
🟢 JavaScript / DSA

🔹 Array Manipulation & Algorithms

- Write a function which will move all the zeros in an array to the right, with minimal time complexity.
  Input: [1,3,0,5,0,5,0] → **Output:** [1,3,5,5,0,0,0]
- Remove prime numbers from an array.
- Reduce (find second largest even number).
- Find second least element from an array.
- Write a function to take an array of sorted elements and find the given target's index — if not present, return where the position of the target will be.
  (Binary search / insertion index)
- Sum of prime numbers from an array.
- Find the sum of prime numbers.
- Palindrome check (string or number).
- Remove duplicate characters from a string.
- Remove duplicates from a linked list (given LL code with duplicates).
- Remove keys corresponding to arrays that contain odd numbers.

🔹 Recursion & Functional Concepts

- Factorial of a number using recursion.
- “Create a Singly Linked List and find the middle element using recursion.”
- Create a generator function to give even numbers reversely from 100 → 1, but print how many numbers are generated within 300 ms.
- Merge two LinkedLists (partially done).
- Create two linked lists using objects, find the sum of last values of both, reverse the resultant sum, find sum of digits, and convert result into a linked list — each digit as a node.

🔹 String / Character Logic

- Write a function to find the extra character that is in t but not in s.
- Input: s = "abcd", t = "acebd" → Output: "e"
- Input: s = "", t = "j" → Output: "j"

- Input: s = "a", t = "aa" → Output: "a"
- Check if parenthesis are valid.
- Console log puzzle:
- console.log(1 + +'1')
- Tomorrow's date in DD/MM/YYYY format.

🔹 **Complex / Nested Structure Problems**

- Find sum of deeply nested numeric arrays:
  Ab = [{ af:[34, 343]}, { af:[34, 343]}, ...] → Find total sum (deep nested sum).

🟠 **Node.js**

🔹 **Core Node.js Concepts**

- Create one **promise** that checks if a target element is present in an array using includes().
  - If present → resolve the promise
  - If resolved → write the result into a file using **fs** module
- How to delete a file using fs
- Using fs.link, fs.stat, etc. — (file manipulation)
- Create child process (practical)
- res.send vs res.write (HTTP response methods)
- Difference between **GET** and **POST**
- Send data through GET (query params)
- Response method to send JSON data
- Write a file using promises
- LocalStorage vs SessionStorage (theoretical + practical differences)
- Core Node modules (os, path, fs, events)
- Streams in Node.js
  - Types of streams
  - Transform / Duplex streams
- Concurrency and event loop in Node

- process.nextTick and setImmediate
- Options method (HTTP preflight)
- Cron job example
- Environment variables
- Middleware examples in Express
    - Types of middleware
    - Application vs Router middleware
- HTTP request structure and headers
- Content negotiation
- Path params vs Query params
- Dynamic routing and Router chaining
- Rate limiting & throttling
- CORS (Cross-Origin Resource Sharing)
- Idempotency (PUT vs POST)
- PM2 / Clustering / Fork vs Spawn
- JWT (structure, verification)
- Refresh token vs Access token
- CSRF prevention
- HTTPS / reverse proxy concepts

🟣 **MongoDB**

🔹 **CRUD, Indexing, and Querying**

- Create a **compound index**.
- Drop an index.
- Create a **geospatial index** (pending).
- Check if a collection is **capped**.
- Create a capped collection.
- $addToSet, $expr, $pullAll, $out, $group, $sum, $exists, $cond, $let
- Query examples:
    - First three names of highest salary people

    - Country-wise people count, average age, and highest salary
    - Upsert query
    - Average marks in class 10
    - Student names where mark > passmark
    - Student names not ending with vowels
- Transaction (start, commit, abort)
- Applying replication to a database
- Partition tolerance (CAP theorem)
- Embedded documents
- Views and materialized views
- BulkWrite / Batch sizing
- Distinct(), alias
- Clustered and Hashed index
- Covered query
- Drawbacks of indexing
- Data structure used in indexing
- Optimization beyond indexing (query optimization)
- GridFS usage (large files)
- $facet and aggregation pipelines
- Durability, consistency, write concern

🔵 **React**

🔹 **Hooks & Components**

- Create a **counter using useReducer**.
- Custom hook to toggle (pending).
- Create props proxy for HOC.
- ForwardRef and useRef.
- Advantages of useRef.
- useLayoutEffect vs useEffect.
- useCallback, React.memo.

- Error boundaries.
- React.Fragment — learn more.
- Conditional rendering.
- Pure components in React.
- Diffing algorithm.
- React Profiler — performance analysis.
- Lazy loading / React.Suspense.
- One-way data binding.
- Controlled vs uncontrolled components.
- Passing props parent → child.
- Custom hook scenario (e.g., API call, form toggle).
- Redux middleware — purpose.
- Rules for hooks.
- useNavigate vs useHistory.
- Types of routers.
- useNavigate parameters.
- MemoryRouter, useLocation.
- Axios interceptors & cancel tokens.
- HTML sanitization.
- Debouncing & Throttling in React.
- Shadow DOM vs Virtual DOM.
- Dead code elimination.
- SEO — CSR vs SSR (and type of website suited for each).
- Why do we need a code bundler (Webpack).

⚫ **Miscellaneous / Cross-Stack**

These involve multi-stack reasoning or are general MERN-related.

- "Type of website suited for CSR and SSR."
  (Front-end architectural decision — React + Node)
- "Tomorrow's date in DD/MM/YYYY format."
  (General JavaScript + React)

- “Create a compound index.”
  (MongoDB)
- “Write one promise and write to file.”
  (Node + JS)
- “Find middle of linked list using recursion.”
  (DSA)
