Here Early?

Here for CSE 421?

Welcome! You're early!

Want a copy of these slides to take notes?

You can download them from the webpage cs.uw.edu/421

Stable Matchings

CSE 421 Spring 26
Lecture 1

Today

Logistics

What is this course?

Start of the content

Staff

Instructor: Robbie Weber

Ph.D. from UW CSE in theory
Sixth year as teaching faculty

Office: CSE2 311
Email: rtweber2@cs.washington.edu

TAs
Aadi Anand
Owen Boseley
Isaiah Greenlee
Caleb Hsu
Emma Huang
Shayla Huang
Brice Liu
Vinay Pritamani

TODO List

Make sure you're on Ed! (check your spam folder for an invite, if not there send an email to Robbie).

Go to section Thursday.

Homework 1 will be out Wednesday (should be able to start after section).

Syllabus

It's all on the webpage:
https://courses.cs.washington.edu/courses/cse421/26sp/

What do we do in this course?

Our goal in this course is to develop a skill set in you:

When you're faced with a new problem you should be able to:

- Extract the needed technical information from the problem
- Use one of a variety of techniques to design an algorithm for the problem.
  - Graph modeling, Divide and Conquer, Dynamic programming, linear programming, greedy algorithms,…
- Justify the correctness of the algorithm (e.g., with a proof).
- Evaluate whether someone else's proposed algorithm is correct and whether it is efficient.

Developing a Skill Set Takes Practice

Lecture will give you the high-level ideas, and a bit of practice.

Section will give you scaffolded practice.

Homework is where you practice (by yourself and with others).

So that on exams you can demonstrate what you've learned.

This class has a reputation for difficulty

It can be difficult!

It's because the skills we're trying to train you in are difficult to master.

I think those skills are still useful and that you can't gain those skills without that significant effort.

Non-LLM aided effort (or minimally-LLM-aided in specific ways)

If you think you'll learn faster/better with an LLM

there are plenty of resources online (free textbooks, videos, leetcode, etc.)

you can go learn independently (take a different class).

If you don't think the skills are useful, go take a different elective!

You don't need this course to graduate.

Please don’t use LLMs, except as allowed

The course is also difficult for staff---it’s very time consuming to grade algorithms problems.

I wouldn’t make them do it if I didn’t think it is the best way for you to learn.

Grading LLM-generated submissions is demoralizing for us.

Every minute we spend on investigating/reporting violations of policy is a minute we’re not spending on helping everyone else learn.

Writing better homeworks, improving sections, more detailed feedback…

Violating the policy has negative impacts on you, the staff, and your fellow students.

The penalties for disallowed LLM usage (or any violation of academic integrity) are significant.

Syllabus video coming this week

With a description of the AI (and human-collaboration) policy, and other relevant policies.

Or just go read the syllabus today.

If you don't intend to follow it, please don't take this course.

We **don't** want this class to feel adversarial. I want this to be the last mention of anything other than algorithms (and helping you learn how to design them) for the rest of the quarter.

This Class is also Exciting

The “aha” moment when you realize how to solve a problem should be very satisfying

We’ll get to see some classic, cool algorithm ideas

We’ll (hopefully!) change the way you think about problem solving.

What is this course?

Algorithms

themes:

"Design Techniques" – not just "here's an algorithm" but "here's a way of thinking about a class of algorithms"

"Modeling" – In the real world, no one will say "I need you to run Prim's algorithm on this graph" they will say "I need you to choose where to build electrical wires so every town is connected to the power plant as cheaply as possible

"Set realistic expectations" – there are some things we (think/know) computers can't do efficiently. How do you recognize these problems?

"Reductions" – if you've already solved a problem, don't solve it again (reuse ideas) and if you know you can't solve a problem, what else can't you solve.

What is this course **not**

NOT: A list of the fastest-known algorithms for common problems.

Not in practice: I'm not concerned with which library is best.

The best library changes over time and by language.

I'm not qualified to keep a list.

I want you to find this course useful 5 years from now.

Not in theory: the best theoretical algorithms often aren't practical…

and when they are, it's often clever combinations/complicated variants of big ideas that we'll see.

What to learn

The goal of this course is to form you into stronger computer scientists.

People who can look at a new problem and figure out how to solve it.

Or who can look at someone else's algorithm idea and explain why it won't work.

Course Topics (Tentative)

Stable Matchings

Graph Algorithms/Modeling (BFS/DFS and some new algorithms)

Greedy Algorithms

Divide & Conquer

Dynamic Programming

Network Flow

Linear Programming

P/NP

What's Coming Up

This week: An extremely useful algorithm.

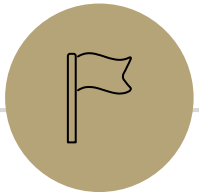
That has had lots of effect on the real world.

Along the way:

Examples of what we mean by an algorithm description

We won't use Java

How do we prove algorithms correct? Resetting expectations on proofs.



Content: Stable Matchings

Stable Matchings (motivation)

Motivation:

You have to assign TAs to instructors.

Two groups of people you need to pair off, with preferences about their matches.
You can't make everyone happy…so at least ensure that everyone listens to you.

There are lots of other similar applications

- Assign doctors to the hospitals where they do residency.
- Assign high schoolers to magnet schools.
- Among many, many other applications.

Motivation

The real world is complicated.
Students shouldn't TA a course they haven't taken.
Instructors need varying numbers of TAs.
There are more TA applicants than positions.
Doctors might want to be in the same city as their partner.

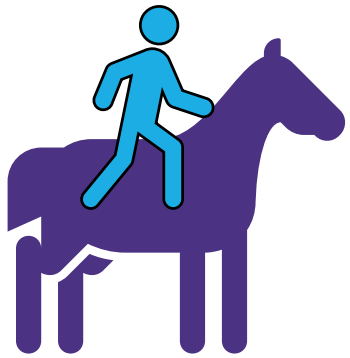
We're going to simplify away the real world constraints.
The core ideas have been adapted to all of these scenarios.

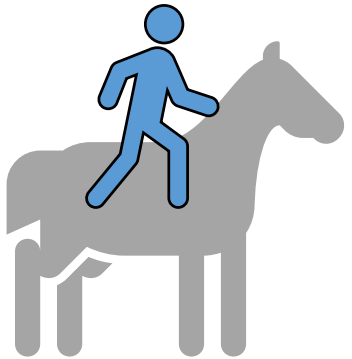
Stable Matching Problem

To simplify. We have two sets:
A set of n horses, and a set of n riders.

Every rider can ride any horse, and vice versa.

We just need to pair them off. What could go wrong?





Stable Matching Problem (formally)

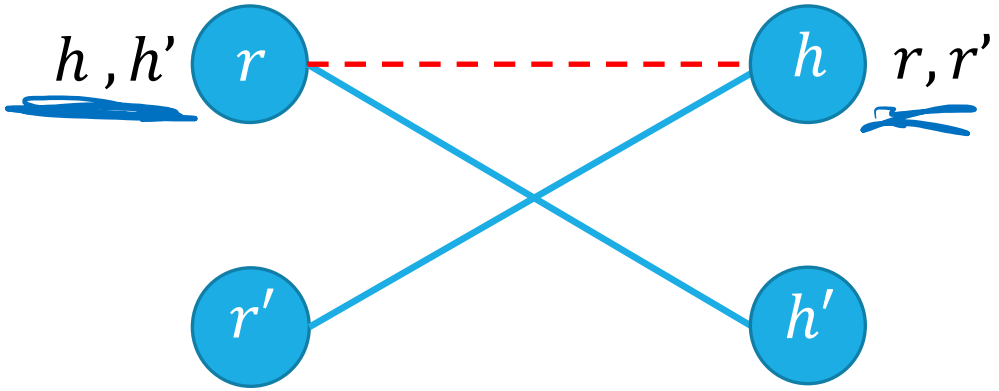
Given two sets $\mathrm{R} = \{r_1, \ldots, r_n\}, H = \{h_1, \ldots, h_n\}$

each agent ranks **every** agent in the other set.

Goal: Match each agent to **exactly one** agent in the other set, respecting their preferences.

How do we "respect preferences"?

Avoid blocking pairs: unmatched pairs (r, h) where r prefers h to their match, and h prefers r to its match.



Stable Matching, More Formally

Perfect matching:

- Each rider is paired with exactly one horse.
- Each horse is paired with exactly one rider.

Stability: no ability to exchange

an unmatched pair r-h is blocking if they both prefer each other to current matches.

Stable matching: perfect matching with no blocking pairs.

Stable Matching Problem

Given: the preference lists of n riders and n horses.
Find: a stable matching.

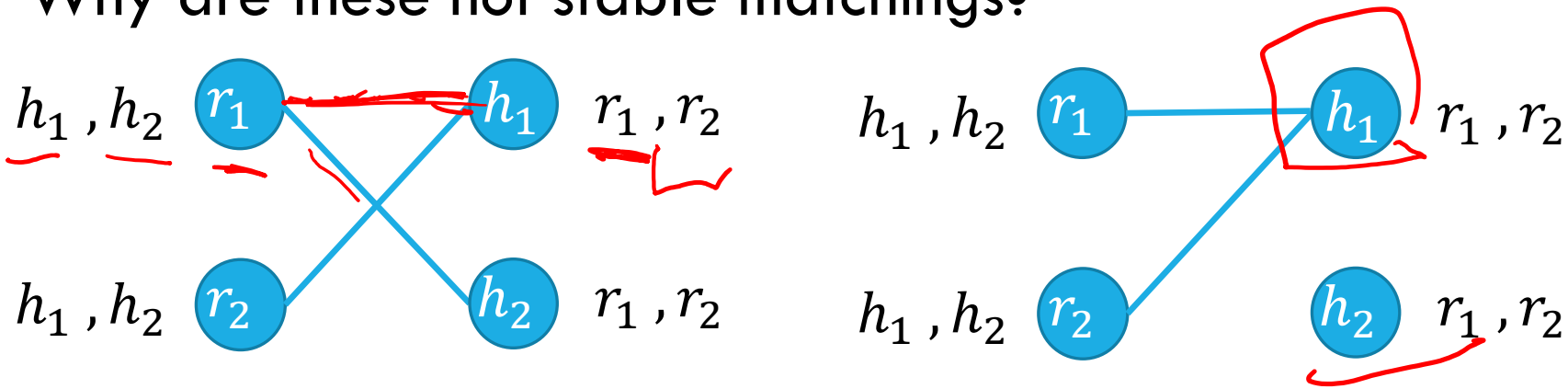
Lecture Activity

To make sure you've got the definition:

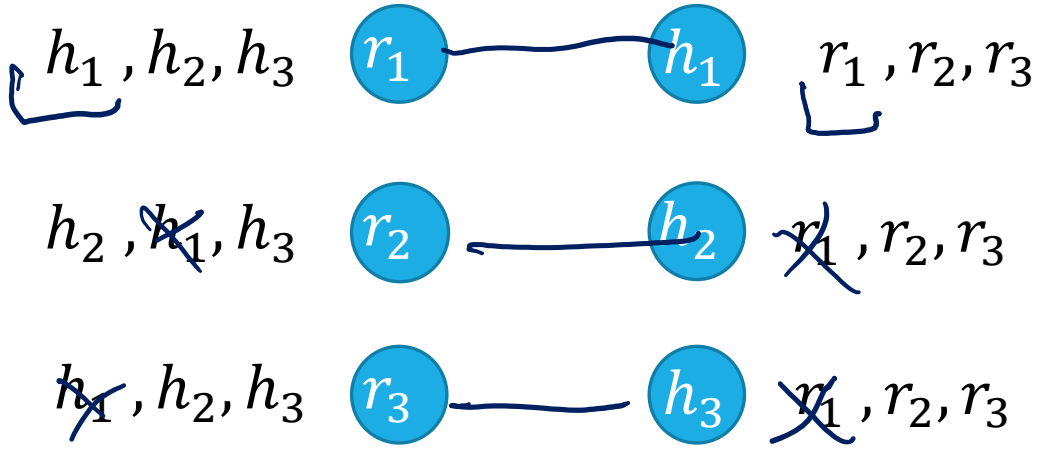
1. Download the activity pdf from the webpage (it's just the next slide in this slide deck) or look at the physical handout.
2. Introduce yourself to those around you.
3. Try the problem.
4. Fill out the polleverywhere

Try it!

Why are these not stable matchings?



Find a stable matching for this instance.



pollev.com/robbie

Questions

- Does a stable matching always exist?
- Can we find a stable matching efficiently?

We'll answer both of those questions in the next few lectures.

Let's start with the second one.

Idea for an Algorithm

Key idea

Unmatched riders “propose” to the highest horse on their preference list that they have not already proposed to.

Send in a rider to walk up to their favorite horse.

Everyone in front of a different horse? Done!

If more than one rider is at the same horse, let the horse decide its favorite.

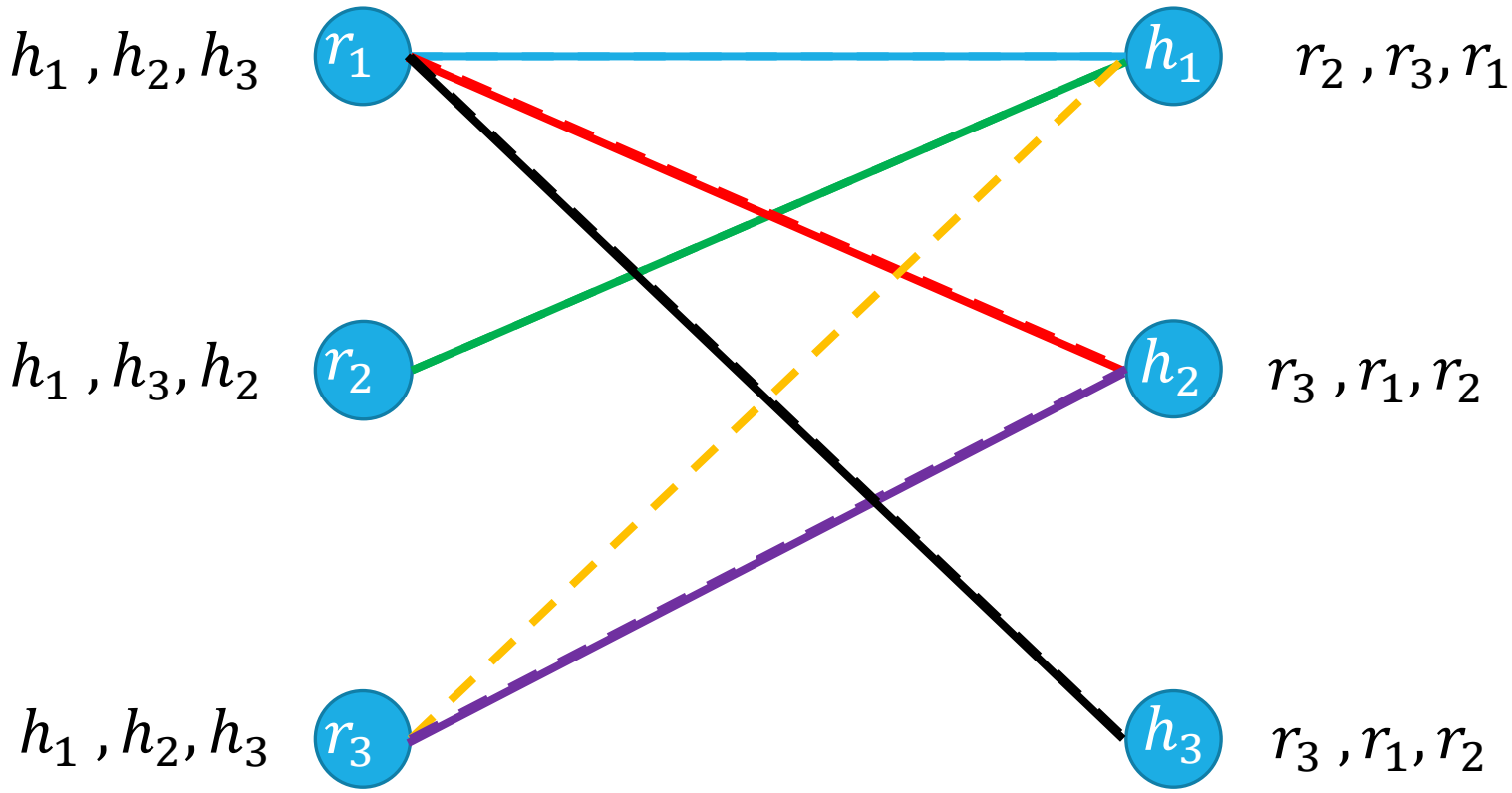
Rejected riders go back outside.

Repeat until you have a perfect matching.

Gale-Shapley Algorithm

```
Initially all r in R and h in H are free
while there is a free r
      Let h be highest on rs list that r has not proposed to
      if h is free
            match (r, h)
      else //h is not free
            Let r' be the current match of h.
            if h prefers r to r'
                  unmatch (r', h)
                  match (r, h)
```

Algorithm Example



Proposals: $r_1, r_2, r_1, r_3, r_3, r_1$

Does this algorithm work?

Does it run in a reasonable amount of time?

Is the result correct (i.e. a stable matching)?

Begin by identifying invariants and measures of progress

- Observation A: r's proposals get worse for them.
- Observation B: Once h is matched, h stays matched.
- Observation C: h's partners get better.

How do we justify these? A one-sentence explanation would suffice for each of these on the homework.

How did we know these were the right observations? Practice. And editing – we wouldn't have found these the first time, but after reading through early proof attempts.

Does this algorithm work?(two claims)

Want to show two things:

1. The code produces the right output (i.e., you get a stable matching)
2. The code runs in a reasonable amount of time.

We'll start with question 2.

Claim 1: If r proposed to the last horse on their list, then all the horses are matched.

Claim 1: If r proposed to the last horse on their list, then all the horses are matched. (2)

Try to prove this claim, i.e. clearly explain why it is true. You might want some of these observations:

Observation A: **r's proposals get worse (for r).**

Observation B: **Once h is matched, h never becomes free again.**

Observation C: **h's partners cannot get worse (for h).**

Hint: r must have been rejected a lot – what does that mean?

Claim 1: If r proposed to the last horse on their list, then all the horses are matched. (3)

Hint: r must have been rejected a lot – what does that mean?

Since we immediately match any horse we un-match in the algorithm, once a horse receives any proposal it is not free for the rest of the algorithm. (Observation B).

Since r proposes to horses on its list in order, every horse on r's list must be matched.

And every horse is on r's list! So once a rider proposes to the last horse on their list, all horses are matched.

Claim 2: The algorithm stops after $O(n^2)$ iterations.

Hint: When do we exit the loop? (Use claim 1).

If every horse is matched, every rider must be matched too.

- Because each horse is matched to exactly one rider and there are an equal number of riders and horses.

Since we don't repeat a proposal, and each of the n riders have lists of length n, It takes at most $O(n^2)$ proposals to get to the end of some rider's list.

Claim 2 now follows from Claim 1.

That's the number of iterations. What about time per iteration?

Wrapping up the running time

We need $O(n^2)$ proposals. But how many steps does the full algorithm execute?

Depends on how we implement it…we're going to need some data structures.

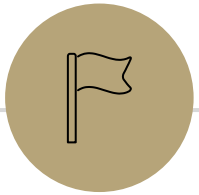
With the right data structures the worst-case running time really is $O(n^2)$. More details in the slides at the end of this deck.

Don’t forget the TODO List

Make sure you’re on Ed! (check your spam folder for an invite, if not there send an email to Robbie).

Go to section tomorrow

Start on HW1



Making GS as efficient as possible

Data Structure Tricks to run Gale-Shapley in $O(n^2)$ time.

Gale-Shapley Algorithm: questions

```
Initially all r in R and h in H are free
while there is a free r
      Let h be highest on r's list that r has not proposed to
      if h is free
          match (r, h)
      else //h is not free
          Let r' be the current match of h.
          if h prefers r to r'
                unmatch (r', h)
                match (r, h)
```

Are each of these operations really $O(1)$?
Assume that you get two `int[][]` with the preferences.

Wrapping up the running time: implementation

We need $O(n^2)$ proposals. But how many steps does the full algorithm execute?

Depends on how we implement it…we're going to need some data structures.

Gale-Shapley Algorithm (pseudocode)

```
Initially all r in R and h in H are free
while there is a free r
      Let h be highest on r's list that r has not proposed to
      if h is free
          match (r, h)
      else //h is not free
          Let r' be the current match of h.
          if h prefers r to r'
                unmatch (r', h)
                match (r, h)
```

Are each of these operations really $O(1)$?
Assume that you get two `int[][]` with the preferences.

Gale-Shapley Algorithm (data structures)

```
Initially all r in R and h in H are free
while there is a free r
    Let h be highest on r's list that r has not proposed to
    if h is free
        match (r, h)
    else //h is not free
        Let r' be the current match of h.
        if h prefers r to r'
            unmatch (r', h)
            match (r, h)
```

Need to maintain free r. What can insert and remove in $O(1)$ time?

Each r should know where it is on its list.

Maintain partial matching

Given two riders, which horse is preferred?

Maintain partial matching

Are each of these operations really $O(1)$?
Assume that you get two `int[][]` with the preferences.
`Horse[i][j]` gives the identity of the j^{th} rider on horse i's list.

What data structures?

Need to maintain free r. What can insert and remove in $O(1)$ time?

Queue, stack, or list (inserting at end) all would be fine.

Maintain partial matching

Two arrays (index `i` has number for partner of agent `i`).

Each r should know where it is on its list.

`int` for each rider (likely store in an array)

Given two riders, which is preferred?

Lookup in `int[][]` takes…$O(n)$ in the worst case. Uh-oh.
Better idea: build "inverse" arrays (given rider, what is their **rank** for horse?).
One time cost of $O(n^2)$ time and space to build, but comparisons $O(1)$.

These aren't the only options – you might decide on an object-based approach (can meet same time bounds up to constant factors)

How do we handle the lookup?

Changing The Question

`Horse[i][j]` asks horse i who their j^{th} favorite rider is.

To ask Horse i, "do you prefer rider 3 or rider 5" we'd have to iterate through `Horse[i][k]` as k goes from 0 to $n-1$, until we see 3 or 5.

It would be better if we could ask "horse i, where is rider 3 on your list?" and "horse i, where is rider 5 on your list?" have it say "2nd on my list" and "8th on my list" to let us say "well $2 < 8$, so you would prefer rider 3."

Let's make a data structure to answer the other question.

Inverse Array

`Inv[i][j]` answers the question "Hey, horse i, what position in your list is rider j?

```
for(int k=0; k<n; k++){
  int riderNum = horse[i][k];
  Inv[i][riderNum]=k;
}
```

`riderNum` is the identity of the rider who is in position k. So when we ask about `riderNum`, the answer should be k.

Inverse Array (construction)

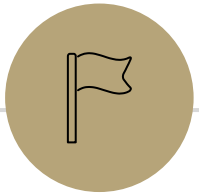
Repeat that code for every i (probably making a 2-D inverse array), and you'll have $O(1)$ access to answer the question "Rider 7, Do you prefer horse 3 or horse 5?

How long does it take to create? $O(n^2)$ time total.

So as a pre-computation step, create the inverse arrays. Then run Gale-Shapley as shown a few slides ago.

Every other step was $O(1)$ time, so…

So the final running time of Gale-Shapley will be $O(n^2)$



Extra Syllabus Stuff

Will cover in the extra video

Textbook

Optional: Algorithm Design by Kleinberg & Tardos

It's a good introduction, and nice as a reference.

There are lots of other books:

Introduction to algorithms by Cormen, Leiserson, Rivest, Stein

One free reference: Algorithms by Jeff Erickson Algorithms.wtf

Logistics – Lecture Activities

I'm going to be teaching with **active learning** in this course.

Why? Because it works.
https://www.pnas.org/content/111/23/8410 a meta-analysis of 225 studies.

Just listening to me isn't as good for you as listening to me then trying problems on your own and with each other.

The answers live help me adjust explanations.

There aren't points associated with completing the activities.

Logistics – where to go?

Slides, homework problems, etc. go up on the webpage

Homework submission on gradescope

Live lecture activities on polleverywhere

Questions on Ed discussion board

Don’t trust canvas – we won’t be updating frequently. We’ll tell you when we’re using it for specific purposes.

Late Policy

A little different from what you're used to!

We're counting late days by ***problems*** instead of by *assignment*.

You can use a "late problem" to turn in a problem up to 24 hours later than otherwise.

That applies to one problem not the other 3 on the assignment

You can use up to 2 per problem (i.e., submission 48 hours after usual deadline)

You have 12 to use.

We'll also drop a few of your lowest scores. Details on the syllabus.

Logistics – Work

8 (approximately) weekly homework assignments

Two midterm exams

In the **evening** of Monday Monday May 4th, Wednesday May 20th.

We'll have alternate exams for people with immovable conflicts, but please put those dates on your calendar now! (details of conflict policy in the syllabus)

One final exam

(as on the official schedule, Monday March 13, 2:30 PM)