The University of Texas
Computer Science

313E Programming Assignment 06
Collapsing Intervals

Setup

This assignment may be completed individually or with a pair programming partner. Be sure to include your and your partner's name and EID in the Python file header. If you are working alone, you may delete one of them.

Complete these steps to prepare for the Assignment.

Check	Description
▢	Download intervals.py, intervals.in, and intervals.out. You will be working on the intervals.py file.
▢	Place all three files in the same folder/directory.
▢	You may not change the file names. Otherwise, the grading script will not work.

intervals.out is a sample of what your output should look like when your program is complete. You do not have to edit this file. **intervals.in** is the file that your program will be reading from.

If you would like to turn on autosave in VSCode, click on **File -> Autosave.**

Problem Description

A closed interval on the number line is denoted by a pair of values as follows: **(3, 8)**. This interval represents all numbers between 3 and 8 inclusive. The first number will always be less than, or smaller than, the second number. Normally, in mathematics, an interval is represented with square brackets. For this programming assignment, you will represent an interval using tuples, which are represented using parentheses in Python.

For this programming assignment, you will be handling overlapping intervals. For instance, given two overlapping intervals such as **(7, 12)** and **(4, 9)**, you should collapse them into a single interval, **(4, 12)**. For intervals such as **(-10, -2)** and **(1, 5)**, these do not intersect and cannot be collapsed.

Requirements

Given the case of a set of overlapping intervals, collapse all of the overlapping intervals, and print the smallest set of non-intersecting intervals. These intervals should be sorted in ascending order using the lower end of each interval. Additionally, you should print the intervals in increasing order using the size of the interval.

1. Your program uses the provided text file **intervals.in** to create your final output. You will be using input redirection to read from this file.
2. You will not write code to read and create tuples from the input file. We have provided this code for you in **main()**. You may **not** make any changes to the main() function.
3. **You may not use any built-in sorting functions or methods.**
4. **You may not change the names or parameters of the functions listed. They must have the functionality as given in the specifications. You can always add more functions than those listed.**
5. **You may not import any additional external libraries in your solution.**

You will modify the following functions to implement the functionality below.

merge_tuples (tuples_list)
This method will take in a list of tuples, **tuples_list**, containing unsorted intervals, and return a list of merged tuples that will be sorted by the lower number of each interval. You may **not** use any built in sorting functions or methods to implement this function. You should implement either insertion or selection sort for this function.

sort_by_interval_size (tuples_list)
This method will take in a list of tuples, **tuples_list**, containing a list of merged tuples, and return a list of tuples sorted by ascending order based on the size of each interval. If two intervals have the size, then it will sort by the lower number in the interval. You may **not** use any built in sorting functions or methods to implement this function. You should implement either insertion or selection sort for this function.

Input

Your input will be from a file called intervals.in. Do not hard code the name of the file in your program. You will need to use input redirection to run this program using **intervals.in**. The input redirection will be handled by the shell (terminal); you will not have to implement this in your code.

You can run the program by typing the following command into your terminal:

```
python3 intervals.py < intervals.in
```

Alternatively, if you are using a Windows computer and are having trouble with input redirection <, you can run:

```
cat intervals.in | python3 intervals.py
```

Here is an example of an input file:

```
15
14 17
-8 -5
26 29
-20 -15
12 15
2 3
-10 -7
25 30
2 4
```

```
-21 -16
13 18
22 27
-6 -3
3 6
-25 -14
```

The first line of the input file is the number of intervals to analyze, between 1 and 100. This will be followed by one line for each interval. Each line will have two numbers. The first number will be the first and smallest interval number, and the second number is the second and largest interval number. These intervals will **not** be in sorted order.

- The code can assume that the file is structured properly, that all tokens are integers and that all integers are within the specified range.

Output

Code to produce the output already exists in main(). The existing code will call the methods that you wrote. Compare the output of your code with the sample output file to verify that you have the correct output.

The output for the provided input file and example above is below.

```
[(-25, -14), (-10, -3), (2, 6), (12, 18), (22, 30)]
[(2, 6), (12, 18), (-10, -3), (22, 30), (-25, -14)]
```

Grading

Visible Test Cases - 75/100 points

The program output exactly matches the sample solution's output. To receive full credit, the program must produce the correct output based on all requirements in this document and pass all the test cases.

Hidden Test Cases - 15/100 points

The program output exactly matches the sample solution's output. To receive full credit, the program must produce the correct output based on all requirements in this document and pass all the test cases.

Style Grading - 10/100 points

The intervals.py file must comply with the PEP 8 Style Guide.

If you are confused about how to comply with the style guide, paste the error or the error ID (e.g. C0116 for missing function or method docstring) in the search bar here in the Pylint Documentation, and you should find

The TAs will complete a manual code review for each assignment to confirm that you have followed the requirements and directions on this document. Deductions will occur on each test case that fails to follow requirements.

Submission

Follow these steps for submission.

Check	Description
▢	Verify that you have no debugging statements left in your code.
▢	Submit **only** intervals.py (the starter code file with your updates) to the given assignment in Gradescope. This will cause the grading scripts to run.
▢	If you have a partner, make sure you are submitting it for you **and** your partner on Gradescope. See this Gradescope documentation for more details.
▢	When the grading scripts are complete, check the results. If there are errors, evaluate the script feedback, work on the fix, test the fix, and then re-submit the file to Gradescope. You can submit as many times as necessary before the due date. NOTE: By default, only the most recent submission will be considered for grading. If you want to use a previous submission for your final grade, you must activate it from your submission history before the due date.

Academic Integrity

Please review the Academic Integrity section of the syllabus. We will be using plagiarism checkers, and we will cross-reference your solution with AI-generated solutions to check for similarities. Remember the goal in this class is not to write the perfect solution; we already have many of those! The goal is to learn how to problem solve, so:

- Don't hesitate to ask for guidance from the instructional staff.
- Discuss with peers about **only** bugs, their potential fixes, **high-level** design decisions, and project clarifying questions. Do not look at, verbalize, or copy another student's code when doing so.

Attribution

Thanks to Dr. Carol Ramsey and Dr. Kia Teymourian for the instructions template and this assignment.