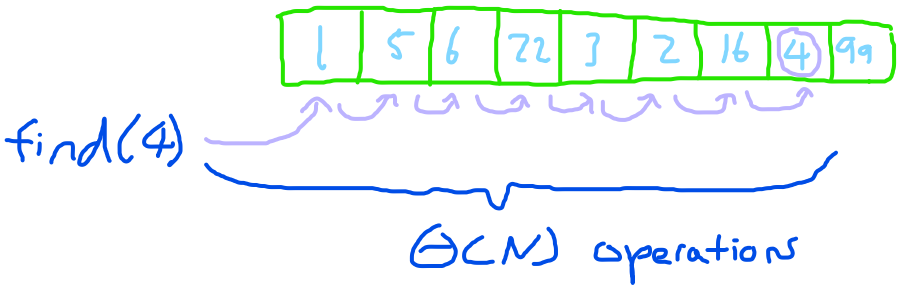
Hashing

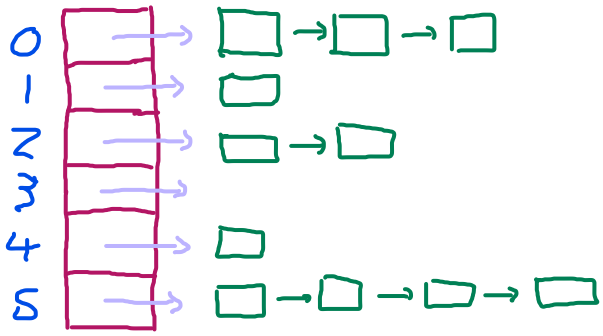
- With lists, we can access an item at any index in $\Theta(1)$! But what can we not do? We cannot find a specific item in $\Theta(1)$.



- Solution for $\Theta(1)$ find? HASHING!

Hashtables (separate chaining array)

- We have a hashtable (array) with every index pointing to a linked list (we call this index a bucket). Every node in the linked list represents an item we have already inserted.



- To put(X), we:

hash function; we will discuss this more later!

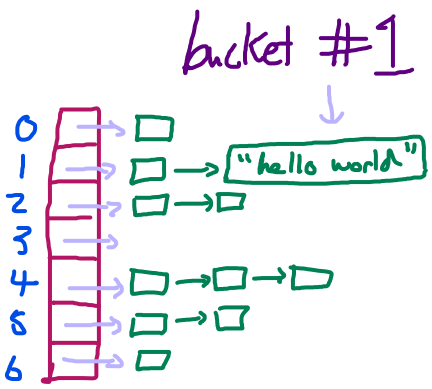
1. Calculate hash code of x: hashCode(X)
2. Reduce our hash code to a hash bucket index:

$$\text{bucket index} = \text{hashCode}(X) \bmod \underbrace{m}_{\#\text{ buckets}}$$

3. Check if bucket index already contains(X)

④ We have a load factor: # items / buckets = 0.75.
When we exceed this load factor, we resize the hashtable geometrically. We prove why we need this later.

"hello world" → hashCode("hello world") → -5132579 → Math.mod(N, M)



- Contains(X)
  ① Calculate bucket index X should go into
  ② for item in linked_list:
  if Key.equals(X) return True
  return False
- Hash code criteria:

Requirements:

① Consistency: if hashCode(X) called multiple times, will always return same thing on same X.

② Equality Constraint: if obj1.equals(obj2),
then hashCode(obj1) == hashCode(obj2)

☆ Note, the opposite is NOT always true. items with the same hash code does not imply equality!

Good hash codes:

- Good hash codes will minimize:

$$Prob(hashCode(x) == hashCode(y))$$

Bad hash code function will not do this and as a result many items end up in the same bucket, making contains take a long time.