Bradfield
SCHOOL of COMPUTER SCIENCE

April 2020

CSI Take-Home Exercises

Admissions to the Bradfield Computer Science Intensive are based on your demonstrated abilities and likelihood of success in the program.

The following exercises are intended for you to demonstrate your abilities in programming, problem-solving, and self-directed learning.

Note: Programming exercises may be completed in any language. You are welcome to use online resources, but please work independently and refrain from discussing these exercises with others.

Once you complete these exercises and **submit your work by email**, we will review the results and then reach out to you about next steps.

1 Network packet capture

Download and install Wireshark. Using Wireshark, capture the network traffic from making an HTTPS request to bradfieldcs.com.

i. Did your system make a DNS query? If so, what was the "time to live" value of the DNS answer? If not, why not?

ii. Find the `SYN/ACK` segment sent from the Bradfield server. What was the server's window size?

iii. During the TLS handshake, which cipher suite did the server select?

Please justify your answers by providing screenshots as appropriate.

2 Range count

Consider the problem of counting occurrences of a given value `x` within a sorted array `a`. For example, if `x = 5` and `a = [1, 1, 2, 4, 5, 5, 7, 9]`, then the count is 2.

i. Write a function that solves this problem by performing a linear scan.

ii. Next, write a function that solves this problem by performing two binary searches.

iii. Finally, benchmark your two functions for random sorted arrays of size 10, 100, …, up to 10,000,000. How does performance compare between the two functions?

3 Time zones

Implement an HTTP server with a single endpoint that allows the client to request the wall-clock time on the server.

- The server should return the time in 24-hour `"hh:mm:ss"` format, e.g. `"18:36:24"`
- The client should be able to optionally specify a time zone, e.g. `"America/Los_Angeles"`

Feel free to use any suitable libraries / frameworks, and to make your own design choices about the API.

4 Exponential backoff

In a distributed system, many operations have a chance of failure. For example, networks might be down, buffers might be full, or services might be restarting.

However, we can make the system more robust by introducing "exponential backoff", in which we retry failing operations with increasing "backoff" periods (wait 1 second, then wait 2 seconds, then wait 4 seconds, etc.) until we either succeed or give up.

In addition, we can add "jitter" to randomize the backoff periods and prevent "thundering herd" effects (e.g. wait a random period between 0.5 and 1.5 seconds, instead of exactly 1 second).

Write a function that performs exponential backoff. Your function should take the following inputs:

- An operation that might fail, represented in any suitable format for your language of choice (e.g. object with method, function pointer)

- Duration of backoff periods
- Criteria for giving up
- Amount of jitter

Feel free to make your own design choices about error handling and logging.

5 Dynamo

Please read the paper Dynamo: Amazon's Highly Available Key-value Store, then answer the following questions:

i. What are the parameters `R`, `W`, and `N`? How many node failures can the system tolerate while still guaranteeing that read operations will return up-to-date values?

ii. What's "eventual consistency", and what are some advantages and disadvantages? What's a specific example of behavior that could happen under this consistency model?

iii. Why does "consistent hashing" (compared to, say, `hash(key) % N`) reduce the amount of data that needs to be moved when a new node comes online?