Integer Overflow Collisions

In Java, the largest int is 2,147,483,647.

Going over this limit results in **overflow**, starting back over at the smallest int.

If there are more unique mappings than unique ints, then **collisions** will still occur!

```
int x = 2147483647;
System.out.println(x);
// 2147483647
System.out.println(x + 1);
// -2147483648
```

```
DataIndexedStringSet disi;
disi.add("melt banana");
disi.contains("subterrestrial anticosmetic");
// true: both strings hash to 839099497
```

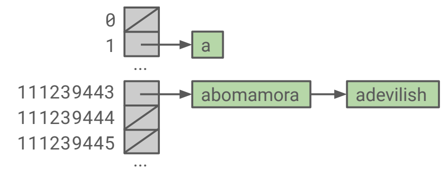

Separate Chaining

Instead of storing a boolean, store a **bucket** of items at the given index.

```
add("a")
add("abomamora")
add("adevilish")
add("abomamora")
contains("adevilish")
```

Each bucket in our array is initially empty. When an item **x** gets added at index **h**…

- If bucket **h** is empty, create a new list containing **x** and store it at index **h**.
- If bucket **h** is already a list, add **x** to this list if it is not already present.

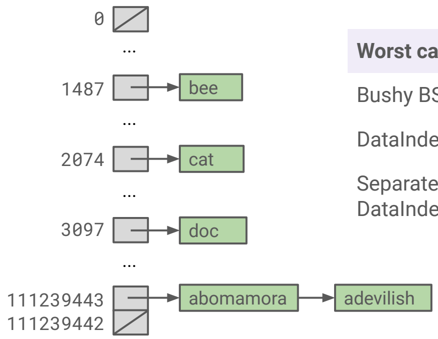


?: Why is it necessary to check if **x** is not already present in the bucket before adding **x**?

?: When would it not be necessary to check if x is already present in the bucket?

Separate Chaining Runtime

Worst case runtime will be proportional to length of longest list, Q.

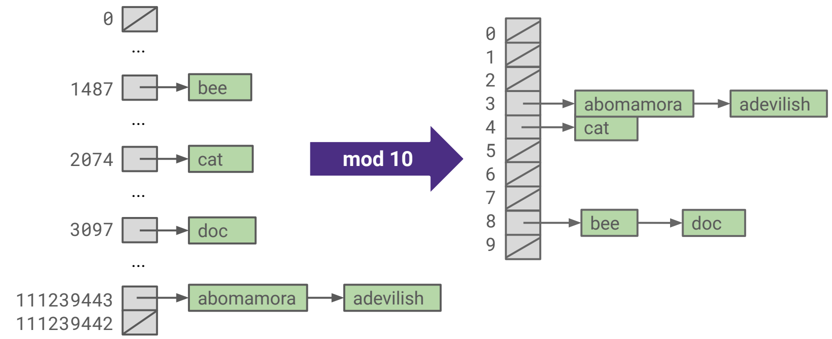


Worst case time	contains(x)	add(x)
Bushy BSTs	Θ(log N)	Θ(log N)
DataIndexedSet	Θ(1)	Θ(1)
Separate Chaining DataIndexedSet	Θ(Q)	Θ(Q)

?: Why is the runtime for separate chaining in terms of Q, the length of the longest list?

Saving memory with Separate Chaining and modulus

Instead of using the raw hash code, take the modulus of the hash code to compute index.



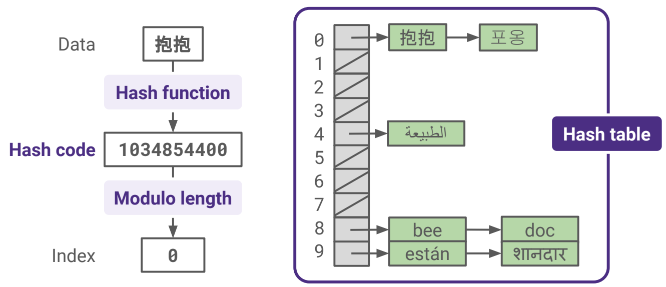
?: Do items with the same hash code (collision) still collide after applying mod 10? What about items with different hash codes?

?: How does this change affect runtime? The length of the longest list, Q?

Hash Table

Data is converted by a **hash function** into an integer representation called a **hash code**.

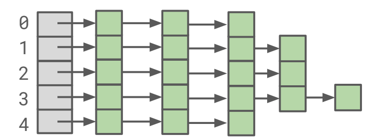
The **hash code** is reduced to a bucket index with the modulo operator.



Hash Table Runtime

Good news. We use way less memory and support any String.

Bad news. Worst case runtime is now Θ(Q), where Q is the length of the longest list.



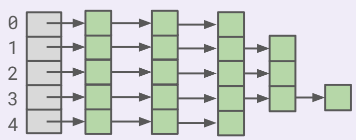
Worst case time	contains(x)	add(x)
Bushy BSTs	Θ(log N)	Θ(log N)
DataIndexedSet	Θ(1)	Θ(1)
Separate Chaining Hash Table	Θ(Q)	Θ(Q)

?: What's a potential problem with saving memory by using the modulus idea?

Improving Hash Table Runtime

Even if items are distributed evenly, lists are of length Q = N / M. For M = 5, Q ∈ Θ(N).

How can we improve our design to guarantee that Q ∈ Θ(1)?



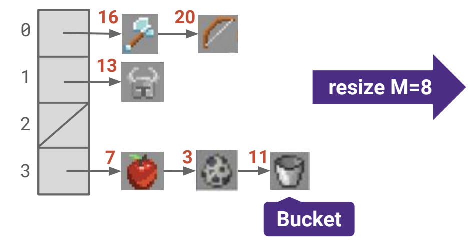
Worst case time	contains(x)	add(x)
Bushy BSTs	Θ(log N)	Θ(log N)
DataIndexedSet	Θ(1)	Θ(1)
Separate Chaining Hash Table	Θ(Q)	Θ(Q)

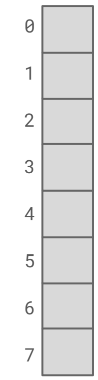
Q1: How can we improve our design to guarantee that Q ∈ Θ(1)?

Hash Table Resizing

When N / M ≥ 1.5, double the number of buckets, M.

N = 6 M = 4 N / M = 1.5



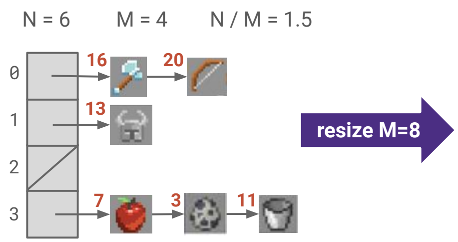


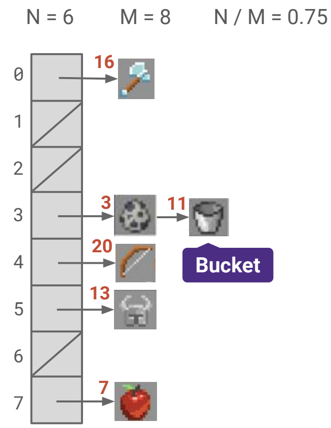
?: After resizing, where will the bucket go?

?: Fill in the resulting hash table after resizing.

Hash Table Resizing

When N / M ≥ 1.5, double the number of buckets, M.





?: What is the best case order of growth of Q with respect to N?

?: What is the worst case order of growth of Q with respect to N?

Resizing Hash Table Runtime

Best case. All items are distributed evenly across M ~ N buckets, so $Q \in \Theta(1)$.

Worst case. All items collide in a single bucket, so $Q \in \Theta(N)$.

contains(x): Compute hash code of **x**, take modulus, search the list of items.

add(x): **Resize** if N / M exceeds the load factor. Add **x** if the table does not **contains(x)**.

Most add operations will be $\Theta(Q)$, but some will be $\Theta(N)$.

If we choose to resize by doubling, tripling, etc. the runtime "on average" will be $\Theta(Q)$.

More detail on resizing in the future.

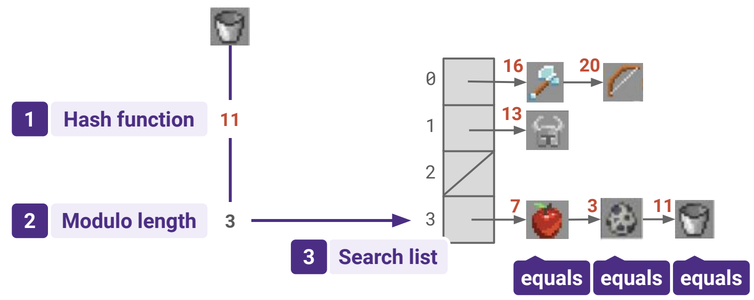
?: As the hash table designer, what can we do to avoid the worst case scenario? What can we do as a hash table user?

Is this a valid hash function?

```
public int hashCode() {
    return 17;
}
```


Q1: Is this a valid hash function?

hashCode Contract



We know that unequal items can return the same hash code.

?: Do equal items need to return the same hash code?