1. Easy map interview questions

You might be tempted to try to read all of the possible questions and memorize the solutions, but this is not feasible. Interviewers will always try to find new questions, or ones that are not available online. Instead, you should use these questions to practice the **fundamental concepts** of maps.

As you consider each question, try to replicate the conditions you'll encounter in your interview. Begin by writing your own solution without external resources in a fixed amount of time.

If you get stuck, go ahead and look at the solutions, but then try the next one alone again. Don't get stuck in a loop of reading as many solutions as possible! We've analysed dozens of questions and selected ones that are commonly asked and have clear and high quality answers.

Here are some of the easiest questions you might get asked in a coding interview. These questions are often asked during the "phone screen" stage, so you should be comfortable answering them without being able to write code or use a whiteboard.

1.1 Two sum

- Text guide (Medium/Hyewon Cho)
- Video guide (NeetCode)
- Code example (LeetCode)

1.2 Roman to integer

- Text guide (Medium/Nerd For Tech)
- Video guide (Java Brains)
- Code example (LeetCode)

1.3 Majority element

- Text guide (LeetCode)
- Video guide (Kevin Naughton Jr.)
- Video guide (take u forward)
- Code example (LeetCode)

1.4 Happy number

- Text guide (Code Study Blog)
- Video guide (Kevin Naughton Jr.)
- Video guide (NeetCode)
- Code example (LeetCode)

1.5 Contains duplicate

- Text guide (Medium/Len Chen)
- Text guide (Medium/Bryam Vicente)
- Video guide (Nick White)
- Code example (LeetCode)

1.6 Valid anagram

- Text guide (LeetCode)
- Video guide (TECH DOSE)
- Code example (LeetCode)

1.7 Intersection of two arrays II

- Text guide (Medium/punitkmryh)
- Video guide (Fisher Coder)
- Code example (LeetCode)

1.8 First unique character in a string

- Text guide (LeetCode)
- Video guide (Kevin Naughton Jr.)
- Code example (LeetCode)

1.9 Isomorphic strings

- Text guide (Educative.io)
- Video guide (Amell Peralta)
- Code example (LeetCode)

1.10 Employee importance

- Text guide (LeetCode)
- Video guide (Naresh Gupta)
- Code example (LeetCode)

1.11 Contains duplicate II

- Text guide (Medium/Kevin Malazarte)
- Video guide (Kevin Naughton Jr.)
- Code example (LeetCode)

1.12 Word pattern

- Text guide (Yu’s Coding)

- Video guide (Nick White)
- Code example (LeetCode)

1.13 Longest palindrome

- Text guide (cheonhyangzhang)
- Video guide (Knowledge Center)
- Video guide (Nick White)
- Code example (LeetCode)

1.14 Next greater element I

- Text guide (TutorialCup)
- Video guide (Nick White)
- Code example (LeetCode)

1.15 Keyboard row

- Text guide (Memogrocery)
- Code example (LeetCode)

1.16 Longest harmonious subsequence

- Text guide (Dev.to/seanpgallivan)
- Text guide (LeetCode)
- Video guide (Algorithms Made Easy)
- Code example (LeetCode)

2. Medium map interview questions

Here are some moderate-level questions that are often asked in a video call or onsite interview. You should be prepared to write code or sketch out the solutions on a whiteboard if asked.

2.1 Longest substring without repeating characters

- Text guide (RedQuark)
- Video guide (Michael Muinos)
- Code example (LeetCode)

2.2 Valid sudoku

- Text guide (GeeksForGeeks)
- Video guide (Nick White)
- Code example (LeetCode)

2.3 Group anagrams

- Text guide (Dev.to/Justin Bermudez)
- Video guide (NeetCode)
- Code example (LeetCode)

2.4 Longest consecutive sequence

- Text guide (LeetCode)
- Video guide (NeetCode)
- Code example (LeetCode)

2.5 LRU cache

- Text guide (LeetCode)
- Video guide (Back to Back SWE)
- Video guide (Byte by Byte)
- Code example (LeetCode)

2.6 Fraction to recurring decimal

- Text guide (cheonhyangzhang)
- Video guide (happygirlzt)
- Code example (LeetCode)

2.7 Top K frequent elements

- Text guide (GeeksForGeeks)
- Video guide (TECH DOSE)
- Code example (LeetCode)

2.8 Insert delete getRandom O(1)

- Text guide (Medium/Len Chen)
- Video guide (Michael Muinos)
- Video guide (TECH DOSE)
- Code example (LeetCode)

2.9 4Sum II

- Text guide (Medium/Haoyu Lei)
- Video guide (Naresh Gupta)
- Code example (LeetCode)

2.10 Copy list with random pointer

- Text guide (Techie Delight)
- Video guide (NeetCode)
- Video guide (Timothy H Chang)
- Code example (LeetCode)

2.11 Word break

- Text guide (Medium/Len Chen)
- Video guide (Knapsak)
- Video guide (NeetCode)
- Code example (LeetCode)

2.12 Find all anagrams in a string

- Text guide (LeetCode)
- Video guide (TECH DOSE)
- Code example (LeetCode)

2.13 Subarray sum equals K

- Text guide (Hello Koding)
- Video guide (Nick White)
- Video guide (TECH DOSE)
- Code example (LeetCode)

2.14 Task scheduler

- Text guide (Medium/Shymmy W. Garcia)
- Video guide (Kevin Naughton Jr.)
- Code example (LeetCode)

2.15 Partition labels

- Text guide (Medium/Gaurav Jain)
- Video guide (Kevin Naughton Jr.)
- Video guide (Nick White)
- Code example (LeetCode)

2.16 Repeated DNA sequences

- Text guide (cheonhyangzhang)
- Video guide (Kevin Naughton Jr.)
- Code example (LeetCode)

2.17 Bulls and cows

- Text guide (ProgramCreek)
- Video guide (Naresh Gupta)
- Code example (LeetCode)

3. Hard map interview questions

Similar to the medium section, these more difficult questions may be asked in an onsite or video call interview. You will likely be given more time if you are expected to create a full solution.

3.1 Minimum window substring

- Text guide (Medium/Algo Shaft)
- Video guide (NeetCode)
- Video guide (Michael Muinos)
- Code example (LeetCode)

3.2 Word ladder

- Text guide (GeeksForGeeks)
- Video guide (NeetCode)
- Video guide (Nick White)
- Code example (LeetCode)

3.3 Word break II

- Text guide (Educative.io)
- Video guide (babybear4812)
- Code example (LeetCode)

3.4 Max points on a line

- Text guide (Medium/Hary Krishnan)
- Video guide (happygirlzt)
- Code example (LeetCode)

3.5 Substring with concatenation of all words

- Text guide (RedQuark)
- Video guide (Coding Simplified)
- Code example (LeetCode)

3.6 Word ladder II

- Text guide (Medium/spylogsster)

- Video guide (TECH DOSE)
- Code example (LeetCode)

3.7 Palindrome pairs

- Text guide (Medium/Hary Krishnan)
- Video guide (Coding Decoded)
- Code example (LeetCode)

3.8 LFU cache

- Text guide (GeeksforGeeks)
- Video guide (happygirlzt)
- Code example (LeetCode)

3.9 Random pick with blacklist

- Text guide (Tutorialspoint)
- Video guide (babybear4812)
- Code example (LeetCode)

3.10 Number of atoms

- Text guide (LeetCode)
- Video guide (Happy Coding)
- Code example (LeetCode)

3.11 Bus routes

- Text guide (LeetCode)
- Video guide (code_report)
- Code example (LeetCode)

3.12 Maximum frequency stack

- Text guide (GeeksForGeeks)
- Video guide (Algorithms Made Easy)
- Code example (LeetCode)

3.13 Subarrays with K different integers

- Text guide (Medium/Nitish Chandra)
- Video guide (Happy Coding)
- Code example (LeetCode)

3.14 Grid illumination

- Text guide (Tutorialspoint)
- Video guide (Shivam Patel)
- Code example (LeetCode)

3.15 Number of submatrices that sum to target

- Text guide (Dev.to/seanpgallivan)
- Video guide (Algorithms Made Easy)
- Code example (LeetCode)

3.16 Minimum operations to make a subsequence

- Text guide (Dev.to/seanpgallivan)
- Video guide (Happy Coding)
- Code example (LeetCode)

3.17 Maximum equal frequency

- Text guide (shengqianliu)
- Video guide (happygirlzt)
- Code example (LeetCode)

4. Map basics

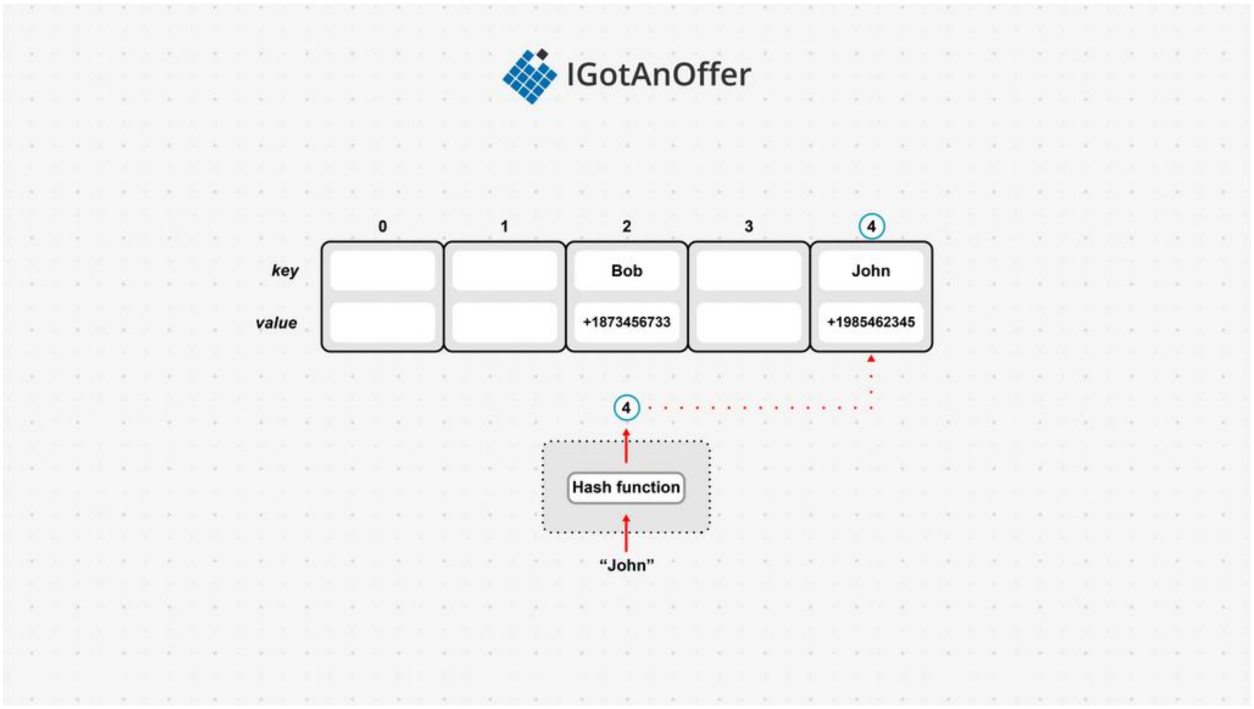
In order to crack the questions above and others like them, you'll need to have a strong understanding of maps, how they work, and when to use them. Let's get into it.

4.1 What are maps?

A map is a data structure that allows us to access a value by key. This is in contrast to an array that allows us to access a value by index. A common kind of map is a hash map (also called a hash table), which stores keys along with associated values (for example, a telephone directory which associates phone numbers with names).

To store data in a hash map, the key is run through a hash function, which can take any input and reduce it to a bounded space. Because a hash function will always produce the same output given the same input, hash maps use the hash value to decide where in the hash map the key should be stored. This means that we can usually retrieve any key or check that exists in O(1) time, as opposed to an array, which would need O(n) time to search for a given key.

Because keys are indexed based on their hash value, hash maps can have gaps. In the example below, "John" is sent through a hash function which outputs the value 4. This means that we store "John" at index 4 of the underlying array, and associate it with John's telephone number.

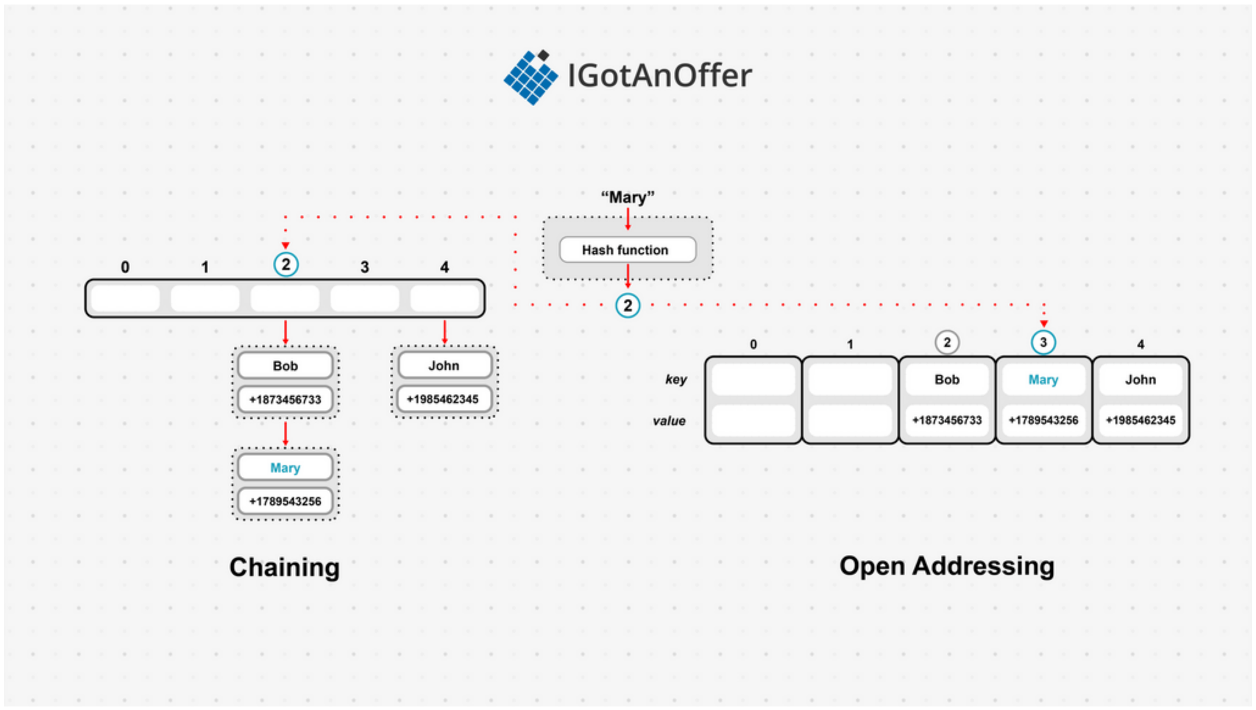


Like arrays, hash maps have a constant access time. But unlike arrays, inserting and deleting is also constant for the most part. Performance only declines should the underlying array reach capacity and need to be resized. Resizing usually creates a new array with double the capacity, and all the elements will need to be hashed again to find their index in the new array.

Since hash functions need to map from an infinite input space to a finite output space, there is always the risk of collisions (two different inputs being hashed to the same value). Most hash map implementations have some kind of collision resolution to work around this. The two main strategies for dealing with collisions are:

- **Open addressing** - If the desired spot is already filled, overflow the value to the next available slot.
- **Chaining** - Every slot is a reference to a linked list, so we can find the correct (ideally short) linked list in O(1) time but then need to traverse it to find the actual value.

In the example below, we are trying to put the key "Mary" into our hash map. The hash function returns the value 2, so we should store this data at the 2nd index, but this is already taken by "Bob." Using the chaining strategy, we simply add a new item to the linked list at index 2. Using open addressing, we move onto the next empty space. In this case, slot 3 is available, so we can store our data there. If we have another collision for index 2, we would have to wrap around and store it at index 0, after checking if 4 is free.



This map implementation uses a hash function, so it is generally referred to as a hash map or hash table. But other implementations also exist to map a key to a value and allow for efficient lookup. For example, we can use a tree to create what is called a tree map.

With tree maps, we don't need a hash function. Instead, we store our keys and values in a tree structure, often a red-black tree. This means that lookups take slightly longer, O(log n), but we get an added advantage in that our keys are stored in order, which is not the case for hash maps.

The underlying implementation of a tree map usually uses a red-black tree for extra efficiency, but a binary search tree can also be used.

4.1.1 Types of maps (Java, Python, C++)

Java provides the "Map" interface with both a concrete "HashMap" and "TreeMap" implementation. HashMap is not thread safe and allows keys to be of type "null." The special type "Hashtable" is thread safe and only allows non-null keys. TreeMap's implementation is based on the red-black tree structure to guarantee log(n) time costs.

C++ has the "map" type from the Standard Template Library (STL). It is a tree map implementation using a red-black tree. C++ 11 introduced an "unordered_map," which is a hash map.

Python has the "dict" type, which is also a hash map implementation. Unlike strongly typed languages, a Python dictionary can contain a mix of different data types, such as strings and integers.

4.1.2 How maps store data

Maps are built on top of other data structures, such as arrays, linked lists, and trees, depending on the exact kind of map. The underlying data structure has a finite number of slots, often referred to as a "bucket," where key-value pairs can be stored.

Many maps are unordered. This means that they do not store the data in a useful order, so they are often unsuitable in cases where you expect to iterate over the data. However, ordered maps, such as the tree map, also exist.

4.1.3 How maps compare to other data structures

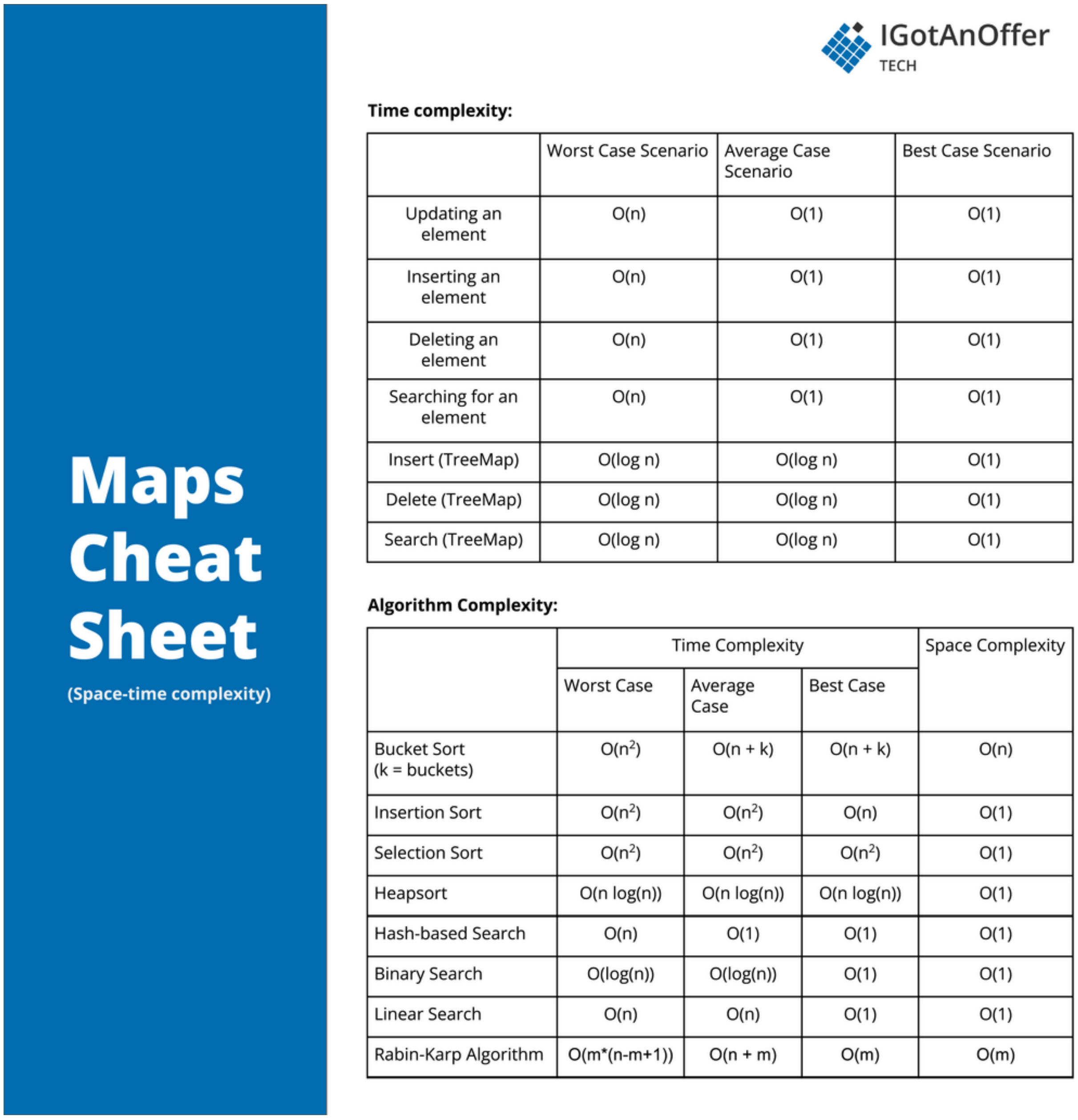
Maps are a unique data structure, used when we want to index values by a key. Examples include storing names and phone numbers, usernames and passwords, or letters and frequency counts.

Maps are more complicated than other data structures such as arrays or linked lists (though some implementations may make use of these). Unless you need the specific properties of maps, such as efficient lookup by value, it is better to use simpler data structures as they have less overhead.

However, because of the unique properties of maps (often O(1) look up time), they can frequently be used to make code more efficient. For example, if you often need to check if a specific value is contained in an array, converting it to a map can drastically reduce the runtime of your code.

Like arrays, hash maps can have some wasted space. In the worst case, it is possible for a hash map's indices to go unfilled when no key hashes to them. This is generally caused by a poorly designed hash function. In extreme cases, a hash map using chaining can end up with all its data stored in a single chain, meaning it has the same properties as a linked list with some extra overhead.

5. Maps cheat sheet



IGotAnOffer TECH

Maps Cheat Sheet

(Space-time complexity)

Time complexity:

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Updating an element	O(n)	O(1)	O(1)
Inserting an element	O(n)	O(1)	O(1)
Deleting an element	O(n)	O(1)	O(1)
Searching for an element	O(n)	O(1)	O(1)
Insert (TreeMap)	O(log n)	O(log n)	O(1)
Delete (TreeMap)	O(log n)	O(log n)	O(1)
Search (TreeMap)	O(log n)	O(log n)	O(1)

Algorithm Complexity:

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Bucket Sort (k = buckets)	O(n^2)	O(n + k)	O(n + k)	O(n)
Insertion Sort	O(n^2)	O(n^2)	O(n)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Hash-based Search	O(n)	O(1)	O(1)	O(1)
Binary Search	O(log(n))	O(log(n))	O(1)	O(1)
Linear Search	O(n)	O(n)	O(1)	O(1)
Rabin-Karp Algorithm	O(m*(n-m+1))	O(n + m)	O(m)	O(m)

You can download the cheat sheet here.