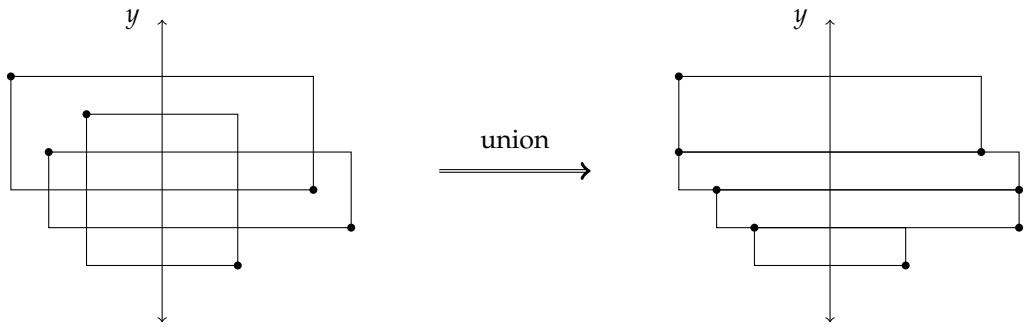
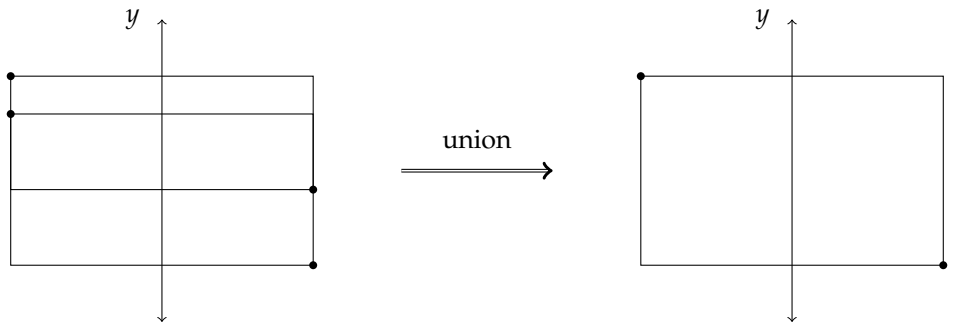
Problem 1. We want to design a divide and conquer algorithm for computing the union of a collection of rectangles. The input rectangles are aligned with the axes and they are all stabbed by the y-axis. Each rectangle is represented by the coordinates of its top-left and bottom-right corners, and the union is representation by a sequence of interior-disjoint rectangles listed from top to bottom. We require that no two consecutive rectangles in the representation can be merged into a single rectangle.

For example, the union of the three rectangles on the left leads to the union represented by the four rectangles on the right:



In each case, the top-left and bottom-right corners of the rectangles in question have been highlighted for reference.

Here is another example where the union of two rectangles on the left leads to the union represented by a single rectangle on the right[1]:



The scaffold provided in Ed uses the following classes:

- Point: Represent a point on the plane with integer coordinates.
- Rectangle: Represents a single rectangle R. Supports membership queries of the form does $x \in R$?
- Union: Represents the union U of a set of rectangles using a sequence of disjoint rectangles in top-to-bottom order. Supports membership queries of the form $x \in U$.

[1] Please note that in this second example, representing the union with two ore more rectangles would be incorrect.

The scaffold provides the top level code for divided and conquer algorithms to computing the union of a set of rectangles, but the merge step of each of these algorithms is missing.

Using the scaffold provided in Ed, **your task** is to implement the missing functions marked with a TODO note. Here is a non-exhaustive list of the main functions:

- merge_unions(union_left, union_right): Compute the union of two Union objects. You can assume the function will only be called by the divide and conquer algorithm, so you only need to handle inputs that can arise from the execution of the union divide and conquer algorithm. Your implementation should run in $O(|union\_left| + |union\_right|)$ time.
- Rectangle.__init__(topleft_point, bottomright_point): Validate input and throw ValueError if it doesn't. Rectangle must intersect the y-axis and the bottom-right corner must lie strictly to the right and strictly below the top-left corner.
- Rectangle.contains(point): Check if point lies on the rectangle.
- Union.contains(point): You can assume that the object is the result of some union divide and conquer execution. Your implementation should run in $O(|union|)$ time.