Algoritmer (datastrukturer) och komplexitet
våren 2009

Ommästarprov 1: Algoritmer

Detta mästarprov är avsett för den som ännu inte är godkänd på mästarprov 1. Det är bara en uppgift och den kan bara ge betyg E. Mästarprovet ska lösas **individuellt** och redovisas både skriftligt och muntligt. Inget samarbete är tillåtet, se vidare hederskodexen.

Du ska lämna in den **skriftliga lösningen** på studerandeexpeditionen (Osquars backe 2, plan 2) senast klockan 12.00 den 13 maj 2009. Den **muntliga redovisningen** är 14 maj 2009. Boka tidigast 11 maj och senast 13 maj en tid för muntlig redovisning med kommandot `bok new adk09`, föregånget av `telnet hippograff` och vid behov `module add resultat`

Flygplansinflygning

Konstruera en algoritm som löser första uppgiften på programmerings-VM som var 21 april. Uppgiftlydelsen från tävlingen finns på baksidan.

För att det inte ska bli för svårt så ska du också få ett tips från CSC-programmeringstävlingsuppgiftsexperten Per Austrin:

> Since the number of planes is at most 8, an optimal solution can be found by simply trying all $8! = 40320$ possible orders for the planes to land. When trying a specific ordering, the largest possible landing window can be computed by binary searching over the maximum possible window and then greedily checking whether a certain window length can be achieved.

Din algoritm ska alltså använda såväl totalsökning som binärsökning och en girig algoritm. Du behöver inte vara så noga med formatet för in- och utdata som man måste vara i riktiga programmeringstävlingar. Algoritmen ska klara också $n > 8$, även om det tar exponentiell tid.

Analysera tidskomplexiteten för algoritmen uttryckt i antalet plan n.

Problem A
A Careful Approach
Input: approach.in

If you think participating in a programming contest is stressful, imagine being an air traffic controller. With human lives at stake, an air traffic controller has to focus on tasks while working under constantly changing conditions as well as dealing with unforeseen events.

Consider the task of scheduling the airplanes that are landing at an airport. Incoming airplanes report their positions, directions, and speeds, and then the controller has to devise a landing schedule that brings all airplanes safely to the ground. Generally, the more time there is between successive landings, the "safer" a landing schedule is. This extra time gives pilots the opportunity to react to changing weather and other surprises.

Luckily, part of this scheduling task can be automated – this is where you come in. You will be given scenarios of airplane landings. Each airplane has a time window during which it can safely land. You must compute an order for landing all airplanes that respects these time windows. Furthermore, the airplane landings should be stretched out as much as possible so that the minimum time gap between successive landings is as large as possible. For example, if three airplanes land at 10:00am, 10:05am, and 10:15am, then the smallest gap is five minutes, which occurs between the first two airplanes. Not all gaps have to be the same, but the smallest gap should be as large as possible.

Input

The input file contains several test cases consisting of descriptions of landing scenarios. Each test case starts with a line containing a single integer n ($2 \le n \le 8$), which is the number of airplanes in the scenario. This is followed by n lines, each containing two integers a_i, b_i, which give the beginning and end of the closed interval $[a_i, b_i]$ during which the i^{th} plane can land safely. The numbers a_i and b_i are specified in minutes and satisfy $0 \le a_i \le b_i \le 1440$.

The input is terminated with a line containing the single integer zero.

Output

For each test case in the input, print its case number (starting with 1) followed by the minimum achievable time gap between successive landings. Print the time split into minutes and seconds, rounded to the closest second. Follow the format of the sample output.

Sample Input

```
3
0 10
5 15
10 15
2
0 10
10 20
0
```

Output for the Sample Input

```
Case 1: 7:30
Case 2: 20:00
```