Hash Tables

09114319: Data Structures and Algorithms

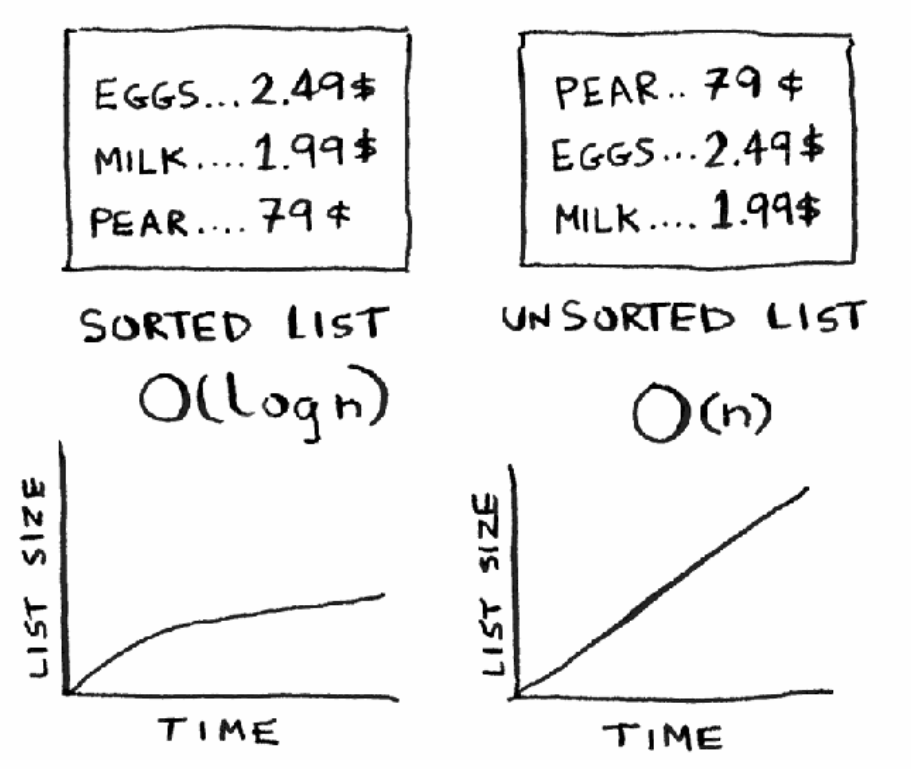
Ratthaprom PROMKAM, Dr. rer. nat.

Department of Mathematics and Computer Science, RMUTT

Outline

- Introduction to Hash Tables
- Hash Functions
- Hash Table Operations
- Applications of Hash Tables
- Handling Collisions
- Performance and Load Factor
- Recap

Finding the Price of an Item



- Imagine you are in a supermarket and want to find the price of an item.
- You could look through a long printed list, but this takes time.
- Even with **binary search**, which is efficient, the search time is $O(\log n)$.

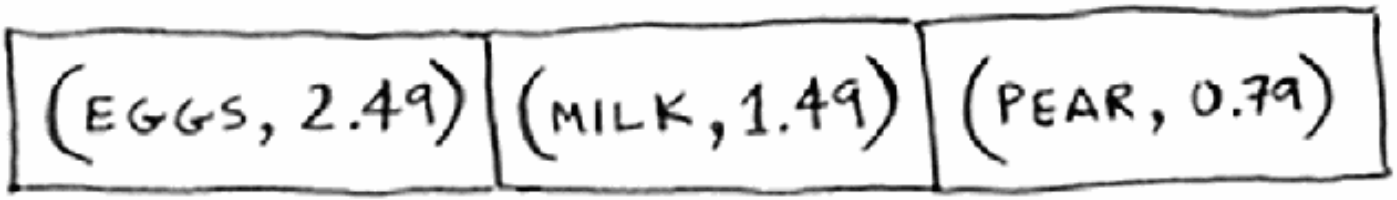
Introducing Maggie

# OF ITEMS IN THE BOOK	SIMPLE SEARCH	BINARY SEARCH	MAGGIE
	O(n)	O(log n)	O(1)
100	10 sec	1 sec	INSTANT
1000	1.6 min	1 sec	INSTANT
10000	16.6 min	2 sec	INSTANT

- Meet **Maggie**, our smart companion who remembers the price of every item.
- When you ask Maggie for a price, she instantly gives you the answer in $O(1)$ time!
- In this chapter, we learn how to build our own **Maggie** using **hash tables**.

Introduction to Hash Tables

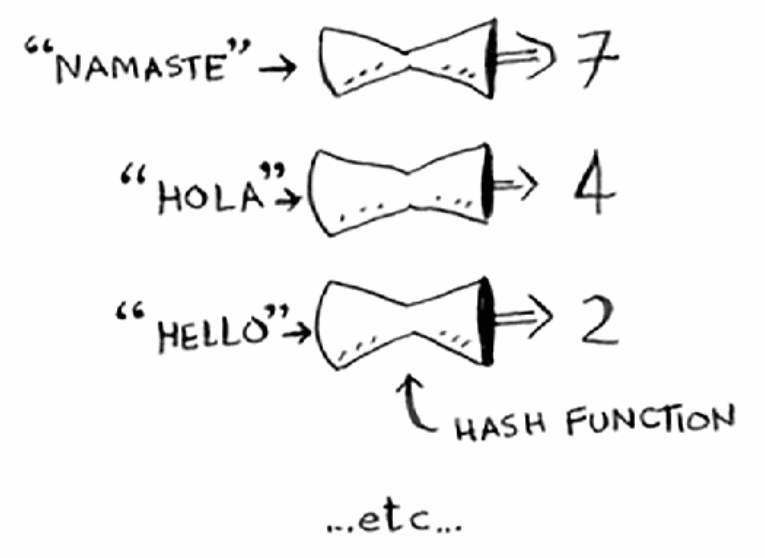
- A **hash table** is a data structure that provides fast lookups using a key-value mapping.



- Searching in a sorted array takes $O(\log n)$, but hash tables can achieve $O(1)$ on average.
- They are widely used in databases, caching, and indexing.

Hash Functions

A **hash function** maps keys to numerical indices.

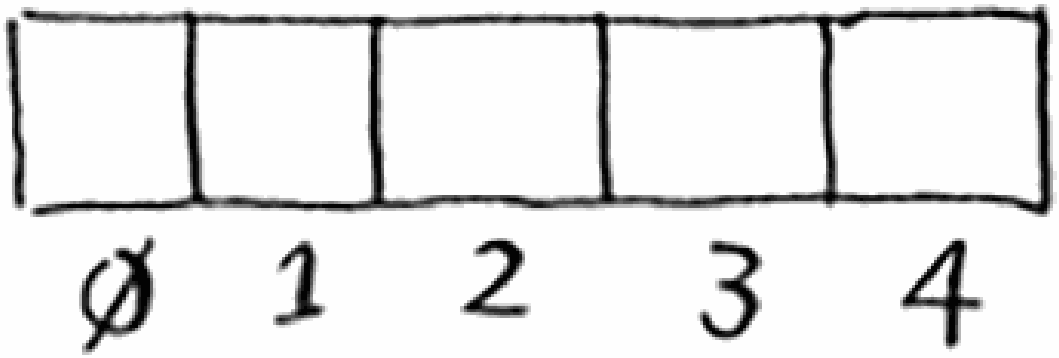


Requirements for a hash function:

Consistency	Same input always produces the same output.
Uniformity	Should distribute keys evenly across the table
Valid indices	The output should be within array bounds.

Example: Supermarket's Item prices

- Start with an empty array:



Example: Supermarket's Item prices

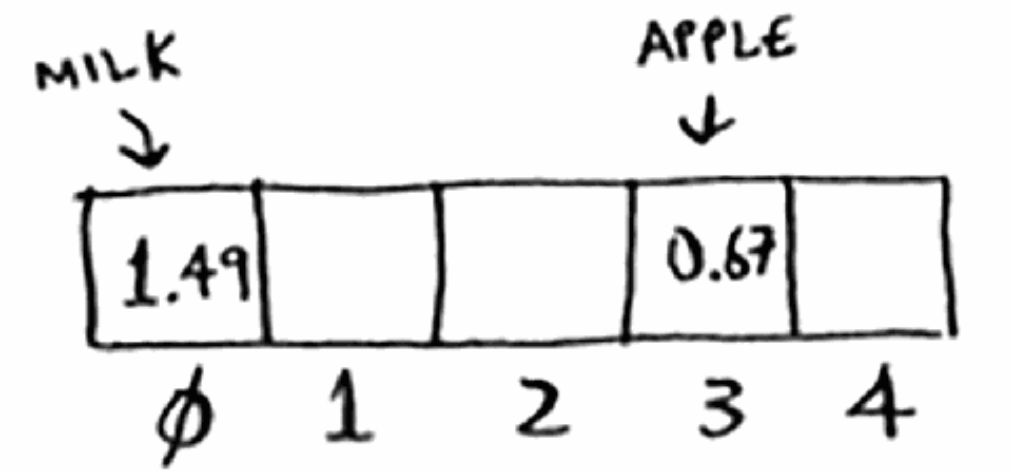
✎ Feed "apple" into the hash function.

So let's store the price of an apple at index 3 in the array.

Example: Supermarket's Item prices

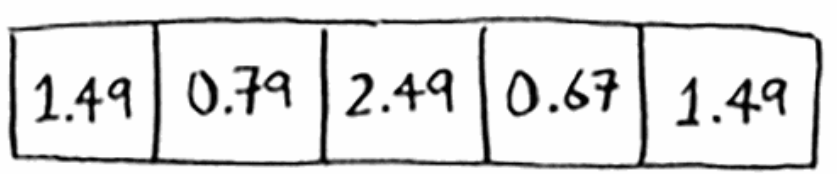
✎ Let's add milk. Feed "milk" into the hash function.

Let's store the price of milk at index 0.



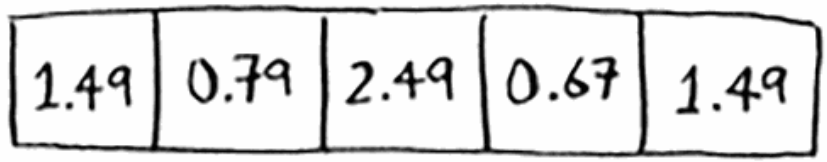
Example: Supermarket's Item prices

- Keep going, and eventually the whole array will be full of prices.

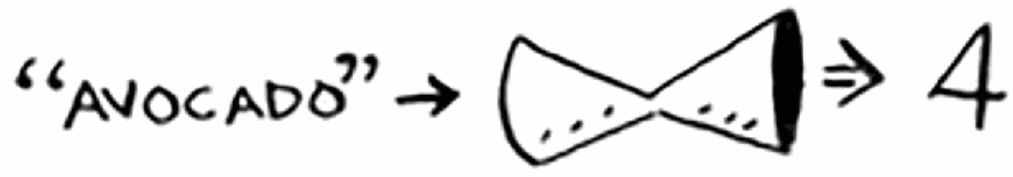


Example: Supermarket's Item prices

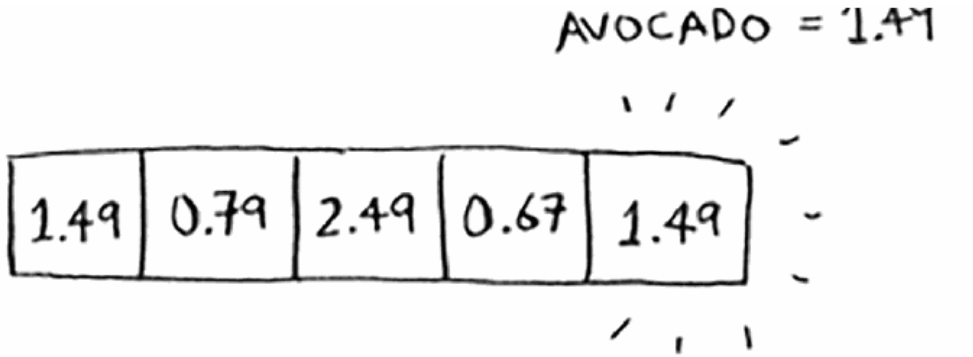
- Keep going, and eventually the whole array will be full of prices.



- Hey, what's the price of an avocado?. Just feed "avocado" into the hash function



It tells you that the price is stored at index 4. And sure enough, there it is.



Example: Python Dictionaries

Many good language will have an implementation for hash tables.
Python has hash tables; they're called **dictionaries**.


```python
hash_table = {}  # Dictionary (hash table)
hash_table["apple"] = 0.67
hash_table["milk"] = 1.49
hash_table["avocado"] = 1.49

print(hash_table["avocado"])  # Output: 1.49
```

Exercise

Which of these hash functions are good?

A. $f(x) = 1$ (Return 1 of any inputs)

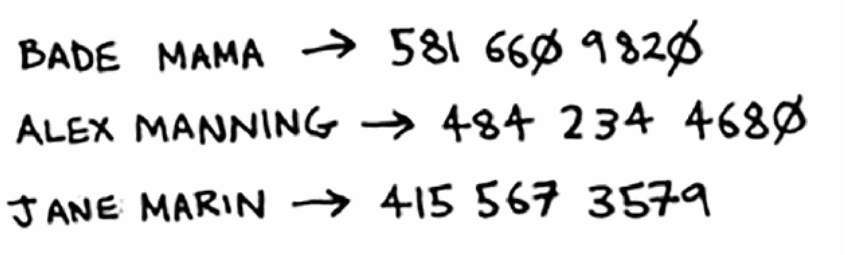
B. $f(x) = \text{rand}(\,)$ (Return a random value every time)

C. $f(x) = \text{next\_empty\_slot}(\,)$ (Return the index of the next slot in the hash table)

D. $f(x) = \text{len}(x)$ (Return the length of the string x)

Applications of Hash Tables: Phonebook

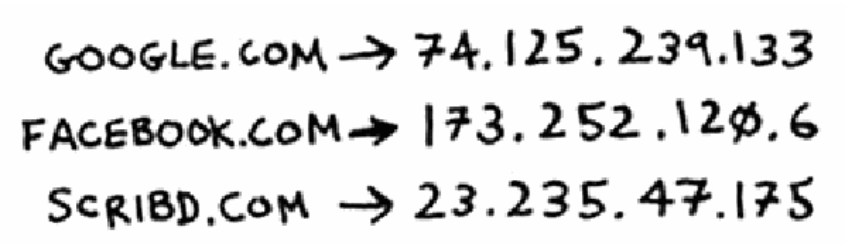
Phonebook: Map names to phone numbers.




```
1 phonebook = {}
2 phonebook["jenny"] = 867530
3 phonebook["emergency"] = 911
4
5 print(phone_book["jenny"]) # Output: 867530
```

Applications of Hash Tables: DNS Resolution

DNS Resolution: Map web addresses to IP addresses.



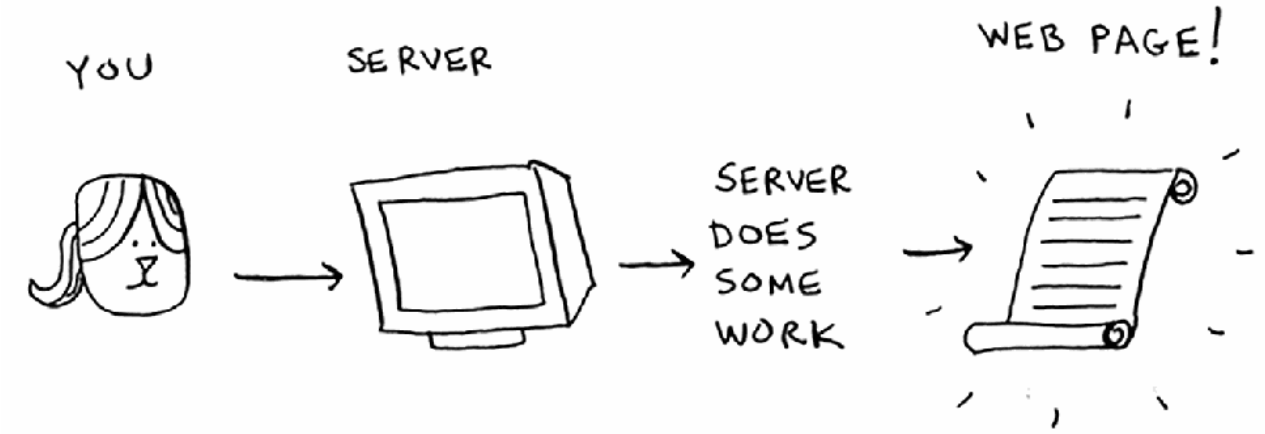

```
1 dns = {}
2 dsn['google.com'] = '74.125.239.133'
3 dns['facebook.com'] = '173.252.120.6'
4 dns['scribd.com'] = '23.235.47.175'
5
6 print(dns["google.com"])
```

Applications of Hash Tables: Preventing duplicate entries

```python
def check_voter(hash_table, name):
    if voted.get(name):
        print("Kick them out!")
    else:
        hash_table[name] = True
        print("Let them vote!")
```

```python
voted = {}
check_voter(voted, "Alice")  # Let them vote!
check_voter(voted, "Bob")    # Let them vote!
check_voter(voted, "Alice")  # Kick them out!
```

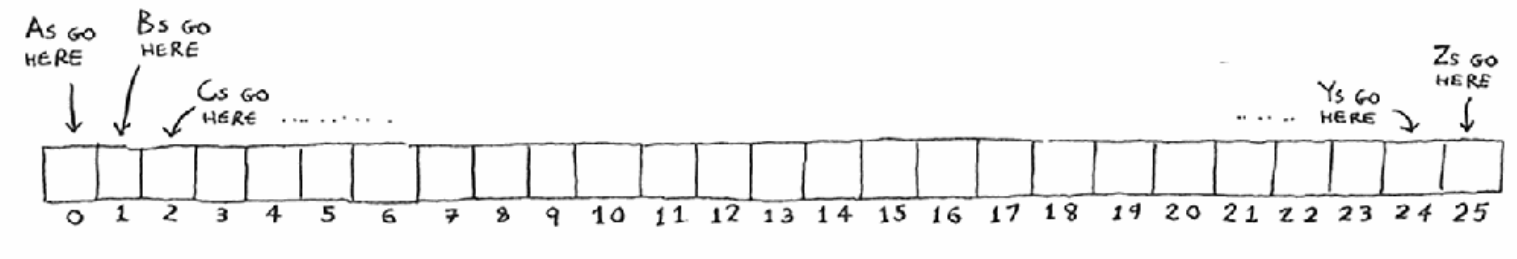
Applications of Hash Tables: Caching Web Pages




```python
def get_page(hash_table, url):
    if hash_table.get(url):
        return hash_table[url]  # Return cached
    else:
        data = fetch_data_from_server(url)
        hash_table[url] = data  # Save to cache
        return data
```

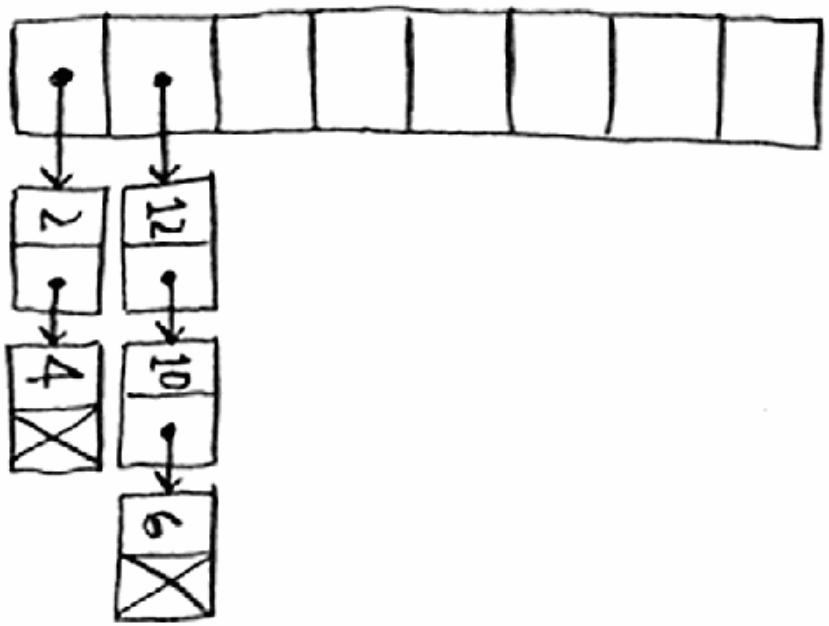
Handling Collisions

✎ **Collisions occur** when two keys hash to the same index.



Handling Collisions

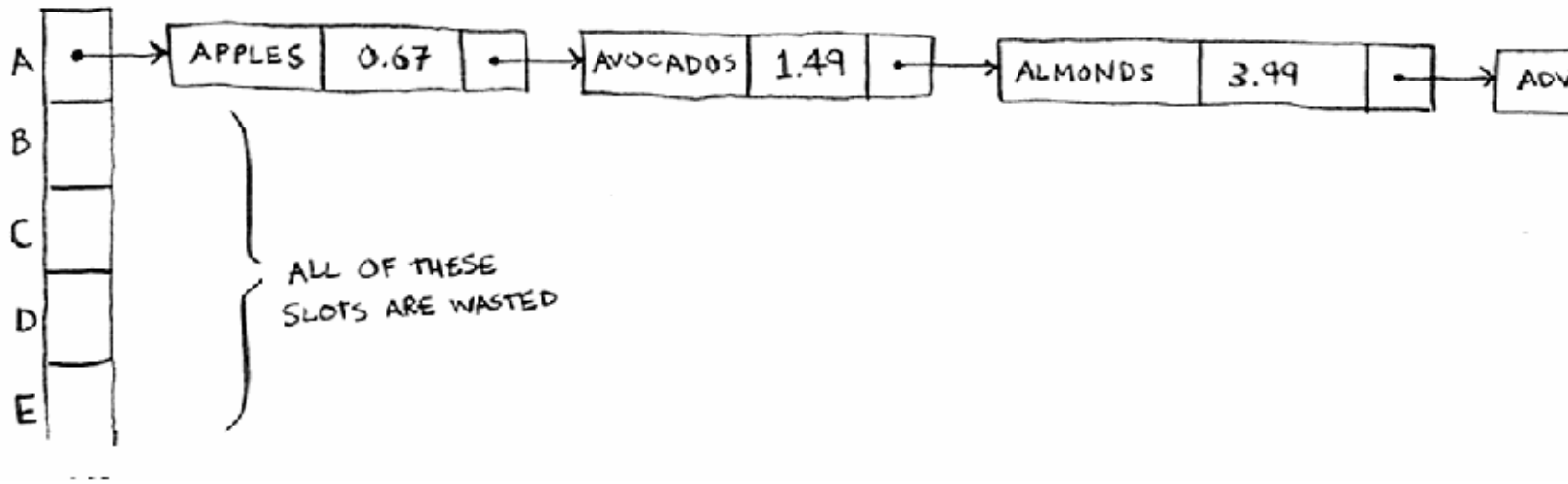
- Strategies to resolve collisions:
  - **Chaining**: Store multiple items at the same index using linked lists.



  - **Open Addressing**: Find another available index.

Example: Chaining for Collision Resolution

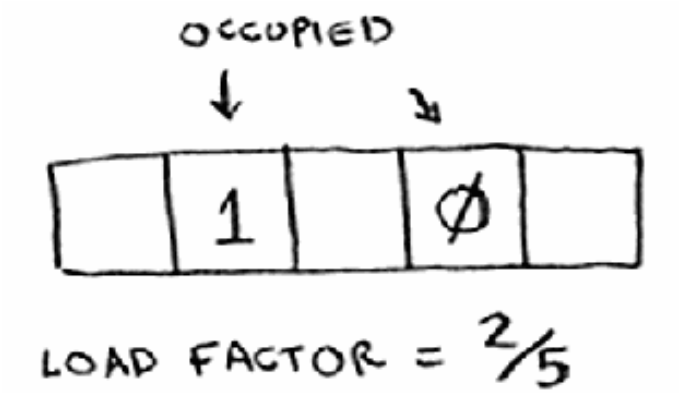
- If "apple" and "avocado" hash to the same slot, store them as a linked list.



- Lookup requires searching within the list at that slot.

Performance of Hash Tables

- **Best case:** $O(1)$ for lookups, inserts, and deletes.
- **Worst case:** $O(n)$ when all keys map to the same index.
- **Load Factor:** entries/slots.



- If load factor exceeds the threshold, resize the table.

Recap

- Hash tables store key-value pairs with fast lookups.
- Collisions are handled using chaining or open addressing.
- Hash tables are used in caching, lookup operations, and duplicate checking.
- Performance is $O(1)$ on average but can degrade to $O(n)$ in the worst case.