1. **Psuedo code for the map-reduce function to return inverted index:**

def mapper(filename, file-contents):

**for** **each** word **in** file-contents:

**emit**(word, filename)

def reducer(word, filenames):

contains = []

**for each** filename **in** filenames:

**if** filename **not in** contains:

contains.**append**(filename)

**emit**(word, contains)

**Reasoning**: The mapper will emit each word in each file, along with the filename that the word is found in. The reducer will then first initialize an empty list. For each word emitted, it will check to see if the filename has already been seen for the word. If the filename is not in the list of filenames that contain the word, it will add the filename to the list ‘**contains**’. The reducer will then **emit** the word along with the list **‘contains’**, which has the filenames of the files that the word is present in.

1. **MapReduce Data Flow**
2. A file is loaded from HDFS storage
3. The file is read using InputFormat to attribute how the file is split and read.
4. InputSplits will create mapper tasks for each split of the file.
5. RecordReader (RR) will feed the mapper key-value pairs for the mapper to execute its function.
6. The Mapper function (the “Map” of MapReduce) will take each key-value pair from the RecordReader and apply a map function to output another key-value pair.
7. (Optional) Combiner will aggregate the output from the mapper to make reducing more efficient.
8. A partitioner will take the output from the from the combiner to allocate the data to multiple reducers.
9. ShuffleSort - The data from the partitioner is shuffled and moved over to separate reducer nodes.
10. Reducer - The data output after mapper - combiner/partitioner/shufflesort is passed into the Reducer, which takes the key-value pairs and applies the reducer function to generate a final output.
11. OutputFormat - the output key-value pairs from the reducer and writes the output back into HDFS.

**3a.** The storage daemons are the NameNode (master), which comprises the metadata for the HDFS, such as the directory, and tracks file location of all files. There is also a secondary NameNode that acts as a backup for the NameNode. The DataNode (slave) stores the data in the HDFS.

**3b.** The Job Tracker will monitor and manage jobs that process the data using MapReduce. TaskTracker runs on each individual machine and will initiate and monitor MapReduce tasks on each machine

**4.** A combiner is effectively a “mini-reducer” in that it functions similarly to a reducer by combining records with the same Key prior to passing the Key-Value pairs to the reducer. The combiner is important as it can reduce the total number of Key-Value pairs that the reducer has to process. An example of this is as follows:

Input:

These words are words  
Words are words and

Mapper Output:

Mapper #1

(These, 1), (words, 1),(are, 1),(words,1)

Mapper #2

(Words,1),(are,1),(words,1),(and,1)

Without a combiner, the reducer must process the following 8 Key-Value pairs:

(These,1),(words,1,1,1,1),(are,1,1),(and,1)

Whereas with a combiner, the reducer would instead be fed 4 Key-Value pairs:

(These,1),(words,4),(are,2),(and,1)

Thus, with a combiner, the time taken and amount of data sent to the reducer are decreased.

**5.** Secondary sort is used when we are sorting data by more than one key. For example, if we were sorting geospatial data, we would need to sort on both Latitude and Longitude, which would require us to use a secondary sorting method. Secondary sorts will also introduce a composite key that allows a natural key to be referenced and a natural value that allows the keys to be sorted.