🎯 300 JavaScript Interview Questions Cheat Sheet

A cheat sheet covering 300 common JavaScript interview questions with answers and code demos.

> 💡 **Tip**
>
> Complete Dev Roadmap with SaaS Boilerplate at thedevspace.io/

1. What is a closure?

A closure is a function that retains access to variables from its outer scope after that scope has closed. Use closures for data privacy and simple factories.

```
function outer() {
  let x = 1;
  return () => x;
}
const fn = outer(); // fn() -> 1
```

2. Difference between var, let, and const?

`var` is function-scoped and can be redeclared; `let` and `const` are block-scoped and safer; `const` cannot be reassigned.

```
let a = 1;
const b = 2; // prefer let/const
```

3. What is hoisting?

Hoisting moves declarations to the top of their scope at runtime so some references exist before assignments. Initializations are not hoisted.

```
console.log(a); // undefined
var a = 5;
```

4. What is the difference between == and ===?

`==` compares with type coercion, `===` is strict and checks type and value; prefer `===` for predictable equality.

```
0 == "0"; // true
0 === "0"; // false
```

5. What is event delegation?

Attach a single listener on a parent to handle child events, useful for dynamic lists and better performance.

```
document.body.addEventListener("click", (e) => {
  if (e.target.matches(".item")) handle(e.target);
});
```

6. What is the 'this' keyword?

`this` refers to the calling context; its value depends on how a function is invoked or bound.

```
const obj = {
  v: 1,
  m() {
    return this.v;
  },
};
obj.m(); // 1
```

7. What is a promise?

A promise represents an eventual async result and lets you attach handlers for success or failure.

```
const p = Promise.resolve("done");
p.then(console.log);
```

8. What is async/await?

Syntax on top of promises that makes async code read like sync code and simplifies error handling.

```
async function f() {
  const r = await fetch("/api");
  return r.json();
}
```

9. What is the difference between null and undefined?

`undefined` means a variable exists with no value; `null` is an explicit empty value.

10. How do you deep clone an object?

Use JSON methods for simple data; use libraries for dates, functions, or circular refs.

```
const clone = JSON.parse(JSON.stringify(obj));
```

11. What is the spread operator?

Expands arrays or object properties into a new array or object for easy copying and merging.

```
const a = [1, 2];
const b = [...a, 3]; // [1,2,3]
```

12. What is a higher-order function?

A function that takes or returns another function. Useful for composition and abstraction.

```
const map = (arr, fn) => arr.map(fn);
```

13. What is call, apply, and bind?

They invoke or produce functions with a specific `this` value; `call` / `apply` call immediately, `bind` returns a new function.

```
fn.call(obj, 1, 2);
fn.apply(obj, [1, 2]);
const g = fn.bind(obj);
```

14. What is prototypal inheritance?

Objects inherit from other objects via the prototype chain; use `Object.create` or `class` syntax to compose behavior.

```
const parent = {
  greet() {
    return "hi";
  },
};
const child = Object.create(parent);
```

15. Map vs forEach?

`map` returns a new array of transformed items; `forEach` runs a function for side effects and returns undefined.

```
const doubled = [1, 2].map((x) => x * 2);
```

16. How do you debounce a function?

Debounce delays invocation until activity stops; useful for input and resize handlers.

```
function debounce(fn, ms) {
  let t;
  return (...a) => {
    clearTimeout(t);
    t = setTimeout(() => fn(...a), ms);
  };
}
```

17. What is the module pattern?

Encapsulate private state in a closure and expose an API to avoid globals.

```
const Counter = (() => {
  let n = 0;
  return { inc: () => ++n, get: () => n };
})();
```

18. What is a pure function?

A function with no side effects that returns the same output for the same input.

19. How to check if a variable is an array?

Use the built-in reliable method.

```
Array.isArray(x);
```

20. Synchronous vs asynchronous code?

Synchronous runs in sequence and can block; asynchronous runs later and does not block the main flow.

21. What is the Temporal Dead Zone?

The period between entering scope and variable declaration where `let` / `const` cannot be accessed. Accessing the variable before its declaration results in a ReferenceError.

```
console.log(a); // ReferenceError
let a = 5;
```

22. What is currying?

Transforming a function with multiple arguments into a sequence of functions each taking one argument. Currying helps create reusable and partially applied functions.

```
function curry(f) {
  return (a) => (b) => f(a, b);
}
const add = (a, b) => a + b;
const curriedAdd = curry(add);
curriedAdd(2)(3); // 5
```

23. What is memoization?

Caching function results to optimize performance. Memoization avoids repeated calculations for the same inputs.

```
function memoize(fn) {
  const cache = {};
  return (...args) => {
    const key = JSON.stringify(args);
    if (!(key in cache)) cache[key] = fn(...args);
    return cache[key];
  };
}
```

24. What is the difference between call stack and event loop?

The call stack tracks function calls (LIFO), while the event loop handles async callbacks from the queue. The event loop enables non-blocking behavior in JavaScript.

25. What is a generator function?

A function that can pause and resume execution using the `yield` keyword. Generators are useful for lazy evaluation and custom iteration.

```
function* gen() {
  yield 1;
  yield 2;
}
const it = gen();
it.next().value; // 1
```

26. What is the difference between shallow and deep copy?

A shallow copy copies top-level properties only, while a deep copy copies all nested objects/arrays. Deep copies are needed for complex data structures.

27. What is the purpose of Symbol in JavaScript?

A unique and immutable primitive value, often used as object property keys to avoid name collisions.

```
const sym = Symbol("desc");
const obj = { [sym]: 42 };
```

28. What is the difference between setTimeout and setInterval?

`setTimeout` runs once after a delay, while `setInterval` runs repeatedly at intervals. Both are used for scheduling code execution.

29. What is the difference between map, filter, and reduce?

`map` transforms each element, `filter` selects elements, and `reduce` combines elements into a single value. These are core array methods for functional programming.

30. What is the purpose of the 'use strict' directive?

Enables strict mode, catching common errors and preventing unsafe actions. Strict mode helps write safer and more predictable code.

```
"use strict";
x = 10; // ReferenceError
```

31. What is a WeakMap/WeakSet?

Collections that hold weak references to objects, allowing garbage collection. Useful for memory management in certain scenarios.

```
const wm = new WeakMap();
let obj = {};
wm.set(obj, 123);
obj = null; // entry can be GC'd
```

32. What is the difference between Object.freeze and Object.seal?

`freeze` makes an object immutable, while `seal` prevents adding/removing properties but allows changing values. Use these for controlling object mutability.

33. What is tail call optimization?

A feature where the JS engine reuses stack frames for tail-recursive calls, improving performance for recursive functions. Not widely supported yet.

34. What is the difference between == and Object.is()?

`==` is loose equality with coercion, while `Object.is()` is strict equality but treats NaN and -0/+0 differently.

```
Object.is(NaN, NaN); // true
Object.is(-0, 0); // false
```

35. What is the difference between function declaration and expression?

A function declaration is hoisted and can be called before its definition, while an expression is not hoisted.

```
function foo() {}
const bar = function () {};
```

36. What is the purpose of async generators?

Functions that yield promises and can be iterated with `for await...of`. Useful for streaming async data.

```
async function* gen() {
  yield await Promise.resolve(1);
}
```

37. What is the difference between microtasks and macrotasks?

Microtasks (promise callbacks) run after the current script, while macrotasks (setTimeout, setInterval, UI events) run later. Understanding the difference helps with async code timing.

38. What is the difference between Object.create and class?

`Object.create` creates an object with a specified prototype, while `class` is syntactic sugar for constructor functions and prototypes. Both are used for inheritance.

39. What is the difference between static and instance methods?

Static methods are called on the class itself, while instance methods are called on object instances.

```
class A {
  static foo() {}
  bar() {}
}
```

40. What is the difference between nullish coalescing (??) and OR (||)?

`??` returns the right value if the left is null or undefined, while `||` returns the right value if the left is falsy (including 0, '', false).

```
0 ?? 5; // 0
0 || 5; // 5
```

41. What is the DOM?

The DOM is a tree representation of the HTML document that JavaScript can read and modify. Use it to access elements and update the page.

42. How do you select elements in the DOM?

Use native selectors to find nodes by id, class, or CSS selector. `querySelector` and `querySelectorAll` accept CSS selectors.

```
document.getElementById("id");
document.querySelector(".class");
document.querySelectorAll("div");
```

43. How do you change the content of an element?

Set `textContent` for plain text or `innerHTML` when you need HTML markup. Prefer `textContent` to avoid XSS.

```
el.textContent = "Hello";
el.innerHTML = "<b>Bold</b>";
```

44. How do you add/remove classes?

Use the `classList` API to toggle or modify classes on an element. It is simple and safe.

```
el.classList.add("active");
el.classList.remove("active");
el.classList.toggle("open");
```

45. How do you handle events in JavaScript?

Attach listeners with `addEventListener` to respond to user actions. Remove listeners when no longer needed.

```
el.addEventListener("click", (e) => {
  /* handle click */
});
```

46. What is event bubbling and capturing?

Events travel down the tree in the capture phase and up in the bubble phase. Bubbling is the most commonly used phase.

47. How do you prevent default behavior?

Call `preventDefault()` on the event to stop the browser's default action like form submit or link navigation.

```
form.addEventListener("submit", (e) => {
  e.preventDefault(); /* custom submit */
});
```

48. How do you stop event propagation?

Use `stopPropagation()` to prevent the event from reaching parent listeners when you need to isolate handling.

```
btn.addEventListener("click", (e) => {
  e.stopPropagation();
});
```

49. What is localStorage and sessionStorage?

Both are key value stores in the browser. `localStorage` persists across sessions while `sessionStorage` clears on tab close.

```
localStorage.setItem("k", "v");
const v = localStorage.getItem("k");
```

50. How do you read/write cookies in JS?

Use `document.cookie` to set cookies and parse it to read values. Cookies are sent with every HTTP request.

```
document.cookie = "name=alice; SameSite=Strict; Secure";
```

51. What is the difference between innerHTML and textContent?

`innerHTML` sets HTML and can introduce XSS if content is untrusted. `textContent` writes plain text and is safer.

52. How do you create and insert elements?

Create elements with `createElement` and attach them with `appendChild` or `append` for flexible insertion.

```
const el = document.createElement("div");
el.textContent = "new";
document.body.appendChild(el);
```

53. How do you remove elements from the DOM?

Call `remove()` on an element to detach it from the document. It is supported in modern browsers.

```
el.remove();
```

54. == vs === in the DOM context?

`==` does type coercion while `===` checks type and value. Use `===` to avoid unexpected results.

55. How do you debounce a DOM event handler?

Debounce delays a function until activity stops so handlers run less frequently. Useful for input and resize events.

```
function debounce(fn, ms) {
  let t;
  return (...a) => {
    clearTimeout(t);
    t = setTimeout(() => fn(...a), ms);
  };
}
```

56. What do defer and async do for scripts?

`defer` runs scripts after HTML parsing in document order. `async` runs scripts as soon as they load and may run out of order.

57. How do you detect if the DOM is ready?

Listen for `DOMContentLoaded` to run code after the DOM is parsed and before resources finish loading.

```
document.addEventListener("DOMContentLoaded", () => {
  /* ready */
});
```

58. clientWidth vs offsetWidth?

`clientWidth` includes padding but not border or scrollbar. `offsetWidth` includes borders and layout offsets.

59. How do you scroll to an element in JS?

Use `scrollIntoView` for smooth scrolling to bring an element into view.

```
element.scrollIntoView({ behavior: "smooth" });
```

60. How do you make an AJAX request in JS?

Use the Fetch API which returns a promise and supports modern async patterns for network calls.

```
fetch("/api/data")
  .then((r) => r.json())
  .then(console.log);
```

61. What is the difference between let, const, and var?

`let` and `const` are block scoped; `var` is function scoped. `const` cannot be reassigned.

```
let x = 1;
const y = 2; // prefer let/const
```

62. What are arrow functions?

Shorter function syntax with lexical `this`. Great for small callbacks.

```
const add = (a, b) => a + b;
```

63. What are template literals?

Strings with interpolation and multi-line support using backticks.

```
const name = "Alice";
const msg = `Hi, ${name}!`;
```

64. What is destructuring?

Extract values from arrays or objects into variables in a single line.

```
const [a, b] = [1, 2];
const { x, y } = { x: 10, y: 20 };
```

65. What is the spread operator?

Expands arrays or object properties into a new array or object for copying and merging.

```
const arr2 = [...arr, 3];
const obj2 = { ...obj, z: 5 };
```

66. What is the rest operator?

Collects remaining arguments into an array in function parameters.

```
function sum(...nums) {
  return nums.reduce((a, b) => a + b, 0);
}
```

67. What are default parameters?

Provide fallback values for function arguments to simplify callers.

```
function greet(name = "Guest") {
  return `Hi, ${name}`;
}
```

68. What are classes?

Syntactic sugar over constructor functions and prototypes for object creation.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    return this.name;
  }
}
```

69. What is inheritance in ES6?

Use `extends` to inherit behavior from a parent class and `super` to call the parent constructor.

```
class Dog extends Animal {
  speak() {
    return `${this.name} barks`;
  }
}
```

70. What are static methods?

Methods on the class itself, not on instances, for utility behavior.

```
class MathUtil {
  static add(a, b) {
    return a + b;
  }
}
```

71. What are modules?

Use `export` and `import` to split code into reusable files and control scope.

```
// lib.js
export function add(a, b) {
  return a + b;
}
// main.js
import { add } from "./lib.js";
```

72. What is a promise?

A promise represents a future value and lets you attach handlers for success or failure.

```
const p = Promise.resolve("ok");
p.then(console.log);
```

73. What is async/await?

Syntactic sugar over promises that makes async code look synchronous and easier to read.

```
async function f() {
  const r = await fetch("/api");
  return r.json();
}
```

74. What is Object.assign?

Copies enumerable properties from sources into a target object for shallow merges.

```
const merged = Object.assign({}, a, b);
```

75. What is Array.find?

Returns the first element that satisfies a predicate or undefined if none match.

```
const first = arr.find((x) => x > 2);
```

76. What is Array.includes?

Checks if an array contains a value and returns a boolean.

```
arr.includes(2);
```

77. What is Symbol?

A primitive for unique keys, useful for hidden or non-colliding object properties.

```
const sym = Symbol("id");
obj[sym] = 1;
```

78. What are Set and Map?

`Set` stores unique values, `Map` stores key-value pairs with any key type.

```
const s = new Set([1, 2]);
const m = new Map([[1, "a"]]);
```

79. What is for...of?

Iterate over iterable objects like arrays and strings with cleaner syntax.

```
for (const x of arr) console.log(x);
```

80. What is Object.entries?

Returns an array of `[key, value]` pairs for an object's own enumerable properties.

```
Object.entries({ a: 1, b: 2 }); // [['a',1],['b',2]]
```

81. What is a promise?

A promise represents a future value for an asynchronous operation. Attach handlers to react to success or failure.

```
const p = new Promise((resolve) => setTimeout(() => resolve("done"),
1000));
p.then(console.log);
```

82. What are the states of a promise?

Pending, fulfilled, and rejected. A promise settles to one final state.

83. How do you consume a promise?

Use `then`, `catch`, and `finally` to handle results and errors.

```
p.then((res) => handle(res))
  .catch((err) => handleErr(err))
  .finally(() => cleanup());
```

84. What is async/await?

Syntax built on promises that makes async code look synchronous. Use try/catch for errors.

```
async function fetchJson(url) {
  const r = await fetch(url);
  return r.json();
}
```

85. How do you handle errors in async/await?

Wrap awaits in try/catch to capture rejections and handle them.

```
async function safe() {
  try {
    return await fetchJson("/api");
  } catch (e) {
    console.error(e);
  }
}
```

86. What is Promise.all?

Waits for all promises to fulfill or for one to reject. Useful for parallel tasks.

```
const [a, b] = await Promise.all([p1, p2]);
```

87. What is Promise.race?

Settles as soon as the first promise settles. Useful for timeouts and fastest result.

```
const first = await Promise.race([p1, p2]);
```

88. What is Promise.any?

Resolves with the first fulfilled promise and ignores rejections. Good for best-success semantics.

```
const winner = await Promise.any([p1, p2]);
```

89. What is Promise.allSettled?

Waits for all promises to settle and returns each result. Useful for reporting multiple outcomes.

```
const results = await Promise.allSettled([p1, p2]);
```

90. How do you convert a callback to a promise?

Wrap the callback API in a new Promise and resolve or reject inside the callback.

```
function wait(ms) {
  return new Promise((r) => setTimeout(r, ms));
}
```

91. What is top-level await?

You can use `await` at the module top level in ES modules to simplify async bootstrapping.

92. What is a microtask?

A microtask runs after the current script and before rendering. Promise callbacks run as microtasks.

93. What is callback hell?

Deeply nested callbacks that make code hard to read. Promises and async/await help flatten control flow.

94. How do you chain promises?

Return values or promises from `then` to form a chain of asynchronous steps.

```
fetch("/api")
  .then((r) => r.json())
  .then((data) => process(data));
```

95. How do you cancel a fetch request?

Use `AbortController` and pass its signal to fetch, then call `abort()` to cancel.

```
const c = new AbortController();
fetch("/api", { signal: c.signal });
c.abort();
```

96. What is the event loop?

The runtime mechanism that processes tasks, microtasks, and rendering to enable non blocking behavior.

97. What is a race condition?

When outcomes depend on timing or ordering of async operations. Avoid with proper synchronization or ordering.

98. How do you retry a failed promise?

Wrap the call in a loop or recursive function that retries on rejection with optional delay.

```
async function retry(fn, times) {
  for (let i = 0; i < times; i++) {
    try {
      return await fn();
    } catch (e) {}
  }
}
```

99. What is a thunk?

A thunk is a zero argument function that delays computation or wraps a value for later evaluation.

100. setTimeout vs setImmediate?

`setTimeout` runs after a delay in browsers. `setImmediate` exists in Node and runs after I/O events.

101. What is a function declaration vs expression?

A declaration creates a named function that is hoisted. An expression creates a function value and is not hoisted the same way.

```
function a() {
  return 1;
}
const b = function () {
  return 2;
};
console.log(typeof a, typeof b);
```

102. What is an IIFE?

An IIFE runs immediately after it is defined and creates a private scope.

```
(function () {
  console.log("run");
})();
```

103. What is lexical scope?

Scope is determined by where functions are written in the source code. Inner functions can access outer variables.

```
function outer() {
  let x = 1;
  function inner() {
    return x;
  }
  return inner();
}
```

104. Global vs local scope?

Global variables are available everywhere. Local variables exist only inside their function or block.

105. What is the arguments object?

A function local object that holds all passed arguments in older APIs. Use rest parameters for a real array.

```
function sum() {
  return arguments[0] + arguments[1];
}
function sumRest(...nums) {
  return nums.reduce((a, b) => a + b, 0);
}
```

106. What is the rest parameter?

It gathers remaining arguments into an array for easier handling.

```
function join(sep, ...parts) {
  return parts.join(sep);
}
```

107. call vs apply vs bind?

`call` and `apply` call a function with a specific `this`. `bind` returns a new function with `this` fixed.

```
fn.call(obj, 1, 2);
fn.apply(obj, [1, 2]);
const bound = fn.bind(obj);
```

108. What is a callback function?

A callback is a function passed into another function and invoked later.

```
setTimeout(() => console.log("done"), 1000);
```

109. What is a higher-order function?

A function that takes or returns another function. It enables composition and reuse.

```
const map = (fn, arr) => arr.map(fn);
```

110. What is recursion?

A function that calls itself to solve smaller parts of a problem.

```
function fact(n) {
  return n <= 1 ? 1 : n * fact(n - 1);
}
```

111. What is closure?

A closure lets a function keep access to variables from its defining scope even after that scope has finished.

```
function makeCounter() {
  let n = 0;
  return () => ++n;
}
const c = makeCounter();
c();
```

112. var vs let vs const in scope?

`var` is function scoped, `let` and `const` are block scoped. `const` prevents reassignment.

113. What is hoisting?

Declarations are conceptually moved to the top of their scope, so functions and vars can be referenced before they appear.

```
console.log(typeof foo);
function foo() {}
```

114. What is the 'this' keyword?

`this` refers to the object that invoked the function. Its value depends on call site and binding.

115. Arrow function 'this' behavior?

Arrow functions inherit `this` from their enclosing scope and do not create their own `this`.

116. What is the module pattern?

A way to encapsulate private state and expose a public API, often using closures.

```
const Counter = (function () {
  let n = 0;
  return {
    inc() {
      return ++n;
    },
  };
})();
```

117. What is function currying?

Currying transforms a function that takes multiple arguments into a sequence of functions each taking one argument.

```
const add = (a) => (b) => a + b;
const add2 = add(2);
add2(3);
```

118. What is function composition?

Composition combines functions so the output of one becomes the input of the next.

```
const compose = (f, g) => (x) => f(g(x));
```

119. Function vs method?

A method is a function that is a property of an object. A function stands alone.

```
const o = {
  m() {
    return 1;
  },
};
o.m();
```

120. Synchronous vs asynchronous functions?

Synchronous functions run to completion before the next step. Asynchronous ones yield control and resume later.

121. What is an array?

An array stores ordered values accessible by numeric indices. Arrays provide many built-in methods for common tasks.

122. How do you add/remove elements from an array?

Use push/pop for the end, unshift/shift for the front, and splice for arbitrary positions.

```
const a = [1, 2];
a.push(3);
a.splice(1, 1);
// result: [1, 3]
```

123. What is an object?

An object stores keyed values. Keys are strings or symbols and values can be any type.

```
const obj = { a: 1, b: 2 };
```

124. What is a Set?

A Set holds unique values and removes duplicates automatically.

```
const s = new Set([1, 2, 2, 3]);
// result: Set { 1, 2, 3 }
```

125. What is a Map?

A Map is a key value collection where keys can be any type, not only strings.

```
const m = new Map([
  [1, "a"],
  ["k", "v"],
]);
```

126. How do you iterate over arrays?

Use for, for...of, forEach, or map/filter/reduce for transforming data.

127. How do you iterate over objects?

Use Object.keys, Object.values, or Object.entries and then loop over the resulting array.

```
Object.entries(obj).forEach(([k, v]) => console.log(k, v));
```

128. How do you check if a value is in an array?

Use includes for simple membership checks or indexOf for older code.

```
arr.includes(2);
```

129. How do you merge arrays or objects?

Use spread for arrays and spread or Object.assign for objects.

```
const merged = [...a, ...b];
const o = { ...aObj, ...bObj };
```

130. What is a shallow copy vs deep copy?

A shallow copy duplicates top level properties only. A deep copy clones nested structures as well.

```
const shallow = [...arr];
const deep = JSON.parse(JSON.stringify(obj));
```

131. How do you remove duplicates from an array?

Convert to a Set then back to an array to remove duplicates quickly.

```
const unique = [...new Set(arr)];
```

132. How do you sort an array?

Use sort with a compare function for numeric or custom ordering.

```
arr.sort((a, b) => a - b);
```

133. How do you find the max/min in an array?

Use Math.max or Math.min with the spread operator for plain arrays.

```
const max = Math.max(...arr);
```

134. How do you group array items by property?

Use reduce to accumulate items into buckets keyed by the property.

```
const grouped = arr.reduce((acc, item) => {
  (acc[item.type] ||= []).push(item);
  return acc;
}, {});
```

135. What is a linked list?

A linked list is a sequence of nodes where each node points to the next. It is not built into JS but can be implemented.

136. What is a stack?

A stack is last in first out. Use arrays push and pop to implement it.

137. What is a queue?

A queue is first in first out. Use arrays shift/unshift or a ring buffer for performance.

138. What is a tree?

A tree is a hierarchical structure with parent and child nodes. Trees model DOM and file systems.

139. What is a hash table?

A hash table offers fast key lookup. JS objects and Maps act like hash tables.

140. What is a graph?

A graph is a set of nodes connected by edges. It models networks and relationships.

141. How do you reverse a string?

Split into characters, reverse the array, then join back into a string.

```
const reversed = str.split("").reverse().join("");
```

142. How do you check for a palindrome?

Compare the string to its reversed form to test symmetry.

```
function isPalindrome(s) {
  return s === s.split("").reverse().join("");
}
```

143. How do you find the factorial of a number?

Use recursion or an iterative loop for small inputs.

```
function fact(n) {
  return n <= 1 ? 1 : n * fact(n - 1);
}
```

144. How do you find the largest/smallest number in an array?

Use Math.max or Math.min with the spread operator for plain arrays.

```
const max = Math.max(...arr);
const min = Math.min(...arr);
```

145. How do you remove duplicates from an array?

Convert to a Set and back to an array to remove duplicates quickly.

```
const unique = [...new Set(arr)];
```

146. How do you flatten a nested array?

Use Array.flat for shallow or deep flattening with Infinity for full depth.

```
const flat = arr.flat(Infinity);
```

147. How do you find the intersection of two arrays?

Filter one array by membership in the other for a simple intersection.

```
const intersection = a.filter((x) => b.includes(x));
```

148. How do you merge two sorted arrays?

Use two pointers to merge in linear time without extra sorting.

```
function merge(a, b) {
  const out = [];
  let i = 0,
    j = 0;
  while (i < a.length && j < b.length) {
    if (a[i] < b[j]) out.push(a[i++]);
    else out.push(b[j++]);
  }
  return out.concat(a.slice(i), b.slice(j));
}
```

149. How do you implement binary search?

Use a loop that halves the search range on each step for a sorted array.

```
function binarySearch(arr, x) {
  let l = 0,
    r = arr.length - 1;
  while (l <= r) {
    const m = Math.floor((l + r) / 2);
    if (arr[m] === x) return m;
    if (arr[m] < x) l = m + 1;
    else r = m - 1;
  }
  return -1;
}
```

150. How do you implement bubble sort?

Compare adjacent elements and swap to bubble largest items to the end.

```
function bubbleSort(arr) {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
    }
  }
  return arr;
}
```

151. How do you implement quick sort?

Pick a pivot, partition elements, then recursively sort partitions.

```
function quickSort(a) {
  if (a.length <= 1) return a;
  const p = a[Math.floor(a.length / 2)];
  const left = a.filter((x) => x < p);
  const mid = a.filter((x) => x === p);
  const right = a.filter((x) => x > p);
  return [...quickSort(left), ...mid, ...quickSort(right)];
}
```

152. How do you implement merge sort?

Split the array, sort halves recursively, then merge them.

```
function mergeSort(a) {
  if (a.length <= 1) return a;
  const mid = Math.floor(a.length / 2);
  const left = mergeSort(a.slice(0, mid));
  const right = mergeSort(a.slice(mid));
  return merge(left, right);
}
```

153. How do you find the nth Fibonacci number?

Use iteration for efficiency or recursion for simplicity on small n.

```
function fib(n) {
  let a = 0,
    b = 1;
  for (let i = 0; i < n; i++) {
    [a, b] = [b, a + b];
  }
  return a;
}
```

154. How do you check if two strings are anagrams?

Sort and compare or count characters to compare frequency maps.

```
function isAnagram(a, b) {
  return a.split("").sort().join("") === b.split("").sort().join("");
}
```

155. How do you count the number of vowels in a string?

Match vowels with a regex and count matches, or loop and tally.

```
const count = (s.match(/[aeiou]/gi) || []).length;
```

156. How do you find the first non-repeating character?

Count occurrences then return the first character with a single count.

```
function firstUnique(s) {
  const m = new Map();
  for (const c of s) m.set(c, (m.get(c) || 0) + 1);
  for (const c of s) if (m.get(c) === 1) return c;
  return null;
}
```

157. How do you implement a stack?

Use an array with push and pop to get LIFO behavior.

```
const stack = [];
stack.push(1);
stack.pop();
```

158. How do you implement a queue?

Use an array with push and shift for FIFO, or a linked list for heavy workloads.

```
const q = [];
q.push(1);
q.shift();
```

159. How do you detect a cycle in a linked list?

Use two pointers moving at different speeds (Floyd's algorithm) to detect loops.

```
function hasCycle(head) {
  let slow = head,
    fast = head;
  while (fast && fast.next) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow === fast) return true;
  }
  return false;
}
```

160. How do you traverse a binary tree?

Use recursion for in-order, pre-order, or post-order traversal.

```
function inOrder(node, visit) {
  if (!node) return;
  inOrder(node.left, visit);
  visit(node.value);
  inOrder(node.right, visit);
}
```

161. What is the Fetch API?

A modern promise based API for making HTTP requests from the browser.

```
fetch("/api/data")
  .then((res) => res.json())
  .then((data) => console.log(data));
```

162. What is the Geolocation API?

An API to get the user's location with explicit permission from the user.

```
navigator.geolocation.getCurrentPosition((pos) => console.log(pos.coords));
```

163. What is the Notification API?

A way to show native system notifications after user grants permission.

```
if (Notification.permission === "granted") new Notification("Hello");
```

164. What is the Clipboard API?

APIs to read from and write to the clipboard in secure contexts.

```
navigator.clipboard.writeText("Copied");
```

165. What is the Fullscreen API?

An API to request fullscreen mode for elements to provide immersive UI.

```
document.documentElement.requestFullscreen();
```

166. What is the Drag and Drop API?

APIs that enable dragging elements and handling drop targets in the page.

167. What is Local Storage?

A synchronous key value store that persists across sessions for the origin.

```
localStorage.setItem("user", "alice");
```

168. What is Session Storage?

A key value store that lasts for the lifetime of the tab or window.

169. What is the History API?

APIs to manipulate the browser history without full page loads for SPA routing.

```
history.pushState({}, "", "/new-url");
```

170. What is the Page Visibility API?

An API to detect when the page becomes hidden or visible to pause or resume work.

```
document.addEventListener("visibilitychange", () => {
  if (document.hidden) console.log("Hidden");
});
```

171. What is the Web Worker API?

A way to run JavaScript in a background thread to avoid blocking the UI.

```
const w = new Worker("worker.js");
w.postMessage({ task: "start" });
```

172. What is the File API?

APIs to read files selected by users or dropped onto the page.

173. What is the Canvas API?

A drawing API for 2D graphics and image manipulation in the browser.

```
const ctx = document.querySelector("canvas").getContext("2d");
ctx.fillRect(0, 0, 100, 100);
```

174. What is the Audio API?

APIs to play and process audio programmatically in the browser.

175. What is the Video API?

APIs to control video playback and handle media streams in the page.

176. What is the Speech Recognition API?

A browser API to convert spoken words into text where supported by the platform.

177. What is the WebSocket API?

A protocol for full duplex real time communication between client and server.

178. What is the Service Worker API?

A background script that can intercept network requests, enable offline caching, and power PWAs.

179. What is the Push API?

An API that works with service workers to deliver push messages to users.

180. What is the Device Orientation API?

An API to read device motion and orientation for responsive experiences on mobile devices.

181. What is XSS (Cross-Site Scripting)?

An attack where malicious scripts are injected into pages. Prevent with output encoding and Content Security Policy.

```
el.textContent = userInput; // use textContent to avoid injecting HTML
```

182. What is CSRF (Cross-Site Request Forgery)?

An attack that tricks a logged in user into making unwanted requests. Use anti CSRF tokens and SameSite cookies.

183. What is CORS?

Cross Origin Resource Sharing controls which origins can access resources. Configure server response headers to allow trusted origins.

```
res.setHeader("Access-Control-Allow-Origin", "https://example.com");
```

184. What is HTTPS?

HTTPS encrypts traffic between client and server to protect data in transit. Always serve sensitive pages over HTTPS.

185. What is Content Security Policy (CSP)?

A header that restricts which sources can load scripts, styles, and other resources to reduce XSS risk.

```
Content-Security-Policy: default-src 'self'
```

186. What is SQL Injection?

An attack that alters database queries via user input. Use parameterized queries or prepared statements to prevent it.

```
db.query("SELECT * FROM users WHERE id = ?", [id]);
```

187. What is authentication vs authorization?

Authentication verifies who a user is. Authorization decides what that user is allowed to do.

188. What is JWT (JSON Web Token)?

A compact token format for securely transmitting claims between parties, often used for stateless auth.

189. What is SameSite cookie?

A cookie attribute that restricts cross site sends to mitigate CSRF.

```
document.cookie = "token=abc; SameSite=Strict";
```

190. What is clickjacking?

An attack that tricks users into clicking hidden UI. Prevent by sending X-Frame-Options or CSP frame ancestors.

```
res.setHeader("X-Frame-Options", "DENY");
```

191. What is the X-Content-Type-Options header?

A header that prevents MIME sniffing to avoid serving content with the wrong type.

```
X-Content-Type-Options: nosniff
```

192. What is the X-XSS-Protection header?

A legacy header that enabled browser XSS filters. Modern CSP is preferred.

193. What is the Referrer-Policy header?

A header that controls how much referrer information is sent to other sites to protect privacy.

194. What is the Strict-Transport-Security header?

A header that forces browsers to use HTTPS for a domain for a set period.

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

195. What is the Permissions Policy header?

A header that controls access to powerful browser features like camera and geolocation.

196. What is input validation?

Checking and sanitizing user input on the server and client to prevent injection and logic errors.

197. What is output encoding?

Escaping data before rendering to prevent XSS, for example using textContent or HTML encoding.

198. What is rate limiting?

Restricting how often clients can call an endpoint to prevent abuse and DoS.

199. What is OAuth?

An open standard for delegated access that lets users grant limited access to their resources on other sites.

200. HTTP vs HTTPS?

HTTPS encrypts requests and responses to protect data and prevent tampering. HTTP is unencrypted and less secure.

201. What is a binary search algorithm?

Efficiently finds a target in a sorted array by halving the search range each step.

```
function binarySearch(arr, x) {
  let l = 0,
    r = arr.length - 1;
  while (l <= r) {
    const m = Math.floor((l + r) / 2);
    if (arr[m] === x) return m;
    if (arr[m] < x) l = m + 1;
    else r = m - 1;
  }
  return -1;
}
```

202. How do you implement a stack in JavaScript?

Use an array and its push/pop methods for LIFO behavior.

```
const stack = [];
stack.push(1);
stack.pop();
```

203. How do you implement a queue in JavaScript?

Use an array with push/shift for FIFO, or a linked list for heavy workloads.

```
const q = [];
q.push(1);
q.shift();
```

204. What is a hash table?

A data structure for fast key lookup. Use Map or plain objects in JS.

```
const m = new Map();
m.set("k", "v");
console.log(m.get("k"));
```

205. How do you traverse a binary tree?

Use recursion for in-order, pre-order, or post-order traversal.

```
function inOrder(node, visit) {
  if (!node) return;
  inOrder(node.left, visit);
  visit(node.value);
  inOrder(node.right, visit);
}
```

206. What is a linked list?

A sequence of nodes where each node points to the next. It is implemented manually in JS.

207. What is a graph?

A set of nodes connected by edges. Use adjacency lists or matrices to represent it.

208. How do you reverse a string?

Split into characters, reverse the array, then join back into a string.

```
const reversed = str.split("").reverse().join("");
```

209. How do you check for a palindrome?

Compare the string to its reversed form to test symmetry.

```
function isPalindrome(s) {
  return s === s.split("").reverse().join("");
}
```

210. How do you find the factorial of a number?

Use an iterative loop or recursion for small inputs.

```
function fact(n) {
  return n <= 1 ? 1 : n * fact(n - 1);
}
```

211. How do you find the largest/smallest number in an array?

Use Math.max and Math.min with the spread operator for plain arrays.

```
const max = Math.max(...arr);
const min = Math.min(...arr);
```

212. How do you remove duplicates from an array?

Convert to a Set and back to an array to remove duplicates quickly.

```
const unique = [...new Set(arr)];
```

213. How do you flatten a nested array?

Use Array.flat for shallow depth or Infinity for full depth. Use recursion if unavailable.

```
const flat = arr.flat(Infinity);
```

214. How do you find the intersection of two arrays?

Filter one array by membership in the other for a simple intersection.

```
const intersection = a.filter((x) => b.includes(x));
```

215. How do you merge two sorted arrays?

Use two pointers to merge in linear time without extra sorting.

```
function merge(a, b) {
  const out = [];
  let i = 0,
    j = 0;
  while (i < a.length && j < b.length) {
    if (a[i] < b[j]) out.push(a[i++]);
    else out.push(b[j++]);
  }
  return out.concat(a.slice(i), b.slice(j));
}
```

216. How do you implement bubble sort?

Repeatedly swap adjacent elements to move larger items to the end.

```
function bubbleSort(arr) {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
    }
  }
  return arr;
}
```

217. How do you implement quick sort?

Choose a pivot, partition elements, then recursively sort partitions.

```
function quickSort(a) {
  if (a.length <= 1) return a;
  const p = a[Math.floor(a.length / 2)];
  const left = a.filter((x) => x < p);
  const mid = a.filter((x) => x === p);
  const right = a.filter((x) => x > p);
  return [...quickSort(left), ...mid, ...quickSort(right)];
}
```

218. How do you implement merge sort?

Split the array, sort halves recursively, then merge them back together.

```
function mergeSort(a) {
  if (a.length <= 1) return a;
  const mid = Math.floor(a.length / 2);
  const left = mergeSort(a.slice(0, mid));
  const right = mergeSort(a.slice(mid));
  return merge(left, right);
}
```

219. How do you find the nth Fibonacci number?

Use iteration for efficiency on larger n to avoid exponential recursion.

```
function fib(n) {
  let a = 0,
    b = 1;
  for (let i = 0; i < n; i++) [a, b] = [b, a + b];
  return a;
}
```

220. How do you check if two strings are anagrams?

Sort both strings and compare, or compare character frequency maps.

```
function isAnagram(a, b) {
  return a.split("").sort().join("") === b.split("").sort().join("");
}
```

221. What is a priority queue?

A queue where elements are served by priority rather than arrival order. Useful for scheduling and Dijkstra's algorithm.

222. How do you implement a min heap?

A binary tree where each parent is less than its children, stored in an array for compact indexing.

```
const heap = []; // simple push/pop with sift operations
```

223. What is depth-first search (DFS)?

An algorithm that explores as far along a branch before backtracking to discover other branches.

```
function dfs(n) {
  if (!n) return;
  console.log(n.val);
  dfs(n.left);
  dfs(n.right);
}
```

224. What is breadth-first search (BFS)?

An algorithm that explores all neighbors at the current depth before moving deeper. Use a queue to implement it.

```
function bfs(root) {
  const q = [root];
  while (q.length) {
    const n = q.shift();
    console.log(n.val);
    if (n.left) q.push(n.left);
    if (n.right) q.push(n.right);
  }
}
```

225. What is a trie?

A tree structure optimized for string retrieval where each node represents a character. Great for autocomplete.

226. How do you detect a cycle in a graph?

Use DFS with a visited set and recursion stack, or use union-find for undirected graphs.

227. What is dynamic programming?

A technique that stores results of subproblems to avoid repeated work and build up an optimal solution.

228. How do you find the shortest path in a graph?

Use Dijkstra for weighted graphs without negative edges or BFS for unweighted graphs.

229. What is a hash collision?

When two keys map to the same hash bucket. Good hash functions and resizing reduce collisions.

230. How do you implement selection sort?

Repeatedly find the minimum from the unsorted part and swap it into place.

```
function selectionSort(a) {
  for (let i = 0; i < a.length; i++) {
    let m = i;
    for (let j = i + 1; j < a.length; j++) if (a[j] < a[m]) m = j;
    [a[i], a[m]] = [a[m], a[i]];
  }
  return a;
}
```

231. How do you implement insertion sort?

Build a sorted region by inserting each item into its correct position in that region.

```
function insertionSort(a) {
  for (let i = 1; i < a.length; i++) {
    let j = i;
    while (j > 0 && a[j] < a[j - 1]) {
      [a[j], a[j - 1]] = [a[j - 1], a[j]];
      j--;
    }
  }
  return a;
}
```

232. How do you find the longest common subsequence?

Use dynamic programming with a table of prefix matches to build the LCS length and sequence.

233. What is a balanced binary tree?

A tree where left and right subtree heights differ by at most one at every node to ensure O(log n) operations.

234. How do you implement a hash function?

Map keys to numeric indices using a deterministic transformation, then mod by table size and handle collisions.

235. What is a union-find data structure?

A structure that tracks disjoint sets with union and find operations, often using path compression for speed.

236. How do you find the lowest common ancestor in a tree?

Walk up from nodes using parent links or use a recursion that checks left and right subtrees to find the first common node.

237. What is a topological sort?

An ordering of nodes in a DAG where every directed edge goes from earlier to later in the order. Use DFS or Kahn's algorithm.

238. How do you implement a circular queue?

Use a fixed array and head/tail indices that wrap using modulo arithmetic to reuse space.

```
// head and tail wrap with (index + 1) % size
```

239. What is a skip list?

A layered linked list structure that gives logarithmic search time by skipping over nodes at higher levels.

240. How do you find the median of an array?

Sort and pick the middle value, or use a selection algorithm or two heaps for streaming data.

241. What is a doubly linked list?

A linked list where each node points to both next and previous nodes. It allows efficient insertions and removals at both ends.

242. How do you detect a cycle in a linked list?

Use two pointers moving at different speeds and check if they meet, which indicates a cycle.

```
function hasCycle(head) {
  let slow = head,
    fast = head;
  while (fast && fast.next) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow === fast) return true;
  }
  return false;
}
```

243. What is a binary search tree (BST)?

A tree where left children are less than the node and right children are greater. It supports fast search, insert, and delete on average.

244. How do you balance a BST?

Use tree rotations or use self balancing trees like AVL or Red Black trees to keep height logarithmic.

245. What is a heap?

A complete binary tree where each parent compares correctly to its children to provide quick access to min or max.

246. How do you implement a hash map?

Use an array of buckets and a hash function to index keys, handling collisions with chaining or open addressing.

247. What is a bloom filter?

A space efficient, probabilistic structure for membership tests that may yield false positives but not false negatives.

248. How do you find the kth largest element in an array?

Use a min heap of size k or use a selection algorithm like Quickselect for average linear time.

249. What is a sliding window algorithm?

Move a window over an array or string and update results incrementally to solve subarray problems efficiently.

250. How do you implement depth limited DFS?

Add a depth parameter and stop recursion when the limit is reached to bound the search.

251. What is a segment tree?

A tree that supports efficient range queries and updates over an array by storing aggregate values at nodes.

252. How do you implement a queue with two stacks?

Use one stack for enqueues and another for dequeues, moving elements only when needed to preserve order.

```
const inbox = [],
  outbox = [];
function enqueue(x) {
  inbox.push(x);
}
function dequeue() {
  if (!outbox.length) while (inbox.length) outbox.push(inbox.pop());
  return outbox.pop();
}
```

253. What is a sparse array?

An array where most entries are empty or undefined, which can save memory when indices are widely spaced.

254. How do you rotate a matrix?

Transpose the matrix and then reverse each row to rotate 90 degrees clockwise.

255. What is union by rank in union-find?

Attach the smaller tree to the larger root to keep trees shallow and speed up find operations.

256. How do you implement a trie for autocomplete?

Store nested nodes for each character and mark word ends to enable efficient prefix lookup.

257. What is a weighted graph?

A graph where edges have weights or costs, often used with shortest path algorithms like Dijkstra.

258. How do you find all paths between two nodes in a graph?

Use DFS to explore all possible routes and backtrack to collect paths.

259. What is the time complexity of binary search?

O(log n) because the search range halves on each step, leading to logarithmic steps.

260. How do you find the majority element in an array?

Count occurrences with a hash map or use Boyer Moore voting algorithm for linear time and constant space.

261. What is a circular linked list?

A linked list where the last node points back to the first, forming a loop. Useful for round robin scheduling.

262. How do you implement a min/max stack?

Track the current min or max with an auxiliary stack so retrieval is O(1).

```
const vals = [],
  mins = [];
function push(x) {
  mins.push(Math.min(x, mins[mins.length - 1] ?? x));
  vals.push(x);
}
function pop() {
  mins.pop();
  return vals.pop();
}
function min() {
  return mins[mins.length - 1];
}
```

263. What is a monotonic stack?

A stack that keeps elements in increasing or decreasing order to solve range or next greater element problems.

```
// next greater example
const res = [],
  st = [];
for (let i = 0; i < arr.length; i++) {
  while (st.length && arr[i] > arr[st[st.length - 1]]) res[st.pop()] =
arr[i];
  st.push(i);
}
```

264. How do you implement an LRU cache?

Combine a hash map for O(1) lookups with a doubly linked list to track recent usage.

```
// map + linked list gives O(1) get and put operations
```

265. What is a binary heap?

A complete binary tree stored in an array that gives O(log n) push and pop for min or max operations.

266. How do you find the shortest path in a weighted graph?

Use Dijkstra's algorithm for non negative weights, or Bellman Ford when negatives exist.

267. What is a topological sort?

An ordering of nodes in a DAG where each directed edge goes from earlier to later nodes, used for scheduling.

268. How do you implement a disjoint set (union-find)?

Use parent pointers with path compression and union by rank to keep operations near constant time.

269. What is a Fenwick tree (binary indexed tree)?

A tree stored in an array that supports prefix sums and updates in O(log n).

```
// update and query using lowbit operations on indices
```

270. How do you implement a trie for prefix search?

Use nested objects or maps for each character and mark terminal nodes for words.

271. What is a graph adjacency list?

A representation where each node stores a list of its neighbors, efficient for sparse graphs.

272. How do you implement a graph adjacency matrix?

Use a 2D array where cell [i][j] stores presence or weight of the edge between nodes i and j.

273. What is a strongly connected component?

A set of nodes in a directed graph where every node is reachable from every other node in the set.

274. How do you implement breadth-first traversal of a graph?

Use a queue and a visited set to visit nodes level by level from a start node.

```
const q = [start];
const seen = new Set([start]);
while (q.length) {
  const n = q.shift();
  for (const nei of adj[n])
    if (!seen.has(nei)) {
      seen.add(nei);
      q.push(nei);
    }
}
```

275. What is depth-limited search?

A DFS variant that stops at a specified depth to avoid infinite paths or control search cost.

276. How do you find the lowest common ancestor in a BST?

Walk from the root comparing values to both targets until you find the split point that is their LCA.

277. What is a balanced AVL tree?

A self balancing BST that keeps subtree heights within one to guarantee O(log n) operations.

278. How do you implement a skip list?

Build multiple levels of linked lists where higher levels skip across nodes to speed up search to O(log n) on average.

279. What is a sparse matrix?

A matrix where most entries are zero or empty and it is stored using compact structures like coordinate lists.

280. How do you find the diameter of a tree?

Run two BFS/DFS: first from any node to find the farthest node, then from that node to find the maximum distance.

281. What is a k-d tree?

A k-d tree partitions k-dimensional space for fast nearest neighbor queries. It splits on alternating axes.

```
function Node(point, axis, left = null, right = null) {
  return { point, axis, left, right };
}
```

282. How do you implement a bloom filter?

A bloom filter uses multiple hashes and a bit array for compact membership checks. It can return false positives but never false negatives.

```
const bits = new Uint8Array(1024);
function add(hashIndices) {
  hashIndices.forEach((i) => (bits[i] = 1));
}
function maybeHas(hashIndices) {
  return hashIndices.every((i) => bits[i]);
}
```

283. What is a suffix tree?

A suffix tree indexes all suffixes of a string for fast substring search. Building one efficiently requires linear algorithms.

284. How do you implement a union-find with path compression?

Union-find tracks disjoint sets. Path compression flattens parent pointers to speed up finds.

```
function find(parent, x) {
  if (parent[x] !== x) parent[x] = find(parent, parent[x]);
  return parent[x];
}
function union(parent, a, b) {
  parent[find(parent, a)] = find(parent, b);
}
```

285. What is a red-black tree?

A red-black tree is a self-balancing BST that keeps height logarithmic. It uses color and rotation rules.

286. How do you implement a max heap?

A max heap keeps the largest value at the root and supports O(log n) insert and remove. Use sift up/down on an array.

```
function heapPush(arr, val) {
  arr.push(val);
  siftUp(arr, arr.length - 1);
}
function heapPop(arr) {
  swap(arr, 0, arr.length - 1);
  const v = arr.pop();
  siftDown(arr, 0);
  return v;
}
```

287. What is a segment tree?

A segment tree stores aggregates for intervals to answer range queries fast. It supports updates and queries in O(log n).

288. How do you implement a trie for word search?

A trie stores characters per node so prefixes map to nodes. It makes prefix lookup and autocomplete fast.

```
class TrieNode {
  constructor() {
    this.children = {};
    this.end = false;
  }
}
```

289. What is a graph adjacency list vs matrix?

An adjacency list stores neighbors per node and suits sparse graphs. A matrix uses O(V^2) space and suits dense graphs.

```
const g = { a: ["b", "c"], b: ["a"] };
```

290. How do you find the shortest path in an unweighted graph?

Use BFS from the source to get shortest paths by edge count. Track predecessors to rebuild the path.

```
function bfs(start) {
  const q = [start];
  const dist = { [start]: 0 };
  while (q.length) {
    const v = q.shift();
    (g[v] || []).forEach((n) => {
      if (!(n in dist)) {
        dist[n] = dist[v] + 1;
        q.push(n);
      }
    });
  }
  return dist;
}
```

291. What is a strongly connected component in a graph?

An SCC is a set of nodes where each node reaches every other node in the set. Tarjan finds SCCs in linear time.

292. How do you implement a circular buffer?

A circular buffer reuses a fixed array with head and tail indices that wrap around. It gives O(1) push and pop.

```
class Ring {
  constructor(n) {
    this.a = new Array(n);
    this.h = 0;
    this.t = 0;
  }
  push(v) {
    this.a[this.t] = v;
    this.t = (this.t + 1) % this.a.length;
  }
}
```

293. What is a monotonic queue?

A monotonic queue keeps elements in order to answer sliding window max or min quickly. Implement with a deque.

294. How do you rotate a linked list?

Make the list circular, compute the new tail, then break the circle. Use length modulo rotation to find the split.

295. What is a weighted graph?

A weighted graph stores costs per edge. Use algorithms that account for weights when finding best paths.

```
const g = {
  a: [
    ["b", 3],
    ["c", 1],
  ],
};
```

296. How do you find the minimum spanning tree?

Kruskal and Prim find MSTs by choosing low cost edges without cycles. Kruskal sorts edges and uses union-find.

297. What is a sparse graph?

A sparse graph has far fewer edges than the maximum possible. Use adjacency lists and linear-time algorithms where possible.

298. How do you implement a topological sort?

Topological sort orders DAG nodes so all edges point forward. Kahn's algorithm uses in-degree counting and a queue.

299. What is a Fenwick tree?

A Fenwick tree supports prefix sums and point updates in O(log n) time using a compact array. It relies on bit operations for index movement.

```
function sum(i) {
  let s = 0;
  for (; i > 0; i -= i & -i) s += bit[i];
  return s;
}
```

300. How do you find the longest increasing subsequence?

Find LIS with DP in O(n^2) or patience sorting with binary search in O(n log n). Patience keeps a tails array and binary searches positions.