Search Algorithms

Searching Algorithms retrieve a specific element from within a data set.

The 3 Types of Search Algorithms

Search algorithms can be classified based on their mechanism of searching into three types:

1. **Linear** (Checks every element in the dataset for the one associated with the desired element in a linear fashion.)
2. **Binary** (Is used in only sorted data sets and repeatedly targets the centre element of the data set which divides the search space in half.)
3. **Hashing** (Directly maps keys to records based on a hash function)

# 1. Linear Search

In a linear search, you iterate through every element in a data set and compare it to the target number.

If your comparison finds a match, the number is in the list. If your algorithm ends without finding a match, the number is not in the list.

def linear\_search(array, n):

for i in array:

if i == n:

return True

else:

return False

#Example

a\_list = [1, 8, 32, 91, 5, 15, 9, 100, 3]

print(linear\_search(a\_list, 91))

>> True

## When to Use a Linear Search

You should consider using a linear search when your data is not sorted.

## Linear Search Time Complexity

A linear search’s time complexity in the worst-case scenario is O(n). In a list of 10 items, your algorithm will take 10 steps.

The best-case scenario for a linear search is O(1) because the item you are looking for could be the first item in the list, so your algorithm will take only one step because it stops as soon

as it finds a match.

## Using ‘in’ keyword instead of creating a linear search

When you are programming in the real world, instead of writing your own linear search, you can use Python’s built- in keyword ‘in’. Here is how to perform a linear search on a list of numbers using Python’s ‘in’ keyword:

unsorted\_list = [1, 45, 4, 32, 3]

print(45 in unsorted\_list)

>> True

You can also use a linear search to find characters in strings. In Python, you can search for a character in a string using a linear search like this:

print('a' in 'apple')

# 2. Binary Search

A binary search searches for an element in a data set by dividing it into halves.

A binary search takes O(log n) time.

A binary search is a faster algorithm for searching as compared to linear search. However, it only works when your data set is sorted.

## Binary Search Flow

1. Retrieves the middle element and then compares it to the target element.
2. If it is lower than the target then it will keep only the elements in the data set before it and then repeat step 1.
3. If its higher than the target then it will keep only the elements in the data set after it and then repeat step 1.
4. The binary search ends once the target element is found or if it has searched the whole data set and did not find the element.

def binary\_search(a\_list, n):

first = 0

last = len(a\_list) - 1

while last >= first:

mid = (first + last) // 2

if a\_list[mid] == n:

return True

else:

if n < a\_list[mid]:

last = mid - 1

else:

first = mid + 1

return False

## When to Use a Binary Search

A binary search takes O(log n) time. It is more efficient than a linear search because you don’t have to search an entire list. Even if you have unsorted data, sometimes it is worth sorting it to take advantage of a binary search.

## Using ‘bisect\_left’ keyword instead of creating a binary search

Use bisect\_left from the bisect module, which finds the index of an existing element in

a sorted list using a binary search

from bisect import bisect\_left

sorted\_fruits = ['apple', 'banana', 'orange', 'plum']

bisect\_left(sorted\_fruits, 'banana')

Linear Search Vs Binary Search

| **Linear Search** | **Binary Search** |
| --- | --- |
| In linear search input data need not to be in sorted. | In binary search input data need to be in sorted order. |
| It is also called sequential search. | It is also called half-interval search. |
| The time complexity of linear search O(n). | The time complexity of binary search O(log n). |
| Multidimensional array can be used. | Only single dimensional array is used. |
| Linear search performs equality comparisons | Binary search performs ordering comparisons |
| It is less complex. | It is more complex. |
| It is very slow process. | It is very fast process. |

# 3. Hashing (Hash Table: The Fastest Search)

Hashing is the process of transforming a given input (like an integer or string) into another value.

A key-value pair consists of two pieces of data mapped together: a key and a value.

1. The key is used to retrieve the value.
2. The value is any type of data value.

A **hash table** is a data structure that stores key and value pairs inside an associative array, whereby a mathematical formula called a hash function determines the index where to store the key-value pairs inside the array.

The **hash function** works by taking a key, which can be a string or integer data type as input and processing this with a mathematical formula like a division, mid Square folding or multiplication method, to output an integer called a hash value, which serves as an index for the hash table array.

Hashing has a time complexity of O(n) in worst-case time.

It is thus the fastest method of searching and is hence better than linear and binary search.

Python implements dictionaries using hash tables thus, you would never actually need to implement your own hash table in Python

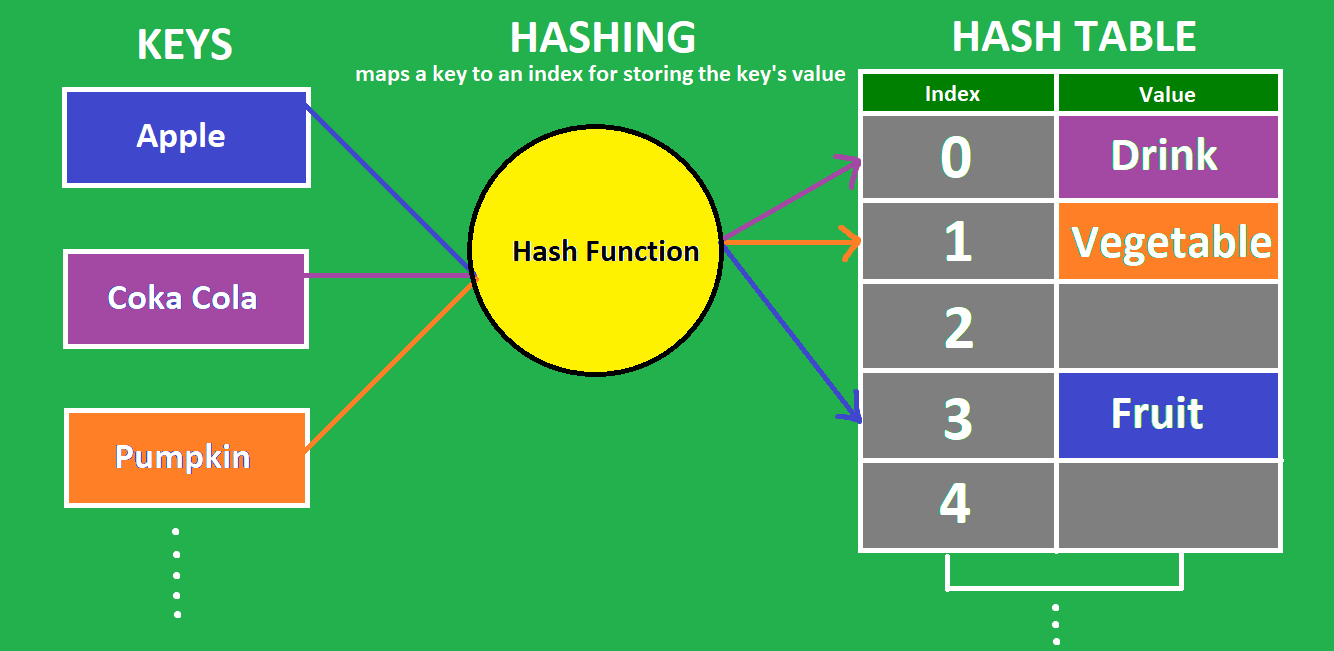
The reason **hash tables are so efficient for searching** is that to determine whether a piece of data is in a hash table, all you have to do is run your data through your hash function and check an array at that index, which is only one step.

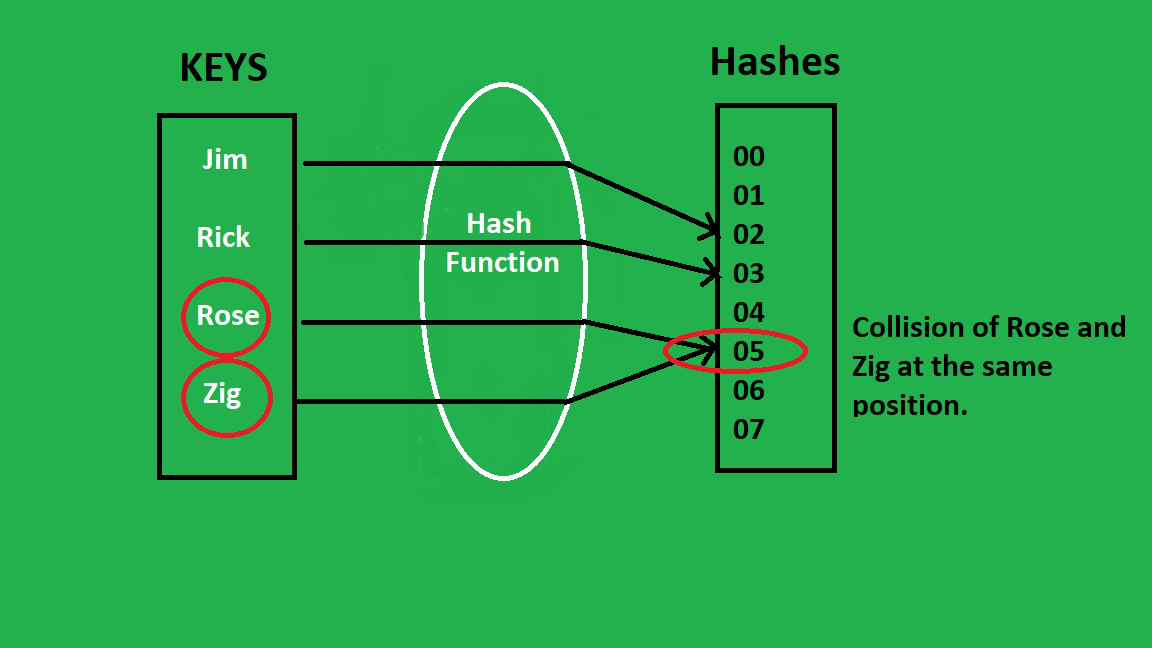
## Collisions

A collision is when two or more hash values are the same. This is a problem because when to hash values are the same it means two or more different keys are mapping to the same one index. When you are programming in Python, you don’t have to worry about collisions because dictionaries handle them for you.

There are two main ways of collision resolution techniques that can be used. You will unlikely be asked about details of collision resolution techniques in interviews:

1. Separate chaining - A linked list is used for each value, so that it stores all the collided items.
2. Open addressing - All entry records are stored in the bucket array itself. When a new entry has to be inserted, the buckets are examined, starting with the hashed-to slot and proceeding in some probe sequence, until an unoccupied slot is found.

Image of a hash table. Note how the keys are transformed by the hash function to become the index of the hash table. The key or index is then mapped to its value.



An example of collision, whereby the hash function makes two or more keys equal to the same hash value /or index.

## When to Use Hashing

You should consider using a hash table whenever you have a large amount of data and you need to access individual data items quickly.

## Hash Table in python

In Python, the Dictionary data types represent the implementation of hash tables.

Thus implement a hash table in python using a python dictionary. Refer to python dictionary notes.