Description

The teachers from Simon's exchange argument comic [1] have realised that getting all the students to stand in a line leading directly away from the camera won't work for one of their classes, because some of the students have identical heights. Instead, they have returned to the chaotic approach of letting the students stand in a crowd. So far, all the students are visible. They have some more students that wish to stand in the back, but they are finding it quite difficult to figure out where to stand, such that they will still be visible.

They decide to draw a diagram of the photo so that they can more easily determine which parts of the background are guaranteed to be visible to the camera. As there are a lot of students, they want you to come up with an algorithm to solve the problem for them. They define the problem as follows:

Input

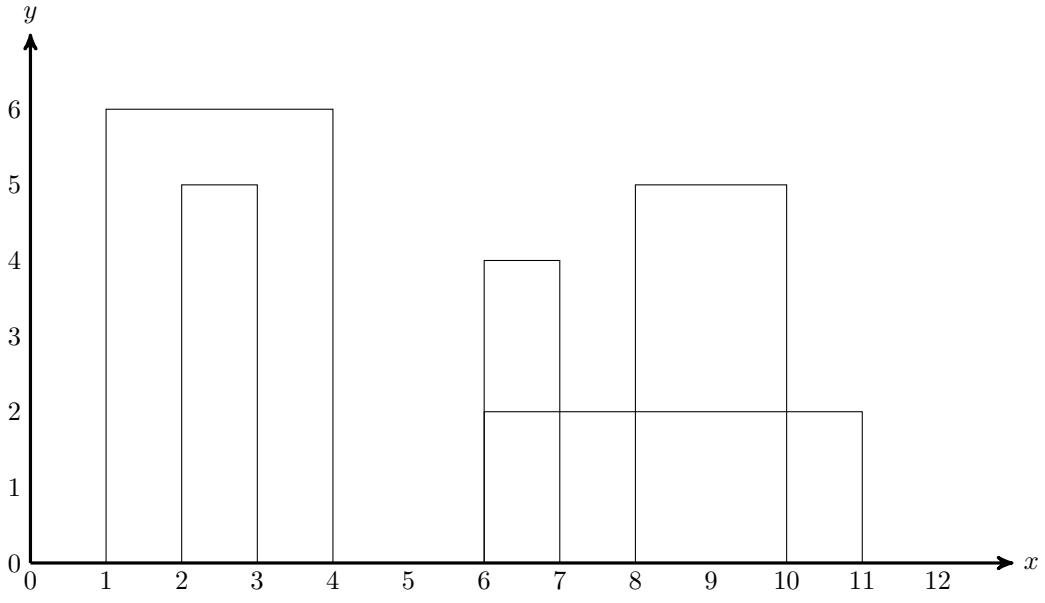
A set of n 3-tuples $(left, right, height)$ which define rectangular areas obscured students

- $left$ is where the left edge of the rectangular student sits on the x-axis. It is a positive integer.
- $right$ is where the right edge of the rectangular student sits on the x-axis. It is an integer and is strictly greater than $left$.
- $height$ is how tall the rectangular student is. It is a positive integer.
- The bottom of each student's rectangle is always at 0 on the y-axis

The students are not given in any particular order, they can overlap, and the same value can appear many times. For example, consider the following input:

```
5
2 3 5
8 10 5
6 7 4
1 4 6
6 11 2
```

It represents $n = 5$ students, the first student is $(2, 3, 5)$, the second is $(8, 10, 5)$ etc. The resulting photo would look something like this:



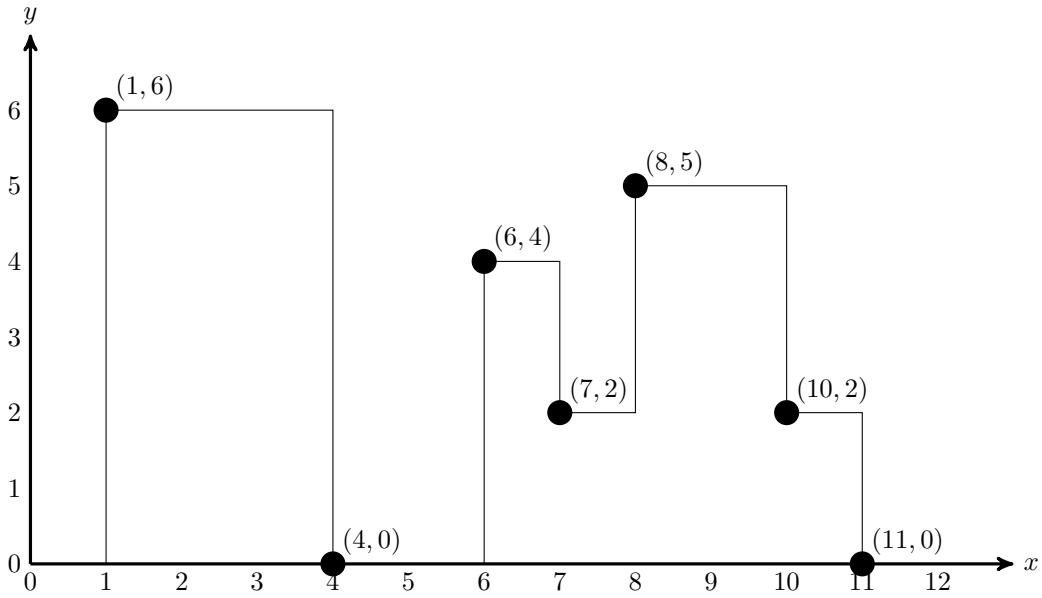
[1] https://edstem.org/courses/1951/discussion/78095

Output

A sequence of points (x, y) which define the obscured area. Each point represents a new height, and must be ordered in strictly increasing x order. The height is initially considered to be at 0. For example, given the input above, the output would be exactly this:

```
1 6
4 0
6 4
7 2
8 5
10 2
11 0
```

Which would look like this, if you drew it on a graph.



(Note: your algorithm doesn't need to output the graph itself – it's just an illustration to help you understand the problem.)

Task 1: [20 marks]

Define a *divide and conquer* algorithm which solves this problem. Describe all parts of your algorithm in words (i.e. not in pseudocode). Take particular care to clearly explain the merge step of your algorithm.

Task 2: [40 marks]

Formally prove that your algorithm gives a correct solution. In particular, take care to prove that the base case(s) of the recursion are valid, and that your merge step always works.

Task 3: [20 marks]

State and explain the complexity of your algorithm (in big-Oh notation). Don't forget to take into account the data structures you use. Consider both:

1. Time complexity
2. Space complexity

Make sure you justify the answers you give, don't merely state the values. As part of solving the time complexity, you will need to define and solve a recurrence (but don't forget to also consider any work done before or after the divide and conquer part of your algorithm.)

Task 4: [20 marks]

Implement your algorithm (on Ed), in the programming language of your choice.

Your program will read the input as a series of lines sent as standard input (i.e. as if the user had typed them at the console)

- The first line is n, the number of students
- The remaining n lines each have three integers representing $left$, $right$, and $height$, separated by a single space

Your program should output one line for each point in the output. Each line should have two integers x and y, separated by a space. The points should be ordered in strictly increasing x coordinate. See the problem description for an example of both input and output.

Some skeleton code is provided in Python which reads the input for you. You can use this code, or start from scratch in this or another language if you prefer.

Please note that the test cases will not cover any weird edge cases around the input format itself (i.e. the input will always be valid, have the expected number of lines, all the numbers will be positive integers, with the specified whitespace etc.)

There will be limits on both *time* and *space* used by your algorithm. So try to keep your implementation reasonably efficient in both. If you get the "killed" message in Ed, it probably means your submission used too much memory – check with a tutor if you're in doubt of what caused this particular error message. If your submission used too much time, or gave the wrong answer, you should get a clearer feedback message from Ed.

Submission details

- Submission deadline is Friday 27th April, at 23:59pm. Late submissions without special consideration will be subject to the penalties specified in the first lecture (25% per day or part thereof.)
- Submit your answers to Tasks 1, 2, 3 as a single document (preferably a pdf or doc file) to Canvas. Your work must be typed text (no images of text, although you can use diagrams if you think it helps.) Please try to be reasonably concise.
- Submit your code for Task 4 to `edstem.org`
- Both your code and report will be subject to automatic and manual plagiarism detection systems. Remember, it's acceptable to discuss high level ideas with your peers, but you should not share the detail of your work, such as (but not limited to) parts of the actual algorithms, (counter)examples, proofs, writing, or code.
- COMP3927: You have an additional question (see next page.) The answer to this question should be submitted separately to the rest of your report, also on canvas (to make it easier for us to coordinate the marking of that question separately.)

Extra question for COMP3927 students only [15 marks]

Given n jobs of duration $t_1, \ldots, t_n$ and m machines $M_1, \ldots, M_m$ schedule the jobs on the machines to minimize the completion time. The completion time is the time it takes to complete all the jobs running the machines in parallel. An example is given in Fig. 1.

i	t_i
1	4
2	2
3	5
4	1
5	3

$m = 3$

M_1 : Jobs 1, 2 time=6

M_2 : Jobs 3,4 time=6

M_3 : Jobs 5 time=3

$\rightarrow$ Completion time = 6

Figure 1: Illustrating an instance for Question 1 with $n = 5$ jobs and $m = 3$ computers. Jobs 1 and 2 are scheduled on machine M_1, jobs 3 and 4 on machine M_2 and job 5 on machine M_3. The completion time of this schedule is 6.

Below we will state two greedy type algorithms, your task is to prove an approximation bound for both of them. To your help you can use the following two simple observations, where C^* denotes the completion time of an optimal solution.

Observation 1.

$$C^* \geq \max_{1 \leq i \leq n} t_i$$

Observation 2.

$$C^* \geq \frac{1}{m} \sum_{1 \leq i \leq n} t_i$$

1. Consider the following naïve algorithm A. Go through the list of jobs (in some arbitrary order), scheduling job i on the machine that currently has the smallest load (finishes first). Prove that algorithm A is a 2-approximation algorithm. [5 points]

2. Consider the following algorithm B. Go through the list of jobs in decreasing order of duration, scheduling job i on the machine that currently has the smallest load (finishes first). Prove that algorithm B is a 4/3-approximation algorithm. *Hint:* One way to prove this bound is by induction. [10 points]