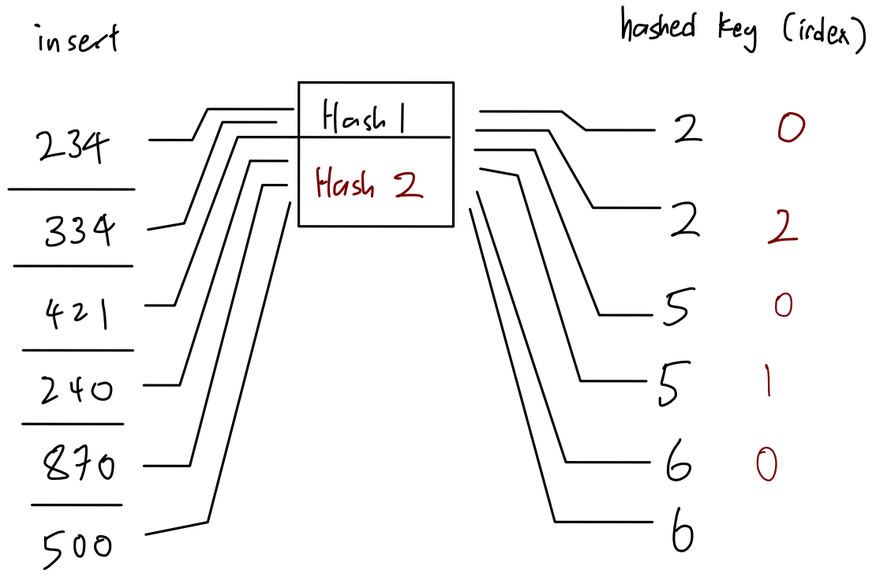
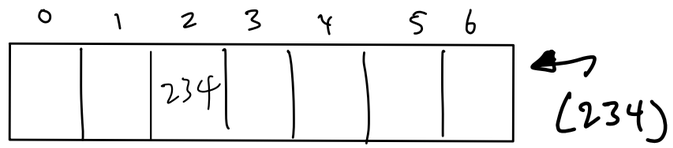
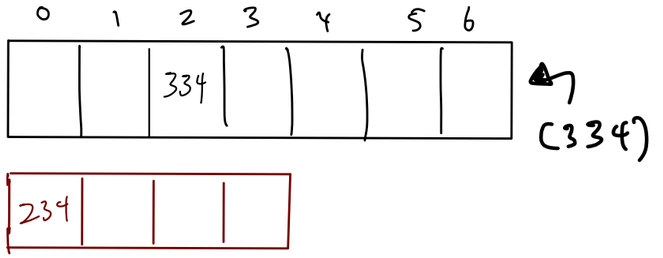
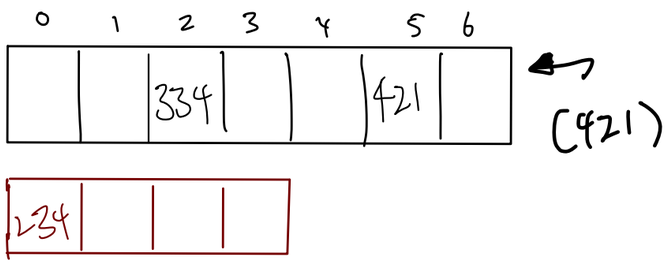
Hash Tables

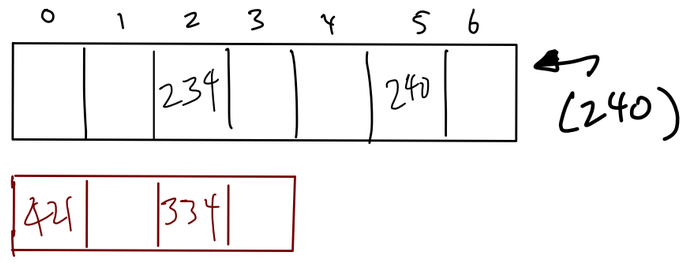
Cuckoo Hashing

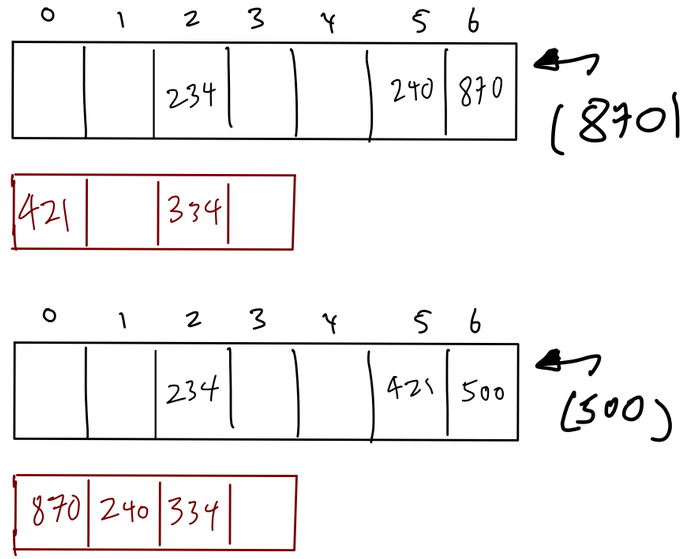












Hash Functions

value % m → m = size of hashtable

- ensures that hashed index is within the table range.

Good Hash functions?

★ More @ tutorial ★

Bad hash function

(Sum of digits) % 7

key

sum(30666910) % 7

31 % 7 = (1)

sum(29378011) % 7

31 % 7 = (1)

} same hashed index

Best hashing method is using both positional & formula hash

(sum(digits) − id[0]) * id[1] % m

sum(digits) → formula

id[0], id[1] → positional

For a given hash table of size m = 13, and a hash function of $total(ord(s[i \dots n]))mod(m)$, and a secondary hash table of size m = 7 and a hash function of $total(ord(s[i \dots n]))\ mod(m)$, compute all hashes of values {“aaxxaa”, “rxkmoo”, “acvxaa”, “bbuxba”, “bxknzg”, “wtfman”, “bbyyaa”, “aykmgz”}.

Hashed index, not hashtable

	0	1	2	3	4	5	6	7
Hash1	4	9	4	4	10	3	8	10
Hash 2	5	0	5	5				

Cuckoo hash all the elements & show each iteration.

Not shown, but element 0, 2, 3 causes infinite loop / cuckoo hash

What is wrong with this hash function, or hash table?

More than 2 elements share the same index for Hashtable 1 & Hashtable 2.

AVL Trees

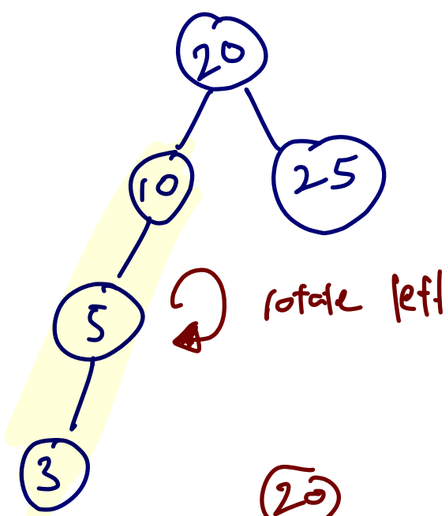
- Basically a balanced binary tree

Balance factor = { height of right subtree − height of left subtree }

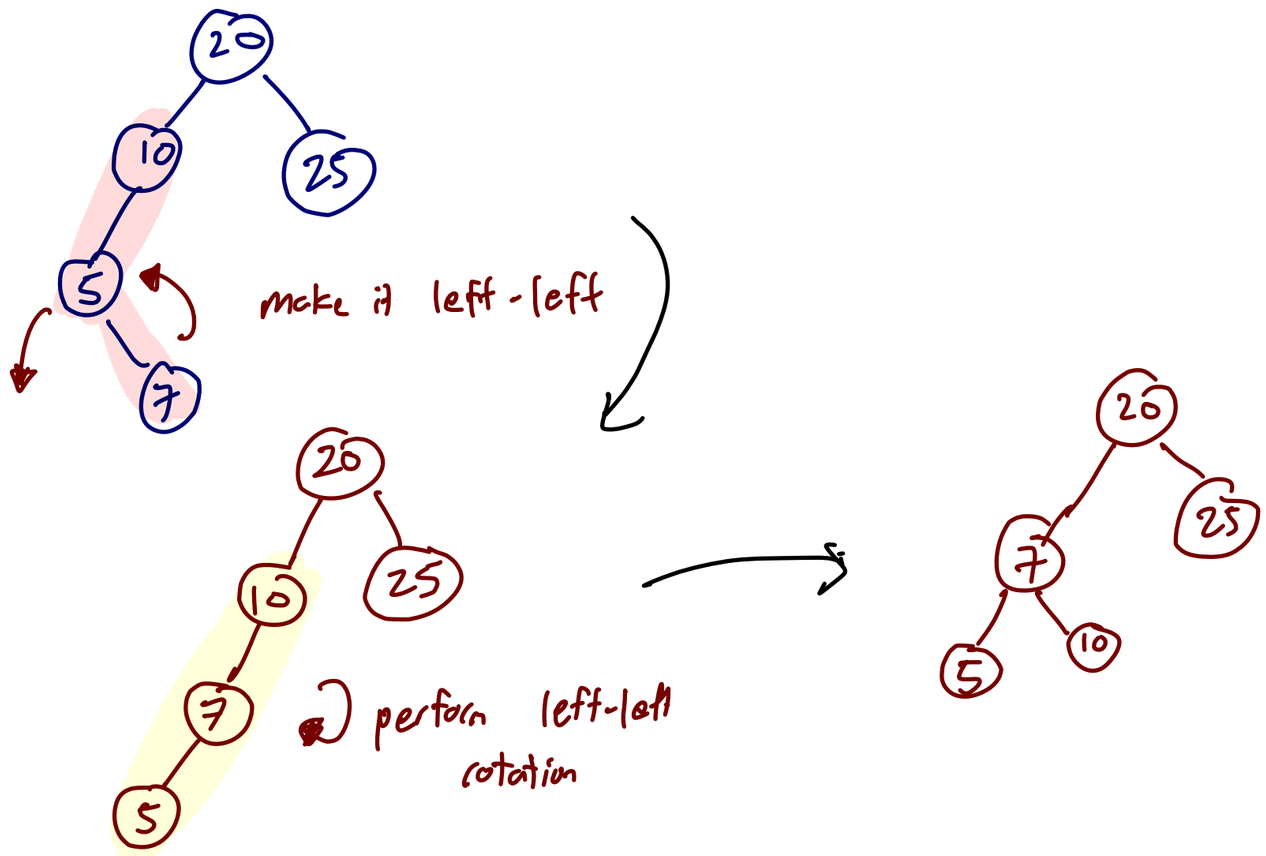
= {-1, 0, 1} ← if outside range, then tree is unbalanced

rotations

left - left (right - right same)

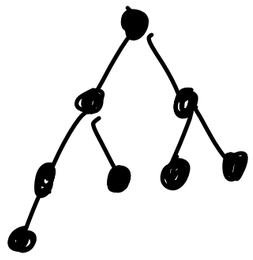


left - right (right - left same)



Show that AVL trees of n-nodes take O(log n) time to perform an insert & rotation.

↳ always balanced



height always approximately $O(\log n)$

When do I use an AVL tree, when do I use a Hash Table?

	AVL tree	Hash table
traversal time	$O(\log n)$ ↑	↓ $O(1)$
space taken	only space needed is created ↓	↑ bunch of extra space