Introduction

As part of our standard recruitment procedures, we ask programming candidates to undertake a small programming exercise. This should be written in Python. The exercise consists of:

- Part 1: writing a program to solve a problem
- Part 2: writing an analysis of other solutions to the problem.

Please design and code this program alone. While you will have colleagues in your normal working environment, we need to see how good you are, not how good your current friends or colleagues are. If we hire you and your level of competence is not as indicated by this test, we will have to carefully consider whether your employment should continue past the probation period. If you do copy code from elsewhere, for any reason (e.g. a library routine), please clearly indicate its source.

You should be able to finish this exercise in 4 hours. Some of the best people can complete it in much less time, but the time that you take will depend ultimately on your skill and the amount of care you use.

All code, comments and documentation should be in English (ideally using British spelling conventions).

Submission

You should submit your complete source code and the analysis document (please note the names required for the files) in a zip file with the name listSorting-PythonTest_<your name>.zip - for example, listSorting-PythonTest_ChiMo.zip. You should email it to yding@nopsec.com and msidagni@nopsec.com in the subject line "Python test results from <your name>".

Part 1: Programming Problem

You are to write a program that takes a list of strings containing integers and words and returns a sorted version of the list.

The goal is to sort this list in such a way that all words are in alphabetical order and all integers are in numerical order. Furthermore, if the nth element in the list is an integer it must **remain** an integer, and if it is a word it must **remain** a word.

In addition, the strings and integers may contain characters that are ascii symbols that neither belong to letter set nor digit set (i.e. "#", "%", ";", etc). You are required to remove them during the process so that the output will contain only letters or digits. For example, if a string is "sym*bo+l", the output should be "symbol". If an integer is "12%3", the output should be "123".

You don't have to worry about handling the following case:
Strings or integers that contain only non-letter-non-digit characters, like "^!?", "&", etc. We will not test your program with this.

Guidelines

The guidelines below are **indications** of maximum size only. Your program should cope with longer words. However, you can use this information to help you select your algorithm.

String length:	100 characters maximum (ASCII encoding only)
Integer range:	-999999 to 999999
Number of words in string list:	100,000 maximum

Input/Output

The input will be a file includes a single, possibly empty, line containing a space-separated list of strings to be sorted. Words will not contain spaces, will contain upper-case, lower-case letters a-z and maybe non-letter-non-digit characters. Integers will be in the range -999999 to 999999, and might also contain non-letter-non-digit characters.

The program must be printed into a file named "result.txt". The content of the file is the list of strings, sorted per the requirements above. Strings must be separated by a single space, with no leading space at the beginning of the line or trailing space at the end of the line.

The program should take the input file name as the first argument, output file as the second argument:

```
root:#  ./listSorting.py <path-to-input-file>/list.txt <path-to-output-file>/result.txt
```

Examples

You will find some test files sent along with this documentation. They are only made for the purpose of helping you to understand the question. Feel free to change them and make up any test cases you like. We will test your program with another test file.

Quality

Quality is paramount. You should make sure that your program is coded in a professional manner and it should be thoroughly commented throughout. While running time is important to us, we do not need you to spend a lot of time tuning and we would like to see your first correct effort. Spend more time getting it right than getting it fast. Also make sure that your program is able to deal with exceptions.

Part 2: Analysis Document

In addition to the program, write an analysis of the algorithm you have chosen and other possible algorithms to solve the problem. Look at the expected running time of the different algorithms as a function of the number of words and the number of duplicate words. This should be no more than 4k of plain ASCII text. It should be stored in a file called **analysis.txt** and submitted along with your program.