Q1.
A.

```
// map #1: returns the pair of (browser, IP)
For each input key-value pair (linenum, linetxt):
      output key-value pair (extract_browser(linetxt), extract_IP(linetxt))

// shuffle #1:
// shuffle creates a value list (browser, (IP1, IP2, ....))
// IP_list = (IP1, IP2, ....), all IPs that are using the browser

// reducer #1:
// returns tuple (browser, number of users of browser)
For each input key-value pair (browser, IP_list):
      num_users = count distinct (IP_list)

      if num_users >= 100:
            output key-value pair (browser, num_users)
```

B.
Example input:
(1, 169.122.23.15 - frank [10/Oct/2000:13:55:36 -0700] “GET /apache pb.gif HTTP/1.0” 200 2326 “http://www.example.com/start.html” “Mozilla/4.08 [en] (Win98; I ;Nav)”)
(2, 169.124.21.16 - frank [10/Oct/2000:13:55:36 -0700] “GET /apache pb.gif HTTP/1.0” 200 2326 “http://www.example.com/start.html” “Mozilla/4.08 [en] (Win98; I ;Nav)”)
(3, 169.152.23.15 - frank [10/Oct/2000:13:55:36 -0700] “GET /apache pb.gif HTTP/1.0” 200 2326 “http://www.example.com/start.html” “Mozilla/4.09 [en] (Win98; I ;Nav)”)

Map Output:
(“Mozilla/4.08”, “169.122.23.15”)
(“Mozilla/4.08”, “169.124.21.16”)
(“Mozilla/4.09”, “169.152.23.15”)

Shuffler Output:
(“Mozilla/4.08”, (“169.122.23.15”, “169.124.21.16”))
(“Mozilla/4.09”, (“169.152.23.15”))

Reducer Output:
// (“Mozilla/4.08”, 2)
// (“Mozilla/4.09”, 1)
Nothing will get outputted since they both have less than 100 users

C.
Yes, I think a combine step would be effective by removing duplicate (Browser, IP) tuples produced by the mapper. Given that we might have duplicate IPs for a given Browser duplicate (Browser, IP) pairs will be outputted by the map step while we only count them once when using the “`count distinct`” in the reduce step. A combine step could be used

after the map step to get rid of the duplicate pairs thus reducing the number of pairs we have to pass along to the next steps and redicing the I/O cost.

Q2.
A.

```
// map #1: returns the pair of (IP, browser)
For each input key-value pair (linenum, linetxt):
      output key-value pair (extract_IP(linetxt), extract_browser(linetxt))

// shuffle #1:
// shuffle creates a value list (IP, (browser1, browser2,...))
// browser_list = (browser1, browser2,...), all browsers used by IP

// reduce #1:
// return pair of (1,IP) where IP uses more than 1 browser
For each input key-value pair (IP, browser_list):
      num_browsers = count distinct (browser_list)

      if num_browsers >= 2:
            output key-value pair (1, IP)


// map #2: Identity map
For each input key-value pair (key, value):
      output key-value pair (key, value)

// shuffler #2
// shuffle creates a values list (1, (IP1, IP2, IP3, …))
// where the IPs are all the IPs that have are using more than 1 browsers

// reducer #2
// it will be passed a single tuple, with a key of 1 and a list of IPs
// we simple return the number of elements that we have in the IP-list
// the -1 is a dummy value, we only care about the num_users
For each input key-value pair (1, IP_list):
      num_users = count distinct (IP_list)

      output (num_users, -1)
```

B.
Only 1 reducer process, because we only pass one tuble to the reducer process with the shared key if 1 and a list of the IP addresses with more than one browser as the value. So we will only have 1 reducer process that simple cound the number of IP addresses we have in the value.

Q6.

```
popcsv = LOAD '/data/pop.csv' USING PigStorage(',') AS (country:CHARARRAY, population:INT);
vax_data = LOAD '/data/vaccination-data.csv' USING PigStorage(',') AS (country:CHARARRAY,
iso3:CHARARRAY, region:CHARARRAY, numVax:INT);
vax_meta = LOAD '/data/vaccination-metadata.csv' USING PigStorage(',') AS (iso3:CHARARRAY,
v_name:CHARARRAY, p_name:CHARARRAY, c_name:CHARARRAY);

highpop = FILTER popcsv BY population > 100000;

joined1 = JOIN highpop BY country, vax_data BY country;

country_info = FOREACH joined1 GENERATE highpop::country, highpop::population, vax_data::iso3,
(float)(((float)vax_data::numVax/(float)(highpop::population*1000)) * 100) AS percent_vaxed;

brands = GROUP vax_meta BY iso3;
```

Describe:

```
brands: {
    group: chararray,
    vax_meta: {
        (
            iso3: chararray,
            v_name: chararray,
            p_name: chararray,
            c_name: chararray
        )
    }
}
```