**Complexity Analysis - Hash Table**

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

In this article, we are going to discuss the performance of hash table.

*Complexity Analysis*

If there are M keys in total, we can achieve the space complexity of O(M) easily when using a hash table.

However, you might have noticed that the time complexity of hash table has a strong relationship with the design.

Most of us might have used an array in each bucket to store values in the same bucket. Ideally, the bucket size is small enough to be regarded as a constant. The time complexity of both insertion and search will be O(1).

But in the worst case, the maximum bucket size will be N. And the time complexity will be O(N) for insertion and search.

*The Principle of Built-in Hash Table*

The typical design of built-in hash table is:

1. The key value can be any hashable type. And a value which belongs to a hashable type will have a hashcode. This code will be used in the mapping function to get the bucket index.
2. Each bucket contains an array to store all the values in the same bucket initially.
3. If there are too many values in the same bucket, these values will be maintained in a height-balanced binary search tree instead.

The average time complexity of both insertion and search is still O(1). And the time complexity in the worst case is O(logN) for both insertion and search by using height-balanced BST. It is a trade-off between insertion and search.