COMPUTER SOCIETY OF INDIA (CSI)
COEP TECH STUDENT CHAPTER

PRESENTS

CODE QUEST 5.0

EDITIORIAL

BOAT CLUB THE FEAST

PROBLEM STATEMENT

COEP's iconic Boat Club is nestled by the serene Mula River, a landmark of athletic and recreational brilliance since its establishment in 1928. Boasting a fleet of over 70 boats, an advanced gymnasium, badminton courts, and a canteen known for its vibrant atmosphere, the club has long been the heart of COEP's campus life. Among the canteen's most cherished offerings is the legendary Vadapav, renowned for its tantalizing flavor and unparalleled freshness.

The canteen's secret chutney is the cornerstone of this delicacy, crafted with a recipe passed down through generations. The recipe calls for precise proportions of hand-ground spices and seasonal ingredients, delivered at peak freshness. Each Vadapav requires V grams of this chutney, and the canteen operates under strict guidelines to ensure no compromise in quality.

PROBLEM STATEMENT

Here's where it gets challenging: the chutney is so fresh that it spoils quickly andmust be used within a specific timeframe. Deliveries of the chutney are made periodically, with each batch arriving at a predetermined time and having a specific weight and shelf life. Orders for Vadapav come in sporadically, and if the canteen runs out of unspoiled chutney, it must immediately halt all operations to preserve its reputation.

Today, the Boat Club is hosting its annual alumni meet, a grand feast attended by dignitaries, former students, and the local rowing champions. Everyone is eagerly awaiting the Vadapav. However, due to an unexpected rainstorm, the delivery schedules have been disrupted, causing chaos in the kitchen. The head chef, in her wisdom, devised a new approach to manage the

PROBLEM STATEMENT

chutney stocks efficiently—but she needs your help to optimize this plan.Your mission, should you choose to accept it, is to help her figure out the maximum number of Vadapav orders that can be fulfilled without spoilage or disruption.

Note: If the chutney spoils at the exact moment an order is placed, it cannot be used to fulfill that order. Furthermore, when an order is successfully fulfilled, V grams of chutney are consumed and cannot be reused for any future orders. If the canteen encounters an order that cannot be fulfilled due to insufficient unspoiled chutney, all remaining orders are immediately canceled, as the canteen must close its Vadapav operations and reassess its chutney management strategy.

INPUT FORMAT

1. The first line contains a single integer, T, the number of test cases.

2. For each test case:
   - The first line contains three integers:
     - D: the number of chutney deliveries.
     - N: the number of Vadapav orders.
     - V: the amount of chutney (in grams) required per Vadapav.
   - The next D lines each contain three integers:
     - Ti: the time (in minutes since the canteen opens) when the i-th chutney delivery arrives.
     - Ci: the amount of chutney (in grams) delivered in the i-th delivery.
     - Si: the time (in minutes) for which the chutney from the i-th delivery remains fresh.
   - The last line of the test case contains N integers: P1, P2, ..., PN: the times (in minutes since the canteen opens) when the Vadapav orders are placed.

OUTPUT FORMAT & CONSTRAINTS

For each test case, output a single line in the following format where: y is the maximum number of Vadapav orders that can be fulfilled without spoilage or disruption.

Constraints on variables:

- (1 <=T<=100) —Number of test cases.
- (1 <=D<=100) —Number of chutney deliveries per test case.
- (1 <=N<=100) —Number of Vadapav orders per test case.
- (1 <=V<=100) —Amount of chutney (in grams) required per order.
- (1 <= T_i <= 10^9) — Delivery times are large integers to introduce complexity.
- (T_i < T_{i+1}) — Delivery times are strictly increasing.
- (1 <=C_i <= 100) — Amount of chutney delivered per de livery.
- (1 <=S_i <= 10^9) — Freshness duration is large to mimic real-world constraints.
- (1 <=P_j <= 10^9) — Order times are large integers.
- (P_j < P_{j+1}) — Order times are strictly increasing.

SAMPLE TESTCASES

(1)
SAMPLE INPUT 0 :
1
4 4 2
2
1102
342
514
1063
34610

SAMPLE OUTPUT 0 : 2

(2)
SAMPLE INPUT 1 : 2
2 4 5
20 8 1000000000
60 4 1000000000
10 30 50 70
3 5 5
20 8 1000000000
50 3 1000000000
60 100 1000000000
30 50 59 70 90

SAMPLE OUTPUT 1 :
0
2

EXPLANATION

Explanation 0

1. Deliveries:

- At time 1, 10 grams of chutney is delivered, spoiling after 2 minutes (usable between times 1–3).
- At time 3, 4 grams of chutney is delivered, spoiling after 2 minutes (usable between times 3–5).
- At time 5, 1 gram of chutney is delivered, spoiling after 4 minutes (usable between times 5–9).
- At time 10, 6 grams of chutney is delivered, spoiling after 3 minutes (usable between times 10–13).

2. Orders:

- Order at time 3: Fulfilled using 2 grams from Delivery 2.
- Order at time 4: Fulfilled using 2 grams from Delivery 2.
- Order at time 6: Cannot be fulfilled as only 1 gram remains (Delivery 3).
- Orders beyond this are ignored as the canteen closes.

SOLUTION

```cpp
#include <bits/stdc++.h>
#define ll long long


using namespace std;

int T, d, n, u;
int m[101], l[101], e[101], t[101];

int solve() {
    cin>>d>>n>>u;
    for(int i=0;i<d;i++) cin>>m[i]>>l[i]>>e[i];
    for(int i=0;i<n;i++) cin>>t[i];

    int id = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    for(int i=0;i<n;i++) {
        while(id < d and m[id] <= t[i]) {
            pq.push({m[id] + e[id], l[id]});
            id++;
        }
        while(!pq.empty() && pq.top().first <= t[i]) pq.pop();
        int rem = u;
        while(rem) {
            if(pq.empty()) return i;
            if(pq.top().second <= rem) {
                rem -= pq.top().second;
                pq.pop();
            } else {
                pair<int, int> p = pq.top(); pq.pop();
                pq.push({p.first, p.second - rem});
                rem = 0;
            }
        }
    }
    return n;
}

int main() {
    cin>>T;
    for(int i=1;i<=T;i++) {
        cout<<""<<solve()<<"\n";
    }
    return 0;
}
```