75 DSA Questions from Leet-Code

BY-Syntax Error

1. Arrays (10 Questions)

1. **1. Two Sum**
2. **121. Best Time to Buy and Sell Stock**
3. **88. Merge Sorted Array**
4. **217. Contains Duplicate**
5. **238. Product of Array Except Self**
6. **53. Maximum Subarray**
7. **15. 3Sum**
8. **56. Merge Intervals**
9. **11. Container With Most Water**
10. **48. Rotate Image**

Solution:

1. Two Sum (LeetCode 1)

Problem: Find two numbers in the array that add up to a specific target.
Solution:

```
java
Map<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < nums.length; i++) {

   int diff = target - nums[i];

   if (map.containsKey(diff)) return new int[]{map.get(diff), i};

   map.put(nums[i], i);

}
```

---

2. Best Time to Buy and Sell Stock (LeetCode 121)

Problem: Maximize profit by choosing one day to buy and another to sell.
Solution:

```
Java
int min = Integer.MAX_VALUE, profit = 0;
for (int price : prices) {
    min = Math.min(min, price);
    profit = Math.max(profit, price - min);
```

```
}
return profit;
```

---

3. Merge Sorted Array (LeetCode 88)

Problem: Merge two sorted arrays into one sorted array.
Solution:

```
java
int i = m - 1, j = n - 1, k = m + n - 1;

while (j >= 0)

   nums1[k--] = (i >= 0 && nums1[i] > nums2[j]) ? nums1[i--] : nums2[j--];
```

---

4. Contains Duplicate (LeetCode 217)

Problem: Check if an array contains duplicates.
Solution:

```
java
Set<Integer> set = new HashSet<>();

for (int num : nums)

   if (!set.add(num)) return true;

return false;
```

---

5. Product of Array Except Self (LeetCode 238)

Problem: Return an array where each element is the product of all other elements.
Solution:

```
java
int n = nums.length;

int[] res = new int[n];

res[0] = 1;

for (int i = 1; i < n; i++) res[i] = res[i - 1] * nums[i - 1];

int right = 1;

for (int i = n - 1; i >= 0; i--) {

   res[i] *= right;

   right *= nums[i];
```

```
}

return res;
```

---

6. Maximum Subarray (LeetCode 53)

Problem: Find the contiguous subarray with the largest sum.
Solution:

```
java
int max = nums[0], curr = nums[0];

for (int i = 1; i < nums.length; i++) {

   curr = Math.max(nums[i], curr + nums[i]);

   max = Math.max(max, curr);

}

return max;
```

---

7. 3Sum (LeetCode 15)

Problem: Find all unique triplets in the array which gives the sum of zero.
Solution:

```
java
Arrays.sort(nums);

for (int i = 0; i < nums.length - 2; i++) {

   if (i > 0 && nums[i] == nums[i - 1]) continue;

   int l = i + 1, r = nums.length - 1;

   while (l < r) {

      int sum = nums[i] + nums[l] + nums[r];

      if (sum == 0) list.add(Arrays.asList(nums[i], nums[l++], nums[r--]));

      else if (sum < 0) l++;

      else r--;

   }

}
```

---

8. Merge Intervals (LeetCode 56)

Problem: Merge overlapping intervals.P
Solution:

```
java
Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
List<int[]> res = new ArrayList<>();
int[] curr = intervals[0];
for (int i = 1; i < intervals.length; i++) {
    if (curr[1] >= intervals[i][0])
        curr[1] = Math.max(curr[1], intervals[i][1]);
    else {
        res.add(curr);
        curr = intervals[i];
    }
}
res.add(curr);
return res.toArray(new int[res.size()][]);
```

---

9. Container With Most Water (LeetCode 11)

Problem: Find the maximum water that can be trapped between two lines.
Solution:

```
java
int l = 0, r = height.length - 1, max = 0;

while (l < r) {

   max = Math.max(max, (r - l) * Math.min(height[l], height[r]));

   if (height[l] < height[r]) l++;

   else r--;

}

return max;
```

---

10. Rotate Image (LeetCode 48)

Problem: Rotate a matrix 90 degrees clockwise.
Solution:

```
Java
for (int i = 0; i < n; i++)
    for (int j = i; j < n; j++)
        swap(matrix, i, j, j, i);
for (int i = 0; i < n; i++)
    for (int j = 0, k = n - 1; j < k; j++, k--)
        swap(matrix, i, j, i, k);
```

2. Strings (10 Questions)

1. **20. Valid Parentheses**
2. **125. Valid Palindrome**
3. **242. Valid Anagram**
4. **49. Group Anagrams**
5. **5. Longest Palindromic Substring**
6. **76. Minimum Window Substring**
7. **28. Find the Index of the First Occurrence in a String**
8. **443. String Compression**
9. **14. Longest Common Prefix**
10. **459. Repeated Substring Pattern**

Solution:

1. Valid Parentheses (LeetCode 20)

```
java
Stack<Character> st = new Stack<>();

for (char c : s.toCharArray()) {

   if (c == '(' || c == '{' || c == '[') st.push(c);

   else if (st.isEmpty() ||

      (c == ')' && st.pop() != '(') ||

      (c == '}' && st.pop() != '{') ||

      (c == ']' && st.pop() != '[')) return false;

}

return st.isEmpty();
```

---

2. Valid Palindrome (LeetCode 125)

```
java
int i = 0, j = s.length() - 1;

while (i < j)

 {

   while (i < j && !Character.isLetterOrDigit(s.charAt(i))) i++;

   while (i < j && !Character.isLetterOrDigit(s.charAt(j))) j--;

   if (Character.toLowerCase(s.charAt(i++)) != Character.toLowerCase(s.charAt(j--)))

    return false;
```

```
}

return true;
```

---

3. Valid Anagram (LeetCode 242)

```
java
int[] freq = new int[26];

for (char c : s.toCharArray()) freq[c - 'a']++;

for (char c : t.toCharArray()) freq[c - 'a']--;

for (int f : freq) if (f != 0)  return false;

return true;
```

---

4. Group Anagrams (LeetCode 49)

```
java
Map<String, List<String>> map = new HashMap<>();

for (String str : strs) {

   char[] arr = str.toCharArray();

   Arrays.sort(arr);

   String key = new String(arr);

   map.computeIfAbsent(key, k -> new ArrayList<>()).add(str);

}

return new

 ArrayList<>(map.values());
```

---

5. Longest Palindromic Substring (LeetCode 5)

```
java
int start = 0, end = 0;

for (int i = 0; i < s.length(); i++) {

   int len1 = expand(s, i, i), len2 = expand(s, i, i + 1);

   int len = Math.max(len1, len2);

   if (len > end - start) {

      start = i - (len - 1) / 2;
```

```
        end = i + len / 2;
    }
}
return s.substring(start, end + 1);

int expand(String s, int l, int r) {
    while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) { l--; r++; }
    return r - l - 1;
}
```

---

6. Minimum Window Substring (LeetCode 76)

```
java
Map<Character, Integer> need = new HashMap<>(), window = new HashMap<>();
for (char c : t.toCharArray()) need.put(c, need.getOrDefault(c, 0) + 1);
int have = 0, needCount = need.size(), left = 0;
int minLen = Integer.MAX_VALUE, start = 0;

for (int right = 0; right < s.length(); right++) {
    char c = s.charAt(right);
    window.put(c, window.getOrDefault(c, 0) + 1);
    if (need.containsKey(c) && window.get(c).equals(need.get(c))) have++;
    while (have == needCount) {
        if (right - left + 1 < minLen) { minLen = right - left + 1; start = left; }
        char d = s.charAt(left++);
        window.put(d, window.get(d) - 1);
        if (need.containsKey(d) && window.get(d) < need.get(d)) have--;
    }
}
return minLen == Integer.MAX_VALUE ? "" : s.substring(start, start + minLen);
```

---

7. Find the Index of the First Occurrence (LeetCode 28)

```
java
for (int i = 0; i <= haystack.length() - needle.length(); i++) {
    if (haystack.substring(i, i + needle.length()).equals(needle)) return
i;
}
return -1;
```

---

8. String Compression (LeetCode 443)

```
java
int i = 0, idx = 0;

while (i < chars.length) {

   char c = chars[i];

   int count = 0;

   while (i < chars.length && chars[i] == c) { i++; count++; }

   chars[idx++] = c;

   if (count > 1)

      for (char ch : String.valueOf(count).toCharArray()) chars[idx++] = ch;

}

return idx;
```

---

9. Longest Common Prefix (LeetCode 14)

```
java
String prefix = strs[0];

for (int i = 1; i < strs.length; i++) {

   while (!strs[i].startsWith(prefix)) {

      prefix = prefix.substring(0, prefix.length() - 1);

      if (prefix.isEmpty()) return "";

   }

}

return prefix;
```

---

10. Repeated Substring Pattern (LeetCode 459)

```
Java
String str = s + s;
return str.substring(1, str.length() - 1).contains(s);
```

Linked Lists (8 Questions)

1. **206. Reverse Linked List**
2. **21. Merge Two Sorted Lists**
3. **19. Remove Nth Node From End of List**
4. **141. Linked List Cycle**
5. **2. Add Two Numbers**
6. **160. Intersection of Two Linked Lists**
7. **234. Palindrome Linked List**
8. **25. Reverse Nodes in k-Group**

Soluition:

Linked Lists

1. Reverse Linked List (LeetCode 206)

```
java
ListNode prev = null, curr = head;

while (curr != null) {

   ListNode next = curr.next;

   curr.next = prev;

   prev = curr;

   curr = next;

}

return prev;
```

---

2. Merge Two Sorted Lists (LeetCode 21)

```
java
ListNode dummy = new ListNode(0), tail = dummy;

while (l1 != null && l2 != null) {

   if (l1.val < l2.val) { tail.next = l1; l1 = l1.next; }

   else { tail.next = l2; l2 = l2.next; }

   tail = tail.next;
```

}

tail.next = (l1 != null) ? l1 : l2;

return dummy.next;

---

3. Remove Nth Node From End of List (LeetCode 19)

```
Java
ListNode dummy = new ListNode(0);
dummy.next = head;
ListNode fast = dummy, slow = dummy;
for (int i = 0; i <= n; i++) fast = fast.next;
while (fast != null) { fast = fast.next; slow = slow.next; }
slow.next = slow.next.next;
return dummy.next;
```

---

4. Linked List Cycle (LeetCode 141)

```
Java
ListNode slow = head, fast = head;
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow == fast) return true;
}
return false;
```

---

5. Add Two Numbers (LeetCode 2)

```
java
ListNode dummy = new ListNode(0), curr = dummy;

int carry = 0;

while (l1 != null || l2 != null || carry != 0) {

   int sum = carry;

   if (l1 != null) { sum += l1.val; l1 = l1.next; }

   if (l2 != null) { sum += l2.val; l2 = l2.next; }

   carry = sum / 10;

   curr.next = new ListNode(sum % 10);

   curr = curr.next;

}

return dummy.next;
```

---

6. Intersection of Two Linked Lists (LeetCode 160)

```
java
ListNode a = headA, b = headB;

while (a != b) {

   a = (a == null) ? headB : a.next;

   b = (b == null) ? headA : b.next;

}

return a;
```

---

7. Palindrome Linked List (LeetCode 234)

```
java
ListNode slow = head, fast = head;

while (fast != null && fast.next != null) { slow = slow.next; fast = fast.next.next; }


ListNode prev = null, curr = slow;

while (curr != null) {

   ListNode next = curr.next;

   curr.next = prev;

   prev = curr;

   curr = next;

}


ListNode left = head, right = prev;

while (right != null) {

   if (left.val != right.val) return false;

   left = left.next; right = right.next;

}

return true;
```

---

8. Reverse Nodes in k-Group (LeetCode 25)

```
Java
ListNode dummy = new ListNode(0);
dummy.next = head;
ListNode pre = dummy, end = dummy;

while (true) {
    for (int i = 0; i < k && end != null; i++) end = end.next;
    if (end == null) break;
    ListNode start = pre.next, next = end.next;
    end.next = null;
    pre.next = reverse(start);
    start.next = next;
    pre = start; end = pre;
}

ListNode reverse(ListNode head) {
    ListNode prev = null, curr = head;
    while (curr != null) {
        ListNode next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

4. Stacks and Queues (6 Questions)

1. **232. Implement Queue using Stacks**
2. **225. Implement Stack using Queues**
3. **155. Min Stack**
4. **739. Daily Temperatures**
5. **150. Evaluate Reverse Polish Notation**
6. **496. Next Greater Element Is**
7. **503. Next Greater Element II (LeetCode 503)**
8. **622.Circular Queue**

Solution:

Stacks and Queues

---

1. Implement Queue Using Stacks (LeetCode 232)

```
java
Stack<Integer> input = new Stack<>();

Stack<Integer> output = new Stack<>();


void push(int x) {

   input.push(x);

}


int pop() {

   if (output.isEmpty())

      while (!input.isEmpty()) output.push(input.pop());

   return output.pop();

}


int peek() {

   if (output.isEmpty())

      while (!input.isEmpty()) output.push(input.pop());

   return output.peek();

}


boolean empty() {

   return input.isEmpty() && output.isEmpty();

}
```

---

2. Implement Stack Using Queues (LeetCode 225)

```
java
Queue<Integer> q = new LinkedList<>();


void push(int x) {

   q.add(x);
```

```
    for (int i = 0; i < q.size() - 1; i++)
        q.add(q.remove());
}

int pop() { return q.remove(); }

int top() { return q.peek(); }

boolean empty() { return q.isEmpty();
 }
```

---

3. Min Stack (LeetCode 155)

```
java
Stack<Integer> st = new Stack<>();
Stack<Integer> minSt = new Stack<>();

void push(int x) {
    st.push(x);
    if (minSt.isEmpty() || x <= minSt.peek()) minSt.push(x);
}

void pop() {
    if (st.pop().equals(minSt.peek())) minSt.pop();
}

int top() { return st.peek(); }

int getMin() { return minSt.peek(); }
```

---

4. Daily Temperatures (LeetCode 739)

```
java
int[] res = new int[temps.length];

Stack<Integer> st = new Stack<>();


for (int i = 0; i < temps.length; i++) {

   while (!st.isEmpty() && temps[i] > temps[st.peek()]) {

      int idx = st.pop();

      res[idx] = i - idx;

   }

   st.push(i);

}

return res;
```

---

5. Evaluate Reverse Polish Notation (LeetCode 150)

```
java
int evalRPN(String[] tokens) {
    Stack<Integer> st = new Stack<>();

    for (String t : tokens) {
        if ("+-*/".contains(t)) {
            int b = st.pop();
            int a = st.pop();

            switch (t) {
                case "+": st.push(a + b); break;
                case "-": st.push(a - b); break;
                case "*": st.push(a * b); break;
                case "/": st.push(a / b); break; // integer division
            }
        } else {
            st.push(Integer.parseInt(t));
        }
    }
    return st.pop();
}
```

---

6. Next Greater Element I (LeetCode 496)

```
java
int[] nextGreaterElement(int[] nums1, int[] nums2) {

   Map<Integer, Integer> map = new HashMap<>();

   Stack<Integer> st = new Stack<>();
```

```
    for (int n : nums2) {
        while (!st.isEmpty() && st.peek() < n)
            map.put(st.pop(), n);
        st.push(n);
    }

    for (int i = 0; i < nums1.length; i++)
        nums1[i] = map.getOrDefault(nums1[i], -1);

    return nums1;
}
```

---

7. Next Greater Element II (LeetCode 503)

```
Java
int[] nextGreaterElements(int[] nums) {
    int n = nums.length;
    int[] res = new int[n];
    Arrays.fill(res, -1);
    Stack<Integer> st = new Stack<>();

    for (int i = 0; i < 2 * n; i++) {
        int num = nums[i % n];
        while (!st.isEmpty() && nums[st.peek()] < num)
            res[st.pop()] = num;
        if (i < n) st.push(i);
    }
    return res;
}
```

---

8. Circular Queue (LeetCode 622)

```
Java
class MyCircularQueue {
    int[] q;
    int front = 0, rear = -1, size = 0, cap;

    MyCircularQueue(int k) {
        q = new int[k];
        cap = k;
    }

    boolean enQueue(int val) {
```

```
        if (isFull()) return false;
        rear = (rear + 1) % cap;
        q[rear] = val;
        size++;
        return true;
    }

    boolean deQueue() {
        if (isEmpty()) return false;
        front = (front + 1) % cap;
        size--;
        return true;
    }

    int Front() { return isEmpty() ? -1 : q[front]; }

    int Rear() { return isEmpty() ? -1 : q[rear]; }

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

    boolean isFull() { return size == cap; }
}
```

5. Binary Search (6 Questions)

1. **704. Binary Search**
2. **34. Find First and Last Position of Element in Sorted Array**
3. **74. Search a 2D Matrix**
4. **33. Search in Rotated Sorted Array**
5. **81. Search in Rotated Sorted Array II**
6. **162. Find Peak Element**

Solution:

704. Binary Search

```
java
int search(int[] nums, int target) {

   int l = 0, r = nums.length - 1;

   while (l <= r) {

      int mid = l + (r - l) / 2;

      if (nums[mid] == target) return mid;

      else if (nums[mid] < target) l = mid + 1;

      else r = mid - 1;
```

```
    }
    return -1;
}
```

---

34. Find First and Last Position of Element in Sorted Array

```
java
int[] searchRange(int[] nums, int target) {
    return new int[]{first(nums, target), last(nums, target)};
}


int first(int[] a, int t) {
    int l = 0, r = a.length - 1, ans = -1;
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (a[m] >= t) r = m - 1;
        else l = m + 1;
        if (a[m] == t) ans = m;
    }
    return ans;
}


int last(int[] a, int t) {
    int l = 0, r = a.length - 1, ans = -1;
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (a[m] <= t) l = m + 1;
        else r = m - 1;
        if (a[m] == t) ans = m;
    }
    return ans;
}
```

---

74. Search a 2D Matrix

```java
boolean searchMatrix(int[][] mat, int target) {
    int m = mat.length, n = mat[0].length;
    int l = 0, r = m * n - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        int val = mat[mid / n][mid % n];
        if (val == target) return true;
        else if (val < target) l = mid + 1;
        else r = mid - 1;
    }
    return false;
}
```

---

33. Search in Rotated Sorted Array

```java
int search(int[] nums, int target) {
    int l = 0, r = nums.length - 1;
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (nums[m] == target) return m;
        if (nums[l] <= nums[m]) {
            if (target >= nums[l] && target < nums[m]) r = m - 1;
            else l = m + 1;
        } else {
            if (target > nums[m] && target <= nums[r]) l = m + 1;
            else r = m - 1;
        }
    }
```

```
    return -1;
}
```

---

81. Search in Rotated Sorted Array II

```
java
boolean search(int[] nums, int target) {
    int l = 0, r = nums.length - 1;
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (nums[m] == target) return true;

        if (nums[l] == nums[m] && nums[m] == nums[r]) {
            l++; r--;
        } else if (nums[l] <= nums[m]) {
            if (target >= nums[l] && target < nums[m]) r = m - 1;
            else l = m + 1;
        } else {
            if (target > nums[m] && target <= nums[r]) l = m + 1;
            else r = m - 1;
        }
    }
    return false;
}
```

---

162. Find Peak Element

```
Java
int findPeakElement(int[] nums) {
    int l = 0, r = nums.length - 1;
    while (l < r) {
        int m = l + (r - l) / 2;
        if (nums[m] > nums[m + 1]) r = m;
        else l = m + 1;
    }
    return l;
}
```

6. Trees (8 Questions)

1. **104. Maximum Depth of Binary Tree**
2. **100. Same Tree**
3. **101. Symmetric Tree**
4. **144. Binary Tree Preorder Traversal**
5. **94. Binary Tree Inorder Traversal**
6. **145. Binary Tree Postorder Traversal**
7. **102. Binary Tree Level Order Traversal**
8. **110. Balanced Binary Tree**

Solution:

104. Maximum Depth of Binary Tree

```
java
int maxDepth(TreeNode root) {
    if (root == null) return 0;
    return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
}
```

---

100. Same Tree

```
java
boolean isSameTree(TreeNode p, TreeNode q) {
    if (p == null && q == null) return true;
    if (p == null || q == null || p.val != q.val) return false;
    return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
}
```

---

101. Symmetric Tree

```
java
boolean isSymmetric(TreeNode root) {
```

```
    return root == null || isMirror(root.left, root.right);
}

boolean isMirror(TreeNode a, TreeNode b) {
    if (a == null && b == null) return true;
    if (a == null || b == null || a.val != b.val) return false;
    return isMirror(a.left, b.right) && isMirror(a.right, b.left);
}
```

---

144. Binary Tree Preorder Traversal

```
java
void preorder(TreeNode root, List<Integer> res) {
    if (root == null) return;
    res.add(root.val);
    preorder(root.left, res);
    preorder(root.right, res);
}
```

---

94. Binary Tree Inorder Traversal

```
java
void inorder(TreeNode root, List<Integer> res) {
    if (root == null) return;
    inorder(root.left, res);
    res.add(root.val);
    inorder(root.right, res);
}
```

---

145. Binary Tree Postorder Traversal

```
java
void postorder(TreeNode root, List<Integer> res) {
    if (root == null) return;
```

```
    postorder(root.left, res);
    postorder(root.right, res);
    res.add(root.val);
}
```

---

102. Binary Tree Level Order Traversal

```
java
List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> res = new ArrayList<>();
    if (root == null) return res;
    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);
    while (!q.isEmpty()) {
        int size = q.size();
        List<Integer> level = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            TreeNode node = q.poll();
            level.add(node.val);
            if (node.left != null) q.add(node.left);
            if (node.right != null) q.add(node.right);
        }
        res.add(level);
    }
    return res;
}
```

---

110. Balanced Binary Tree

```
Java
boolean isBalanced(TreeNode root) {
    return height(root) != -1;
}

int height(TreeNode node) {
```

```
    if (node == null) return 0;
    int l = height(node.left), r = height(node.right);
    if (l == -1 || r == -1 || Math.abs(l - r) > 1) return -1;
    return 1 + Math.max(l, r);
}
```

7. Recursion and Backtracking (7 Questions)

1. **39. Combination Sum**
2. **46. Permutations**
3. **78. Subsets**
4. **51. N-Queens**
5. **17. Letter Combinations of a Phone Number**
6. **90. Subsets II**
7. **37. Sudoku Solver**

Solution:

39. Combination Sum

```
java
void backtrack(int[] nums, int target, int start, List<Integer> curr, List<List<Integer>> res) {
   if (target == 0) { res.add(new ArrayList<>(curr)); return; }
   if (target < 0) return;

   for (int i = start; i < nums.length; i++) {
      curr.add(nums[i]);
      backtrack(nums, target - nums[i], i, curr, res);
      curr.remove(curr.size() - 1);
   }
}
```

---

46. Permutations

```
java
void backtrack(int[] nums, List<Integer> curr, boolean[] used, List<List<Integer>> res) {
   if (curr.size() == nums.length) { res.add(new ArrayList<>(curr)); return; }

   for (int i = 0; i < nums.length; i++) {
```

```
      if (used[i]) continue;
      used[i] = true;
      curr.add(nums[i]);
      backtrack(nums, curr, used, res);
      curr.remove(curr.size() - 1);
      used[i] = false;
    }
}
```

---

78. Subsets

```
java
void backtrack(int[] nums, int start, List<Integer> curr, List<List<Integer>> res) {
    res.add(new ArrayList<>(curr));
    for (int i = start; i < nums.length; i++) {
      curr.add(nums[i]);
      backtrack(nums, i + 1, curr, res);
      curr.remove(curr.size() - 1);
    }
}
```

---

51. N-Queens

```
Java
void solve(int n, int row, char[][] board, List<List<String>> res,
boolean[] col, boolean[] d1, boolean[] d2) {
    if (row == n) {
        List<String> sol = new ArrayList<>();
        for (char[] r : board) sol.add(new String(r));
        res.add(sol);
        return;
    }

    for (int c = 0; c < n; c++) {
        if (col[c] || d1[row - c + n] || d2[row + c]) continue;
        board[row][c] = 'Q';
        col[c] = d1[row - c + n] = d2[row + c] = true;

        solve(n, row + 1, board, res, col, d1, d2);

        board[row][c] = '.';
```

```
            col[c] = d1[row - c + n] = d2[row + c] = false;
        }
    }
```

---

17. Letter Combinations of a Phone Number

```
java
void backtrack(String digits, int idx, StringBuilder curr, List<String> res, String[] map) {
   if (idx == digits.length()) { res.add(curr.toString()); return; }


   for (char c : map[digits.charAt(idx) - '0'].toCharArray()) {
      curr.append(c);
      backtrack(digits, idx + 1, curr, res, map);
      curr.deleteCharAt(curr.length() - 1);
   }
}
```

---

90. Subsets II

```
java
void backtrack(int[] nums, int start, List<Integer> curr, List<List<Integer>> res) {
   res.add(new ArrayList<>(curr));
   for (int i = start; i < nums.length; i++) {
      if (i > start && nums[i] == nums[i - 1]) continue;
      curr.add(nums[i]);
      backtrack(nums, i + 1, curr, res);
      curr.remove(curr.size() - 1);
   }
}
```

---

37. Sudoku Solver

```
Java
boolean solve(char[][] board) {
    for (int r = 0; r < 9; r++) {
        for (int c = 0; c < 9; c++) {
```

```
            if (board[r][c] == '.') {
                for (char ch = '1'; ch <= '9'; ch++) {
                    if (isValid(board, r, c, ch)) {
                        board[r][c] = ch;
                        if (solve(board)) return true;
                        board[r][c] = '.';
                    }
                }
                return false;
            }
        }
    }
    return true;
}

boolean isValid(char[][] b, int r, int c, char ch) {
    for (int i = 0; i < 9; i++)
        if (b[r][i] == ch || b[i][c] == ch ||
            b[3*(r/3)+i/3][3*(c/3)+i%3] == ch) return false;
    return true;
}
```

8. Dynamic Programming (10 Questions)

1. **70. Climbing Stairs**
2. **198. House Robber**
3. **322. Coin Change**
4. **300. Longest Increasing Subsequence**
5. **1143. Longest Common Subsequence**
6. **62. Unique Paths**
7. **5. Longest Palindromic Substring**
8. **718. Maximum Length of Repeated Subarray**
9. **416. Partition Equal Subset Sum**
10. **53. Maximum Subarray**

Solution:

70. Climbing Stairs

```
java
int climbStairs(int n) {

   if (n <= 2) return n;

   int a = 1, b = 2;

   for (int i = 3; i <= n; i++) {

      int c = a + b;

      a = b;

      b = c;
```

```
    }
    return b;
}
```

---

198. House Robber

```java
int rob(int[] nums) {
    int prev = 0, curr = 0;
    for (int n : nums) {
        int temp = Math.max(curr, prev + n);
        prev = curr;
        curr = temp;
    }
    return curr;
}
```

---

322. Coin Change

```java
int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, amount + 1);
    dp[0] = 0;
    for (int c : coins)
        for (int i = c; i <= amount; i++)
            dp[i] = Math.min(dp[i], dp[i - c] + 1);
    return dp[amount] > amount ? -1 : dp[amount];
}
```

---

300. Longest Increasing Subsequence

```java
int lengthOfLIS(int[] nums) {
```

```
    int[] dp = new int[nums.length];
    Arrays.fill(dp, 1);
    int res = 1;
    for (int i = 1; i < nums.length; i++)
        for (int j = 0; j < i; j++)
            if (nums[i] > nums[j])
                dp[i] = Math.max(dp[i], dp[j] + 1);
    for (int x : dp) res = Math.max(res, x);
    return res;
}
```

1143. Longest Common Subsequence

```
java
int longestCommonSubsequence(String a, String b) {
    int m = a.length(), n = b.length();
    int[][] dp = new int[m + 1][n + 1];
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            dp[i][j] = a.charAt(i - 1) == b.charAt(j - 1)
                    ? dp[i - 1][j - 1] + 1
                    : Math.max(dp[i - 1][j], dp[i][j - 1]);
    return dp[m][n];
}
```

---

62. Unique Paths

```
java
int uniquePaths(int m, int n) {
    int[][] dp = new int[m][n];
    for (int i = 0; i < m; i++) dp[i][0] = 1;
    for (int j = 0; j < n; j++) dp[0][j] = 1;
    for (int i = 1; i < m; i++)
```

```
        for (int j = 1; j < n; j++)
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
    return dp[m - 1][n - 1];
}
```

---

5. Longest Palindromic Substring

```
java
String longestPalindrome(String s) {
    int start = 0, end = 0;
    for (int i = 0; i < s.length(); i++) {
        int len1 = expand(s, i, i), len2 = expand(s, i, i + 1);
        int len = Math.max(len1, len2);
        if (len > end - start) {
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }
    return s.substring(start, end + 1);
}


int expand(String s, int l, int r) {
    while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
        l--; r++;
    }
    return r - l - 1;
}
```

---

718. Maximum Length of Repeated Subarray

```
java
int findLength(int[] A, int[] B) {
    int m = A.length, n = B.length, res = 0;
```

```
    int[][] dp = new int[m + 1][n + 1];
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            if (A[i - 1] == B[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                res = Math.max(res, dp[i][j]);
            }
    return res;
}
```

---

416. Partition Equal Subset Sum

```
java
boolean canPartition(int[] nums) {
    int sum = 0;
    for (int n : nums) sum += n;
    if (sum % 2 != 0) return false;
    int target = sum / 2;
    boolean[] dp = new boolean[target + 1];
    dp[0] = true;
    for (int n : nums)
        for (int j = target; j >= n; j--)
            dp[j] = dp[j] || dp[j - n];
    return dp[target];
}
```

---

53. Maximum Subarray

```
Java
int maxSubArray(int[] nums) {
    int curr = nums[0], max = nums[0];
    for (int i = 1; i < nums.length; i++) {
        curr = Math.max(nums[i], curr + nums[i]);
        max = Math.max(max, curr);
    }
    return max;
```

```
}
```

9. Graphs (6 Questions)

1. **133. Clone Graph**
2. **200. Number of Islands**
3. **207. Course Schedule**
4. **785. Is Graph Bipartite?**
5. **994. Rotting Oranges**
6. **323. Number of Connected Components in an Undirected Graph**

Solution:

133. Clone Graph

```
java
Node cloneGraph(Node node) {
    if (node == null) return null;
    Map<Node, Node> map = new HashMap<>();
    return dfs(node, map);
}

Node dfs(Node node, Map<Node, Node> map) {
    if (map.containsKey(node)) return map.get(node);
    Node copy = new Node(node.val);
    map.put(node, copy);
    for (Node nei : node.neighbors)
        copy.neighbors.add(dfs(nei, map));
    return copy;
}
```

---

200. Number of Islands

```
java
int numIslands(char[][] grid) {
    int count = 0;
```

```
    for (int i = 0; i < grid.length; i++)
        for (int j = 0; j < grid[0].length; j++)
            if (grid[i][j] == '1') {
                dfs(grid, i, j);
                count++;
            }
    return count;
}

void dfs(char[][] g, int i, int j) {
    if (i < 0 || j < 0 || i >= g.length || j >= g[0].length || g[i][j] == '0') return;
    g[i][j] = '0';
    dfs(g, i+1, j); dfs(g, i-1, j); dfs(g, i, j+1); dfs(g, i, j-1);
}
```

---

207. Course Schedule

```
java
boolean canFinish(int num, int[][] pre) {
    List<List<Integer>> g = new ArrayList<>();
    int[] indeg = new int[num];
    for (int i = 0; i < num; i++) g.add(new ArrayList<>());
    for (int[] p : pre) { g.get(p[1]).add(p[0]); indeg[p[0]]++; }

    Queue<Integer> q = new LinkedList<>();
    for (int i = 0; i < num; i++) if (indeg[i] == 0) q.add(i);

    int done = 0;
    while (!q.isEmpty()) {
        int cur = q.poll(); done++;
        for (int nei : g.get(cur))
            if (--indeg[nei] == 0) q.add(nei);
```

```
    }
    return done == num;
}
```

---

785. Is Graph Bipartite?

```
java
boolean isBipartite(int[][] g) {
    int n = g.length;
    int[] color = new int[n];
    for (int i = 0; i < n; i++)
        if (color[i] == 0 && !dfs(g, i, 1, color))
            return false;
    return true;
}

boolean dfs(int[][] g, int i, int c, int[] color) {
    if (color[i] != 0) return color[i] == c;
    color[i] = c;
    for (int nei : g[i])
        if (!dfs(g, nei, -c, color)) return false;
    return true;
}
```

---

994. Rotting Oranges

```
java
int orangesRotting(int[][] g) {
    int m = g.length, n = g[0].length, fresh = 0, time = 0;
    Queue<int[]> q = new LinkedList<>();
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++) {
            if (g[i][j] == 2) q.add(new int[]{i, j});
```

```
        if (g[i][j] == 1) fresh++;
      }

    int[][] dirs = {{1,0},{-1,0},{0,1},{0,-1}};
    while (!q.isEmpty() && fresh > 0) {
      for (int s = q.size(); s > 0; s--) {
        int[] cur = q.poll();
        for (int[] d : dirs) {
          int x = cur[0] + d[0], y = cur[1] + d[1];
          if (x<0||y<0||x>=m||y>=n||g[x][y]!=1) continue;
          g[x][y]=2; fresh--; q.add(new int[]{x,y});
        }
      }
      time++;
    }
    return fresh==0 ? time : -1;
}
```

---

323. Number of Connected Components in an Undirected Graph

```
Java
int countComponents(int n, int[][] edges) {
    List<List<Integer>> g = new ArrayList<>();
    for (int i = 0; i < n; i++) g.add(new ArrayList<>());
    for (int[] e : edges) {
        g.get(e[0]).add(e[1]);
        g.get(e[1]).add(e[0]);
    }
    boolean[] vis = new boolean[n];
    int count = 0;
    for (int i = 0; i < n; i++)
        if (!vis[i]) { dfs(g, i, vis); count++; }
    return count;
}

void dfs(List<List<Integer>> g, int i, boolean[] vis) {
    if (vis[i]) return;
    vis[i] = true;
    for (int nei : g.get(i)) dfs(g, nei, vis);
}
```

10. Bit Manipulation (4 Questions)

1. **136. Single Number**
2. **190. Reverse Bits**
3. **191. Number of 1 Bits**
4. **268. Missing Number**

Solution:

136. Single Number

```
Java
int singleNumber(int[] nums) {
    int res = 0;
    for (int n : nums) res ^= n;
    return res;
}
```

Explanation: XOR operation cancels out numbers that appear twice, leaving only the number that appears once.

---

190. Reverse Bits

```
Java
int reverseBits(int n) {
    int res = 0;
    for (int i = 0; i < 32; i++) {
        res = (res << 1) | (n & 1);
        n >>>= 1;
    }
    return res;
}
```

Explanation: Process each bit of the number, shifting the result left and appending the current bit of `n` to the result.

---

191. Number of 1 Bits

```
Java
int hammingWeight(int n) {
    int count = 0;
```

```
    while (n != 0) {
        count += n & 1;
        n >>>= 1;
    }
    return count;
}
```

Explanation: Count the number of set bits (1s) by repeatedly masking the least significant bit and shifting the number right.

---

268. Missing Number

```
Java
int missingNumber(int[] nums) {
    int res = nums.length;
    for (int i = 0; i < nums.length; i++)
        res ^= i ^ nums[i];
    return res;
}
```