Sum on TI89: F3→sum(eq,i,lo,hi)

Complexity: $n!>2^n>n^3>n^2>n\lg n>n>\text{sqrt}(n)>\lg n>1$

Notations:

- Θ bounds fcn to within factors (tight bound)
- O=upper, Ω=lower
- Any linear fcn is also $O(n^2)$
- Use O for guaranteed blanket statement
- Right side coarser than left(i.e. $2n^2+\Theta(n)=\Theta(n^2)$)

Common fcns:

- Mod: $a\%n=a-n\lfloor a/n\rfloor$, $0\le a\%n<n$
- Polynomials: given $d\ge 0$, a poly in n of degree d is $p(n)=\sum_{i=0}^{d} a_i n^i$ where $a_0,a_1...a_d$ are the coeffs. of the poly and $a_d\neq 0$
- Exponentials for all real x: $e^x=1+x+x^2/2!+x^3/3!+...=\sum_{i=0}^{\infty}\frac{x^i}{i!}\ge 1+x$

Insertion sort(A) -- $\Theta(n^2)$

- For i=2...A.len
  - key=A[i]
  - J=i-1
  - While j>0 & A[j]>key
    - A[i+1]=A[i]
    - I--
  - A[i+1]=key
- Loop invariants: original elems of A[1...i-1], but now sorted
  - Initialization: is it true prior to 1^{st} loop?
  - Maintenance: if true before 1^{st}, still true before next?
  - Termination: after loop, invar. gives useful property that helps show if alg is correct

Binary search(sorted A, L(start@0), R(start@n-1)) -- O(lgn)

- If L>R return 1
- m=(L+R)/2
- If A_m<T, binsearch(A,m+1,R)
- If A_m>T, binsearch(A,L,m-1)
- Return m

Divide and conquer:

- If $n>c\to aT(n/b)+D(n)+C(n)$
- Conq+div+comb
- D=const, $C=n\to D(n)+C(n)=\Theta(n)$
- a=2, $b=2\to 2T(n/2)+\Theta(n)$ if n>1 (2 subprobs of ½ size)
- $=2T(n/2)+cn$ where c=time for n=1
- ⅔ and ⅓ subs: if D+C linear→recur=$T(⅔n)+T(⅓n)+\Theta(n)$
- Some have inconsistent sub sizes (ex: lin. search, sub=n-1→$T(n-1)+\Theta(n)$)
- **Merge sort -- Θ(nlgn)**
- Methods of solving recurrences:
  - Substitution: guess bound, subst. into eq./ineq. w/ T(m<n), show that it works for c>0, show that it holds for boundary conditions($n\ge n_0$)
  - Recursion tree: used to find good guess for subst. Sum cost of each level, then sum per-lvl costs
  - Master: $T(n)=aT(n/b)+f(n)$
    1. If $f(n)=O(n^{\log\_b(a)-e})$ for $e>0\to T(n)=\Theta(n^{\log\_b(a)})$
    2. If $f(n)=\Theta(n^{\log\_b(a)})\to T(n)=\Theta(n^{\log\_b(a)}\lg n)=\Theta(f(n)\lg n)$
    3. If $f(n)=\Omega(n^{\log\_b(a)+e})$ for $e>0\to T(n)=\Theta(f(n))$

Dynamic programming: when subprobs share subsubprobs

- When to use:
  - Optimal substructure: exhibited if an opt sol contains within it opt sols to subprobs
  - Overlapping subprobs: subprobs revisited multiple times
- Steps to develop:
  1. Characterize structure of an optimal solution
  2. Recursively define the value of opt sol
  3. Compute val of opt sol, typically bottom-up
  4. (Optional)Construct opt sol from computed information
- Top-down: write recursively in natural manner, but save result of each subprob
- Bottom-up: depends on natural subprob 'size.' Sort by size and solve smallest first
- 0-1 Knapsack: items are indivisible
  - **Brute force -- $O(n2^n)$**
    - N items→2^n combos
    - Itr through combos, find highest value
  - **DP -- O(nW)**
    - For w=0...W
      - B[0,w]=0
    - For i=0...n
      - B[i,0]=0
      - For w=0...W
        - If $w_i\le w$
          - If b_i+B[i-1,w-w_i]>B[i-1,w]
            - B[i,w]=b_i+B[i-1,w-w_i]
          - Else B[i,w]=B[i-1,w-w_i]
        - Else B[i,w=B[i-1,w]
- Longest increasing subsequence:
  - L[i] is len of LIS ending in A_i
  - Subs don't have to be consecutive
  - L[n]=1+max(L[j] for all j such that $A_j<A_n$)
  - LIS=max(L[1],L[2]...L[n])
  - **Recursive=$O(2^n)$**
  - **Bottom-up(DP)=$O(n^2)$**
- Longest common subsequence:
  - Bases must appear in same order, but not necessarily consecutively
  - Brute force requires enumerating all subs of X and checking if subs of Y, keeping track of longest
    - **Since X has 2^n subs, exp time required ($O(n2^m)$)**
  - **DP -- Θ(mn)**
    - B[1..m,1...n] to construct opt sol, c[0...m,0...n] to store length
    - For i=1...m,j=1...n
      - If X[i]==Y[j]...etc.

Greedy: always make *locally* optimal choice

- Elements:
  1. Determine opt substr
  2. Develop recursive sol
  3. Show that making greedy choice→only 1 subprob left
  4. Prove it's always safe to make greedy choice
  5. Develop recursive alg that implements greedy strat
  6. Convert rec alg into iterative alg
- **Fractional Knapsack -- O(nlogn)**
  - Keep taking item with highest value (benefit/weight ratio)
  - fKnap(S,W)
    - For i in S
      - $x_i=0$
      - $v_i=b_i/w_i$ //value
    - w=0 //current total weight
    - While w<W
      - Remove item i with highest v_i
      - $x_i=\min(w_i,W-w)$
      - $w=w+x_i$
- Scheduling:
  - Choose activity w/ earliest finish time and start time≥last act finish
  - **recursively(s,f,k,n) -- Θ(n)**
    - m=k+1
    - While m≤n & s[m]<f[k]
      - m=m+1
    - If m≤n
      - Return $\{a_m\}$Urecursively(s,f,m,n)
    - Else return
  - **iteratively(s,f) -- Θ(n)**
    - n=s.len
    - A=$\{a_1\}$
    - k=1
    - For m=2...n
      - If s[m]≥f[k]
        - A=AU$\{a_m\}$
        - k=m
    - Return A
- **coinchange(V,A) -- Θ(A)**
  - For i=n...0
    - While A≥V[i]
      - A=A-V[i]
      - C[i]++
  - min=sum(C)
- **huffman(C) -- O(nlgn)**
  - n=|C|
  - Q=C
  - For i=1...n-1
    - z.left=x=extract_min(Q)
    - z.right=y=extract_min(Q)
    - z.freq=x.freq+y.freq

        - Insert(Q,z)
    - Return extract_min(Q) //return root of tree

Graph Problems:

- Adjacency lists: good for *sparse* graphs, usually method of choice
- Adjacency matrix: only when graph is *dense* (|E| is close to $|V|^2$) or need to tell fast if 2 verts are adj
- **BFS -- O(V+E)** (linear in size of adj-list G)
    - Produces breadth-first tree
    - Finds shortest path from s to each v
- **DFS -- O(V+E)**
    - Search may repeat from multiple sources
        - Produces depth-first forest (several df trees)
    - Records timestamps of vert discovery and completion of examining v's adj-list
    - Discovery and finish times have well-nested *parenthesis structure*
    - **Topological Sort -- Θ(V+E)**
        - Linear ordering of all verts of *directed acyclic graph* such that if G contains edge (u,v), u appears before v in the ordering
        - Call DFS(G) and insert each vert onto front of linked list
- MSTs: Least cost to connect all verts acyclically (both algs greedy)
    - **Kruskal's -- O(ElgV)**
        - Set is a forest, edge added always lightest in the graph that connects two diff components
    - **Prim's -- O(ElgV), reduced to O(E+VlgV) using fib heaps if |V| << |E|**
        - Uses single tree, starts at arb root, adds lightest edge at each step
- Single-source shortest paths:
    - **Bellman-Ford -- O(VE)**
        - Weighted, directed graph
        - Allows negative edges, no negative cycles
        - Returns bool whether there is a neg cycle reachable from source, if there is, no solution exists, otherwise, produces paths and weights
        - Relaxation: testing if can improve path to v by going through u (source)
    - *By relaxing edges of weighted dag by topo sort, can compute sh paths from single src in **O(V+E)***
    - **Dijkstra's -- O((V+E)lgV), which is O(ElgV) if all verts are reachable from source. O(V^2) if E=o(V^2/lgV) <u>(always O(V^2), acc. to wiki)</u>**
        - Weighted, non-neg directed graph
        - Always chooses closest (greedy, but still guarantees optimal)

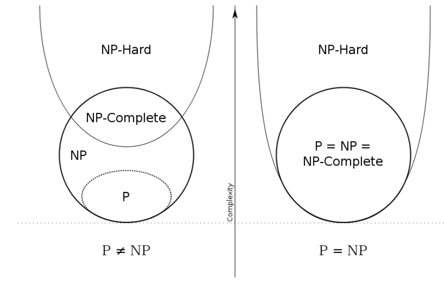
Linear Programming:

- Standard: all constraints ≤, obj fcn is maximization, all variables subject to non-neg
    - Change min to max by inverting obj fcn
    - Split x w/ no non-neg into x'-x'' where x', x''≥0
    - Change = to ineq by splitting into ≤ and ≥
    - Change ≥ to ≤ by inverting signs
- Slack: all eqs. except non-neg constraints
    - Make slack variables = right side of ineq - left
    - All slack vars non-neg
    - Omit "maximize" and "subject to," and non-negs, rep obj fcn w/ z=...
- **Shortest path:** max d(dest), d(src)=0, for ea. edge x->y, dy-dx≤edge weight
- **Product Mix Problem:** How many of each type of tie to make per month, given material cost and max amt of material available?
    - Find profit per tie, use as coeff for obj fcn
    - Constraints: sum of yards/type * type <= available for ea. material, min and max units of ea. type, non-negs
- **Transshipment:** Given m warehouses and n retailers with given supply and demand levels and costs from each W to R, how much product should be shipped along each route?
    - Add cost of each possible route for obj fcn
    - Constraints: Sum leaving W_i≤supply, sum entering R_j≥demand, non-negs for each route
- **Scheduling:** Given demand for buses at certain times, cover all buses with fewest drivers possible (ea. driver works 8 hr. shifts)
    - Min sum of # of drivers @ each 4 hr interval
    - Constraints: sum of two 4hr chunks for each 8hr shift (incl $x_{20}+x_4$)≥# needed for later chunk

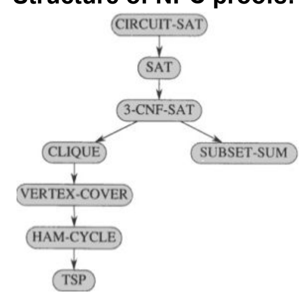
Complexity Classes:

- <u>P</u>: those problems solvable in poly time
- <u>NP</u>: problems verifiable in poly time (any problem in P is in NP)
- <u>NPH</u>: a poly time alg for the problem would imply a poly time alg for every prob in NP
- <u>NPC</u>: prob is NP and at least as hard as any problem in NP (both NPH and in NP)

- **Reductions:**
    - If $X \le_p Y$: ***Y is at least as hard as X***, if X is NPC and Y is in NP then Y is NPC, if Y in P then X in P
    - Given NPC X and Y and Z in P: $X \le_p Y$, if P≠NP, then no NPC can be solved in poly time
    - ***NPC problems can be reduced to one another***
- <u>3-CNF satisfiability</u>: bool formula contains vars whose values are 0/1, bool connectives like ∧, ∨, and □, and ( ). Bool formula is *satisfiable* if some assignment of 0s and 1s will cause it to evaluate to 1.
- <u>Bin Packing</u>: Can objects of diff vols be packed into k bins of vol V?
- <u>Circuit-SAT</u>: Given a bool circuit, is there a set of inputs that makes the circuit output True?
- <u>Clique</u>: Clique in an undirected G is subset of verts. Does a clique of size k exist in the graph?
- <u>Ham-Cycle/Path</u>: Does G have a ham cyc/path? (visits each vert exactly once)
- <u>Independent-Set</u>: Max size ind-set in G (set of verts in G, no two of which are adj)
- <u>K-Color</u>: Can verts of undir. G be colored w/ ≤ k colors, such that no vert matches any neighbors?
- <u>Knapsack</u>: Can a value of ≥V be reached without exceeding W?
- <u>Long-Path</u>: Find simple path (no repeated verts) of max length in G. Has linear time solution for dags.
- <u>Subset-Sum</u>: Given a set or multiset of ints, is there a non-empty subset (consec.) whose sum is 0?
- <u>TSP</u>: Shortest Hamiltonian Cycle in G?
- <u>Vertex-Cover</u>: Does G have vertex cover of size ≤k? (set of verts such that each edge of G is incident to at least one vert of the set)
- **P=NP vs. P≠NP:**



- **Structure of NPC proofs:**



(X->Y=X reduces to Y)