✅ 3️⃣ Linked Lists 8️⃣ Reverse Linked List (Leetcode 206) — iterative pointer manipulation. 9️⃣ Merge Two Sorted Lists (Leetcode 21). 🔟 Linked List Cycle (Leetcode 141) — slow & fast pointer (Floyd’s algo). give me questions related to this.

Great! You’re focusing on fundamental and super important linked list questions. Let’s expand on these concepts so you fully master them.

Here’s a **list of related questions** for each topic (Reversing, Merging, Cycle detection), plus a few advanced variations to deepen your understanding:

## ✅ 8️⃣ Reverse Linked List (Leetcode 206)

**Core concept:** Iterative pointer manipulation (prev, curr, next).

### Related & follow-up questions

1️⃣ **Reverse a Linked List (recursive)** — try writing a recursive version.  
2️⃣ **Reverse a sublist (Leetcode 92: Reverse Linked List II)** — reverse only between positions *m* and *n*.  
3️⃣ **Reverse nodes in k-group (Leetcode 25)** — reverse every k nodes in the list.  
4️⃣ **Palindrome Linked List (Leetcode 234)** — reverse the second half and compare.  
5️⃣ **Swap Nodes in Pairs (Leetcode 24)** — similar manipulation logic.  
6️⃣ **Rotate List (Leetcode 61)** — reverse parts to rotate.

## ✅ 9️⃣ Merge Two Sorted Lists (Leetcode 21)

**Core concept:** Dummy node + iterative merging.

### Related & follow-up questions

1️⃣ **Merge K Sorted Lists (Leetcode 23)** — generalization using priority queue (heap) or divide & conquer.  
2️⃣ **Sort List (Leetcode 148)** — sort a linked list using merge sort.  
3️⃣ **Intersection of Two Linked Lists (Leetcode 160)** — different setup but involves merging-like pointer logic.  
4️⃣ **Add Two Numbers (Leetcode 2)** — merging digit by digit with carry.

## ✅ 🔟 Linked List Cycle (Leetcode 141)

**Core concept:** Slow & fast pointer (Floyd’s Tortoise and Hare).

### Related & follow-up questions

1️⃣ **Linked List Cycle II (Leetcode 142)** — find the node where the cycle begins.  
2️⃣ **Happy Number (Leetcode 202)** — same slow-fast pointer logic, but in number sequence.  
3️⃣ **Middle of the Linked List (Leetcode 876)** — slow & fast pointer to find the middle node.  
4️⃣ **Detect intersection of two linked lists (Leetcode 160 alternative approach)** — can also be done using slow-fast idea.  
5️⃣ **Remove cycle from linked list (follow-up)** — after detecting a cycle, remove it safely.

## 💡 Bonus linked list practice questions

✅ **Remove N-th Node From End of List (Leetcode 19)** — uses two pointers.  
✅ **Reorder List (Leetcode 143)** — split, reverse second half, merge.  
✅ **Partition List (Leetcode 86)** — split nodes by value.  
✅ **Copy List with Random Pointer (Leetcode 138)** — advanced pointer manipulation.  
✅ **Flatten a Multilevel Doubly Linked List (Leetcode 430)** — more pointer variants.

### 💥 Recommended sequence to practice

1️⃣ Master reverse iterative & recursive.  
2️⃣ Do all cycle-related questions.  
3️⃣ Move to merge (merge 2, then k, then sort list).  
4️⃣ Pick advanced reorder/partition/flatten for final strengthening.

✅ If you'd like, I can also prepare **step-by-step visual diagrams**, code templates (Python, C++, Java), or a mini "roadmap sheet" summarizing these problems with links and difficulty. Want me to do that for you?

✅ 5️⃣ Trees (must!) 1️⃣3️⃣ Binary Tree Inorder Traversal (Leetcode 94) — recursion & iterative. 1️⃣4️⃣ Maximum Depth of Binary Tree (Leetcode 104). 1️⃣5️⃣ Lowest Common Ancestor of a BST (Leetcode 235) — recursive logic. okay givce me question that would help me in the interview

Absolutely! You're targeting **core binary tree problems** that are **must-knows** for interviews — recursion, traversal, depth, and common ancestor. These form the **foundation for tree-based DSA rounds**.

### ✅ 5️⃣ Binary Trees — Core Interview Questions

Let’s expand your three focus problems and include **high-yield related questions** grouped by concept. These will boost your tree intuition and prep you for any twist.

## 🌿 1️⃣3️⃣ Binary Tree Inorder Traversal (Leetcode 94)

**Core:** Inorder traversal (Left → Root → Right)

### Related traversal questions:

1. **Preorder Traversal (Leetcode 144)** — Root → Left → Right
2. **Postorder Traversal (Leetcode 145)** — Left → Right → Root
3. **Level Order Traversal (Leetcode 102)** — BFS using a queue
4. **Zigzag Level Order (Leetcode 103)** — alternate levels
5. **Iterative Traversals (all types)** — stack-based simulation
6. **Morris Traversal (advanced)** — inorder with O(1) space (no recursion or stack)

📌 **Interview Tip:** Be able to do **both recursive & iterative** versions. That’s often an interviewer prompt:

“Now do it without recursion.”

## 🌲 1️⃣4️⃣ Maximum Depth of Binary Tree (Leetcode 104)

**Core:** Depth = longest path from root to leaf  
→ Use DFS recursion or BFS level-order

### Related depth/height structure problems:

1. **Minimum Depth (Leetcode 111)**
2. **Diameter of Binary Tree (Leetcode 543)** — longest path between any 2 nodes
3. **Balanced Binary Tree (Leetcode 110)** — check if height diff ≤1
4. **Same Tree / Subtree (Leetcode 100/572)** — compare two trees
5. **Invert Binary Tree (Leetcode 226)** — classic mirror recursion
6. **Check Tree Symmetry (Leetcode 101)**

## 🌳 1️⃣5️⃣ Lowest Common Ancestor of a BST (Leetcode 235)

**Core:** Use BST property — if p < root < q, root is LCA.

### Related & advanced questions:

1. **LCA in Binary Tree (Leetcode 236)** — no BST property, general tree
2. **Distance Between 2 Nodes in a BST** — use LCA to compute
3. **Path to a Node in Binary Tree** — find path from root to node
4. **LCA Follow-up**:
   * What if nodes don’t exist?
   * What if multiple LCAs?

## 🌟 Bonus High-Impact Tree Questions for Interviews

These are common in FAANG and product-based interviews:

| 💡 Question | 🔢 Leetcode | Concept |
| --- | --- | --- |
| Construct Binary Tree from Inorder & Preorder/Postorder | 105 / 106 | Recursion + HashMap |
| Serialize and Deserialize Binary Tree | 297 | BFS or DFS encoding |
| Binary Tree Right Side View | 199 | Level order + tracking rightmost |
| Binary Tree Level Order Traversal II | 107 | Bottom-up BFS |
| Validate Binary Search Tree | 98 | Inorder validation |
| Kth Smallest in BST | 230 | Inorder + counter |
| Convert BST to Sorted DLL | GFG/Custom | Recursion with prev tracking |

## 🧠 Practice Order (What to do first)

1. ✅ **Traversal mastery** — both recursive and iterative
2. ✅ **Depth-based logic** — max depth, diameter, balanced check
3. ✅ **BST property usage** — LCA, validation, Kth smallest
4. ✅ **Reconstruction** — build tree from traversal
5. ✅ **Serialization** — key for interviews involving persistence or networks
6. ✅ **Advanced** — symmetry, mirror, flatten, etc.

Would you like:

* A **visual roadmap table** with links and difficulty?
* A **C++ code template sheet** for all major tree patterns?
* Or maybe a **mock interview round** just for trees?

Let me know!

i need more practsice on the tree and where recursion logic have to e applied so goive me probelms on that

Absolutely, Rakshith — if you want to **master trees and recursion**, you need problems that **force you to think recursively**, either in:

* Traversing the tree
* Building/transforming the tree
* Using return values from subtrees to make decisions at the current node

## 🌳🔥 Ultimate Recursion-Based Tree Problem Set

Each of these will **strengthen recursion thinking** on trees — base case, recursive calls, and combining results.

### 🔁 Traversal & Base Recursion

Start here — build your recursion fluency with standard traversals and simple use cases.

1. **Binary Tree Inorder Traversal** — [Leetcode 94]
2. **Preorder / Postorder Traversal** — [Leetcode 144 / 145]
3. **Invert Binary Tree** — [Leetcode 226]
4. **Maximum Depth of Binary Tree** — [Leetcode 104]
5. **Sum of Left Leaves** — [Leetcode 404]
6. **Leaf-Similar Trees** — [Leetcode 872]

### 🧠 Use Return Values from Subtrees

Learn to return something meaningful from recursive calls.

1. **Same Tree** — [Leetcode 100]
2. **Subtree of Another Tree** — [Leetcode 572]
3. **Balanced Binary Tree** — [Leetcode 110]
4. **Diameter of Binary Tree** — [Leetcode 543]
5. **Path Sum** — [Leetcode 112]
6. **Has Path Sum II (print paths)** — [Leetcode 113]
7. **Symmetric Tree** — [Leetcode 101]

### 📚 Construct or Transform Tree (Recursion Heavy)

Here you **build** or **convert** trees — these boost recursion logic a lot.

1. **Construct Binary Tree from Preorder and Inorder Traversal** — [Leetcode 105]
2. **Construct from Postorder and Inorder** — [Leetcode 106]
3. **Convert Sorted Array to BST** — [Leetcode 108]
4. **Flatten Binary Tree to Linked List** — [Leetcode 114]
5. **Binary Tree to DLL** — [GFG problem]
6. **Serialize and Deserialize Binary Tree** — [Leetcode 297]
7. **Construct Tree from String** — [Leetcode 536]

### 📏 Ancestors, Distance, LCA — More Advanced Use of Recursion

These require backtracking or using return values to find paths or special nodes.

1. **Lowest Common Ancestor (Binary Tree)** — [Leetcode 236]
2. **Distance Between 2 Nodes in Binary Tree** — [Custom]
3. **Find All Nodes Distance K from Target** — [Leetcode 863]
4. **Path Sum III (Count paths anywhere)** — [Leetcode 437]
5. **Count Good Nodes in Binary Tree** — [Leetcode 1448]
6. **Find Duplicate Subtrees** — [Leetcode 652]

### ✨ Bonus Practice to Push Your Recursion Boundaries

1. **Burning Tree Problem** — [GFG / Hard]
2. **Find Max Sum Path Between Any Two Nodes** — [Leetcode 124]
3. **All Nodes at Distance K in Binary Tree** — [Leetcode 863]
4. **Recover Binary Search Tree** — [Leetcode 99]
5. **BST to Greater Tree** — [Leetcode 538]
6. **Maximum Width of Binary Tree** — [Leetcode 662]

### 🧩 Pattern Recognition Tips for Recursion in Trees:

* **If you must visit all nodes → recursion** is natural
* **If you need to combine results from left/right → return something** from recursive calls
* **If tree must be built/flattened/converted → recursion is key**
* **In LCA-type problems → use recursion to bubble up info**

### 🚀 Want Even More?

* I can create a **progress checklist with links**
* Or a **visual map** of when to apply each recursion idea
* Or a set of **mock problems with only problem statements**, so you solve them raw

Let me know how you'd like to continue.

Select One A.A.S HS GED A.S A.A. B.S. B.A. M.J. MBA M.S. M.A. PhD JD hich suits me

To suggest which degree suits you best, I need to know:

1. **Your current education level** (e.g., 12th grade, undergrad, postgrad)?
2. **Your field or career goal** (e.g., software development, data science, business, law)?
3. **Are you in India or abroad?** Because some of these degrees (like A.A.S, GED) are U.S.-specific.

But here's a quick breakdown of what each option means:

| **Abbreviation** | **Full Form** | **Type** | **Best for** |
| --- | --- | --- | --- |
| **A.A.** | Associate of Arts | 2-year undergrad | General arts, humanities (USA) |
| **A.S.** | Associate of Science | 2-year undergrad | Science, pre-tech (USA) |
| **A.A.S.** | Associate of Applied Science | 2-year technical | Job-ready skills (USA) |
| **B.A.** | Bachelor of Arts | 3–4 year degree | Arts, psychology, business |
| **B.S.** | Bachelor of Science | 3–4 year degree | STEM (e.g., CS, engineering) |
| **HS** | High School | Pre-college | High school diploma |
| **GED** | General Educational Development | Alt. to high school | US-based high school equivalent |
| **M.A.** | Master of Arts | Postgrad | Humanities, social science |
| **M.S.** | Master of Science | Postgrad | STEM, research, tech |
| **MBA** | Master of Business Administration | Postgrad professional | Business, leadership |
| **M.J.** | Master of Journalism | Postgrad | Media, journalism |
| **PhD** | Doctor of Philosophy | Research Doctorate | Research, teaching |
| **JD** | Juris Doctor | Law degree | Becoming a lawyer (US) |

### 🔍 If you're from India and into tech:

* You're likely pursuing or have completed a **B.E. / B.Tech (≅ B.S.)**
* After that, your ideal further options might be:
  + **M.S.** — for tech or CS specialization (especially abroad)
  + **MBA** — if you want to move into management or business
  + **PhD** — if you're into research or teaching

### ✅ Based on your past chats (tech, Python, audio projects, internships):

**Best match: B.S. (Bachelor of Science)** if you're still in undergrad  
Then aim for:  
— M.S. (Master of Science) for tech  
— or MBA if you're leaning toward product/business side

Let me know your **current degree + goal**, and I’ll confirm which fits best.

okay lets focus on bst

Perfect! Binary Search Trees (BSTs) are **super important in interviews** — they test both your understanding of tree recursion *and* how to leverage BST properties (left < root < right).

## ✅ What You Must Master in BSTs

We’ll go through these categories with Leetcode problems and key concepts:

### 🔹 1. **Basic BST Operations**

**Goal:** Understand structure and recursive nature.

| Problem | Concept |
| --- | --- |
| 🔸 Search in BST – Leetcode 700 | Classic recursive vs iterative |
| 🔸 Insert into BST – Leetcode 701 | Tree-building recursion |
| 🔸 Delete Node in BST – Leetcode 450 | All 3 cases (0, 1, 2 children) |
| 🔸 Validate BST – Leetcode 98 | Inorder traversal or bounds check |

### 🔹 2. **BST Property-Based Traversals**

**Use**: Since inorder traversal of BST gives sorted array

| Problem | Concept |
| --- | --- |
| 🔸 Kth Smallest in BST – Leetcode 230 | Inorder + counter |
| 🔸 Two Sum IV – Leetcode 653 | DFS + set or 2-pointer with inorder |
| 🔸 [Convert BST to Sorted Doubly Linked List – GFG / Custom] | Inorder + pointer management |
| 🔸 BST to Greater Tree – Leetcode 538 | Reverse Inorder + cumulative sum |

### 🔹 3. **Lowest Common Ancestor**

* BST allows you to **reduce search space** easily.

| Problem | Concept |
| --- | --- |
| 🔸 LCA in BST – Leetcode 235 | If both nodes < or > current → recurse accordingly |
| 🔸 LCA in Binary Tree – Leetcode 236 | General tree — no BST assumption |

### 🔹 4. **Range-Based / Sum-Based Problems**

| Problem | Concept |
| --- | --- |
| 🔸 Range Sum of BST – Leetcode 938 | Prune left/right based on bounds |
| 🔸 Trim a BST – Leetcode 669 | Return subtree within range |

### 🔹 5. **Constructing BST**

| Problem | Concept |
| --- | --- |
| 🔸 Sorted Array to BST – Leetcode 108 | Mid element = root |
| 🔸 Sorted List to BST – Leetcode 109 | Find middle of list recursively |

### 🔹 6. **Advanced**

| Problem | Concept |
| --- | --- |
| 🔸 Recover BST – Leetcode 99 | Fix 2 swapped nodes in O(1) space |
| 🔸 Unique BSTs – Leetcode 96 | Catalan number DP |
| 🔸 Generate Unique BSTs – Leetcode 95 | Recursively build all combinations |

## 🧠 Recursion Pattern for BST

Most BST recursion follows:

cppCopyEditTreeNode\* solve(TreeNode\* root, int key) {  
 if (!root) return base\_case;  
  
 if (key < root->val)  
 root->left = solve(root->left, key);  
 else if (key > root->val)  
 root->right = solve(root->right, key);  
 else  
 // process when found (e.g., delete, return)  
}

## 🔄 Suggested Practice Order

1. ✅ Search, Insert, Delete in BST (700, 701, 450)
2. ✅ Validate BST (98), Inorder (94)
3. ✅ LCA in BST (235), LCA in Binary Tree (236)
4. ✅ Kth Smallest (230), Range Sum (938), Trim BST (669)
5. ✅ Sorted Array to BST (108), Sorted List to BST (109)
6. ✅ Recover BST (99), Generate Unique BSTs (95)

### ⚡ Want help?

* I can make you a **Google Sheet with links + status tracker**
* Or give you a **daily problem schedule** for BST-only
* Or generate **custom mock BST coding questions**

Let me know how you'd like to practice next!

okay give me loist of problems i did in linled binary tree with their links

Awesome, Rakshith! Based on our previous conversations, here's a summary of **linked list and binary tree problems** you’ve either done or discussed, along with **Leetcode links** so you can revisit or track your practice.

## ✅ **Linked List Problems You've Done / Discussed**

| # | Problem | Link | Concept |
| --- | --- | --- | --- |
| 1️⃣ | Reverse Linked List | Leetcode 206 | Iterative/recursive reversal |
| 2️⃣ | Merge Two Sorted Lists | Leetcode 21 | Dummy node + merging |
| 3️⃣ | Linked List Cycle | Leetcode 141 | Floyd’s Cycle Detection |
| 4️⃣ | Linked List Cycle II | Leetcode 142 | Find starting node of cycle |
| 5️⃣ | Remove N-th Node From End | Leetcode 19 | Two pointers |
| 6️⃣ | Reorder List | Leetcode 143 | Split → reverse → merge |
| 7️⃣ | Add Two Numbers | Leetcode 2 | Digit-wise addition |
| 8️⃣ | Swap Nodes in Pairs | Leetcode 24 | Recursion or pointer manipulation |
| 9️⃣ | Sort List | Leetcode 148 | Merge Sort on linked list |
| 🔟 | Copy List with Random Pointer | Leetcode 138 | Deep copy with hash map |
| 1️⃣1️⃣ | Intersection of Two Linked Lists | Leetcode 160 | Length alignment |

## ✅ **Binary Tree Problems You've Done / Discussed**

| # | Problem | Link | Concept |
| --- | --- | --- | --- |
| 1️⃣3️⃣ | Binary Tree Inorder Traversal | Leetcode 94 | Recursive + iterative |
| 1️⃣4️⃣ | Maximum Depth of Binary Tree | Leetcode 104 | DFS recursion |
| 1️⃣5️⃣ | Lowest Common Ancestor of BST | Leetcode 235 | BST-based LCA |
| 1️⃣6️⃣ | Symmetric Tree | Leetcode 101 | Mirror recursion |
| 1️⃣7️⃣ | Diameter of Binary Tree | Leetcode 543 | Max path length |
| 1️⃣8️⃣ | Balanced Binary Tree | Leetcode 110 | Postorder + height check |
| 1️⃣9️⃣ | Path Sum | Leetcode 112 | Root-to-leaf sum |
| 2️⃣0️⃣ | Invert Binary Tree | Leetcode 226 | Swap left/right recursively |
| 2️⃣1️⃣ | Validate Binary Search Tree | Leetcode 98 | Inorder or min/max bounds |
| 2️⃣2️⃣ | Construct Binary Tree from Inorder & Preorder | Leetcode 105 | Recursion + hash map |
| 2️⃣3️⃣ | Sorted Array to BST | Leetcode 108 | Mid = root strategy |
| 2️⃣4️⃣ | Kth Smallest Element in BST | Leetcode 230 | Inorder traversal + counter |

Would you like me to:

* ✅ Make a **Google Sheet** with these for tracking?
* 📅 Create a **practice schedule** from these?
* 📊 Highlight what’s left to learn in both linked lists and trees?

Let me know how you'd like to move forward!

give me word of it

Sure! Here's the list of **Linked List** and **Binary Tree** problems you've done or discussed, formatted as a clean **Word-style document** that you can paste into MS Word or Google Docs:

## ✅ **Linked List Problems – Practice Summary**

| S.No | Problem Name | Leetcode Link | Key Concept |
| --- | --- | --- | --- |

1. **Reverse Linked List** | Leetcode 206 | Iterative and Recursive Reversal |
2. **Merge Two Sorted Lists** | Leetcode 21 | Dummy Node + Merging |
3. **Linked List Cycle** | Leetcode 141 | Floyd’s Cycle Detection |
4. **Linked List Cycle II** | Leetcode 142 | Detect Cycle Start Node |
5. **Remove N-th Node From End** | Leetcode 19 | Two-Pointer Technique |
6. **Reorder List** | Leetcode 143 | Reverse + Merge Strategy |
7. **Add Two Numbers** | Leetcode 2 | Digit-by-Digit Addition |
8. **Swap Nodes in Pairs** | Leetcode 24 | Recursive or Pointer Swapping |
9. **Sort List** | Leetcode 148 | Merge Sort on Linked List |
10. **Copy List with Random Pointer** | Leetcode 138 | Deep Copy with Hash Map |
11. **Intersection of Two Linked Lists** | Leetcode 160 | Length Alignment Trick |

## 🌳 **Binary Tree / BST Problems – Practice Summary**

| S.No | Problem Name | Leetcode Link | Key Concept |
| --- | --- | --- | --- |

1. **Binary Tree Inorder Traversal** | Leetcode 94 | Recursive + Iterative |
2. **Maximum Depth of Binary Tree** | Leetcode 104 | DFS Recursion |
3. **Lowest Common Ancestor of BST** | Leetcode 235 | BST-Based LCA |
4. **Symmetric Tree** | Leetcode 101 | Mirror Comparison |
5. **Diameter of Binary Tree** | Leetcode 543 | Max Path in Subtree |
6. **Balanced Binary Tree** | Leetcode 110 | Height-Based Balance Check |
7. **Path Sum** | Leetcode 112 | Root-to-Leaf Path Sum |
8. **Invert Binary Tree** | Leetcode 226 | Recursive Swapping |
9. **Validate Binary Search Tree** | Leetcode 98 | Inorder or Min/Max Bounds |
10. **Construct Tree from Preorder & Inorder** | Leetcode 105 | Build Tree via Recursion |
11. **Convert Sorted Array to BST** | Leetcode 108 | Divide and Conquer |
12. **Kth Smallest in BST** | Leetcode 230 | Inorder + Count |

Let me know if you'd like me to:

* Export this to .docx format for direct download
* Add a checkbox column so you can track solved vs unsolved
* Generate similar lists for graphs, arrays, etc.

Would you like me to export this to a Word file for you now?