Review Questions for Final Exam of CS 202 (Not cumulative)

Networking

1. How does a hierarchy help manage large systems?
    - How to deliver a message from one node to another in a hierarchy?
2. How does TCP (transmission control protocol) ensure an entire message is delivered from a source node to a destination node?
    - What types of problems must TCP be able to handle? (dropped packets, slow packets, out of order packets)
    - How does TCP use sequence numbers to solve those problems?
3. What can go wrong when too many nodes try to send messages through a network?
    - What solution is used in TCP? (congestion control; send fewer messages when see your messages are being dropped)
4. What is the HTTP protocol used for? How does it relate to TCP/IP?

Sorting

1. For calculating complexity: Given multiple loops specified with Scratch code, you should be able to count the number of times a given statement will execute
    a. Order complexity from least to greatest: log N, N, N log N, N^2, N!
2. What data structure is useful for sorting? (list) What is the difference between keys and data?
3. Why is sorting useful? (searching is faster, finding min/max, finding duplicates, finding gaps, TCP, google joining multiple search terms)
4. How does Selection sort work? (Idea: Repeatedly move minimum element from unsorted list to sorted list; size of sorted list increases each time while size of unsorted list decreases each time).
    a. How to use selection sort with just one list, in-place? (Idea: Swap minimum element with the key it is replacing)
    b. What is the complexity of selection sort for randomly ordered keys? ($O(N^2)$; For each of N positions in list, must find the minimum key, which requires N comparisons)
5. How does Insertion sort work? (Idea: Repeatedly move the next unsorted key into its correct location in the sorted list; size of sorted list increases each time while size of unsorted list decreases each time).
    a. What is the complexity of insertion sort for randomly ordered keys? ($O(N^2)$; For each of N keys, must compare it to all other sorted keys to find its location, which requires N comparisons)
6. Given a list before and after one iteration of the outer repeat loop, can you identify if selection or insertion sort was used?
7. Can you identify a recursive algorithms? (Contains a base case and a set of rules to reduce other cases toward base case; example: factorial).
8. How does merge sort work? (Idea: Repeatedly merge two sublists into one larger list until done)
    a. What is the complexity of merge sort? (O(N log N): N comparisons to merge two lists of size N/2; log N merges needed)
9. How does quicksort work? (Idea: Repeatedly pick some element as pivot; move all keys less than pivot lower and all keys greater than pivot higher; pivot in correct final position. Repeat quicksort on keys to left and right of pivot)
    a. In the best case, what does the pivot do? (Divides the keys into two groups of equal size) Worst case? (Pivot is highest or lowest key and does no work other than finding location of the pivot)
    b. What is the complexity of quicksort in expected case? (O(N log N))

Finding Web Pages

1. Why are many machines used to be a web service like Google? (performance, service availability, data availability, price/performance, incremental scalability reliability – deliver correct results even when some nodes misbehave)
2. What must a search engine do before a user performs a search? (crawl, index, rank)
3. How does web crawler find pages? (follows graph of URL links from known pages)
    - How frequently should pages be crawled? (function of popularity of page and frequency of modifications)
4. What is contained in an index and how is it updated? (it is a mapping between search terms and the pages containing those terms)
    - How can one find pages containing multiple search terms? What is the complexity if indices are not sorted? ($O(N^2)$) What is the complexity if indices are sorted? ($O(N)$)
5. How can multiple pages with desired search terms be ranked?
    a. How are the goals of web page creators and users different?
    b. Why can't one use content on the page to do ranking?
    c. How can links be used to help with ranking?

Cryptography

1. What are the goals of information security? (confidentiality – no eavesdropping, integrity – no modifications, availability – can get message through, authenticity – people are who they say they are)
2. What is the idea of "shared secret cryptography"?
3. How can you crack a ceasar (shift) cipher? (Enumerate all 26 possibilities; easy to crack)
4. How can you crack a substitution cipher? (Use frequency analysis of popularity of different letters)
5. How is a one-time pad used in cryptography? (Different secret for every letter)

Catching Liars

1. How to solve logic puzzles involving liars and truth tellers? (Create truth table of all possible combinations of liar, truth teller; evaluate if speaker could make statement or not for that scenario)
2. Why might computers in a web service "lie"?
3. What is a fail-stop failure model? (works correctly or stops/crashes) How many nodes are needed to implement a reliable service if f nodes might fail? (f+1)
4. If one assumes fail-stop + consistent liars, how many nodes are needed if node nodes might fail? (2f+1; need f+1 good nodes to outvote the f liars)
5. What is a byzantine failure? (Can sometimes lie, can give response calculated to cause worst possible damage) Harder to handle than other failure models.

P vs. NP

1. Is it harder to solve a problem or check that a solution is correct?
    - What is a problem in the set of P? (Problems for which solutions exist in polynomial time)
    - What is a problem in the set of NP? (Problems where solutions can be checked in polynomial time. Not known if polynomial time solutions exist!)
2. What algorithm can you use to find if there exists a path between nodes of a graph? ($O(E)$ where E is the number of edges)
3. What greedy algorithm can you use to find a minimal spanning tree over a graph? (Idea: Repeatedly connect closest node to existing sub-graph)
4. How expensive are algorithms that must enumerate every possible combination? ($O(N!)$)
5. What is the travelling salesperson problem? (Just know the problem, not how to solve it!) Is there a known greedy solution? (No!)