CS7800: Advanced Algorithms Fall 2025
Homework 1: Due Friday, September 26, 2025

Jonathan Ullman

Philosophy/Policies

This is a PhD-level core class, and I think of it as having two goals: (1) Ensure an appropriate working knowledge of algorithms and algorithmic reasoning for a PhD computer scientist. (2) Give you a "mental workout" by exposing you to new skills and challenging problems. These assignments and policies are structured to balance these two goals. Towards this end, assignments have two types of problems:

- *Assigned problems* for you to complete and submit by the due date (unless using late days). These will be graded and checked for consistency with the class honesty policy. I will provide full solutions for these problems.
- *Suggested problems* that you do not need to complete, but will be useful for guiding your practice and stretching your ability. You may work on these problems any way you like, with no honesty policy. I can't promise full solutions for all of these problems but am happy to discuss them in office hours.

Unless I indicate otherwise, both assigned and suggested problems are related to the lecture material and are considered "fair game" for exams.

All solutions must be typed in LaTeX! You'll need to learn LaTeXeventually to write papers and there are many resources to get started. I'd suggest starting with the source code for this assignment.

This is a PhD-level course and I trust that you will figure out how to use these assignments to perform well on the exams and get what you need out of the course. Thus, I have kept the honesty policies as lightweight as possible.

- *Collaboration is allowed and encouraged!* To make sure you are contributing, limit collaboration to groups of at most three. Identify your collaborators on your solutions.
- *You're adults and scholars, so act like it!* It would be easy to solve these problems using AI or other sources, but you wouldn't learn anything or exercise your brain, and it would waste my time giving feedback on solutions you didn't write. However, policing AI-based cheating is time consuming at best, and hopeless at worst, so I simply won't bother. I ask that you respect me and yourself enough to do the work yourself.

Assigned Problems (Collected and Graded)

Problem 1 Your local school is planning its annual bake sale to raise funds and needs parents to help staff the event. The event runs from time 0 to time T. The parents were surveyed, and each of n parents provided one specific time window where they can volunteer, with parent i volunteering for the window of s_i until f_i. Since each parent will eat something from the table as compensation, you'd like to use the smallest number of different parents to staff the event. Specifically, find the smallest set $S \subseteq \{1, 2, \dots, n\}$ so that $\bigcup_{i \in S}[s_i, f_i] \supseteq [0, T]$. Design a greedy algorithm that solves this problem. For simplicity, you may assume that all the values s_i and f_i are distinct, and that it is possible to staff the entire event from time 0 to time T. For full credit your algorithm should run in time $O(n \log n)$, but slower algorithms will receive significant partial credit.[1]

Problem 2 The NASA Near Earth Object Program lists potential future Earth impact events that the JPL Sentry System has detected based on currently available observations. Sentry is a highly automated collision monitoring system that continually scans the most current asteroid catalog for possibilities of future impact with Earth over the next 100 years.

This system allows us to predict that i years from now, there will be x_i tons of asteroid material that has near-Earth trajectories. In the mean time, we can build a space laser that can blast asteroids. However, each laser blast will require exajoules of energy, and so there will need to be a recharge period on the order of years between each use of the laser. The longer the recharge period, the stronger the blast—after j years of charging, the laser will have enough power to obliterate d_j tons of asteroid material. You must find the best way to use the laser.

The input to the algorithm consists of the vectors $(x_1, \dots, x_n)$ and $(d_1, \dots, d_n)$ representing the incoming asteroid material in years 1 to n, and the power of the laser d_i if it charges for i years. The output consists of the optimal schedule for firing the laser to obliterate the most material.

Example: Suppose $(x_1, x_2, x_3, x_4) = (1, 10, 10, 1)$ and $(d_1, d_2, d_3, d_4) = (1, 2, 4, 8)$. The best solution is to fire the laser at times 3, 4. This solution blasts a total of 5 tons of asteroids.

2.1 Construct an input on which the following "greedy" algorithm returns the wrong answer:

```
Algorithm 1: BadLaser($x_1, \dots, x_n, d_1, \dots, d_n$)
1  Compute the smallest $j$ such that $d_j \geq x_n$ or set $j = n$ if no such $j$ exists
2  Shoot the laser at time $j$;
3  if $j < n$ then return BadLaser($x_1, \dots, x_{n-j}, d_1, \dots, d_{n-j}$)
```

Intuitively, the algorithm figures out how many years j are needed to blast all the material in the last time slot. It shoots during that last time slot, and then accounts for the j years required to recharge for that last slot, and then recursively applies the same rule for slots 1 through $n - j$.

2.2 Let $\text{OPT}(j)$ be the maximum amount of asteroid we can blast from year 1 to year j. Write a recurrence for $\text{OPT}(j)$. Justifying that your recurrence is correct (a few sentences should suffice).

[1]**Hint:** Trying to come up with an $O(n \log n)$ time algorithm on the first try might cause you to overthink things. First think about what the parents the algorithm should choose and how to prove correctness, then separately think about how to implement those choices in $O(n \log n)$ time.

2.3 Using your recurrence, design an efficient dynamic programming algorithm to output the optimal set of times to fire the laser. You may use either a top-down or bottom-up approach. Your algorithm needs to output the optimal set of times to fire the laser, not just the number $\textsc{opt}(n)$.

2.4 Analyze the running time of your algorithm.

Optional Problems (Not Collected, Not Graded)

These problems will mostly be pointers to problems I like in the two textbooks. Links to these resources are available on Piazza.

Problem 1 Show that there are instances of the stable matching problem with exponentially many distinct stable matchings. Specifically, for every n, construct preferences for n hospitals and n doctors/residents and argue that the number of distinct stable matchings for these preferences is $\Omega(b^n)$ for some constant $b > 1$. For a challenge, try to make b as large as possible.

Problem 2 Kleinberg-Tardos Chapter 4, Exercises 6 (Greedy Algorithms)

Problem 3 Kleinberg-Tardos Chapter 4, Exercises 8 and 9 (Minimum Spanning Trees)

Problem 4 Kleinberg-Tardos Chapter 5, Exercises 1 and 6 (Dynamic Programming)