```
#include<iostream>
#include<queue>
using namespace std;
struct data
{ string s; double b; int r; };
struct cmp:
public binary_function<data, data, bool>
{ bool operator()(data x, data y)
   {if(x.b < y.b) return true;
    if(x.b==y.b) if(x.r > y.r) return true;
    return false;
   }
};
priority_queue<data, vector<data>, cmp> Q;
int main()
{int c=0;
 int n; cin >> n;
 for(int i=1;i<=n;i++)
 { int m; cin >> m;
   for(int j=1;j<=m;j++)
    { data d;
      cin >> d.s >> d.b;
      d.r=c; c++;
      Q.push(d);
    }
   int k; cin >> k;
   for(int j=1; j<=k; j++) Q.pop();
 }
 cout << (Q.top()).s << endl;
}
```


Група С

Задача С1. РЕДИЦИ, автор Емил Келеведжиев

Да разгледаме редицата (3,1,4,2,4). Тя може да бъде разбита на две подредици – (3,4) и (1,2,4), които са строго растящи, т.е. елементите им са сортирани в строго нарастващ ред. *Разбиване* означава, че всяка от двете подредици трябва има поне по един елемент, обединението на елементите на двете подредици трябва да се състои от елементите на дадената редица и никой елемент от дадената редица не трябва едновременно да е елемент и на двете подредици. Забележете, че за редицата (4,8,1,5,3) такова разбиване не може да се направи. Напишете програма **SEQ**, която "познава" дали дадена редица от цели положителни числа може да бъде разбита на две строго растящи подредици.

Вход

На първия ред на стандартния вход е зададен броят N на редиците, които програмата трябва да провери, $N \leq 10$. Всеки от следващите N реда дефинира по една редица. Редът започва с броя M на елементите в съответната редица, $2 \leq M \leq 100\,000$. В същия ред, след един интервал, са зададени елементите на редицата, разделени един от друг с интервали. Всички елементи са цели положителни числа, не надминаващи 1 милиард.

Изход

На стандартния изход програмата трябва да изведе низ с дължина N, съставен от нули и единици (без разделящи интервали), с резултатите от направените проверки. За поредната редица низът трябва да съдържа цифрата 1, ако е възможно тази редица да се разбие на две строго растящи подредици, или цифрата 0, ако такова разбиване не е възможно.

ПРИМЕР

Вход	**Изход**
2 5 3 1 4 2 4 5 4 8 1 5 3	10

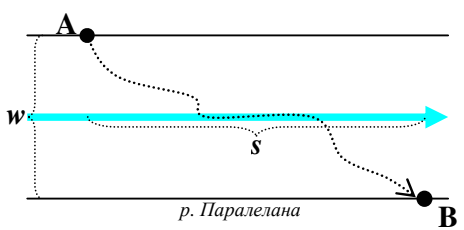
Решение

Наивната идея да образуваме всички възможни разбивания на редицата в две подредици и да проверяваме, дали получените подредици са строго растящи, ще доведе до алгоритъм с експоненциално нарастване на броя на стъпките, при увеличаване размерите на задачата. Алгоритъм с линейна сложност се получава по следния начин. Да допуснем, че успешно сме разбили началото (съставено от първите I елемента) на редицата и a и b са последните (максималните) елементи на двете подредици. За началото, съставено от първия елемент, такова разбиване е възможно и единствено – в едната подредица е първият елемент, а втората е празна. Максималният измежду a и b да означим с max. Да разгледаме (I+1)-вия елемент c. Ако $c > max$, трябва да го причислим към подредицата на max. Ако $c < max$ – опитваме да го причислим към другата подредица. Ако това е възможно – получили сме разбиване на първите I+1 елемента. Ако с добавянето му там строгата монотоннос на подредицата се нарушава, ясно е, че задачата няма решение.

```
#include <iostream>
using namespace std;
void check(int k)
{
   int a; cin >> a; int b=0;
   int i=2; int c;
   while(i<=k)
   {
      cin >> c;
      if(c>a) a=c; else if(c>b) b=c;
      else
      {  i++; while(i<=k){cin >> c; i++;}
         cout << 0; return;
      }
      i++;
   }
   cout << 1;
}
int main()
{
   int n; cin >> n;
   for(int i=1;i<=n;i++)
   { int k; cin >> k; check(k); }
   cout << endl;
}
```

Задача С2. ПРЕСИЧАНЕ, автор Павлин Пеев

Река Паралелана почти замръзна! Преди беше лесно да стигнете с кануто си от коя да е точка A на единия бряг до коя да е точка B, разположена на другия бряг и надолу по течението. Сега по средата е останало водно пространство точно колкото за едно кану. Това все пак е нещо, но нали, за да стигнете до водата, се налага да влачите кануто по леда, а като излезете от водата – да го влачите отново до крайната точка? А в този студ ви се иска да стигнете колкото може по-бързо! Както се вижда от картата на местността, реката в този участък е с прави, успоредни брегове. Широчината ѝ е w км, а хоризонталното преместване от A до B е s км. Вашата скорост по леда е a км/ч, а във водата – b км/ч, като, естествено, $a < b$. Напишете програма **CROSSING**, която да намира минималното необходимо време за придвижване от точка A до точка B.



Вход

На единствения ред на стандартния вход са зададени, разделени с интервали, числата w, s, a и b, в тази последователност. Ако някое от числата има дробна част, тя е разделена от цялата част с точка и е с не повече от два десетични знака. Числата са положителни, като $w < s \le 40$, а $a < b \le 20$.

Изход

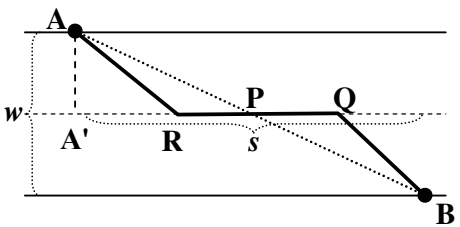
Програмата трябва да изведе на стандартния изход един ред с минималното време, за което можете да стигнете от A до B, закръглено с точност до секунда, в стандартен времеви формат – часове, двоеточие, минути, двоеточие, секунди – и без водещи нули.

ПРИМЕР

Вход	**Изход**
1.5 6 3.12 9.6	1:4:47

Решение

От съображения за симетрия и равнопоставеност, маршрутът трябва да има вид, подобен на начупената линия ARQB на чертежа – симетричен относно P. Тогава е достатъчно да пресметнем времето за достигане до точка P и да го удвоим.



Остава да определим положението на точка R върху средната права. Ясно е, че тя не може да е по-наляво от A', мястото ѝ е в затворената отсечка A'P. Също така е ясно, че с преместването на R по A'P от ляво на дясно, търсеното време или само ще нараства, или само ще намалява, или ще намалява донякъде и после ще започне да расте, т. е., ще има една най-малка стойност – или в A', или в P (когато отсечката RQ ще се изроди в точка), или някъде между тях. На тези разсъждения се основава конкретната реализация на алгоритъма „клатене" – търсене на интервал с ширина 0.1, съдържащ точката P, търсене в него на интервал с ширина 0.01, съдържащ P и т.н., до намиране на интервал с ширина 0.00001, съдържащ P, което е напълно достатъчно за точност до секундата в исканията формат.

```
#include <stdio.h>
#include <math.h>
typedef struct {double x,y;} Point;
double w,s,a,b;
Point A,P;

//Разстояние между точки
double dist(Point p1, Point p2)
{  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+
               (p1.y-p2.y)*(p1.y-p2.y));
}
//Време за изминаване на ARP
double getTime(Point R)
{  return dist(A,R)/a+dist(R,P)/b;  }
```