Describe your map/reduce algorithm for solving the three's company problem.

a. Describe the operation of the mapper and reducer. How does this combination solve the three's company problem?

b. What is the potential parallelism? How many mappers does you implementation allow for? Reducers?

Mapper :
Lets call the first person of the file of friends as Host.
I create a keys with 3 elements : <host, friendX, friendY> where friendX and friendY are unique pair of friends.
I sort the 3 elements of this key such that all the keys produced by any mapper is in sorted order.
Eg : if 100, 200, 300 form a triangle : there will be 3 keys produced
<100,200,300>
<100,200,300>
<100,200,300>
Value of all key is 1.

Reducer :
The reducer takes the set of keys and counts the number of times the keys is repeated.
If the key is produced 3 times, the set of 3 friends form a triangle.
The 3 lines of triangle are produced in sorted order
Eg :
<100,200,300>
<200,100,300>
<300,100,200>

On combiners

a. Why do we leave the combiner class undefined in Step 4?

We do not need the combiner class
Combiner is run on the mapper output and reduces the input to reducer. It is often replaced by reducer
My output after the mapper is the least size possible for maximum parallism (as I have used a sorted key) so I don’t need a combiner.

Let’s gently analyze the total work done by the algorithm.

a. How many messages do your mappers output? Assuming the Hadoop runtime has to sort these messages how much total work does the algorithm do (in Big-O notation)? You should assume that there are n friends list each of length O(n).

b. How does this compare with the performance of serial implementations linked here? Describe the tradeoff between complexity and parallelism (*Note*: it is more reasonable to compare your implementation with the naive O(n^3) algorithm than the optimized O($n^{2.37}$) algorithm.)

a) There are n people with n friends each
Each person will produce (n^3)/2 Messages
Total messages (worst case)=(n^3)/2

Sorting algorithm order = O(nlogn)

Thus overall order of the algorithm = O(n^3)
But because we have parallelized reducers working, same keys go to the same reducer. This Algorithms becomes ~**O((n^3)/6num_of_procs)** as there will be a max of(N^3)/6 unique keys distributed among processors

As compared to the naïve O(n^3) method : the present algorithm is producing only unique triples in sorted order, thus there is no redundant pairs produced, (because of always being sorted)
Eg : O(n^3) algo produces <100,200,300> in all the 6 combinations. This algorithm is thus not parallalised effectively as there will be a lot more times the reducer will be called O(n^3) times.

Present algo only produces <100,200,300> in a sorted order 3 times.

In the O(n^2.37) algorithm, there is a betterment due to matrix multiplication efficiency. We don't need a matrix multiplication in the parallelized version. Instead we distribute out mapped keys to processors.

We are compromising order of algorithm, and adding on steps to make the algorithm concurrent. Thus we can now handle large amount of data in this method. Also we are adding on costs