Problem 1:

For the following two programs, how many times is the “Example” line printed?

a.

```
#include "csapp.h"

int main()
{
    int i;

    for (i = 3; i > 0; i--)
        Fork();
    printf("Example\n");
    exit(0);
}
```

There is a for loop that iterates 3 times and each time it enters the for loop fork doubles so 2^3 = 8 times that example is printed

b.

```
#include "csapp.h"

void try()
{
    Fork();
    printf("Example\n");
    Fork();
    return;
}

int main()
{
    try(); Fork();
    printf("Example\n");
    exit(0);
}
```

When try is initially called example will print twice because of the fork in the try function, then it will turn into 4 processes because of the 2nd fork in the try function. Following the try function it is forked again leading to 8 processes so example is printed another 8 times. This leads to a total of 10 times that example is printed.

Problem 2:

Process	Start Time	End Time
A	3	5
B	4	7
C	6	8
D	2	9

For each pair of processes, indicate whether they run concurrently (Y) or not (N):

Processes	Concurrent?
AB	Y
AC	N
AD	Y
BC	Y
BD	Y
CD	Y

Problem 3:
Consider the following program:

```c
void end(void){
      printf("2\n");
}
int main(){
      if (Fork() == 0)
            atexit(end);
      if(Fork() == 0){
            printf("0\n");
      }
      else{
            printf("1\n");
      }
      exit(0);
}
```

Determine which of the following outputs are possible. Use a process graph to show your work. Note: the **`atexit`** function takes a pointer to a function and adds it to the list of functions (initially empty) that will be called when the exit function is called.

```
                                   —---------*------/exit(2)
                                 /        printf(0)
                  —---*-----------*---------*----------/exit(2)
                /   atextit      fork   printf(1)
               /            —-------*-----------/exit
              /          /       printf(0)
—-*----------*----------*-----------/ exit
  Fork      fork       printf(1)
```

🟩 Green = possible
🟥 Red = not possible

- 🟩 A. 112002
- 🟥 B. 211020
- 🟩 C. 102120
- 🟥 D. 122001
- 🟩 E. 100212

Problem 4:
One of your colleagues is thinking of using signals to allow a parent process to count events that occur in a child process. The idea is to notify the parent each time an event occurs by sending it a signal and letting the parent's signal handler increment a global counter variable, which the parent can then inspect after the child has terminated. However, when he runs the test program in the following figure on his system, he discovers that when the parent calls printf, counter always has a value of 2, even though the child has sent five signals to the parent. Perplexed, he comes to you for help. Can you explain the bug?

```c
#include "csapp.h"

int counter = 0;

void handler(int sig)
{
    counter++;
    sleep(1); /* Do some work in the handler */
    return;
}

int main()
{
    int i;

    Signal(SIGUSR2, handler);

    if (Fork() == 0) {  /* Child */
        for (i = 0; i < 5; i++) {
            Kill(getppid(), SIGUSR2);
            printf("sent SIGUSR2 to parent\n");
        }
        exit(0);
    }

    Wait(NULL);
    printf("counter=%d\n", counter);
    exit(0);
}
```

There are signals that are sent but not received(pending signals), and there can only be one at a time, so the following signals are ignored. To fix this bug Wait() needs to be called in a different location so that all of the child processes are reaped correctly. This will handle and clear the current process so that the following processes can be handled and cleared as well.