Q1

a) Write a MapReduce program in Java to report the smallest weight among all the weighted inbound edges for each node in the graph

Input:

100 10 3
110 10 3
200 10 1
150 130 30
110 130 67
10 101 15

How my MapReduce program handles this

Mapper

- Breaks each line above into tokens (tab separated). Each line would result into an array of 3 tokens
- Wastes the first token and sets the next two tokens as key and value respectively.

After the entire input has been parsed, Mapper's output would look like this,

Key	Value
10	3
10	3
10	1
130	30
130	67
101	15

Reducer

- Now that the values for each key have been grouped into an iterable list, the reducer runs through the list and picks the minimum value for each key
- The reducer then writes into the context the key and the minimum value picked above.

After the reducer has run the output would look like this which is also the output of the MapReduce program,

Key	Value
10	1
130	30
101	15

b)To accomplish this, we will use the department ID as the key, as it is the only common thing between the two different types of data in the file. We will write the MapReduce program in such a way that the output is sorted and one type of data is ordered ahead of another type of data for the reducer. So, for the same department ID key we'll make sure that the name of the department is ordered before the names of students. Also, we'll make sure that the final output is ordered because we'll define our own compareTo function for keys and our own partitioner to partition keys between reducers.

```
class key:
        int id
        int tag
        function compareTo(key):
                val = this.id.compareTo(key.id)
                if (val == 0):
                        val = this.tag.compareTo(key.tag)
                return val
class partitioner:
        function getPartition(key):
                return key.id.hashCode() % number_of_partitions
class mapper:
        function map(line, context):
                words = line.split(",")
                if (words[0] == “Department"):
                        key.tag = 1
                        key.id = parseInt(words[1])
                        value = words[2]
                        context.write(key, value)
                else:
                        key.tag = 2
                        key.id = parseInt(words[2])
                        value = words[1]
                        context.write(key, value)
class reducer:
        function reduce(key, values, context):
                int i
                departmentName = values[0]
                for(i = 0; i < values.length; i++):
                // Skip all other values which are department names
                        if (! values[i].toString() == departmentName):
                                break
                for (int j = i; j < values.length; j++):
                        text = ""
                        text = text + key.id.toString() + ", "
                        text = text + values[j].toString() + ", "
                        text = text + departmentName
                        context.write(nullKey, text)
```

Trace of the input
Output of Mapper: key: [values]
        1234: [CS, Alice, Bob]
        1123: [CSE, Carol]
Output of Reducer:
        1123, Carol, CSE
        1234, Bob, CS
        1234, Alice, CS