CSCI 3100: Algorithms

Today:
- Big-O
- Algorithm analysis
- Recursion

**Algorithms by Complexity**

MORE COMPLEX

LETPAD  QUICKSORT  GIT  MERGE  SELF-DRIVING CAR  GOOGLE SEARCH  BACKEND  SPRAWLING EXCEL SPREADSHEET BUILT UP OVER 20 YEARS BY A CHURCH GROUP IN NEBRASKA TO COORDINATE THEIR SCHEDULING
3 parts to every algorithm:

1. 

2. 

3. 

+ sometimes 4:
This week: why you should have paid attention in discrete math & data structures!

Topics to recall:
Runtimes:
What is big-O analysis?

Why use it?
Formal def:

Let $f$ and $g$ be functions $\mathbb{R} \rightarrow \mathbb{R}$ (or $\mathbb{Z} \rightarrow \mathbb{R}$). We say that:

$f(n) = O(g(n))$ if there exist constants $C$ and $n_0$ such that:

$|f(n)| \leq C|g(n)|$ for all $n > n_0$.
Big-O: functions ranking

**BETTER**
- $O(1)$: constant time
- $O(\log n)$: log time
- $O(n)$: linear time
- $O(n \log n)$: log linear time
- $O(n^2)$: quadratic time
- $O(n^3)$: cubic time
- $O(2^n)$: exponential time

**WORSE**

![Big-O Complexity](image)

The graph shows the complexity of different functions as the number of elements increases, with $O(1)$ being the most efficient and $O(2^n)$ the least efficient.
Example proof:

\[ f(x) = x^2 + 2x + 1 \text{ is } O(x^2) \]

pf:
Key thm:

Let \( f(x) \) be a polynomial of degree \( n \), so
\[
\sum_{i=0}^{n} a_i x^i
\]
where each \( a_i \in \mathbb{R} \).

Then \( f(x) = O(x^n) \).

pf sketch:
Induction: recursion's twin

A method of proving a statement which depends on the statement being true for smaller values.

Required pieces:
Aside: I think of this as "automating" a proof:

Show true for \( n=1 \).

Show if \( n \) holds, then \( n+1 \) must also.

\( \Rightarrow \) Get all \( n \) for free!
Example: \[ \sum_{i=0}^{n} i = \]
Example: The gossip problem

- There are \( n \) people, each of whom knows a unique secret.
- Every time 2 of them talk, they share every secret they know.

Q: How many phone calls are necessary before everyone knows all the secrets?
Thm: For $n \geq 4$, $2n-4$ calls are enough.

Pf:
Now: Recursion

- Induction started at the bottom and builds up.

Recursion: the natural dual idea:
Recurrence relations:

\[ H(n) = 2H(n-1) + 1 \]

\[ M(n) = 2M\left(\frac{n}{2}\right) + n \]

\[ T(n) = T\left(\frac{3n}{4}\right) + n \]

How to solve?
Recursive algorithms:
Based on reduction:
Reduce to a smaller instance of the same problem.

Necessary pieces (like induction):
Classical example: Towers of Hanoi

The Tower of Hanoi puzzle
Strategy: think recursively!

Start small:

[Diagram of a simple structure]
**Bigger picture:**

**HANOI(n, src, dst, tmp):**
if $n > 0$

- HANOI($n - 1$, $src$, $tmp$, $dst$)
- move disk $n$ from $src$ to $dst$
- HANOI($n - 1$, $tmp$, $dst$, $src$)

The Tower of Hanoi algorithm; ignore everything but the bottom disk
Proof of correctness:
Runtime:
Next time:

- Merge sort
- Master them
- Other classical algorithms