Q1a) When the following is the input to the Q1a program:

```
src   tgt   weight
100   10    7
200   10    7
110   10    2
100   130   30
110   130   67
10    101   15
```

There are 6 mappers that process a single line of input and write to context the output, <sourceNode, weight>.
The inputs and outputs are shown below.

Mapper	Mapper Input	Mapper Output
1	100 10 7	<100, 7>
2	200 10 7	<200, 7>
3	110 10 2	<110, 2>
4	100 130 30	<100, 30>
5	110 130 67	<110, 67>
6	10 101 15	<10, 15>

The mapping doesn't necessarily occur in any specific order. The map output is then shuffled and sorted and the result is passed to the reducer. Each reducer has all mapped output for the same key for this program. The reducer iterates through all inputs to it, and finds the maximum of all the values passed to it for the same key. It then outputs the key that was passed to it as the key and the maxWeight calculated as the value. There is a combiner here as well, which takes the mapper output and reduces while there is data being mapped. Once the data is completely mapped all data is reduced, including the combiner output. The reducer output is shown below.

Reducer	Reducer Input	Reducer Output
1	<100, 7>, <100, 30>	<100, 30>
2	<200, 10>	<200, 10>
3	<110, 2>,<110, 67>	<110, 67>
4	<10, 15>	<10, 15>

There is no particular order in which the reduce job is completed. The final output of the program for the given input is:
10 15
100 30
110 67
200 10

Q1b) For this program the concept of secondary sorting was used. There was a composite key created which was used as the map output key and the value of the source node was used as the map output value. When the records arrive at the reducer the mapreduce framework would be automatically be sorted by the key in a certain order. The order was specified by the programmer in this case as the compareTo method of the composite key class was overridden. The composite keys would firstly be sorted by the target node of the edge in ascending order, then by the weight of edge in descending order, and then by the source node of the edge in ascending order.

The mapped output would then be partitioned using a custom partitioner which instructed the reducer to only consider the target when taking the mapped output. However, the partitioner merely routes the key, value pairs with the same target to the same reducer. It's the GroupComparator that makes sure that the reducer sees all input it receives as those having the same key. The GroupComparator has a compare method which takes into consideration only the target when comparing two composite keys. Therefore, the GroupComparator makes sure that the reducer actually considers the set of records sent by the partitioner to be of equal key. Our end goal for this program is to make it seem like to the reducer that the set of key, value pairs it gets, have a single key and all values belong to the same key. The whole point of secondary sorting is to take advantage of the map reduce framework to do the sorting job for us, so that the reducer doesn't have to do anything but take the first value of it as the output value and the target node of the first key to be the output key. The input as it arrives to the reducer might look like this.

Reducer	Key	Value
1	10 7 100	100
1	10 7 200	200
1	10 2 110	110
2	130 67 110	110
2	130 30 100	100
3	101 15 10	10

The output of the reducers might look like,

Reducer	OutputKey	OutputValue
1	10	100
2	130	110
3	101	10

The final output would be:

10	100
130	110
101	10.