Q1

fork() either returns a positive process ID of the child process to the parent, 0 to the child, and -1 if the fork fails.

Following the code, there's one original process initially, then the first fork() is called and creates C1:

The parent process continues to the first if block as fork() returns a positive value. Then inside this parent process, A second fork() is called, which causes the parent process again, and creates the second fork P2. Then P2 goes to the else block, printing "2 ". It also creates another child process C2, which is from the second fork inside the original parent and it goes to the if (!fork()) block. C2 then calls fork() again: c2 then prints "1 ", since it's in the if (!fork()). Child process C3 also goes to printing "1 " because it's inside the block that follows fork() without any condition.

The child C1 goes to the else block because fork() returns 0, then it goes directly to printing "3 ".

After each printf() call for "1 ", "2 ", or "3 ", there's another "4 " printed by every process before they terminate.

Therefore, the output would likely be something like:

"3 4 " from the first child C1.

"2 4 " from the parent after the second fork P2.

"1 4 "from the second child C2 and one from the child of the second child C3.

However, because the processes run concurrently, the exact order of these outputs can be different from run to run. It is probably 3 4 2 4 1 4 1 4, but can be 2 4 3 4 1 4 1 4 or any other order.

Code:

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("\n\n");
    if (fork()) {
        if (!fork()) {
            fork();
            printf("1 ");
        } else {
            printf("2 ");
        }
    } else {
        printf("3 ");
    }
    printf("4 ");

    return 0;
}
```

Output:

```
2 4 3 4 1 4 1 4
student @ COMP2401 : 03:53:14
● ~/Desktop/Assignments/A4/output # ./"test"


3 4 2 4 1 4 1 4
student @ COMP2401 : 03:53:14
● ~/Desktop/Assignments/A4/output # ./"test"
```

Q2

Given that the list is 1, 2, -3, -4, 9, -1, and that head is 1, following the code:

Iteration 1:

Prev = 1
current = 2 as head.next = 2

2 >= 1 is true, so prev = current = 2
skip else

Iteration 2:

Prev = 2
current = -3 as 2.next = -3

-3 >= 2 is false, go to else

Prev.next = current.next (2.next is now -4)
current.next = head = 1 (-3.next is now 1)
head = current = -3 (head is now -3)
current = prev = 2

Which means the list is now: -3 1 2 -4 9 -1

Iteration 3:

Prev = 2
current = -4 as 2.next is -4

-4 >= 2 is false, go to else

Prev.next = current.next (2.next is now 9)
current.next = head = -3 (-4.next is now -3)
head = current = -4 (head is now -4)
current = prev = 2

Which means the list is now: -4 -3 1 2 9 -1

Iteration 4:

Prev = 2
current = 9 as 2.next is 9

9 >= 2 is true, so prev = current = 9
skip else

Iteration 5:

Prev = 9
current = -1 as 9.next is -1

-1 >= 9 is false, go to else

Prev.next = current.next (9.next is now null)
current.next = head = -4 (-1.next is now -4)
head = current = -1 (head is now -1)
current = prev = 9

Which means the list is now: -1 -4 -3 1 2 9

Iteration 6:

Prev = 9
current = null as 9.next is null

Exit the loop

Therefore the final output would be {-1, -4, -3, 1, 2, 9}