EX three

1. Can A2 execute before B1?

Reasoning:

- Initially: `start = 0`, `done = 0`.[1]
- For A to reach A2, it must pass `P(done)`, so `done` must be 1 at that moment.
- `done` can only become 1 by `V(done)` in process B, which is executed **after** B1.
- Therefore, A2 cannot start before B1 has finished, because it is blocked on `P(done)` until B executes `V(done)` following B1.

So **no execution** allows A2 before B1; the semaphore `done` enforces B1 → A2.

2. If `V(start)` is moved after A2

New (incorrect) code:

Process A:	Process B:
A1; P(done); A2; V(start);	P(start); B1; V(done);

Execution:

- Initially: `start = 0`, `done = 0`.
- Suppose A runs first. A1 executes, then A executes `P(done)` and blocks because `done = 0`.
- B cannot start B1 because it is blocked on `P(start)` and `start = 0`.
- A will never execute `V(start)` (it is after A2, and A is blocked before A2).
- B will never execute `V(done)` (it is after B1, and B is blocked before B1).

Both processes are blocked forever: **deadlock** occurs. The required order A1 → B1 → A2 is also violated because the program never progresses.

3. Generalization to 6-step order

Goal:

$A1 \rightarrow B1 \rightarrow A2 \rightarrow B2 \rightarrow A3 \rightarrow B3$.[1]

Use three semaphores, all initialized to 0:

- `start1` controls B1 after A1.
- `start2` controls A2 after B1.
- `start3` controls B2 after A2.
- `start4` controls A3 after B2.
- `start5` controls B3 after A3.

All are semaphores; initial values: `start1 = start2 = start3 = start4 = start5 = 0`.

Process A	Process B

A1;
V(start1); // allow B1
P(start2); // wait for B1
A2;
V(start3); // allow B2
P(start4); // wait for B2
A3;
V(start5); // allow B3
	P(start1);
	B1;
	V(start2); // allow A2
	P(start3);
	B2;
	V(start4); // allow A3
	P(start5);
	B3;

This guarantees the strict global order
A1 (then signal) → B1 → A2 → B2 → A3 → B3, using only simple binary semaphores and the same pseudocode style.