#include <iostream>

#include <vector>

using namespace std;

// funzione che restituisce la frequenza della key nel vector

int algoritmo(vector<int> v, int low, int high, int key)

{

//Se abbiamo controllato tutto il vector e non abbiamo trovato l'elemento

if ((low > high) || (low == high && v[low] != key))

return 0;

//se abbiamo trovato l elemento

if (low == high && v[low] == key)

return 1;

//seguendo l'approccio dividi et impera dividiamo il vettore in due e richiamiamo l'algoritmo

return algoritmo(v, low, (low + high) / 2, key) + algoritmo(v, 1 + (low + high) / 2, high, key);

}

int main()

{

int n;

cin>>n;

vector<int> v;

while(n){

int key;

int num\_elementi;

cin>>key;

cin>>num\_elementi;

while(num\_elementi){

int temp;

cin>>temp;

v.push\_back(temp);

num\_elementi--;

}

cout << algoritmo(v, 0, v.size()-1, key)<<endl;;

n--;

}

return 0;

}

//La complessità degli inserimenti maggiore è quella del while per il riempimento del vettore che sarà pari ad O(n)

//La chiamata alla funzione algoritmo è invece pari a O(logn)

//La Complessità totale è dunque O(Nlogn)

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

// funzione di stampa

void print(vector<int> v){

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

}

//funzione che valuta tutti i possibili numeri primi la cui somma è somma

void backtracking(int total, int num, int somma, int index, vector<int> &sol, vector<int> &primi)

{

//se il totale è uguale a somma e il totale è raggiunto usando num primi

if (total == somma && sol.size() == num)

{

print(sol);

return;

}

//se il totale è maggiore di somma o se l'indice ha raggiunto l'ultimo elemento

if (total > somma || index == primi.size())

return;

// aggiungo l'iesimo elemento di primi a sol

sol.push\_back(primi[index]);

// aggiungo l'iesimo elemento a total

backtracking(total+primi[index], num, somma, index+1, sol, primi);

// elimino la mossa precedente

sol.pop\_back();

// escludo l'elemento iesimo dallo studio

backtracking(total, num, somma, index+1, sol, primi);

}

//funzione per controllare se un numero è primo

bool controlloseprimo(int n)

{

//escludo i numeri minori e uguali a 1

if (n <= 1)

return false;

// controllo fino a n

for (int i = 2; i < n; i++)

if (n % i == 0)

return false;

return true;

}

void sommeconprimi(int num\_elementi, int somma, int primo, vector<int> &sol , vector<int> &primi)

{

int numerosucc = primo + 1;

for (int i = numerosucc; i <= somma ; i++){

// se è primo lo aggiungo al vettore

if (controlloseprimo(i)){

primi.push\_back(i);

}

}

//controllo che i numeri primi siano più di num\_elementi altrimenti non posso fare la somma

if (primi.size() < num\_elementi)

return;

backtracking(0, num\_elementi, somma, 0, sol, primi);

}

int main()

{

int n;

cin>>n;

int dim = n;

while(n){

vector<int> sol;

vector<int> primi;

int somma;

int primo;

int num\_elementi;

cin>>somma;

cin>>num\_elementi;

cin>>primo;

cout<<endl;

cout<<"CASO DI TEST "<<dim - n +1 <<endl;

sommeconprimi(num\_elementi, somma, primo, sol, primi);

n--;

sol.clear();

primi.clear();

}

return 0;

}

//La complessità maggiore è quella della chiamata ricorsiva della funzione di backtracking che è esponenziale