Q1. Describe how Linked List is implemented in Pintos.

In Pintos, a list contains two list_elem which are head and tail respectively while each linked list element contains pointers to its previous and next elements and thus a linked list does not use dynamic memory allocation. Particularly, a list_elem can only be part of one list at a time since its previous and next list_elem are confirmed.

To construct a list with specific type, for example Integer, we use int_list_elem which contains an integer and a list_elem.

For construction, we declare and initialize a list and use list_push_back() method to push a list_elem, instead of int_list_elem, into this list. Method list_entry() can get the integer value by using the offset of int_list_elem and list_elem.

Methods for sort and insert are also available with provided list element comparison function.

Q2. How to use integer to simulate real number calculation(Suggested examples)?

We can use fixed-point number representation to use integer to simulate real number.

For example, for a 32-bit integer, we divide it into 3 parts which are one sign bit, 15 bits ahead of the decimal point and the remaining rightmost 16 bits for fraction part. In this case, a real number times 2^16 will be used in Pintos to represent this number and the maximum integer representation 32-bit 1 in Pintos becomes (2^31 -1)/(2^16) for real number value.

Many operations, like addition and substraction, on fixed-point numbers are straightforward. While for multiplication and division we have to divide or multiply an extra 2^16 to rebalance the results and sometimes use 64-bit operation in case of overflow.

Suppose x, y are fixed-point representation. For addition and subtraction, it is (x+y) and (x-y) while for multiplication and division, it is ((int64_t)x*y/2^16) and ((int64_t) x*2^16)/y.

Q3. Why can multi-level feedback scheduling avoid starvation?

Thread priority is recalculated once every fourth clock tick according to
priority = PRI_MAX − (recent_cpu/4) − (nice×2).

The scheduler has 64, from 0 to 63, priorities with 63 representing the PRI_MAX which has the highest priority.

Integer "nice" denotes how nice a thread is compared to other threads with a positive nice value cause a thread decreasing its priority and a negative one receiving more CPU time.

"recent_cpu" is an estimate of the CPU time the thread has used recently. It can preventing starvation since a thread that has not received any CPU time recently will have a recent_cpu of 0 and thus it will not decrease its priority because of recent_cpu and receives CPU time soon while threads that have recently been scheduled on the CPU will have a lower priority the next time the scheduler picks a thread to run.