Animal Feeding

Objective
Give practice with Sorting in C.
Give practice with Binary Search in C.
Give practice with Heaps in C.

Story
Your employees have too many mangos. The mangos go bad after some time and leave a sticky, smelly residue. The park is beginning to look and smell like a pig sty every day. Luckily, you have intercepted the communications with the local mango providers and your employees. Now you know in advance when the mangos arrive for the next year. Additionally, you can estimate when each shipment spoils based on the provider.

Your animals can consume food at a constant, non-negative, real rate (in pounds per minute). The more animals you have the higher this rate, but the more it will cost to maintain the park. Your objective is to find out the smallest rate at which the animals need to be able to consume food such that no amount of mango spoils in the upcoming year. You can assume that if there are no mangos in your park, then the animals will instead eat some of the other non-perishable food your park has.

Problem
Given the shipment description of all the mangos (arrival time, spoil time, and size in pounds). Determine the smallest rate the animals in your park can consume food such that none of the mangos spoil.

Input
Input will begin with a line containing 1 integer, $\boldsymbol{n}$ ($1 \leq \boldsymbol{n} \leq 200{,}000$), representing the number of shipments of mangos. The following n lines will contain the description of a single mango shipment. The mango shipment description will consist of a line with 3 integers, $\boldsymbol{T_a}$, $\boldsymbol{T_s}$, and $\boldsymbol{S}$ ($1 \leq \boldsymbol{T_a} < \boldsymbol{T_s} \leq 525{,}600$; $1 \leq \boldsymbol{S} \leq 2{,}000$), representing the arrival time, the spoil time, and the size in pounds of the shipment respectively. Note the shipments are not guaranteed to be in their arrival order.

Output
Output a single line that contains a single non-negative real number representing the rate (in pounds per minute) that your animals need to consume food.

Your answer will be accepted if it is within 10^{-5} absolute or relative error. For example if the answer is 2,000,000, then an acceptable answer would be in the range of 1,999,980 to 2,000,020. Additionally, if the answer is 0.25, then an acceptable answer would be in the range, 0.24999 to 0.25001

Sample Input	**Sample Output**

`5`
`15 20 5`
`5 10 10`
`13 20 4`
`27 48 2`
`21 100 6`
	`2.000000`

`3`
`1 4 3`
`3 6 3`
`5 8 3`
	`1.285714`

Explanation

Case 1

We have 5 shipments. In terms of arrival we have the following situation.

- The first shipment arrives at time 5 and spoils 5 times units later (time 10); it has 10 pounds of mangos.
- The second shipment arrives at time 13 and spoils 7 time units later (time 20); it has 4 pounds of mangos.
- The third shipment arrives at time 15 and spoils 5 time units later (time 20); it has 5 pounds of mangos.
- The fourth shipment arrives at time 21 and spoils 79 time units later (time 100); it has 6 pounds of mangos.
- The fifth shipment arrives at time 27 and spoils 21 time units later (time 48); it has 2 pounds of mangos.

A rate of 2 pounds per time unit can allow for us to

- Finish the first shipment just as it spoils.
- Finish the second shipment at time 15
- Finish the third shipment at time 17.5
- Finish the fourth shipment at time 24
- Finish the fifth shipment at time 28

Any rate less than 2 would not work since the first shipment would expire

Case 2
There are 3 shipments.

- The first shipment arrives at times 1 and spoils 3 time units later; it has 3 pounds of mangos
- The second shipment arrives at time 3 and spoils 3 time units later; it has 3 pounds of mangos
- The third shipment arrives at time 5 and spoils 3 time units later; it also has 3 pounds of mangos

At a rate of 9/7 we can

- Finish the first shipment at time 3⅓
- Finish the second shipment at time 5⅔
- Finish the third shipment at time 8 (when it spoils).

Hints

Binary Search: You should search for the smallest rate the works. The search can be done using a binary search. The lowest rate would be 0. There is always a rate that can work, since we could get all the animals to eat all the mangos in a single minute. This rate works because each shipment will last for at least one minute.

Simulation: For the guessed rate you should simulate time passing and animals eating the mangos. Keeping track of time is incredibly useful for determining which shipments of mangos can be consumed.

I recommend skipping time, and not simulating minute by minute.

Sorting: Shipments should be added to the list of usable shipments by their arrival time. For this reason you should **sort** the array shipments by **arrival** time prior to simulating.

Shipment Consumption Decision: The animals can eat any shipment of mangos that have arrived. The shipment that the animals should focus on is always the ones that expire first. To determine which shipment should be consumed you can store the available shipments in a **heap** sorted by **spoil** time.

Functions: Functions were very useful in making the code easier to understand IMO.

(More Hints on Following Page).

Suggested Solution Header:

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define PARENT(x) (((x)-1)/2)
#define RIGHT(x)  ((x)*2+2)
#define LEFT(x)   ((x)*2+1)

typedef struct ArrayList ArrayList;
typedef struct Shipment Shipment;

// A Shipment struct
struct Shipment {
    int arrive, depart; // The arrival and spoil time of the shipment
    int originalSize;   // The size of the original shipment
    double curSize;     // The size of the shipment while in the heap
};

// The heap/array list struct
struct ArrayList {
    Shipment * array;
    int size, cap;
};

// Array List Prototypes
ArrayList * createList();
void cleanList(ArrayList * list);
void append(ArrayList * list, Shipment shipment);

// Heap Prototypes
void swap(ArrayList * heap, int index1, int index2);
int compare(ArrayList * heap, int index1, int index2);
void percolateUp(ArrayList * heap, int index);
void percolateDown(ArrayList * heap, int index);
void heapAdd(ArrayList * heap, Shipment shipment);
void heapRemove(ArrayList * heap);
Shipment front(ArrayList * heap);

// General Functions
// Sort an array of shipments
void sort(Shipment * array, int numShipments);

// Remove the shipments that can be consumed between the given old and new times
// If any shipment expires before finishing consumption, then return 0
// If no shipment is found to expire return 1
int update(ArrayList * heap, int oldTime, int newTime, double consumptionRate);

// Return 1 if the guessed consumption rate works for the given array of shipments
// Return 0 if the guessed consumption rate does not work
int canDo(double consumptionRate, Shipment * array, int numShipments);
```

(Hints on update and canDo functions on following page)

Update Function: The following is the comments used in my update function

```
// Remove the shipments that can be consumed between the given old and new times
// If any shipment expires before finishing consumption, then return 0
// If no shipment is found to expire return 1
   // Keep track of the current time
   // Loop while there is some value in the heap
      // Determine the time required to finish consuming the current shipment
      // Determine the time when the shipment would finish consumption
      // Check if we cannot finish the shipment before spoiling
      // Check if we can finish the shipment before the end of the update
         // Update time and remove from heap
      // Otherwise
         // Update the remaining size of the shipment and stop the simulation
```

Can Do Function: The following is the comments used in my canDo function

```
// Return 1 if the guessed consumption rate works for the given array of shipments
// Return 0 if the guessed consumption rade does not work
   // Create a heap
   // Special case the first shipment
   // Loop through the remaining shipments
      // Update the heap based on the time of the current shipment
      // Handle if the update was invalid :(
      // Initialize the current size of the current shipment and add it to the heap
   // Update the time to the last possible time
   // Clean up any memory
   // Return based on the validity of the last update
```

Grading Criteria

- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, "Please enter the number of friends:")
    - 10 points
- Sort the shipments by the arrive time
    - 5 points
- Use reasonable low rate and high rate for a binary search
    - 5 points
- Binary search for a reasonable number of times.
    - 5 points
- Use some code to check that a midpoint rate works for the list of shipments
    - 10 points
- Programs will be tested on 10 cases
    - 5 points each

No points will be awarded to programs that do not compile using "gcc -std=gnu11 -lm".

Sometimes a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use a heap. ***Without this programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.