Master Theorem

$T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$

$a > 0 \quad b > 1 \quad d \geq 0$

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Common Equation Runtimes

$T(n) = 2T(\frac{n}{2}) + c \rightarrow O(n)$

$T(n) = T(\frac{n}{2}) + c \rightarrow O(\log n)$

$T(n) = T(\frac{n}{2}) + cn \rightarrow O(n)$

$T(n) = 2T(\frac{n}{2}) + cn \rightarrow O(n \log n)$

$T(n) = 2T(n-1) + 1 \rightarrow O(2^n)$

$T(n) = 4T(\frac{n}{2}) + n^2 \rightarrow O(n^2 \log n)$

$T(n) = 5T(\frac{n}{2}) + n \rightarrow O(n^{2.3})$

Linear Programming

Several real variables

As many linear constraints ($\geq, =, \leq$) as you want (capacity, flow, etc.)

A linear objective

Lempel-Ziv

0 | 1 | 00 | 01 | 011 | 000 ...

Dictionary:	Encoding:
0 - empty	00,
1 - 0	01,
2 - 1	10,
3 - 00	11,
4 - 01	41,
5 - 001	30,
6 - 000	⋮

Dynamic Programming

1. Meaning of each table entry
2. Write down recurrence relation
3. Size of table; order you fill it in
   → make sure recurrence doesn't use undefined values
4. Induction hypothesis: your recurrence (2) computes values that respect the meaning (1) for all entries previously filled out (induction time)
5. Induction Step
6. Find the answer where? (e.g. $T[n]$)

Greedy Steps

1. Your greedy alg. makes choices $c_1, \cdots, c_n$
2. We will show by induction on k that $c_1, \cdots, c_k$ (IH) can be completed into some opt. solution
3. Let OPT_k be the opt. solution that extends $c_1, \cdots, c_k$
4. Show that c_{k+1} is in OPT_k or that c_{k+1} can somehow be "swapped into" OPT_k w/o hurting it

Entropy of P

P_j = probability of j

$H(P) = -\sum_j P_j \log P_j$

$H(P^n) = n \cdot H(P)$

Capacity of s-t cut

Sum of all capacities leaving the s side of cut

→ upper bound on s-t flow

Huffmann

Keep merging smallest probabilities

Left 0, right 1

Always within 1-bit of opt. solution

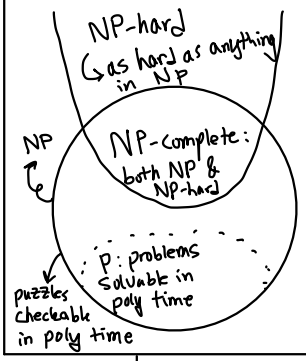
Kruskal

Add smallest edge that won't form cycle, repeat

DJ

```
for i ← 1 ~ n
    T[i] ← max(1,
        max  T[k]+1
        k<i : S[k] ≤ S[i]
        and L[k] < L[i])
return max_i T[i]
```

NP-completeness



Reduction

$B \leq_p A$

Reducing problem B to A

NP complete to your problem

Show yes → yes, no → no

(or yes → yes, yes ← yes)

Some NP-complete Problems

- **Independent Set**: given input G, k, is there a subset of k non-adjacent vertices?
- **Vertex Cover**: Given G, k, is there a set of $\leq k$ nodes that "covers" all edges
- **Edge Cover**: Same as Vertex, but now are there set of $\leq k$ edges that "covers" all nodes
- **3SAT**: 1 clause, which is 3 literals OR'd together, which is either x_i or $\overline{x_i}$. Can we satisfy all clauses?
- **Set Packing**: in the universe $1, \cdots, n$ with k sets $S_1, \cdots, S_m$, are there k of these sets w/ no overlaps (disjoint)?
- **Set Cover**: are there k of these sets that cover $1, \cdots, n$?
- **Circuit SAT**: Given circuit c, are there inputs that make c output "true"?
- **Subset Sum**: You have positive integers $c_1, \cdots, c_n$, is there a subset that adds up to k?
- **Hamiltonian Path**: is there a path in graph G that visits every vertex exactly once
- **Hamiltonian Cycle**: Same as above but cycle
- **Knapsack Problem**: if we have items w/ weights and values, fill up knapsack below weight limit while maximizing value

Yell into text editor

```
OOB, uninitialized → ∞
T[0,0] ← 0
for i ← 1 ~ n
    for j ← 0 ~ i-1
        T[i,j] ← min(2 + T[i-j, j],
                     1 + T[i-1, j])
    T[i,i] ← 3 + min_{j ∈ {0,...,i-1}} T[i,j]
return min_j T[n,j]
```

Some Example Recurrence Relationships

Battle game: Send i soldiers to battle j, $B[i,j]$ = score

$T[i,j]$ = best score possible using i soldiers for battles $1 \sim j$

$= \max_{k \in \{0, \cdots, i\}} B[k,j] + T[i-k, j-1]$

Answer @ $T[n,m]$ ($n \rightarrow$ total soldiers, $m \rightarrow$ last battle)

Longest Common Subsequence: $O(ij) = O(mn)$

$T[i,j]$ = LCS between $A[1, \cdots, i]$, $B[1, \cdots, j]$

Out-of-bounds access returns 0

$$T[i,j] = \begin{cases} \max(T[i-1,j], T[i,j-1]) & \text{if } A[i] = B[j] \\ 1 + T[i-1, j-1] & \text{otw} \end{cases}$$

Making Change: for n cents using $c_1, \cdots, c_k$

$T[i,j]$ = fewest # of coins to make i cents

$= \min(T[i-c_j, j] + 1, T[i, j-1])$ $O(nk)$

Bellman-Ford

$T[i,j,k] = \min_s T[i,s,k-1] + \text{len}(s \to j)$

if $T[\cdot,\cdot,n-1] \neq T[\cdot,\cdot,n]$ return $-\infty$

Big-O comparisons

$\log(\log n)^5 < \frac{\log n}{\log\log n} < e^{\log\log n} < \sqrt{n} + (\log n)^5 < 7^{\log n}$

(← slower)

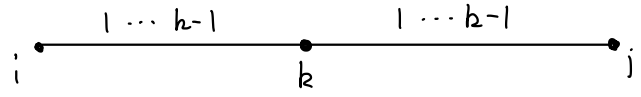
$< n^2(\log n)^4 + 2n^3 < 1000^{\sqrt{n}} < 3^{n/2} < (n+1)!$

Fast →

Floyd-Warshall

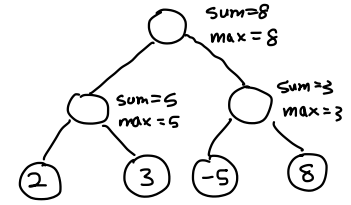
$T[i,j,k] = \min(T[i,j,k-1], T[i,k,k-1] + T[k,j,k-1])$

if $T[i,i,n] < 0$ for any i, return $-\infty$



$T[i,k,k-1]$ + $T[k,j,k-1]$

Metadata-Assisted Sum



sum = l.sum + r.sum

max = max(l.max, l.sum + r.max)

↳ cumulative max of sum starting from index 0

Knapsack Problem

n items, bag can hold k weight

w[i] → weight of item i

v[i] → value of item i

T[i,j] = using items $1,\cdots,i$, max value w/ weight j?

$= \max(T[i-1,j],\ v[i] + T[i-1, j-w[i]])$

Edit Distance

Best cost to edit string A to B

Cost 1 to insert, delete, modify,

0 to use

$T[i,j]$ = best cost editing first i of A into first j of B

$T[0,j] = j$ // insert j times

$T[i,0] = i$ // delete i times

$$T[i,j] = \min \begin{cases} T[i,j-1]+1 & \text{// ins} \\ T[i-1,j]+1 & \text{// del} \\ T[i-1,j-1] + (1 \text{ if } A[i] \neq B[j]) & \text{modify or use} \end{cases}$$

Best Tour

```
func Tour(A, B)
  for i ← 1 ~ n
    T[i] ← max(A[i], max_{h+B[h]<i} T[h] + A[i])
  return max_i T[i]
```

Valid Tours

```
func Tour(A, B)
  for i ← 1 ~ n
    T[i] ← 1 + Σ_{h:h+B[h]<i} T[h]
  return Σ_{i=1}^{n} T[i]
```

Dune Merging

$T[i,j]$ = best cost of merging dunes $i,\cdots,j$

$$= \begin{cases} 0 & \text{if } i=j \\ \min_{k \in \{i,\cdots,j-1\}} (T[i,k] + T[k+1,j] + c[i,j,k]) & \text{otw} \end{cases}$$

cost of merging (left pile i~j, right pile j+1~k)

Greedy Starbucks

$x \leftarrow$ drink our alg. makes @ time i+1

$y \leftarrow$ OPT makes @ time i+1

If $x=y$, IS holds; we are done!

Else, swap x into OPT @ time i+1 w/o hurting optimality

$d_1 \leftarrow$ time remaining on drink x

$d_2 \leftarrow$ time on drink y

B/c of our greedy alg. choosing x, $\boxed{d_1 \leq d_2}$

$c_1, c_2, \cdots c_{d_1+d_2} \leftarrow$ time OPT spends making drinks

Since $c_{d_1+d_2}$ is constant between both, we just need to minimize completion time of first-to-finish drink. But b/c of this, no drink can finish faster than d_1. Thus we can swap in x during OPT work time $c_1, c_2, \cdots, c_{d_1+d_2}$.

Word Wrapping

```
func Wrap(A, m)
  T[0] ← 0
  for i ← 1 ~ n
    T[i] ← min_{j ∈ {0,···,i-1}} T[j] + (m - ((i-j-1) + Σ_{k=j+1}^{i} A[k]))^2
  return min_{i : (n-i-1) + Σ_{k=i+1}^{n} A[k] ≤ m} T[i]
```