ReversingLabs Written Python Development Test

This written test is a part of the ReversingLabs interviewing process. In order to promote a candidate to the next phase of the interviewing process, the solution of the test should satisfy the following criteria:

1. code readability and simplicity
2. performance complexity understanding (big O notation)
3. completeness (no more and no less than the task requires)
4. demonstrated and/or documented resource utilization awareness
5. clean code awareness (like separation of concerns, modularity, validation, etc.)
6. production best practices (like proper error and resource handling or external library usage)

The code must adhere to the PEP 8 style guide. The code should be designed so that it can be used as a library or integrated into an existing library (note: you do not need to package your solutions). You may add helper functions and name guards for demonstrative purposes.

Time taken to solve the written test is not evaluated and will not be a reason for candidate disqualification. We will hire all candidates who successfully pass the interviewing process, you are not competing with other candidates.

Task #1

Input data is contained in two disk files. Both files contain multiple entries separated by a newline character. The first file is of the following form:
`<first name> <ID number>`

The other file contains entries of the following format:
`<last name> <ID number>`

Write a program that, based on the information contained in input files, creates an output file with the format:
`<first name> <last name> <ID number>`

Example input:

```
Adam 1234
John 4321

Anderson 4321
Smith 1234
```

Expected output:
```
Adam Smith 1234
John Anderson 4321
```

Extension #1: sort output entries by the ID number
Extension #2: input data is too big to fit into main memory.

Task #2

Write a function that takes a file path and a directory path as input, moves the file to the provided directory path and outputs the new file path of the file. It is not necessary to preserve the file name.
The user doesn't know if another user is executing the same function at the same time or which arguments are used; the program should gracefully handle all possible conflicts.

- Example:
  - user1 running the program:
    - # python securely_move.py /src/file1.txt /dst/
  - user2 running the program:
    - # python securely_move.py /src/file2.txt /dst/
  - user1 should get "/dst/file1.txt" as the output,
    user2 should get "/dst/file2.txt" as the output.

Task #3

Write a function that takes a file consisting solely of lines in lowercase English letters (a-z) as input and reorders each line's characters lexicographically. The output file should contain the original lines from the input file, but in their lexicographical order. The output file path may be given as an optional argument.

- Example:

    Input file:

    this
    is
    an
    example

    Sorted rows (intermediate step):

    aeelmpx
    an
    hist
    is

    Output file:

    example
    an
    this
    is

Because:

"aeelmpx" ("example" sorted) is before "an" ("an" sorted) and so on.

Warning: Assume that the file fits into memory and that its lines are very long. Optimize the complexity with this in mind.