Problema A
Pánico en el Supermercado

Con el nuevo brote de Culonavirus, el gobierno ha declarado una cuarentena general. En pánico por lo que acaba de pasar, la gente (Tú incluido) se ha armado de valor, con guantes y mascarilla, y ha salido a comprar al supermercado, para coger provisiones de comida para los próximos días. Ya en el supermercado, al abrir las puertas has empezado a correr con tu carrito y has echado lo primero que has encontrado en las baldas. No has mirado ni qué era, ni si te gustaba o no, ni como venía. Tu has comprado lo que has podido y directo a casa. El problema lo has encontrado cuando, al llegar a casa, has visto la despensa. Tienes apenas un par de armarios y de baldas, y has hecho espacio como has podido. Ahora te preguntas si podrás meter toda la compra en la despensa. Por suerte, algunos productos vienen en packs, así que esos productos has decidido dividirlos de forma individual y meterlos en la despensa divididos. A fin de cuentas, no te importa compartir una lata de maiz con la vecina con tal de poder colocar las cosas más fácilmente. Dada la lista de productos que has comprado, lo que ocupa cada uno, y si son o no divisibles, junto a la lista de huecos en la despensa, ¿Podrías decirnos si cabe la comida en la despensa, o si por el contrario, tienes que regalar comida a tus vecinos?

Input

La entrada estará compuesta por un primer número C, indicando el número de casos de prueba, cada uno comenzando por un número N de productos comprados. A continuación, acompañarán N líneas, cada una con dos números, T y D. El primero, T, es el tamaño de un producto completo, y a continuación D, el número de veces que se puede dividir ese producto. ($D = 1$ significa que el producto es indivisible). Se garantiza que todas las divisiones de productos serán exactas. Después aparecerá un número H indicando el número de huecos en la despensa, seguido de H líneas, cada una con el tamaño de cada hueco en la despensa. Solo se puede colocar un producto (O una división de un producto) en cada hueco de la despensa.

La entrada debe ser leída de forma estándar.

Output

Para cada caso de prueba, la salida será una sola línea indicando "SUFICIENTE HUECO" si cabe todo en la despensa al colocar los productos, o "REGALAR COMIDA A LA VECINA" si los productos no caben en la despensa.

La salida debe ser escrita de forma estándar

Entrada ejemplo	**Salida ejemplo**
2 5 6 1 7 1 8 4 9 3 2 1 10 2 2 2 6 3 3 2 3 7 2 2 16 2 4 1 2 10 20	SUFICIENTE HUECO REGALAR COMIDA A LA VECINA

Constraints

- $1 \leq N \leq 10000$
- $1 \leq D_i \leq T_i \leq 10000$
- $0 \leq H \leq 300000$
- $1 \leq H_i \leq 10000$

Problema B
Ordenando con Bubble Sort

El primer algoritmo para ordenar arrays que enseñan en cualquier lado es el infame *Bubble Sort*; hace su trabajo, pero es muy lento si se quiere ordenar algo grande. Hasta ahora era la única manera que conocía John von Neumann de ordenar sus array, pero recientemente en el curso de programación competitiva le han explicado el *Merge Sort*.
Ahora John quiere comprobar cuánto mejora el nuevo algoritmo respecto al anterior. Para ayudarse con los cálculos necesita varias métricas, una de ellas es el número de intercambios que realiza el *Bubble Sort* al ordenar un array dado.
Repasando los apuntes ha visto que el *Bubble Sort* que le han explicado recorre el array de izquierda a derecha comparando cada posición con su siguiente; si el valor de la segunda posición es menor al de la primera se intercambian los números. Así, después de una pasada se asegura que el valor más grande queda en la última posición. Este proceso se repite, ignorando cada vez un elemento más del final del array, hasta quedarse con un único elemento.

Input

La entrada comenzará con una línea indicando el número de casos de prueba T.
Cada caso de prueba constará de dos líneas siquiendo el siguiente formato:

- El número de elementos del array N.
- N números V separados por espacios, describiendo el array.

La entrada debe ser leída de forma estándar.

Output

Por cada array hay que imprimir una línea indicando el número de intercambios que hace el *Bubble Sort* que le han explicado a John al ordenar el array.

La salida debe ser escrita de forma estándar

Entrada ejemplo	**Salida ejemplo**
2 6 -4 -6 -9 -12 -13 -15 6 1 -6 6 9 10 -15	15 6

Entrada ejemplo	**Salida ejemplo**
1 4 -20 -8 14 -16	2

Constraints

- $1 \leq T \leq 5$
- $1 \leq N \leq 1000$
- -10000 $\leq V_i \leq$ 10000
- $V_i \neq V_j \ \forall \ i, j$

Problema C
Tim el hambriento

Tim quiere salir a comer pero últimamente todos los restaurantes de la ciudad están tardando demasiado en servir sus platos, por lo que ha tenido una idea para tener que esperar el menor tiempo posible para disfrutar de su comida. Va a pedir los platos que más sacien su apetito hasta que quede satisfecho, incluso si esto le hace no poder acabar el último plato.
En primer lugar Tim analiza los platos del menú para determinar cuanto sacian su apetito en una escala que ha llamado Timones.
Te han pedido que escribas un programa que le ayude a determinar el número mínimo de platos que tiene que pedir en cada restaurante para saciarse; para ello te ha dado la cantidad de restaurantes que quiere visitar, los Timones que necesita para saciar su apetito al llegar al restaurante y cuantos Timones sacian los platos del menú.

Input

La primera linea contiene el número de restaurantes que visitará Tim R.
Después cada restaurante empezará una linea con dos números, H indicando la cantidad de Timones necesarios para satisfacer su hambre en ese restaurante y P el número platos del menú del restaurante.
La siguiente linea contiene P valores con los Timones que sacia cada plato del menú

La entrada debe ser leída de forma estándar.

Output

En la salida tienes que indicar para cada restaurante el número mínimo de platos que necesita para satisfacer su hambre H.
Si Tim no consigue satisfacer su hambre se debe mostrar *Sinpa*, ya que tiene menos vergüenza que comida en el estomago y decide salir corriendo sin pagar.

La salida debe ser escrita de forma estándar

Entrada ejemplo	**Salida ejemplo**
2 10 3 1 3 1 10 4 6 1 2 4	Sinpa 2

Constraints

- $1 \leq R \leq 20$
- $1 \leq H \leq 100000000$
- $2 \leq P \leq 150000$
- $0 \leq P_i \leq 10000$
- La suma de todos los P es inferior a 500000 para cada entrada

Problema D
Becario Precario Binario

A Donald Knuth por fin le han contratado en su consultora favorita, pero sus compañeros de trabajo han decidido darle la bienvenida con una tarea muy tediosa; le tocará reiniciar una máquina *legacy*, es decir, que nadie sabe como funciona.
Para reiniciarla correctamente tiene que ir transfiriendo e instalando archivos en el orden que se indica en el pósit pegado en la caja de la propia máquina. El problema es que para conseguir los archivoa a instalar tiene que bajar con una memoria USB al sótano del edificio y copiarlos a mano desde *El Almacen*, otra máquina, más antigua que el propio edificio, en la que se guardan los vestigios de todos los proyectos.
Ya que ir hasta *El Almacen* y volver es en sí un reto, solamente puede realizar K viajes en un día. Para colmo, sus compañeros no le han dado un *pen drive* para copiar los datos, así que tendrá que comprar uno con su propio dinero, pero como acaba de empezar a currar tampoco tiene mucho dinero para gastar. Tiene exactamente un día para reiniciar la máquina y quiere saber la capacidad mínima de la memoria USB que le permitirá copiar todos los datos en K viajes, ya que, al fin y al cabo, cuantos más megas, más euros le costará.
Una cosa a tener en cuenta es que no se venden memorias de menos de 16 MB, ya que no resultaban rentables.

Input

La entrada comenzará con una línea indicando N, el número de archivos a copiar, y K el número de viajes que puede hacer en un día.
A continuación vendrán, en orden, los tamaños de los ficheros (en MB), T_i, que tiene que copiar.

La entrada debe ser leída de forma estándar.

Output

Solamente hay que imprimir un número entero, el tamaño de memoria USB (en MB) mínimo que debe comprar Donald para completar su tarea con éxito.

La salida debe ser escrita de forma estándar

Entrada ejemplo	**Salida ejemplo**
6 3 24 2 22 2 10 31	34

Entrada ejemplo	**Salida ejemplo**
`10 4` `18 46 43 6 58 29 24 25 18 45`	`107`

Constraints

- $1 \leq N, K \leq 100000$
- $1 \leq T_i \leq 1000$

Problema E
Elecciones en la URJC

Este año en la URJC ha habido un gran número de elecciones, por lo que el equipo informático de la universidad se ha puesto a trabajar para recopilar los resultados. Han elaborado un sistema informático al que pueden acceder los responsables de cada campus para introducir los resultados.
Sin embargo, por la rapidez con la que se ha tenido que elaborar este sistema, no se ha terminado de pulir muy bien, y en cada campus lo están utilizando de una forma.
Por un lado, algunas personas están metiendo el número exacto de votos que ha conseguido cada candidato, mientras que otras han decidido introducir el porcentaje de voto que ha conseguido cada uno. Y por si eso fuera poco, en algunos campus el trabajo se ha quedado a la mitad, lo han hecho mezclando los dos métodos al haberlo hecho dos trabajadores distintos, e incluso hay casos donde ni siquiera han terminado de introducir los resultados.
Al menos tenemos la certeza de que los datos ya introducidos al programa son exactos, ya que cada candidato solo puede aparecer una vez por campus. Además, los trabajadores se han asegurado al meter los resultados de cada candidato, que no falta ningún voto por contar para ese candidato en concreto.
Con los resultados actuales, y teniendo en cuenta que, en cada campus, los votos sin contar todavía se le podrían asignar a cualquier candidato cuyos votos no se hayan introducido todavía en ese campus. ¿Podría determinarse un ganador exacto? En caso contrario, ¿Quiénes podrían haber ganado?

Input

La entrada se compone de un único caso de prueba. Cada caso de prueba comienza por un número C de candidatos en las elecciones, seguido de C líneas con los nombres de cada candidato.
A continuación aparecerá un número N de campus que han participado en las elecciones.
Para cada campus, la entrada contendrá en primer lugar, una línea con el nombre del campus, un número P con la cantidad de resultados introducidos en ese campus, y otro número V con la cantidad de votos totales correctos contados en ese campus. Posteriormente aparecerán P líneas con el nombre del candidato, seguido del número K_i o porcentaje L_i de votos recibido. En caso de aparecer un porcentaje, este será un número entero sin decimales.
Los nombres de candidato y de campus son strings que solo contienen letras mínusculas del alfabeto inglés. Se garantiza que el porcentaje de voto será siempre exacto en relación con el número de votos, y también se garantiza que ningún campus introducirá más votos de los totales contados para dicho campus.

La entrada debe ser leída de forma estándar.

Output

La salida tendrá en primer lugar el número de candidatos posibles, acompañado de los candidatos con opciones a ganar, en orden alfabético. En caso de empate entre dos o más de los candidatos, todos ellos serán candidatos con opciones a ganar.

La salida debe ser escrita de forma estándar

Entrada ejemplo	**Salida ejemplo**
4 alvaro maria hector pablo 3 arguelles 3 50 alvaro 10 pablo 5 maria 20% vicalvaro 4 20 alvaro 25% maria 10% hector 45% pablo 20% mostoles 2 100 hector 30 maria 20	2 alvaro hector

Constraints

- $1 \leq C \leq 10000$
- $1 \leq N \leq 300$
- $0 \leq P \leq C$
- $1 \leq V \leq 3000000$
- $0 \leq K_i \leq V$
- $0 \leq L_i \leq 100$