CS 180 - Recap of our semester

Announcements
- HW is due
- Review session: Friday at 4pm in Linux lab (check webpage Thursday in case location changes)
- Final Monday at noon
- Office hours: Wed. morning, Friday morning
Data Structures Covered

- Stacks
- Queue
- Vectors
- Lists
- Heaps
- Tree Structures
- BST
- AVL Trees
- Huffman Trees
- Heaps
- Hashing
- Graphs
- Searching
- Sorting
Data Structures

Some data structures have limited functionality, but as a result are extremely efficient.

Ex. - stack + queue

- graph representing (space vs speed)
"Full-featured" data structures

More versatile data structures have trade-offs: to get something faster, you sacrifice speed in another area.

Ex:

- vectors
  - `O(n)`-operator `[]` ➞ `O(n)`
  - `O(n)`-insert ➞ `O(1)`

- AVL trees: `O(log n)`
Randomized or Expected

Some work well in practice, but have no theoretical guarantees.

Ex:

- hashing
- treap - random priority
- quicksort
So which is best?

Ans: Depends.
In-order insertions: \(1, 2, 3, \ldots, n\)
Reverse Order inserts: \( n, n-1, n-2, \ldots, 1 \)

- Inserting at beginning - \( O(n) \) each time

Lookup + Insert: \( O(1) \)
Random Inserts

Both list & vectors aren't good

(height of tree)

Hashing!
Note: hashing does lack extra functionality so not always the right choice
Questions: insert \((q, 2)\)

Heap: unique!