**CoGS assignment**

**Lili Feng**

***Problem 1: Efficient implementation***

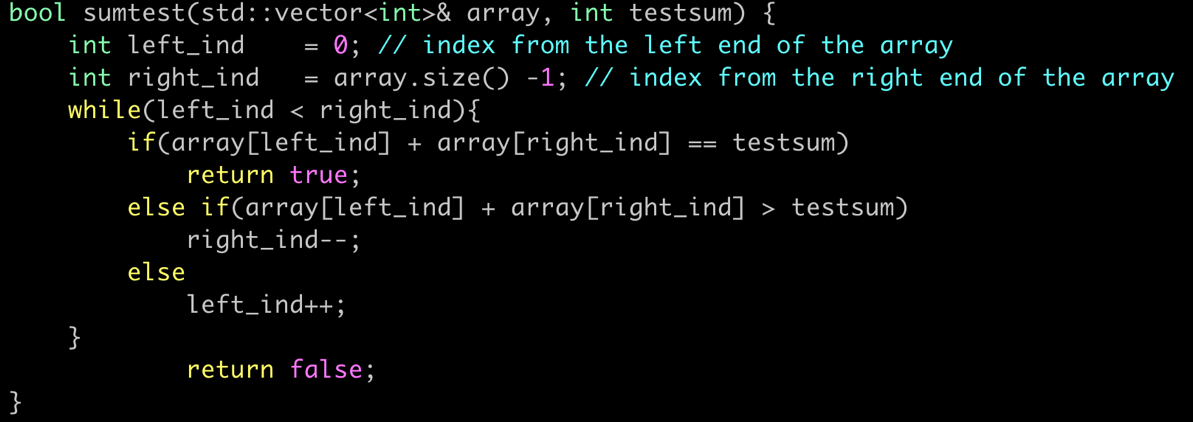
1. Write a function “sumtest(array of ints, int)” that tests whether any combination of two elements in the array sums up to a testsum and returns a boolean if this is so. Assume that the array of integers is sorted. For example,

sumtest([1,5,7,9], 6) should return true because 1 + 5 = 6

sumtest([1,5,7,9], 7) should return false.

Because this problem is about efficient implementation, so I choose to use C++ rather than Python to solve the problem.

The function is in the cpp file with path “./problem\_1/sumtest.cpp” and below is a screenshot of the function.



***Figure 1. Screenshot of the sumtest function***

**Instructions to compile and run the code**

The code has been successfully compiled with gcc 4.8.5 (the default gcc version comes with conda install -c anaconda gcc) on Linux and MacOS machines.

To compile the code:

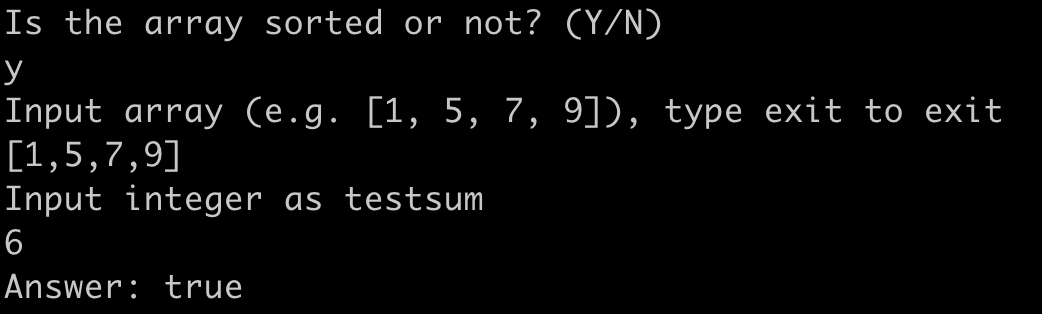
$ cd ./problem\_1

$ ./compile.sh

To run the code:

$ ./sumtest

And below is a screenshot of an example of sorted array



1. What is the fastest implementation that you can construct?

***Figure 1. The “shrink” algorithm***

Fig. 1 illustrates the algorithm that I implement in the function. There are two indices start with pointing to the left and right ends of the input array. The left index will move to the right if the sum of the two elements from the array is too small, while the right index will move to the left if the sum of the two elements from the array is too large.

1. What is its runtime as function of the length of the array N?

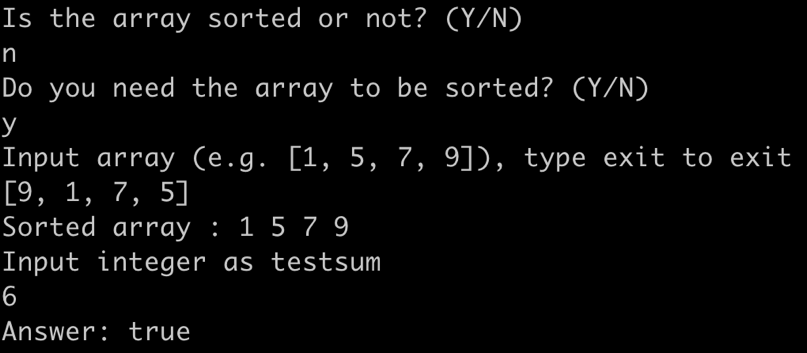
My implementation has a time complexity of O(N), with the worst case that the whole array has to be scanned.

1. How does the runtime change if the array needs to be sorted first?

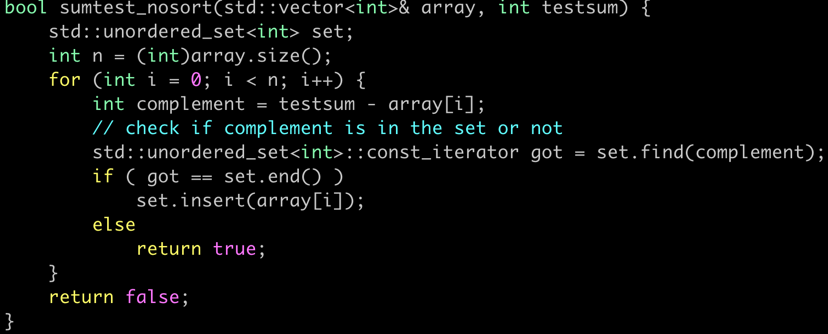
If the array is not sorted, there are two ways to solve the problem.

* 1. Sort the array at first, and then use the same sumtest function mentioned above.

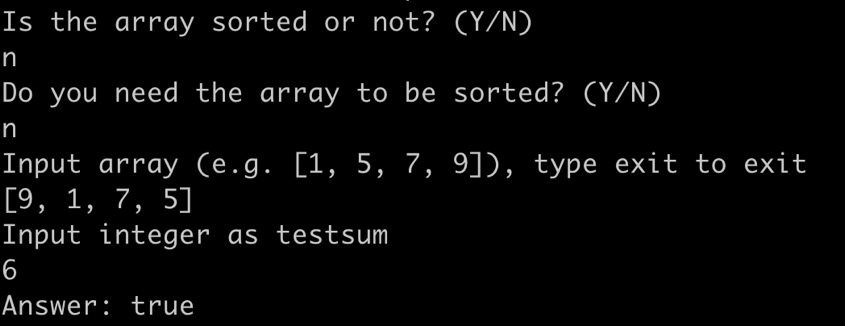
In C++, there is a sort function can be applied to sort the input array. The time complexity is O(Nlog(N)) for sorting. Note that the time complexity for running the sumtest is O(N). As a sum, the time complexity will be O(Nlog(N)) + O(N) ~ O(N(log(N)+1)). Below is a screenshot of running the code with an un-sorted array.



* 1. In fact, **there is no need to sort the array**. Below is the sumtest\_nosort function for un-sorted array



The function uses unordered\_set. Because the time complexity for operation of unordered\_set is O(1), so the time complexity for the sumtest\_nosort is O(N). Below is a screenshot of running the code with an un-sorted array without sorting it.



***Problem 2: Web Application***

In a web framework of your choice (Django, Flask, etc.) implement an API allowing users to post anonymous messages.

* GET /
  + get a JSON body with a list of messages and a total count
  + accepting the following parameters
    - page: page number (default: 1)
    - per\_page: number of items per-page (default: 20, max: 100)
* POST /
  + expect a JSON body and store it along with the following keys:
    - source\_ip
    - timestamp (in UTC)

So for example:

POST / {“numbers”: [1, 2, 3]}

GET /

{“messages”: [{“numbers”: [1, 2, 3], “source\_ip”: “1.2.3.4”, “timestamp”: “2019-01-01T12:34:56Z”}, “total\_count”: 1]

GET /?page=2

{“messages”: []}

*Don’t worry about:*

*- authentication*

*- the exact formatting of output, so long as the idea is conveyed*

*- data storage between runs, but feel free to include that if time permits or it makes things easier for you*

**Ans:** I choose to use flask to solve this problem, and test of “GET” and “POST” request is performed with **Postman** on MacOS. The data is stored locally on a “**data.json**” file. The python script with the code is “./problem\_2/solution.py”. Below is a screenshot of the code.

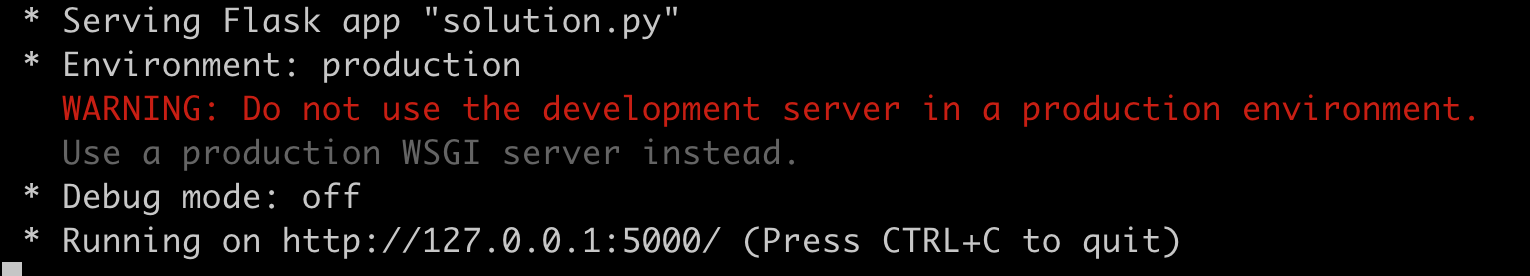


To run the web application

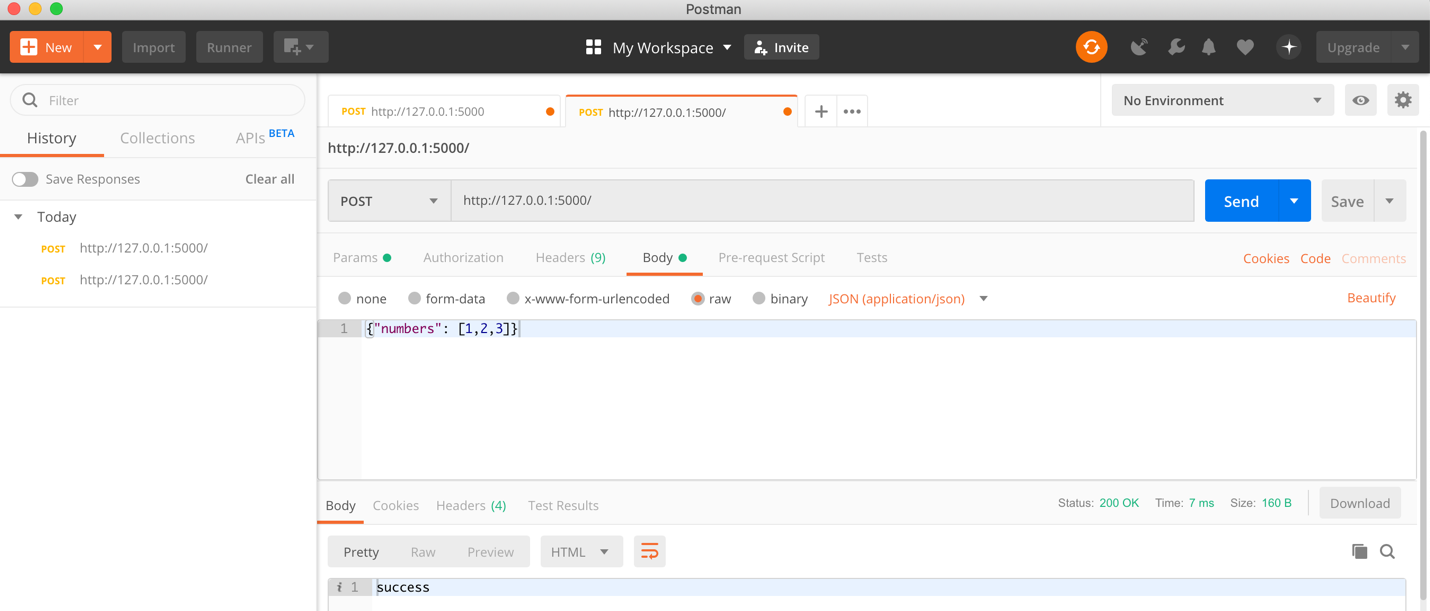
$ cd ./problem\_2

$ export FLASK\_APP=solution.py

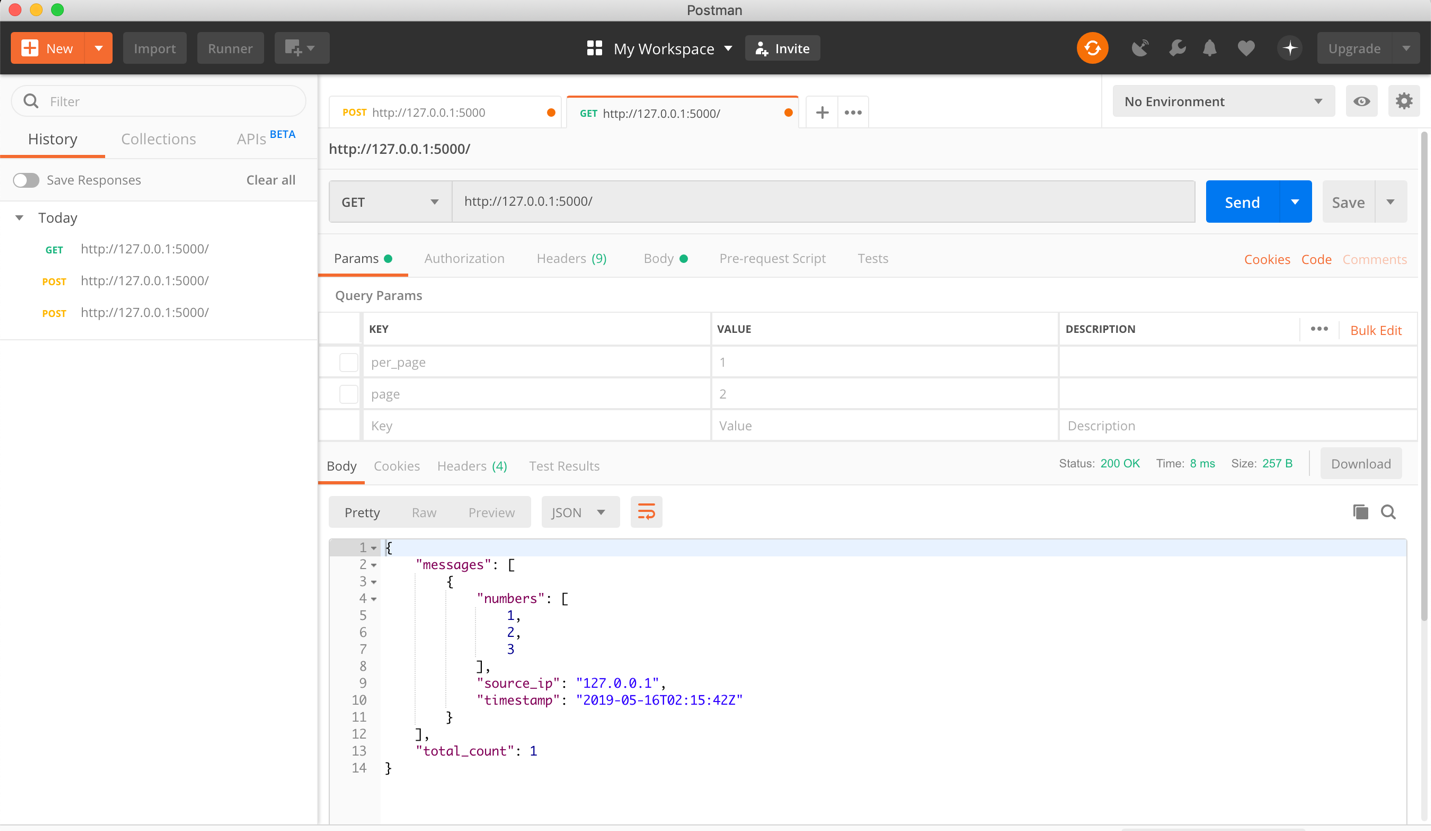
$ flask run



Then, use the URL link: <http://127.0.0.1:5000/> with the **Postman** to POST messages



The posted messages are stored in a local “data.json” file, we can get the information with GET request.



***Problem 3: Data pipeline in science***

*Preparation*: (0) download problemset\_home.zip in this google drive and unpack it. It contains a data folder and two jupyter notebooks. (1) Install and start a SQL server of your choice, [configure](https://tutorials.datajoint.io/setting-up/local-database.html) and generate a user, and (2) install datajoint for python (pip install datajoint).

*Introduction:* A common problem in scientific programming is the integration of data pipelines where multiple steps are executed consecutively with a set of shared and derived parameters.

Open the *neuro\_pipeline.ipynb* notebook. In it, you will find a typical scientific data pipeline that executes the following five steps: (a) Read a datafile [voltage as function of time], (b) Plot the data, select a reasonable threshold and check whenever the voltage exceeds the threshold. (c) From these threshold crossings, define spiketimes. See the notebook for further information as to what a *spike* is in neuroscience. (d) From the spiketimes, calculate the inter-spike-intervals, i.e. the time between two spikes. (e) Fit the interspike interval distribution with a log-normal distribution and print the fitparameters.

In the course of this exercise, you will expand this data pipeline and change the backend to a SQL database.

**Task 1**: (a) Write a new function to fit the inter-spike-intervals with a Poissonian distribution, calculate and print the mean. (b) replace the fixed threshold of -40 with a method to adapt to different noise level, i.e. fix the threshold to negative 5 times the standard deviations of the voltage values.

Please use Jupyter Notebook to open the **modified** “./problem\_3/neuro\_pipeline.ipynb”

**Task 2**: (in words) how would **you** describe the essential blocks of this pipeline? What is the relational structure? Are there obvious things about the code that should be improved?

The essential blocks of this pipeline include loading data, finding spikes and statistical analysis of the spike data.

Relational structure is a data structure that storing data instead of a single large table but with multiple tables and links/relations between those tables. In this case, we may have a table of users, a table of experiment id and a table of the data.

One obvious thing that I notice is that setting the threshold value to be negative **5 times** the standard deviations of the voltage values may result in **zero** spikes detected. An adaptive algorithm to determine the threshold value may be desirable.

**Task 3**: write a datajoint pipeline, as you have outlined in (2). As a guiding tutorial, we have provided you with the file *datajoint\_example.ipynb* . Remember to include a “computed” blocks, including the one for the log-Gaussian and the Poissonian fit.

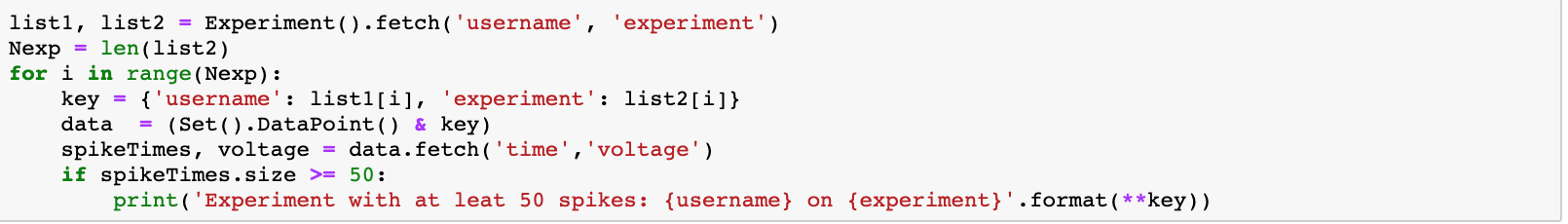
Please check the “./problem\_3/datajoint\_lfeng.ipynb” file.

**Task 4**: After constructing the database, populate it with all the datafiles in the /data folder (use populate).

Please check the “./problem\_3/datajoint\_lfeng.ipynb” file.

**Task 5**: As all data is now managed with a SQL backend, how would you query this data from the database directly? Provide the queries to (a) get the means of the Gaussian fits and (b) get all Experiments with at least 50 spikes.

1. Use **LogGaussianFit().fetch('mu')**

****