Elevator Maintenance

===================

You've been assigned the onerous task of elevator maintenance - ugh! It wouldn't be so bad, except that all the elevator documentation has been lying in a disorganized pile at the bottom of a filing cabinet for years, and you don't even know what elevator version numbers you'll be working on.

Elevator versions are represented by a series of numbers, divided up into major, minor and revision integers. New versions of an elevator increase the major number, e.g. 1, 2, 3, and so on. When new features are added to an elevator without being a complete new version, a second number named "minor" can be used to represent those new additions, e.g. 1.0, 1.1, 1.2, etc. Small fixes or maintenance work can be represented by a third number named "revision", e.g. 1.1.1, 1.1.2, 1.2.0, and so on. The number zero can be used as a major for pre-release versions of elevators, e.g. 0.1, 0.5, 0.9.2, etc (Commander Lambda is careful to always beta test her new technology, with her loyal henchmen as subjects!).

Given a list of elevator versions represented as strings, write a function solution(l) that returns the same list sorted in ascending order by major, minor, and revision number so that you can identify the current elevator version. The versions in list l will always contain major numbers, but minor and revision numbers are optional. If the version contains a revision number, then it will also have a minor number.

For example, given the list l as ["1.1.2", "1.0", "1.3.3", "1.0.12", "1.0.2"], the function solution(l) would return the list ["1.0", "1.0.2", "1.0.12", "1.1.2", "1.3.3"]. If two or more versions are equivalent but one version contains more numbers than the others, then these versions must be sorted ascending based on how many numbers they have, e.g ["1", "1.0", "1.0.0"]. The number of elements in the list l will be at least 1 and will not exceed 100.

Languages

=========

To provide a Python solution, edit solution.py

To provide a Java solution, edit Solution.java

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Python cases --

Input:

```
solution.solution(["1.11", "2.0.0", "1.2", "2", "0.1", "1.2.1", "1.1.1", "2.0"])
```

Output:

```
    0.1,1.1.1,1.2,1.2.1,1.11,2,2.0,2.0.0
```

Input:

```
solution.solution(["1.1.2", "1.0", "1.3.3", "1.0.12", "1.0.2"])
```

Output:

```
    1.0,1.0.2,1.0.12,1.1.2,1.3.3
```

-- Java cases --

Input:

```
Solution.solution({"1.11", "2.0.0", "1.2", "2", "0.1", "1.2.1", "1.1.1", "2.0"})
```

Output:

```
    0.1,1.1.1,1.2,1.2.1,1.11,2,2.0,2.0.0
```

Input:

```
Solution.solution({"1.1.2", "1.0", "1.3.3", "1.0.12", "1.0.2"})
```

Output:

```
    1.0,1.0.2,1.0.12,1.1.2,1.3.3
```

Use verify [file] to test your solution and see how it does. When you are finished editing your code, use submit [file] to submit your answer. If your solution passes the test cases, it will be removed from your home folder.