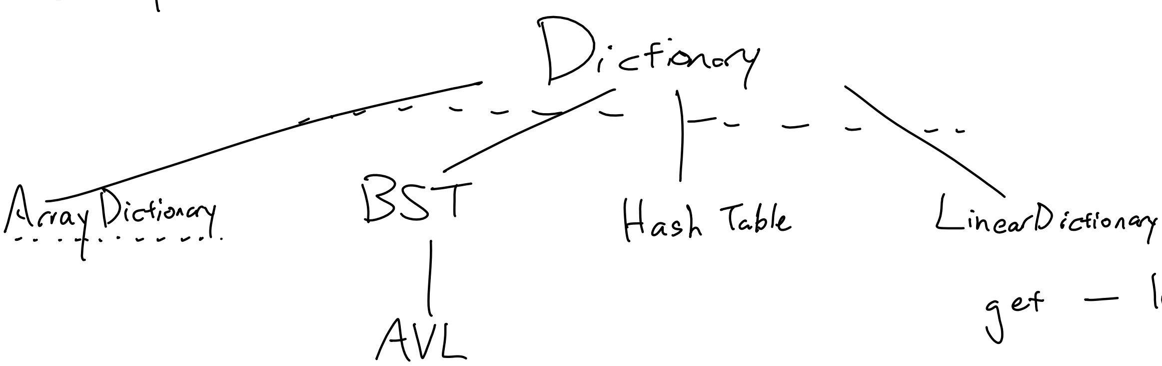
Dictionary operations

- get
- insert
- update
- remove

$O(\log n)$	amortized average $O(1)$	$O(n)$
Balanced BSTs?	Hashtable	

Dictionary is an ADT

Implementations?



get — linear search through list $O(n)$

insert — linear ~~search~~ just to make sure key is not present $O(n)$

Hash Table:

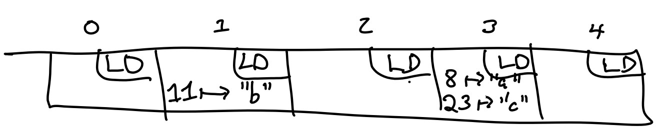
```
Method insert(K key, V value):
    ensureCapacity()
    index ← hash(key) mod this→capacity
    this→array[index].insert(key, value)
End Method
```

Hash table terms

- size: # of key/value pairs
- capacity: length of array
- collision: multiple key/value pairs have keys which hash to same index

Collision resolutions

- Linear probing
- Forward chaining



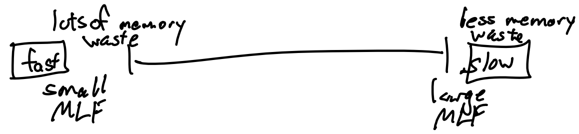
insert(8, "a")
insert(11, "b")
insert(23, "c")

Suppose I have # of buckets B (B = capacity) and a hashtable of size n. On average, how many keys are in each bucket? n/B

Want: # of buckets proportional to n.

Hash table

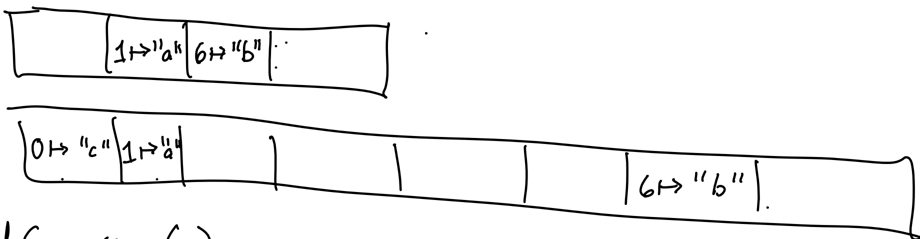
size – # of k/v pairs
array – buckets (LinearDictionary)
capacity – size of array
load factor – $\frac{\text{size}}{\text{capacity}}$



max load factor (MLF): when LF > MLF, increase capacity

Strategy: hash & mod key to index; put k/v pair in LD matching that index

New Problem: for any constant # of buckets, each bucket has O(n) things in it
Old Solution: have # of buckets O(n) by growing table whenever it gets too full; i.e. when LF > MLF, double capacity



Method expandCapacity ( )
  array ← new array of size 2*cap
  oldarray ← this→array
  this→array ← array
  this→capacity ← this→capacity * 2
  for each k/v pair in oldarray:
    index ← hash(k) % this→capacity
    this→array[index].insert(k, v)
EndMethod

be careful about hashes!

insert: * ensure cap
  * hash key to index
  * insert into LD

amortized O(1)
O(1)
average O(1)

amortized average O(1)

amortized 

"average"
ooooo → O(1)
ooooo → O(1)
ooooo → O(n)?
⋮ ⋮
mean → O(1)