Lecture 1: Introduction and Proofs of Correctness

Sam McCauley

February 5, 2025

Welcome!

- I’m Sam
    - Call me Sam :); he pronouns

- This is algorithms (CS 256)

- Dialogue is encouraged! Please let me know if you have questions or comments.

What is This Course?

Day to day of Algorithms

- No coding in this class
- Focus is on high-level strategies (a.k.a. algorithms)
- English descriptions, proofs, short answers/counterexamples

Question for the class: Why are *you* taking the course?

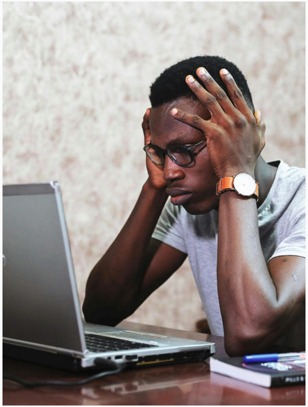
Two Broad Questions about Algorithms

- Correctness: does this algorithm work?

- Running time: how fast is this algorithm?

What We'll Look at in This Class

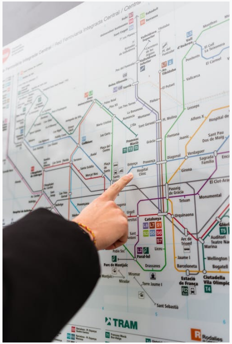
1. Given a piece of code, or high-level strategy, does it work?
2. Does it *always* work?
3. Or: what does it do?
4. Is it fast?
5. If we move to another domain, will it still be fast?



Why Algorithms?

- It’s a different way of thinking about computer science
- Some of you may use it a lot
- All of you (in my opinion) will benefit from having seen it

Proofs



- You and I will largely communicate via proofs
- Proofs: structure on top of intuition
- Remove *ambiguity*
- Strengthens intuition

How I View This Course

Algorithms is (kind of) a *writing* course

Specifically:

- You can't just (say) run your program to know if it's correct
- There are often multiple right answers
- An answer may get full points, but may still have room for significant improvements in clarity
- Best way to get better is to practice!

Takeaways from this

- We'll spend lots of time in class practicing how to *evaluate* proofs
  - Goal: give you the tools to know going in if a proof is correct or not
- It's OK to struggle with some topics!
  - Come to office hours!
  - You can have strong performance in the class without feeling like an expert

Course Resources and Overview

Tools We'll Use

- Course website
- Overleaf/latex
- Gradescope

What are LLMs (ChatGPT, etc) good at?

What are they OK at?

What are they bad at?

LLMs (my opinion)

What are some things LLMs are good at?

- Writing effectively
- Using APIs
- Interpreting “human” input
- Regurgitating very well-known concepts (and code); simple refactoring

LLMs

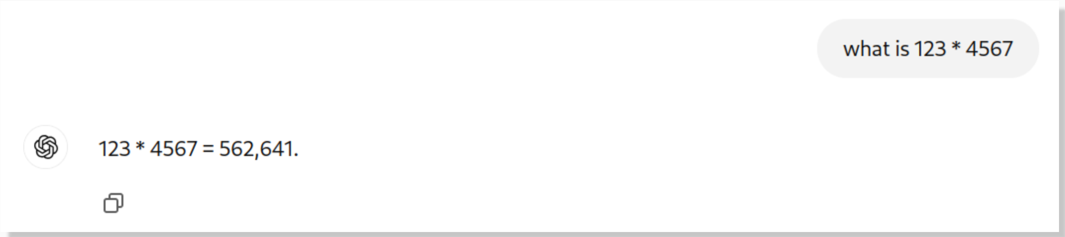
What are some things LLMs are OK at?

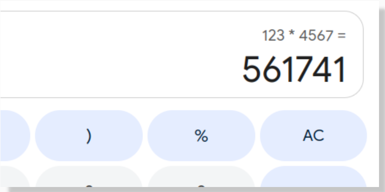
- Giving answers to factual questions
  - The answers are usually phrased quite well
  - Oftentimes it’s right. But reasonably often it’s not. (Does not affect its confidence)
  - Can be a good tool if these caveats are kept in mind
- Along similar lines: giving proofs/explanations/pseudocode of well-known algorithms
- Fingers

LLMs

What are some things LLMs are bad at?

- Computation and calculation





LLMs

What are some things LLMs are bad at?

- Computation and calculation
- Algorithms and proofs
  - It will look *very* confident
  - The answers are *very* wrong
  - We’ll use this in class sometimes

Questions about course resources?

Plan for Rest of Today

- Intro/review: reading pseudocode, expectations for proofs, etc.
- Use some likely-familiar algorithms as examples
  - And some algorithms that, probably, none of you have seen before
- Goal: Good foundation to get you started
- On Monday we'll move to the "Stable Matching" problem

Pseudocode

Pseudocode

- We will give algorithms in two ways in this course:
  - English descriptions, and
  - Pseudocode
- Code is a way for humans to *unambiguously* give computers instructions
- Pseudocode is a way for humans to communicate with each other
  - Keeps the structure of code
  - Does not rely on language-specific knowledge
- You will not need to write pseudocode in this course

Writing Pseudocode

- *Looks* very much like simple Python
- Basic keywords: `if`, `else`, `while`, etc.
- Basic arithmetic operations `+ - * / %`, use superscripts for exponents, write log
- Assume `0`-indexed arrays, inclusive `for` loops
- Explain any non-trivial steps in English
- Idea: make it as clear as possible!

Pseudocode Example 1

```
function findElement(A):
    minSoFar = A[0]
    for i = 1 to n-1:
        if A[i] < minSoFar:
            minSoFar = A[i]  # we found a new smallest
    return minSoFar
```

Pseudocode Example 2

It's OK to use sets in pseudocode. Instead of library functions, write in English (if unambiguous!).

```
function findEven(A):
    B = ∅
    for x ∈ A:
        if x % 2 == 0:
            B = B ∪ {x}.
    Sort B using Merge Sort // O(n log n) time
    return B
```

(Recall:) Two Questions about Algorithms

- Correctness: does this algorithm work?

- Running time: how fast is this algorithm?

Let's start with (a very simple example of) correctness!

Algorithm Correctness

Correctness today

- We'll prove, in detail, that some algorithms are correct
- Some (but not all) review
- Correctness *can* be obvious, and is often omitted
  - For practice, we'll start with some cases where correctness is not so interesting. The focus will be on transfer from discrete math to CS
  - We'll talk about how short English explanations can be an effective alternative to formal proofs
  - We'll soon get to some non-obvious proofs

Algorithmic Invariants

Definition (Invariant)

If we stop an algorithm in the middle of its execution, what can we guarantee about its state?

- Heart of all algorithms
- When looking at an algorithm for the first time, ask yourself what invariants it satisfies
- Loops are often key. What is the code doing each time a loop runs from top to bottom?
- A proof by induction is a formal way of analyzing an invariant

Example 0: Finding Maximum

```
indexOfLargest = 0
for j = 1 to i:
    if A[j] > A[indexOfLargest]
        indexOfLargest = j
```

- What does this code do?
- *Intuitively*, in 1-2 sentences, why?
- What Invariant does it satisfy? (What happens each time the `for` loop runs?)
    - One answer: after k iterations, `indexOfLargest` contains the index of the largest element in $A[0] \ldots A[k]$.
- In pairs: how can we formalize this with an inductive proof? (What are the pieces of an inductive proof?)

Example 0: Finding Maximum

```
indexOfLargest = 0
for j = 1 to i:
    if A[j] > A[indexOfLargest]
        indexOfLargest = j
```

Proof.

I.H.: After k iterations, `indexOfLargest` contains the index of the largest element in $A[0] \dots A[k]$.

Base case: after 0 iterations, `indexOfLargest` is 0; $A[0]$ is the largest element in $A[0] \dots A[0]$.

Inductive Step: (contd. next slide) □

Example 0: Finding Maximum

```
findMax(A, i):
indexOfLargest = 0
for j = 1 to i:
    if A[j] > A[indexOfLargest]
        indexOfLargest = j
```

Proof.

I.H.: After k iterations, `indexOfLargest` contains the index of the largest element in $A[0] \ldots A[k]$.

Induc. Step: Assume I.H. is true for some k.

During the $k+1$st iteration, if $A[k+1] > A[\texttt{indexOfLargest}]$, then by the I.H. $A[k+1]$ is the largest element in $A[0] \ldots A[k+1]$. After the $k+1$st iteration, $\texttt{indexOfLargest} = k+1$, and the I.H. is true for $k+1$.

Otherwise, `indexOfLargest` remains the same, and the I.H. is true for $k+1$ since $A[\texttt{indexOfLargest}]$ is the largest element in $A[0] \ldots A[k+1]$. $\square$

Example 1: Selection Sort

```
selectionSort(A):
    for i = |A|-1 to 0:
        indexOfLargest = 0
        for j = 1 to i:
            if A[j] > A[indexOfLargest]
                indexOfLargest = j
        swap(A, i, indexOfLargest)

swap(A, i, j): // swaps A[i] and A[j]
    temp = A[i]
    A[i] = A[j]
    A[j] = temp
```

- What does the inner loop of selection sort do?
- *Intuitively*, in 1-2 sentences, why is this algorithm correct?

Proofs in CS 256

- Proofs are a language for you to communicate with me
- Level of detail: judgment call
- Rule of thumb: imagine you're explaining to a skeptical classmate
  - They are trying to understand you; are willing to fill in details
  - But they are always asking questions
- Skeptical rubber duck explanation

Example 2: Insertion Sort

```
insertionSort(A):
    for i = 0 to |A| - 1:
        j = i
        while j > 0 and A[j-1] > A[j]:
            swap(A[j-1], A[j]) # swaps A[j-1] and A[j]
            j = j - 1
```

- What invariant can we guarantee after the outer loop executes *i* times?

- *Intuitively*, in 1-2 sentences, why is this algorithm correct?

Insertion Sort 2-sentence Explanation of Correctness

- Invariant: after k iterations of the outer loop, items in $A[0]$ through $A[k]$ are in sorted order.
- So after $n-1$ iterations, $A[0]$ through $A[n-1]$ are in sorted order—the array is sorted!
- This invariant maintained because on the $k+1$st iteration of the outer loop, the inner loop swaps $A[k+1]$ with each larger element among the first k elements.
- How could we turn this into a proof by induction?

Power of Invariants

- Can help figure out why algorithms work
- Or *don't* work! Great for bug finding
- *No universal rule* for finding invariants. Some tips:
  - Try small examples, see what happens
  - What are we trying to solve? What kind of partial work is helpful?
  - What internal state would make the algorithm *wrong*? Can this happen?

Correctness in CS 256

- I will frequently ask you to *explain* correctness
- I will occasionally ask you to *prove* correctness
- Level of detail is a judgment call.
  - We'll practice

Skeptical rubber duck

Questions about Correctness?