Maps

What does a map consist of?

- Keys
  - Generic
- Values
  - Generic

Operations:

- get(Object)
  - returns the value for the key or null if not in map
- put(K, V)
  - returns last value for the key or null if it did not previously exist in the map
- containsKey(Object)
- remove(Object)
  - returns the value for the key or null if not in map
- containsValue(Object)
- clear
- size

Example Map

Key	Value
Earth	planet
Ganymede	moon
Venus	planet
Sirius	star
Andromeda	galaxy
Aldebaran	star
Pleiades	star cluster
Pluto	dwarf planet
Europa	moon
Ceres	dwarf planet
M67	star cluster

Different Views

- entrySet
  - Why is this a set?
    - unique elements
- keySet
  - Why is this a set?
    - unique elements
- values
  - Why is this a collection?
    - not unique
    - would not have duplicates if it was a set

What does an entrySet of our map look like?
{<Earth, planet>, <Ganymede, moon>, <Venus, planet>, <Sirius, star>, <Andromeda, galaxy>, <Aldebaran, star>, <Pleiades, star cluster>, <Pluto, dwarf planet>, <Europa, moon>, <Ceres, dwarf planet>, <M67, star cluster>}

What operations do we have for our entrySet?

- size
- remove(Object) -> Entry<K,V>
  - returns a boolean
- contains(Object)
- clear
- iterator

What does the keySet of our map look like?
{Earth, Ganymede, Venus, Sirius, Andromeda, Aldebaran, Pleiades, Pluto, Europa, Ceres, M67}

What does the values collection of our map look like?
[planet, moon, planet, star, galaxy, star, star cluster, dwarf planet, moon, dwarf planet, star cluster]

Do we have to implement keySet and values? Why?

- No! The implementation uses entrySet, so we only need to implement the entrySet and the default implementation of these will work!

Threading

- Nodes of BST are connected in-order for use by the iterator
- Traverse this just like a singly linked list
- Dummy node at the beginning of the list
- Adding and removing are very similar to adding and removing to a regular BST, except now you need to make sure you take care of the “next” pointers

Binary Search

What is the time complexity of searching for an element in an array? Why?

- O(n)
- We need to iterate through the array, one at a time, until we find the element

What is the time complexity of searching for an element in a BST? Why?

- O(logn)
- Every time we go left or right we cut our search space approximately in half
  - There is no point in searching for an element in both left and right subtrees

How can we improve on searching in an array by using techniques of a BST?

- Do something similar--always cut our search space in half while searching
- Array must be sorted!
- Start with a lo (inclusive) index and a hi (exclusive) index
- Compare the element at the midpoint index with the thing we are searching for
- If the thing we are searching for is smaller than the element at the midpoint, we know the element must be to the left, so we update hi
- If the thing we are searching for is bigger than the element at the midpoint, we know it must be on the right, so we update lo
- If the thing we are searching for is equivalent to the element at the midpoint, we can return the midpoint index