Introduction

• Don’t learn all details at once; simply learn they exist

• **Algorithms in blue**
• **Techniques in green** – general principles for algorithm design
• **Challenges in red**

• In practice, most problems solved with language and library features

• When something hard, need more tools

• Items start easy, get harder
Sorting

- **Bubble sort** - $O(n^2)$ - still good for small sizes
- **Quicksort** - $O(n \log n)$ expected, $O(n^2)$ worst – randomized pivot – adversary possible – **Technique: divide and conquer**
- Comparison based sort provably $O(n \log n)$
  - So don’t compare. Index instead 😊
- **Radix sort** - $O(nk) \sim O(n)$, either direction, good when sparse, $k$ passes
- **Bucket sort** - $O \left( n + \frac{n^2}{k} + k \right) \sim O(n)$, MSD to LSD, good when dense
- Other sorts for different scales: disk, internet scale, cache aware…
- **Timsort** – state of art, uses known ordered pieces to direct future work.
Searching sorted array

- **Linear** – $O(n)$
- **Binary** – $O(\log n)$
  - divide and conquer
- **Fibonacci** – $O(\log n)$ - move Fibonacci sized ratios – can be better
- **Dictionary** – Can be $O(1)$ for well structured data, often $O(\log \log n)$
String searching

- Search for length $m$ word in length $n$ text over size $k$ alphabet.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocess</th>
<th>Matching</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>None</td>
<td>$O(nm)$</td>
<td>none</td>
</tr>
<tr>
<td>Knuth-Morris-Pratt  (KMP)</td>
<td>$O(m)$</td>
<td>$O(n)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>Boyer-Moore (BM)</td>
<td>$O(m + k)$</td>
<td>Best $O(n/m)$, worst $O(mn)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>Aho-Corasick (AC)</td>
<td></td>
<td>$O(n + m)$</td>
<td></td>
</tr>
</tbody>
</table>

- KMP: when a mismatch occurs, the word itself allows skipping ahead, bypassing re-examination of previously matched characters
- BM: **preprocesses** the word for speedy decisions, useful when word searched multiple times across multiple texts. **Technique: time-space tradeoff**
- AC: find all matches to a set of words
Sound matching

• **Soundex** – matches English words phonetically, patented 1918 and 1922
  • In SQL, many languages

• **Algorithm**
  • Keep first letter of word
  • Drop all other a, e, i, o, u, y, h, w
  • Except first, Replace with numbers:
    • b,f,p,v -> 1
    • c,g,j,k,q,s,x,z -> 2
    • d,t -> 3
    • L -> 4
    • m,n->5
    • r->6
  • Collapse certain duplicate numbers
  • Truncate to three numbers, or extend with zeroes

• Ex: Robert and Rupert -> R163, Rubin ->R150
String similarity

• Is the string “kitten” closer to “sitting” or to “potato”?

• **Levenshtein distance** : # of changes, insert, deletes between strings.
  • Also called **edit distance**.

• Can do in matrix, once cell at a time
  • Cell[i,j] = min {up + 1, left + 1, diag + χ(i,j)}
    where χ(i,j) is str1[i] != str2[j]

• Can reduce to 2 rows of memory

• **Technique: dynamic programming**
  • A form of **memorizing**
  • Very powerful way to design solutions.

• **Challenge**: time Fibonacci numbers with and without memorizing.
Backtracking search

• Want to search some finite space, discrete moves

• **Dancing Links** (Knuth popularized, 1979)
  • Items represented as grid of doubly linked nodes
  • DoMove/UndoMove each two pointer changes
  • Sudoku solver, combinatorial solvers
  • Heuristic S – choose branch with fewest subproblems
    ~exponential gain on depth, linear in cost
    • Technique: trim choices heuristic

• Ex

```cpp
void Solve(State state, int depth) {
  if (Solved(state)) {
    Log(state, depth);
    return;
  }
  auto moves = GenerateMoves(state);
  for (auto m : moves) {
    DoMove(state, m);
    Solve(state, depth + 1);
    UndoMove(state, m);
  }
}
```
Randomness

• Understand your use cases
• C/C++ rand() generally not very good or fast
• **Mersenne Twister** outdated and has bad properties
• **PCG** series for non-crypto pretty good, small, many tunable sizes, very fast

• Be careful:
  • \( \text{rand()} \% n \) is not uniform
  • \((\text{int})(\text{rand01()} \times N)\) is not uniform
  • To pick \([0,N-1]\) uniformly, take rand and randMax, sample till in multiple of range, then mod
  • Assumes \text{rand}() uniform over randMax, not always true!

• **Challenge**: implement rand3 from rand2.
• **Challenge**: implement a fair coin toss from a biased coin.
• **Challenge**: implement uniform rand 1 to N.
Shuffle and sample

• Common error: shuffle all 1 to N. Not uniform
  for (int i = 0; i < max; ++i)
    int j = rand() % max;
    swap(item[i], item[j]);

• Fischer-Yates shuffle
  for (int i = 0; i < n-1; ++i)
    int j = randN(i, n); // i ≤ j < n
    swap(item[i], item[j]);

• Psychological issues
  • Shuffle bag, Fibonacci shuffle (not sure of name)
  • Sphere sampling, area sampling (blue noise)
Sampling

• Problem: stream of data, want uniform random sample size k, don’t know length, cannot store.

• **Reservoir sampling:**
  • Keep first k
  • For each following sample, random to see if it should be in reservoir
    • \( J = \text{rand}(1,i) \) inclusive on seeing ith sample.
    • If \( j \leq k \) then \( R[j] = S[i] \)

• **Challenge**: can do with exponentially fewer rand calls.
  • anecdote of solving something yourself
Randomized algorithms

- Use *randomness* as a resource in an algorithm. Sometimes gives speedups.
- There are efficient randomized algorithms for which no efficient non-randomized algorithm is known.
- Unknown whether P = BPP: unknown is an arbitrary randomized algorithm that runs in polynomial time with a small error probability can be derandomized to run in polynomial time without using randomness.

- **Las Vegas algorithms** – correct answer, finite time, like QuickSort with random pivots.
- **Monte Carlo algorithms** – possible incorrect answer, possible infinite time
  - Run till probability of incorrect < probability of outrageous other errors: RAM cosmic rays, hardware failures, etc.
  - `IsPrime(n)` is currently deterministically $O((\log n)^{7.5})$, but k rounds of random Miller-Rabin is $O(k (\log n)^3)$.
- Example: two places compute massively large result, want to prove to other. Can send log n bits to prove exponentially small error.
- Examples:
  - In ECC, most random codes good
  - Crypto – random keys, nonces needed
  - Prime testing for RSA done probabilistically
  - QuickSort needs good random pivots
  - Network traffic backoff, related collision avoidance, retries
  - Evolutionary algorithms like Neural Networks, Genetic Algorithm, Simulated Annealing, Stochastic Gradient Descent (SGD).
- **Technique**: randomness for performance or adversary thwarting.
Sequences

- Longest Common Subsequence
- Longest Increasing Subsequence
  - Optimal substructure and overlapping subproblems implies try dynamic programming (DP)
- Max Subarray Sum
  - Naïve $O(n^3)$
  - Kadane algorithm $O(n)$
  - 2D version is $O(n^3)$
- Summed range trick
- Summed area table (also called Integral tables)
- Challenge (DP): # ways to change $10.00$ USD?
Problem: Find kth largest in unordered array

- Soln 1: sort, index, $O(n \log n)$, space $O(1)$ in place, $O(n)$ otherwise
- Soln 2: heap, keep top k, walk all items, $O(n \log k)$, space $O(k)$
- Soln 3: **QuickSelect**: like QuickSort, but stop early, $O(n)$ avg, $O(n^2)$ worst
- Soln 4: **Median of Medians**: $O(n)$ always, worst space $O(\log n)$
  - Slightly tricky, but implementable
  - Time space tradeoff
  - When used in QuickSort pivot, worst case sort reduces $O(n^2) \rightarrow O(n \log n)$
Heaps

• Also called priority queues
• Tree: parents $\geq$ children (max-heap)
  • Binary tree – simple code, bad merge
    • Can be stored efficiently in array
• Binomial heap – quick heap merges
• Pairing heap – simple to code
• Fibonacci heaps (complicated to code)
• Brodal queue (not good in practice)
• Many, many more
Skip list

• $O(\log n)$ insert, search on n ordered elements
  • Thus gives best array search speed. But exponentially faster inserts
  • Random choice of levels where to put new elements

• Easy to code, good performance
  • Tricks to make cache friendly

• Not good against adversary, can be made so

• Challenge: implement one
Disjoint Set

- Want to track used items from fixed set of items
  - Memory pages
  - FAT allocation
  - Allocation Bitmap nice method, uses 0/1
- More states than 0/1?
- **Disjoint-set** data structure (1964)
  - Tracks set of elements partitioned into disjoint sets
  - Near constant time add new set, merge sets, determine if elements in same set
  - Each item a node with parent pointer, can be array, also rank of item (depth)
  - Find path compression – walk pointers up over time
    - Technique: opportunistic data structure updating
  - Union merges smaller tree to root of larger tree
Hashing

• Hash table: array of values indexed by hashing keys.
• Hash collision: many ways to resolve
  • Linear probing – look for next free. Delete requires moving items.
• Cuckoo hashing: 2 hash functions, check both places first, easier to delete.
  • Collisions push items from one value to the other until stops.
    Based on cuckoo pushing eggs around.
    Need load factor at most 50%
• Perfect hashing: given fixed N items, can construct a hash that fits into N slots.
• Hopscotch hashing: Each bucket in a neighborhood (think cache line). If neighborhood full, resize table. Well suited for concurrent hashing.
Bloom Filters

• Efficient to **test if element is not member of a set**.
• Set array of $m$ bits to 0. $k$ different hash functions, mapping each element to one of the $m$ bits.
• False negatives not possible, false positives possible
• Vastly shrinks size of hash table
• Add: compute $k$ hashes, set the bits
• Query: check if all $k$ hashes bits set
• Cannot remove items (easily)
• $m$ and $k$ allow tuning the performance, error prob $\left(1 - e^{-kn/m}\right)^k$
• Roughly, 10 bits per element gives < 1% false positive rate, no matter how many elements
Pathfinding

• Find optimal path through discrete space
• Modeled as graph search
  • **A* search** (pronounced “A star”) very popular
    • Have list of paths so far
    • Use *heuristic* to find which to extend
    • Repeat

• Lots of tricks and rules to find best heuristics
• Very common in games, simple robotics, industrial optimization
Graph

• Shortest path problem: find shortest path(s) in a graph $G = (E, V)$
• **Dijkstra’s algorithm** - $O(V^2)$ – single path, non-negative edge weights
  • Trajan with Fibonacci heap 1984 $O(E + V \log V)$
  • Mark all as unvisited, each dist infinity
  • For each current in unvisited, set neighbor dists
  • Mark current as visited, add unvisited
  • Repeat till all visited
• **Bellman-Ford algorithm** if edges negative $O(VE)$, space $O(E)$
• **Floyd-Warshall algorithm** – all pairs of paths $O(V^3)$
• Lots and lots of graph algorithms
Graph

• Given graph with capacities on edges, what is max that can be pushed from source to sink?
  • Origin: 1954 analysis of Soviet railway traffic, now in all network models
  • Called the Max Flow problem
  • Ford-Fulkerson - $O(E \text{ max } f)$ weights $f$ rational

• Max-flow min-cut theorem: max flow source to sink is same as the min-cut: cut minimal costs edges to disconnect source from sink
  • Useful to understand network reliability
Metrics

• Metric is abstract notion of distance (think Euclidean 2D or 3D space)
  • $d(x, y) \geq 0$
  • $d(x, y) = 0 \iff x = y$
  • $d(x, y) = d(y, x)$
  • $d(x, z) \leq d(x, y) + d(y, z)$ Triangle inequality

• Technique: leverage triangle inequality
  • speed up all sorts of complex searching
  • Ex: explain color art speed up, photomosaic speedup

• Ex: Hamming distance: number of bits where strings differ
• Sqrt trick: long side + ½ short side accurate to within ~11.8%, massively faster
  • Often compare distance squared instead of sqrt
  • Many, many more speed tricks
Closest points

• Problem: Given $n$ points in metric space, find closest pair of points.
• Naïve $O(n^2)$
• $O(n \log n)$ in general
  • 2D: sort on x, split in half, recurse on left and right to get 2 values, check near middle carefully

• If floor function can be computed in constant time, $O(n \log \log n)$
• If randomization + floor function, $O(n)$
  • Technique: randomized algorithms
Partitioning

- For looking up things and neighbor questions in metric space
- 2D **Quadtree** – each node has 4 children
- 3D **Octree** – each node has 8 children
- **K-d tree** – each leaf is n dim point, creates splitting plane

- Most suffer from “curse of dimensionality”
  - High dimensional stuff “mostly on edges”, not “middles....”
  - Ex: n sphere mostly on edge, same as hypercube

- Many, many more (**Ball tree**, hybrid stuff....)
Clustering

• Want to group items by similarity under some metric

• **K-means clustering**
  • Iterative refinement (**Lloyd’s Algorithm**)  
  • Pick \( k \) initial items  
  • Assign each to nearest of \( k \)  
  • Compute new centroids  
  • Repeat, stop when assignments no longer move

• Optimal solution is NP-hard
Concurrent Data Structures

• Locks: deadlock, livelock, priority inversion, problems with interrupts
  • Coarse grained locking loses parallelization opportunities, fine grained adds more locking overhead, hard to get correct.
• Non-blocking: a stopped thread cannot block other progress
• Wait-free (strongest): guaranteed systemwide progress, starvation free
• Lock-free (middle): guaranteed systemwide progress, individual threads can starve
• Obstruction-free (weakest): single thread guaranteed to make progress if all other obstructing threads suspended.
• Implemented with read-modify-write primitive. CAS most common.
• Examples:
  • wait-free queue (2011), SRSW ring-buffer with only a memory barrier, linked lists, stack, …
Immutable data structures

- Needed as things become more parallel
- Each update returns new view of item
  - old views still valid

- Relaxed Radix Balanced Trees (RRB Tree)
  - Efficient immutable vector
  - Random access, update, push back, slice right/left \( \sim O(1) \)
  - Concat, insert, push front \( O(\log n) \)
- In many more modern languages (Clojure, Scala, more)

- Immer library github C++: Persistent immutable data structures
- Nicely performant

- Many, many more avenues, research, algorithms, data structures
  - Technique: immutable data structure for parallelization
Data Compression

- **Lossless**
  - Run Length Encoding (RLE): AAAAAAA => 7A
  - LZ77 – sliding window of history
  - LZ78 – explicit dictionary
  - Huffman – optimal for power of 2 probabilities
  - Arithmetic – optimal for fixed probabilities, is “limit” of Huffman
  - PPM – prediction by partial matching
  - BWT – in bzip, novel invertible “sorting” transform
  - Context-adaptive binary arithmetic coding (CABAC) – basis of most modern

- **Lossy**
  - Exploit perceptual bias
  - Exploit temporal changes
  - JPEG, MP3, H.265, Speex

  - **Compressive Sensing** (Tao, Candes, .., 2004)
    - Massive breakthrough: proved cases where can reconstruct signal with exponentially fewer samples than Nyquist limit
    - Design of radiating systems, radar through wall imaging, antenna design, MRI

  - **Challenge**: Implement a few of the first lossy versions
Boolean

- **SAT** – Boolean Satisfiability
  - Given a Boolean expression in $k$ variables, does there exist a True/False assignment making the expression true?
  - Fundamental, many real world problems can be cast into it.
  - Essential part of EDA toolbox. Even Excel has decent ones built in
  - Solvers called **SAT solvers**
    - SAT solvers useful for many problems, despite NP hard
    - Can often handle millions of variables in millions of clauses
    - Knuth TAOCP chapter especially nice presentation
    - Annual competition in the 2010s drove state of the art much higher

- Boolean minimization
  - **Karnaugh Map**, implemented as Quine-McClusky algorithm
  - Faster evaluation
  - Smaller circuits
Cache aware

- Being cache careful can give 10-100x performance speedups
- Huge cache variety across machines
- Naïve: compile per platform, carefully tune, costly and tricky, maintenance
- Problem: Make cache aware code portable

- Cache aware algorithms
  - also called cache oblivious algorithms
  - Technique: break problem recursively into $2^n$ sizes
  - One eventually fits in cache
- Many exist: funnel sort, quicksort, matrix transposition, array reversal, b-trees, priority queues
Control Theory

- Sources of information, noisy, want to merge
  - Sensor fusion (GPS, accelerometer, clock, compass, cameras,..)
  - Robots
  - Self-driving cars
  - Industrial machines
  - Phone

- **Kalman filter** – algorithm to do sensor fusion with uncertainty
  - Extended Kalman Filter (EKF)

- **PID controller**
  - proportional–integral–derivative controller
  - Drones
  - Engines
  - Industrial control

- **Challenge**: make simple PID that tracks some value, then poke at the value and see how the PID responds
The End

Questions?
Change Problem

• How many ways to make change for $10.00?
• Coins & bills = \{1,5,10,25,50,100,200,500,1000\} = 9 items, S0 to S8
• \(C(m, n)\) = number of ways to total \(n\) with items S0 through Sm
• Key insight: split computation to those using \(Sm\) and those not
  • Technique: split problem into mutually exclusive cases
• \(C(m, n) = C(m, n - Sm) + C(m - 1, n)\)
• End cases:
  • if \(n < 0\) or \(m < 0\) then \(C=0\)
  • if \(n == 0\) then \(C = 1\)
• Dynamic programming or memorization: \(C(8,1000) = 3,237,135\)
TODO

• Highlight general techniques
• Final slide with techniques for review
• On final slide, list techniques one side, algo names other
• Techniques: meet in middle (sqrt trick)