Matthew Tran
CS170
pg. 1

- List have cycle?
  - slow ptr, fast ptr.
  - if ever at same place, then cycle
  - Correctness:
    - both enter cycle at some time
    - distance decreases every step
  - $O(n)$ runtime, $O(1)$ space
- Asymptotic Review
  - ignore constants — ∞ also works ↓
  - ignore smaller order terms
  - Big O $f(n) = O(g(n))$
    - $f(n) \le c \cdot g(n)$ for big n
    - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
  - Big Θ (Theta) $f(n) = \Theta(g(n))$
    - $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
    - $k_1 \cdot g(n) \le f(n) \le k_2 \cdot g(n)$
    - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c,\ c > 0$
  - Big Ω (Omega) $f(n) = \Omega(g(n))$
    - $g(n) = O(f(n))$
    - $c \cdot g(n) \le f(n)$ for big n
    - $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
- Fast Multiplication (better than $O(n^2)$)
  - $x = 2^{n/2} x_H + x_L,\ y = 2^{n/2} y_H + y_L$
  - $\boxed{x_H \mid x_L}$ $\boxed{y_H \mid y_L}$
  - $x \cdot y = 2^n x_H y_H + 2^{n/2}(x_H y_L + x_L y_H) + x_L y_L$
    - Need 3 terms
    - $P_1 = (x_H + x_L)(y_H + y_L) = x_H y_H + x_H y_L + x_L y_H + x_L y_L$
    - $P_2 = x_H y_H,\ P_3 = x_L y_L,\ (x_H y_L + x_L y_H) = P_1 - P_2 - P_3$
  - $T(n) = 3T(\frac{n}{2}) + \Theta(n)$
  - $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.58})$
- $a^{\log_b n} = n^{\log_b a}$; $b = a^{\log_a b}$
- Master's Theorem $T(n) = aT(\frac{n}{b}) + O(n^d)$
  - $T(n) = \begin{cases} O(n^d) & d > \log_b a \\ O(n^{\log_b a}) & d < \log_b a \\ O(n^d \log_b n) & d = \log_b a \end{cases}$
- Fast Matrix Mult (better than $O(n^3)$)
  - $\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$
  - $P_1 = A(F-H)$ $\quad P_5 = (A+D)(E+H)$
    $P_2 = (A+B)H$ $\quad P_6 = (B-D)(G+H)$
    $P_3 = (C+D)E$ $\quad P_7 = (A-C)(E+F)$
    $P_4 = D(G-E)$
  - $\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3+P_7 \end{bmatrix}$
  - $T(n) = 7T(\frac{n}{2}) + O(n^2)$
  - $O(n^{\log_2 7}) \approx O(n^{2.81})$ // best so far $O(n^{2.36})$
- Mergesort $O(n \log n)$
- Comparison Sort Lower Bound $\Omega(n \log n) = \Omega(\log n!)$
- Median Finding (k^{th} smallest element) ($\lfloor \frac{n}{2} \rfloor + 1$ elt)
  - Select(k, S)
    - if $k=1$ and $|S|=1$, return $S[0]$
    - rand pivot elt. v from S
    - S_L = elt < v ; S_v = elt = v ; S_R = elt > v
    - if $k \le |S_L|$, Select(k, S_L)
    - elif $k \le |S_L| + |S_v|$, return v
    - else Select($k - |S_L| - |S_v|$, S_R)
  - worst case $\Theta(n^2)$
  - average case $\Theta(n)$
    - $T(n) \le T(3/4\, n) + O(n)$
- Select Pivot
  - groups of size 5, S = medians of each group, return median of S
  - $x \ge$ and $\le \frac{3}{10} n$ elements, ≥ 3 elt. of $\frac{1}{2}$ of $\frac{n}{5}$ groups
  - $T(n) \le T(\frac{n}{5}) + T(\frac{7}{10} n) + O(n)$
    - $O(n)$ - computing medians and partitioning
    - $T(\frac{n}{5})$ for median of S
    - $T(\frac{7}{10} n)$ for recursive call
  - $O(n)$ problem decreases geometrically $\frac{1}{5} + \frac{7}{10} = \frac{9}{10}$

- Fast Fourier Transform
  - useful for counting, like 3SUM like probs
  - $p(x) = p_0 + p_1 x + \dots + p_d x^d$
    $= \sum_{i=0}^{d} p_i x^i = \langle p_0, p_1, \dots, p_d \rangle$
  - assume math take $O(1)$ time
  - Horner's method $p(x) = p_0 + x(p_1 + x(\dots))$
  - addition (add coeffs) - $\Theta(d)$ time
  - $p(x) \cdot q(x)$ $\Theta(d^2)$ naive in coeff form
    - $\Theta(n)$ for n points
  - $p(x) = P_{even}(x^2) + x P_{odd}(x^2)$
    - $P_{even}(\alpha) = p_0 + p_2 \alpha + p_4 \alpha^2 + \dots$
    - $P_{odd}(\alpha) = p_1 + p_3 \alpha + p_5 \alpha^2 + \dots$
  - idea: evaluate at n^{th} roots of unity
    - $1 < \dots$ squaring n^{th} root gives $n/2^{th}$ root $e^{j\frac{2\pi i}{N}}$
  - $\Theta(n \log n)$ ↑ gives us $p(x)$ at n pts. fast
- DFT
  - $M_n(\omega) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}$
  - $M_n(\omega) = \frac{1}{n} M_n(\omega^{-1})$
  - easy to invert $M_n(\omega)\begin{bmatrix} p_0 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} p(\omega_0) \\ \vdots \\ p(\omega_{n-1}) \end{bmatrix}$
  - $M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega)^*$
  - gives us $\Theta(n \log n)$ polynomial multiplication
- Graphs!
  - 4 color theorem, useful for scheduling
  - matrix representation | adjacency list

edge(u,v)?	$O(1)$	$O(\lvert V \rvert)$
neighbors of u?	$O(\lvert V \rvert)$	$O(d)$
Space	$O(\lvert V \rvert^2)$	$O(\lvert E \rvert)$

  - Find all nodes reachable from v
    - Explore(v):
      ```
      visited[v] = True
      for each edge (v,w) in E:
          if not visited[w]:
              explore(w)
      ```
    - Proof by contradiction/induction
    - $O(|V| + |E|)$
  - DFS Connected Components (Undirected)
    - DFS(G):
      ```
      ccnum = 0
      for each v in V:
          if not visited[v]:
              explore(v)
              ccnum += 1
      ```
    - explore(v):
      ```
      visited[v] = true
      previsit(v)
      for each edge (v,w) in E:
          if not visited[w]:
              explore(w)
      postvisit(v)
      ```
    - previsit(v):
      ```
      cc[v] = ccnumber
      ```
  - Using pre/post numbers
    - previsit: pre[v] = clock; clock += 1
    - postvisit: post[v] = clock; clock += 1
    - intervals either contained or disjoint
  - edge(u,v) ↳
    - Tree edge - interval v ∈ interval of u; u on stack before v
    - Back edge - int(u) ∈ int(v) (cycle!) u after v, but v ≠ u
  - $O(|V| + |E|)$

- DFS Directed $O(|V| + |E|)$
  - for edge(u,v)
    - Tree/Forward - int(v) in int(u)
    - Forward - " (just not in tree)
    - Back - int(u) in int(v)
      - edge to ancestor
    - Cross - int(v) before int(u)
      - v explored before u visited
  - DAG (Directed Acyclic Graph)
    - cycle if back edge $O(|V| + |E|)$
    - Topological Sort / Linearize
    - output in reverse post order number
    - Source - no in edge, highest post #
    - Sink - no out edge, lowest # post
  - SCC (Strongly Connected Component)
    - path any node to any other node
    - any directed graph = DAG of SCCs
      - bc any cycle collapses into SCC
    - algorithm
      1. explore on sink component
      2. output visited nodes
      3. repeat
    - highest post order is in source
    - property: C and C' with edge (C → C') then highest post # in C > any in C'
      - bc C explored first
    - reverse edges, source becomes sink
    - $O(|V| + |E|)$
- Shortest Paths
  - BFS (Breadth First Search)
    - layer by layer, use Queue
    - $O(|V| + |E|)$
    - don't ~~work~~ do edge lengths
  - Dijkstra's Algorithm
    - for each(v): $d(v) = \infty$
    - $d(s) = 0$
    - Q.insert(s, 0) # Priority Queue
    - while u = Q.deleteMin():
      - for each edge (u,v):
        if $d(v) > d(u) + \ell(u,v)$:
        d(v) = "
        Q.insertOrDecrease(v, d(v))
    - $O((|V| + |E|) \log |V|)$
      - binary heap $O(\log n)$
      - fibonacci: $O(1)$ decrease, $O(\log |V|)$ delete
        - $O(|V| \log |V| + |E|)$
  - Negative Edges
  - Bellman-Ford — optimization - break if no updates
    - update all edges $|V| - 1$ times
    - $O(|V||E|)$ or if complete $O(|V|^3)$
    - Do $|V|$ times to find negative cycle
  - DAG Shortest Path
    - linearize
    - process in topological order
    - $O(|V| + |E|)$
- Tips
  - log good for optimize product
  - Use BellmanFord - good for neg edges
    - good for ~~neg~~ cycle detection
  - Proof by contradiction
  - cycle property - heaviest edge in any cycle not in MST
  - $1 + \dots + n = \frac{n(n+1)}{2}$
  - L'Hospital's
  - divide conquer w/ boundaries - keep left and right pointers across middle boundary, and keep track of stuff
  - $e^{j\theta} = \cos\theta + j\sin\theta$
  - geometric series $\frac{1}{1-\alpha}$, $\frac{1-\alpha^{n+1}}{1-\alpha}$

- Greedy Algorithm
  - do whatever action gives most immediate benefit
  - arguments tend to be: if have alternate away, can swap to make better
  - Minimum Spanning Tree (MST)
    - Tree
      - n-1 edges, no cycles, connected
    - want cheapest subgraph
    - Cut Property
      - smallest edge across any cut is in MST (same)
    - Kruskal's
      - sort edges
      - add n-1 edges starting from smallest that don't make cycle
      - use Disjoint Sets
      - $O(nm)$ # $O(|V||E|)$ → $O(|E| \log |V|)$
    - Prim's
      - Dijkstra's but keep track of distance to MST
      - only look at edge weight
      - $O((m+n) \log n)$
        == $O(|E| \log |V|)$
    - Disjoint Sets
      - $\pi(x)$ for each x
      - pointers to a root
      - Union by rank
        - keep size of root "rank"
        - merge into bigger rank
        - $O(\log(n))$ find
        - Path compression $\log^* n$ ↑ logs until = 1
  - Compression
    - Prefix Free Codes
      - no code prefix of another
      - no confusion
      - Tree Rep, every code is a leaf
      - expected length for N chars : $\lceil \log_2 N \rceil$
        - but encode w/ freq ∑ freq·length
          if < 2, good
        - another view of cost
          - small nodes, each node is sum of child freq
    - Huffman Coding
      - merge smallest ones
      - keep going
      - correctness - if depth diff, can swap, it'll be better
  - Horn Formulas (Horn SAT)
    - boolean variables either True or False
    - literal - var x or $\bar{x}$
    - implication $f(x) \Rightarrow y$
    - singleton - degenerate implication $\Rightarrow x$ where x is True
    - pure negative clause - ∧ of neg literals
    - min True? to satisfy all
    - alg
      - all False
      - add True to RHS until all True
      - if not, then failed
  - Set Cover
    - sets, min such that $\cup S_i = B$
    - pick set that covers the most, then remove from rest and repeat
    - not always optimal
    - if n items, optimal k, then $\le k \ln n$

- Cartesian Sum - polynomials
- Remove leaf in DFS leaves G connected
- DFS - redraw w/ only edges used
- $T(n) = T(n - n^{1/3}) + 1$ == $O(n^{2/3})$

Matthew Tran
CS170
pg. 2

- Dynamic Programming
  - General Approach
    - recursive definition of subproblem
    - some base cases
    - store results of subproblems
    - run in reverse order, small to big
    - like recursion & memoization but not
  - Popular Subproblems
    - String - prefix, suffix, n of them
    - Triangulation - $i \to j$, n^2
    - Trees - rooted subtree
    - TSP - all possible subsets
  - Ex.
    - Longest Path in DAG
      - input: topo sorted DAG $1 \dots n$
      - output: length of longest path
      - subprob: $L(i)$ longest path ending at i
      - recurrence relation:
        - $L(i) = \max L(k) + 1$ for (k,i) edge
      - solve from $n \dots 1$ $\Rightarrow O(|V|+|E|)$
    - Knapsack
      - inputs: set of items (weight, value) - $(w_1, v_1) \dots (w_n, v_n)$
      - outputs: items weight $< W$, max value
      - if $\exists$ partition $\Rightarrow$ Longest Path in DAG
      - subprob: $K(w, i)$ - max val, $\le w$, $\{1 \dots i\}$ items
      - recurrence relation
        - $K(w,i) = \begin{cases} \max(K(w-w_i, i-1) + v_i, \; K(w, i-1)) \\ \text{not in } (i) \quad K(w, i-1) \end{cases}$
      - Base Case: $K(0, i) = 0$ $\quad O(wn)$
      - solve $i = 1 \dots n$, $w = 1 \dots W$; return $K(W, n)$
    - Edit Distance $(1 \dots i) \Rightarrow (1 \dots j)$
      - min of all possibilities, substring $\quad O(ij)$
    - Strategy
      - max of all possible options, base case ϵ win/lose
    - Shortest Path all Points $v_1, \dots v_n$ Floyd Warshall
      - subprob: $D(i,j,k)$ - shortest path i to j using $\{1 \dots k\}$
        - $D(i,j,0) = w_{ij}$ if exists, ∞ else
        - $D(i,j,n)$ = length shortest path
      - $D(i,j,k) = \min\{D(i,j,k-1),\; D(i,k,k-1) + D(k,j,k-1)\}$
      - $k = 0 \dots n$, $i = 1 \dots n$, $j = 1 \dots n$ $\quad O(n^3)$
- Linear Programming
  - Variables $x_1, \dots x_n$
  - max/min a linear function, subject to linear constraints
  - Properties
    - Feasible region always convex
    - optimal occurs at one of corners of region
    - vertex is intersection of constraints $\binom{m}{n}$
  - Tricks
    - max $\Leftrightarrow$ min - mult coeff by -1
    - $ax_i \dots \le b \Rightarrow ax_i + s = b$, $s \ge$ "slack variable"
    - $ax = b \Rightarrow ax \le b$ and $ax \ge b$
    - $x \Rightarrow x = x^+ - x^-$, $x^+ \ge 0, x^- \ge 0$
    - $\min |e| \Rightarrow e = e^+ - e^-$, $\min e^+ + e^-$, $e^+ \ge 0, e^- \ge 0$
    - $\min \max |e| \Rightarrow m = \max |e|$, $\min m$, $m \ge 0$
      - note $|x| \le c \Rightarrow x \le c$ and $-x \le c$
  - Standard Form
    - $\min cx$
    - $Ax \ge b$ ← book says $Ax = b$ and add more vars
    - $x \ge 0$
  - Primal - $Ax \le b$, $\max cx$, $x \ge 0$ ← google says this is standard form
  - Dual - $A^T y \ge b$, $\min by$, $x \ge 0$; or $\min y^T b$, $y^T A \ge C$, $y \ge 0$
    - multiply each eqn by y_i and sum, $y_i \ge 0$
    - don't have to use all eqns. when finding dual, just need enough to have same form as objective
    - forms an upper bound
  - Weak Duality - Primal(P) $\le$ Dual(D)
  - Strong Duality - if LP bounded, then dual bounded and same value $P = D$
  - Complementary Slackness - given A, b, c and feasible x, y
    - optimal iff $x_i(c_i - (y^T A)_i) = 0$ and $y_i(b_i - (Ax)_i) = 0 \Rightarrow cx = by$
  - Integer LP - strong duality doesn't necessarily apply
  - Degeneracy - intersection $>n$ constraints ⋇ $\Rightarrow$ infinite loop, perturb problem a bit to fix
  - Unboundness - optimal or unbounded improvement, simplex can find difference
  - Simplex Algorithm
    - m constraint, n variables $\quad$ $m \times n$ matrix A $\quad$ - runtime $O(nm)$ to update once, could be exponential, fast in practice
    - Canonical Form - $\max c^T x$, $Ax \le b$, $x \ge 0$
    - Start at origin (if feasible)
    - Testing optimality $\Rightarrow$ all $c_i \le 0$ (increase will be suboptimal), if any $c_i > 0$, not optimal ← Δ x_i will $\uparrow c_i$
    - not optimal $\Rightarrow$ move to next vertex $\Rightarrow$ change coordinates $\Rightarrow$ pick x_i for $c_i > 0$ ← make sure to pick tightest bound ↑ check other constraints ← keep going until optimal
      - $\Rightarrow$ all same var $(y_i = x_j)$, but x_i, pick constraint with x_i and increase until tight $y_i = b_i - a_i x$

- Greedy Algorithm pt. 2
  - get best thing at each step
  - Exchange argument
- Maximum Flow
  - directed graph G, source s, sink t, capacities $c_e > 0$
  - find flow f_e for $e \in E$
    1. $0 \le f_e \le c_e$
    2. $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,w) \in E} f_{uw}$ for u not s or t
       - in = out except for source and target
    3. $\max \sum_{(s,u) \in E} f_{su}$ max flow out of source (or into target)
  - given integer c_e, there is integer solution.
  - Ford-Fulkerson - $O(mF)$ where F is flow
    - start all $f_e = 0$
    - find s-t path w/ > 0 capacity
    - add to flow along path and reduce flow on reverse edge
      - ↓ capacity on edge, ↑ on reverse edge
    - continue until no s-t path
  - Residual Graph $\quad$ for given f
    - same vertices
    - forward edges $e \in E$, w/ capacity c_e (or $c_e - f_e$)
    - reverse edges of e w/ capacity 0 (or f_e)
    - so sum of forward/reverse are c_e
  - Optimality
    - S-T cut
      - partition V into S and T where $s \in S$ and $t \in T$
      - sum of edges $S \to T$ give upper bound on flow
      - max flow min cut theorem
        - max st flow = min cut
  - Edmonds Karp
    - implementation of Ford-Fulkerson but uses shortest path (BFS) w/ available and more than flow
    - shortest path increases monotonically $(\ge)$
    - $O(|V||E|^2)$
    - augmenting path - path used in each step of Ford Fulkerson
  - Bipartite Matching
    - max size one to one matching of graph
    - reduce max flow but not always integer solution
    - use augmented alternating paths
      - use edge, then not, then do, etc
      - repeat until unmatched node
    - use directed graph, $U \to V$ if in matching, $V \to U$ if not
      - find path between unmatched nodes on left to right
      - until everything matched, or output a cut.
  - Zero Sum Games
    - gain of P1 = loss of P2, n gain + loss = 0
    - Nash Equilibrium - no incentive to change strategy
    - $m \times n$ payoff matrix A, row strat $x = (x_1, \dots x_m)$ $y = (y_1, \dots y_n)$ $\sum = 1$
    - payoff for (x, y) - $p(x,y) = x^T A y$, row max, col min
      - equilibrium pair (x^*, y^*) $\quad p(x^*, y^*) = \min_y (x^*)^T A y = \max_x x^T A y^*$
      - max: $A^{(i)} y^* = (x^*)^T A y^* = \min_j (A^T)^{(j)} x^*$
    - $R = \min_y \max_x (x^T A y)$ $\quad$ dual $\quad C = \max_x \min_y (x^T A y)$
      - strong duality $R = C$
    - $R(y) = \max_x x^T A y$, $C(x) = \min_y x^T A y$
    - Find (x^*, y^*)? MW alg!
      - approximate equilibrium $R(y) - C(x) \le \epsilon$
      - assume A payoffs $(0,1)$, $T = \frac{\log n}{\epsilon^2}$ days do:
        - 1) m pure column and m experts, y_t is col strat on day t use MW to make list
        - 2) Each day, play best row response to y_t (row A that max col expected loss) x_t is "indicator" vector for the row
      - $R(y^*) - C(x^*) \le 2\epsilon$
      - $O(nm \frac{\log n}{\epsilon^2})$ basically linear
      - LP $O(n^3 m)$ (Fast LP $O(\sqrt{n+m})$ know algorithms)
  - Minimax Theorem
    - $\min_{y \in \Delta} \max_{x \in \Delta} x^T A y - \max_{x \in \Delta} \min_{y \in \Delta} x^T A y = 0$
  - LP formulation, matrix A rows A_i, cols $A^{(i)}$

$C = \max z$	$R = \min z$
$\forall i \; a^{(i)} \cdot x \ge z$	$\forall i \; a_i y \le z$
$\sum_i x_i = 1$	$\sum_i y_i = 1$
$x_i \ge 0$	$y_i \ge 0$

- Multiplicative Weights
  - n experts, loses diff amount each day, best choice? ← know everyone's predictions
  - Perfect Expert
    - minimize regret
      - regret $\equiv$ loss/gain $-$ best loss/gain
    - Algorithm 1 - pick 1
      - mistake bound: $n-1$
        - lowerbound: adversary, worst case
        - upperbound: every mistake $\Rightarrow$ lose 1 expert
    - Algorithm 2 - majority of all "perfect"
      - mistake bound - $\log n$
        - each mistake at least halves "perfect" experts
  - Imperfect Experts
    - Algorithm 1 - weighted majority
      1. all $w_i = 1$
      2. predict w/ weighted majority of experts
      3. $w_i = (1-\epsilon) w_i$ if wrong
      - Analysis
        - potential function $\sum w_i$ (initially n)
        - best expert makes $\le m$ mistakes; M = mistakes alg makes
        - $\sum w_i$ mult by $(1 - \frac{\epsilon}{2})$ w/ each mistake
        - $(1-\epsilon)^m \le \sum w_i \le (1 - \frac{\epsilon}{2})^M n$
        - $\Rightarrow m \ln(1-\epsilon) \le M(\ln(1 - \epsilon/2)) + \ln n$
        - $\Rightarrow (-\epsilon - \epsilon^2) m \le M (-\epsilon/2) + \ln n$
        - $\Rightarrow -\epsilon(1+\epsilon) m \le M(-\epsilon/2) + \ln n$
        - $\Rightarrow 2(1+\epsilon) m \ge M \frac{-2 \ln n}{\epsilon}$
      - $M \le 2(1+\epsilon) m + \frac{2 \ln n}{\epsilon}$
        - as $m \to \infty$, at best $2x$ of expert $(1+\epsilon)$
    - Algorithm 2 - Randomized
      - each expert loses $\ell_i^t \in [0,1]$ on day t ← like 0 if wrong, 1 if right, similar to alg 1
      1. $w_i = 1$
      2. Choose i w/ prob w_i / W, $W = \sum_i w_i$
      3. $w_i = w_i (1-\epsilon)^{\ell_i^t}$
      - Analysis
        - $W(t) = \sum_i w_i$ at time t, $W(0) = n$ ← like m, but like confidence
        - best expert loses L^* total $(1-\epsilon)^{L^*}$
        - $L_t = \sum_i \frac{w_i}{W} \cdot \ell_i^t$ - expected loss in time t
        - for $\epsilon \le 1/2$, $W(t+1) \le W(t)(1 - \epsilon L_t)$ loss → weight loss
          - $W(t+1) = \sum_i (1-\epsilon)^{\ell_i^t} w_i \le \sum_i (1 - \epsilon \ell_i^t) w_i$ $\quad 0 \le \epsilon \le 1/2$
            - $\le \sum_i w_i - \epsilon \sum_i w_i \ell_i^t$
            - $\le W(t)(1 - \epsilon L_t)$
        - $(1-\epsilon)^{L^*} \le W(T) \le n \prod_t (1 - \epsilon L_t)$
          - $\Rightarrow L^* \ln(1-\epsilon) \le \ln n + \sum \ln(1 - \epsilon L_t)$
          - $-L^*(\epsilon + \epsilon^2) \le \ln n - \epsilon \sum L_t$
        - $\sum_t L_t \le (1+\epsilon) L^* + \frac{\ln n}{\epsilon}$ $\quad (1+\epsilon)$
          - $L^* \to \infty$, total expected loss = loss of best expert
          - no factor of 2!
  - All Alg: Gains instead of Loss
    - $g_i^t \in [0,1]$, gain on day t
    - MW w/ $(1+\epsilon)^{g_i^t}$
    - $G \ge (1-\epsilon) G^* - \frac{\log n}{\epsilon}$, G^* payoff of best expert
  - Scaling: loss not $[0,1]$ but $[0, \rho]$
    - $L \le (1+\epsilon) L^* + \frac{\rho \log n}{\epsilon}$
  - Formulas
    - Proof Idea: Taylor Expansion, $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots$
    - For $\epsilon \le 1$, $x \in [0,1]$
      - $(1+\epsilon)^x \le 1 + \epsilon x$
      - $(1-\epsilon)^x \le 1 - \epsilon x$
    - For $\epsilon \in [0, 1/2]$
      - $-\epsilon - \epsilon^2 \le \ln(1-\epsilon) \le -\epsilon$
      - $\epsilon - \epsilon^2 \le \ln(1+\epsilon) \le \epsilon$
  - Stock Framework (Book)
    - money divided into n stocks, $\ell_i^{(t)}$ is loss (or gain) on day t
    - loss on day $t = \sum_i x_i \ell_i^{(t)}$, over T days $\sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} \ell_i^{(t)}$
      - $x_i^{(t)}$ is percent spent on stock i, $\sum_{i=1}^{n} x_i^{(t)} = 1$, $x_i^{(t)} \ge 0$
    - $R_T = \sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} \ell_i^{(t)} - \min_x \sum_{t=1}^{T} \sum_{i=1}^{n} x_i \cdot \ell_i^{(t)}$ ← fixed
    - $R_T \le 2\sqrt{T \ln n}$
    - $R_T \le \epsilon T + \frac{\ln n}{\epsilon}$; Theorem: all losses $\in (0,1)$, $0 < \epsilon \in [0, 1/2]$, T steps
    - if $T \ge 4 \ln n$ and $\epsilon = \sqrt{\frac{\ln n}{T}} \Rightarrow R_T \le 2\sqrt{T \ln n}$
    - $\frac{R_T}{T} = O\left(\sqrt{\frac{\ln n}{T}}\right)$ goes to 0 w/ T

- Zero Sum stk

			B	
		i	w	f
	d	-10	3	3
A	s	4	-1	-3
	r	6	-9	2

max z (optimal A strat)

$-10d + 4s + 6r \geq z$

$3d - s - 9r \geq z$

$3d - 3s + 2r \geq z$

$d + s + r = 1$

$d, s, r \geq 0$

dual

min z (optimal B strat)

$-10i + 3w + 3f \leq z$

$4i - w - 3f \leq z$

$6i - 9w + 2f \leq z$

$i + w + f = 1$

$i, w, f \geq 0$

- MST not always same as Shortest Paths Tree
- some DP don't exist bc each subproblem depends on each other

 very useful for counterexamples

- Run Ford Fulkerson if not sure about min cut
- ↑ increases flow w/ each iteration
- upper bound FF (Ford Fulkerson) = max flow (not tight)
- tight upper bound FF - alternate path into smallest edge
- LP solution for dual upperbounds primal (or vice versa)
- Huffman Coding - bushy or spindly or both
  - most frequent min 1, max log n
  - least frequent min log n, max n-1
- MST - heaviest edge in cycle never included
- MST - wings has good counterexample
- Kruskal's good proof that Tree
- Horn - greedy, only set true if need to
- zero sum - if they're adapted, u can do better than the optimal
- "unique" usually false

- P vs. NP
  - P - can find solution in polynomial time
  - NP - verifying solution easy (polynomial time)
    - P ⊆ NP - just run alg, check if same
    - optimizations usually not in NP
      - decision/budget usually are
  - Reductions
    - $A \leq_p B \triangleq$ A reduces to B
      - can use alg for B to solve A
  - NP Complete
    - every problem in NP reduces to it
    - all NP Complete can reduce to each other
      - cycle; just reduce to one w/ above form to prove
  - Coping with NP-Completeness
    - Approximation Algorithms
      - greedy/stuff like that
      - usually k·OPT solutions upper bound
  - SATISFIABILITY (SAT) ∈ NP-Complete
    - Conjunctive Normal Form (CNF) - and of ors
    - each clause needs a literal that is true
    - # possibilities are exponential
  - Search Problem (~~[illegible]~~ budget = decision
    - solution checkable in polynomial time
  - Horn SAT ∈ P, linear
    - clause contains at most one positive literal
    - greedy algorithm, find min # true
  - 2SAT ∈ P
    - 2 literals per clause
    - turn into implication $A \vee B = (\bar{A} \Rightarrow B) \wedge (\bar{B} \Rightarrow A)$
    - find SCC, sinks are true
  - 3SAT ∈ NP Complete
  - Traveling Salesman Problem (TSP) ∈ NP-Hard
    - vertices + distances, budget b
    - want tour ≤ b, each vertex once
    - decision version (budget) ∈ NP-Complete
    - metric TSP (Δ inequality) ∈ NP-Complete
  - Hamiltonian/Rudrata Cycle ∈ NP-Complete
    - special case of TSP
    - Hamiltonian Path ∈ NP Complete
  - Minimum K-cut ∈ NP-complete if k there
    - cut separate into k connected components
    - $O(|V|^{k^2})$ alg
    - $2 - 2/k$ approximation; mincut
      - use min max flows, remove heaviest
  - Integer Linear Programming (ILP)
    - decision problem ∈ NP Complete
  - 3Dimensional Matching ∈ NP Hard, search complete
  - Independent Set ∈ NP Complete, search
    - no 2 vertex share edge
  - Vertex Cover ∈ NP Complete, search; NP hard
  - Set Cover ∈ NP Hard, search NP Complete
  - Longest Path ∈ NP hard, search NP Complete
  - Knapsack ∈ NP complete
  - Balanced Cut ∈ NP Complete
  - Bipartite Matching, unary knapsack, independent set in trees, LP, Euler path, min cut all ∈ P
- Reductions
  - Rudrata (s,t) Path ≤ Rudrata Cycle
    - add
  - 3SAT ≤ Independent set
    - each clause, each literal edge between negation; want # of clauses size set
  - SAT ≤ 3SAT
    - replace all >3 w/ set of clauses, w/ new var
  - independent set ≤ vertex cover
    - take min set in cover |V|-g
  - ZOE (Zero One IP) ≤ Rudrata Cycle
  - NP → SAT
    - use circuit SAT

- Backtracking
  - don't solve if wrong
  - go back up tree
- Branch and Bound
  - basically generate partial solutions
  - if is complete, update best so far
  - only add partial if cost < best so far; lower bound on total cost
  - need good heuristic/lower bound
- Approximation Algorithms
  - Vertex Cover
    - Set Cover greedy $O(\log n)$
    - use maximal matching ≤ 2OPT
  - Clustering ∈ NP-Hard
    - metrics, minimize diameter
    - pick pt in set
    - pick centers furthest away
    - 2OPT
  - TSP (metric)
    - note MST ≤ remove one edge from TSP ≤ TSP
    - use MST, but if repeat, bypass it
    - 2OPT
    - if TSP has poly time approx, then Rudrata poly, so no poly alg for general TSP approx
  - Knapsack
    - pick ε
    - remove precision (floor div)
    - $O(n^3/\varepsilon)$
    - OPT(1-ε) (less than worse)
  - Local Search Heuristics
    - replace w/ smaller in neighborhood
    - like try small changes, see if better
    - question boundary is neighborhood
  - TSP
    - try 2 change $O(n^2)$
    - more general 3 change $O(n^3)$
    - start random, update
  - Dealing with Local Optima
    - randomization and restart
    - Simulated annealing
      - try also w/ bigger cost w/ probability something like $e^{-\Delta/T}$, Δ = diff
      - concept of temperature T
        - start at large T, then "cool down" to 0
        - start by wandering, then settle in local
      - issue of how to change temp
- Streaming
  - huge stream, don't know if end,
  - memory limitation
  - one pass thru data
  - poly($\log n$) bits of memory
  - Probability Review
    - union bound: $P(A) + P(B) \leq P(A \cup B)$
    - if independent $P(A \wedge B) = P(A) \cdot P(B)$
    - expectation $E(X) = \sum P(X=v) \cdot v$
    - linearity of expectation $E(X+Y) = E(X) + E(Y)$
    - if indep $E(X \cdot Y) = E(X) \cdot E(Y)$
    - Markov's Inequality: if $X \geq 0$, all $t > 0$

      $$P(X \geq t) \leq \frac{E(X)}{t}$$
    - variance $\sigma^2 = E((X - E(X))^2) = E(X^2) - E(X)^2$
    - Chebyshev's Inequality

      $$P(|X - E(X)| \geq c \cdot \sigma) \leq \frac{1}{c^2}$$
    - One sided Hoeffding Bound
      - $X_1 \dots X_n$ are iid Bernoulli:

      $$P\left(\frac{1}{t} \sum_{i=1}^{t} X_i - E(X_i) \geq \varepsilon\right) \leq e^{-2\varepsilon^2 t}$$

- Sampling
  - ex. Random Sampling Vote Estimation
    - pick t voters X_i, $E(X_i) = p$
    - $\tilde{p} = \frac{1}{t} \sum X_i$
    - within ε w/ prob 1-δ unbiased approx
    - $t = \frac{1}{2\varepsilon^2} \log_e\left(\frac{2}{\delta}\right)$ via Chernoff Hoeffding
  - Reservoir Sampling
    - pick random element of stream
    - don't know length
    - maintain reservoir (current item)
    - replace w/ $p = \frac{1}{n}$
    - proof by induction
    - to sample t elements, size t reservoir, random between 1 and n
- Counting Distinct Elements
  - Distinct Elements Algorithm
    - hash function $h \to [0,1]$
    - get min val α; uniformly random
    - output 1/α
    - $O(\log n)$ bits
    - Random Hash Assumption
    - $E(\min h) = \frac{1}{k+1}$
    - $(1 - \frac{\varepsilon}{k})^k$ within probs
    - better to keep t smallest; out $\frac{t}{t^{th} \text{ smallest}}$, $\frac{t}{k}$
  - Pseudorandom Functions
    - need ~~[illegible]~~ lookup table of uniformly random
    - range $[1: \dots R]$ $R > n/c$
    - hash Family H
      - need one sufficiently random
      - needs pairwise independence
      - 1 in r^2 random

        $P(h(a) = r) = \frac{1}{R}$, $P(\text{same}) = \frac{1}{R^2}$
      - ex pick prime p

        $h_{a,b}(x) = a \cdot x + b \bmod p$

        all pairs $O(\log p)$ bits
  - Heavy Hitters
    - find element appear majority
    - $\log \Sigma + \log n$ bits memory
    - what if want all > f?
    - Count Min Sketch
      - $O((\log n)^2)$ memory
      - param l and B
      - $l = 2\log n$, $B = 2/\varepsilon$
      - all of l×B array
      - l random functions → [B]
      - for each x
        - for i = 1 to l
          - $M[i, h_i(x)]$++
      - estimate x # = $\min_i M[i, h_i(x)]$
      - $f_x \leq \min M[i, h_i(x)] \leq f_x + \frac{n}{B}$ w/ expectation
  - Memory Lower Bounds
    - if deterministic alg using $o(\min(|\Sigma|, n))$ memory solve heavy hitters or distinct elements then compress all bits to $o(L)$
      - contradiction, so still need exact streaming algs

- Extra
  - one to one = in 2 equal
  - universal hash - only as many bits to select first
    - proof by contradiction good (more direct actually)
      - show that if same, probability $\frac{1}{R}$, $\frac{1}{R^2}$
  - distinct element alg useful
    - think sets
  - ≥ 1 + deg f + deg g points for FFT
  - for "only if" draw implication
  - MST is lower bound in TSP
  - $a^{p-1} \equiv 1 \bmod p$ if a≠p
  - ln may help for probability weight stuff
  - Edit Distance

    $E[i,j]$ min from $x[1 \dots i]$ to $y[1 \dots j]$

    $$E[n,m] = \min \begin{cases} 1 + E[i, j-1] & \text{insertion} \\ 1 + E[i-1, j] & \text{deletion} \\ 1 + E[i-1, j-1] & \text{if } x[i] \neq y[j] \text{ sub} \\ 0 + E[i-1, j-1] & \text{if } x[i] = y[j] \text{ keep} \end{cases}$$
  - maximize probability is the max prob of product
  - remember Dijkstra's for nonnegative edges
  - $h_{a_1, a_2}(x_1, x_2) = a_1 x_1 + a_2 x_2 \bmod$ prime is universal
  - Longest Increasing Subsequence $O(n \log n)$
  - LP in P, simplex worst case exponential, average really good
  - always consider edge cases, base
  - Factoring ∈ NP
  - approx - rounding or greedy
  - big k OPT not uncommon