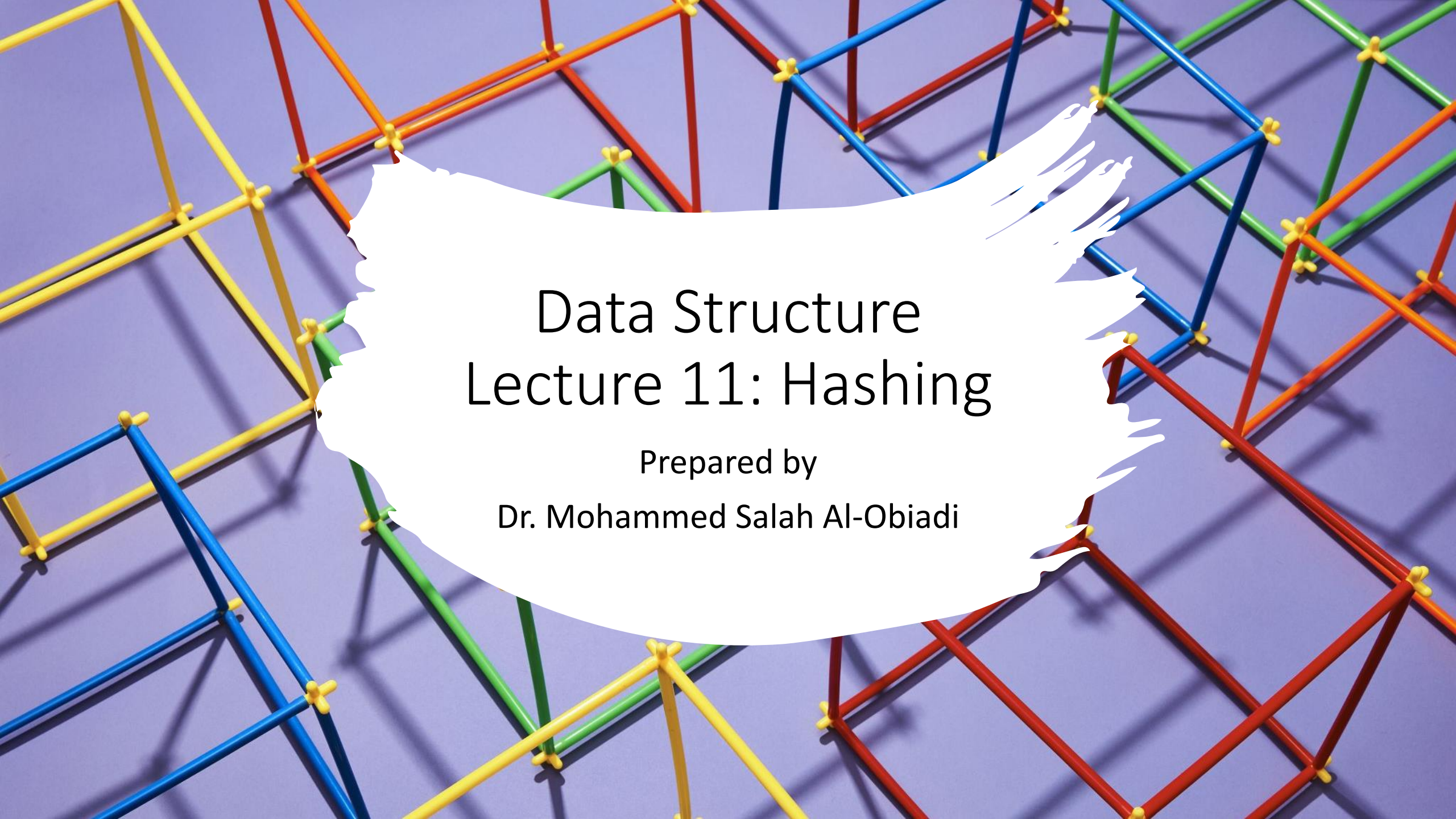
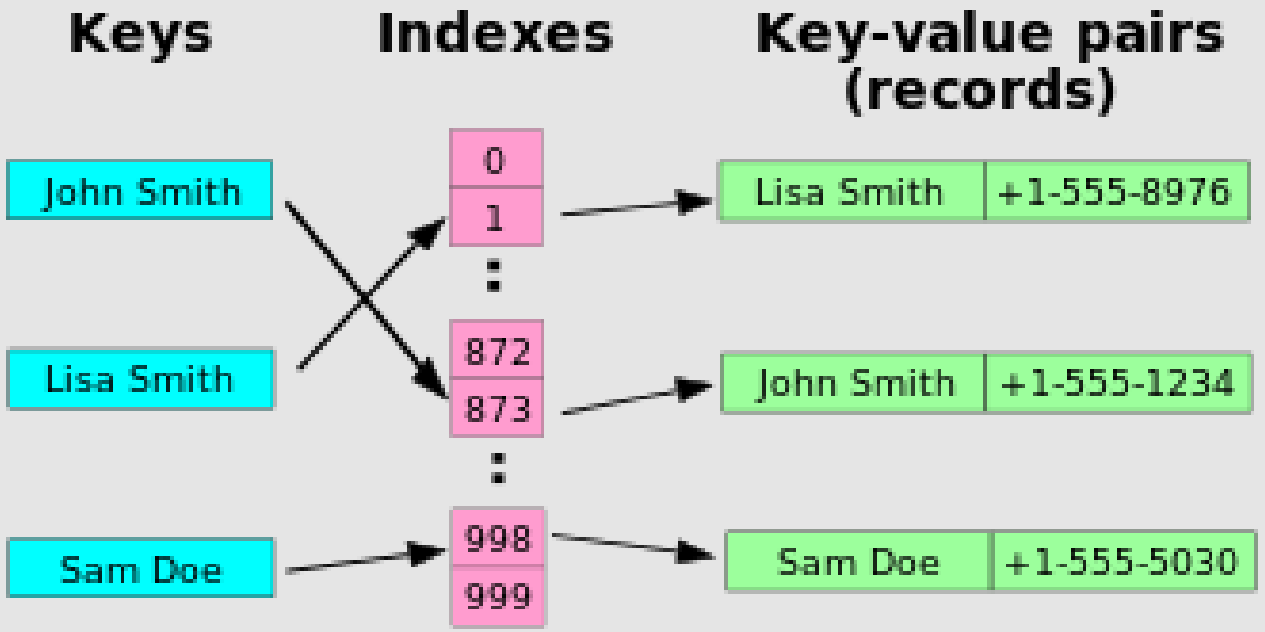


Data Structure
Lecture 11: Hashing

Prepared by

Dr. Mohammed Salah Al-Obiadi

Hash Table



- Hashing is a technique used for storing and retrieving information as quickly as possible.
- It is used to perform optimal searches and is useful in implementing symbol tables.
- Arrays, linked list, trees requires O(N) or log(N) to, search, traverse, add or delete, while with hashtable it requires only O(1) time to do these operations.
- A hash table is a *array* with mapping function.

Why hash table is important

- If we have a set of names Ahmed, Yaser, Yasien, Mina, Mustafa, Lina.
- Let's suppose that you put these names in an array or a linked list.
- Let’s suppose that you are looking for the name Lina.
- Then, you will need to match all the array or linked list elements to find that name because it’s the last name in the list.
- This name costs loop of 6 to find it and if the list of names = n, then it costs O(n) to find it.
- With hash table, you give the name Lina, and you get it immediately in O(1). How is that? We will see in the next slides.

Time Complexity in Big O notation

Algorithm	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(1)	O(*n*)
Insert	O(1)	O(*n*)
Delete	O(1)	O(*n*)

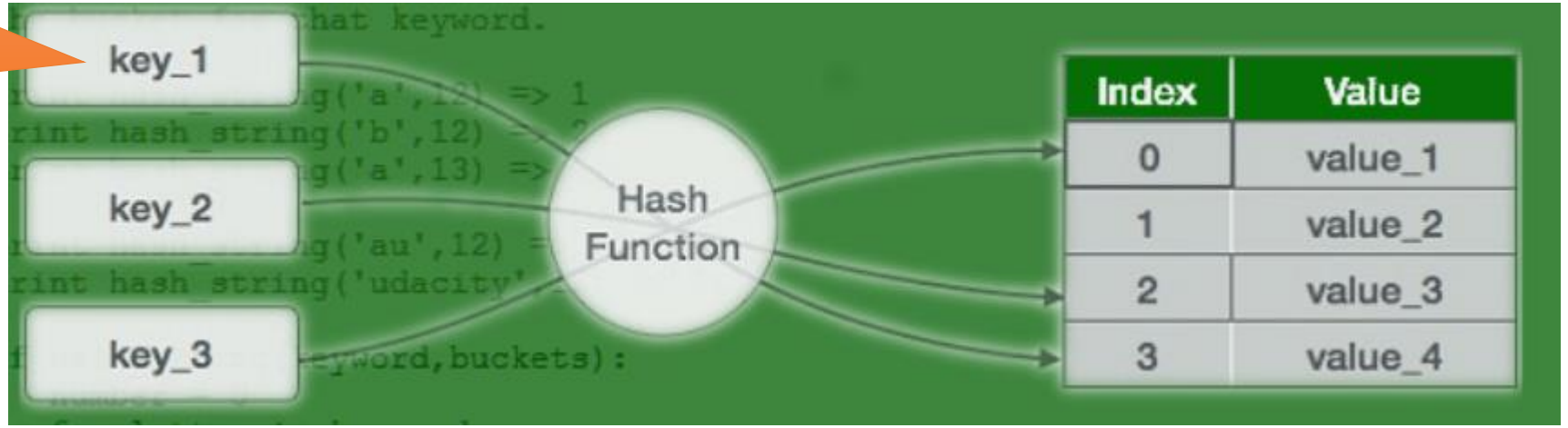
Components of Hashing

- Hashing has four key components:
  1. Hash Table
  2. Hash Functions
  3. Collisions
  4. Collision Resolution Techniques

Hash Table

- Hash table or hash map is a data structure that stores the keys and their associated values.
- Hash table is a kind of array.
- It uses a hash function to map keys to their associated values.
- The terms widely used with hash table are Key, value. See Figure down.

Key here can be number, name, text, picture!!



Hash Functions

The hash function is used to transform the key into the index.

The hash function should map each possible key to a unique slot index.

A hash function that maps each item into a unique slot is referred to as a *perfect hash function.*

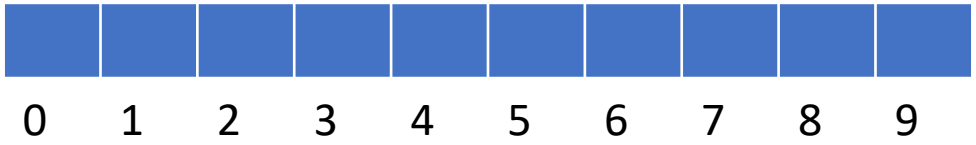
A hash function can lead to collisions.

Collisions happen when a hash function for two keys produce same index.

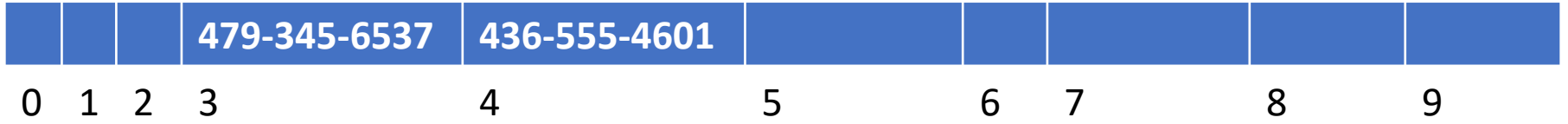
Our goal is to create a hash function that minimizes the number of collisions

How to choose your hash function?

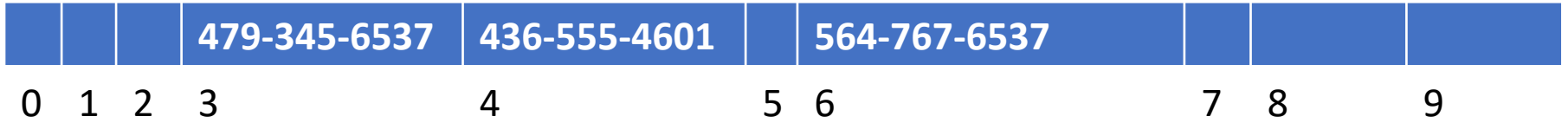
- Example of phone number 436-555-4601.
- Consider the hash function h(x) = sum of numbers % hashtable size.
- The % is the mode or division reminder and x is the phone number.
- Suppose hashtable size=10.



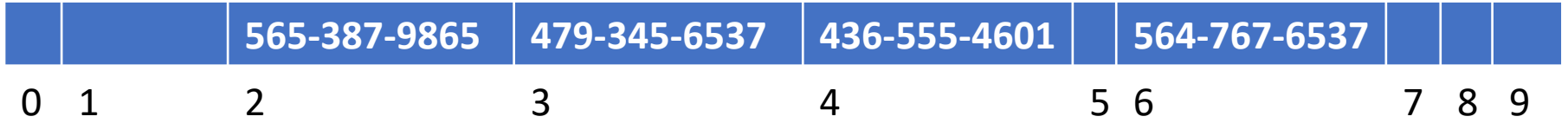
- So, h(436-555-4601)= (4+3+6+5+5+5+4+6+0+1) %10 = 4
- Another phone (479-345-6537)
- h(479-345-6537)= (4+7+9+3+4+5+6+5+3+7) %10 = 3



- h(564-767-6537)= (5+6+4+7+6+7+6+5+3+7) %10 = 6



- h(565-387-9865)= (5+6+5+3+8+7+9+8+6+5) %10 = 2



- h(343-387-9865)= (3+4+3+3+8+7+9+8+6+5) %10 =6, here we have collision because index 6 already full.

- Problems with hashtable:
  1. Number of elements can be larger than hashtable size.
  2. An input that can lead to same index which is called collision

Characteristics of Good Hash Functions

1. Minimize collision
2. Be easy and quick to compute
3. Distribute key values evenly in the hash table
4. Use all the information provided in the key

Collisions

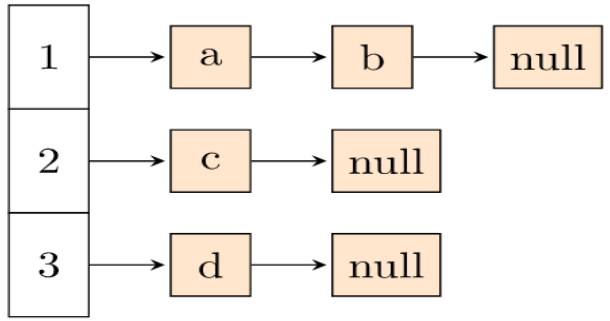
Hash functions are used to map each key to a different address space

But it is not possible to create such a hash function and the problem is called *collision*.

Collision is the condition where two records are stored in the same location

Collision Resolution Techniques

- Collision resolution is The process of finding an alternate location.
- There are a number of collision resolution techniques:
  - **Direct Chaining:** An array of linked list application
    - Separate chaining



  - **Open Addressing:** Array-based implementation
    - Linear probing (linear search)
    - Quadratic probing (*n*onlinear search)
    - Double hashing (use two hash functions)

Linear probing

The interval between probes is fixed at 1.

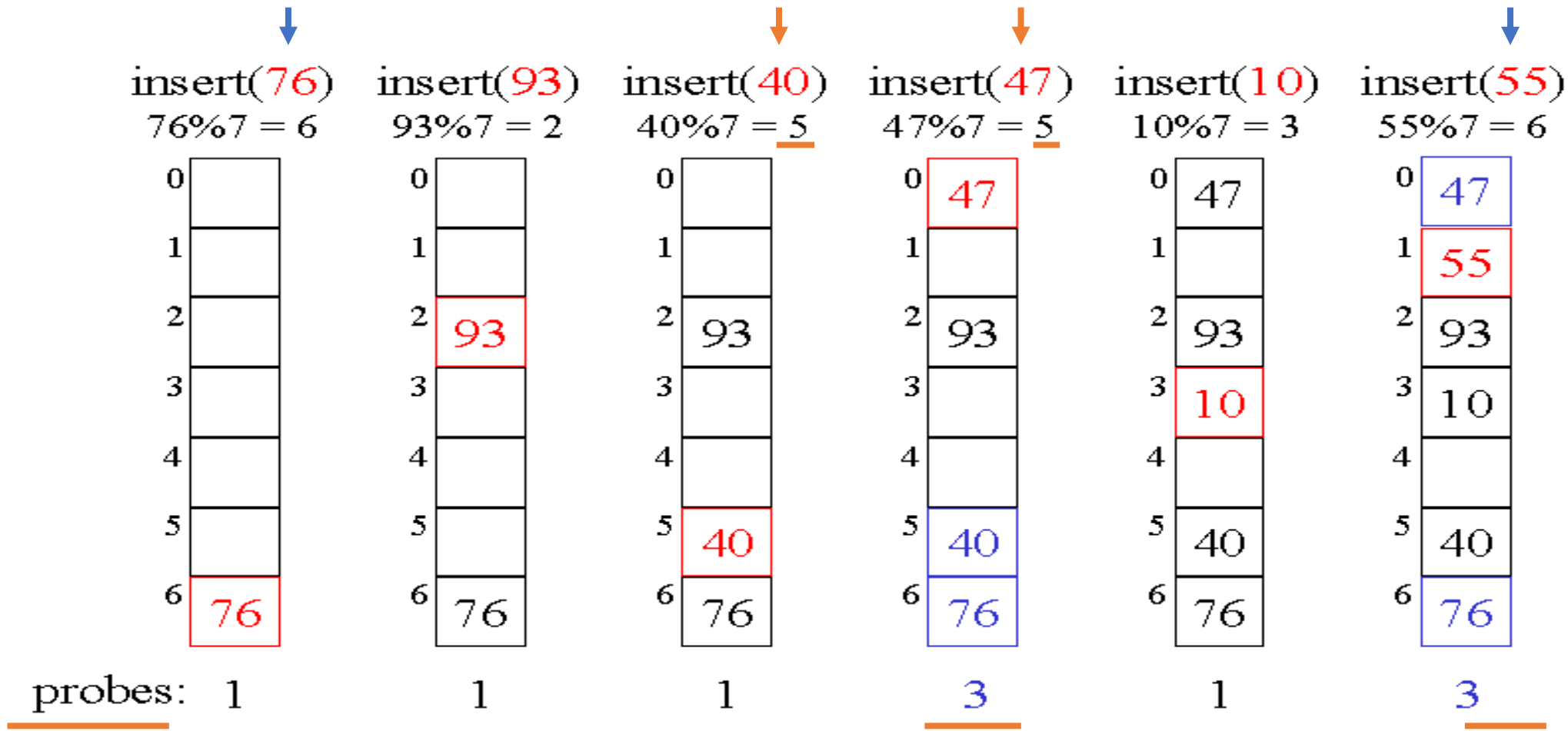
In linear probing, we search the hash table sequentially, starting from the original hash location.

If a location is occupied, we check the next location.

$$rehash(key) = (n + 1)\%\ tablesize$$

Where n is the current location found for key

Linear Probing Example



Quadratic Probing

Linear probing can lead to cluster of keys together.

The problem of Clustering can be eliminated if we use the quadratic probing method

In quadratic probing, we start from the original hash location n.

If a location is occupied, we check the next location.

$$rehash(key) = (n + k^2)\% \ tablesize$$

Where n is the current location found for key

If a location is occupied, we check the locations $i + 1^2$, $i + 2^2$, $i + 3^2$, $i + 4^2$

Example of Quadratic Probing

- Let us assume that the table size is 11 (0..10)
- Insert keys
  - 31 *mod* 11 = 9
  - 19 *mod* 11 = 8
  - 2 *mod* 11 = 2
  - 13 *mod* 11 = 2 → $2 + 1^2$ mod 11 = 3
  - 25 *mod* 11 = 3 → $3 + 1^2$ mod 11 =4
  - 24 *mod* 11 = 2 → $2 + 1^2$ mod 11 , $2 + 2^2 = 6$
  - 21 *mod* 11 = 10
  - 9 *mod* 11 = 9 → $9 + 1^2$, $9 + 2^2$ *mod* 11, $9 + 3^2$ *mod* 11=7

0	
1	
2	2
3	13
4	25
5	5
6	24
7	9
8	19
9	31
10	21