Complete DSA Roadmap - From Zero to Hero (Java)

A Step-by-Step Guide with Theory and Practice Problems

📋 Table of Contents

Java Fundamentals Revision

Week 1: Strengthening Basics

Day 1-2: If-Else Statements

Theory:

Conditional Statements allow your program to make decisions and execute different code based on conditions.

Syntax:

```java
if (condition) {
    // code if condition is true
} else if (anotherCondition) {
    // code if anotherCondition is true
} else {
    // code if all conditions are false
}
```

Ternary Operator:

```java
result = (condition) ? valueIfTrue : valueIfFalse;
```

Switch Statement:

```java
switch (variable) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
}
```

Practice Problems (15 problems):

Easy Level:

1. Check if a number is positive, negative, or zero
2. Check if a number is even or odd
3. Find the largest of two numbers
4. Find the largest of three numbers
5. Check if a year is a leap year
6. Grade calculator (marks to grade conversion)
7. Check if a character is vowel or consonant
8. Check if a number is divisible by both 3 and 5

Medium Level: 9. Day of week using switch case (1-7 → Monday-Sunday) 10. Month name and days in that month 11. Simple calculator using switch 12. Check if a triangle is valid (sum of two sides > third side) 13. Classify triangle (equilateral, isosceles, scalene) 14. Electricity bill calculator (different slabs) 15. Quadrant finder (x, y coordinates)

Day 3-5: Loops

Theory:

1. For Loop:

```java
for (initialization; condition; increment/decrement) {
    // code to execute
}
```

2. While Loop:

```java
while (condition) {
    // code to execute
}
```

3. Do-While Loop:

```java
do {
    // code to execute
} while (condition);
```

Important Concepts:

- **Break:** Exit the loop immediately
- **Continue:** Skip current iteration and move to next
- **Nested Loops:** Loop inside another loop

Practice Problems (25 problems):

Basic Patterns:

1. Print numbers from 1 to N
2. Print numbers from N to 1
3. Sum of first N natural numbers
4. Print even numbers from 1 to N

5. Print odd numbers from 1 to N
6. Factorial of a number
7. Multiplication table of a number
8. Count digits in a number
9. Sum of digits in a number
10. Reverse a number

Number Properties: 11. Check if a number is prime 12. Print all prime numbers up to N 13. Check if a number is palindrome 14. Check if a number is Armstrong number ($153 = 1^3 + 5^3 + 3^3$) 15. Find GCD (Greatest Common Divisor) of two numbers 16. Find LCM (Least Common Multiple) of two numbers 17. Find all divisors of a number 18. Check if a number is perfect number (sum of divisors = number) 19. Print Fibonacci series up to N terms

Pattern Printing (Nested Loops): 20. Square pattern of stars

```
****
****
****
****
```

21. Right-angled triangle

```
*
**
***
****
```

22. Inverted right-angled triangle

```
****
***
**
*
```

23. Pyramid pattern

```
  *
 ***
*****
*******
```

24. Diamond pattern

```
  *
 ***
*****
 ***
  *
```

25. Number pyramid

```
   1
  121
 12321
1234321
```

Day 6-7: Functions/Methods

Theory:

Method Definition:

```java
accessModifier returnType methodName(parameters) {
    // method body
    return value; // if returnType is not void
}
```

Types of Methods:

1. **No return, No parameters**
2. **No return, With parameters**
3. **With return, No parameters**

4. **With return, With parameters**

Important Concepts:

- Method Overloading (same name, different parameters)
- Pass by Value in Java
- Recursion (method calling itself)

Practice Problems (15 problems):

Basic Functions:

1. Create a method to check if number is even
2. Create a method to find maximum of three numbers
3. Create a method to calculate factorial
4. Create a method to check if number is prime
5. Create a method to print Fibonacci series
6. Create a method to calculate power (x^n)
7. Create a method to reverse a number
8. Create a method to check palindrome

Method Overloading: 9. Overload area() method for circle, rectangle, triangle 10. Overload sum() method for 2, 3, and 4 integers

Recursion: 11. Factorial using recursion 12. Fibonacci using recursion 13. Sum of digits using recursion 14. Power calculation using recursion 15. Print numbers 1 to N using recursion

---

Phase 1: Problem Solving Basics

Week 2-3: Time Complexity and Arrays

Day 1-2: Time and Space Complexity

Theory:

Big O Notation describes how the runtime or space requirements grow as input size increases.

Common Time Complexities:

- **O(1)** - Constant: Access array element
- **O(log n)** - Logarithmic: Binary search
- **O(n)** - Linear: Loop through array once
- **O(n log n)** - Linearithmic: Merge sort, quick sort
- **O(n^2)** - Quadratic: Nested loops
- **O(2^n)** - Exponential: Recursive fibonacci
- **O(n!)** - Factorial: Generate permutations

How to Calculate:

1. Count the basic operations
2. Drop constants: O(2n) → O(n)
3. Keep dominant term: O(n^2 + n) → O(n^2)

Space Complexity: Consider memory used by:

- Variables
- Data structures
- Recursive call stack

Practice Problems (10 problems):

1. Find time complexity of single loop
2. Find time complexity of nested loops
3. Find time complexity of loop with i = i * 2
4. Find time complexity of recursive factorial
5. Find time complexity of binary search
6. Analyze space complexity of array creation
7. Analyze space complexity of recursive function
8. Compare two algorithms with different complexities
9. Optimize O(n^2) solution to O(n)
10. Identify worst-case complexity in given code

Day 3-7: Arrays

Theory:

Array is a collection of elements of the same type stored in contiguous memory locations.

Declaration and Initialization:

```java
// Declaration
int[] arr;
int arr[];

// Initialization
arr = new int[5]; // size 5, default values 0

// Declaration + Initialization
int[] arr = {1, 2, 3, 4, 5};
int[] arr = new int[]{1, 2, 3, 4, 5};
```

Important Operations:

- Access: O(1)
- Search: O(n)
- Insert at end: O(1)
- Insert at beginning/middle: O(n)
- Delete: O(n)

Common Array Methods:

```java
arr.length        // size of array
Arrays.sort(arr)  // sort array
Arrays.toString(arr) // convert to string
Arrays.fill(arr, value) // fill with value
```

Practice Problems (35 problems):

Basic Operations (Day 3):

1. Find the largest element in array
2. Find the smallest element in array
3. Find second largest element
4. Find second smallest element
5. Reverse an array
6. Count even and odd numbers
7. Sum of all elements
8. Average of all elements
9. Find element at given index
10. Linear search in array

Array Manipulation (Day 4): 11. Left rotate array by 1 position 12. Left rotate array by D positions 13. Right rotate array by 1 position 14. Right rotate array by D positions 15. Move all zeros to end 16. Move all negative to left 17. Rearrange positive and negative alternatively 18. Find missing number in array (1 to N) 19. Find the repeating and missing number 20. Remove duplicates from sorted array

Subarray Problems (Day 5): 21. Maximum subarray sum (Kadane's Algorithm) 22. Minimum subarray sum 23. Subarray with given sum 24. Longest subarray with sum K 25. Count subarrays with given sum 26. Maximum product subarray 27. Print all subarrays 28. Count all subarrays with sum 0

Sorting and Searching (Day 6): 29. Find majority element (appears > n/2 times) 30. Sort array of 0s, 1s, and 2s (Dutch National Flag) 31. Find intersection of two arrays 32. Find union of two arrays 33. Merge two sorted arrays 34. Check if array is sorted 35. Equilibrium point in array (sum of left = sum of right)

Advanced Problems (Day 7): 36. Leaders in array (greater than all right elements) 37. Stock buy and sell (one transaction) 38. Stock buy and sell (multiple transactions) 39. Trapping rain water 40. Maximum difference (arr[j] - arr[i] where j > i) 41. Minimum jumps to reach end 42. Longest consecutive sequence 43. Find all pairs with given sum 44. Find triplet with given sum 45. Container with most water

---

Week 4: Strings

Day 1-2: String Theory

Theory:

Strings in Java:

- Strings are immutable (cannot be changed)

- Stored in String Pool for memory efficiency
- Use StringBuilder/StringBuffer for mutable strings

String Creation:

```java
String s1 = "Hello";          // String literal
String s2 = new String("Hello"); // String object
```

Important Methods:

```java
length()                  // size of string
charAt(index)             // character at index
substring(start, end)     // extract substring
indexOf(char/string)      // find index
contains(string)          // check if contains
equals(string)            // compare content
equalsIgnoreCase(string)  // compare ignoring case
toLowerCase()             // convert to lowercase
toUpperCase()             // convert to uppercase
trim()                    // remove spaces from ends
split(delimiter)          // split into array
replace(old, new)         // replace characters
```

StringBuilder:

```java
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");      // add at end
sb.insert(5, ",");        // insert at position
sb.delete(5, 6);          // delete range
sb.reverse();             // reverse string
sb.toString();            // convert to String
```

Practice Problems (30 problems):

Basic Operations (Day 1):

1. Reverse a string

2. Check if string is palindrome
3. Count vowels and consonants
4. Count words in a string
5. Convert first letter of each word to uppercase
6. Count frequency of each character
7. Find first non-repeating character
8. Find first repeating character
9. Remove all spaces from string
10. Remove duplicates from string

String Manipulation (Day 2): 11. Check if two strings are anagrams 12. Check if string is rotation of another 13. Longest substring without repeating characters 14. Longest repeating character replacement 15. Check if string contains all unique characters 16. Reverse words in a string 17. Reverse each word in string 18. Check if strings are rotations 19. Count palindromic substrings 20. Longest palindromic substring

Pattern Matching (Day 3): 21. Check if string matches pattern 22. Find all occurrences of substring 23. Implement strStr() - find needle in haystack 24. Valid palindrome (ignoring non-alphanumeric) 25. Check if parentheses are valid 26. Longest common prefix 27. Compare version numbers 28. Roman to integer conversion 29. Integer to Roman conversion 30. String compression (aabbbcc → a2b3c2)

Phase 2: Basic Data Structures

Week 5-6: Searching and Sorting

Day 1-3: Searching Algorithms

Theory:

1. Linear Search:

- Search element one by one
- Time Complexity: O(n)
- Space Complexity: O(1)

```
java
```

```
int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

2. Binary Search:

- Works only on sorted arrays
- Divide search space in half each time
- Time Complexity: O(log n)
- Space Complexity: O(1)

```
java

int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Binary Search Variations:

- Lower Bound: First position where element >= target
- Upper Bound: First position where element > target
- Search in rotated sorted array
- Search in 2D matrix

Practice Problems (20 problems):

Basic Search (Day 1):

1. Implement linear search

2. Implement binary search (iterative)
3. Implement binary search (recursive)
4. Find first occurrence of element
5. Find last occurrence of element
6. Count occurrences in sorted array
7. Search insert position
8. Find floor and ceiling in sorted array

Binary Search Applications (Day 2): 9. Find peak element 10. Search in rotated sorted array 11. Find minimum in rotated sorted array 12. Search in 2D matrix (row-column sorted) 13. Find square root using binary search 14. Find nth root of a number 15. Koko eating bananas

Advanced Binary Search (Day 3): 16. Aggressive cows problem 17. Book allocation problem 18. Painter's partition problem 19. Split array largest sum 20. Median of two sorted arrays

---

Day 4-7: Sorting Algorithms

Theory:

1. Bubble Sort:

- Compare adjacent elements and swap
- Time: $O(n^2)$, Space: $O(1)$

```java
void bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(arr, j, j+1);
            }
        }
    }
}
```

2. Selection Sort:

- Select minimum and place at beginning
- Time: $O(n^2)$, Space: O(1)

3. Insertion Sort:

- Insert elements in sorted portion
- Time: $O(n^2)$, Space: O(1)

4. Merge Sort:

- Divide and conquer
- Time: O(n log n), Space: O(n)

```java
void mergeSort(int[] arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

5. Quick Sort:

- Choose pivot, partition array
- Time: O(n log n) average, $O(n^2)$ worst
- Space: O(log n)

6. Counting Sort:

- Count occurrences
- Time: O(n + k), Space: O(k)

Practice Problems (15 problems):

Sorting Implementations (Day 4):

1. Implement bubble sort

2. Implement selection sort
3. Implement insertion sort
4. Implement merge sort
5. Implement quick sort
6. Implement counting sort

Sorting Applications (Day 5-6): 7. Sort array by frequency of elements 8. Sort 0s, 1s, and 2s 9. Sort even numbers before odd 10. Kth largest element 11. Kth smallest element 12. Merge two sorted arrays without extra space 13. Count inversions in array 14. Minimum difference between heights

Advanced Sorting (Day 7): 15. Merge intervals 16. Meeting rooms II 17. Sort array by parity 18. Largest number from array 19. Custom sort string 20. Relative sort array

---

Week 7-8: Hashing

Day 1-2: HashMap and HashSet Theory

Theory:

HashMap:

- Stores key-value pairs
- Average O(1) for insert, delete, search
- Uses hash function to compute index

```java
```

```java
HashMap<Key, Value> map = new HashMap<>();

// Operations
map.put(key, value);       // insert/update
map.get(key);              // retrieve value
map.containsKey(key);      // check if key exists
map.remove(key);           // delete
map.size();                // number of entries
map.keySet();              // all keys
map.values();              // all values
map.entrySet();            // all entries

// Iterate
for (Map.Entry<K, V> entry : map.entrySet()) {
    K key = entry.getKey();
    V value = entry.getValue();
}
```

HashSet:

- Stores unique elements
- O(1) for add, remove, contains

```java
HashSet<Type> set = new HashSet<>();

set.add(element);
set.remove(element);
set.contains(element);
set.size();
```

LinkedHashMap: Maintains insertion order **TreeMap:** Maintains sorted order (O(log n))

Practice Problems (30 problems):

Basic Hashing (Day 3-4):

1. Find frequency of each element
2. Find first repeating element
3. Find first non-repeating element

4. Two sum problem
5. Check if pair with given sum exists
6. Count pairs with given sum
7. Find all pairs with difference K
8. Longest consecutive sequence
9. Subarray with sum 0
10. Subarray with given sum

Intermediate (Day 5-6): 11. Largest subarray with sum K 12. Longest subarray with equal 0s and 1s 13. Count subarrays with equal 0s and 1s 14. Group anagrams together 15. Isomorphic strings 16. Check if array is subset of another 17. Union and intersection of arrays 18. Find symmetric pairs 19. Vertical sum in binary tree (preview) 20. Largest subarray with sum 0

Advanced (Day 7-8): 21. Count distinct elements in every window 22. Smallest window containing all characters 23. Longest substring with K unique characters 24. Longest substring without repeating characters 25. Subarrays with K different integers 26. Find itinerary from tickets 27. LRU Cache implementation (preview) 28. Design HashMap 29. Four sum problem 30. Subarray sum equals K (count)

Phase 3: Intermediate Data Structures

Week 9-11: Linked Lists

Day 1-2: Linked List Theory

Theory:

Singly Linked List:

```
java
```

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList {
    Node head;

    // Insert at beginning
    void insertAtBeginning(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }

    // Insert at end
    void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

    // Display
    void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
```

```
    }
}
```

Important Techniques:

- **Two Pointer / Fast-Slow:** Find middle, detect cycle
- **Dummy Node:** Simplify edge cases
- **Recursive Approach:** For reversal, deletion

Doubly Linked List:

```java
class DNode {
    int data;
    DNode prev;
    DNode next;
}
```

Circular Linked List:

- Last node points to first node

Practice Problems (35 problems):

Basic Operations (Day 3-4):

1. Create linked list from array
2. Insert at beginning
3. Insert at end
4. Insert at specific position
5. Delete first node
6. Delete last node
7. Delete node at position
8. Search for an element
9. Count nodes in linked list
10. Display linked list

Intermediate (Day 5-7): 11. Reverse a linked list (iterative) 12. Reverse a linked list (recursive) 13. Find middle of linked list 14. Detect cycle in linked list 15. Find starting point of cycle 16. Remove cycle from linked list 17. Check if linked list is palindrome 18. Remove nth node from end 19. Find intersection point of two lists 20. Merge two sorted linked lists

Advanced (Day 8-10): 21. Add two numbers represented as lists 22. Subtract two numbers as lists 23. Multiply two numbers as lists 24. Clone linked list with random pointer 25. Flatten a multilevel linked list 26. Sort linked list (merge sort) 27. Reverse linked list in groups of K 28. Rotate linked list 29. Delete node without head pointer 30. Segregate even and odd nodes 31. Remove duplicates from sorted list 32. Remove duplicates from unsorted list 33. Swap nodes in pairs 34. Rearrange linked list (L0→Ln→L1→Ln-1...) 35. Split circular linked list into two halves

---

Week 12-13: Stacks

Day 1-2: Stack Theory

Theory:

Stack: LIFO (Last In First Out)

Implementation using Array:

```java
```

```
class Stack {
    int[] arr;
    int top;
    int capacity;

    Stack(int size) {
        arr = new int[size];
        capacity = size;
        top = -1;
    }

    void push(int x) {
        if (isFull()) {
            System.out.println("Stack Overflow");
            return;
        }
        arr[++top] = x;
    }

    int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow");
            return -1;
        }
        return arr[top--];
    }

    int peek() {
        if (isEmpty()) return -1;
        return arr[top];
    }

    boolean isEmpty() {
        return top == -1;
    }

    boolean isFull() {
        return top == capacity - 1;
    }
}
```

Java Stack Class:

```java
Stack<Integer> stack = new Stack<>();
stack.push(element);
stack.pop();
stack.peek();
stack.isEmpty();
stack.size();
```

Common Applications:

- Expression evaluation
- Parentheses matching
- Function call management
- Undo/Redo operations

Practice Problems (25 problems):

Basic Stack (Day 3):

1. Implement stack using array
2. Implement stack using linked list
3. Implement two stacks in one array
4. Implement stack using queue
5. Get minimum element in O(1) (Min Stack)

Parentheses Problems (Day 4): 6. Valid parentheses 7. Minimum add to make parentheses valid 8. Minimum remove to make valid parentheses 9. Score of parentheses 10. Longest valid parentheses

Expression Problems (Day 5): 11. Evaluate postfix expression 12. Evaluate prefix expression 13. Infix to postfix conversion 14. Infix to prefix conversion 15. Basic calculator

Monotonic Stack (Day 6): 16. Next greater element 17. Next smaller element 18. Previous greater element 19. Previous smaller element 20. Stock span problem

Advanced (Day 7): 21. Largest rectangle in histogram 22. Maximal rectangle in binary matrix 23. Trapping rain water using stack 24. Remove K digits 25. Decode string 26. Simplify path 27. Daily temperatures 28. Car fleet 29. Asteroid collision 30. Validate stack sequences

---

Week 14: Queues

Day 1-2: Queue Theory

Theory:

Queue: FIFO (First In First Out)

Implementation using Array (Circular Queue):

```java
```

```
class Queue {
    int[] arr;
    int front, rear, size, capacity;

    Queue(int capacity) {
        this.capacity = capacity;
        arr = new int[capacity];
        front = 0;
        size = 0;
        rear = capacity - 1;
    }

    boolean isFull() {
        return size == capacity;
    }

    boolean isEmpty() {
        return size == 0;
    }

    void enqueue(int item) {
        if (isFull()) return;
        rear = (rear + 1) % capacity;
        arr[rear] = item;
        size++;
    }

    int dequeue() {
        if (isEmpty()) return -1;
        int item = arr[front];
        front = (front + 1) % capacity;
        size--;
        return item;
    }

    int peek() {
        if (isEmpty()) return -1;
        return arr[front];
    }
}
```

Java Queue Interface:

```java
Queue<Integer> queue = new LinkedList<>();
queue.offer(element);  // add
queue.poll();          // remove and return
queue.peek();          // view front
queue.isEmpty();
queue.size();
```

Deque (Double Ended Queue):

```java
Deque<Integer> deque = new ArrayDeque<>();
deque.offerFirst(element);
deque.offerLast(element);
deque.pollFirst();
deque.pollLast();
```

Practice Problems (20 problems):

Basic Queue (Day 3):

1. Implement queue using array
2. Implement queue using linked list
3. Implement circular queue
4. Implement queue using two stacks
5. Implement stack using two queues

Queue Applications (Day 4-5): 6. Reverse a queue 7. Reverse first K elements of queue 8. Interleave first and second half 9. Generate binary numbers from 1 to N 10. First negative integer in every window of size K

Deque Problems (Day 6): 11. Sliding window maximum 12. Maximum of all subarrays of size K 13. Minimum of all subarrays of size K 14. Sum of minimum and maximum of all subarrays

Advanced (Day 7): 15. LRU Cache implementation 16. Rotten oranges (BFS) 17. Time needed to inform all employees 18. Level order traversal (tree preview) 19. Design circular deque 20. Maximum sum circular subarray

---

Phase 4: Advanced Data Structures

Week 15-17: Binary Trees

Day 1-3: Tree Theory

Theory:

Binary Tree Node:

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}
```

Tree Terminology:

- **Root:** Top node
- **Leaf:** Node with no children
- **Height:** Longest path from node to leaf
- **Depth:** Distance from root to node
- **Level:** Nodes at same depth

Tree Traversals:

1. Inorder (Left-Root-Right):

```java
```

```java
void inorder(TreeNode root) {
    if (root == null) return;
    inorder(root.left);
    System.out.print(root.val + " ");
    inorder(root.right);
}
```

2. Preorder (Root-Left-Right):

```java
void preorder(TreeNode root) {
    if (root == null) return;
    System.out.print(root.val + " ");
    preorder(root.left);
    preorder(root.right);
}
```

3. Postorder (Left-Right-Root):

```java
void postorder(TreeNode root) {
    if (root == null) return;
    postorder(root.left);
    postorder(root.right);
    System.out.print(root.val + " ");
}
```

4. Level Order (BFS):

```java
```

```
void levelOrder(TreeNode root) {
    if (root == null) return;
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);

    while (!queue.isEmpty()) {
        TreeNode node = queue.poll();
        System.out.print(node.val + " ");

        if (node.left != null) queue.offer(node.left);
        if (node.right != null) queue.offer(node.right);
    }
}
```

Practice Problems (40 problems):

Basic Traversals (Day 4-5):

1. Inorder traversal (recursive)
2. Inorder traversal (iterative)
3. Preorder traversal (recursive)
4. Preorder traversal (iterative)
5. Postorder traversal (recursive)
6. Postorder traversal (iterative)
7. Level order traversal
8. Zigzag level order traversal
9. Reverse level order traversal
10. Vertical order traversal

Tree Properties (Day 6-7): 11. Height of binary tree 12. Diameter of binary tree 13. Check if tree is balanced 14. Count total nodes 15. Count leaf nodes 16. Sum of all nodes 17. Maximum element in tree 18. Minimum element in tree 19. Check if two trees are identical 20. Mirror/Symmetric tree check

Tree Views (Day 8): 21. Left view of tree 22. Right view of tree 23. Top view of tree 24. Bottom view of tree 25. Boundary traversal

Path Problems (Day 9-10): 26. Root to leaf paths 27. Print all paths with given sum 28. Maximum path sum (any node to any node) 29. Maximum path sum (leaf to leaf) 30. Lowest common ancestor

Tree Construction (Day 11): 31. Construct tree from inorder and preorder 32. Construct tree from inorder and postorder 33. Construct tree from parent array 34. Serialize and deserialize binary tree 35. Clone a binary tree

Advanced Problems (Day 12-13): 36