Problem 1

Suppose you have a new computer just set up. `dig` is one of the most useful DNS lookup tool. You can check out the manual of `dig` at `http://linux.die.net/man/1/dig`. A typical invocation of `dig` looks like: `dig @server name type`.

Suppose that on April 19, 2017 at 15:35:21, you have issued "`dig google.com a`" to get an IPv4 address for `google.com` domain from your caching resolver and got the following result:

```
; <<>> DiG 9.8.3-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17779
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             239     IN      A       172.217.4.142

;; AUTHORITY SECTION:
google.com.             55414   IN      NS      ns4.google.com.
google.com.             55414   IN      NS      ns2.google.com.
google.com.             55414   IN      NS      ns1.google.com.
google.com.             55414   IN      NS      ns3.google.com.

;; ADDITIONAL SECTION:
ns1.google.com.         145521  IN      A       216.239.32.10
ns2.google.com.         215983  IN      A       216.239.34.10
ns3.google.com.         215983  IN      A       216.239.36.10
ns4.google.com.         215983  IN      A       216.239.38.10

;; Query time: 81 msec
;; SERVER: 128.97.128.1#53(128.97.128.1)
;; WHEN: Wed Apr 19 15:35:21 2017
;; MSG SIZE rcvd: 180
```

1. What is the discovered IPv4 address of `google.com` domain?
2. If you issue the same command 1 minute later, how would "ANSWER SECTION" look like?
3. When would be the earliest (absolute) time the caching resolver would contact one of the `google.com` name servers again?
4. When would be the earliest (absolute) time the caching resolver would contact one of the `.com` name servers?

Problem 1 continued on next page...

1. 172.217.4.142
2. Exactly the same; the TTL is 239 seconds, so in 60 seconds the caching resolver will just serve the IP address from cache because the record hasn't expired yet.
3. 239 seconds – that's when the google.com A record expires from cache and the Google nameservers must be contacted again for the A record.
4. 55414 seconds – in this amount of time the authoritative nameserver records for `google.com` will expire from cache and the `.com` nameservers would have to be contacted again for these records. Note that the A records for these name servers in the `ADDITIONAL` section will still be in cache when this happens, but we still need valid nameserver records to verify that these A records belong to the authoritative nameservers of `google.com`.

Problem 2

In most of cases, we rely on caching resolvers to provide recursive DNS query service for us. In this task, you will be a human caching resolver using `dig` command as your tool.

Look up an "SRV" resource record (a record that specifies the hostname and port number of a server(s) for some service) for `_ndn._udp.ucla.edu.ndn._homehub._autoconf.named-data.net`.

In your answer, include the exact commands you have used, including IP addresses of the autoritative name servers to which you were sending DNS queries, explain the returned result of each query (what is returned), and indicate for how long you supposed to cache the returned information.

You can start with one of well-known IP addresses of the DNS root servers, e.g., `198.41.0.4`.

Assuming we have nothing in cache:

1. `dig @198.41.0.4 +norecurse <url> SRV` ⇒ TTL 172800 seconds (valid in cache for this long)
   We query the root nameserver. This gives us a list of authoritative servers for the .net TLD.
2. `dig @192.5.6.30 +norecurse <url> SRV` ⇒ TTL 172800 seconds
   Now we ask the .net DNS servers if they have the SRV record for this URL. They give us the nameservers for named-data.net.
3. `dig @129.82.138.8 +norecurse <url> SRV` ⇒ TTL 86400 seconds
   Query the named-data.net name servers for the SRV record of our URL. They respond with the SRV record which contains: the hostname `spurs.cs.ucla.edu` and the port number 6363. They also respond with a list of authoritative nameserver hostnames which control the `_autoconf.named-data.net` subdomain.
4. NOTE: there could have been an extra query here. It just so happens that this particular name server had the SRV record we were looking for. If we had chosen a different nameserver from step 2 we might have had to send another query to `_autoconf.named-data.net` nameservers to get our SRV record. (I tried it and this is what happened)

The `+norecurse` option makes sure that these queries are one-off; that is, that none of these servers try to finish the query process for us recursively.

Problem 3

Suppose that you walked into Boelter Hall and get connected to `CSD` WiFi network, which automatically gave you IP address of the local caching resolver. However, initially, it doesn't allow you to do anything unless you type your username and password in a popup window (or if you try to go to any website in your browser).

1. Explain a mechanism of how does the "`CSD`" network achieve this / which features of DNS/HTTP make it possible.

After you successfully logged in, you can start using the Internet. Suppose the caching resolver has just rebooted and its cache is completely empty; RTT between your computer and the caching resolver is $10ms$ and RTT between the caching resolver and any authoritative name server is $100ms$; all responses have TTL 12 hours.

2. If you try to go to `ucla.edu`, what would be minimum amount of time you will need to wait before your web browser will be able to initiate connect to the UCLA's web server?
3. What would be the time, if a minute later you will decide to go to `ccle.ucla.edu`?
4. What would be the time, if another minute later you will decide to go to `piazza.com`?
5. What would be the time, if another minute later you will decide to go to `gradescope.com`?

1. The CSD network "tricks" you by giving the IP address of a cache resolver which resolves all non-allowed hostnames for non-whitelisted IPs (read: clients that aren't logged in) to the (local) IP address of the server hosting the login page. The cache resolver is able to do this because the client will implicitly trust that the cache resolver's response is accurate, so the cache resolver can resolve hostnames to whatever IP it likes. Then all the HTTP server has to do is serve the login page to the client, and whitelist the client's IP if the login was successful. Once whitelisted, the cache resolver will resolve hostnames for that requests from that IP normally.
2. 10ms (query caching resolver) + 100ms (query root) + 100ms (query .edu) +
   100ms (query ucla.edu nameserver for A record)
3. 10ms (query caching resolver) + 100ms (query ucla.edu nameserver for A record)
4. 10ms (query caching resolver) + 100ms (query root) + 100ms (query .com) +
   100ms (query piazza.com nameserver for A record)
   (I tried this one using `dig` and the ucla.edu nameserver had an A record for ccle.ucla.edu)
5. 10ms (query caching resolver) + 100ms (query .com) +
   100ms (query gradescope.com nameserver for A record)

Problem 4

1. An online chatting application is going viral. To optimize user experience, its developers decided to use CDN service to deliver superb chat application performance for clients around the world. What mechanisms CDN services use to help developers to do so? In your answer include specific mechanisms and basic idea how these mechanisms work.

2. What are the factors that go into designing a CDN server selection strategy? Name at least four.

1. The CDN allows for rapid delivery of messages for clients that are geographically close to each other by connecting them both to a server that is roughly inbetween or near each of their locations. The issue with a traditional centralized server approach is, for example, if the server is in the US and the clients are in China, each message sent will impose the RTT from China to the US (assuming message push). This is clearly not ideal for clients in China. Instead, a CDN router can estimate the location of each client and connect them both to a server in their vicinity (or roughly inbetween them) in order to minimize the RTT.

2. (a) Geographic location – assume the caching resolver is geographically close to the client and take your best guess as to where the request is coming from. E.g. if the request is coming from 8.8.8.8, we know that IP is registered to Google Inc., which is located in Mountain View, so send this caching resolver a record with the IP of the Bay Area node and tell it to cache the record.

   (b) Server availability/average response time (e.g. if the server for your region is under load, it might be faster to go to a server that's a little further away but more available)

   (c) RTT (ping) (correlates with location) – the routing node can estimate latency from various CDN server nodes to the client and chose the server with the lowest latency. If we cache these latencies it would be a big performance win (pinging takes an RTT) (or use Anycast).

   (d) ISP cost – it could cost more to serve from a node peered with a Tier 1 ISP, and an alternative could provide comprable performance and cost less money for the CDN. It is a business, after all.