1. (a)
r1(x)1 must precede w2(x)2. This implies that w1(y)2 must precede r2(z)1.
r2(z)1 must precede w3(z)2. This implies that r2(z)1 must precede r3(y)1.
From the above two statements, w1(y)2 must precede r3(y)1.
However, r3(y)1 must precede w1(y)2 as the initial value is 1.
Therefore, it is not possible to construct a linear sequence.

(b) The linear sequence which is equivalent to the given execution is
r1(x)1, r2(y)1, w1(y)2, r3(y)2, w2(z)3, r3(z)3, r3(x)1, w2(x)2, r1(x)2

2. See attached

3. Total = 23
Q(enqueue) > 23/2
Q(dequeue) > 23/2
Q(peek) > 23/4
Q(dequeue) + Q(peek) > 23

(a) A possible quorum assignment is Q(enqueue) = 12, Q(dequeue) = 18, Q(peek) = 6
(b) Q(enqueue) = 12, Q(dequeue) = 12, Q(peek) = 12

4. In two-phase commit, the blocking scenario involves the case when all nodes which know of the decision (all of the votes) have crashed. If the coordinator does not crash, then it will receive votes from all cohorts that have not crashed. For those cohorts that crash, the coordinator will assume a No vote. Hence, the coordinator (which does not fail) will always be available and will know of the decision. So, no cohort will have to wait – they can always contact the coordinator to find out what decision to make.

5. a. We will look at each operation in the order given and give the read and write timestamp after each operation.

Initial values: x = 0, y = 0; z= 0; rt(x) = 0, wt(x) = 0, rt(y) = 0; wt(y) = 0 ; rt(z) = 0; wt(z) = 0
r1(x): return value 0; rt(x) = 1, wt(x) = 0, rt(y) = 0; wt(y) = 0 ; rt(z) = 0; wt(z) = 0
r3(y): return value 0; rt(x) = 1, wt(x) = 0, rt(y) = 3; wt(y) = 0 ; rt(z) = 0; wt(z) = 0
r2(x): return value 0; rt(x) = 2, wt(x) = 0, rt(y) = 3; wt(y) = 0 ; rt(z) = 0; wt(z) = 0
w3(z)2: success rt(x) = 1, wt(x) = 0, rt(y) = 3; wt(y) = 0 ; rt(z) = 0; wt(z) = 3
w1(z)5: skip rt(x) = 1, wt(x) = 0, rt(y) = 3; wt(y) = 0 ; rt(z) = 0; wt(z) = 3
w1(y)4: abort T1 since rt(y) is already 3
w2(y)3: abort T2 since rt(y) is already 3
r3(y): return 0

b. T1: rl(x); r1(x)0; wl(z); w1(z)5;wl(y);w1(y)4;ul(x,y,z)
T2: rl(x); r2(x)0 wl(y); w2(y)3; ul(x,y)
T3: rl(y); r3(y)0; wl(z); w3(z)2 r3(y)0; ul(y,z);

6. a.

(i) The values in array A are only read by T1 and T2. Hence, transactions of type T1 will never abort (the only reason they would abort is if the write timestamp is larger than read timestamp).

(ii) For T2, the read operations on array A will always succeed. A write operation aborts if the read timestamp is higher than the write timestamp. However, arrays in B are never read. Hence, T2 will never abort.

(iii) Since array B is never read, all operations of transaction type T3 will succeed. Hence, they will never abort.

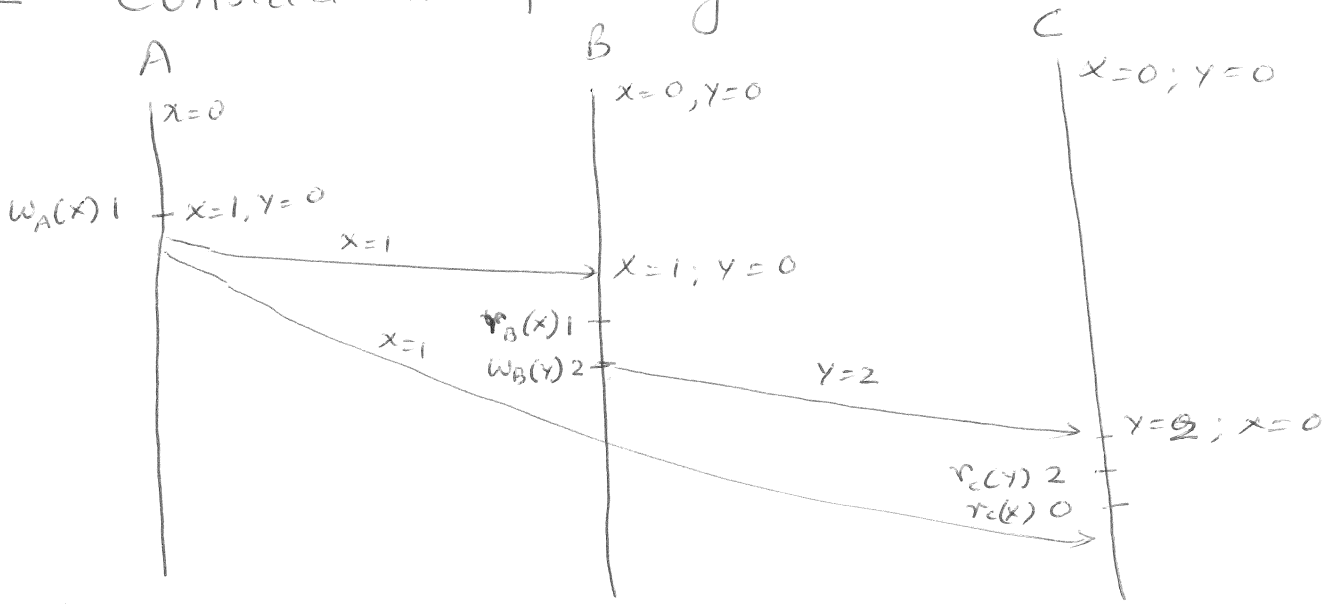
b.

(i) Since transaction of type T1 only read elements in A and all other transactions also read elements in A, there will be no waiting for read locks. So, transactions of type T1 will never wait for a lock.

(ii) The common elements between transactions of types T1 and T2 are elements in A, and both of them read these elements. Hence, transactions of type T2 will not wait for a lock acquired by T1.

c. We can simplify 2-phase locking as follows: Transactions of type T1 do not acquire locks (that is, they can read elements of A without locking). Transactions of type T2 do not lock elements in A, but they do need to acquire the lock on the element that they write in B. Transactions of type T3 will follow the standard two phase locking protocol.

Q2 Consider the following scenario



- we have the following sequence

A B C

$w_A(x)$ 1

$r_B(x)$ 1

$w_B(y)$ 2

$r_c(y)$ 2

$r_c(x)$ 0

There is no linear sequence equivalent to this sequence.