Analyzing data from packet capture

You all are familiar with the concept of *packet capture*, in which a network analyst collects data on the packets being trasmitted across a network, so that he or she can analyze this dataset for interesting or alarming features. For example, this file (labs/pcap.csv.gz) contains a packet capture session on a real network, organized into CSV format (first uncompress it with `gunzip` - it's big, so open it using less, head, or tail, rather than OpenOffice or Excel). Each packet observed is given a unique numerical ID, a time in seconds after the capture began that the connection began, the identifier of the source (IP or MAC address), the identifier of the destination, the type of protocol used, and the length, in bytes, of the packet.

I asked our NSA Visiting Professor, Paul Troncone, to give me a list of things he might want from such a packet capture while performing an analysis. He gave me:

- IPs that do the most/least talking by numbers of packets
- IPs that do the most/least talking by total number of bytes
- Protocols handling the most/least talking by numbers of packets
- Protocols handling the most/least talking by total number of bytes
- The most and least common connections (source IP/destination IP pairs)
- The connections (source IP/destination IP pairs) moving the most data

Part 1: Due Nov. 25

Within a file called `pcap.py`, make a class called `PCAP`. The goal of this class is to read in a PCAP file and generate statistics based on the traffic in that PCAP. It should have:

- **(10 pts)** A constructor, which accepts the filename of a .csv file in the format of the file linked to above and reads in the data from that file.
- **(15 pts)** A method called `packetsBySource`, which returns a list of tuples, where the first element of the tuple is the source (not the destination) address, and the second element is the number of packets for that address in the file.
- **(15 pts)** A method called `bytesBySource`, which does the same, but with the second number in the tuple being the total number of bytes that each IP address has sent.
- **(15 pts)** A method called `packetsByProtocol`.
- **(15 pts)** A method called `bytesByProtocol`.
- **(15 pts)** A method called `packetsByConnection`, in which the tuples are, in order, the sending IP, the receiving IP, and the number of packets. For instance, if in the entire file address `A` sends 10 packets to address `B` there should be a tuple `(A, B, 10)`. For the purposes of this function, consider connections to be directed, i.e. `(A, B, 10)` means that `A` sent `B` 10 packets
  . If `B` sent `A` any packets, that would be in a separate tuple.
- **(15 pts)** A method called `bytesByConnection`.

Strategy: for each of these functions what you are doing is aggregating rows of data based on the information in a target column. For instance, in `packetsBySource` you need to add every row in the file together that has the same source IP address and output a list of addresses with the number of times they were seen in the file. Another way of saying this is that you need to have a counter (a variable) for each address and increment it by one every time you see a line with that address in it. What ADT does that sound like? You don't need to implement any data structures in this lab, you can use the ones built into Python.

Part 2: Due Dec. 2

Once you have the aggregate data from Part 1, you need to sort it by packets/bytes. We could use Python's built-in sort function, but that wouldn't work very well because our data is *tuples*, containing labels (source address, protocol, connection, etc.) as well as the data you want to sort on (packets/bytes). Python doesn't know that the second value in the tuple is what you want to sort by! You will start by writing your own sort function that lets you specify what column to sort by, then you will learn how Python can be made to sort by specific values inside of a list.

For Part 2, you should:

- **(40 pts)** Write a utility function in `pcap.py` that is outside of your PCAP class (in the global scope) called `slowSort(lst, col)` which sorts the list of tuples `lst` based on the information in the column specified by the variable `col`. For example, if `col=1` then it would sort based on the second value in each tuple (1 = second because we are using zero-based indexing). `slowSort` should use either Selection Sort or Insertion Sort, your choice.
- **(40 pts)** You should also write a second sorting function `fastSort` that takes the same arguments but uses Merge Sort to sort the list of tuples.
- **(20 pts)** Read the Python documentation for its `sort` function here (https://docs.python.org/3.3/howto/sorting.html). There is also a walkthrough here (https://www.programiz.com/python-programming/methods/list/sort) that you might find useful. Write a function `pythonSort(lst, col)` that uses Python's built-in sort method to correctly sort the list of tuples according to the specified column. This means you will have to use the `key` argument as described in the Python documentation.