# Hashing - Introduction

**Hashing** is a method of storing and retrieving data from a database efficiently, Suppose that we want to design a system for storing employee records keyed using phone numbers. And we want the following queries to be performed efficiently:

1. Insert a phone number and the corresponding information.
2. Search a phone number and fetch the information.
3. Delete a phone number and the related information.

We can think of using the following data structures to maintain information about different phone numbers.

1. An array of phone numbers and records.
2. A linked list of phone numbers and records.
3. A balanced binary search tree with phone numbers as keys.
4. A direct access table.

For **arrays and linked lists**, we need to search in a linear fashion, which can be costly in practice. If we use arrays and keep the data sorted, then a phone number can be searched in O(Logn) time using Binary Search, but insert and delete operations become costly as we have to maintain sorted order. With a **balanced binary search tree**, we get a moderate search, insert and delete time. All of these operations can be guaranteed to be in O(Logn) time. Another solution that one can think of is to use a **direct access table** where we make a big array and use phone numbers as indexes in the array. An entry in the array is NIL if the phone number is not present, else the array entry stores pointer to records corresponding to the phone number. Time complexity wise this solution is the best of all, we can do all operations in O(1) time. For example, to insert a phone number, we create a record with details of the given phone number, use the phone number as an index and store the pointer to the record created in the table. This solution has many practical limitations. The first problem with this solution is that the extra space required is huge. For example, if the phone number is of n digits, we need O(m * 10n) space for the table where m is the size of a pointer to the record. Another problem is an integer in a programming language may not store n digits. Due to the above limitations, the Direct Access Table cannot always be used. **Hashing** is the solution that can be used in almost all such situations and performs extremely well as compared to above data structures like Array, Linked List, Balanced BST in practice. With hashing, we get O(1) search time on average (under reasonable assumptions) and O(n) in the worst case.

> Hashing is an improvement over Direct Access Table. The idea is to use a hash function that converts a given phone number or any other key to a smaller number and uses the small number as an index in a table called a hash table.

**Hash Function:** A function that converts a given big phone number to a small practical integer value. The mapped integer value is used as an index in the hash table. In simple terms, a hash function maps a big number or string to a small integer that can be used as an index in the hash table. A good hash function should have following properties:

1. It should be efficiently computable.
2. It should uniformly distribute the keys (Each table position be equally likely for each key).

For example, for phone numbers, a bad hash function is to take the first three digits. A better function will consider the last three digits. Please note that this may not be the best hash function. There may be better ways. **Hash Table:** An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry. **Collision Handling**: Since a hash function gets us a small number for a big key, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

- **Chaining:** The idea is to make each cell of the hash table point to a linked list of records that have the same hash function value. Chaining is simple, but it requires additional memory outside the table.
- **Open Addressing:** In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we one by one examine the table slots until the desired element is found or it is clear that the element is not present in the table.