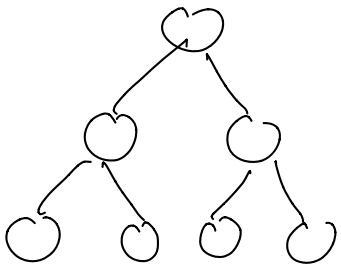
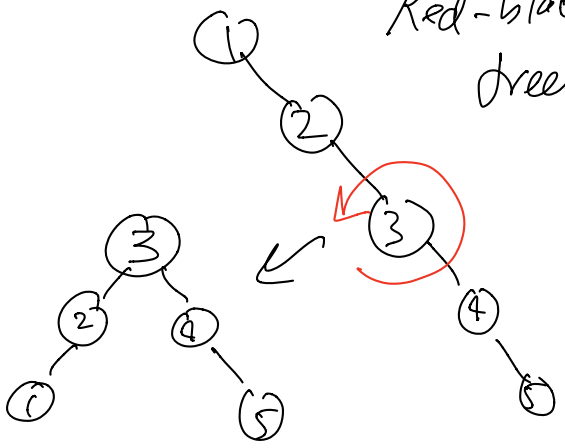
Finish BSTs, start hashing

Hashing/Hash table		Avg	Worst
BST	— add/insert/put	— O(log n)	O(n)
	— search/contains	— O(log n)	O(n)
	— remove/delete	— O(log n)	O(n)





Red-black trees

Hashing/Hashtables

Suppose I put 2^{100} items into a BST.

(Assume we have a balanced BST)

How many steps will it take to add/search/delete?

↳ 100

2^{100} > # of atoms in the universe

Can we do better? (than log n)?

Yes! (w/ caveats)

Purpose of hashtable — reduce add/search/remove → O(1)

How is this possible?

Central idea ⇒ hash function

↳ takes a piece of info you're trying to store

↳ returns an integer

↓

serves as a lookup index into an array.

hash(Plumbing) → 742

hash(Attorney) → 36

Hash table

hash functions — hashCode (Java)
hash (C++/Python)
h

Hashtable — data structure — used to implement Maps/Sets.

↳ under the hood ⇒ an arrayList.

↳ This will store our key-value pairs (MAP) // our items for a set

- has to be an arrayList b/c we need random access.

↳ jump to any index in the array w/out traversing

- Unlike a regular arrayList, we will not let the user control where items go.
- we (programmers) are going to determine which index will hold each item.

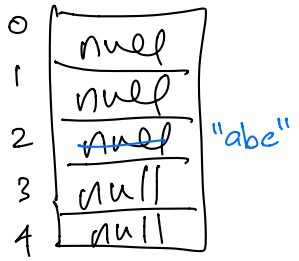
Rat wants to keep track of who can eat in the dining hall

(1) Suppose the plan is all you can eat (all the time)

↳ Set (most appropriate)

(all we care about is whether or not a student has bought a meal plan)

- In a hashtable, the array list always starts empty (full of null)



② Suppose we need to keep track of swipes.

↳ Map<R#, swipes#>
Key Value

- You use the hash function to determine the index where a new item should go.

- Example - Hashtable of strings

hashCode("abc") → 2

Get something out - hashCode("abc") → 2

"Collision" - where 2 items in the hashtable hash to the same index. hashCode("def") → 2