Happy Landings

Purpose: vectors, next permutation, testing, brute force

Due: May 29

Ultra-Modern software Development (UMD) has hired you to write a program to help air traffic controllers. Air traffic controllers have to help land planes. As part of their job they schedule the landings of the planes. For safety reasons they want as much time as possible between landings. However since the planes have a finite amount of fuel, they each radio ahead with a start and end time in which they must land or run out of fuel and crash. Your job is to write a program whose input is a list of start and end times and which outputs the schedule of landings where the closest two landings are as far apart as possible.

For example if plane i can land in the interval $I_i = [s_i, t_i]$ and you are given the data on three planes, $I_1 = [0, 10]$, $I_2 = [5, 12]$, $I_3 = [10, 16]$, one should schedule plane 1 at 0, plane 2 at 8 and plane 3 at 16, so the minimum distance between landings is 8.

Given $I_1 = [0, 10]$ and $I_2 = [10, 20]$, one should schedule plane 1 at 0, plane 2 at 20, so the minimum distance between landings is 20.

Given $I_1 = [0, 10]$, $I_2 = [5, 10]$, $I_3 = [0, 15]$, one should schedule plane 1 at 0, plane 2 at 7.5 and plane 3 at 15, so the minimum distance between landings is 7.5.

Input: The input will be from the keyboard. The input will consist of multiple test cases. The first line of the test case will contain an integer t, the number of test cases. Each test case will be on 2 lines. The first line of the test case will contain the integer $n > 1$ indicating the number of planes to schedule. The second line of the test case will contain the n pairs of positive integers. The i^{th} pair of times represent the interval for plane i, $i = 1, 2, \ldots$. The two integers represent the start and end times (respectively) in minutes since the start of the controllers shift. Each time is separated by a single space.

Output: : The output will be the screen. For each test case there will be 1 line of output containing a sequence of planes numbers (each separated by a single space) followed by the minimum time between planes to the nearest hundredth of a minute.

Sample Input

```
2
2
0 20 0 10
3
5 12 0 10 10 16
```

Sample Output

```
2 1 20.00
2 1 3 8.00
```

Hints:

1. Create a class or struct to hold each interval. It can hold more information if you desire.
2. Your program can process 1 test case and output it immediately, before reading the next test case

Required Algorithm

For each permutation of n planes numbers $(p_1, p_2, \ldots, p_n)$

1. Land the plane p_i, at its start time s_{p_i} and call this L_0
2. Search for an optimal Δt (the distance between landings) so that if we
   (a) Land each plane at L_i, $i > 1$ where $L_i = \max(s_{p_i}, \Delta t + L_i\text{-}1)$ is feasible
   (b) 2. and Δt is as large as possible (to within .01)
3. If the schedule is feasible and has a larger Δt then update best so far, the save best schedule and Δt.

How the program will be graded

You will turn in 2 documents. A memo (table 1) and source code (table 2) . Putting items in unexpected places will result in earning a 0 for that item.

Memo

What	pts
Name	1
Time Analysis O() of every function[1,2] (in terms of the number of planes and the total time	8
Space Analysis O() of every function[1,2]	8
Test Plan[3] with at least 4 original nontrivial BLACK BOX tests [4]	12

[1]The main() is a function.

[2]All analysis should be worst case based on the number of planes as well as the earrlies start and the lastest final time.

[3]A test plan is a table with 4 columns and 1 row per test. The columns are named Reason for the test, actual input data, expected output data, and actual output. You do NOT have to have a working program to write a test plan. Each reason should be **unique.**

[4]A non trivial test contains only legal data (data the conforms to the input specification)

Source Code Document

What	pts
Name	1
Description[5]	4
Style	8
pre/post conditions	7
Functionality using the STL	50

[5]The description should be written to some one who knows NOTHING about the program. It should discuss what the program does (in your own words). After reading the description the user should be able to create legal input and predict the output.