Memoization:

- Use a n-dimensional array if need be, one dimension for each param + one for each result
- Alternatively, use a HashMap, and make custom Pair() that overrides equal.

Random Sequences:

- Simpulations, cryptography, games.
- Pseudorandom - hard to predict, usualy uniform distribution within a range
- Linear Congruental - seed X_0, then $X_i = (aX_{i-1} + c) \mod m$. period = m.
- Additive Generator - X_n random for $n < 55$, then $= X_{n-24} + X_{n-55} \mod 2^e$. Period = $2^f(2^{55} - 1)$
- Shuffling - swap shuffle from tail to start.
- Floyd selection alg - either put number in, or counter. (see lec32 pg 15)

Java:

- java.util.Random, Random() based on time, also exists Random(seed).
- next{int, double, long, float, boolean, gaussian}
- Collections.shuffle(list, random)

Graphs:

- Definition: A set of vertices and edges connecting those vertices.
- Uses: Makefiles, maps, any related data.
- Topological sort - order on number line, all "arrows" point one way.
- shortest path (Djikstras) - find shortest path to location. Expensive without heuristic filtering. (A*)
- Minimal spanning tree - telephone wire problem. Long path is okay, we just want minimal sum of edge lengths.
- union-find structures - two sub-graphs, combine with one edge.
- memory management as graph (loops, counters on pointers.)

Threads:

- Multiple threads - used for concurrent tasks
- Thread takes a Runnable and executes the run method, or executes thread.start()
- Used for GUI apps, networking, parallel computing, etc.

Memory Management:

- Scheduler alternates rapidly between each thread.
- sleep() triggers a context switch, if there are other candidates to run.
- notify() and notifyAll() - used in objects to let them know a lock has been released.
- Unpredictable - we can't control order of thread execution
- concurrency problems - read, pause, write. During pause value may change causing error.
- Solution to concurrency - locks. Only one thread accesses object at a time.
- Problem with above statement - dead lock issues. Each thread waiting for the other.
- message passing - solution v2. This allows threads to process tasks in their own time.

Traversals:

- preVisit, visit, postVisit. Use marked set and fringe of action items.
- BFS - 1.Enqueue the root node
  2.Dequeue a node and examine it.
  3. If the element sought is found in this node, quit the search and return a result.
  4. Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
  5. If the queue is empty, every node on the graph has been examined quit the search and return "not found".
  6. If the queue is not empty, repeat from Step 2.
- DFS - like BFS, but use stack instead of queue.

Random (from old finals):

- Binary trees have 2^{k-1} elements at a level k, where k is the level number. Total number of nodes is 2^N, where N is the number of levels. Visit N nodes with BFS, means you visit $2N$ with iterative deepening.
- Memoized functions have funky run times. Use multidimensional integer arrays for args and results, and be sure to include the fill time in the dimension, ie, memo[x][y][z] = $O(xyz)$ and memo[x][y] with $O(z)$ for fills is still $O(xyz)$

Trie:

- Long strings and chains, follow down till you see diff. Ie, bob and bo go b-o-"",-b

Sort Identification:

- Check for LSD/Radix - easy to spot digits in order
- Check what changed between each step, note indices and properties - know Shell's sort
- Try to spot pivots (column where same element occurs) - merge sort or quick sort

Run time analysis:

- Find most constly statement
- Be on the lookout for loops
- Know summation formulas: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- Know the progression of growth - order of times: $1 < log(n) < n < nlog(n) < n^2 < 2^n < O(n!)$

1. Relax.
2. You will do GREAT!
3. The "A" is yours.

CS61b Final Study Guide, by Ivan "Vania" Smirnov