We want to build a set of integers and detect whether or not a particular integer is in that set.

And we want to do that quickly

So the basic idea is we’re going to take an integer-call it i. and we’re going to hash it.

What a hash function does is it converts i to a different integer, perhaps, in some range.

We’re going to then use it this integer to index into a list of lists

I have a function called create, which uses this global variable and creates a list of lists, each element of which is empty. Because initially, we have no elements in the set.

When I want to insert an element in the set, I’ll call the function insert, which will hash the element.

What’s the property of this hash function?

Hash is many-to-one. That is to say, an infinite number of different integers will hash to the same value. Because after all, I have a set in which I can store any integer or any positive integer, at least and there are only 47 buckets. So it’s pretty obvious that many integers will hash to the same bucket.

When two different elements hash to the same bucket, we have what is called a collision.

There are lots of different ways to handle collisions. What I’ve shown you here is probably the simplest way, which is called linear rehashing.

I’m glad somebody has a question. You have to ask loudly.

AUDIENCE: when you take the modulus 47, what does that return? 0

So hashElem always returns 0? No . It depends what I’m hashing. Sorry, I thought if you were saying that if I asked 47 mod 47. If I take 48 mod 47, I get 1. If I take 49 mod 47. It’s the remainder.

It will be the length of the bucket, the size of the bucket. Now, I don’t know how many elements will be in the bucket. But what will this depend on?. It will depend upon the number of buckets.

If I have a million buckets, I’ll get a lot fewer collisions than if I have two buckets.

[325, 325]---we say we happen to have one bucket that’s got two elements in it.Happen to be the same.

[34, 987654321]---this very big number happened to hash to the same value of 30 as 34. Here we have two elements in it.

A good hash function has the property that it will widely disperse the values you hash. So they end up in different buckets, rather than some stupid hash function that tends to put everything in the same bucket.