# 6.046

6.006 pre-requisite:
Data structures such as heaps, trees, graphs
Algorithms for sorting, shortest paths,
graph search, dynamic programming

Several modules:
- Divide & Conquer - FFT, randomized algs
- Optimization - greedy, dynamic prog
- Network Flow
- Intractability (and dealing with it)
- Linear programming
- Sublinear algorithms, approximation algs
- Advanced topics

Read course information & objectives on Stellar
Register on stellar for 6.046 (if you haven't already) and for a section
Pay particular attention to course collaboration policy!

# Theme of today's lecture

Very similar problems can have very different complexity.

Recall: P: class of problems solvable in polynomial time. $O(n^k)$ for some constant $k$

Shortest paths in a graph $O(V^2)$ e.g.

NP: class of problems verifiable in polynomial time.

Hamiltonian cycle a directed graph $G(V,E)$ is a simple cycle that contains each vertex in $V$.

Determining whether a graph has a hamiltonian cycle is NP-complete but verifying that a cycle is hamiltonian is easy.

$P \subseteq NP$ but is $P = NP$?

NP-complete: problem is in NP and is as hard as any problem in NP.

If any NPC problem can be solved in poly time, then every problem in NP has a poly time solution.

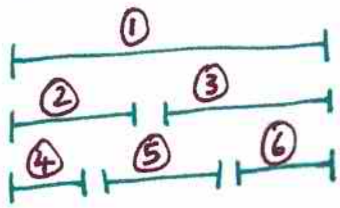
# Interval Scheduling

Resources & requests

Requests $1, \dots, n$, single resource

$s(i)$ start time, $f(i)$ finish time $s(i) < f(i)$

Two requests $i$ & $j$ are compatible if they don't overlap, i.e., $f(i) \le s(j)$ or $f(j) \le s(i)$




3 compatible requests

Goal: select a compatible subset of requests of maximum size.

Claim: We can solve this using a greedy algorithm.

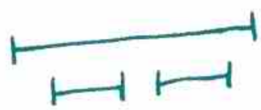
A greedy algorithm is a myopic algorithm that processes the input one piece at a time with no apparent look ahead

# Greedy Interval Scheduling

1. Use a simple rule to select a request $i$.
2. Reject all requests incompatible with $i$.
3. Repeat until all requests are processed.

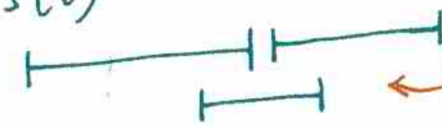
## Possible rules?

1. Select request that starts earliest, i.e., minimum $s(i)$



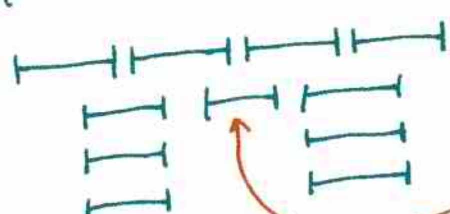
long one is earliest. bad!

2. Select request that is smallest, i.e., minimum $f(i) - s(i)$



smallest. bad!

3. For each request find # incompatibles. Select the one with minimum # incompatibles.



bad selection!

4. Select request with earliest finish time, i.e., minimum $f(i)$

**Claim:** Greedy algorithm outputs a list of intervals $(\langle s(i_1), f(i_1)\rangle, \langle s(i_2), f(i_2)\rangle, \ldots, \langle s(i_k), f(i_k)\rangle)$ such that $s(i_1) < f(i_1) \leq s(i_2) < f(i_2) \cdots \leq s(i_k) < f(i_k)$

**Proof:** If $f(i_j) > s(i_{j+1})$ interval $j+1$ $j$ intersect. Contradicts Step 2 of algorithm.

**Claim:** Given list of intervals $L$, greedy algorithm with earliest finish time produces $k^*$ intervals, where $k^*$ is optimal.

**Proof:** Induction on $k^*$

Base case: $k^* = 1$. Any interval works.

Suppose claim holds for $k^*$ and we are given a list of intervals whose optimal schedule has $k^*+1$ intervals, namely

$$S^*[1, 2, \ldots, k^*+1] = \langle s(j_1), f(j_1)\rangle, \ldots, \langle s(j_{k^*+1}), f(j_{k^*+1})\rangle$$

Say that $S[1, \dots k] = \langle S(i_1), f(i_1)\rangle, \cdots \langle S(i_k), f(i_k)\rangle$
is what the greedy algorithm gives.

By construction $f(i_1) \leq f(j_1) \leftarrow$ earliest finish time

Create schedule (this is valid!)

$$S^{**} = \langle S(i_1), f(i_1)\rangle, \langle S(j_2), f(j_2)\rangle, \dots, \langle S(j_{k^*+1}), f(j_{k^*+1})\rangle$$

This is also optimal.

Define $L'$ = set of intervals with $S(i) \geq f(i_1)$

Since $S^{**}$ is optimal for $L$, $S^{**}[2, \dots, k^*+1]$ is optimal for $L'$.

$\therefore$ optimal schedule for $L'$ has $k^*$ size.

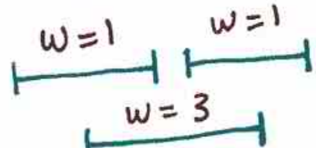
By inductive hypothesis, running greedy algorithm on $L'$ should produce a schedule of size $k^*$

By construction, running greedy algorithm on $L'$ gives us $S[2, \dots k]$

This means $k-1 = k^*$ or $k = k^*+1$
and $S[1, \dots k]$ is optimal.

## Weighted Interval Scheduling

Each request $i$ has weight $w(i)$
Schedule subset of requests with maximum weight.




greedy algo no longer works

## Dynamic Programming

Subproblems are

$$R^x = \{ \text{request } j \in R \mid s(j) \geq x \}$$

If we set $x = f(i)$ then $R^x$ is the set of requests later than request $i$

$n$ different subproblems, one for each request
Only need to solve each subproblem once & memoize

## DP Guessing

Try each request $i$ as a possible FIRST

If we pick request as the first request then remaining requests are $R^{f(i)}$

Note: There may be requests compatible with $i$ that are not in $R^{f(i)}$ but we are picking $i$ as the first request (i.e., we are going in order)

$$opt(R) = \max_{1 \leq i \leq n} \left( w_i + opt\left(R^{f(i)}\right)\right)$$

Running time? $O(n^2)$

Exercise: Use sorting initially & reduce DP complexity to $O(n)$. Overall complexity will be $O(n \log n)$

# NON-IDENTICAL MACHINES.

requests $1, \ldots n$, $s(i), f(i)$ as before
$m$ machine types $\mathcal{T} = \{T_1, \ldots T_m\}$
weight of 1 for each request.
$Q(i) \subseteq \mathcal{T}$ is set of machines that request $i$ can be serviced on.
Maximize the number of jobs that can be scheduled on the $m$ machines.

NP Can clearly check that any given subset of jobs with machine assignments is legal.
Can $k \le n$ requests be scheduled? NP-complete
Maximum requests should be scheduled. NP-hard.

## Dealing with Intractability

1) Approximation algorithms: Guarantee within some factor of optimal in poly time.
2) Pruning heuristics to reduce (possibly exp) runtime on "real-world" examples
3) Greedy or other suboptimal heuristics that work well in practice ← no guarantees