1. Easy heap 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 heaps.

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 Kth largest element in a stream

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

1.2 Last stone weight

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

1.3 The K weakest rows in a matrix

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

2. Medium heap 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 Kth largest element in an array

- Text guide (GeeksForGeeks)
- Video guide (Back to Back SWE)

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

2.2 Meeting rooms II

- Text guide (Zirui)
- Video guide (NeetCode)
- Video guide (Kevin Naughton Jr.)
- Code example (Seanforfun)

2.3 Top K frequent elements

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

2.4 Kth smallest element in a sorted matrix

- Text guide (Medium/Ayoub Omari)
- Video guide (alGOds)
- Code example (LeetCode)

2.5 Task scheduler

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

2.6 Super ugly number

- Text guide (GeeksForGeeks)
- Video guide (Ren Zhang)
- Code example (LeetCode)

2.7 Find K pairs with smallest sums

- Text guide (GeeksForGeeks)
- Video guide (Yusen Zhang)
- Code example (LeetCode)

2.8 Sort characters by frequency

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

2.9 Split array into consecutive subsequences

- Text guide (LeetCode)

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

2.10 Top K frequent words

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

2.11 Network delay time

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

2.12 Reorganize string

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

2.13 Cheapest flights within K stops

- Text guide (libaoj)
- Video guide (Persistent Programmer)
- Code example (LeetCode)

2.14 K closest points to origin

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

2.15 Maximum number of events that can be attended

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

2.16 Longest continuous subarray with absolute diff less than or equal to limit

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

2.17 Path with minimum effort

- Text guide (Dev.to/seanpgallivan)

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

2.18 Furthest building you can reach

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

2.19 Maximum number of eaten apples

- Text guide (GeeksForGeeks)
- Video guide (Lead Coding by FRAZ)
- Code example (LeetCode)

2.20 Find Kth largest XOR coordinate value

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

2.21 Number of restricted paths from first to last node

- Text guide (walkccc)
- Video guide (Lead Coding by FRAZ)
- Code example (LeetCode)

2.22 Maximum average pass ratio

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

2.23 Number of orders in the backlog

- Text guide (Programmerall)
- Video guide (Cherry Coding)
- Code example (LeetCode)

2.24 Single-threaded CPU

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

2.25 Seat reservation manager

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

- Code example (LeetCode)

2.26 Process tasks using servers

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

2.27 Get biggest three rhombus sums in a grid

- Text guide (LeetCode)
- Video guide (Happy Coding)
- Code example (kamyu104)

2.28 The number of the smallest unoccupied chair

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

2.29 Remove stones to minimize the total

- Text guide (Chowdera)
- Video guide (CodeinDepth)
- Code example (LeetCode)

3. Hard heap 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 Merge K sorted lists

- Text guide (Medium/Ruslan Rakhmedov)
- Video guide (Back to Back SWE)
- Code example (LeetCode)

3.2 The skyline problem

- Text guide (Brian Gordon)
- Video guide (Tushar Roy)
- Video guide (mCoding)
- Code example (Medium/Dimka Maleev)

3.3 Find median from data stream

- Text guide (Medium/Kode Shaft)
- Video guide (NeetCode)

- Code example (LeetCode)

3.4 Trapping rain water II

- Text guide (Medium/Akanksha)
- Video guide (Sam Shen)
- Code example (LeetCode)

3.5 Sliding window median

- Text guide (Aaronice)
- Video guide (LeetCode Learning)
- Video guide (Eric Programming)
- Code example (LeetCode)

3.6 IPO

- Text guide (LeetCode)
- Video guide (Ren Zhang)
- Code example (hacker78)

3.7 Course schedule III

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

3.8 Smallest range covering elements from K lists

- Text guide (GeeksForGeeks)
- Video guide (IDeserve)
- Code example (LeetCode)

3.9 Swim in rising water

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

3.10 Minimum cost to hire K workers

- Text guide (shengqianliu)
- Video guide (AlgoCademy)
- Video guide (Shiran Afergan)
- Code example (LeetCode)

3.11 K-th smallest prime fraction

- Text guide (libaoj)

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

3.12 Minimum number of refueling stops

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

3.13 Reachable nodes in subdivided graph

- Text guide (libaoj)
- Text guide (LeetCode)
- Video guide (Coding Decoded)
- Code example (LeetCode)

3.14 Construct target array with multiple sums

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

3.15 Maximum performance of a team

- Text guide (Dev.to/seanpgallivan)
- Video guide (babybear4812)
- Code example (LeetCode)

3.16 Find the Kth smallest sum of a matrix with sorted rows

- Text guide (Tutorialspoint)
- Video guide (Ren Zhang)
- Video guide (alGOds)
- Code example (LeetCode)

3.17 Max value of equation

- Text guide (LeetCode)
- Video guide (Lead Coding by FRAZ)
- Code example (Walkccc)

3.18 Minimize deviation in array

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

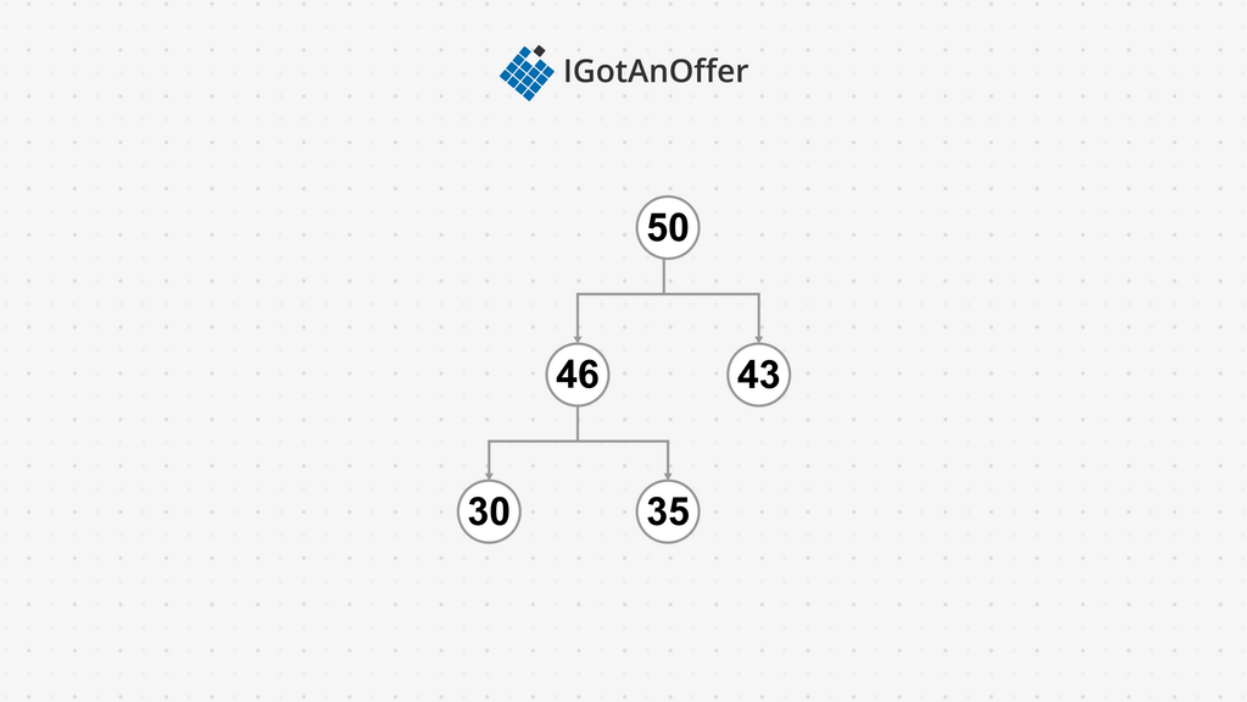
4. Heap basics

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

4.1 What are heaps?

A heap is a tree-based data structure that implements a priority queue. A priority queue functions much like a regular stack or queue, except that each element has a priority. It is this priority that determines which element is returned when dequeuing from the priority queue, rather than the order in which the element was added. This is useful for applications like a hospital queue where we want to serve the patient with the highest priority first.

A heap must satisfy the **heap property**. The heap property states that a node must have a value, or key, greater than or equal to (≥) the value of its child node for a *max-heap*, and less than or equal to (≤) to the value of its child node for a *min-heap*. The relative ordering of the child nodes does not matter, i.e. it is not required that the smaller child be on the left or right of the parent node.



Heaps are commonly implemented as binary trees, known as binary heaps. Binary heap trees must be almost complete. An almost complete binary tree has each level completely filled except for the very last level.

To insert or add to a binary heap, the new element is added to the last level at the first available spot. After adding the new element, the heap property is checked. If it is violated, the new element is swapped upwards with its parent node and the heap property checked again. More

than one swap may be necessary to satisfy the heap property, in some cases up to the root node. This is called heapify-up, or a swim operation.

Popping from a heap returns the root element. The root is replaced by the last element in the heap, violating the heap property. In order to restore the heap property, a heapify-down, or sink operation, is performed, in which the root is compared to its largest child. If the order is incorrect, the nodes are swapped. This action is repeated for each child node swapped, until nothing is swapped or a leaf is reached.

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

Java provides the "PriorityQueue" class, which is based on the heap structure. The priority queue relies on "natural order," making it a min-heap implementation. But passing a custom comparator to the constructor allows you to implement it as a max-heap. All items added to the priority queue class must be comparable, so a *null* element cannot be added.

Various C++ helper functions allow you to make an iterator operate as a heap. Among them are:

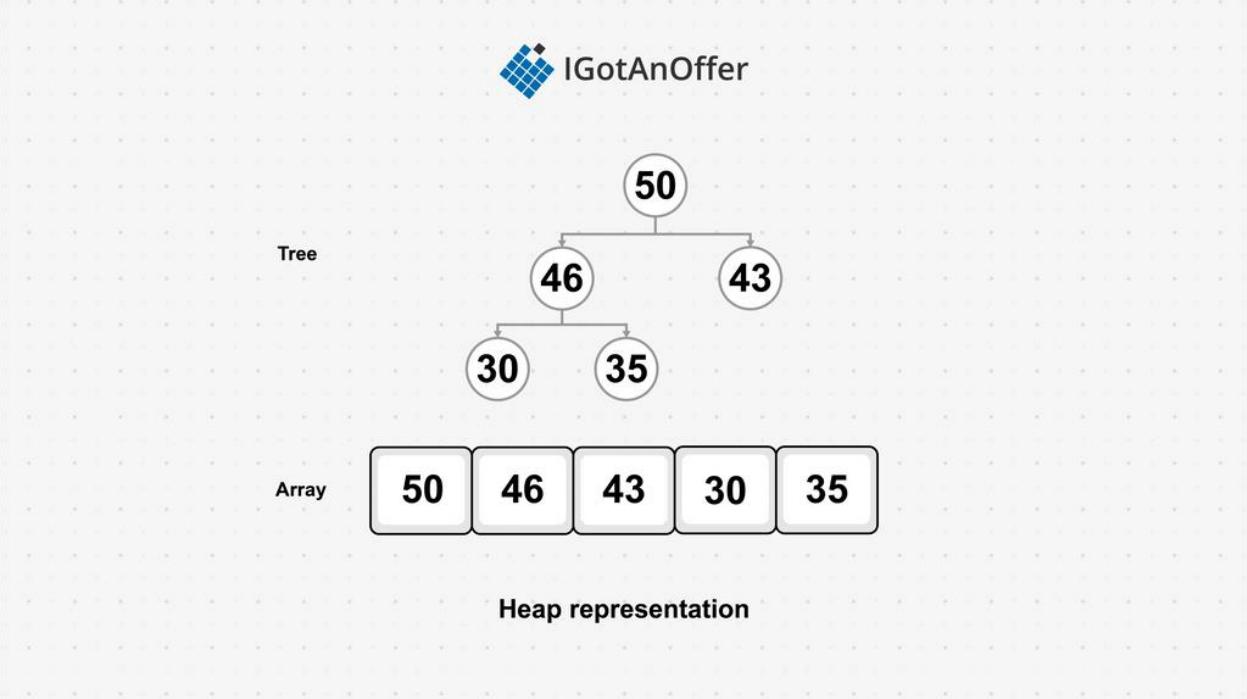
- "make_heap," which rearranges the elements in an iterator to form a heap;
- "push_heap," which adds a new element to a heap and restores the heap property; and
- "pop_heap," which removes the root element and restores the heap property.

Heaps in C++ are implemented as max-heap by default, but a custom comparator can be used to implement min-heaps. The "priority_queue" container adapter in C++ encapsulates these function calls on any container, like vector or deque containers.

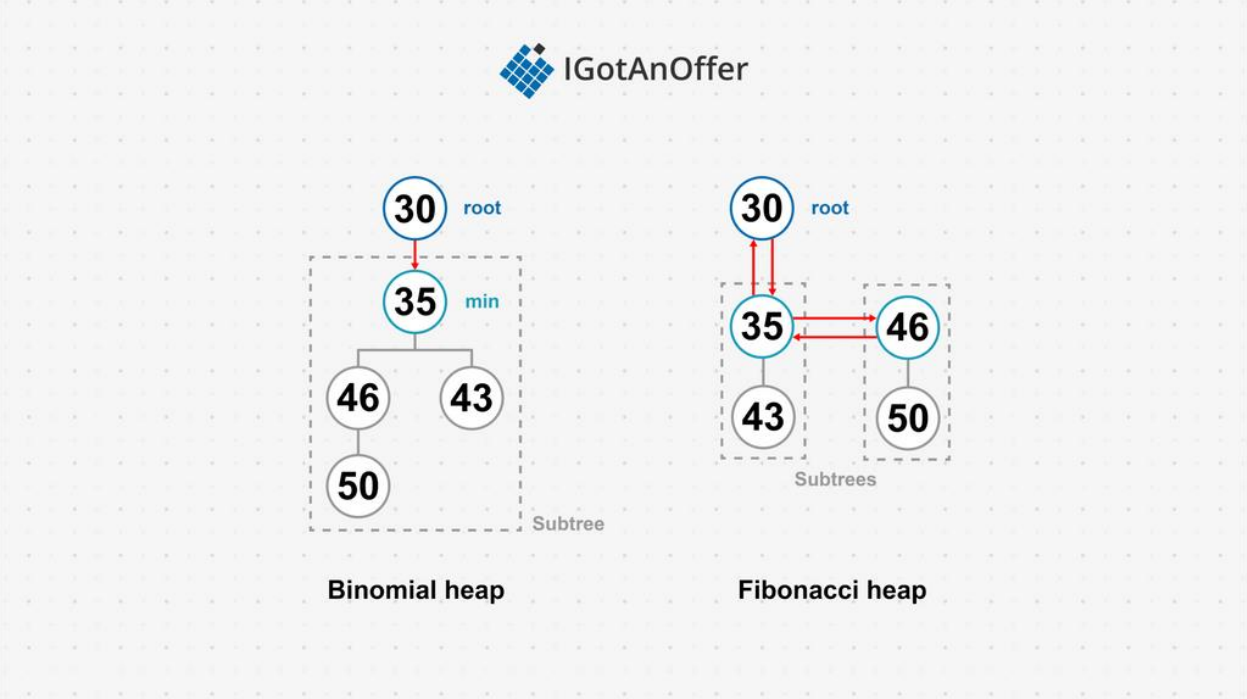
Python has the "heapq" type, which is a min-heap implementation.

4.1.2 How heaps store data

Heaps are usually implemented using an array representation of a binary tree. This requires mapping an almost complete binary tree as an array: the root of the tree is at arr[0], the left child for any parent *i* is at arr[2i + 1] and the right child at arr[2i + 2], as you can see in the diagram below. Java and Python use arrays for their heap implementations. C++ also uses arrays when using the vector container.



Variants of heap implementations also exist, two of which are binomial heaps and Fibonacci heaps. A binomial heap is a set of binomial trees, while a Fibonacci heap is a set of normal trees. In both variants, the subtrees must satisfy the min-heap property. This means that the root of each tree contains the minimum element for that tree, and the minimum root of all the trees will be the minimum value for the whole collection. The roots of subtrees are stored in a linked list in binomial heaps and in a doubly linked list in Fibonacci heaps, allowing for iteration over the roots. This iteration adds some time complexity, so both implementations also have a direct pointer to the minimal root.



4.1.3 How heaps compare to other data structures

Heaps are a subtype of priority queues, which allow quick access to the element with the highest priority. They are also similar to any sorted list, tree, or array. A sorted list has an insert cost of O(n); a sorted tree takes O(log n) to find the min/max; and a sorted array has a delete of O(n).

Heaps are also similar to stacks and regular queues, since most implementations have push, pop and peek methods. Heaps, however, are specialized for popping the element with the highest priority, instead of the first in, first out (FIFO) rule for queues and last in, first out (LIFO) rule for stacks.

5. Heaps cheat sheet

Heaps Cheat Sheet

(Space-time complexity)

IGotAnOffer TECH

Time complexity:

	Worst Case Scenario	Average Case Scenario	Best Case Scenario
Insert	O(log n)	O(logn)	O(1)
Delete	O(log n)	O(log n)	O(1)
Find min/max	O(1)	O(1)	O(1)
Search	O(n)	O(n)	O(1)
Insert (Fibonacci/Binomial)	O(log n)	O(1)	O(1)
Increase/Decrease key	O(log n)	O(log n)	O(1)
Extract min/max	O(log n)	O(log n)	O(log n)

Algorithm Complexity:

	Time Complexity			Space Complexity
	Worst Case	Average Case	Best Case	
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Smoothsort	O(n log(n))	O(n log(n))	O(n)	O(n)
Quick select	$O(n^2)$	O(n)	O(n)	O(1)
Linear Search	O(n)	O(n)	O(1)	O(1)
Dijkstra's shortest path	$O(V^2)$	O(E * log(V))	O(E * log(V))	O(V)