Data Structures and Algorithms (DSA) Topics and Questions (Python Context)

Core Concepts

Theory

- What is a data structure?
- What are the types of data structures?
- What is the difference between linear and non-linear data structures?
- What is a hierarchical data structure?
- What is the difference between contiguous and non-contiguous data structures?
- What is an algorithm?
- Why do we need algorithms?
- What is a dynamically typed language?
- What is the difference between mutable and immutable data in Python?
- What is a string mutable or immutable in Python?
- Why is a string immutable in Python?
- What is memory allocation?
- What are the types of memory allocation?
- What is static memory allocation?
- What are the advantages of static memory allocation?
- What is dynamic memory allocation?
- What are the advantages of dynamic memory allocation?
- What is the difference between static and dynamic memory allocation?
- What is stack memory?
- What is heap memory?
- What is the difference between stack and heap memory?
- What is virtual memory?
- What is the relationship between virtual memory and data structures?
- What are the security benefits of virtual memory?
- What is a memory peak?
- What is memory deadlock?
- What is a memory leakage?
- What is a memory leakage in Python?
- How to prevent memory leakage in Python?

- What is a garbage collector and how does it work?
- What is stack overflow?
- What is heap overflow?
- What is memory padding?
- What is a circular reference?
- What is a byte?
- What is the difference between kilobyte and kibibyte?
- What is a gigabyte?
- What is a nibble vs. a byte?
- What is character encoding?
- What is UTF-8?
- What is ASCII?
- What are escape sequences?
- What are control characters?
- What is string encoding?
- How do computers interpret strings?
- How much memory is used by strings?
- What is string sanitization?
- What is string validation?

Complexity Analysis

Theory

- What is complexity analysis?
- What is time complexity?
- What is space complexity?
- What is asymptotic analysis?
- What is Big O notation?
- What is Big Theta notation?
- What is Big Omega notation?
- How does Big O notation differ from Big Theta and Big Omega?
- What does O(1) complexity mean?
- What is O(log n) vs. O(n log n)?
- What is the best-case, average-case, and worst-case time complexity?
- When would you choose space complexity over time complexity?
- What are the time complexities for common list operations (e.g., accessing, inserting, deleting)?
- What is the time complexity of adding an element to a linked list?
- What is the time complexity of searching an element in a linked list?
- What is the time complexity of deleting an element from the middle of a linked list?
- What is the time complexity of binary search?
- What is the worst-case time complexity of binary search?
- What is the best-case time complexity of linear search?

- What is the time complexity for four nested loops?
- What are logarithmic functions and values?
- What are asymptotic notations for quadratic equations?
- What is the space complexity of recursion?

Arrays (Lists in Python)

Theory

- What is a list in data structures?
- What is the difference between traditional arrays and Python lists?
- What is a homogeneous vs. heterogeneous list?
- What is a jagged list?
- What is a sparse list?
- What is a multidimensional list?
- What is a sublist?
- What are array.array or numpy arrays in Python?
- What are different operations in lists?
- What are the applications of lists in real-world problems?
- What is list amortization?
- What is the difference between linked lists and lists?
- What are the time complexities for common list operations (e.g., accessing, inserting, deleting)?
- What is memory allocation in lists?

Practical

- Find the minimum and maximum number from a list.
- Find the second largest element in a list.
- Find the second smallest element in a list.
- Find the third largest element in a list without sorting.
- Find the k-th largest element in a list.
- Find the frequency of occurrence of each number in a list.
- Create a function to find the average of even numbers in a list.
- Find the common element between lists.
- Find the last occurrence of an element in a sorted list with duplicate values.
- Find the minimum from a rotated sorted list recursively.
- Find a sublist with the maximum number of elements in continuously increasing order.
- Find a combination of two numbers in a list that sum to a target value (e.g., 4 or 5).
- Find two numbers that sum to a target value from a sorted list in a single iteration.
- Remove the smallest number from a list.
- Remove a specific element from a list.
- Reverse a list.
- Flatten a multidimensional list.

- Remove a sublist containing the largest number from a 2D list.
- Find the sum of a column in a 2D list and add it as the last column.
- Check if a target is in a 2D list and return its index (e.g., target = 8, output = [1, 2]).
- Remove duplicates from a list.
- Append every unique element without using built-in methods.
- Append and prepend elements to a list.

Strings

Theory

- What are the concepts of strings?
- What is the difference between a character and a string?
- What is a character array (list of characters in Python)?
- Why are strings immutable in Python?
- What are string permutations?
- How to extract a substring from a string?
- What are control characters?

Practical

- Find the first non-repeating character in a string.
- Find the last non-repeating character in a string.
- Check if two strings are anagrams of each other.
- Reverse a string without using built-in methods.
- Reverse each word in a string without using built-in methods (e.g., "HELLO WORLD" → "OLLEH DLROW").
- Remove extra whitespace between words in a string.
- Convert a string to title case.
- Find a string that begins with uppercase and ends with a period.
- Find the longest substring palindrome in a given string.
- Check if parentheses are balanced in a string (e.g., are_balanced("[()()()]") → true).
- Find the longest substring without vowels from a string.
- Find the longest consecutive repeating characters in a string.
- Find the shortest word in a string.
- Extract digits from a string.
- Remove all occurrences of a specific character from a string (e.g., remove "l" from "hello").
- Remove a character from a string using recursion.
- Count words in a sentence without using built-in functions.
- Find the palindrome prefix in a string.
- Implement string permutations.
- Find the longest substring without repeating characters.
- Convert PascalCase to snake_case.
- Reverse the letters of all words in a string without using built-in functions.

Linked Lists

Theory

- What is a linked list?
- Why is a linked list considered a linear data structure?
- What are the advantages of linked lists?
- What are the disadvantages of linked lists?
- What are the applications of linked lists?
- What are the differences between a singly linked list and a doubly linked list?
- What is a circular linked list?
- What is a circular doubly linked list?

Practical

- Remove odd element nodes from a linked list.
- Remove duplicates from a linked list without using other data structures.
- Reverse a singly linked list.
- Reverse a doubly linked list.
- Reverse print a doubly linked list.
- Sort elements in a linked list.
- Merge two sorted linked lists.
- Detect a cycle in a singly linked list (Floyd's cycle detection algorithm).
- Find the middle of a linked list in a single iteration (fast and slow pointer).
- Traverse a doubly linked list.
- Implement a singly linked list with proper encapsulation (using classes).
- Implement a singly linked list with a condition that only one duplicate is allowed.
- Swap the first and last nodes of a singly linked list.
- Print all elements of a linked list in order and reverse order.

Recursion

Theory

- What is recursion?
- What is a base case in recursion?
- What is the difference between direct and indirect recursion?
- What is tail recursion vs. head recursion?
- What is binary recursion?
- What is mutual recursion?
- What are the advantages of recursion?
- What are the disadvantages of recursion?

- What are the applications of recursion?
- What is the space complexity of recursion?
- What is the time complexity of recursion?
- What are recursive base conditions?
- What is the difference between recursion and loops?
- What are the limitations of recursion?
- What is backtracking?

Practical

- Implement a recursive function to find the sum of elements in a list.
- Implement a recursive function to find the largest element in a list.
- Print the first 10 elements of the Fibonacci series using recursion.
- Implement factorial using recursion.
- Implement a recursive function to reverse a string.
- Remove a character from a string using recursion.
- Remove even numbers from a list using recursion.
- Implement binary search using recursion.
- Find the sum of even numbers using recursion.
- Find the sum of digits using recursion.
- Recursively remove a character from a string (e.g., remove "l" from "hello").
- Implement a recursion that recurs only 5 times.
- Implement a palindrome check using recursion.
- Print the Fibonacci series under a limit using recursion.
- Remove duplicates from a linked list using recursion.

Binary Search

Theory

- What is binary search?
- What is the difference between binary search and linear search?
- What is the complexity of binary search?
- What is the limitation of binary search?
- What is the best-case time complexity of binary search?
- What is the worst-case time complexity of binary search?

Practical

- Implement binary search.
- Implement binary search using recursion.
- Implement binary search and replace a number with 0.
- Find the minimum from a rotated sorted list using binary search.

- Implement binary search on a list of strings.

Sliding Window and Two-Pointer Techniques

Theory

- What is the sliding window algorithm?
- What is the two-pointer technique?
- What is the fast and slow pointer approach?

Practical

- Implement the "Buy and Sell Stock" problem using the sliding window approach.
- Find the middle of a linked list in a single iteration using fast and slow pointers.
- Detect a cycle in a linked list using fast and slow pointers (Floyd's algorithm).
- Remove the nth node from the end of a linked list using the two-pointer technique.
- Find a sublist with the maximum number of elements in continuously increasing order.