75 DSA Questions from Leet-Code

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:

```
cpp
Copy code
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> map; // Value to index map
    for (int i = 0; i < nums.size(); ++i) {
        int complement = target - nums[i];
        if (map.find(complement) != map.end()) {
            return {map[complement], i};
        }
        map[nums[i]] = i;
    }
    return {};
}
```

---

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

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

```
cpp
Copy code
int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX, maxProfit = 0;
    for (int price : prices) {
        minPrice = min(minPrice, price);
        maxProfit = max(maxProfit, price - minPrice);
    }
    return maxProfit;
}
```

---

3. Merge Sorted Array (LeetCode 88)

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

```
cpp
Copy code
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    int i = m - 1, j = n - 1, k = m + n - 1;
    while (i >= 0 && j >= 0) {
        nums1[k--] = (nums1[i] > nums2[j]) ? nums1[i--] : nums2[j--];
    }
    while (j >= 0) {
        nums1[k--] = nums2[j--];
    }
}
```

---

4. Contains Duplicate (LeetCode 217)

Problem: Check if an array contains duplicates.
Solution:

```
cpp
Copy code
bool containsDuplicate(vector<int>& nums) {
    unordered_set<int> set;
    for (int num : nums) {
        if (set.count(num)) return true;
        set.insert(num);
    }
    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:

```
cpp
Copy code
vector<int> productExceptSelf(vector<int>& nums) {
    int n = nums.size();
    vector<int> res(n, 1);
    int left = 1, right = 1;
    for (int i = 0; i < n; ++i) {
        res[i] *= left;
        left *= nums[i];
        res[n - 1 - i] *= right;
        right *= nums[n - 1 - i];
    }
    return res;
}
```

---

6. Maximum Subarray (LeetCode 53)

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

```
cpp
Copy code
int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0], currentSum = nums[0];
    for (int i = 1; i < nums.size(); ++i) {
        currentSum = max(nums[i], currentSum + nums[i]);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum;
}
```

---

7. 3Sum (LeetCode 15)

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

```
cpp
Copy code
vector<vector<int>> threeSum(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    vector<vector<int>> res;
    for (int i = 0; i < nums.size() - 2; ++i) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;
        int left = i + 1, right = nums.size() - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum == 0) {
                res.push_back({nums[i], nums[left], nums[right]});
                while (left < right && nums[left] == nums[left + 1]) ++left;
                while (left < right && nums[right] == nums[right - 1])
--right;
                ++left; --right;
            } else if (sum < 0) {
                ++left;
            } else {
                --right;
            }
        }
    }
    return res;
}
```

---

8. Merge Intervals (LeetCode 56)

Problem: Merge overlapping intervals.
Solution:

```
cpp
Copy code
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    sort(intervals.begin(), intervals.end());
    vector<vector<int>> merged;
    for (auto& interval : intervals) {
        if (merged.empty() || merged.back()[1] < interval[0]) {
            merged.push_back(interval);
        } else {
            merged.back()[1] = max(merged.back()[1], interval[1]);
        }
    }
    return merged;
}
```

---

9. Container With Most Water (LeetCode 11)

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

```
cpp
Copy code
int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1, maxWater = 0;
    while (left < right) {
        maxWater = max(maxWater, min(height[left], height[right]) * (right -
left));
        if (height[left] < height[right]) ++left;
        else --right;
    }
    return maxWater;
}
```

---

10. Rotate Image (LeetCode 48)

Problem: Rotate a matrix 90 degrees clockwise.
Solution:

```
cpp
Copy code
void rotate(vector<vector<int>>& matrix) {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        for (int j = i; j < n; ++j) {
            swap(matrix[i][j], matrix[j][i]);
        }
    }
    for (auto& row : matrix) {
        reverse(row.begin(), row.end());
    }
}
```

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)

```
cpp
Copy code
bool isValid(string s) {
    stack<char> st;
    unordered_map<char, char> map = {{')', '('}, {'}', '{'}, {']', '['}};
    for (char c : s) {
        if (map.count(c)) {
            if (st.empty() || st.top() != map[c]) return false;
            st.pop();
        } else {
            st.push(c);
        }
    }
    return st.empty();
}
```

---

2. Valid Palindrome (LeetCode 125)

```
cpp
Copy code
bool isPalindrome(string s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        while (left < right && !isalnum(s[left])) ++left;
        while (left < right && !isalnum(s[right])) --right;
        if (tolower(s[left]) != tolower(s[right])) return false;
        ++left; --right;
    }
    return true;
}
```

---

3. Valid Anagram (LeetCode 242)

```
cpp
Copy code
bool isAnagram(string s, string t) {
    if (s.size() != t.size()) return false;
    vector<int> count(26, 0);
    for (char c : s) count[c - 'a']++;
    for (char c : t) {
        if (--count[c - 'a'] < 0) return false;
    }
    return true;
}
```

---

4. Group Anagrams (LeetCode 49)

```
cpp
Copy code
vector<vector<string>> groupAnagrams(vector<string>& strs) {
    unordered_map<string, vector<string>> map;
    for (string s : strs) {
        string sorted = s;
        sort(sorted.begin(), sorted.end());
        map[sorted].push_back(s);
    }
    vector<vector<string>> result;
    for (auto& pair : map) result.push_back(pair.second);
    return result;
}
```

---

5. Longest Palindromic Substring (LeetCode 5)

cpp
Copy code

```
string longestPalindrome(string s) {
    int n = s.size(), start = 0, maxLength = 1;
    vector<vector<bool>> dp(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i) dp[i][i] = true;
    for (int i = n - 1; i >= 0; --i) {
        for (int j = i + 1; j < n; ++j) {
            if (s[i] == s[j] && (j - i == 1 || dp[i + 1][j - 1])) {
                dp[i][j] = true;
                if (j - i + 1 > maxLength) {
                    start = i;
                    maxLength = j - i + 1;
                }
            }
        }
    }
    return s.substr(start, maxLength);
}
```

---

6. Minimum Window Substring (LeetCode 76)

cpp
Copy code

```
string minWindow(string s, string t) {
    unordered_map<char, int> freq;
    for (char c : t) freq[c]++;
    int left = 0, minLen = INT_MAX, count = 0, start = 0;
    for (int right = 0; right < s.size(); ++right) {
        if (--freq[s[right]] >= 0) ++count;
        while (count == t.size()) {
            if (minLen > right - left + 1) {
                minLen = right - left + 1;
                start = left;
            }
            if (++freq[s[left++]] > 0) --count;
        }
    }
    return minLen == INT_MAX ? "" : s.substr(start, minLen);
}
```

---

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

cpp
Copy code

```
int strStr(string haystack, string needle) {
    int m = haystack.size(), n = needle.size();
    for (int i = 0; i <= m - n; ++i) {
        if (haystack.substr(i, n) == needle) return i;
    }
    return -1;
}
```

---

8. String Compression (LeetCode 443)

```
cpp
Copy code
int compress(vector<char>& chars) {
    int index = 0, n = chars.size();
    for (int i = 0; i < n;) {
        char c = chars[i];
        int count = 0;
        while (i < n && chars[i] == c) {
            ++i;
            ++count;
        }
        chars[index++] = c;
        if (count > 1) {
            for (char digit : to_string(count)) chars[index++] = digit;
        }
    }
    return index;
}
```

---

9. Longest Common Prefix (LeetCode 14)

```
cpp
Copy code
string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty()) return "";
    string prefix = strs[0];
    for (int i = 1; i < strs.size(); ++i) {
        while (strs[i].find(prefix) != 0) {
            prefix = prefix.substr(0, prefix.size() - 1);
            if (prefix.empty()) return "";
        }
    }
    return prefix;
}
```

---

10. Repeated Substring Pattern (LeetCode 459)

```
cpp
Copy code
bool repeatedSubstringPattern(string s) {
    string doubled = s + s;
    return doubled.substr(1, doubled.size() - 2).find(s) != string::npos;
}
```

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)

```
cpp
Copy code
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    while (head) {
        ListNode* nextNode = head->next;
        head->next = prev;
        prev = head;
        head = nextNode;
    }
    return prev;
}
```

---

2. Merge Two Sorted Lists (LeetCode 21)

```
cpp
Copy code
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if (!list1) return list2;
    if (!list2) return list1;
    if (list1->val < list2->val) {
        list1->next = mergeTwoLists(list1->next, list2);
        return list1;
    } else {
        list2->next = mergeTwoLists(list1, list2->next);
        return list2;
    }
}
```

---

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

```
cpp
Copy code
ListNode* removeNthFromEnd(ListNode* head, int n) {
    ListNode dummy(0, head);
    ListNode* slow = &dummy, *fast = &dummy;
    for (int i = 0; i <= n; ++i) fast = fast->next;
    while (fast) {
        slow = slow->next;
        fast = fast->next;
    }
    slow->next = slow->next->next;
    return dummy.next;
}
```

---

4. Linked List Cycle (LeetCode 141)

```
cpp
Copy code
bool hasCycle(ListNode* head) {
    ListNode* slow = head, *fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
```

---

5. Add Two Numbers (LeetCode 2)

```
cpp
Copy code
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode dummy, *current = &dummy;
    int carry = 0;
    while (l1 || l2 || carry) {
        int sum = carry;
        if (l1) sum += l1->val, l1 = l1->next;
        if (l2) sum += l2->val, l2 = l2->next;
        carry = sum / 10;
        current->next = new ListNode(sum % 10);
        current = current->next;
    }
    return dummy.next;
}
```

---

6. Intersection of Two Linked Lists (LeetCode 160)

```
cpp
Copy code
ListNode* getIntersectionNode(ListNode* headA, ListNode* headB) {
    ListNode* a = headA, *b = headB;
    while (a != b) {
        a = a ? a->next : headB;
        b = b ? b->next : headA;
    }
    return a;
}
```

---

7. Palindrome Linked List (LeetCode 234)

```
cpp
Copy code
bool isPalindrome(ListNode* head) {
    ListNode* slow = head, *fast = head, *prev = nullptr;
    while (fast && fast->next) {
        fast = fast->next->next;
        ListNode* temp = slow->next;
        slow->next = prev;
        prev = slow;
        slow = temp;
    }
    if (fast) slow = slow->next;
    while (slow) {
        if (slow->val != prev->val) return false;
        slow = slow->next;
        prev = prev->next;
    }
    return true;
}
```

---

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

```
cpp
Copy code
ListNode* reverseKGroup(ListNode* head, int k) {
    ListNode* temp = head;
    for (int i = 0; i < k; ++i) {
        if (!temp) return head;
        temp = temp->next;
    }
    ListNode* prev = nullptr, *current = head, *next = nullptr;
    for (int i = 0; i < k; ++i) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head->next = reverseKGroup(next, k);
    return prev;
}
```

4. Stacks and Queues (6 Questions)

1. **232. Implement Queue using Stacks**
2. **225. Implement Stack using Queues**
3. **739. Daily Temperatures**
4. **496. Next Greater Element I**
5. **20. Valid Parentheses**
6. **155. Min Stack**

Solution:

Stacks and Queues

---

1. Implement Queue Using Stacks (LeetCode 232)

```
cpp
Copy code
class MyQueue {
    stack<int> s1, s2;
public:
    void push(int x) {
        s1.push(x);
    }
    int pop() {
        peek();
        int top = s2.top();
        s2.pop();
        return top;
    }
    int peek() {
        if (s2.empty()) {
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        return s2.top();
    }
    bool empty() {
        return s1.empty() && s2.empty();
    }
};
```

---

2. Implement Stack Using Queues (LeetCode 225)

```
cpp
Copy code
class MyStack {
    queue<int> q1, q2;
public:
    void push(int x) {
        q2.push(x);
        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }
        swap(q1, q2);
    }
    int pop() {
        int top = q1.front();
        q1.pop();
        return top;
    }
    int top() {
        return q1.front();
    }
    bool empty() {
        return q1.empty();
    }
};
```

---

3. Min Stack (LeetCode 155)

cpp
Copy code

```
class MinStack {
    stack<int> s, minS;
public:
    void push(int val) {
        s.push(val);
        if (minS.empty() || val <= minS.top()) minS.push(val);
    }
    void pop() {
        if (s.top() == minS.top()) minS.pop();
        s.pop();
    }
    int top() {
        return s.top();
    }
    int getMin() {
        return minS.top();
    }
};
```

---

4. Daily Temperatures (LeetCode 739)

cpp
Copy code

```
vector<int> dailyTemperatures(vector<int>& temperatures) {
    stack<int> s;
    vector<int> res(temperatures.size(), 0);
    for (int i = 0; i < temperatures.size(); ++i) {
        while (!s.empty() && temperatures[i] > temperatures[s.top()]) {
            int index = s.top();
            s.pop();
            res[index] = i - index;
        }
        s.push(i);
    }
    return res;
}
```

---

5. Evaluate Reverse Polish Notation (LeetCode 150)

cpp
Copy code

```
int evalRPN(vector<string>& tokens) {
    stack<int> s;
    for (string& token : tokens) {
        if (token == "+" || token == "-" || token == "*" || token == "/") {
            int b = s.top(); s.pop();
            int a = s.top(); s.pop();
            if (token == "+") s.push(a + b);
            else if (token == "-") s.push(a - b);
            else if (token == "*") s.push(a * b);
            else s.push(a / b);
        } else {
            s.push(stoi(token));
        }
    }
    return s.top();
}
```

6. Next Greater Element I (LeetCode 496)

```
cpp
Copy code
vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {
    unordered_map<int, int> map;
    stack<int> s;
    for (int num : nums2) {
        while (!s.empty() && s.top() < num) {
            map[s.top()] = num;
            s.pop();
        }
        s.push(num);
    }
    vector<int> res;
    for (int num : nums1) {
        res.push_back(map.count(num) ? map[num] : -1);
    }
    return res;
}
```

7. Next Greater Element II (LeetCode 503)

```
cpp
Copy code
vector<int> nextGreaterElements(vector<int>& nums) {
    int n = nums.size();
    vector<int> res(n, -1);
    stack<int> s;
    for (int i = 0; i < 2 * n; ++i) {
        while (!s.empty() && nums[s.top()] < nums[i % n]) {
            res[s.top()] = nums[i % n];
            s.pop();
        }
        if (i < n) s.push(i);
    }
    return res;
}
```

8. Circular Queue (LeetCode 622)

```
cpp
Copy code
class MyCircularQueue {
    vector<int> q;
    int head, tail, size;
public:
    MyCircularQueue(int k) : q(k), head(-1), tail(-1), size(k) {}
    bool enQueue(int value) {
        if (isFull()) return false;
        if (isEmpty()) head = 0;
        tail = (tail + 1) % size;
        q[tail] = value;
        return true;
    }
    bool deQueue() {
        if (isEmpty()) return false;
        if (head == tail) head = tail = -1;
        else head = (head + 1) % size;
        return true;
    }
    int Front() {
        return isEmpty() ? -1 : q[head];
    }
    int Rear() {
        return isEmpty() ? -1 : q[tail];
    }
    bool isEmpty() {
        return head == -1;
    }
    bool isFull() {
        return (tail + 1) % size == head;
    }
};
```

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

```
cpp
Copy code
int search(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) return mid;
        if (nums[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

---

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

```
cpp
Copy code
vector<int> searchRange(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    vector<int> result(2, -1);

    // Find the first position
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] < target) left = mid + 1;
        else right = mid - 1;
        if (nums[mid] == target) result[0] = mid;
    }

    left = 0, right = nums.size() - 1;
    // Find the last position
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] <= target) left = mid + 1;
        else right = mid - 1;
        if (nums[mid] == target) result[1] = mid;
    }

    return result;
}
```

---

74. Search a 2D Matrix

```
cpp
Copy code
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int rows = matrix.size(), cols = matrix[0].size();
    int left = 0, right = rows * cols - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        int midElement = matrix[mid / cols][mid % cols];
        if (midElement == target) return true;
        if (midElement < target) left = mid + 1;
        else right = mid - 1;
    }
    return false;
}
```

---

33. Search in Rotated Sorted Array

```
cpp
Copy code
int search(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) return mid;

        // Check which side is sorted
        if (nums[left] <= nums[mid]) { // Left side is sorted
            if (nums[left] <= target && target < nums[mid]) right = mid - 1;
            else left = mid + 1;
        } else { // Right side is sorted
            if (nums[mid] < target && target <= nums[right]) left = mid + 1;
            else right = mid - 1;
        }
    }
    return -1;
}
```

---

81. Search in Rotated Sorted Array II

cpp
Copy code

```
bool search(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) return true;

        // Handle duplicates
        if (nums[left] == nums[mid] && nums[right] == nums[mid]) {
            left++;
            right--;
        } else if (nums[left] <= nums[mid]) { // Left side is sorted
            if (nums[left] <= target && target < nums[mid]) right = mid - 1;
            else left = mid + 1;
        } else { // Right side is sorted
            if (nums[mid] < target && target <= nums[right]) left = mid + 1;
            else right = mid - 1;
        }
    }
    return false;
}
```

---

162. Find Peak Element

cpp
Copy code

```
int findPeakElement(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] > nums[mid + 1]) right = mid;
        else left = mid + 1;
    }
    return left;
}
```

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

```
cpp
Copy code
int maxDepth(TreeNode* root) {
    if (!root) return 0;
    return 1 + max(maxDepth(root->left), maxDepth(root->right));
}
```

---

100. Same Tree

```
cpp
Copy code
bool isSameTree(TreeNode* p, TreeNode* q) {
    if (!p && !q) return true;
    if (!p || !q || p->val != q->val) return false;
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}
```

---

101. Symmetric Tree

```
cpp
Copy code
bool isMirror(TreeNode* t1, TreeNode* t2) {
    if (!t1 && !t2) return true;
    if (!t1 || !t2 || t1->val != t2->val) return false;
    return isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
}

bool isSymmetric(TreeNode* root) {
    return isMirror(root, root);
}
```

---

144. Binary Tree Preorder Traversal

```
cpp
Copy code
vector<int> preorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> st;
    if (root) st.push(root);
    while (!st.empty()) {
        TreeNode* curr = st.top();
        st.pop();
        result.push_back(curr->val);
        if (curr->right) st.push(curr->right);
        if (curr->left) st.push(curr->left);
    }
    return result;
}
```

---

94. Binary Tree Inorder Traversal

cpp
Copy code

```
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> st;
    TreeNode* curr = root;
    while (curr || !st.empty()) {
        while (curr) {
            st.push(curr);
            curr = curr->left;
        }
        curr = st.top();
        st.pop();
        result.push_back(curr->val);
        curr = curr->right;
    }
    return result;
}
```

---

145. Binary Tree Postorder Traversal

cpp
Copy code

```
vector<int> postorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> st;
    TreeNode* lastVisited = nullptr;
    while (root || !st.empty()) {
        if (root) {
            st.push(root);
            root = root->left;
        } else {
            TreeNode* node = st.top();
            if (node->right && node->right != lastVisited) {
                root = node->right;
            } else {
                result.push_back(node->val);
                lastVisited = node;
                st.pop();
            }
        }
    }
    return result;
}
```

---

102. Binary Tree Level Order Traversal

cpp
Copy code

```
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> result;
    if (!root) return result;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        int size = q.size();
        vector<int> level;
        for (int i = 0; i < size; ++i) {
            TreeNode* curr = q.front();
            q.pop();
            level.push_back(curr->val);
            if (curr->left) q.push(curr->left);
            if (curr->right) q.push(curr->right);
        }
        result.push_back(level);
    }
    return result;
}
```

---

110. Balanced Binary Tree

cpp
Copy code

```
int height(TreeNode* root) {
    if (!root) return 0;
    int leftHeight = height(root->left);
    int rightHeight = height(root->right);
    if (leftHeight == -1 || rightHeight == -1 || abs(leftHeight -
rightHeight) > 1)
        return -1;
    return 1 + max(leftHeight, rightHeight);
}

bool isBalanced(TreeNode* root) {
    return height(root) != -1;
}
```

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

cpp
Copy code

```
void backtrack(vector<int>& candidates, int target, int start, vector<int>&
current, vector<vector<int>>& result) {
    if (target == 0) {
        result.push_back(current);
        return;
    }
    for (int i = start; i < candidates.size(); i++) {
        if (candidates[i] > target) continue;
        current.push_back(candidates[i]);
        backtrack(candidates, target - candidates[i], i, current, result);
        current.pop_back();
    }
}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;
    backtrack(candidates, target, 0, current, result);
    return result;
}
```

---

46. Permutations

```
cpp
Copy code
void permuteHelper(vector<int>& nums, int start, vector<vector<int>>& result)
{
    if (start == nums.size()) {
        result.push_back(nums);
        return;
    }
    for (int i = start; i < nums.size(); i++) {
        swap(nums[start], nums[i]);
        permuteHelper(nums, start + 1, result);
        swap(nums[start], nums[i]);
    }
}

vector<vector<int>> permute(vector<int>& nums) {
    vector<vector<int>> result;
    permuteHelper(nums, 0, result);
    return result;
}
```

---

78. Subsets

```
cpp
Copy code
void backtrack(vector<int>& nums, int start, vector<int>& current,
vector<vector<int>>& result) {
    result.push_back(current);
    for (int i = start; i < nums.size(); i++) {
        current.push_back(nums[i]);
        backtrack(nums, i + 1, current, result);
        current.pop_back();
    }
}

vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> result;
    vector<int> current;
    backtrack(nums, 0, current, result);
    return result;
}
```

---

51. N-Queens

```
cpp
```

```
Copy code
bool isSafe(vector<string>& board, int row, int col, int n) {
    for (int i = 0; i < row; i++)
        if (board[i][col] == 'Q') return false;
    for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)
        if (board[i][j] == 'Q') return false;
    for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)
        if (board[i][j] == 'Q') return false;
    return true;
}

void solve(int row, vector<string>& board, vector<vector<string>>& result,
int n) {
    if (row == n) {
        result.push_back(board);
        return;
    }
    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col, n)) {
            board[row][col] = 'Q';
            solve(row + 1, board, result, n);
            board[row][col] = '.';
        }
    }
}

vector<vector<string>> solveNQueens(int n) {
    vector<vector<string>> result;
    vector<string> board(n, string(n, '.'));
    solve(0, board, result, n);
    return result;
}
```

---

17. Letter Combinations of a Phone Number

```
cpp
Copy code
void backtrack(string& digits, int index, string& current, vector<string>&
result, vector<string>& mapping) {
    if (index == digits.size()) {
        result.push_back(current);
        return;
    }
    for (char ch : mapping[digits[index] - '0']) {
        current.push_back(ch);
        backtrack(digits, index + 1, current, result, mapping);
        current.pop_back();
    }
}

vector<string> letterCombinations(string digits) {
    if (digits.empty()) return {};
    vector<string> mapping = {"", "", "abc", "def", "ghi", "jkl", "mno",
"pqrs", "tuv", "wxyz"};
    vector<string> result;
    string current;
    backtrack(digits, 0, current, result, mapping);
    return result;
}
```

---

90. Subsets II

```
cpp
Copy code
void backtrack(vector<int>& nums, int start, vector<int>& current, 
vector<vector<int>>& result) {
    result.push_back(current);
    for (int i = start; i < nums.size(); i++) {
        if (i > start && nums[i] == nums[i - 1]) continue;
        current.push_back(nums[i]);
        backtrack(nums, i + 1, current, result);
        current.pop_back();
    }
}

vector<vector<int>> subsetsWithDup(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    vector<vector<int>> result;
    vector<int> current;
    backtrack(nums, 0, current, result);
    return result;
}
```

---

37. Sudoku Solver

```
cpp
Copy code
bool isValid(vector<vector<char>>& board, int row, int col, char c) {
    for (int i = 0; i < 9; i++) {
        if (board[i][col] == c || board[row][i] == c || board[row / 3 * 3 + i 
/ 3][col / 3 * 3 + i % 3] == c)
            return false;
    }
    return true;
}

bool solve(vector<vector<char>>& board) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (board[i][j] == '.') {
                for (char c = '1'; c <= '9'; c++) {
                    if (isValid(board, i, j, c)) {
                        board[i][j] = c;
                        if (solve(board)) return true;
                        board[i][j] = '.';
                    }
                }
                return false;
            }
        }
    }
    return true;
}

void solveSudoku(vector<vector<char>>& board) {
    solve(board);
}
```

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

```
cpp
Copy code
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

```
cpp
Copy code
int rob(vector<int>& nums) {
    int prev1 = 0, prev2 = 0;
    for (int num : nums) {
        int temp = max(prev1, prev2 + num);
        prev2 = prev1;
        prev1 = temp;
    }
    return prev1;
}
```

---

322. Coin Change

```
cpp
Copy code
int coinChange(vector<int>& coins, int amount) {
    vector<int> dp(amount + 1, amount + 1);
    dp[0] = 0;
    for (int coin : coins) {
        for (int j = coin; j <= amount; j++) {
            dp[j] = min(dp[j], dp[j - coin] + 1);
        }
    }
    return dp[amount] > amount ? -1 : dp[amount];
}
```

---

300. Longest Increasing Subsequence

cpp

Copy code

```
int lengthOfLIS(vector<int>& nums) {
    vector<int> dp(nums.size(), 1);
    int maxLength = 1;
    for (int i = 1; i < nums.size(); i++) {
        for (int j = 0; j < i; j++) {
            if (nums[i] > nums[j]) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        maxLength = max(maxLength, dp[i]);
    }
    return maxLength;
}
```

1143. Longest Common Subsequence

cpp

Copy code

```
int longestCommonSubsequence(string text1, string text2) {
    int m = text1.size(), n = text2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (text1[i - 1] == text2[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    return dp[m][n];
}
```

---

62. Unique Paths

cpp

Copy code

```
int uniquePaths(int m, int n) {
    vector<vector<int>> dp(m, vector<int>(n, 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

cpp

Copy code

```
string longestPalindrome(string s) {
    int n = s.size();
    vector<vector<bool>> dp(n, vector<bool>(n, false));
    int maxLength = 1, start = 0;

    for (int i = 0; i < n; i++) dp[i][i] = true;

    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            if (s[i] == s[j] && (len == 2 || dp[i + 1][j - 1])) {
                dp[i][j] = true;
                if (len > maxLength) {
                    maxLength = len;
                    start = i;
                }
            }
        }
    }
    return s.substr(start, maxLength);
}
```

718. Maximum Length of Repeated Subarray

```
cpp
Copy code
int findLength(vector<int>& nums1, vector<int>& nums2) {
    int m = nums1.size(), n = nums2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    int maxLength = 0;
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (nums1[i - 1] == nums2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                maxLength = max(maxLength, dp[i][j]);
            }
        }
    }
    return maxLength;
}
```

416. Partition Equal Subset Sum

```
cpp
Copy code
bool canPartition(vector<int>& nums) {
    int sum = accumulate(nums.begin(), nums.end(), 0);
    if (sum % 2 != 0) return false;
    int target = sum / 2;
    vector<bool> dp(target + 1, false);
    dp[0] = true;

    for (int num : nums) {
        for (int j = target; j >= num; j--) {
            dp[j] = dp[j] || dp[j - num];
        }
    }
    return dp[target];
}
```

53. Maximum Subarray

```
cpp
Copy code
int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0], currentSum = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        currentSum = max(nums[i], currentSum + nums[i]);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum;
}
```

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

```
cpp
Copy code
Node* cloneGraph(Node* node) {
    if (!node) return nullptr;

    unordered_map<Node*, Node*> visited;
    queue<Node*> q;
    q.push(node);

    visited[node] = new Node(node->val);

    while (!q.empty()) {
        Node* curr = q.front();
        q.pop();

        for (Node* neighbor : curr->neighbors) {
            if (!visited.count(neighbor)) {
                visited[neighbor] = new Node(neighbor->val);
                q.push(neighbor);
            }
            visited[curr]->neighbors.push_back(visited[neighbor]);
        }
    }
    return visited[node];
}
```

200. Number of Islands

cpp
Copy code

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

int numIslands(vector<vector<char>>& grid) {
    int count = 0;
    for (int i = 0; i < grid.size(); i++) {
        for (int j = 0; j < grid[0].size(); j++) {
            if (grid[i][j] == '1') {
                count++;
                dfs(grid, i, j);
            }
        }
    }
    return count;
}
```

---

207. Course Schedule

cpp
Copy code

```
bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    vector<vector<int>> graph(numCourses);
    vector<int> inDegree(numCourses, 0);

    for (auto& pre : prerequisites) {
        graph[pre[1]].push_back(pre[0]);
        inDegree[pre[0]]++;
    }

    queue<int> q;
    for (int i = 0; i < numCourses; i++) {
        if (inDegree[i] == 0) q.push(i);
    }

    int count = 0;
    while (!q.empty()) {
        int course = q.front();
        q.pop();
        count++;
        for (int neighbor : graph[course]) {
            if (--inDegree[neighbor] == 0) q.push(neighbor);
        }
    }
    return count == numCourses;
}
```

---

785. Is Graph Bipartite?

cpp
Copy code

```
bool isBipartite(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> color(n, -1);

    for (int i = 0; i < n; i++) {
        if (color[i] != -1) continue;

        queue<int> q;
        q.push(i);
        color[i] = 0;

        while (!q.empty()) {
            int node = q.front();
            q.pop();
            for (int neighbor : graph[node]) {
                if (color[neighbor] == -1) {
                    color[neighbor] = 1 - color[node];
                    q.push(neighbor);
                } else if (color[neighbor] == color[node]) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

---

994. Rotting Oranges

cpp
Copy code

```
int orangesRotting(vector<vector<int>>& grid) {
    int m = grid.size(), n = grid[0].size();
    queue<pair<int, int>> q;
    int fresh = 0;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (grid[i][j] == 2) q.push({i, j});
            if (grid[i][j] == 1) fresh++;
        }
    }

    int minutes = 0;
    vector<pair<int, int>> directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

    while (!q.empty() && fresh > 0) {
        int size = q.size();
        for (int i = 0; i < size; i++) {
            auto [x, y] = q.front();
            q.pop();
            for (auto [dx, dy] : directions) {
                int nx = x + dx, ny = y + dy;
                if (nx >= 0 && ny >= 0 && nx < m && ny < n && grid[nx][ny] ==
1) {
                    grid[nx][ny] = 2;
                    q.push({nx, ny});
                    fresh--;
                }
            }
        }
        minutes++;
    }
    return fresh == 0 ? minutes : -1;
}
```

323. Number of Connected Components in an Undirected Graph

```
cpp
Copy code
void dfs(vector<vector<int>>& graph, vector<bool>& visited, int node) {
    visited[node] = true;
    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            dfs(graph, visited, neighbor);
        }
    }
}

int countComponents(int n, vector<vector<int>>& edges) {
    vector<vector<int>> graph(n);
    for (auto& edge : edges) {
        graph[edge[0]].push_back(edge[1]);
        graph[edge[1]].push_back(edge[0]);
    }

    vector<bool> visited(n, false);
    int count = 0;

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            count++;
            dfs(graph, visited, i);
        }
    }
    return count;
}
```

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

cpp
Copy code

```cpp
int singleNumber(vector<int>& nums) {
    int result = 0;
    for (int num : nums) {
        result ^= num;
    }
    return result;
}
```

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

---

190. Reverse Bits

cpp
Copy code

```cpp
uint32_t reverseBits(uint32_t n) {
    uint32_t result = 0;
    for (int i = 0; i < 32; i++) {
        result = (result << 1) | (n & 1);
        n >>= 1;
    }
    return result;
}
```

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

cpp
Copy code

```cpp
int hammingWeight(uint32_t 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

cpp
Copy code

```cpp
int missingNumber(vector<int>& nums) {
    int n = nums.size();
    int totalXOR = 0, arrayXOR = 0;
    for (int i = 0; i <= n; i++) {
        totalXOR ^= i;
    }
    for (int num : nums) {
        arrayXOR ^= num;
    }
    return totalXOR ^ arrayXOR;
}
```