Maps [most useful DS ever, after lists]

* aka, dictionary (hashtable, but that’s misnomer) symbol table, associative array
* A map is maintains a collection of association between pairs of data.
  + in particular, it allows us to save a piece of information (a value) that we can later search for by specifying a KEY.
  + the goal of a map is to let us search through all these pairs very quickly, given a key, and return the value that is associated with that key.
* aka dictionaries because it’s a perfect analogy for keys and values.
  + When we want to look up the definition of a word (a value), we look up the word itself (a key). In particular, dictionaries (paper ones) are usually sorted to allow us to look up the keys and their corresponding values quickly.
* Called associative array because an array allows us to look up any element in an array given its index. [draw example using R# & major] So a map extends this metaphor to let the index be any kind of data, not just integers.
* Typical places we see maps:
  + Rhodes college R# -> major
  + paper dictionary: word -> definition
  + book index: term -> relevant pages
  + symbol table: variable -> data type, address in memory
  + bank: account number -> balance
  + airline: flight number -> start city, ending city
* Primary purpose of a map is to associate a key with a corresponding value. These are sometimes called (key, value) pairs.
* The two major operations we do on symbol tables are PUT, and GET
  + PUT(key, value) [adds a new key-value pair into the map] sometimes called ADD, INSERT
  + GET(key) [returns value] [retrieves a value, given a key] sometimes called GET, LOOKUP, FIND, SEARCH
* Other operations we usually have: DELETE(key)/REMOVE, SIZE, CONTAINS
* One of the largest differences between lists/arrays and maps is that usually maps do not have to have any ordering of the key-value pairs, or at least the order is not controlled by the user.
  + For lists and arrays, each item in the list always has an element to its left, and an element to its right. We always know what it means when we say get me the first element in the list or the last element.
  + For maps, this isn't (necessarily) true.
  + From the user's perspective, usually maps have no order, and a user certainly doesn't care about if a specific k-v pair is stored at location 8 or 80.
  + In fact, usually the programmer will pick where these k-v pairs are stored to make searching this data structure as fast as possible (even faster than linear time, as we'll see in the next few classes).
* Let’s look at a few implementations.
  + Suppose we have a Map implemented with an ARRAY.
  + to make this more concrete, say we are implementing a map that stores info for a bank. account # (int) -> balance (double).  
    KVPair { int, double }
  + ArrayList<KVPair> kvList
  + put = O(n) [for expanding]
  + get = O(n) [must walk the array]
  + remove = O(n) [for shifting]
  + SORTED ARRAY
  + put = O(n)
  + get = O(log n)  
    remove = O(n)
  + UNSORT LINK LIST
  + put = O(1)
  + get = O(n)
  + remove = O(n) [b/c search time]
  + SORTED LL
  + put = O(n)
  + get = O(n)
  + remove = O(n)
* All of these have at least two O(n) operations. What if we could do all of these in O(log n?)
* Sets
* Conceptually very similar to maps, except they only store keys, not values.
* So the purpose of a map is to let you look up a value given a key.  
  + Where as a set is used when all you care about is whether the key is present or not, since there is no value.
* Example contrasting set/map:
  + So when you go to the Rat, and you swipe your card, what do you think is happening behind the scenes? Let's assume that they're taking your R# from your card and accessing how many swipes you have left.
  + So there's a map R# -> # of swipes remaining
  + But suppose a college uses a different meal plan model (probably financially infeasible) where once you sign up for the meal plan, you can visit the Rat as often as you want, whenever you want. In that case, we don't need to store the number of swipes remaining; all we care about is whether you're on the meal plan or not.
  + In that case, we don't need a map, because there's no concept of # of swipes remaining. Instead, we just need to know if you're on the meal plan or not, which is a binary decision.
  + A set makes more sense here
* The two major operations we do on sets are ADD, and CONTAINS
  + ADD(key) [adds a new item into the set] sometimes called ADD, INSERT
  + CONTAINS(key) [returns t/f] [given a key] sometimes called, LOOKUP, FIND, SEARCH
  + Other operations we usually have: DELETE(key)/REMOVE, SIZE, CONTAINS
* Any data structure used to represent a MAP can also be used for a SET, just by taking away the "value" portion, leaving only the keys.
* Arraylist, sorted arraylist, LL, sorted LL.