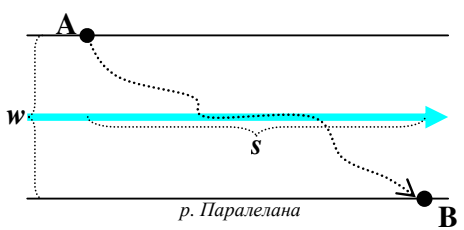
```
#include <iostream>
using namespace std;
void check(int k)
{
   int a; cin >> a; int b=0;
   int i=2; int c;
   while(i<=k)
   {
      cin >> c;
      if(c>a) a=c; else if(c>b) b=c;
      else
      {  i++; while(i<=k){cin >> c; i++;}
         cout << 0; return;
      }
      i++;
   }
   cout <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

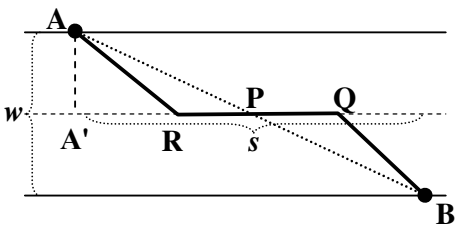
Програмата трябва да изведе на стандартния изход един ред с минималното време, за което можете да стигнете от A до B, закръглено с точност до секунда, в стандартен времеви формат – часове, двоеточие, минути, двоеточие, секунди – и без водещи нули.

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;  }
```