A. Sum of Squares

Solution:

A naive solution is using a loop from 1 to n and calculates the sum of them.

Time complexity O(n)

Notice that T is up to 1000 and n is up to 10^6, we will get TLE by using this solution.

A faster solution is, using the formula $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$, we can get the solution in O(1) time complexity. Notice that n*(n+1)*(2n+1) $\approx 10^{18}$ > max_int, we should use long long (or long in JAVA) to avoid overflow.

B. Binary Search

Solution:

We can use the basic binary search to check whether a number is in a sorted sequence.

Time complexity O(n) (Since we need to read the data)

C. Index Twin

Solution:

Using the idea before, we transform a[i] + a[j] = m to a[j] = m – a[i], that means for every a[i], we want to find whether number m – a[i] is in a sorted sequence. This is the problem B.

Time complexity O(nlogn)

D. Index Triple

Solution:

Using the same idea as problem C. We transform a[i] + a[j] + a[k] = m to a[k] = m – a[i] – a[j], that means for every a[i] + a[j], we want to find whether number m – a[i] – a[j] is in a sorted sequence. This is also problem B. But we need to find all a[i] + a[j]. We can using Brute Force(two for loops) to get this.

Time complexity O(n^2logn)

E. Monopoly

Solution:

After analyzing the problem, we can find the answer f[i] = $a[i]^p$ or f[i] = $a[i]^{(p+1)}$. Where p = K / N. So the problem becomes how to calculate a^b. We can use a for loop to do this, but notice that K is really large. This solution will get TLE.

Time complexity O(K / N)

There is a faster way to calculate power by using the idea of binary search. If b is an even number, we have $a^b = a^{2^{\frac{b}{2}}}$, if b is an odd number, we get multiply the result by a, then b = b-1 is an even number. So each time let b = b/2, a = a * a. If b is odd we multiply the result by a. We can get the result in logb times multiplication. We also need to deal with overflow, this is based on the problem A.

Time complexity O(log(K / N))

F. CS203 Seat Arrangement

Solution:

Like cut ropes in Lab1, we can use binary search for the largest minimum distance. But we also need to notice one thing: The first seat must sit a student. You can just think in this situation, the space we can use is max. Why not the last seat? Actually, the first seat and the last seat is equivalent. If the distance is correct. You can move the last student to any end of seat. So the problem is how to check a distance D is correct. We can use a loop, and an integer to record how many student we have arranged. We put a student by distance larger than D. When we reach the end of seat, just check the students we have arranged is K or not.

Time complexity O(n)

G. Find Score X in Lab1 in CS203

Solution:

Like problem B, we have to find the left most number we want. When we find a number equals to m, we just find the left area.

Time complexity O(qlogn)

H. Cuperman’s Cup Game

Solution:

If we start at the final situation, we can notice that if we start at (2x, 0), we at most move log(2x) steps. Since every situation we start must be one of these log(2x). So if we move more than log(2x) steps the result is -1. So we can just use recursion to solve this problem. Time complexity O(log(n + m))