



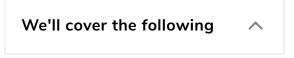






A Quick Overview of Hash Tables

Let's combine all the different operations discussed previously and test out the functionality of our complete hash table class.



- Complete Implementation
- Overview

Complete Implementation#

In the previous lessons, we discussed each aspect of a hash table in detail. Below, you can find the complete hash table class.

```
main.py
HashTable.py
HashEntry.py
     from HashTable import HashTable
  2
  3 table = HashTable() # Create a HashTable
  4 print(table.is_empty())
  5 table.insert("This", 1)
   table.insert("is", 2)
    table.insert("a", 3)
    table.insert("Test", 4)
    table.insert("Driver", 5)
    print("Table Size: " + str(table.get_size()))
 10
     print("The value for 'is' key: " + str(table.search("is")))
 11
```

```
12 table.delete("is")
13 table.delete("a")
14 print("Table Size: " + str(table.get_size()))
15
```

The Set class and dict class in Python work in the same way as the implementation above. So we have basically understood what goes on at the back-end!

Overview#

To sum up the discussion here, hash tables are the ideal data structure when you have a large amount of data and you need all basic operations to work in constant time. The challenge with hash tables comes in the form of deciding the optimized hash function. The large memory cost must also be taken care of.

Here are the time complexities for basic hash table operations:

Operation	Average	Worst
Search	0(1)	0(n)
Insertion	0(1)	0(n)
Deletion	0(1)	0(n)

In the next lesson, we will compare the performance of hash trees.

Interviewing soon? We've partnered with Hired so that \times companies apply to you instead of you applying to them. See how \odot



! Report an Issue

