Welcome to 6.046!

6.046: Design and Analysis of Algorithms

Instructors: Debayan Gupta, Aleksander Madry, Bruce Tidor

Course Websites: Stellar + Piazza (make sure to sign up!)

Course Information Handout: ← READ CAREFULLY

(1) Pre Requisites: 6.006 = "Basic" Algorithms
{ 6.042 = Discrete Math + Proofs }
CLEAR

(2) Recitations:
→ On Fridays
→ Assigned by the Registrar
→ In case of schedule conflict, see https://goo.gl/dQ24ge

(3) Problem sets:
→ 10 weekly psets
→ ≤ 10 grace days, ≤ 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

④ Exams: 2 quizzes (in-class) + Final

→ Quiz I — Tue, March 14, 7:30-9:30 PM
→ Quiz II — Thu, April 13, 7:30-9:30 PM
→ Final — TBD

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

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

What is this class about?

6.006 = "algorithmic literacy"

6.046 = Art and craft of algorithms

Course schedule:

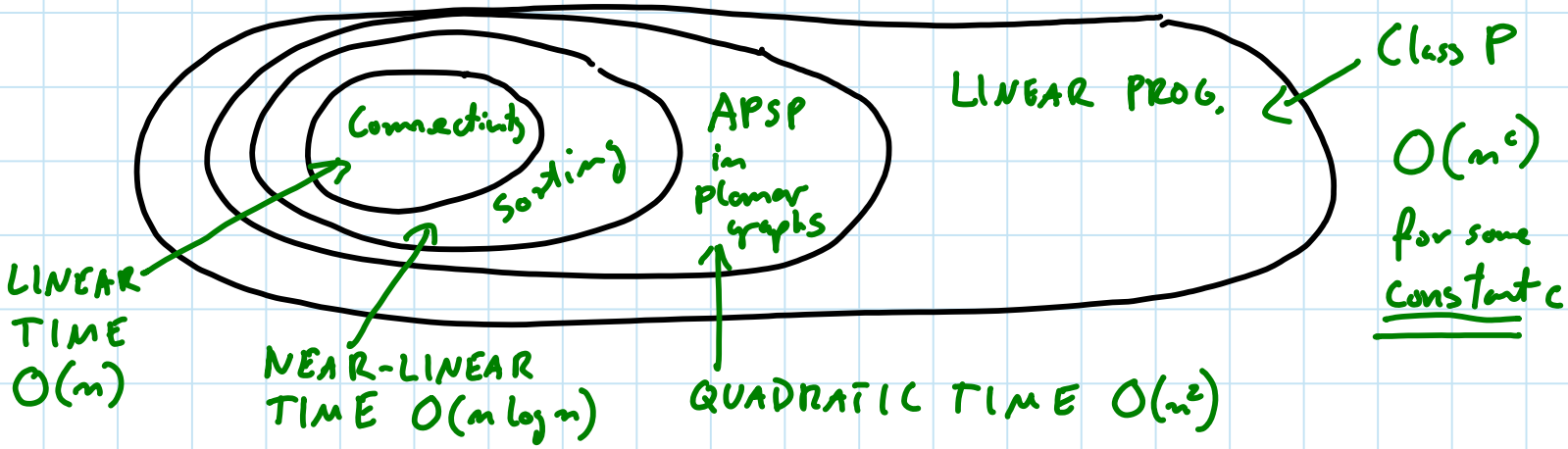
Theme I: Techniques

1. Divide & Conquer (like in MergeSort)
2. Amortized Analysis (as in prepaid phones)
3. Greedy Approach (sometimes, big rock is the best rock)
4. Incremental Improvement (start bad, get better)
5. Linear Programming (geometry meets algebra)
6. Randomization / Random Walks (power of random coins)
7. Continuous Optimization ("continuous greedy")
8. Dynamic Programming (memoization & beyond)
9. Reductions (A → B)
10. Approximation algorithms (within 1%)

Theme II: Computing Paradigms

1. Online Algorithms (hedging against future)
2. Sampling Algorithms (MC-MC)
3. Streaming Algorithms (little space)
4. Parallel Computing (do stuff simultaneously)
5. Distributed Computing (communication)

Complexity Classes at a Glance :



P - class of problems solvable in $O(n^c)$ time, with $c > 0$ constant
E.g. APSP in a graph

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: $P = NP$?

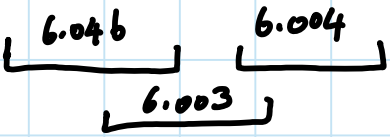
NP-complete problem: Problem is in NP and if it is in P too
then $P = NP$ (i.e. the "hardest" prob. in NP)
$\Rightarrow$ if $P \neq NP$ then NP-comp. is not in P

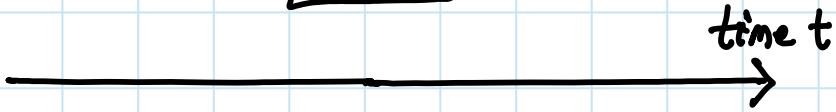
More in Lec 19-20

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)





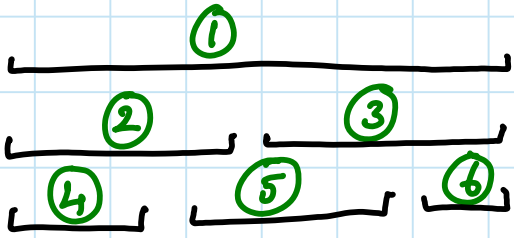
- Single resource (human, CPU, classroom etc...)
- n requests to use each for an interval

$r_i = [a_i, b_i]$ $\quad R = \{r_1, \ldots, r_n\}$

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

GOAL Find maximum number of mutually compatible requests.

EXAMPLE



} optimal = {4, 5, 6}

CLAIM "Greedy is best".

More in Lecture 6

Greedy algorithm ≈ Repeatedly make locally best choice with no "look-ahead"

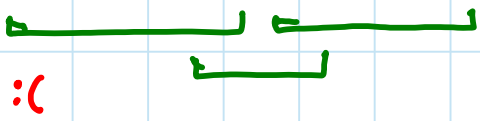
Greedy Interval Scheduling

- use a simple rule to pick an interval r_i
- include r_i in solution; discard all intervals incompatible with r_i
- repeat until no more intervals.
  (⇒ all scheduled or incompatible)

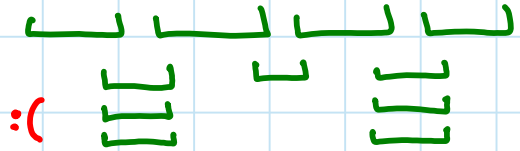
Million-$ question: Which rule to use ?

Possible Rules

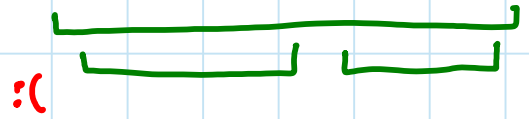
① Select smallest interval
i.e, minimum $b_i - a_i$



② select interval with fewest incompatibilities



③ Select interval that starts earliest, i.e, minimum a_i



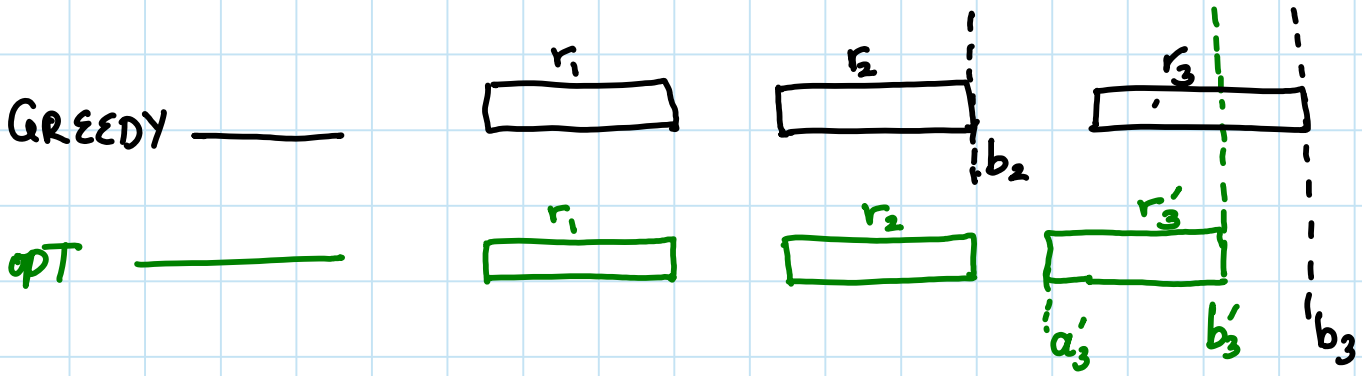
④ Select interval that finishes first, i.e, minimum b_i

:)

Theorem: Greedy Interval Scheduling with rule ④ is optimal.

PROOF BY AN "EXCHANGE/HYBRID ARGUMENT"

(Transform any solution to the greedy solution with No loss in quality)



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

CLAIM $a_3' > b_2$.

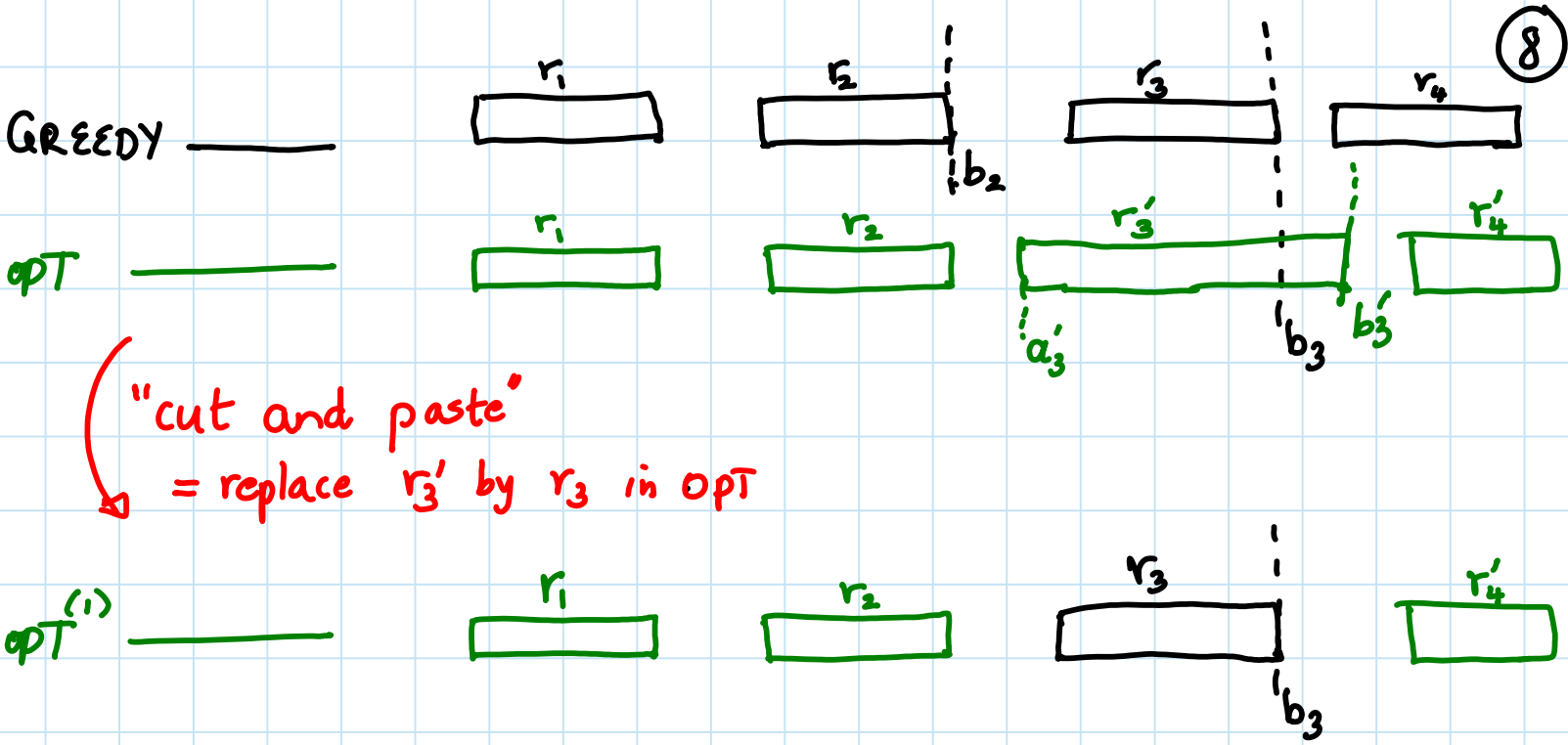
pf: If not, GREEDY and OPT must differ in r_2, contrary to the assumption that r_3 is the first location where they differ. ■

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 $OPT^{(1)}$

⋮

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

$\therefore$ GREEDY produces an optimal solution :)

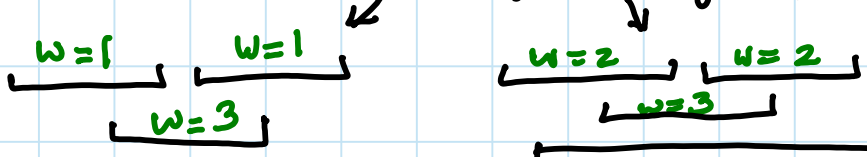
Running Time = $O(n \lg n)$

- sort b_i in ascending order } $O(n \lg n)$
- 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 } $O(n)$

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, ...)



More in Lec 17-18

- Dynamic Programming
  - sort by start time so that $a_1 \le a_2 \le \dots \le a_n$
  - Is $r_1 \in OPT(R)$? Don't know, so guess

Case ① $r_1 \notin OPT(R)$

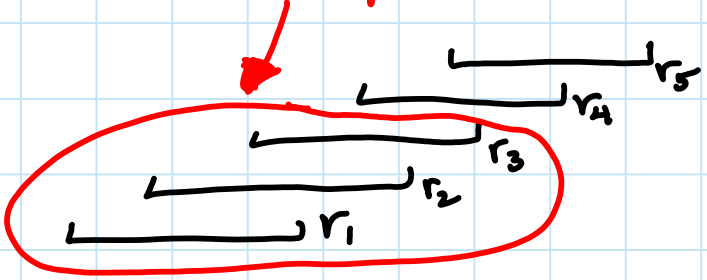
$\Rightarrow OPT(R) = OPT(R - r_1)$

Case ② $r_1 \in OPT(R)$

$\Rightarrow OPT(R) = w_1 + OPT(R - r_1 - r_2 \dots - r_i)$

where $r_2, r_3, \dots, r_i$ are incompatible with r_1

Key Lemma: incompatibilities (r_i) form a prefix



So, $OPT(R) = \max \{ OPT(R - r_1),$
$w_1 + OPT(R - r_1 - r_2 - \cdots - r_i) \}$

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

$\Rightarrow O(n^2)$ 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 $J_i = \{J_{i1}, J_{i2}, \ldots, J_{ik}\}$
    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 19-20

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

  Lecture 21

- 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

Problem	Complexity	Technique
Interval Scheduling	P O(n lg n) time	Greedy
Weighted Interval Schedulg	P O(n lg n) time	Dyn. Prog.
Flexible Interval scheduling	NP-complete	?