Final announcements

- HW due
- Review session Monday - bring questions!
- Final: Wed. at 8am, here
- Keep an eye on blackboard/git for more grading
- Today: have to sub 1-2pm in my office at 2pm
- Request: instructor evals!
  (You'll have time at end)
- Tuesday: In, but likely down in Linux classroom.
Data Structures we've seen:

- Stacks + queues
- Simple (singly linked) list
- Vectors
- Lists
- Trees:
  - General
  - Heaps, BST, AVLs, Huffman
  - Hashing
  - Graphs
  - Treaps

Also:

- C++ - tons! (low-level)
- Sorting / searching
Trade-offs:

Simple & limited:
- stacks \[ \mathcal{O}(1) \]
- queues \[ \mathcal{O}(1) \]
- even priority queues (heaps)
- hashing

Why use?

Fast!
Overhead savings.
“Full-featured”:
- Vectors
- Lists
- Trees

Trade-offs are key!

Consider:
- your data
- how you'll use it
Practical vs theoretical:

Some have poor theoretical guarantees, but are amazing in practice.

- hashing
- quicksort
- even inserting in a vector (well - push_back)
Some data:

Insertions done in order:

1, 2, 3, ..., n

![Graph showing performance of different data structures](image)

- Linear search: $O(n^2)$
- Vectors
- Hashing

As $n$ grows, the performance of linear search degrades drastically.
Reverse order inserts

\[ n, n-1, n-2, \ldots, 1 \]

\( t \leftarrow \text{gcd} Y f \Rightarrow \text{raw desks} \)
Random inserts:

Fix $n$ and insert 1-$n$ in random order.

Lists + Vectors

AVL

ABST

Hashing
Take away:

- Hashing - wow! Caveat: limited, don’t implement yourself!
- Also - your data does matter.
- Performance varies drastically.
- These are “asymptotic”, but remember that constant factors can still be meaningful.
Now:

Thanks for a lovely (if busy!) semester!
I hope to see you all around next year.

Questions: about the final, or next week?

(+ finally - evaluations!!)
(you have time!)