Data Structures and Algorithms (DSA 2) Topics and Questions

Sorting Algorithms

Theory

- What is Insertion Sort, and how does it work?
- What is the best-case time complexity of Selection Sort?
- What is the time complexity of Merge Sort (best, average, worst case)?
- Why is the complexity of Merge Sort O(n log n)?
- What is the worst-case time complexity of Quick Sort?
- What is the average-case time complexity of Quick Sort?
- What is the reason for $O(n^2)$ in the worst case for Quick Sort?
- What is the importance of pivot selection in Quick Sort?
- Does the pivot affect performance in Quick Sort?
- What is the disadvantage of Quick Sort over Merge Sort?
- What is the advantage of Merge Sort over Quick Sort?
- Why is Merge Sort preferred for linked lists?
- Why is Bubble Sort a stable sorting algorithm?
- What are stable sorting algorithms?
- What is in-place sorting?
- What are the disadvantages of Merge Sort?
- What is the best sorting algorithm for a partially sorted small array?
- What is the best sorting algorithm when memory is limited?
- What is the best sorting algorithm when memory is abundant and speed is a requirement?
- What is Heap Sort?
- What are the use cases of different sorting algorithms?
- What is the main disadvantage of Selection Sort?
- What is the divide and conquer strategy in sorting?

Practical

- Implement Bubble Sort.
- Implement Insertion Sort.
- Implement Selection Sort.

- Implement Merge Sort.
- Implement Quick Sort without using additional arrays.
- Sort a string using Merge Sort.
- Sort an array of strings using Merge Sort.
- Sort an array of objects based on the `.amount` property.
- Sort an array alphabetically using Quick Sort.
- Merge two sorted arrays into a single sorted array in O(n) time.
- Merge two sorted linked lists using Merge Sort properties.
- Check if an array is sorted with linear time complexity.

Stack

Theory

- What are the operations that can be performed on a stack (e.g., push, pop, peek)?
- What is the purpose of a stack pointer?
- What is stack overflow vs. stack underflow?
- What is a monotonic stack?
- What are the applications of a stack?
- How is a stack used in undo-redo operations?
- What is the difference between a stack and an array?
- When to use a stack instead of an array?
- What is the time complexity of pushing an element into a stack?
- What is a call stack?

Practical

- Implement a stack using a linked list.
- Implement a stack using a queue.
- Reverse a string using a stack.
- Reverse a stack.
- Reverse a stack using recursion.
- Sort a stack using a temporary stack.
- Delete a specific node from a stack.
- Delete the middle element from a stack.
- Implement a stack that rejects duplicate values.
- Implement a stack with methods to push, pop, and get the current highest number in O(1) complexity.
- Check if a string is a palindrome using a stack.
- Check for balanced parentheses using a stack (LeetCode #20).
- Check for valid parentheses and get the count of invalid pairs (modified LeetCode #20).
- Implement a MinStack (stack with O(1) retrieval of minimum element).

Queue

Theory

- What are the operations that can be performed on a queue (e.g., enqueue, dequeue, peek)?
- What are the types of queues (e.g., circular queue, priority queue, double-ended queue)?
- What are the applications of a circular queue?
- What are the applications of a double-ended queue?
- What is a monotonic queue?
- What is a bounded queue?
- What is a circular buffer?
- What are the applications of a priority queue?
- What is the difference between a queue and a stack?

Practical

- Implement a queue using a linked list.
- Implement a queue using a stack.
- Implement a circular queue.
- Implement a circular queue with a maximum length.
- Implement a double-ended queue using a linked list.
- Reverse a queue.
- Implement enqueue, dequeue, and display operations for a queue.
- Convert a stack into a queue.

Hash Table

Theory

- What is a hash table?
- How do hash functions work?
- What is a hash collision?
- Is it possible to avoid hash collisions?
- What are the methods to resolve hash collisions?
- What is open addressing?
- What is linear probing vs. quadratic probing?
- What is double hashing?
- What is separate chaining?
- What is a load factor in a hash table?
- What is rehashing?
- What are the applications of a hash table?
- Why would a hash table be used in database indexing?

- What is the difference between a hash table and a hash set?
- What is the time complexity of operations in a hash table?
- What is hashing vs. encryption?
- What are popular hashing algorithms (e.g., SHA1, MD5, CRC32)?
- What are the pros and cons of open hashing vs. closed hashing?
- How does separate chaining affect time complexity?
- What is a perfect hash function?

Practical

- Implement a hash table with collision handling (chaining with linked list).
- Implement a hash table with collision handling (open addressing).
- Implement linear probing.
- Implement quadratic probing.
- Implement double hashing.
- Implement rehashing.
- Find the first non-repeating character in a string using a hash table.
- Find the frequency of characters in a string using a hash table (e.g., "Mississippi").
- Find the first non-repeating character in a string using an inbuilt hash table (Map, e.g., "swiss").
- Remove duplicates from a string using a hash table.
- Find the least occurred number in a string using a hash table.
- Check if a string contains duplicates using a hash table.
- Find two numbers in an array that add up to a target sum using a hash table (LeetCode #1).
- Find uncommon elements from two different arrays using a hash table.
- Check if two strings are valid anagrams using a hash table.
- Find the occurrence of each character in a string using an inbuilt hash table (Map).
- Build a URL from a base URL and query parameters passed as a dictionary.

General Algorithms

Theory

- What is the divide and conquer strategy?
- What is the sliding window pattern?
- What is backtracking?

Practical

- Implement the Two Sum problem (LeetCode #1).
- Implement the Valid Parentheses problem (LeetCode #20).
- Merge two sorted linked lists (LeetCode #21).
- Find the subarray with the maximum sum (Kadane’s algorithm).
- Find the first missing number in an array.

- Convert a string like "APPLE" to "A-pp-ppp-llll-eeeee".
- Swap the first and last characters in a string.
- Remove odd-indexed elements from an array.
- Remove the longest string from an array.
- Find the second longest word in a sentence.
- Find the longest consecutive repeating characters in a string.
- Find a common character from two strings.
- Check if a string is balanced (parentheses).
- Convert the first character of a string to uppercase.

Additional Topics

Theory

- What is the size of a character in UTF-8 (grapheme)?
- What are control characters?
- What is the usage of `setInterval` and `clearInterval`?

Practical

- Practice problems from Blind 75 LeetCode and learn optimal solutions (refer to neetcode.io/practice and YouTube).
- Sort an array of students based on age.
- Find duplicate students in an array.
- Move zeroes to the end of an array.
- Practice application-level problems to improve logic and coding speed.
- Understand brute force and optimal solutions for problems.
- Debug code on the go and learn syntaxes properly.
- Practice problems on platforms like LeetCode, HackerRank, and GeeksforGeeks.

Guidelines

- Focus on understanding problems and solutions rather than memorizing.
- Read questions and think through solutions before implementing.
- Practice explaining solutions to improve clarity.
- Refer to resources like FreeCodeCamp and YouTube for learning optimal solutions.
- Improve presentation and logic-building skills.
- Practice medium-level problems with time constraints.
- Avoid memorizing applications of data structures; understand their usage.