List Problem

- design a data structure that supports:
  - add to end
  - add at index
  - remove by value
  - get at index
- and make it efficient (time, space, energy, ...)

Set Problem (no ordering/indexing no dups)

- design D.S. that supports
  - add an elt
  - remove
  - contains
- make it efficient

ideas
- array O(n) O(n) O(n)
- binary tree
- array list O(1) O(n) O(n)
- linked list O(1) O(n) O(n)

How to organize a set so that contains is fast?

- You get a bunch of ints in advance
  - organize however you want
- then, you will get a bunch of contains calls
  - answer as fast as possible

- ordered binary tree (BST)
  - balanced
  - O(log n) time for contains

- hash table: map elements to indices in an array

- sort it, then binary search
  - O(log n)
- make big array (horrible in space)
  - O(1) time

Hash tables

- map elements to indices in an array (hash function)
- answer contains by looking there

- design the hash function
- handling collisions

We have looked at solutions

- arrays
- ArrayList
- linked lists

] problem: list problem

- binary trees + BSTs] ?

List Problem

Design a D.S. that supports

- add (at an index)
- remove value
- get at index

(Make it efficient (time, space, energy...))

Set Problem

- no duplicates
- order doesn't matter

Design a P.S.

- add element (not at an index)
- equality
- contains
- size
- remove element

(make it efficient)

Set Solutions

- TreeSet, HashSet, ListSet???
- list: array ArrayList LinkedList
  - contains: loop through the list
    O(n)
  - add: O(1)
  - remove: O(n)

Give you n numbers

- you can organize them

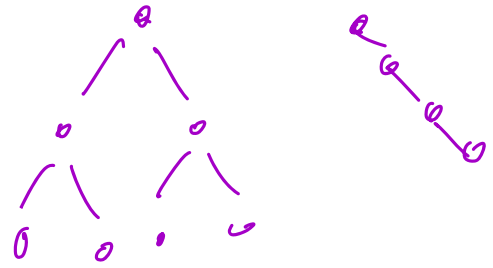
Then I will ask contains questions

- answer quickly

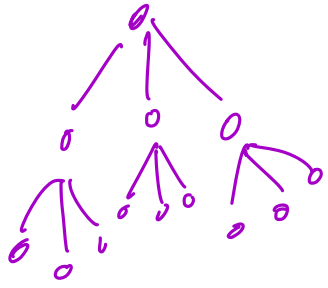
- Binary search tree

  contains O(log n)

  (balance it)



- ternary search tree



- make a huge array of booleans

  of length 4 billion

  - index into array O(1)

- sort the data contains: O(log n)