1.

R1	R2	R3
10	2	7

Claim

Claim	R1	R2	R3
P1	3	1	4
P2	6	1	3
P3	3	2	2
P4	4	2	2

All	R1	R2	R3
P1	2	1	1
P2	5	1	1
P3	2	0	1
P4	0	0	2

Need	R1	R2	R3
P1	1	0	3
P2	1	0	2
P3	1	2	1
P4	4	2	0

R1	R2	R3
1	0	2

Need	R1	R2	R3
P1	1	0	3
P2	0	0	0
P3	1	2	1
P4	4	2	0

R1	R2	R3
6	1	3

Need	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	2	1
P4	4	2	0

R1	R2	R3
8	2	4

Need	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

R1	R2	R3
10	2	7

2.

The banker's algorithm is $O(n^2 * m)$ complexity for the number operations preformed. Once a process is selected each resource needs to be updated in the available resources array. In the worst case every process has to be checked before a process is found that satisfies N(I, j) <= W(i) condition. This ends up being n + n-1 + n-2…+1 comparisons, which is O(n * n), since there are m resources it ends up being O(m * n * n) => $O(m * n^2)$

3.

a.

V1({A, B}. {A}) → succeeds. No other validations have occurred
V3({C, D}, {B}) → succeeds, read set of T3 and write set of T1 do not intersect, and write set of T3 and write set of T1 do not intersect
V2({B, C}, {A}) → T1 has completed but completed after T2 started; however, the read set of T2 and the write set of T1 do not intersect. T3 has not yet completed. The write sets of T3 and T2 do not intersect, so no W-W conflict. However, the read set of T2 intersects with the write set of T3, thus there is a R-W conflict and the validation fails.

b.

V1({A, B}. {A}) → succeeds. No other validations have occurred
V3({C, D}, {B}) → succeeds. Same as before.
V2({B, C}, {A}) → W-R conflict. The write set of T2 intersects with the read set of T2.

4.

a.

Transaction A	Transaction B
Acquire read lock on x	
	Acquire read lock on z
x <= 0 blocks trying to acquire write lock on z	
	Acquires read lock on y
	Blocks trying to acquire write lock on x

b.

```
Transaction A{
request read lock on x;
read x;

if x > 0
{
        request write lock on y;
        write y;
        release write lock on y;
}
else
{
        request write lock on z;
        write z;
        release write lock on z;
}
release read lock on x;
}

Transaction B{
        request write lock on x;
        request read lock on y;
        request read lock on z;

        read z;
        read y;
        write x;

        release write lock on x;
        release read lock on y;
        release read lock on z;
```

}

c. A transaction could block waiting for an object just because Ri < Rj, when it could be working on available objects that it actually needs first, which is obviously inefficient.