Problem 1. Date Matching Problem

(Time Limit: 2 seconds)

Problem Description

In the internet developed modern era, the service of social network has become inseparable part of our lives. Dating websites are very popular in the service of community networks. In general, the mating websites need users to join them to be memberships. At the same time, we may also need to introduce ourselves, the city where we live, age, habits...etc. After we join the website, we could enjoy the service of making friends, and also use website to filter our target before we looking for friends, such as entering the age range, living city, interest, height and weight. Here we practice to design an efficient matching system.

There are N boys and N girls, and we want to find a matching, i.e., pairing each boy with a girl. Of course each boy/girl cannot be in two pairs. For the i-th boy and the j-th girl, a score P[i][j] can be obtained if they are paired. Instead of finding a matching of maximum total score, we only want to find a matching by the following greedy method: iteratively finding the maximum score P[i][j] of the unmatched boys and girls and then pair the i-th boy with the j-th girl. Please write a program to find such a matching.

The following is an example of the scores. The greedy algorithm first finds the score 120 and pairs boy 1 with girl 2. In the next iteration score 80 is chosen, and finally score 0 is chosen. The total score is 120+80+0=200. Note that the best possible total score is 100+90+80=270 in this example.

Boy\Girl	1	2	3
1	100	120	50
2	0	90	70
3	20	40	80

Input Format

The first line of input is the number of test cases. Each test case start with a line containing N, and the score matrix is followed in the next N lines, where N is a

positive integer at most 250. The scores are nonnegative integers no more than 250000. Note that the scores are all different in a test case.

Output Format

For each test case, output the score of the greedy matching in one line.

Example

Sample Input:	Sample Output:
2 3 100 120 50 0 90 70 20 40 80 2 1 2 3 4	200 5

Problem 2. Community Detection in Social Media

(Time Limit: 1 seconds)

Problem Description

Social media are computer-mediated tools that allow people to create, share or exchange information through Online Social Networks (OSN). Community structure is one of the most commonly observed features of OSN in reality. In the context of social networks, community structure refers to the occurrence of groups of nodes that are more closely related internally than with the rest of the network. Formally, a social network is represented as a weighted graph (V, E), where V is the set of individuals in the social network, $E : V \times V \rightarrow \aleph$ is a function that assigns to each pair of individuals a nonnegative integer that denotes the proximity between them. For any two group of individuals $G_1, G_2 \subseteq V$, we define the proximity between them as $P(G_1,G_2)=\max\{E(v_1,v_2)|v_1 \in G_1, v_2 \in G_2\}$. For any $k \leq |V|$, a k-community of the social network is a partition $\mathbf{C}=\{V_i \subseteq V|1 \leq i \leq k\}$ such that $V_i \cap V_j = \varnothing$ for any $i \neq j$ and $\cup_{i=1}^{k} V_i = V$. To measure if $\mathbf{C}=\{V_i \subseteq V|1 \leq i \leq k\}$ is a well-partitioned community structure, we can defined its proximity coefficient as $P(\mathbf{C})=\max\{P(V_i,V_j)|\ 1 \leq i < j \leq k\}$. Obviously, $P(\mathbf{C})$ measure the maximal proximity of different communities in the partition and it is expected that a well-partitioned community structure should have low proximity coefficient. Therefore, for a given social network (V,E) and an integer $2 \leq k \leq |V|$, the objective is to find the minimal proximity coefficient that k-communities of the social network can achieve, i.e., $\min\{P(\mathbf{C})|\ \mathbf{C}$ is a k-community of (V, E)$\}$.

Technical Specification

- The number of individuals in a social network is from 2 to 1000.
- The number of communities is from 2 to $|V|$.

Input Format

The first line is an integer which indicates the number of test cases. Each test case consists of several lines. The first line of a test case contains 3 numbers *k*, *n*, and *m*, where *n* is the number of individuals in the social network, *m* is the number of individual pairs with non-zero proximity values, i.e., $m=|\{(i,j)|E(i,j)>0\}|$, and *k* is the number of communities to be found. The second to the *m*+1st lines of a test case specify the non-zero values of the proximity function, each of which contains 3 numbers *i*, *j*, *e* ($0\leq i\neq j<$n and $e>0$) such that *e* denotes the proximity between *i* and *j*, i.e., $E(i,j)$.

Output Format

For each test case, output the minimal proximity coefficient of *k*-communities of the given social network.

Example

Sample Input:	**Sample Output:**
2 3 6 4 0 4 9 0 5 14 1 3 31 1 5 5 4 11 11 0 1 6 0 6 23 1 3 23 1 10 5 2 3 24 2 5 11 2 9 55 3 5 9 3 10 8 7 10 7 8 9 5	5 6

Problem 3. Treasure

(Time Limit: 3 seconds)

Problem Description

Alice and Bob are good friends, and they always like to venture together. One day they find out a bag of treasure, with various kinds of precious gems and jewels inside. They want to share it with each other, while ensuring partitioning as fair as possible. However, they have no idea about how to divide these treasures into two sets fairly, and thus, they come to you, a master in algorithm and programming, to help them out.

As you may guess, the same treasure may bring different feelings to different people. For example, a beautiful rose-shaped transparent gem may fascinate Alice, while Bob may prefer the golden knife-shaped one. In order to make the allocation as fair as possible, they first give each treasure a pair of values $\boldsymbol{a_i}$, $\boldsymbol{b_i}$, denoting the willingness of Alice and Bob if he or she receive the i^{th} treasure. Then you will need to find out a partition that minimizes the difference of total willingness Alice and Bob get finally.

Furthermore, they expect the number of treasure one get differs no more than one to the other person. For example, if there are five treasures, then Alice gets two of them and Bob gets three of them; or alternatively, Alice gets three of them and Bob gets two of them.

For instance, given a bag with three treasures, the willingness of Alice to these treasures are (a_1, a_2, a_3) = (3, 5, 4), and the willingness of Bob are (b_1, b_2, b_3) = (2, 7, 3). One of optimal allocations to minimize the difference of total willingness of Alice and Bob is that Alice takes the second one, and Bob takes the first and the third one. In this case, both of them get willingness of five in total, resulting in zero difference. Note that there may be several allocations which meet the requirements, and you only care about the difference of total willingness, rather than the total amount of willingness they get. As you can see, Alice can also get the first and the third one, and Bob get the second one, resulting in zero difference as well, where both of them get seven willingness in total.

The problem becomes as follows. Given totally $\boldsymbol{n}$ treasures in the bag, each with the willingness of Alice and Bob, help them to find out an optimal allocation such that

the difference of total willingness of Alice and Bob is minimized, under the condition that the total number of treasure one finally gets differs no more than one to the other person.

Technical Specification

- The number of treasures $\boldsymbol{n}$ will be in the range: $1 \leq \boldsymbol{n} \leq 35$.
- The value of willingness $\boldsymbol{a_i}$, $\boldsymbol{b_i}$ will be in the range: $0 \leq \boldsymbol{a_i}, \boldsymbol{b_i} \leq \mathbf{10^8}$.

Input Format

The first line is an integer $\boldsymbol{t}$ which indicates the number of test cases. Then there will be exactly $\boldsymbol{t}$ sets of test cases following

Each test case will begin with one integer $\boldsymbol{n}$, denoting the number of treasures in the bag. The next line will contain n integers, $\boldsymbol{a_i}$, separated by exactly one space, denoting the willingness of Alice to these n treasures. The third line also contains $\boldsymbol{n}$ integers, $\boldsymbol{b_i}$, denoting the willingness of Bob to these $\boldsymbol{n}$ treasures.

Output Format

For each test case, output an integer in one line, which is the minimized difference of total willingness of Alice and Bob, with the requirements mentioned above.

Example

Sample Input:	**Sample Output:**
2 3 3 5 4 2 7 3 2 2 4 3 5	0 1

Problem 4. Coloring Statues

(Time Limit: 1 seconds)

Problem Description

Yagami, the God of night, is an immortal. Since he can live forever, he always feels bored. One day, he finds there are *n* statues in a row. For convenience, we number them from 1 to *n*. These statues are colored either in black or in white initially. As the God of night, Yagami loves darkness. He wants to color all statues in black. But just coloring those white statues immediately is too boring to Yagami. Since he is immortal, Yagami determines to spend many days to color all statues in black by the following recoloring daily procedure every day.

1. If all statues are black, Yagami terminates the procedure.
2. Yagami only colors statues into black or white, and he only colors one statue a day.
3. Yagami does not generate a color pattern of statues which appeared previously. He can only generate a new color pattern by recoloring any one element in a statue set *S*.
4. If there are multiple possible choices satisfying all constraints above, then Yagami will change the color of the statue of smallest index in *S*.

For example, statue 1 is black, and both statue 2 and statue 3 are white initially. In the first day, Yagami will change the color of statue 1 into white, since recoloring any statue will generate a new color pattern. In the second day, Yagami will color statue 2 into black, since coloring statue 1 into black will generate the initial color pattern. In the third day, Yagami will color statue 1 into black again. In the fourth day, Yagami will color statue 3 into black. Since all statues are black, Yagami will not color any of these statues after the fourth day.

	Statue 1	Statue 2	Statue 3
Initially	Black	White	White
After 1 day	White	White	White
After 2 days	White	Black	White
After 3 days	Black	Black	White
After 4 days	Black	Black	Black
After 5 days	Black	Black	Black

Your task is: given the initial color pattern of n statues, compute how many statues are colored in black after t days.

Technical Specification

- There are at most 30 test cases.
- $0 < n \leq 1000$ and $0 < t < 10^{19}$.

Input Format

The first line is an integer which indicates the number of test cases. Each test case consists of two lines. The first line of each test case contains two integers, n and t, separated by a blank. The second line contains n integers $c_1, \ldots, c_n$. For each $1 \leq i \leq n$, $c_i = 0$ if statue i is white, and $c_i = 1$ if statue i is black.

Output Format

For each test case, output the number of the statues colored in black.

Example

Sample Input:	**Sample Output:**
6 3 1 1 0 0 3 2 1 0 0 3 3 1 0 0 3 4 1 0 0 3 5 1 0 0 10 9876543210987654321 0 1 0 1 0 1 0 1 0 1	0 1 2 3 3 10

Problem 5. CPU

(Time Limit: 1 seconds)

Problem Description

Modern computer CPUs can run at different speeds for different purposes. They can run at high speed to get the result fast, or they can run at low speed to conserve energy. We assume that a CPU has k speeds $s_1, s_2, \ldots, s_k$. When the CPU runs at speed s_i then each instruction will take t_i seconds of time, and e_i joule of energy.

We illustrate this trade-off between time and energy with an example. When the CPU runs a program of I instructions at speed s_i we need $I \times t_i$ seconds of time and $I \times e_i$ joule of energy. Now if the CPU runs a program with 4 instructions at speed s_1, then it takes $4 \times 10 = 40$ seconds and $4 \times 1 = 4$ joules. If we run the program at speed s_3, then it takes $4 \times 3 = 12$ seconds and $4 \times 80 = 320$ joules, which is faster but more energy consuming.

Speed	Time	Energy
s_1	$t_1 = 10$	$e_1 = 1$
s_2	$t_2 = 5$	$e_2 = 10$
s_3	$t_3 = 3$	$e_3 = 80$

Now we consider the problem of running a set of programs on a CPU. We assume that we have n programs $p_1, \ldots, p_n$, where p_i has I_i instructions. We can run the programs in any order, but we must run them one after another without interruption. The CPU can run a program at any speed from $s_1, \ldots, s_k$, but a program will run at only one speed. The goal is to schedule an execution order and select execution speeds for all programs so as to minimize the total cost.

The first part of the cost is energy. The energy cost of a program is the joule consumed by a program, times the cost of a joule in dollars. We use R_e to denote this rate of energy. For example, if R_e is 3 and a program has 4 instructions, then it takes $4 \times 1 \times 3 = 12$ dollars of energy cost to run the program at speed s_1. We also define the total energy cost of a set of programs to be the sum of the energy cost of individual programs. If we run three programs p_1, p_2, p_3 with the numbers of instructions and speeds, then the total energy cost will be $12 + 1200 + 60 = 1272$ dollars.

	I_i	Speed	Energy cost	Execution time	Turnaround time	Time cost
p_2	4	s_1	4×1×3=12	4×10=40	40	40×6=240
p_1	5	s_3	5×80×3=1200	5×3=15	40+15=55	55×6=330
p_3	2	s_2	2×10×3=60	2×5=10	40+15+10=65	65×6=390

The second part of the cost is time. The execution time of a program p is the time for p to finish, and the waiting time of p is the sum of the execution time of all programs that are scheduled before it. The turnaround time of a program is the sum of its waiting time and its execution time. The time cost of a program is the product of its turnaround time and a rate R_t, which is the cost of a second in dollars.

We illustrate the concept of time cost with an example. We assume that the execution order is p_2, p_1, then p_3. The execution time of p_2, p_1, and p_3 are 40, 15, and 10 respectively, therefore the turnaround time of p_2 is 40, the turnaround time for p_1 is 40 + 15 = 55, and the turnaround time for p_3 is 40 + 15 + 10 = 65. Now we assume that R_t is 6, which means one second of time is worth 6 dollars. The total time cost is therefore $40 \times 6 + 50 \times 6 + 65 \times 6 = 960$ dollars.

Now we can combine the energy cost and time cost. We have total energy cost 1272 dollars and total time cost 960 dollars, so the final total cost is 1272 + 960 = 2232 dollars. Now we can formally define the task. You are given the numbers of instructions of n programs, k pairs of (t_i, e_i), rate R_e and R_t. The task is to determine the order and speeds to run these n programs, so that the total cost is minimized.

We can also generalize this problem to multiple CPUs. If we have C CPUs, we want to distribute the programs to these CPUs so that the total cost, which is the sum of costs of individual CPU, is minimized.

Write program to compute the minimum cost to run a set of programs in a multiple CPU system.

Input Format

There may be multiple test cases. Please process until your program read an EOF (End of File).

For each test case:

Line 1: C, the number of CPUs.
Line 2: k_1, the number of frequency of the first CPU.

Line 3: $e_{1,1}$, $e_{1,2}$, $e_{1,3}$, …, $e_{1,k1}$, energy per instruction of different speeds of the first CPU.
Line 4: $t_{1,1}$, $t_{1,2}$, $t_{1,3}$, …, $t_{1,k1}$, time per instruction of different speeds of the first CPU.
Line 5: k_2, the number of frequency of the second CPU.
Line 6: $e_{2,1}$, $e_{2,2}$, $e_{2,3}$, …, $e_{2,k2}$, energy per instruction of different speeds of the second CPU.
Line 7: $t_{2,1}$, $t_{2,2}$, $t_{2,3}$, …, $t_{2,k2}$, time per instruction of different speeds of the second CPU.
…
Line 3C+2: n (the number of programs), R_e, R_t.
Line 3C+3 to 3C+n+2: The number of instructions of the n programs.

$1 \leq C \leq 5$
$1 \leq k_i \leq 10$
$e_{i,j}$, $t_{i,j} \leq 100$
$n \leq 100000$
R_e, $R_t \leq 100$
$I_i \leq 10000$
All numbers in input are non-negative integers.

Output Format

For each test case, output the minimum cost in a single line.

Example

Sample Input:	Sample Output:
1 3 1 10 80 10 5 3 3 1 1 5 4 2 2 3 1 10 80 10 5 3	185 139

3 1 10 80 10 5 3 3 1 1 5 4 2 EOF	