Problema 1

Se crea 26 priority_queue, uno por letra. Puede guardar en un $unordered\_map < char, priority\_queue >$ o puede usar un arreglo de 26 elementos.

Se crea $unordered\_map < char, int >$ de las prioridades extras recibidas por sorteo.

Al llegar nuevo cliente: insertar al priority queue de la letra correspondiente con prioridad (prioridad original - prioridad de la letra).

Al sortear letra: sumar prioridad a la prioridad de letra.

Al servir: sacar los 26 top del priority queue, sumarles la prioridad de la letra correspondiente. Luego, sacar el máximo para retornar su nombre, y quitar el máximo del priority queue.

Problema 2

Se crea $unordered\_map < std :: string, vector < int >>$. Se itera todos los strings y empuja la posición encontrado en el map.

Problema 3

La idea es usar std::sort pasando una función que compare de acuerdo a número de divisores. Calculando números de divisores en esa función puede ser lento. Por lo tanto se recomienda generar una tabla de números dados, al número de divisores en un unordered_map. Para generar dicha tabla, es útil tener la tabla de números primos hasta 10^5 que se generó en la sesión 2.

Problema 4

S incluye los números que aparecieron en posición 1 del input pero nunca aparecieron en posición 2. T incluye los números que aparecieron en posición 2 pero nunca en posición 1.

Para este tipo de problemas, se usa un unordered_set y se inserta cuando lo ve un número en posición 1 luego lo quita si lo ve en numero 2.

Problema 5

Puede usar $unordered\_map < int, int >$ donde el value es el número de veces encontrado tal número.

Problema 6

Usar un set, si ve un numero A, lo mete al set. Y luego, se busca si $suma - A$ existe en el set, y si existe entonces ese par es el deseado.

Ten cuidado cuando $A + A = suma$ entonces solamente hay un par válido si efectivamente hay 2 A dentro del input.