ECS 122A Lecture 1

— How do I talk abt algs?
(- Partnered HWs (some)
+ Communication

Course overview

- Design and Analysis of Algorithms
  - ↳ Principles + Paradigms
    - Divide & Conquer
    - Dynamic Programming
    - Greedy ☆
    - Data structures
    - NP-hardness
  - ↳ Runtime (~~space~~ quarter too short)
  - ↳ Correctness (Proofs!)

not just list of standard algs (e.g. Dijkstra's)

Why? Most of you won't need proofs in career

- Firm expectation before writing code
- Intuition comes after practice w/ rigour

Warm up
↳ Show of hands, why take the course?

- Required course
- Exploring theory, foundation for grad school
- Job interviews

Many companies still use alg problems

☆ Something to learn for everyone, regardless of your experience

(Lecture capture + no attendance)

Logistics

Course webpage (all material) → homeworks, handwritten notes etc
Piazza (announcements + discussions)

Read course missive in full + see mailing list

Gradescope (HW submissions) + Canvas integration

Grading:

Every component will be curved.
Percentages for reference only.
Grade boundary cases will be considered holistically

HWs (written + coding)	40%
Midterm ~ In class	25%
Final	35%
Bonus: extra credit HW	Max 10%
Bonus: collab Hours	5%

~ HW0 + CA0 out this wk. Graded on completion.

HWs (Roughly weekly. HW vs CA)

HW0 + CA0
HW1
CA1
HW2
CA2
HW3
CA3
HW4

Thanksgiving, more time than 1 week

Written HW: groups of 3
~ HW1 + HW2, random groups
~ HW3+4, can choose
~ Be responsible to your partners

Written problem grades: $\surd^-$ $\surd$ $\surd^+$

LaTeX recommended for convenience but not necessary ← Correctness + Style (Communication clarity & Preciseness + Conciseness)

Late policy: 24 hours in advance (except emergencies / be reasonable)

★ Collab policy (academic honesty):

(Very lax, encourage collab + talk to each other)

Sign full collab policy on HW0

(Can look for answers online / LLM, just cite sources + who you worked with)

- Write everything in your own words
  You MUST understand what you turned in

- Reserve right to meet with you to gauge your understanding of your submission

Leetcode solns/"proofs" are nonsense. LLMs train on that slop, can't write algs proofs correctly

See also slides on Gradescope Coding Assignments

all discussions run as collab hours, plus more

Collab hours (Doing this since 2016)

NOT just office hours

Homework help w/ guidance from TA/tutor/me

Collab w/ other groups

Make friends! ← Learning community, not just a queue to talk to TA for 5 minutes.

↑ ineffective

⚠ Algorithms can be challenging

Some HW problems can feel hard

IT IS OK TO STRUGGLE

Me / Suraj / Tutors all struggled / still do

Come to collab hours to struggle productively!

(Learn the material, otherwise you won't do well in exams)

Do NOT harass TA/tutors

5% bonus: Show up to collab hours between 3 Oct & 10 Oct

Will try to keep lectures conversational

Analysis of algs

Selection sort - 2 implementations
- 2 analyses

Version 1

```
for i = 1 to n
    Find index j ∈ [i..n] minimizing A[j]
    Swap A[i] with A[j]
```

upper bound work in each iter, then sum

Runtime: $O(\sum i) = O(n^2)$

or just $O(n \cdot n) = O(n^2)$

loop i; find min

Correctness:

Very useful for analyzing algorithms

High-level strategy? Induction!

Induction hypothesis:

Loop invariant: At the end of the i^{th} iteration, $A[1..i]$ contains the smallest i elements in sorted order. & A is a permutation of original input

(Complete English sentences for typed work please)

Base case: $i=0$ (start of loop) ✓

Induction step:
Assume statement true for some $i \geq 0$.

Now consider executing $i+1^{st}$ loop.

- Min of A excluding smallest i elements is $i+1$st smallest, (because A is permutation of input)
- Swap step gets it to right pos.
- Rest of $A[1..i]$ didn't change ✓

Case of $i=n$ implies A is sorted at the end of loop ☐

Pseudo code: don't just invent your own dialect of C or Python. Use English if useful for communication.

Version 2

```
function sort(A)                    if size(A) == 1
                                        return A
                                    else
    Find arg min_j A[j]
    B := A ; B[j] = A[1]
    Return A[j] :: sort(B[2..size(A)])
```

(Induct on thing that is changing)

Runtime recurrence:

$T(n) \leq c_1 \cdot n + T(n-1)$

$T(1) \leq 1$ where $c_1 \geq 1$

Induction on n that

$T(n) \leq c_2 \cdot n^2$ if c_2 is suff large.

Base case: $c_2 \geq 1$ would work

Induction step:

$T(n) \leq c_1 \cdot n + c_2 \cdot (n-1)^2$

$= c_2 \cdot n^2 - 2 \cdot c_2 n + c_1 \cdot n + c_2$

$\leq c_2 \cdot n^2$ if $c_2 \geq c_1$ (and $n \geq 1$)

Induction on n that

$T(n) \leq c_1 \cdot n^2$

Base case: $c_1 \geq 1$ by assumption

Induction step:

$T(n) \leq c_1 \cdot n + c_1/2 \cdot (n-1)^2$

$= c_1 \cdot n^2 - 2 \cdot c_1 n + c_1 \cdot n + c_2$

$\leq c_1 \cdot n^2$

Correctness:

Induction again!

Hypothesis: sort returns a sorted version of its input array of size n

Base case ✓

Induction step:

Assume statement for some $n \geq 1$.

Now for an array of size $n+1$

$B[2..n+1]$ is a permutation of the n largest elements of A.

and $A[j]$ is the smallest.

So $A[j] = \text{Sort}(B[2..n+1])$ is the sorted permutation of A □

So what have we learnt?

- What variable are we inducting on?
  ~ Something that changes during computation

  Loops: loop index

  Recursion: input size usually

- Proof structure should match code structure.

- Precision: induction hypothesis didn't use a vague word like "correct"

  spelled out what it means in each context