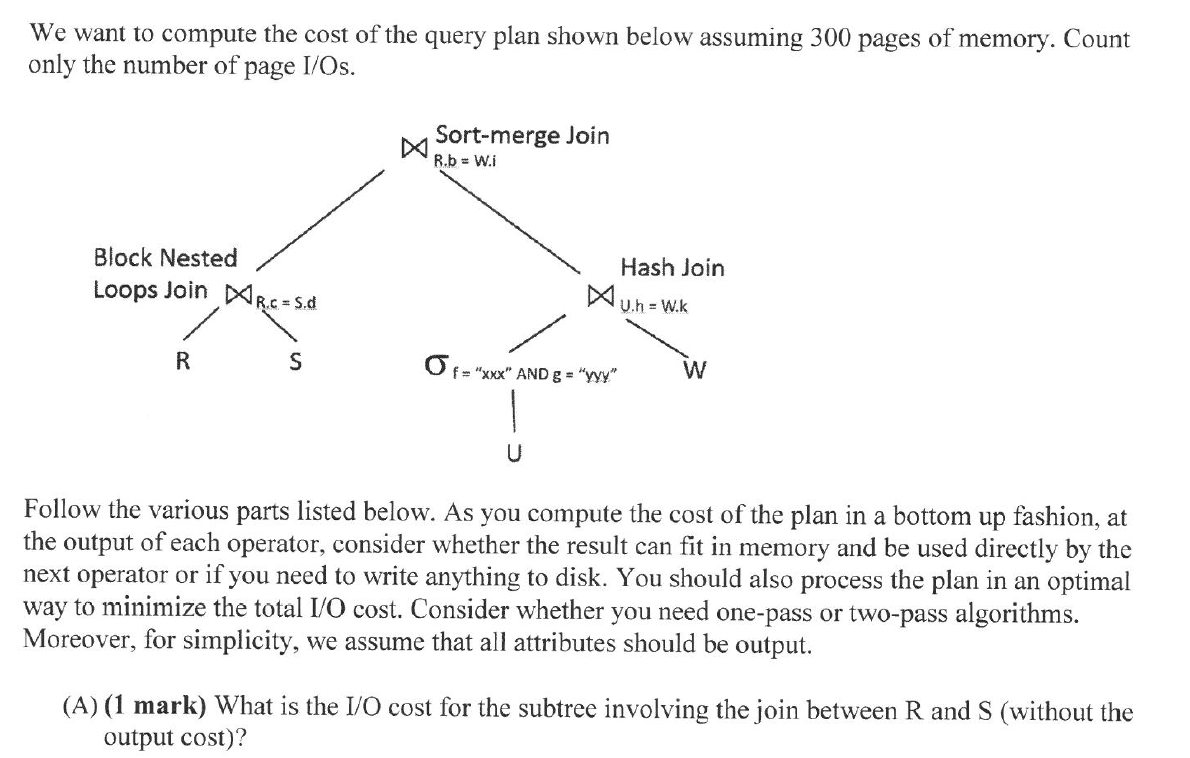
Q1

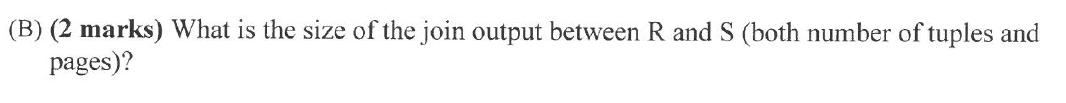


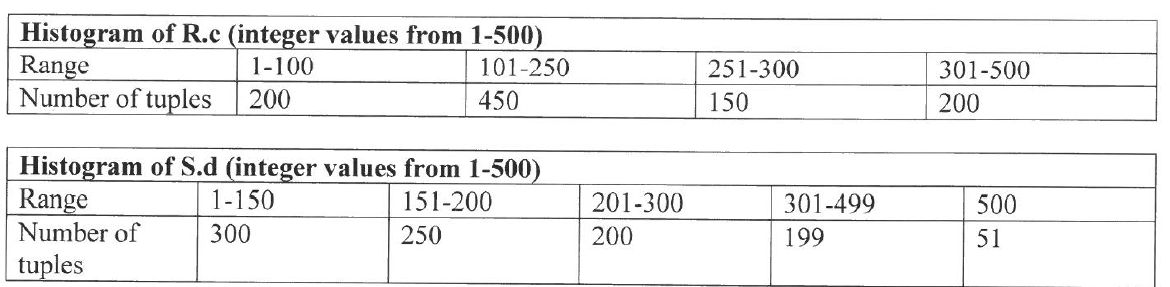
We have 300 pages of memory

|R| = 100 pages

|S| = 100 pages

Simply scan all R pages and all S pages and perform in-memory sort. Total cost = **200** I/Os /





Assume uniform distribution

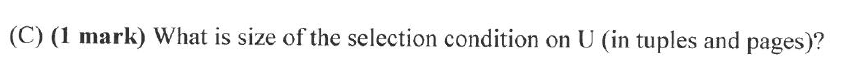
Let R JOIN S be T1

|  |  |  |  |
| --- | --- | --- | --- |
| Range | # of R tuples | # of S tuples | # of T1 tuples wrong |
| 1-100 | 200 | 200 | 40000  100 \* 2 \* 2 = 400 |
| 101-150 | 150 | 100 | 15000  50 \* 3 \* 2 = 250 |
| 151-200 | 150 | 250 | 37500  50 \* 3 \* 5 = 400 |
| 201-250 | 150 | 100 | 15000  50 \* 3 \* 2 = 300 |
| 251-300 | 150 | 100 | 15000  50 \* 3 \* 2 = 300 |
| 301-499 | 199 | 199 | 39601  199 \* 1 \* 1 = 199 |
| 500 | 1 | 51 | 51  1 \* 1 \* 51 = 51 |
| Total | 1000 | 1000 | **162152** |

Assume size of tuples are all the same and one page can fit 10 tuples.

# tuples in T1 = **162152** x **2300**

# pages in T1 = **16216** x 460 (5 tuples per page)

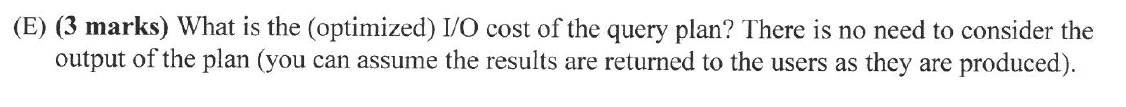


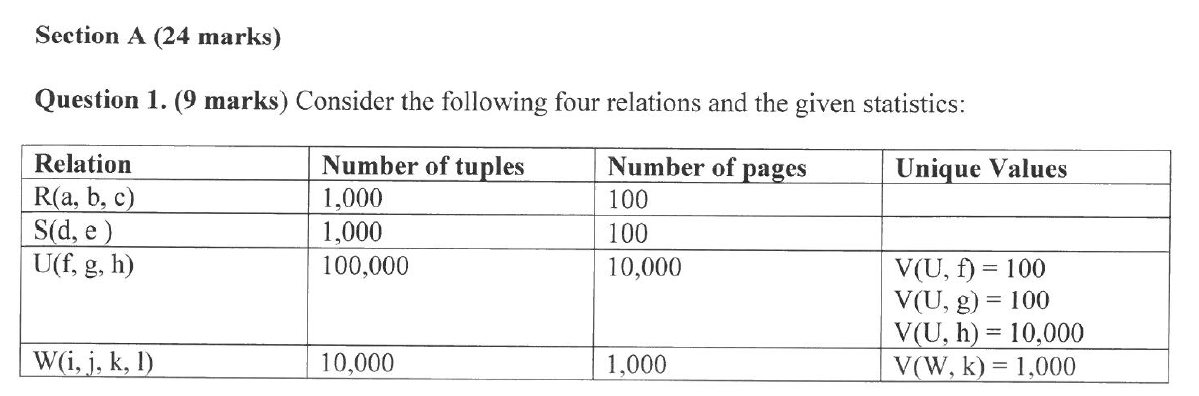
Assuming uniform distribution, size of select result = 1/100 \* 1/100 \* 100000 = **10 tuples**

**# tuples = 10**

**# pages = 1**

/





V(U,h) = 10 (by independence assumption, max possible values)

V(W,k) = 1000

Every tuple in U will match with 10000/1000=10 tuples in W

Result size = 10 by 10 = 100 tuples, need only 10 pages to store results.

Reserve 10 pages in memory to store results so that there is no need to write out.

Hash join U and W:

U: 1 page

W: 1000 pages

Cost:

* (2+1) \* (1000 + 1) = **3003** I/Os

Sort Merge Join T1 and T2

# pages in T1 = 16216 pages 460

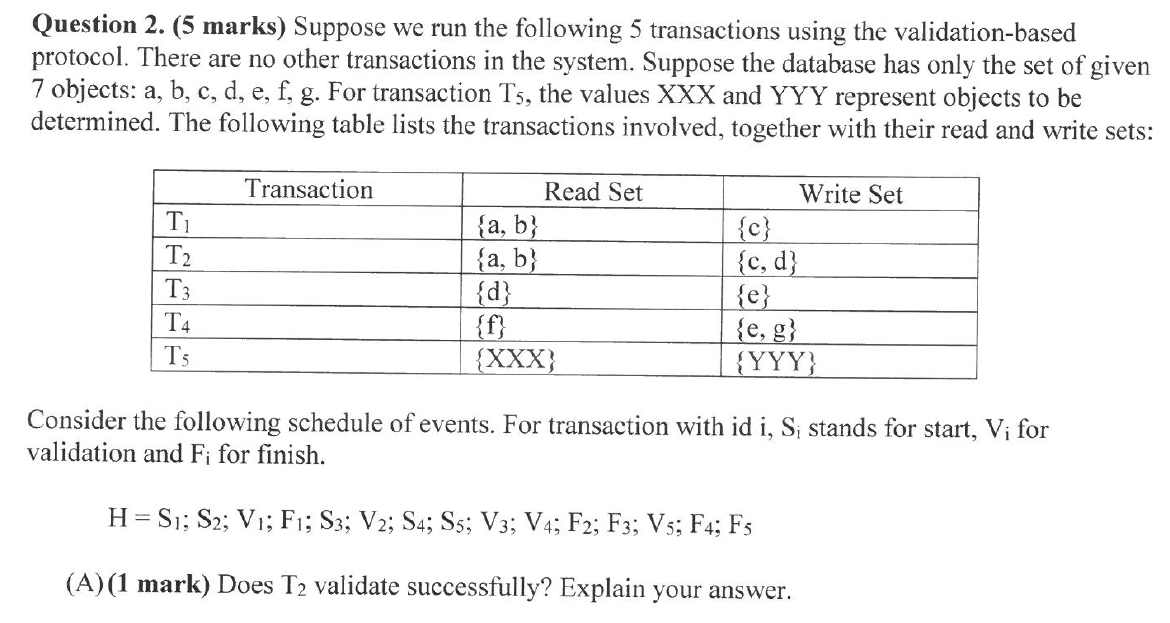
# pages in T2 = 10 pages in memory

Sort T1 = 2 \* 16216 \* (1+ceil(1+log289(16216/290)))

= **64864** I/Os

Merge = **16216** I/Os

**Total** = 16216 + + 64864 + 3003 + 200 + 100 + 100 + 10000 + 1000 = **95483 I/Os**



S2 < F1 < V2: Check if R set of T2 overlaps with W set of T1. No. Therefore, T2 validates successfully.

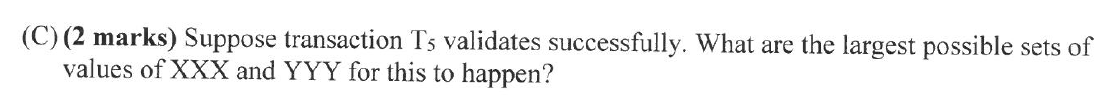
/



No need to check with T1 since T3 starts after T1 finishes

S3 < V3 < F2: Check if R and W set of T3 overlaps with W set of T2. Yes, T3 reads ‘d’ that is written by T2. Therefore, T3 does not validate successfully.

/



Checks if T4 validates successfully.

V4 < F2. Check if R and W set of T4 overlaps with W set of T2. No. T4 validates successfully.

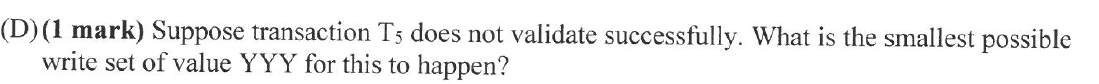
S5 < F2. R set of T5 must not overlap with W set of T2

V5 < F4. R and W set of T5 must not overlap with W set of T4.

XXX = {a,b,f} // no c,d,e,g

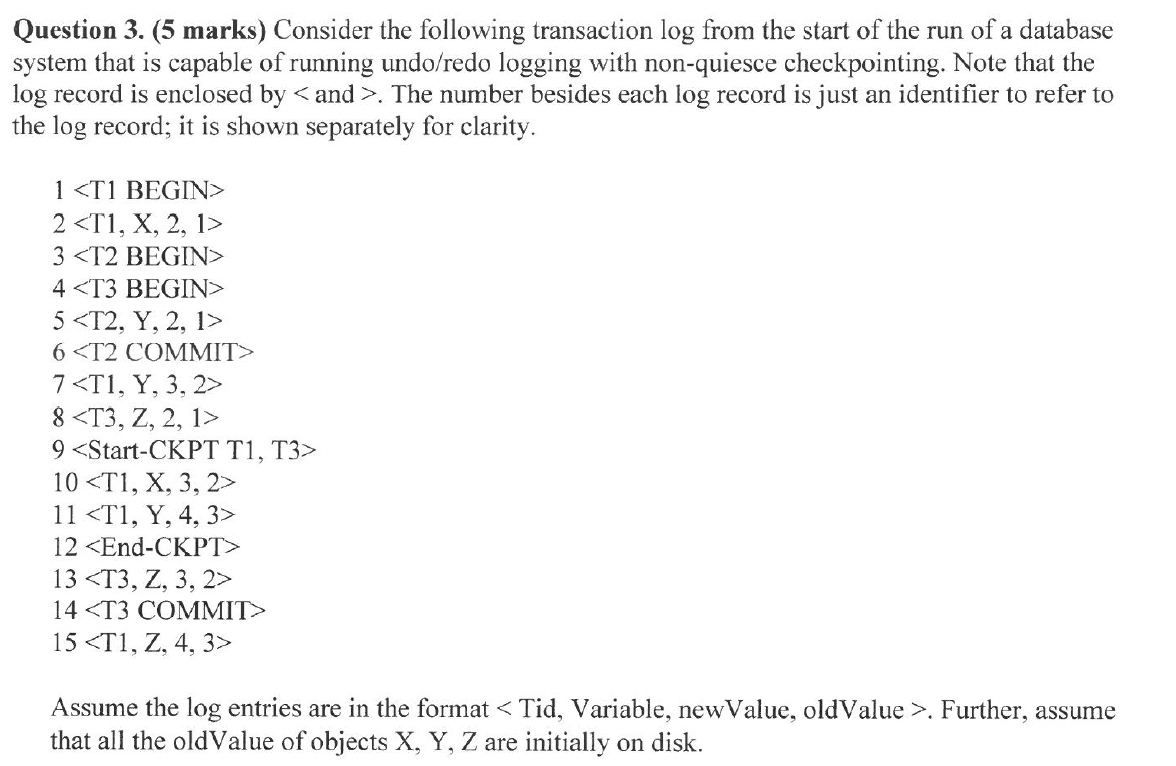
YYY = {a,b,c,d,f} // no e,g

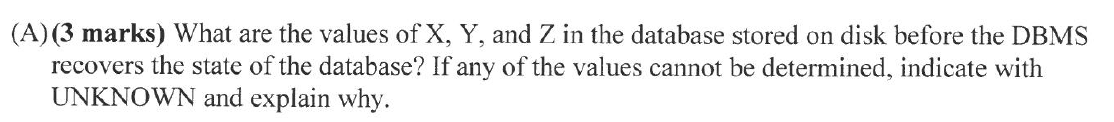
/



Smallest possible write set is empty set. T5 could possibly fail validation due to its read set (XXX) overlapping with the write set of T2 and T4.

/



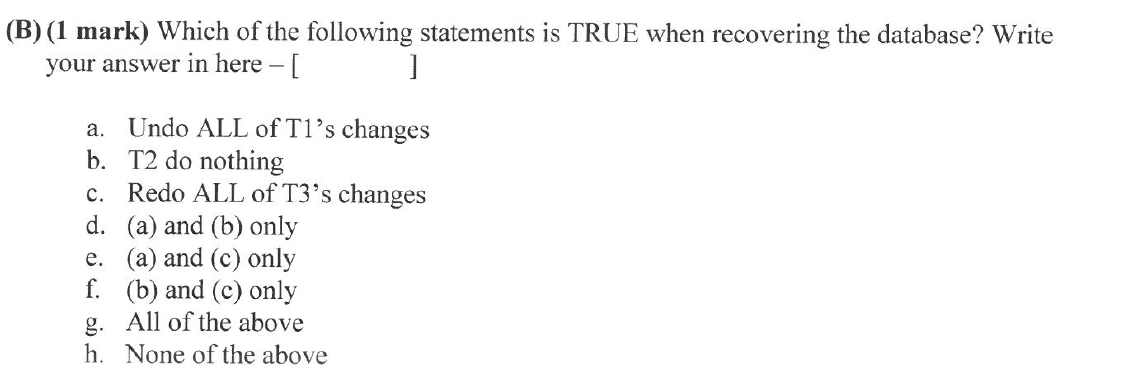


X=Unknown. T1 has not committed yet. Updates on X may or may not be reflected on disk. Can be 1,2,3,4

Y=Unknown. Definitely not 1 because T2 has committed its write on Y. But T1 has not committed yet. Updates on Y by T1 may or may not be reflected on disk. Can be 2,3,4

Z=Unknown. Definitely not 1 because T3 has committed and changes before the Start CKPT must be reflected on disk. Can be 2,3,4

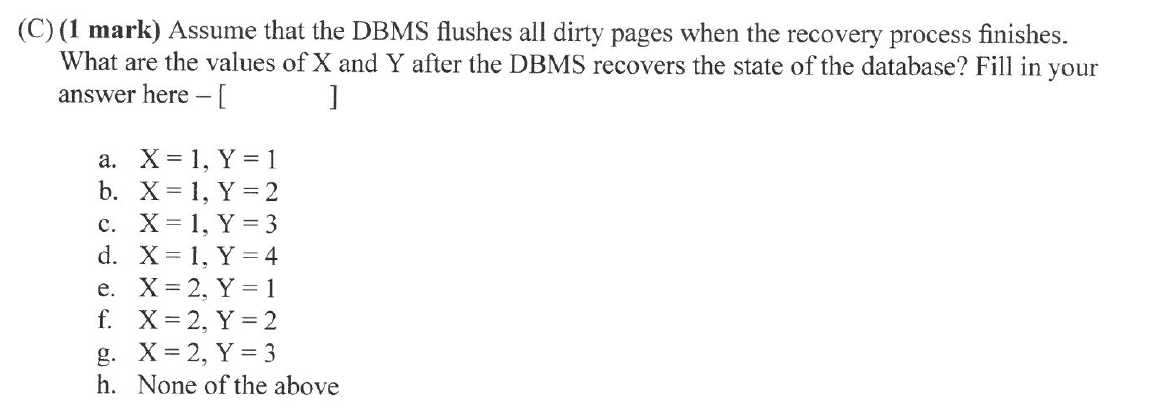
/ all unknown. The checkpoint flushed every to disk, but then all the data objects were modified by transactions after the checkpoint. Since we are using NO FORCE, any dirty page could be written to disk, so we don’t know the contents of the database on the disk at the time of crash



[d]

No need to redo 8 <T3,Z,2,1> because actions before the start ckpt for committed Xact are guaranteed to be reflected on disk

/



[b]

T1 is aborted. Changes of T2 remains

/



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Requestor | | | | | |
| Holder | IS | IX | IU | S | U | X |
| IS | T | T | T | T | T / | F |
| IX | T | T | T | F | F / | F |
| IU | T | T | T / | F / or T | F / | F |
| S | T | F | T / | T | T | F |
| U | F / or T | F / | F / | F | F | F |
| X | F | F | F | F | F | F |

Section B

1. G. S is serializable, conflict equivalent to T1,T2,T3, 2,1,3 too. 321 not possible cos of conflicting actions on C between T2 and T3. /
2. C x D [S cannot be produced by a 2PL scheduler and is not serializable]
   1. S = R1(A), W1(A), R2(B), W2(C), R2(A), R3(C), W3(B), R1(B)
   2. S = **L1(A),** R1(A), W1(A), **U1(A),** **L2(B),** R2(B), **L2(C),** W2(C), **L2(A),** R2(A), **U2(B), U2(A), U2(C), L3(C)** R3(C), **L3(B),** W3(B), **U3(C), U3(B), L1(B),** R1(B), **U1(B)**
3. A /
4. B /
5. G. Using SIX for C is redundant or E /
6. F x G Oh I saw wrong option, meant to choose all are correct
7. E /
8. C /
9. G / or A or F
10. G x E Invalid to say STEAL+NO-FORCE gives you fastest recovery performance
11. D /
    1. L1(A)
    2. T2 dies
    3. L3(B)
    4. T4 aborts
    5. T1 waits for B from T3
    6. T2 dies
    7. L3(C)
    8. L4 dies
    9. T1 waits for B from T3
    10. T2 dies
    11. U3(B)
    12. T4 dies
    13. L1(B)
    14. T2 dies
    15. U3(C)
    16. T4 dies
    17. U1(A)
    18. L2(A)
    19. T4 dies
    20. U1(B)
    21. L2(C)
    22. T4 dies
    23. U2(A)
    24. L4(A)
    25. U2(C)
    26. L4(D)
    27. U4(A)
    28. U4(D)
12. D /
13. F /
14. D /
15. F /
16. A (Does 2PL avoid deadlock?) /