Report of Question 1:

Solution to the first question:

Law of Violence: For each person, go through all the people behind and check for reverse pairs. The time complexity is O(n^2). This approach is very simple, but can become very slow when dealing with large queues. For example, if there are 1,000 people in the queue, 499,500 pairs need to be checked. Therefore, the brute force method is only applicable to small cohorts.

Merge sort: During the merge sort process, the number of reverse pairs can be counted. The time complexity is O(nlogn). This method is faster than the brute force method, but requires more code. Specifically, we can use divide and conquer to divide the queue into two subarrays and recursively calculate the reverse logarithm in each subarray. We can then compute the reverse logarithm across the two subarrays. Finally, we merge the two subarrays and calculate the reverse logarithm between them.

Tree arrays: Use tree arrays to maintain prefix sums and then count the number of reverse pairs. The time complexity is O(nlogn). This method is faster than merge sort, but can be more difficult to implement. Specifically, we can use a tree array to maintain the number of occurrences of each height. We can then use the prefix and count the number of heights that appear before each height. Finally, we can count the number of reverse pairs.

I choose the way to solve the first question:

Merge sort is a kind of divide-and-conquer sorting algorithm, its core idea is to break down a big problem into a small problem, divide a large array into several small arrays, and then merge a little bit. Specifically, merge sort divides the array to be sorted into two subsequences containing n/2 elements each, then recursively sorts the two subsequences, and finally merges the two sorted subsequences to get the final sorted sequence. During the merge sort process, the number of reverse pairs can be counted. The time complexity is O(nlogn). This method is faster than the brute force method, but requires more code. Specifically, we can use divide and conquer to divide the queue into two subarrays and recursively calculate the reverse logarithm in each subarray. We can then compute the reverse logarithm across the two subarrays. Finally, we merge the two subarrays and calculate the reverse logarithm between them.

Advantages of merge sort algorithm:

Merge sort is a comparison-based sorting algorithm with a time complexity of O(nlogn), where n is the number of elements to be sorted. The advantage of merge sort is that it guarantees that the time required to sort any array of length n is proportional to nlogn. In addition, merge sort is also a stable sorting algorithm, that is, in the sorting process of the same size elements can maintain the order before sorting. Both the brute force method and the tree array method require O(n^2) time complexity, so merge sort is more efficient when dealing with large data.

Question 2:

Solution:

Renko Usami’s problem can be solved using the properties of Pisano Period. For a given value of N and M >= 2, the series generated with Fi modulo M (for i in range (N)) is periodic. The period always starts with 01. The Pisano Period is defined as the length of the period of this series. To compute F(N) mod M, we need to find the remainder when N is divided by the Pisano Period of M. Then calculate F(N)remainder mod M for the newly calculated N. Here are three ways to calculate fn mod m:

Method 1: Recursion method

Recursion is a simple method, but its time complexity is O(2^n), so it is not suitable for large-scale computation. The basic idea of this method is to convert the calculation of fn mod m to the calculation of f(n-1) mod m and f(n-2) mod m. Specifically, we can define a recursive function that takes n and m as arguments and returns the value of fn mod m. In the function, we first check if n is less than or equal to 1. If so, return f0 or f1 mod m. Otherwise, we calculate fn mod m as (af(n-1) + bf(n-2)) mod m.

Method 2: Matrix multiplication

Matrix multiplication is a faster method with a time complexity of O(log n). The basic idea of this method is to convert the calculation of fn mod m to the form of matrix multiplication. Specifically, we can define A 2x2 matrix A = [[a, b], [1, 0]] and a 2x1 vector V = [[f1], [f0]], where f0 and f1 are known values. Then, we can get fn mod m by calculating A^(n-1) \* V.

Method 3: Fast power

Fast exponents are a faster method with a time complexity of O(log n). The basic idea of this method is to convert the calculation of fn mod m to the form of power operation. Specifically, we can define a function powmod(x, y, z) that takes three arguments x, y, and z and returns the value of x^y mod z. We can then get the values of f(n-1) mod m and f(n-2) mod m by calling powmod(f0, n-1, m) and powmod(f1, n-2, m). Finally, we can get the value of fn mod m by calculating (af(n-1) + bf(n-2)) mod m.

The above three methods can be used to calculate fn mod m, but they each have advantages and disadvantages. Recursive method is easy to understand, but the time complexity is higher; Matrix multiplication and fast exponents are both faster than recursion, but require more code to implement.

Choose matrix multiplication:

Matrix multiplication and fast powers can reduce a lot of computation, and thus solve the problem quickly.