1.

> 1. The overhead of creating the thread. Creating a thread needs extra time by system calls but a small sequential program can finish the work before that.
> 2. High Race Condition/synchronization. So many atomic operations affect the speed of the multithreaded version.
> 3. Dependencies among tasks. Tasks are hard to parallelize because they need to wait for the prior task to be finished.

2.

> (a) The program with a higher instruction count may have a lower CPI. Because CT is fixed for the same machine and $ET = IC * CPI * CT$, so the latter program has a much lower CPI such that the total cycles is smaller.
>
> (b) $ET = IC * CPI * CT = \text{total number of cycles}/\text{frequency}$, so the machine with a higher MIPS may have a higher total number of cycles and a higher number of instructions in millions. Or may have a lower frequency.
>
> (c) The program with a lower CPI may have a higher instruction count. Like the example shown in the class. The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C. The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C. The second one is faster but has a lower CPI with higher IC.

3.

> a. Two thread is used. Because the first three instruction is dependent on the previous one, which is the first thread. Then "c++;" can be the second thread.
>
> b.
>
> Actually, one thread will be better as there are only a few codes. But if make it more parallelizable, it could be:
>
> ```
> x++;
>
> //thread 1:
> a = x + 2;
> //thread 2:
> b = x + 4;
> //thread 3:
> c++;
> ```