CS106 Data Structure & Java Fundamentals

BitTiger丨来自硅谷的终身学习平台

Claire & Bob

Chapter 3 - Data Structures

- ArrayList
- LinkedList
- HashTable/Map/Set
- Tree
- Stack/Queue
- Heap

Students' Feedback

- Guidance on self-learning

Chapter 2 - Data Structures

- ArrayList
- LinkedList
- HashTable/Map/Set
- Tree
- Stack/Queue
- Heap

ArrayList - method review

- create - List<String> list = new ArrayList<>();
- add - list.add(e) - return: boolean
- remove - list.remove(index) - return: element removed
- get - list.get(index) - return: element
- size - list.size() - return: int

Learning by Doing

Cracking the code interview - Missing ranges

ArrayList - time complexity analysis

size: 0
elementData.length: 0

```
import java.util.ArrayList;

ArrayList<Integer> a = new ArrayList<>();
```

ArrayList - add

size: 1
elementData.length: 10

```
ArrayList<Integer> a = new ArrayList<>();

a.add(1);
```

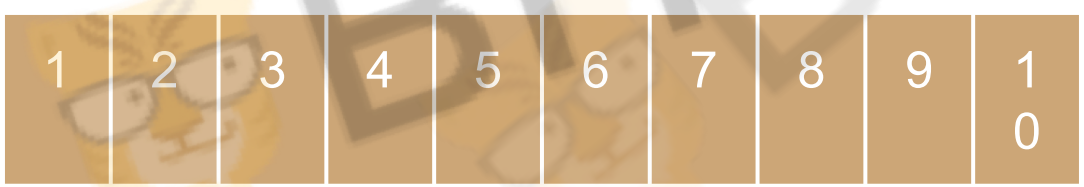
ArrayList - add

size: 10
elementData.length: 10

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}
```



ArrayList - resize

```
int oldCapacity = elementData.length;
int newCapacity = oldCapacity + oldCapacity >> 1;
```

```
oldCapacity = 10;
newCapacity = 10 + 10/2;
```

0	0	0	0	1	0	1	0

10

>>

0	0	0	0	0	1	0	1

5

ArrayList - resize

size: 10
elementData.length: 10

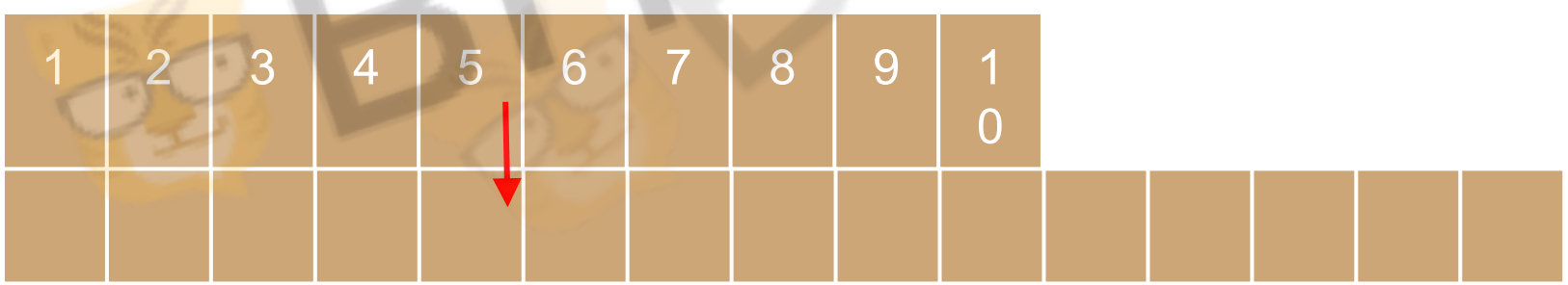
```
newCapacity =
10 + 10 >> 1 = 15
```

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
```



ArrayList - resize

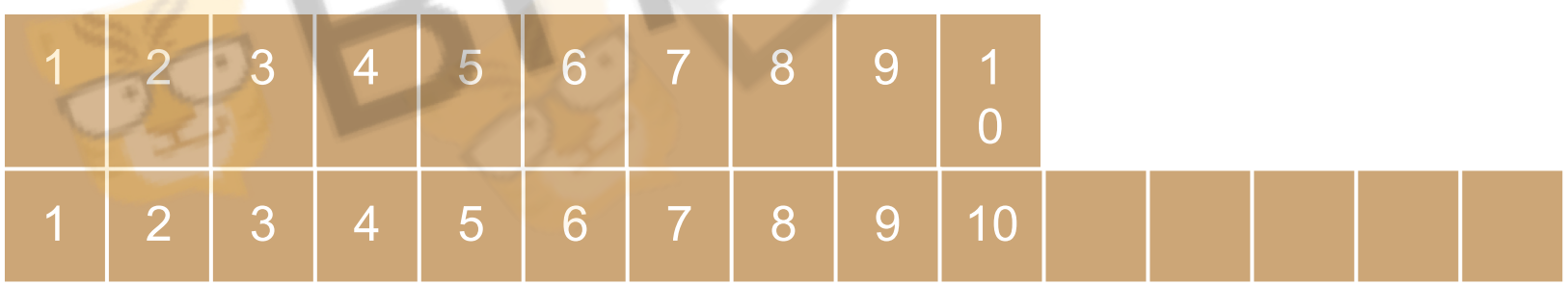
size: 10
elementData.length: 10

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
```



ArrayList - resize

size: 10
elementData.length: **15**

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
```

ArrayList - resize

size: 11
elementData.length: 15

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
```

Amortized O(1)

ArrayList - remove

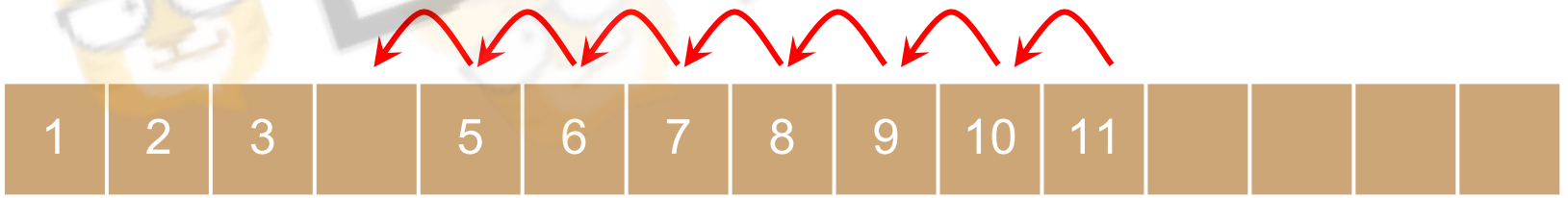
size: 11
elementData.length: 15

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
a.remove(3);
```



ArrayList - remove

size: 11
elementData.length: 15

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
a.remove(3);
```

ArrayList - remove

size: **10**
elementData.length: 15

a[index]
a[5]

```
ArrayList<Integer> a = new ArrayList();

a.add(1);

for (int i = 2; i <= 10; i++) {
  a.add(i);
}

a.add(11);
a.remove(3);
```

ArrayList - time complexity analysis

- Resizable Array,
- Access: O(1)
- Add: O(1) **Amortized**
- Remove: O(n)

Chapter 3 - Data Structures

- ArrayList
- LinkedList
- HashTable/Map/Set
- Tree
- Stack/Queue
- Heap

What's Linkedlist?

- A list of Node
- node = value + pointer to next node
- The last node will point to null

LinkedList in Java

Doubly linked list

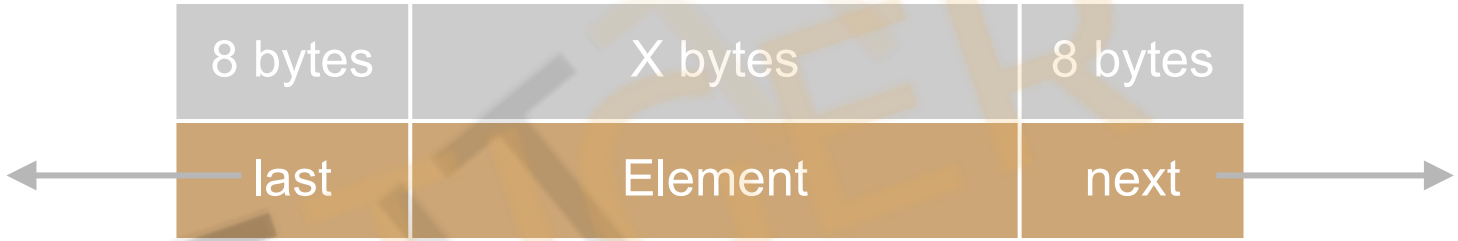
source code - http://developer.classpath.org/doc/java/util/LinkedList-source.html

```
public class LinkedList<E> extends … {
    Node<E> first;
    Node<E> last;
}
```

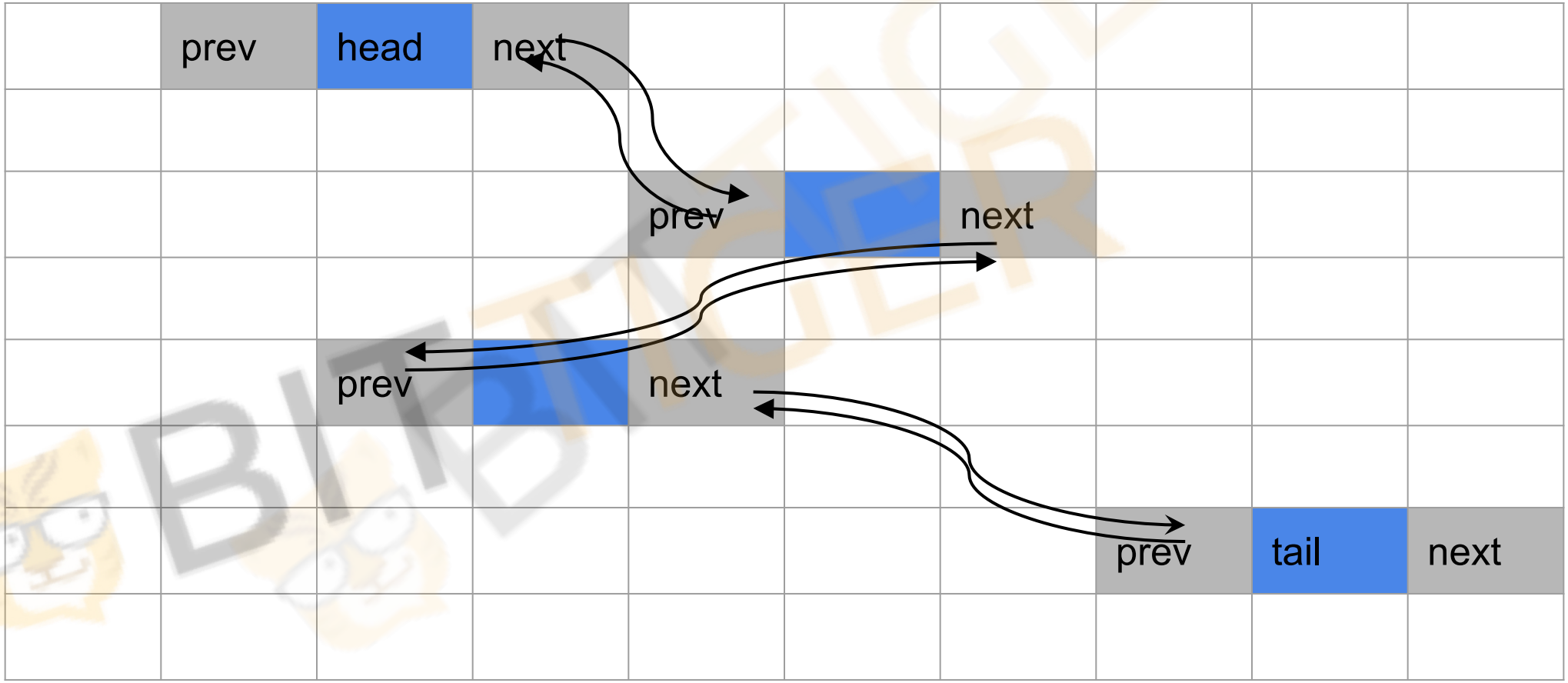
```
private static class Node<E> {
    E item;
    Node<E> next;
    Node<E> prev;

    Node(Node<E> prev, E element, Node<E> next) {
        this.item = element;
        this.next = next;
        this.prev = prev;
    }
}
```

Doubly LinkedList



Doubly LinkedList



LinkedList in Java

- Access: O(n)
- AddFirst: O(1)
- AddLast: O(1)
- RemoveFirst: O(1)
- RemoveLast: O(1)
- Set: O(n)

```
public class LinkedList<E> extends … {
    Node<E> first;
    Node<E> last;
}
```

```
private static class Node<E> {
    E item;
    Node<E> next;
    Node<E> prev;

    Node(Node<E> prev, E element, Node<E> next) {
        this.item = element;
        this.next = next;
        this.prev = prev;
    }
}
```

Comparison

	ArrayList	LinkedList
Random Access	O(1)	O(n)
Add	O(1) Amortized	O(1)
Remove	O(n)	O(1)
Set	O(1)	O(n)

Learning by Doing

Cracking the coding interview - Plus one linked list

Take-away

- use dummy node to simplify corner cases
- oneway, use two pointers

Chapter 3 - Data Structures

- ArrayList
- LinkedList
- HashTable/Map/Set
- Tree
- Stack/Queue
- Heap

Object - hashCode()

```
/* Class Object is the root of the class hierarchy. Every class  has
Object as a superclass. All objects, including arrays, implement the
methods of this class.*/

public class Object {

  public native int hashCode();

  public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
  }
}
```

Object - hashCode()

```
int[] a = {1, 2, 3};

System.out.println(a);
```

```
>> [I@61bbe9ba
```

[I	int[]
@	
61bbe9ba	Address

```
public String toString() {
  return getClass().getName() + "@" +
Integer.toHexString(hashCode());
}
```

HashCode

Default: memory address

Object to hashcode:

(1) One to one

(2) One to many

(3) Many to one

(4) Many to many

Hashcode design 3 principles:

1. **Same code for repeated calling in same application**
2. **Equal ⇒ Same hash code**
3. **Same hash code !=> Equals, but good to have**

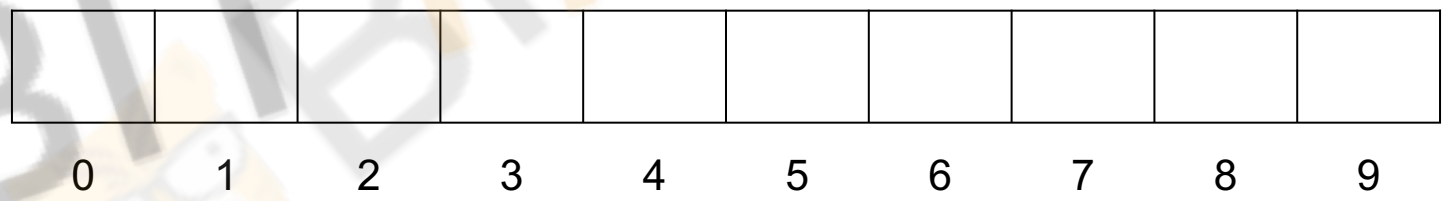
Hash Table

- Searching an element in an array needs ____ time complexity.
- Unsorted array: O(n)
- Sorted array: O(log n) -- Why?
- What if we want to build an array-based data structure with
  - Near O(1) search time complexity
  - Near O(1) to add a new element
  - So we use hashcode to build index!

Hash Table

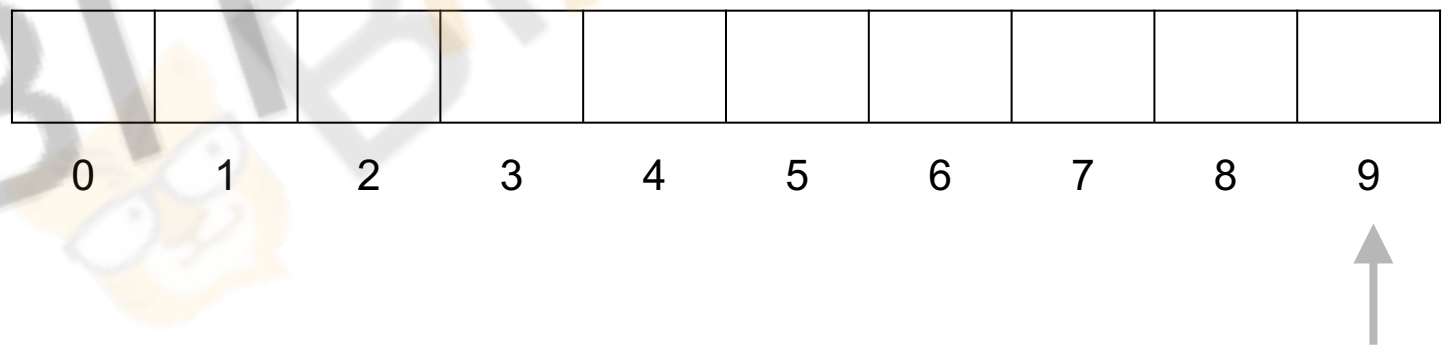
Mapping: hashCode ⇒ arrayIndex

arrayIndex = hashCode % length



Hash Table

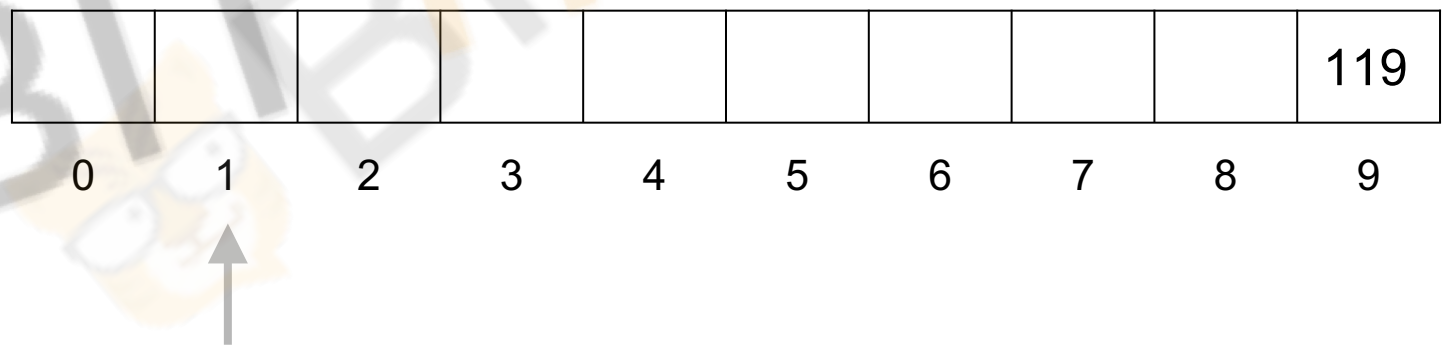
add(119) 119 % 10 = 9



Hash Table

add(119) 119 % 10 = 9

add(321) 321 % 10 = 1

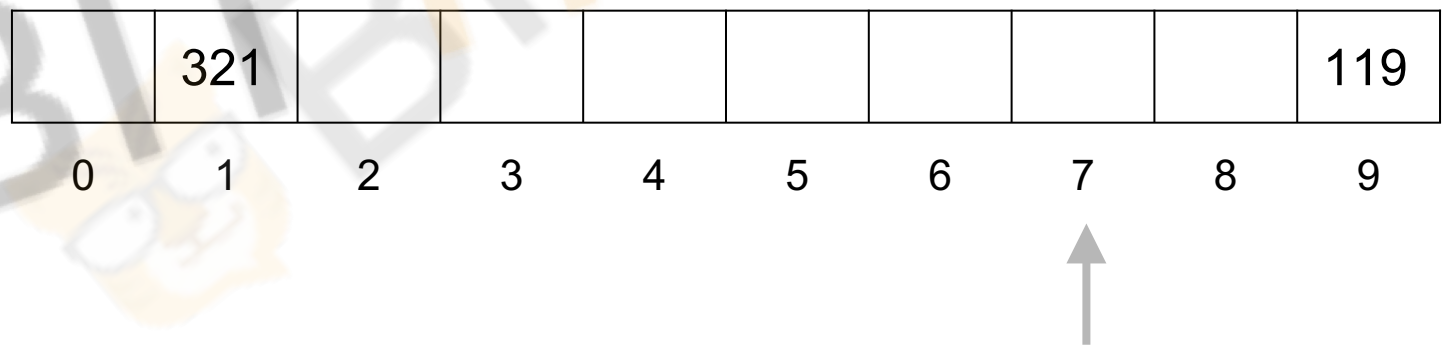


Hash Table

add(119) 119 % 10 = 9

add(321) 321 % 10 = 1

add(157) 157 % 10 = 7

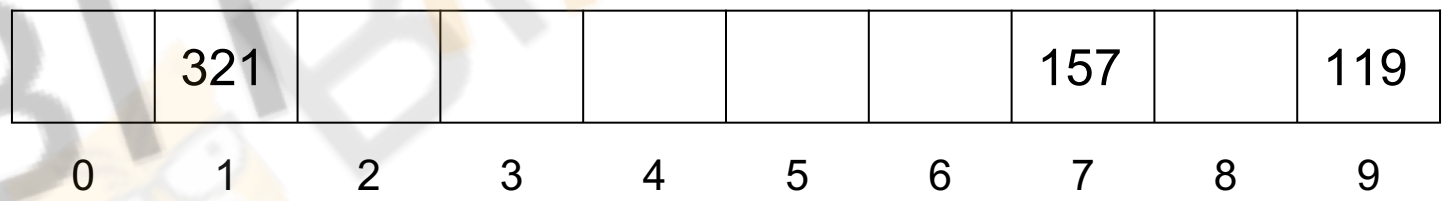


Hash Table

add(119) 119 % 10 = 9

add(321) 321 % 10 = 1

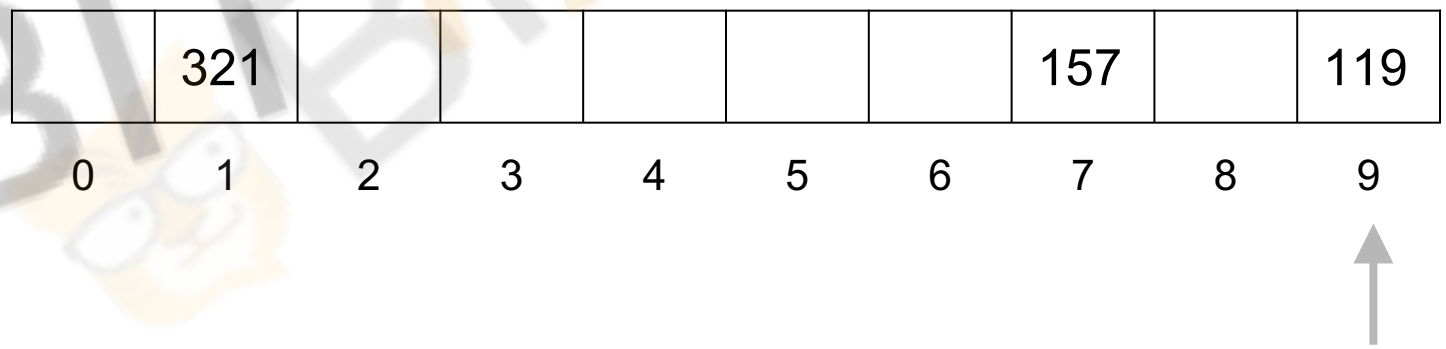
add(157) 157 % 10 = 7



Hash Table

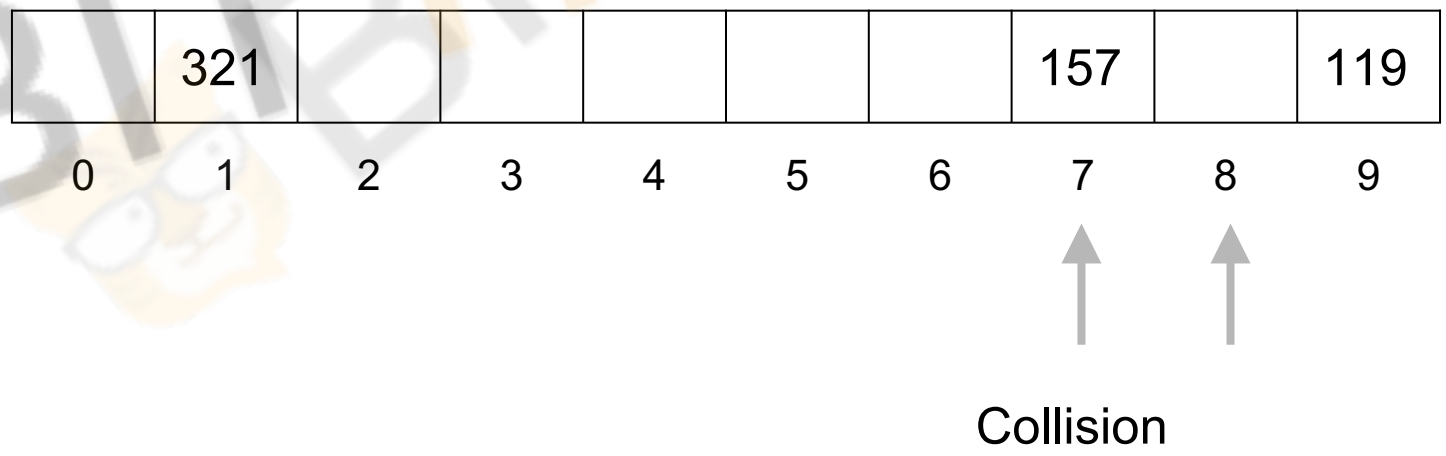
add(119)	119 % 10 = 9
add(321)	321 % 10 = 1
add(157)	157 % 10 = 7
search(119)	119 % 10 = 9

So far:
add() -- O(1)
search() -- O(1)



Hash Table: Linear Probing

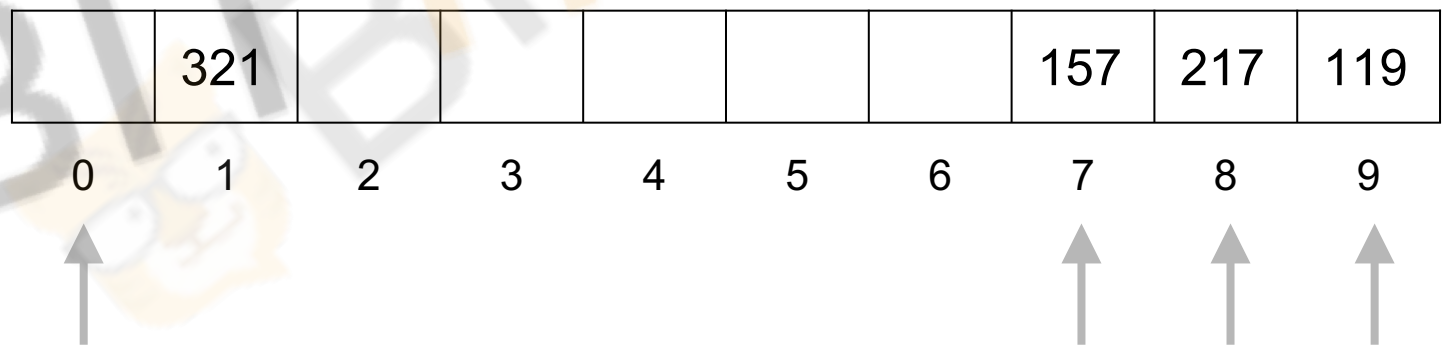
add(217) 217 % 10 = 7



Hash Table: Linear Probing

add(217) 217 % 10 = 7

add(487) 487 % 10 = 7

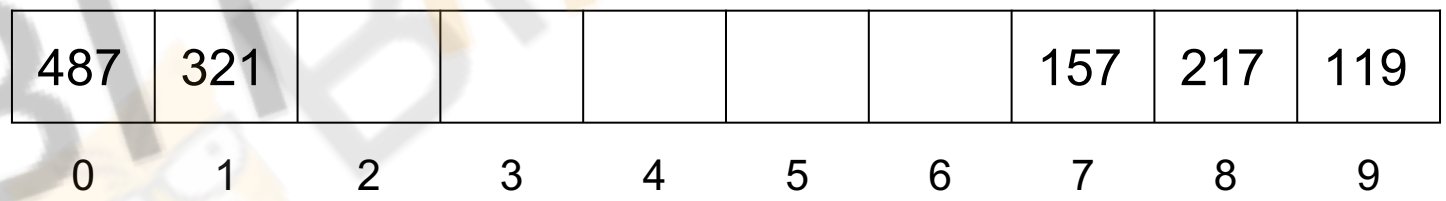


Hash Table: Linear Probing

add(217) 217 % 10 = 7

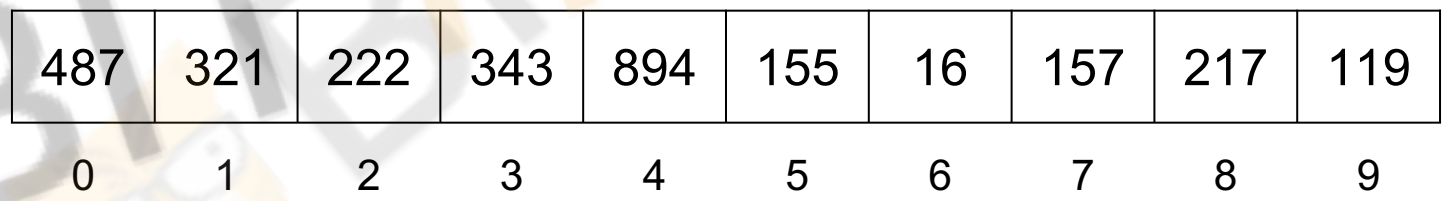
add(487) 487 % 10 = 7

Clustering!



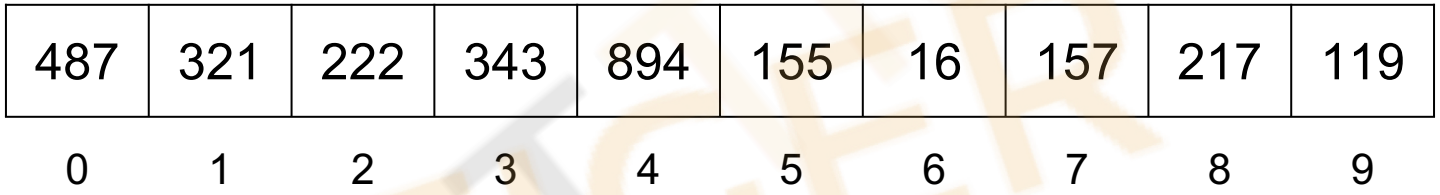
Hash Table: Rehashing

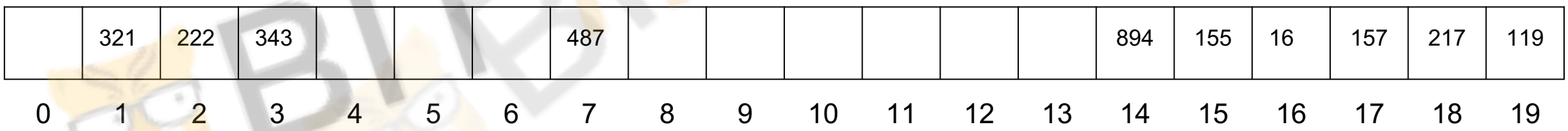
add(12) 12 % 10 = 2



Hash Table: Rehashing

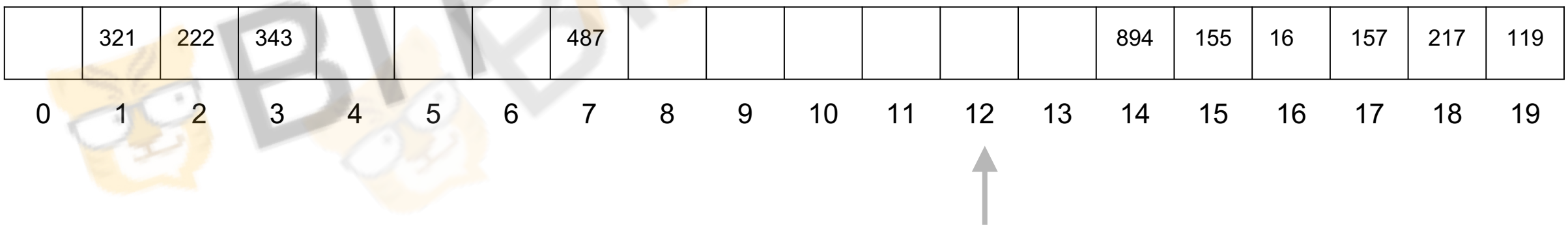
add(12)





Hash Table: Rehashing

add(12) 12 % 20 = 12

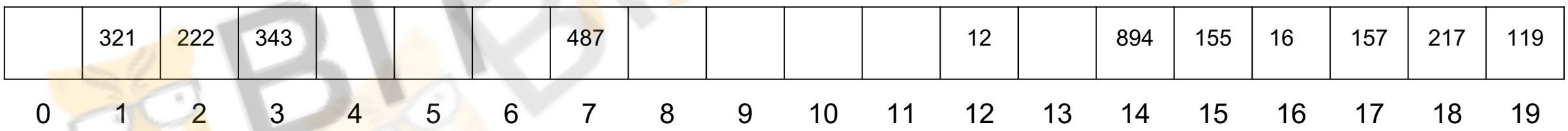


Hash Table: Rehashing

add(12)

12 % 20 = 12

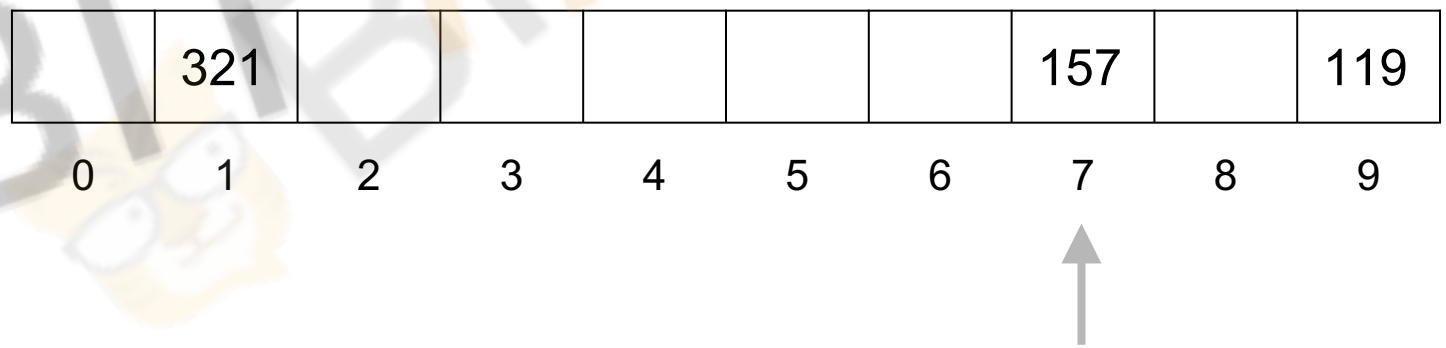
Load Factor
= number of entries / length



Let play back a little bit

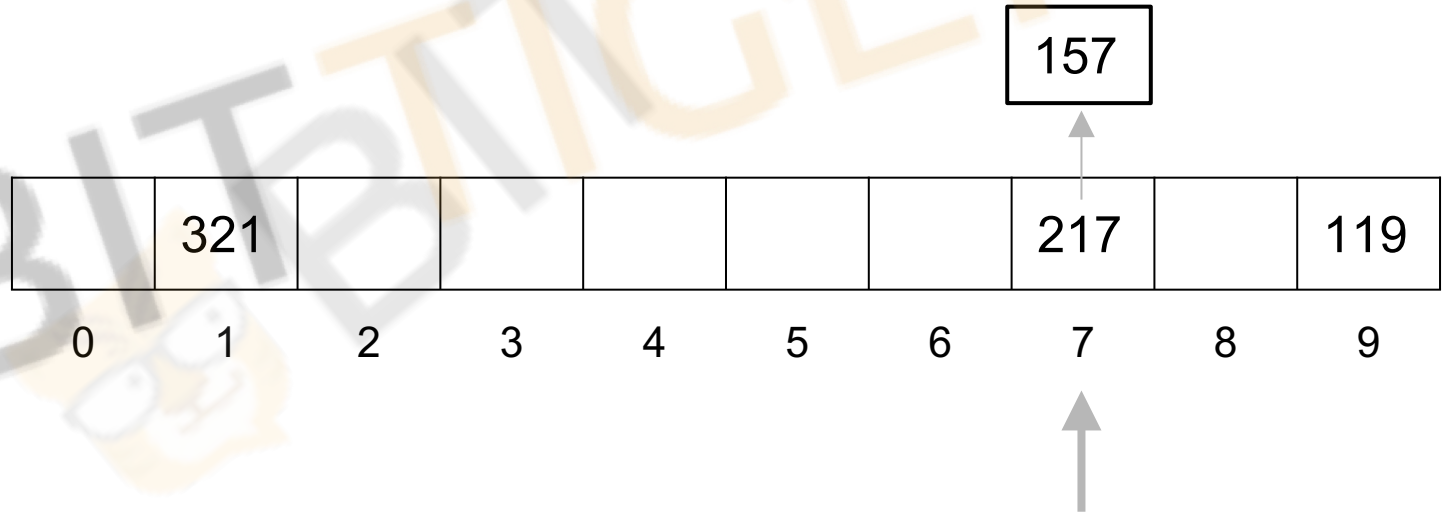
Hash Table: Separate Chain

add(217) 217 % 10 = 7



Hash Table: Separate Chain

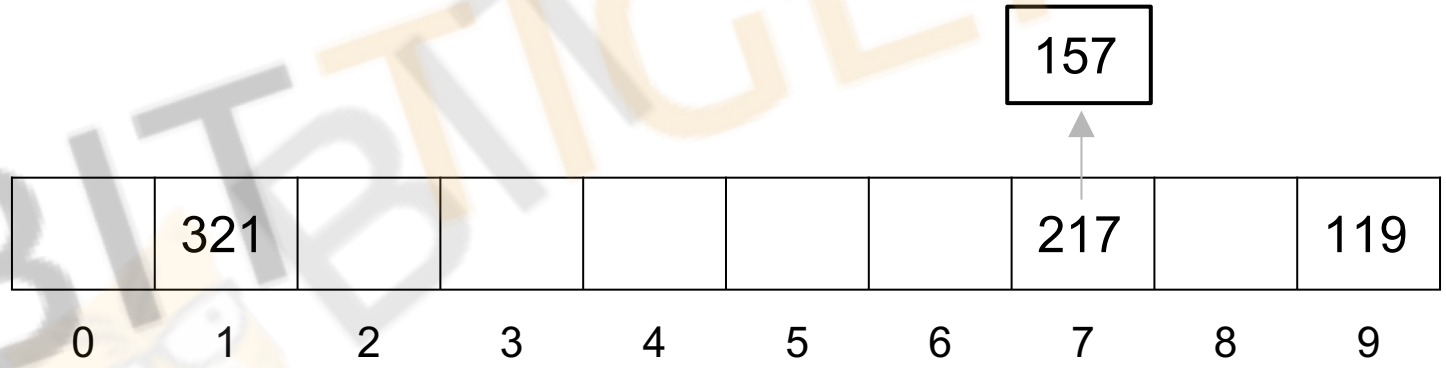
add(217) 217 % 10 = 7



Hash Table: Separate Chain

add(217) 217 % 10 = 7

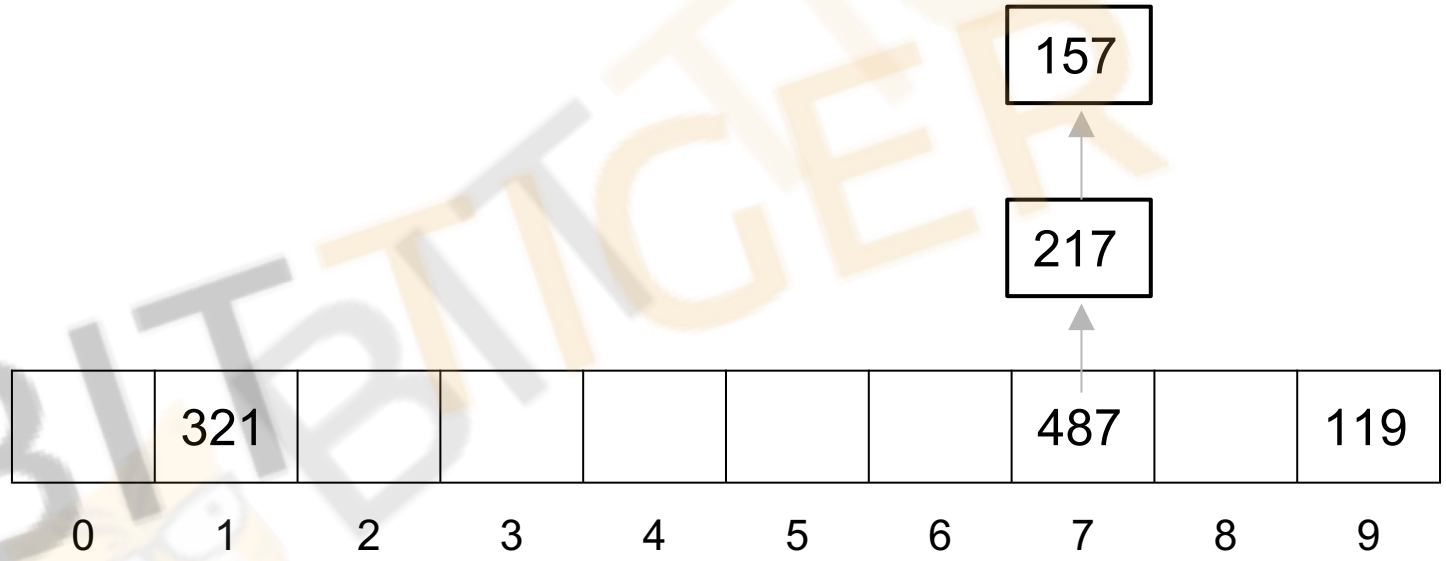
add(487) 487 % 10 = 7



Hash Table: Separate Chain

add(217) 217 % 10 = 7

add(487) 487 % 10 = 7



Hash Table

- Use hashcode and array length to get index of array.
- Collision
  - Linear Probing (Clustering)
  - Separate Chain (Java implementation)
- Rehashing when number of elements is growing faster (⅔)
- Hash Table
  - Near O(1) search time complexity
  - Near O(1) to add a new element

HashTable v.s. HashMap

	HashTable (Legacy API)	**HashMap**
Synchronization	synchronized/thread-safe	Non thread-safe/ Better performance
Null	Don’t allow null keys	allows one null key and any number of null values
Others		LinkedHashMap/ConcurrentHashMap

HashMap: Iteration

321, 487, 217, 157, 119

HashMap (prior to Java 7)

- Separate Chain HashTable Implementation
- In Java 8, HashMap has some performance improvement!

HashSet

```
public HashSet(int initialCapacity) {
    map = new HashMap<>(initialCapacity);
}

public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}

private static final Object PRESENT = new Object();
```

HashSet is a wrapper class of HashMap.
Value is a constant dummy object.

HashMap and HashSet in Java

HashMap - https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html	HashSet - https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html
containsKey(Object key) -- boolean	size() -- int
containsValue(Object value) -- boolean	contains(Object o) -- boolean
get(Object key) - v	remove(Object o) -- boolean
put(Object key, Object value) - v	add(E e) - boolean

Learning by Doing

Cracking the coding interview - TwoSum