)
		W(C)
W(A)		
Commit		
	Commit	
		Commit

S2

T1	T2	T3
R(B)		
W(C)		
	R(A)	
		R(A)
Commit		
	R(B)	
	W(B)	

	Commit	
		W(C)
		W(A)
		Commit

a) Strict 2PL with timestamps used for deadlock prevention. Suppose the wait-die policy is used

S1

1. T1 gets a shared lock on B
2. T2 gets a shared lock on C
3. T2 asks for an exclusive lock on B, which conflicts with the shared lock on B held by T1, since T2 has lower priority, T2 is aborted.
4. T3 gets a shared lock on A
5. T3 gets a exclusive lock on C
6. T1 asks for an exclusive lock on A, which conflicts with the shared lock held by T3. since T1 has higher priority, T1 is blocked
7. T3 now finishes read, write, commits and releases all the locks.
8. T1 wakes up, gets the lock, proceeds and finishes.
9. T2 now can be restarted successfully.

S2

1. T1 gets a shared lock on B
2. T1 gets an exclusive lock on C
3. T2 gets a shared lock on A
4. T3 gets a shared lock on A

5. T1 finishes read, write, commits and releases all the locks.
6. T2 gets an exclusive lock on B.
7. T2 finishes write, commits and releases all the locks.
8. T3 gets an exclusive lock on C
9. T3 upgraded the shard lock on A to an exclusive lock.
10. T3 finishes and releases all the locks.

b) Strict 2PL with timestamps used for deadlock prevention. Suppose the wound-wait policy is used

S1:

1. T1 gets a shared lock on B.
2. T2 gets a shared lock on C
3. T2 asks for an exclusive lock on B, which conflicts with the shared lock on B held by T1. Since T2 has lower priority than T1, it waits.
4. T3 gets a shared lock on A
5. T3 asks for an exclusive lock on C , which conflicts with the shared lock held by T2. Since T3 has lower priority than T2, it waits.
6. T1 asks for an exclusive lock on A ,which conflicts with the shared lock held by T3. Since T1 has higher priority than T3, T3 aborts and T1 gets the exclusive lock on A.
7. T1 finishes and releases all the locks.
8. T2 wakes up, acquires the lock on B and finishes.
9. T3 now can be restarted successfully.

S2

1. T1 gets a shared lock on B
2. T1 gets an exclusive lock on C.
3. T2 gets a shared lock on A.
4. T3 gets a shared lock on A.

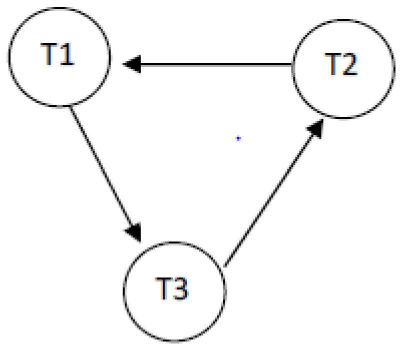
5. T1 finishes, commits and releases all the locks.
6. T2 gets an exclusive lock on B.
7. T2 finishes, commits and releases all the locks.
8. T3 gets an exclusive lock on C
9. T3 upgrades the lock on A to an exclusive lock.
10. T3 finishes, commits and releases all the locks.

c) Strict 2PL with deadlock detection. (Show the waits-for graph in case of deadlocks).

S1:

1. T1 gets a shared lock on B.
2. T2 gets a shared lock on C
3. T2 asks for an exclusive lock on B, which conflicts with the shared lock on B held by T1. T2 waits for T1
4. T3 gets a shared lock on A
5. T3 asks for an exclusive lock on C , which conflicts with the shared lock held by T2. T3 waits for T2
6. T1 asks for an exclusive lock on A ,which conflicts with the shared lock held by T3. T1 waists for T3.
7. a deadlock is detected.

The waits-for graph:



S2: the same as the above

d) Conservative 2PL

S1,

1. T1 gets a shared lock on B and an X lock on A
2. T2 gets a shared lock on C and waits for T1 to release the shared lock on B
3. T3 waits for T1 to release the lock on A and waits for T2 to release the shared lock on C
4. T1 commits and releases the two locks
5. T2 wakes up and get an X lock on B
6. T2 commits and releases the two locks
7. T3 wakes up and gets an X lock on B and a shared lock on A

S2,

1. T1 gets locks on both B and C, commits, and releases all the locks.
2. T2 gets locks on A and B, commits, and releases all the locks.
3. T3 gets locks on both A and C, commits, and releases all the locks.

d) Optimistic concurrency control

Each transaction will execute, read values from the database and write to a private workspace; they then acquire a timestamp to enter the validation phase. The timestamp of transaction Ti is i.

For sequence S1, since T1 has the earliest timestamp, it commits without any problem. For T2, T1 completes before T2 begins its Write phase, and WriteSet(T1) ∩ ReadSet(T2) is empty, so T2 commits without any problem. For T3, none of the three conditions holds, so T3 is aborted and restarted.

For sequence S2, since T1 has the earliest timestamp, it will commit without any problem. For T2, T1 completes after T2 begins its Read phase, and WriteSet(T1) ∩ ReadSet(T2) is empty, so T2 commits without any problem. Similaryly, T3 commits without any problem

f) Timestamp concurrency control without the Thomas Write Rule

S1

T1	T2	T3
R(B)		
	R(C)	
	W(B)	
		R(A)
		W(C)
W(A)		
Commit		
	Commit	

		Commit

1. T1 reads B   RTS(B) = 1
2. T2 reads C. RTS(C) = 2
3. T2 writes B,   (TS(T2) =2 > RTS(B)=1, allowed) WTS (B) = 2
4. T3 reads A, RTS(A) = 3
5. T3 writes C, (TS(T3) =3 > RTS(C)=2, allowed) WTS (C) = 3
6. T1 writes A, (TS(T1)<RTS(A), disallowed)
7. Abort and restart T1, T2 and T3 commits

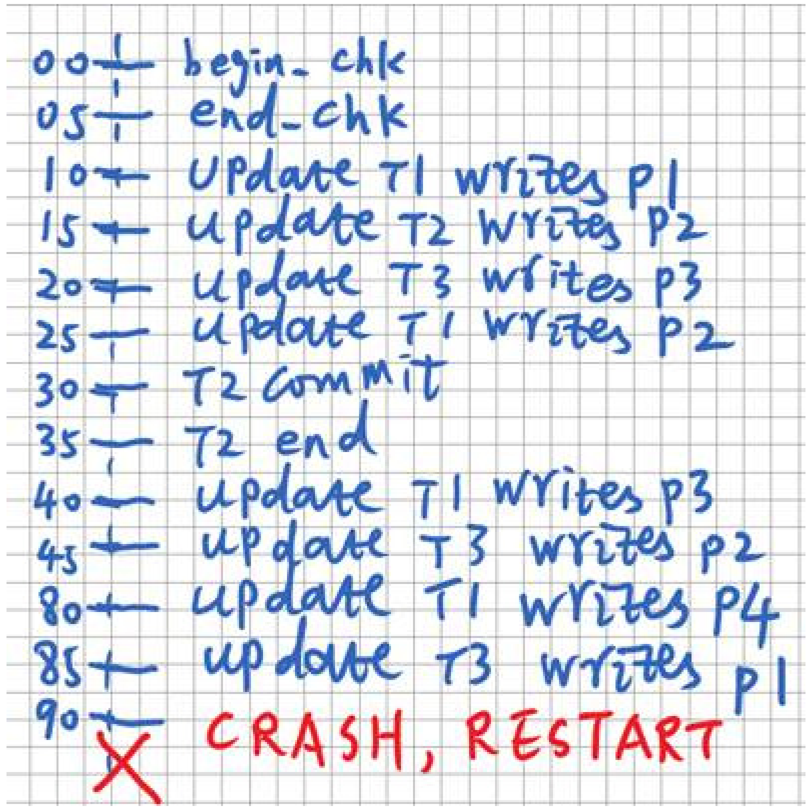
S2: all the actions of the 3 transactions are allowed and the 3 transactions commit successfully

g) Timestamp concurrency control with the Thomas Write Rue

the same as in f)

---

5. Consider the execution shown in following figure:



```
00 begin_chk
05 end_chk
10 update T1 writes P1
15 update T2 writes P2
20 update T3 writes P3
25 update T1 writes P2
30 T2 commit
35 T2 end
40 update T1 writes P3
45 update T3 writes P2
80 update T1 writes P4
85 update T3 writes P1
90 CRASH, RESTART
```

1) **After the crash, what is done during the Analysis phase? Give the tables gotten in this phase.**

In the analysis phase, the main task is to recover the transaction table and the dirty page table.

Dirty page Table

PageID	recLSN
P1	10
P2	15

P3	20
P4	80

Transaction table

TransID	State	LastLSN
T1	U	80
T3	U	85

2) What is done during the Redo phase? The answers should be in this format: Redo LSN xx, Redo LSN xx , …

Redo LSN   10

Redo LSN   15

Redo LSN   20

Redo LSN   25

Redo LSN   40

Redo LSN   45

Redo LSN   80

Redo LSN   85

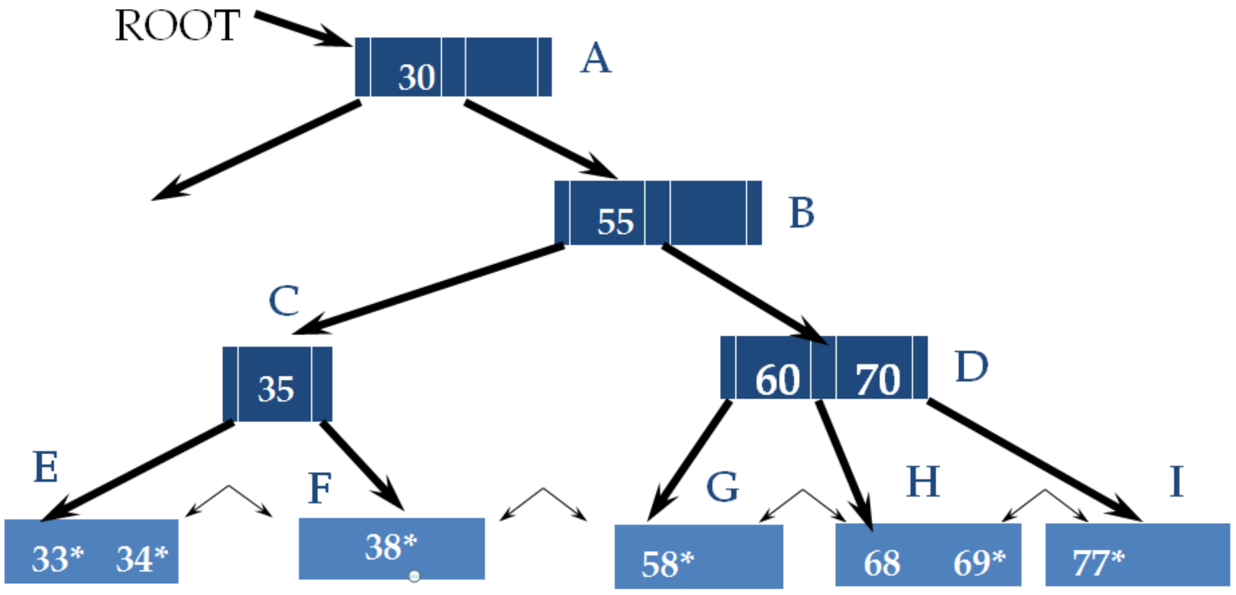
3) Show the log records after recovery, including all prevLSN and undonextLSN values in log records. Also show the initial toUndo list and the list at each step.

ToUndo = {80, 85}

100:

LSN	PrevLSN	Log record	UndonextLSN	ToUndo
100	85	CLR: Undo T3 LSN 85	45	{45, 80}
110	80	CLR: Undo T1 LSN 80	40	{40, 45}
120	100	CLR: Undo T3 LSN 45	20	{20,40}
130	110	CLR: Undo T1 LSN 40	25	{20,25}
140	130	CLR: Undo T1 LSN 25	10	{ 20,10}
150	120	CLR: Undo T3 LSN 20	-	{10}
160	150	T3 end	-	{10}
170	140	CLR: Undo T1 LSN 10	-	-
180	170	T1 end	-	-

6. Consider the following B+ tree with order 1, show step by step how to get and release locks for the following operations.



a) Search 38*

S lock on A

S lock on B, release the lock on A

S lock on C, release the lock on B

S lock on F, release the lock on C

b) Delete 68*

X lock on A

X lock on B

X lock on D, release lock on A, B

X lock on H, release lock on D

c) Insert 56*

X lock on A

X lock on B, release the lock on A

X lock on D

X lock on G, release lock on B and D

d) Insert 80*

X lock on A

X lock on B, release the lock on A

X lock on D,

X lock on I, release the lock on B and D

e) Delete 34*

X lock on A

X lock on B

X lock on C

X lock on E, release the locks on A,B and C

---

7. Consider a disk with an average seek time of 12ms, average rotational delay of 6ms, and a transfer time of 1ms for a 4K page. Assume that the cost of reading/writing a page is the sum of these values (i.e., 19ms) unless a sequence of pages is read/written. In this case, the cost is the average seek time plus the average rotational delay (to find the first page in the sequence) plus 1ms per page (to transfer data). You are given 640 buffer pages and asked to sort a file with 40,000,000 pages. Assume that you begin by creating sorted runs of 640 pages each in the first pass. Evaluate the cost of the following approaches for the subsequent merging passes:

a) Do 639-way merges.

b) Create 512 'input' buffers of 1 page each, create an 'output' buffer of 128 pages, and do 512-way merges.
c) Create 64 'input' buffers of 8 page each, create an 'output' buffer of 128 pages, and do 64-way merges.
d) Create 32 'input' buffers of 16 pages each, create an 'output' buffer of 128 pages, and do 32-way merges.
e) Create 16 'input' buffers of 32 pages each, create an 'output' buffer of 128 pages, and do 16-way merges.
f) Create 8 'input' buffers of 64 pages each, create an 'output' buffer of 128 pages, and do 8-way merges.

In Pass 0, 62500 sorted runs of 640 pages each are created. For each run, we read and write 640 pages sequentially.
The I/O cost per run is $2 * (12+6+1*640) = 1316$ ms.
Thus, the I/O cost for Pass 0 is $40,000,000/640 * 1316 = 82,250,000$ms.

For each of the cases discussed below, this cost must be added to the cost of the subsequent merging passes to get the total cost. Also, the calculations below are slightly simplified by neglecting the effect of a final read/written block .

a) For 639-way merges, only 2 more passes are needed. The first pass will produce 62500/640 = 98 sorted runs; these can then be merged in the next pass. Every page is read and written individually, at a cost of 19ms per read or write, in each of these two passes. The cost of these merging passes is therefore 2∗(2∗19)∗40,000,000 = 3,040,000,000ms.(The formula can be read as 'number of passes times cost of read and write perpage times number of pages in file'.)
b) With 512-way merges, only two additional merging passes are needed. Every page in the file is read and written in each pass, but the effect of blocking is different on reads and writes. For reading, each page is read individually at a cost of 19ms.Thus, the cost of reads (over both passes) is 2 ∗ 19 ∗ 40,000,000 = 1,520,000,000ms. For writing, pages are written out in blocks of 128 pages. The I/O cost per block is 12 + 6 + 1 ∗ 128 = 146ms. The number of blocks written out per pass is 40,000,000/128 = 312,500, and the cost per pass is 312500∗146 = 45,625,000ms. The cost of writes over both merging passes is therefore 2 ∗ 45,625,000 = 91,250,000ms. The total cost of reads and writes for the two merging passes is 1,520,000,000ms + 91,250,000ms = 1,611,250,000ms.
c) With 64-way merges, 3 additional merging passes are needed. For reading, pages are read in blocks of 8 pages, at a cost per block of 12 + 6 + 1 ∗ 8 = 26ms. In each pass, 40,000,000/8 = 5,000,000 blocks are read. The cost of reading over the 3 merging passes is therefore 3 ∗ 5,000,000 ∗ 26 = 390,000,000ms. For writing, pages are written in 128 page blocks, and the cost per pass is 45,625,000ms as before. The cost of writes over 3 merging passes is 3 ∗ 45,625,000ms = 136,875,000ms, and the total cost of the merging passes is 390,000,000ms + 136,875,000ms = 525,875,000ms.
d) With 32-way merges, 4 merging passes are needed. For reading, pages are read in blocks of 16 pages, at a cost per block of 12 + 6 + 1 ∗ 16 = 34ms. In each pass, 40,000,000/16 = 2,500,000 blocks are read. The cost of reading over the 4 merging passes is therefore 4∗ 2,500,000 ∗ 34 = 340,000,000ms. For writing, pages are written in 128 page blocks, and the cost per pass is 45,625,000ms as before. The cost of writes over 4 merging

passes is 4 ∗ 45,625,000ms = 182,500,000ms, and the total cost of the merging passes is 340,000,000ms + 182,500,000ms = 522,500,000 ms.

e) With 16-way merges, 4 merging passes are needed. For reading, pages are read in blocks of 32 pages, at a cost per block of 12 + 6 + 1 ∗ 32 = 50ms. In each pass, 40,000,000/32 = 1,250,000 blocks are read. The cost of reading over the 4 merging passes is therefore 4∗ 1,250,000 ∗ 50 = 250,000,000ms. For writing, pages are written in 128 page blocks, and the cost per pass is 45,625,000ms as before. The cost of writes over 8 merging passes is 4 ∗ 45,625,000ms =182,500,000ms, and the total cost of the merging passes is 250,000,000ms + 182,500,000ms = 432,500,000 ms.

e) With 8-way merges, 6 merging passes are needed. For reading, pages are read in blocks of 64 pages, at a cost per block of 12 + 6 + 1 ∗ 64 = 82ms. In each pass, 40,000,000/64 = 625,000 blocks are read. The cost of reading over the 6 merging passes is therefore 6∗ 625,000 ∗ 82 = 307,500,000ms. For writing, pages are written in 128 page blocks, and the cost per pass is 45,625,000ms as before. The cost of writes over 8 merging passes is 6 ∗ 45,625,000ms =273,750,000ms, and the total cost of the merging passes is 307,500,000ms + 273,750,000ms = 581,250,000 ms.

---

8. Suppose we have a relation Books which contains 4 attributes: bid, btitle, pid and price. All attributes are of the same length. bid is the primary key. The relation contains 10,000 pages and there are 50 buffer pages available. Consider the following query:

```
SELECT DISTINCT    bid, pid
FROM  Books
```

If you use the optimized version of the Consider the optimized version of the sorting-based projection algorithm: The initial sorting pass reads the input relation and creates sorted runs of tuples containing only attributes bid and btitle. Subsequent merging passes eliminate duplicates while merging the initial runs to obtain a single sorted result (as opposed to doing a separate pass to eliminate duplicates from a sorted result containing duplicates).

a) How many sorted runs are produced in the first pass? What is the average length of these runs? (Assume that memory is utilized well and replacement sorting is used.) What is the I/O cost of this sorting pass?
In pass 0, we read in the entire file of 10,000 pages, remove unwanted attributes and writes out 10,000*2/4 = 5,000 pages.
Since replacement sorting is used, (5000/(2*40)) = 75 runs are generated. On average, each run contains 80 pages.
The I/O cost is 15,000 pages.
b) How many additional merge passes are required to the final result of the projection query? What is the I/O cost of these additional passes?
The number of additional passes: $\log_{49}80$ = 2
the cost is 2 x 5,000+5,000 = 15,000

---

9. Consider a relation with this schema:

Authors(<u>aid: integer</u>, aname: string, age: integer, sal: integer)

Suppose that the following indices, all using Alternative (2) for data entries, exist:

- a hash index on aid
- a hash index on age
- a clustered B+ tree index on <age, sal> (the height of the tree is 4)

- an unclustered B+ tree index on sal (the height of the tree is 4)

Each author record is 60 bytes long, and you can assume that each index data entry is 15 bytes long (the index entry of the index on <age,sal> is 30 bytes long). The Author relation contains 40,000 pages and each page contains 40 tuples.

1)). Consider each of the following selection conditions and, assuming that the reduction factor (RF) for **EACH term** that matches an index is 0.2, compute the cost of the most selective access path for retrieving all Author tuples that satisfy the condition:

(a) sal > 100

Since there is no clustered index on sal, file scan is the most selective access path. The cost is 40,000

(b) age = 25

age = 25 The clustered B+ tree index on <age, sal> would be the best option here, with a cost of 4 (lookup) + 40,000 *30/60 * 0.2 (number of leaf pages) + 40000 pages * 0.2 (number of data pages)= 12,004.

Although the hash index has a lesser lookup time, the potential number of record lookups (40,000 pages * 0.2 * 40 tuples per page = 32,000) renders the clustered index more efficient.

(c) age > 20

The same as (b)

(d) aid = 1, 000

If aid is a candidate key, the hash index on aid is the most selective access path since only one record will be in each bucket. Thus, the total cost is roughly 1.2 (lookup) + 1(record access) which is 2 or 3.

If aid is not a candidate key, we'd better use file scan which costs 40,000 page I/0s.

(e) sal > 200 $\land$ age = 20

It's similar to (b). The clustered B+ tree index on <age, sal> would be the best option here. The cost is:

4 (lookup) + 40,000 *30/60 * 0.2*0.2 (number of leaf pages) + 40000 pages * 0.2*0.2 (number of data pages) = 2404

(f) sal > 200 $\land$ title = 'Mr.'

The cheapest method available is a simple filescan, with a cost of 40000 I/Os.

2) Suppose that , for the selection conditions (b), (e), you want to compute the average salary for qualifying tuples. Describe the least expensive evaluation method and state its cost for each of the two conditions.

For (b), the best option is to use the clustered index on < age,sal >, since it will avoid a relational lookup. The cost of this operation is 4 (B+ tree lookup) + 40000 * 0.2 * 30/60 (due to smaller index tuple sizes) = 4004.

Similarly for (e), the cost is 4 (lookup) + 40,000 *30/60 * 0.2*0.2 = 804

---

10. Consider the join of two tables TA and TB on the condition TA.s = TB.s. The cost metric is the number of page I/Os and the cost of writing out the result should be uniformly ignored. You're given the following information

Relation TA contains 10,000 tuples and has 20 tuples per page.
Relation TB contains 3000 tuples and also has 20 tuples per page.
Attribute s of relation TA is the primary key for TA.
Both relations are stored as simple heap files.
Neither relation has any indices built on it.
32 buffer pages are available.

a). What is the cost of joining TA and TB using a page-oriented simple nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?

Let M = 10,000/20 = 500 be the number of pages in TA, N = 3,000/20 = 150 be the number of pages in TB, and B = 32 be the number of buffer pages available.
The basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples.
Total cost would be
#pagesinouter + (#pagesinouter * #pagesininner)
which is minimized by having the smaller relation be the outer relation.
TotalCost = N + (N * M) = 150 + 150*500 = 75,150
The minimum number of buffer pages for this cost is 3.

b). What is the cost of joining TA and TB using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?

This time read the outer relation in blocks, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block. The number of blocks of outer tables is: (150/30) = 5
The total cost is: N + M * [N/(B-2)] = 150 + 500 * 5= 2,650
The minimum number of buffer pages for this cost is 32.

c). What is the cost of joining TA and TB using a sort-merge join? What is the minimum number of buffer pages required for this cost to remain unchanged?

Since B > sqrt(M) , we can use the refinement to Sort-Merge
TotalCost = 3*(M + N) = 800
The minimum number of buffer pages required is sqrt(500) = 23 . With 23 buffer pages, both tables can be sorted using the same number of passes. So the cost remains the same

d). What is the cost of joining TA and TB using a hash join? What is the minimum number of buffer pages required for this cost to remain unchanged?
Same as C.