7.15 Compare the circular-wait scheme with the various deadlock-avoidance schemes (like the banker's algorithm) with respect to the following issues:

(a) Runtime overheads

(b) System throughput

By using a deadlock-avoidance scheme there is obviously an increased runtime overhead vs not using one (such as manually attempting to avoid deadlock, which has virtually no runtime avoidance.) This is because deadlock-avoidance schemes will need keep track of allocated resources, maximum needed resources, total available resources etc, and must run extra checks before allocating resources by checking whether or not it would introduce a state that could induce deadlock. Again, these are all things a static method would not need to do.

However, at the cost of this extra runtime overhead, a deadlock-avoidance scheme will be able to increase the systems throughput versus a static method. This is because a deadlock avoidance scheme will be able to more optimally and dynamically manage resources for any given situation, thus maximizing resource usage allowing more processes to run more efficiently.

7.16 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, and new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumastances?

(a) Increase **Available**.
This can pretty much always be made, if the system is in a safe state, then there is sufficient resources for all processes introducing more resources does not change this. The sequence of process which justifies the safe state of the system will continue to justify the safe status with more available resources. Note that by introducing more resources, there may be more possible ordering of the processes which maintain a safe state.

(b) Decrease **Available**.
This is much more risky, if resources are removed, the system will need to be reevaluated, and it is possible that the system will enter a deadlock. For example, at an extreme, if at least one process needs resources, and you remove all resources, you have now introduced deadlock. There are situations where decreasing available resources may be okay, for example if you have a safe state which you add n (of the same) resources too, and then remove n, you are back where you started which was initial safe, thus you are still safe.

(c) Increase **Max** for one process.
This could introduce deadlock. The trivial example is if you increase resources beyond what is available. Another example might be if this process is the only process that could run first in the safe sequence, but by increasing its maximum required resources, you could introduce a situation where it no longer fits that role, no process can run first, thus no such sequence exists anymore, therefore the system is no longer in a safe state and deadlock is possible.

(d) Decrease **Max** for one process.
This will always be fine, if the system is in a safe state, having a process require less of a particular resource is never a bad thing. By doing this, more possible orderings of processes may maintain a safe state, in particular, the process which requires less maximum resources may be able to be executed earlier in the sequence than it was before decreasing maximum resources.

7.18 Consider a system consisting of m resources of the same type being shared by n processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold.

(a) The maximum need of each process is between one resource and m resources.

(b) The sum of all maximum needs is less than $m + n$.

Proof. If a process requires more than m resources ($\neg(a)$), then we can imagine a scenario where that process holds all m resources at once and creates deadlock as it waits for the more (not available) required resources. In this sense, $\neg(a)$ will produce a deadlock situation, therefor (a) must hold in a deadlock free environment.

Now assume both (b) and (a) hold, and that a deadlock exists (contradiction). We can further assume that all m resources are allocated. As if they are not in deadlock state, a process needs more than available resources, thus we can allocate them to that process (but will of course still be in a deadlock state). The sum of current needs + the currently allocated = the sum of maximum needs, which is less than $m + n$ by property (a). Since we have assumed everything to be allocated, that means that current needs + $m < m + n \Rightarrow$ current needs $< n$. Thus we can conclude that there exists at least one process who does not currently need resources. Thus the system cannot yet be in deadlock as this process has resources to execute, this contradicts our assumption that we were in deadlock. Therefore (a) and (b) guarantee that the system is deadlock free.

□

7.22 Consider the following snapshot of a system:

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	3	0	1	4	5	1	1	7	2	1	0	3
P_1	2	2	1	0	3	2	1	1	1	0	0	1
P_2	3	1	2	1	3	3	2	1	0	2	0	0
P_3	0	5	1	0	4	6	1	2	4	1	0	2
P_4	4	2	1	2	6	3	2	5	2	1	1	3
$Total$	12	10	6	7								

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the processes may complete. Otherwise, illustrate why the state is unsafe.

(a) **Available** = (0,3,0,1)
Note that total memory = $(12+0, 10+3, 6+0, 7+1) = (12, 13, 6, 8)$
$P_2 \to (3, 4, 2, 2) \to P_4 \to (7, 6, 3, 4) \to P_0 \to (10, 6, 4, 8) \to P_1 \to (12, 8, 5, 8) \to$ $P_3(12, 13, 6, 8)$.
Thus we can see from the above sequence that the system is in a safe state according to the banker's algorithm.

(b) **Available** = (1,0,0,2)
Note that total memory = $(12+1, 10+00, 6+0, 7+2) = (13, 10, 6, 9)$
Initially, we can only run $P_1 \to (3, 2, 1, 2)$, next we must run $P_2 \to (6, 3, 3, 3)$ then things start getting much easier, as we have enough resources to run any process.
$P_3 \to (6, 8, 4, 3) \to P_4 \to (10, 10, 5, 5) \to P_0 \to (13, 10, 6, 9)$.
Thus again we see that following the above sequence we are able to run every process, and are therefore in a safe state according to the banker's algorithm.

7.22 What is the optimistic assumption made in the deadlock-detection algorithm? How can this assumption be violated?

The assumption is that once all tasks are in a safe state, that they will make no further requests will be made. Doing so would violate the assumption and could lead to an unsafe state, and thus deadlock.