Q1a) The mapReduce algorithm is designed to find the largest outbound edge in the given graph. The program does this by creating a Hadoop job client, which submits the *jar* to the Resource Manager, which schedules the tasks to the Node Managers and monitors them.

A Map is a function which transforms the input data into a key-value pair and puts the entries with the same key in a iterable list. From the toy graph in the question, the input to the Mapper is:

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

In the code, the mapper allocates the source 'src' in a variable 'word1' and target 'tgt' in another variable 'word2'. The weight is stored in a IntWritable called 'one'. 'word1' is set as the key and weight as the value. The output of the mapper is a key and a iterable list of weights like as follows:

```
100 -> {7,30} , 200-> {7}, 110, -> {2,67}, 10 ->{15}
```

Now, Reduce is a function which collects all the keys and performs some computation on them, *reducing* them to single value. From the code, the reduce function will iterate through the list 'values' and for every key, find the max weight. This weight is then written as the output of the reduce class. For the toy graph, the output of the reduce function would look as follows:

```
Key   value
10    15
100   30
110   67
200   7
```

Thus we get the biggest outbound edge in the graph. This method shows a dramatic increase in performance compared to using loops and hash-maps to find the same values. The Hadoop framework is designed with big data in mind, and is built to handle multi-terabyte datasets in a parallel manner on large clusters. The compute nodes and storage nodes are usually the same, ie. MapReduce framework and Hadoop Distributed File System are running on the same set of nodes. This allows a high aggregate bandwidth across the cluster. As the input data is divided in chucks which are processed in a completely parallel manner, the systems gives very high throughput.

Q1b) The advanced mapReduce algorithm is used to sort the nodes before they reach the Reducer. We do this by making a custom GroupingComparator, which sorts the nodes before giving them to the reducer.

We do this by making a custom Composite Key, by defining it in our class in Java. Certain functions of the key, like set(), get(), and compare() are written to define it as a valid key object. The compareTo() function, (Line 84) is the function where the sorting occurs. We describe this in detail later. We also override the toString function, to have our key print out the output in a custom defined format.

The Mapper function reads 3 values from the input tsv file and puts them in an instance of the composite key. This value is sent to the grouping comparator. So, for the toy graph, the output of the Mapper function would be the input data as the key, and a null value as the value.

The grouping comparator gets these values. It takes 2 Composite key elements and uses the compareTo() function on each of them. The compare to function first checks if the targets are similar, then compares the weights. We multiply each return value for the weight by -1 to sort it in the descending order. After this if the compare value is still zero, the source is checked and compared to the other source. This sorts the src in ascending order. (Line 84-90) For the toy graph the output of the Grouping comparator would look like:

```
           tgt   src   wt
<KeyPair>  {10   100   7}
<KeyPair>  {10   200   7}
<KeyPair>  {10   110   2}
<KeyPair>  {101  10    10}
<KeyPair>  {130  110   67}
<KeyPair>  {130  110   30}
```

The figures shown above are the keys to the reducer. The value to the reducer, would ofcourse be a iterable NullWritable. The Reducer simply takes these entries and prints the first one it gets for each distinct key.
The output of the Reducer for the toy graph will be:

```
tgt   src
10    100
101   10
130   110
```

In this way, we accomplish secondary sort, using the least computationally intensive method way possible. The alternative to sort the values in the sorter would be faster, albeit require a significantly larger amount of memory to run. There is a high risk of running in to ‘out of memory errors’ using the alternative.