

Welcome to 6.046!

6.046: Design and Analysis of Algorithms

Instructors: Mauricio Karchmer, Alexander Mading, Bruce Tidor

Course Websites: [Stellor](#) + [Piazza](#) (make sure to sign up!)

Course Information Handout: ← READ CAREFULLY

- ① Prerequisites:
    - G.006 = "Basic" Algorithms
    - { G.042 = Discrete Math + Proofs }

CLEAR
  - ② Recitations:
    - On Fridays
    - Assigned by the Registrar
    - In case of schedule conflict, see LMODs
  - ③ Problem sets:
    - 10 weekly psets
    - $\leq 10$  grace days,  $\leq 2$  per pset
    - 2 weakest psets counted with half weight
    - TRUE collaboration - good copying - BAD
    - No "course bibles"
- MAKE SURE  
TO READ  
THE COLLABORATION  
POLICY !

IN DOUBT? ASK US FOR ✓  
CLARIFICATION 0

④ Exams: 2 quizzes (in-class) + Final ②

- Quiz I Tue, March 10, 7:30-9:30 PM
- Quiz II Thur, April 16, 7:30-9:30 PM
- Final TBD

Note: If you have conflict  $\Rightarrow$  let us know now  
(Interviews are NOT a valid excuse)

⑤ Grading policy:

Psets 10%  
Quiz I 25%  
Quiz II 25%  
Final 40%

⑥ Lecture videos: → In general, not as valuable as attending the lecture  
→ good for review

⑦ Important advice: → It is not about memorization (will allow cheat sheets on exams)

Note: No office hours on the pset due days  
→ Start working early!

- Can only learn by doing
- homework essential
- recitations - a great resource
- office hours!
- tutoring services

What is this class about?

G.006 = "algorithmic literacy"

G.046 = Art and craft of algorithms

Course schedule:

### Theme I: Techniques

- ① Divide & Conquer
- ② Amortized Analysis
- ③ Greedy Approach
- ④ Incremental Improvement
- ⑤ Linear Programming
- ⑥ Randomization / Random Walks
- ⑦ Continuous Optimization
- ⑧ Hashing
- ⑨ Reductions
- ⑩ Approximation algorithms

(like in MergeSort)  
 (as in prepaid phones)  
 (sometimes, big rock is the best rock)  
 (start bad, get better)  
 (geometry meets algebra)  
 (power of random coins)  
 ("continuous greedy")  
 (store data "randomly")  
 ( $A \rightarrow B$ )  
 (within 1%)

+ A bit on  
 intractability

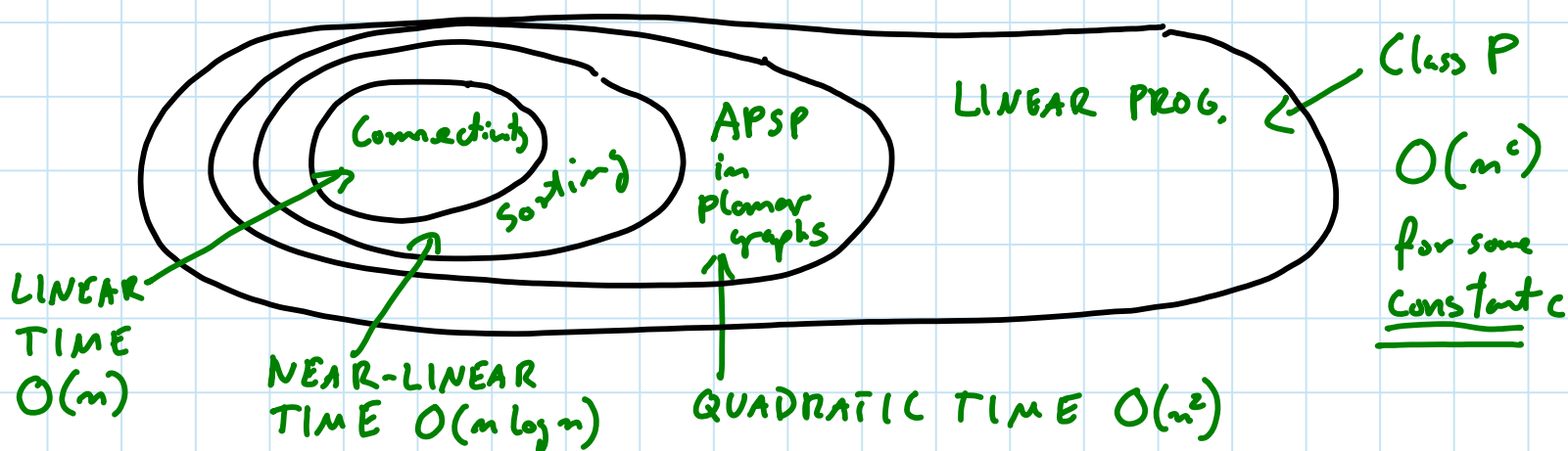
### Theme II: Computing Paradigms

- ① Online Algorithms
- ② Sampling Algorithms
- ③ Streaming Algorithms
- ④ Distributed Computing

(hedging against future)  
 ( $M - mc$ )  
 (little space)  
 (communication)

# Complexity Classes at a Glance:

(4)



P - class of problems solvable in  $O(n^c)$  time, with  $c > 0$  constant  
E.g. Is this list of numbers sorted?

NP - class of problems verifiable in polynomial time

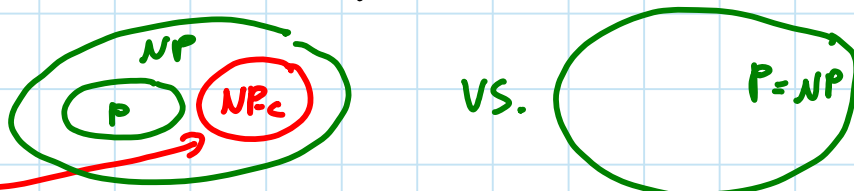
E.g. Determine if a graph has an Eulerian/Hamiltonian cycle

Eulerian cycle - a cycle that visits each edge exactly once

Hamiltonian cycle - a cycle that visits each vertex exactly once

Clearly:  $P \subseteq NP$

Central question:  $P = NP?$



NP-complete problem: Problem is in NP and if it is in P too then  $P = NP$  (i.e. the "hardest" prob. in NP)  
 $\Rightarrow$  if  $P \neq NP$  then NP-comp. is not in P

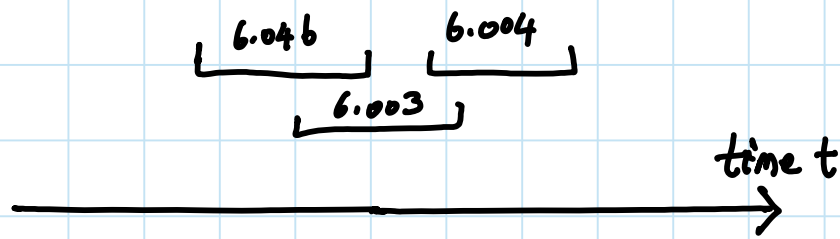
More in Lec 20-21

Message for today: Very similar problems can have very different solutions & complexity

E.g. Checking if a graph has an Eulerian cycle is in  $O(n)$   
checking if there is a Hamiltonian cycle is NP-complete

# Interval Scheduling

(Scheduling classes Ver 1)



- Single resource (human, CPU, classroom etc..)
- n requests to use each for an interval

$$r_i = [a_i, b_i]$$

$$R = \{r_1, \dots, r_n\}$$

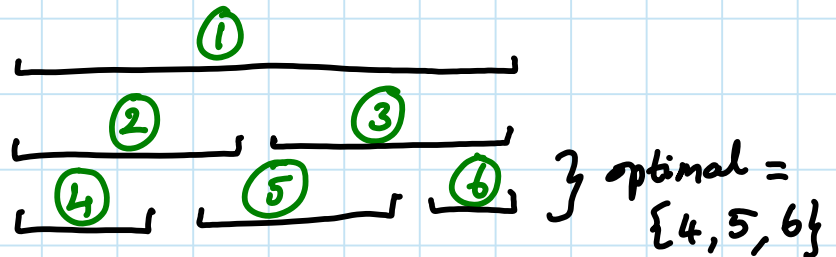
- requests  $i$  &  $j$  compatible iff  $r_i \cap r_j = \emptyset$

$$\text{Inc}(r) = \{r' : r \cap r' \neq \emptyset\}$$

**GOAL**

Find maximum number of mutually compatible requests.

**EXAMPLE**



**CLAIM**

"Greedy is best".

Greedy algorithm  $\approx$  Repeatedly make locally best choice with no "look-ahead"

More in Lecture 10

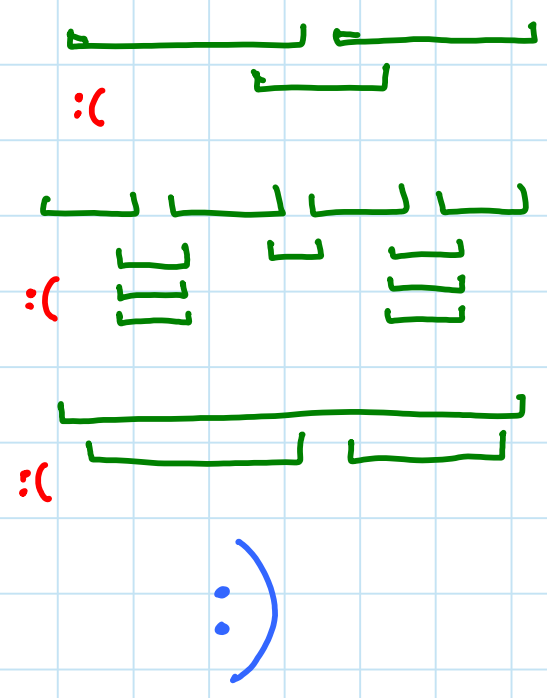
# Greedy Interval Scheduling

- use a simple rule to pick an interval  $r_i$
- include  $r_i$  in solution; discard all intervals  $r'$  incompatible with  $r_i$   $\in I_{inc}(r_i)$
- repeat until no more intervals.  
( $\Rightarrow$  all scheduled or incompatible)

Million-\$ question: Which rule to use?

## Possible Rules

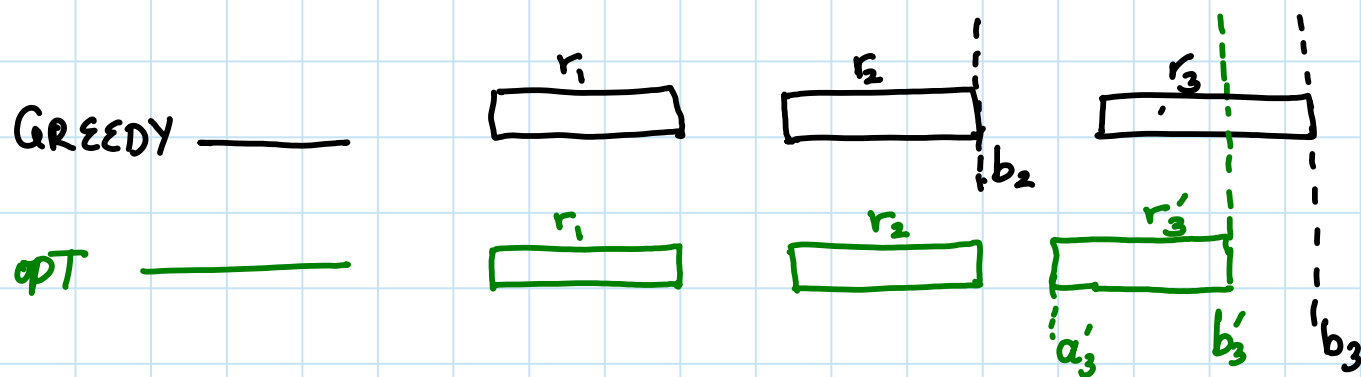
- ① Select smallest interval  
i.e, minimum  $b_i - a_i$
- ② select interval with fewest  
incompatibilities
- ③ Select interval that starts  
earliest, i.e, minimum  $a_i$
- ④ select interval that finishes  
first, i.e, minimum  $b_i$



Theorem: Greedy Interval Scheduling  
with rule ④ is optimal.

# PROOF BY AN "EXCHANGE/HYBRID ARGUMENT" ⑦

(Transform any <sup>optimal</sup> solution into the greedy solution with No loss in quality)



Let  $r_i$  be the first interval where GREEDY & opt differ (here,  $i=3$ )

CLAIM  $a_3' > b_2$

Proof:  $b_2 = b_2' < a_3'$  & opt needs to be feasible  
(i.e.  $[a_2, b_2] \cap [a_3', b_3'] = \emptyset$ )

Now, consider two cases.

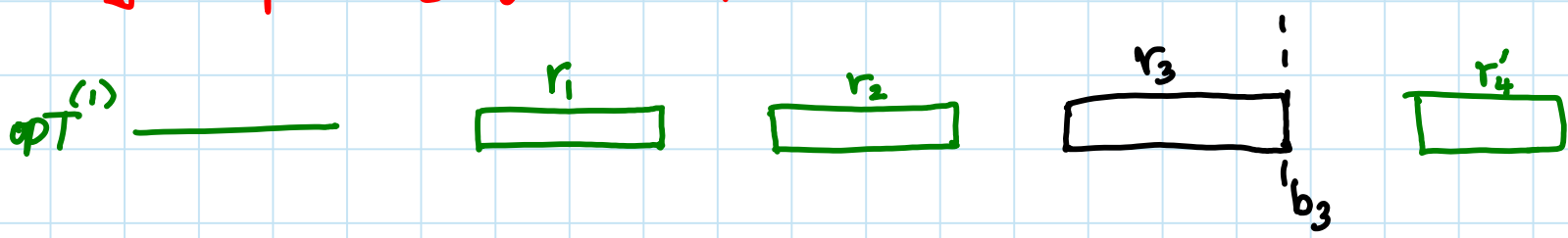
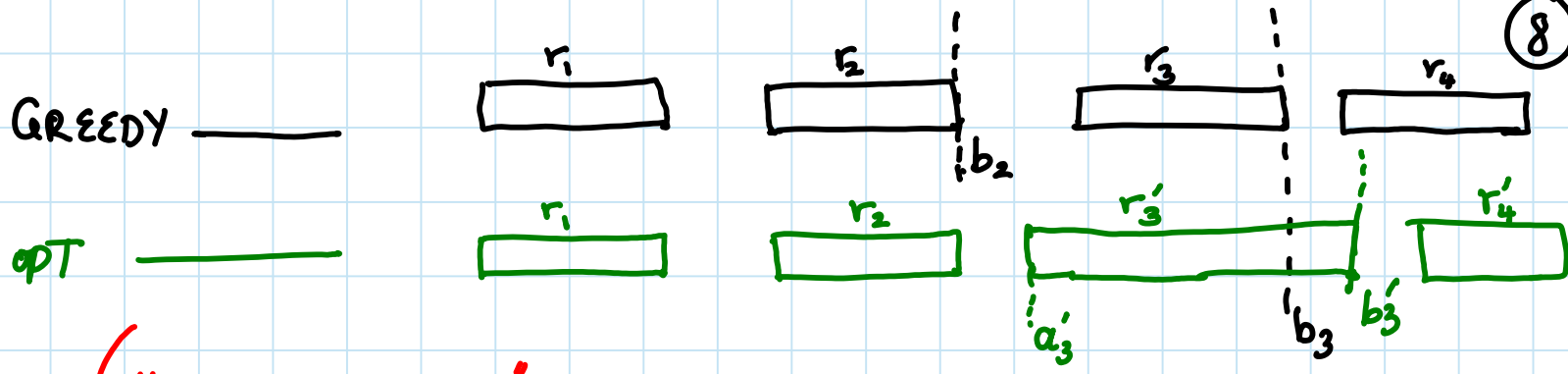
Case 1 (as above)

$$b_3' < b_3$$

This cannot happen as GREEDY would then have chosen  $r_3'$  instead of  $r_3$ .

Case 2 (see below)

$$b_3' \geq b_3$$



**CLAIM**  $OPT^{(1)}$  is as good as OPT.

**CLAIM**  $OPT^{(1)}$  agrees with GREEDY on one more request (than OPT)

- 
- 
- continue with GREEDY and  $OPT^{(1)}$
- 
- 

$$\#jobs(OPT) = \#jobs(OPT^{(1)}) = \#jobs(OPT^{(2)}) = \dots = \#jobs(OPT^{(k)}) = \#jobs(GREEDY)$$

$\therefore$  GREEDY produces an optimal solution :)

Running Time =  $O(n \lg n)$

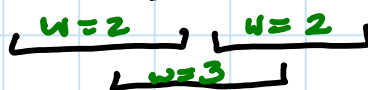
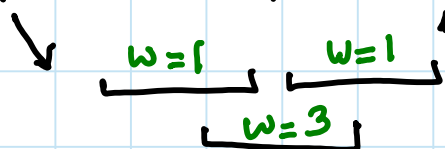
- sort  $b_i$  in ascending order
  - consider each interval  $r_i$  in order, remembering end time  $b_j$  of last scheduled interval  $r_j$
  - if  $a_i > b_j$  schedule  $r_i$  else discard  $r_i$
- $\left. \begin{array}{l} \text{first two steps} \\ \text{third step} \end{array} \right\} \begin{array}{l} O(n \lg n) \\ O(n) \end{array}$



# Weighted Interval Scheduling (Scheduling classes Ver 2)

- Each interval  $r_i$  has weight  $w_i$
- Goal: Schedule (non-conflicting) subset of max weight

- Greedy seems to fail  
(earliest start, earliest finish, max weight first, ...)



As in 6.006

- Dynamic Programming

- sort by start time so that  $a_1 \leq a_2 \leq \dots \leq a_n$
- Is  $r_i \in \text{opt}(R)$ ? Don't know, so guess

Case (1)  $r_i \notin \text{opt}(R)$

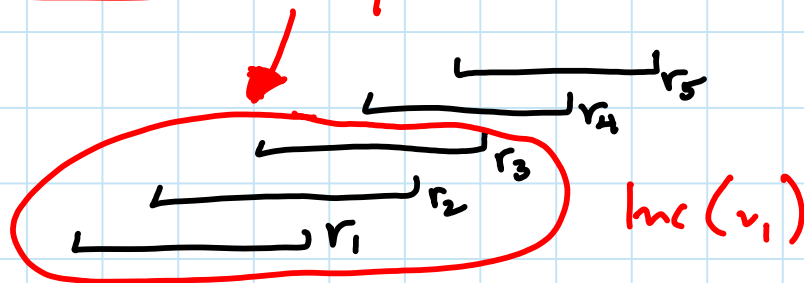
$$\Rightarrow \text{opt}(R) = \text{opt}(R - r_i)$$

Case (2)  $r_i \in \text{opt}(R)$

$$\Rightarrow \text{opt}(R) = w_i + \text{opt}(R - \text{inc}(r_i))$$

where  $\text{inc}(r_i)$  are requests incompatible with  $r_i$

Key Lemma: incompatibilities  $\text{inc}(r_i)$  form a prefix



$$\text{So, } \text{opt}(R) = \max \left\{ \text{opt}(R - r_1), \right. \\ \left. w_1 + \text{opt}(R - \underbrace{r_1 - r_2 - \dots - r_i}_{\text{inc}(r_1)}) \right\}$$

- Subproblems:  $n$   
one for each suffix  $r_j, r_{j+1}, \dots, r_n$
- 2 "guesses"
- $O(n)$  time to find the right suffix  $(R - r_1 - r_2 - \dots - r_i)$

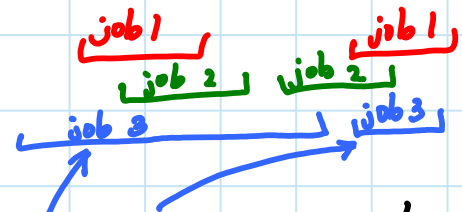
$\Rightarrow O(n^2)$  total time

Ex: Reduce to  $O(n \lg n)$  time

Hint: Binary Search

### Job Interval Scheduling (scheduling recitations)

- Job  $i$  consists of
  - $k$  possible intervals  $J_i = \{J_{i1}, J_{i2}, \dots, J_{ik}\}$   
when job  $i$  can be scheduled
  - $n$  such Jobs



- Goal: Find subset of intervals of maximum size
  - at most one from each set  $J_i$
  - non-conflicting

- NP-complete

## Lecture 20-21

- a greedy algorithm finds a solution within factor 2 of optimal.

## Lecture 22

- better algorithm achieves  $\frac{e}{e-1} \approx 1.6$

[Chuzhoy Ostrovsky Rabani 2006]

Take-Home 1: Scheduling classes is in P, scheduling recitations is NP-hard

Take-Home 2:

Small changes in problem statement can cause a big change in whether it is tractable and what are the best techniques to use

<u>Problem</u>	<u>Complexity</u>	<u>Technique</u>
Interval Scheduling	P $O(n \lg n)$ time	Greedy
Weighted Interval Scheduling	P $O(n \lg n)$ time	Dyn. Prog.
Flexible Interval scheduling	NP-complete	?