# 50 Essential DSA Problems with Java & C++ Code

### 1. Two Sum

**Problem:** Given an array of integers, return indices of the two numbers such that they add up to a specific target.

**Java**

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement))
            return new int[]{map.get(complement), i};
        map.put(nums[i], i);
    }
    return new int[]{};
}
```

**C++**

```
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> mp;
    for(int i=0;i<nums.size();i++) {
        int diff = target-nums[i];
        if(mp.count(diff))
            return {mp[diff], i};
        mp[nums[i]] = i;
    }
    return {};
}
```

### 2. Maximum Subarray (Kadane's algorithm)

**Java**

```
public int maxSubArray(int[] nums) {
    int max = nums[0], sum = nums[0];
```

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

**C++**

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

## 3. Move Zeroes to End

**Java**

```
public void moveZeroes(int[] nums) {
    int pos = 0;
    for (int i = 0; i < nums.length; i++)
        if (nums[i] != 0)
            nums[pos++] = nums[i];
    while (pos < nums.length)
        nums[pos++] = 0;
}
```

**C++**

```
void moveZeroes(vector<int>& nums) {
    int pos = 0;
    for (int num : nums)
        if (num != 0)
            nums[pos++] = num;
    while (pos < nums.size())
```

```
        nums[pos++] = 0;
}
```

### 4. Find Duplicate Number

**Java**

```
public int findDuplicate(int[] nums) {
    int slow = nums[0], fast = nums[0];
    do {
        slow = nums[slow];
        fast = nums[nums[fast]];
    } while (slow != fast);
    slow = nums[0];
    while (slow != fast) {
        slow = nums[slow];
        fast = nums[fast];
    }
    return slow;
}
```

**C++**

```
int findDuplicate(vector<int>& nums) {
    int slow = nums[0], fast = nums[0];
    do {
        slow = nums[slow];
        fast = nums[nums[fast]];
    } while (slow != fast);
    slow = nums[0];
    while (slow != fast) {
        slow = nums[slow];
        fast = nums[fast];
    }
    return slow;
}
```

### 5. Longest Substring Without Repeating Characters

**Java**

```
public int lengthOfLongestSubstring(String s) {
    Set<Character> set = new HashSet<>();
    int left = 0, max = 0;
    for (int right = 0; right < s.length(); right++) {
        while (!set.add(s.charAt(right)))
            set.remove(s.charAt(left++));
        max = Math.max(max, right - left + 1);
    }
    return max;
}
```

**C++**

```
int lengthOfLongestSubstring(string s) {
    unordered_set<char> set;
    int left=0, right=0, maxlen=0;
    while(right < s.length()) {
        while(set.count(s[right])) set.erase(s[left++]);
        set.insert(s[right++]);
        maxlen = max(maxlen, right-left);
    }
    return maxlen;
}
```

## 6. Check Anagram

**Java**

```
public boolean isAnagram(String s, String t) {
    int[] count = new int[26];
    for (char c : s.toCharArray()) count[c-'a']++;
    for (char c : t.toCharArray()) count[c-'a']--;
    for (int i : count) if (i != 0) return false;
    return true;
}
```

**C++**

```
bool isAnagram(string s, string t) {
    int count[26] = {};
    for(char c : s) count[c-'a']++;
    for(char c : t) count[c-'a']--;
    for(int n : count) if(n != 0) return false;
    return true;
}
```

### 7. Reverse a Linked List

**Java**

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

**C++**

```
ListNode* reverseList(ListNode* head) {
    ListNode *prev = nullptr, *curr = head;
    while (curr) {
        ListNode* next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

### 8. Merge Two Sorted Lists

**Java**

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0);
    ListNode current = dummy;
    while(l1 != null && l2 != null) {
        if(l1.val < l2.val) {
            current.next = l1;
            l1 = l1.next;
        } else {
            current.next = l2;
            l2 = l2.next;
        }
        current = current.next;
    }
    current.next = l1 != null ? l1 : l2;
    return dummy.next;
}
```

**C++**

```
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    ListNode dummy(0), *tail = &dummy;
    while(l1 && l2){
        if(l1->val < l2->val)
            tail->next = l1, l1 = l1->next;
        else
            tail->next = l2, l2 = l2->next;
        tail = tail->next;
    }
    tail->next = l1 ? l1 : l2;
    return dummy.next;
}
```

## 9. Detect Cycle in Linked List

**Java**

```
public boolean hasCycle(ListNode head) {
    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;
}
```

**C++**

```
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;
}
```

## 10. Remove Nth Node From End

**Java**

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    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;
}
```

**C++**

```
ListNode* removeNthFromEnd(ListNode* head, int n) {
    ListNode dummy(0); dummy.next = head;
```

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

## 11. Valid Parentheses

### Java

```
public boolean isValid(String s) {
    Stack<Character> stack = new Stack<>();
    for(char c: s.toCharArray()){
        if(c == '(' || c == '{' || c == '[') stack.push(c);
        else{
            if(stack.isEmpty()) return false;
            char open = stack.pop();
            if( (c == ')' && open != '(') ||
                (c == ']' && open != '[') ||
                (c == '}' && open != '{'))
                return false;
        }
    }
    return stack.isEmpty();
}
```

### C++

```
bool isValid(string s) {
    stack<char> st;
    for(char c : s){
        if(c=='(' || c=='[' || c=='{') st.push(c);
        else{
            if(st.empty()) return false;
            char open = st.top(); st.pop();
            if((c==')'&&open!='(')||(c==']'&&open!='[')||(c=='}'&&open!='{')) return
false;
        }
```

```
    }
    return st.empty();
}
```

## 12. Implement Stack Using Queues

### Java

```
class MyStack {
    Queue<Integer> q = new LinkedList<>();
    public void push(int x) {
        q.add(x);
        for(int i=0; i<q.size()-1; i++)
            q.add(q.remove());
    }
    public int pop() { return q.remove(); }
    public int top() { return q.peek(); }
    public boolean empty() { return q.isEmpty(); }
}
```

### C++

```
class MyStack {
    queue<int> q;
public:
    void push(int x) {
        q.push(x);
        for(int i=0; i<q.size()-1; i++)
            q.push(q.front()), q.pop();
    }
    int pop() { int t=q.front(); q.pop(); return t; }
    int top() { return q.front(); }
    bool empty() { return q.empty(); }
};
```

## 13. Next Greater Element

### Java

```
public int[] nextGreaterElements(int[] nums) {
    int[] res = new int[nums.length];
    Stack<Integer> stack = new Stack<>();
    for(int i = nums.length - 1; i >= 0; --i){
        while (!stack.isEmpty() && stack.peek() <= nums[i]) stack.pop();
        res[i] = stack.isEmpty() ? -1 : stack.peek();
        stack.push(nums[i]);
    }
    return res;
}
```

**C++**

```
vector<int> nextGreaterElements(vector<int>& nums) {
    vector<int> res(nums.size(), -1);
    stack<int> st;
    for(int i = nums.size()-1; i >= 0; i--){
        while(!st.empty() && st.top() <= nums[i]) st.pop();
        if(!st.empty()) res[i] = st.top();
        st.push(nums[i]);
    }
    return res;
}
```

## 14. Maximum Depth of Binary Tree

### Java

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

**C++**

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

```
}
```

## 15. Invert Binary Tree

### Java

```
public TreeNode invertTree(TreeNode root) {
    if(root == null) return null;
    TreeNode left = invertTree(root.left);
    root.left = invertTree(root.right);
    root.right = left;
    return root;
}
```

### C++

```
TreeNode* invertTree(TreeNode* root) {
    if(!root) return nullptr;
    swap(root->left, root->right);
    invertTree(root->left);
    invertTree(root->right);
    return root;
}
```

## 16. Serialize and Deserialize Binary Tree

### Java (LeetCode Style)

```
public class Codec {
    public String serialize(TreeNode root) {
        if(root == null) return "#";
        return root.val + "," + serialize(root.left) + "," + serialize(root.right);
    }
    public TreeNode deserialize(String data) {
        Queue<String> queue = new LinkedList<>(Arrays.asList(data.split(",")));
        return helper(queue);
    }
    private TreeNode helper(Queue<String> queue) {
        String val = queue.poll();
```

```
        if(val.equals("#")) return null;
        TreeNode node = new TreeNode(Integer.parseInt(val));
        node.left = helper(queue);
        node.right = helper(queue);
        return node;
    }
}
```

**C++**

```
string serialize(TreeNode* root) {
    if(!root) return "#,";
    return to_string(root->val) + "," + serialize(root->left) + serialize(root->right);
}
TreeNode* deserializeUtil(queue<string>& q) {
    string val = q.front(); q.pop();
    if(val == "#") return nullptr;
    TreeNode* node = new TreeNode(stoi(val));
    node->left = deserializeUtil(q);
    node->right = deserializeUtil(q);
    return node;
}
TreeNode* deserialize(string data) {
    queue<string> q;
    string val;
    istringstream ss(data);
    while(getline(ss, val, ','))
        q.push(val);
    return deserializeUtil(q);
}
```

## 17. Lowest Common Ancestor in BST

### Java

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if (root.val > p.val && root.val > q.val)
        return lowestCommonAncestor(root.left, p, q);
    if (root.val < p.val && root.val < q.val)
        return lowestCommonAncestor(root.right, p, q);
    return root;
}
```

### C++

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    if (root->val > p->val && root->val > q->val)
        return lowestCommonAncestor(root->left, p, q);
    if (root->val < p->val && root->val < q->val)
        return lowestCommonAncestor(root->right, p, q);
    return root;
}
```

## 18. Symmetric Tree

### Java

```
public boolean isSymmetric(TreeNode root) {
    return root == null || isMirror(root.left, root.right);
}

private boolean isMirror(TreeNode t1, TreeNode t2) {
    if (t1 == null || t2 == null) return t1 == t2;
    return t1.val == t2.val && isMirror(t1.left, t2.right) && isMirror(t1.right,
t2.left);
}
```

### C++

```
bool isSymmetric(TreeNode* root) {
    return root == nullptr || isMirror(root->left, root->right);
}
bool isMirror(TreeNode* t1, TreeNode* t2) {
    if (!t1 || !t2) return t1 == t2;
    return t1->val == t2->val && isMirror(t1->left, t2->right) && isMirror(t1->right,
t2->left);
}
```

## 19. Number of Islands

### Java

```
public 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[][] grid, int i, int j) {
    if (i < 0 || i >= grid.length || j < 0 || j >= grid[0].length || grid[i][j] != '1')
        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);
}
```

### C++

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

## 20. Detect Cycle in Directed Graph (DFS)

### Java

```
public boolean hasCycle(int V, List<List<Integer>> adj) {
    boolean[] visited = new boolean[V], recStack = new boolean[V];
    for (int i = 0; i < V; i++)
        if (dfs(i, adj, visited, recStack))
            return true;
    return false;
}
private boolean dfs(int v, List<List<Integer>> adj, boolean[] visited, boolean[]
recStack) {
    if (recStack[v]) return true;
    if (visited[v]) return false;
    visited[v] = recStack[v] = true;
    for (int u : adj.get(v))
        if (dfs(u, adj, visited, recStack)) return true;
    recStack[v] = false;
    return false;
}
```

### C++

```
bool dfs(int v, vector<vector<int>>& adj, vector<bool>& visited, vector<bool>& recStack)
{
```

```
    if(recStack[v]) return true;
    if(visited[v]) return false;
    visited[v]=recStack[v]=true;
    for(int u: adj[v])
        if(dfs(u, adj, visited, recStack)) return true;
    recStack[v]=false;
    return false;
}
bool hasCycle(int V, vector<vector<int>>& adj) {
    vector<bool> visited(V, false), recStack(V, false);
    for(int i=0;i<V;i++)
        if(dfs(i, adj, visited, recStack)) return true;
    return false;
}
```

## 21. Dijkstra's Algorithm (Shortest Path)

### Java

```
public int[] dijkstra(int n, List<List<int[]>> graph, int src) {
    int[] dist = new int[n];
    Arrays.fill(dist, Integer.MAX_VALUE);
    dist[src] = 0;
    PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a->a[1]));
    pq.offer(new int[]{src, 0});
    while(!pq.isEmpty()) {
        int[] cur = pq.poll();
        int u = cur[0], d = cur[1];
        if(d > dist[u]) continue;
        for(int[] edge : graph.get(u)) {
            int v = edge[0], w = edge[1];
            if(dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.offer(new int[]{v, dist[v]});
            }
        }
    }
    return dist;
```

```
}
```

**C++**

```
vector<int> dijkstra(int n, vector<vector<pair<int,int>>>& graph, int src) {
    vector<int> dist(n, INT_MAX);
    dist[src]=0;
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;
    pq.push({0, src});
    while(!pq.empty()){
        int u = pq.top().second, d = pq.top().first; pq.pop();
        if(d > dist[u]) continue;
        for(auto& edge: graph[u]){
            int v = edge.first, w = edge.second;
            if(dist[v] > dist[u]+w) {
                dist[v]=dist[u]+w;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}
```

## 22. 0/1 Knapsack DP

**Java**

```
public int knapsack(int W, int[] wt, int[] val, int n) {
    int[][] dp = new int[n+1][W+1];
    for(int i=1;i<=n;i++)
        for(int w=1;w<=W;w++)
            if(wt[i-1]<=w)
                dp[i][w]=Math.max(val[i-1]+dp[i-1][w-wt[i-1]], dp[i-1][w]);
            else dp[i][w]=dp[i-1][w];
    return dp[n][W];
}
```

**C++**

```
int knapsack(int W, vector<int>& wt, vector<int>& val, int n) {
    vector<vector<int>> dp(n+1, vector<int>(W+1, 0));
    for(int i=1;i<=n;i++)
        for(int w=1;w<=W;w++)
            if(wt[i-1]<=w)
                dp[i][w]=max(val[i-1]+dp[i-1][w-wt[i-1]],dp[i-1][w]);
            else dp[i][w]=dp[i-1][w];
    return dp[n][W];
}
```

## 23. Coin Change (Min Coins)

**Java**

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

**C++**

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

## 24. Longest Increasing Subsequence

**Java**

```
public int lengthOfLIS(int[] nums) {
    int[] dp = new int[nums.length];
    int len = 0;
    for(int n: nums) {
        int i = Arrays.binarySearch(dp, 0, len, n);
        if(i<0) i=-(i+1);
        dp[i]=n;
        if(i==len) len++;
    }
    return len;
}
```

**C++**

```
int lengthOfLIS(vector<int>& nums) {
    vector<int> dp;
    for(int n: nums) {
        auto it = lower_bound(dp.begin(), dp.end(), n);
        if(it == dp.end()) dp.push_back(n);
        else *it = n;
    }
    return dp.size();
}
```

## 25. Edit Distance (Levenshtein distance)

**Java**

```
public int minDistance(String word1, String word2) {
    int m = word1.length(), n = word2.length();
    int[][] dp = new int[m+1][n+1];
    for(int i=0;i<=m;i++) dp[i][0]=i;
    for(int j=0;j<=n;j++) dp[0][j]=j;
    for(int i=1;i<=m;i++)
        for(int j=1;j<=n;j++)
            if(word1.charAt(i-1)==word2.charAt(j-1))
                dp[i][j]=dp[i-1][j-1];
            else
                dp[i][j]=1+Math.min(dp[i-1][j-1],Math.min(dp[i-1][j],dp[i][j-1]));
```

```
    return dp[m][n];
}
```

**C++**

```
int minDistance(string word1, string word2) {
    int m=word1.size(), n=word2.size();
    vector<vector<int>> dp(m+1, vector<int>(n+1));
    for(int i=0;i<=m;i++) dp[i][0]=i;
    for(int j=0;j<=n;j++) dp[0][j]=j;
    for(int i=1;i<=m;i++)
        for(int j=1;j<=n;j++)
            if(word1[i-1]==word2[j-1])
                dp[i][j]=dp[i-1][j-1];
            else
                dp[i][j]=1+min({dp[i-1][j-1], dp[i-1][j], dp[i][j-1]});
    return dp[m][n];
}
```

## 26. Spiral Order Matrix Traversal

**Java**

```
public List<Integer> spiralOrder(int[][] matrix) {
    List<Integer> res = new ArrayList<>();
    if(matrix.length==0) return res;
    int top=0, bottom=matrix.length-1, left=0, right=matrix[0].length-1;
    while(top<=bottom && left<=right){
        for(int i=left;i<=right;i++) res.add(matrix[top][i]);
        top++;
        for(int i=top;i<=bottom;i++) res.add(matrix[i][right]);
        right--;
        if(top<=bottom)
            for(int i=right;i>=left;i--) res.add(matrix[bottom][i]);
        bottom--;
        if(left<=right)
            for(int i=bottom;i>=top;i--) res.add(matrix[i][left]);
        left++;
    }
```

```
    return res;
}
```

**C++**

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    vector<int> res;
    if(matrix.empty()) return res;
    int top=0, bottom=matrix.size()-1, left=0, right=matrix[0].size()-1;
    while(top<=bottom && left<=right){
        for(int i=left;i<=right;i++) res.push_back(matrix[top][i]);
        top++;
        for(int i=top;i<=bottom;i++) res.push_back(matrix[i][right]);
        right--;
        if(top<=bottom)
            for(int i=right;i>=left;i--) res.push_back(matrix[bottom][i]);
        bottom--;
        if(left<=right)
            for(int i=bottom;i>=top;i--) res.push_back(matrix[i][left]);
        left++;
    }
    return res;
}
```

## 27. Rotate Matrix by 90 Degrees

**Java**

```
public void rotate(int[][] matrix) {
    int n = matrix.length;
    for(int i=0;i<n;i++)
        for(int j=i;j<n;j++){
            int t = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = t;
        }
    for(int i=0;i<n;i++)
        for(int j=0;j<n/2;j++){
            int t = matrix[i][j];
```

```
            matrix[i][j] = matrix[i][n-1-j];
            matrix[i][n-1-j]=t;
        }
}
```

**C++**

```
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(int i=0;i<n;i++)
        reverse(matrix[i].begin(), matrix[i].end());
}
```

## 28. Set Matrix Zeroes

### Java

```
public void setZeroes(int[][] matrix) {
    boolean firstRow = false, firstCol = false;
    int m = matrix.length, n = matrix[0].length;
    for(int i=0;i<m;i++) if(matrix[i][0]==0) firstCol=true;
    for(int j=0;j<n;j++) if(matrix[0][j]==0) firstRow=true;
    for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
            if(matrix[i][j]==0)
                matrix[i][0]=matrix[0][j]=0;
    for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
            if(matrix[i][0]==0||matrix[0][j]==0)
                matrix[i][j]=0;
    if(firstCol) for(int i=0;i<m;i++) matrix[i][0]=0;
    if(firstRow) for(int j=0;j<n;j++) matrix[0][j]=0;
}
```

### C++

```
void setZeroes(vector<vector<int>>& matrix) {
    bool firstRow = false, firstCol = false;
    int m = matrix.size(), n = matrix[0].size();
    for(int i=0;i<m;i++) if(matrix[i][0]==0) firstCol=true;
    for(int j=0;j<n;j++) if(matrix[0][j]==0) firstRow=true;
    for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
            if(matrix[i][j]==0)
                matrix[i][0]=matrix[0][j]=0;
    for(int i=1;i<m;i++)
        for(int j=1;j<n;j++)
            if(matrix[i][0]==0||matrix[0][j]==0)
                matrix[i][j]=0;
    if(firstCol) for(int i=0;i<m;i++) matrix[i][0]=0;
    if(firstRow) for(int j=0;j<n;j++) matrix[0][j]=0;
```

```
}
```

## 29. Word Search in Matrix

### Java

```
public boolean exist(char[][] board, String word) {
    for(int i=0;i<board.length;i++)
        for(int j=0;j<board[0].length;j++)
            if(backtrack(board, word, i, j, 0)) return true;
    return false;
}
private boolean backtrack(char[][] board, String word, int i, int j, int k) {
    if(k==word.length()) return true;
    if(i<0||j<0||i>=board.length||j>=board[0].length||board[i][j]!=word.charAt(k))
return false;
    char temp = board[i][j];
    board[i][j] = '.';
    boolean found = backtrack(board, word, i+1, j, k+1) ||
                    backtrack(board, word, i-1, j, k+1) ||
                    backtrack(board, word, i, j+1, k+1) ||
                    backtrack(board, word, i, j-1, k+1);
    board[i][j] = temp;
    return found;
}
```

### C++

```
bool backtrack(vector<vector<char>>& board, string& word, int i, int j, int k) {
    if(k == word.size()) return true;
    if(i<0||j<0||i>=board.size()||j>=board[0].size()||board[i][j]!=word[k]) return
false;
    char temp = board[i][j];
    board[i][j] = '.';
    bool found = backtrack(board, word, i+1, j, k+1) ||
                 backtrack(board, word, i-1, j, k+1) ||
                 backtrack(board, word, i, j+1, k+1) ||
                 backtrack(board, word, i, j-1, k+1);
    board[i][j] = temp;
```

```
    return found;
}
bool exist(vector<vector<char>>& board, string word) {
    for(int i=0;i<board.size();i++)
        for(int j=0;j<board[0].size();j++)
            if(backtrack(board, word, i, j, 0)) return true;
    return false;
}
```

## 30. Find Median of Two Sorted Arrays

**Java**

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    int[] nums = new int[nums1.length + nums2.length];
    int i=0, j=0, k=0;
    while(i<nums1.length && j<nums2.length)
        nums[k++] = nums1[i]<nums2[j] ? nums1[i++] : nums2[j++];
    while(i<nums1.length) nums[k++] = nums1[i++];
    while(j<nums2.length) nums[k++] = nums2[j++];
    int n = nums.length;
    if(n%2==0) return (nums[n/2-1]+nums[n/2])/2.0;
    return nums[n/2];
}
```

**C++**

```
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    vector<int> nums(nums1.size() + nums2.size());
    merge(nums1.begin(), nums1.end(), nums2.begin(), nums2.end(), nums.begin());
    int n = nums.size();
    if(n%2==0) return (nums[n/2-1]+nums[n/2])/2.0;
    return nums[n/2];
}
```

## 31. Maximum Product Subarray

**Java**

```
public int maxProduct(int[] nums) {
    int res = nums[0], maxP = nums[0], minP = nums[0];
    for(int i=1;i<nums.length;i++) {
        int tmp = maxP;
        maxP = Math.max(nums[i], Math.max(maxP*nums[i], minP*nums[i]));
        minP = Math.min(nums[i], Math.min(tmp*nums[i], minP*nums[i]));
        res = Math.max(res, maxP);
    }
    return res;
}
```

**C++**

```
int maxProduct(vector<int>& nums) {
    int res = nums[0], maxP = nums[0], minP = nums[0];
    for(int i=1;i<nums.size();i++) {
        int temp = maxP;
        maxP = max(nums[i], max(maxP*nums[i], minP*nums[i]));
        minP = min(nums[i], min(temp*nums[i], minP*nums[i]));
        res = max(res, maxP);
    }
    return res;
}
```

## 32. Trie Insert and Search

### Java

```
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEnd = false;
}
class Trie {
    TrieNode root = new TrieNode();
    public void insert(String word) {
        TrieNode node = root;
        for(char c: word.toCharArray()) {
            int i = c-'a';
            if(node.children[i]==null)
```

```
                node.children[i]=new TrieNode();
            node = node.children[i];
        }
        node.isEnd=true;
    }
    public boolean search(String word) {
        TrieNode node = root;
        for(char c: word.toCharArray()) {
            int i=c-'a';
            if(node.children[i]==null) return false;
            node = node.children[i];
        }
        return node.isEnd;
    }
}
```

**C++**

```
struct TrieNode {
    TrieNode* children[26];
    bool isEnd;
    TrieNode() { for(int i=0;i<26;i++) children[i]=nullptr; isEnd=false;}
};
class Trie {
    TrieNode* root;
public:
    Trie() { root = new TrieNode(); }
    void insert(string word) {
        TrieNode* node = root;
        for(char c : word) {
            int i = c-'a';
            if(!node->children[i]) node->children[i]=new TrieNode();
            node = node->children[i];
        }
        node->isEnd=true;
    }
    bool search(string word) {
        TrieNode* node = root;
        for(char c : word) {
            int i=c-'a';
```

```
            if(!node->children[i]) return false;
            node = node->children[i];
        }
        return node->isEnd;
    }
};
```

## 33. Activity Selection Problem

### Java

```
public List<Integer> activitySelection(int[] start, int[] end) {
    int n = start.length;
    List<int[]> activities = new ArrayList<>();
    for(int i=0;i<n;i++) activities.add(new int[]{start[i], end[i], i});
    activities.sort(Comparator.comparingInt(a -> a[1]));
    List<Integer> res = new ArrayList<>();
    int prevEnd = -1;
    for(int[] act: activities)
        if(act[0] >= prevEnd) {
            res.add(act[2]);
            prevEnd = act[1];
        }
    return res;
}
```

### C++

```
vector<int> activitySelection(vector<int>& start, vector<int>& end) {
    int n = start.size();
    vector<tuple<int, int, int>> activities;
    for(int i=0;i<n;i++) activities.push_back({end[i], start[i], i});
    sort(activities.begin(), activities.end());
    vector<int> res;
    int prevEnd = -1;
    for(auto& act: activities)
        if(get<1>(act) >= prevEnd) {
            res.push_back(get<2>(act));
            prevEnd = get<0>(act);
```

```
        }
    return res;
}
```

## 34. Largest Rectangle in Histogram

### Java

```
public int largestRectangleArea(int[] heights) {
    Stack<Integer> stack = new Stack<>();
    int max = 0;
    for(int i=0;i<=heights.length;i++) {
        int h = i == heights.length ? 0 : heights[i];
        while(!stack.isEmpty() && h < heights[stack.peek()]) {
            int height = heights[stack.pop()];
            int width = stack.isEmpty() ? i : i-stack.peek()-1;
            max = Math.max(max, height*width);
        }
        stack.push(i);
    }
    return max;
}
```

### C++

```
int largestRectangleArea(vector<int>& heights) {
    stack<int> st;
    int max_area=0, n=heights.size();
    for(int i=0;i<=n;i++){
        int h = i==n ? 0: heights[i];
        while(!st.empty() && h < heights[st.top()]) {
            int height = heights[st.top()]; st.pop();
            int width = st.empty() ? i : i-st.top()-1;
            max_area = max(max_area, height*width);
        }
        st.push(i);
    }
    return max_area;
```

```
}
```

## 35. Count All Subsets

### Java

```
public List<List<Integer>> subsets(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    backtrack(res, new ArrayList<>(), nums, 0);
    return res;
}
private void backtrack(List<List<Integer>> res, List<Integer> curr, int[] nums, int
start) {
    res.add(new ArrayList<>(curr));
    for(int i=start;i<nums.length;i++){
        curr.add(nums[i]);
        backtrack(res, curr, nums, i+1);
        curr.remove(curr.size()-1);
    }
}
```

### C++

```
vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> res;
    vector<int> curr;
    function<void(int)> dfs = [&](int i) {
        if (i == nums.size()) { res.push_back(curr); return; }
        dfs(i+1);
        curr.push_back(nums[i]);
        dfs(i+1);
        curr.pop_back();
    };
    dfs(0);
    return res;
}
```

## 36. Generate All Permutations

**Java**

```
public List<List<Integer>> permute(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    boolean[] used = new boolean[nums.length];
    backtrack(res, new ArrayList<>(), nums, used);
    return res;
}
private void backtrack(List<List<Integer>> res, List<Integer> curr, int[] nums,
boolean[] used) {
    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(res, curr, nums, used);
        curr.remove(curr.size()-1);
        used[i] = false;
    }
}
```

**C++**

```
vector<vector<int>> permute(vector<int>& nums) {
    vector<vector<int>> res;
    vector<int> curr;
    vector<bool> used(nums.size(), false);
    function<void()> backtrack = [&]() {
        if(curr.size() == nums.size()) { res.push_back(curr); return; }
        for(int i=0;i<nums.size();i++) {
            if(used[i]) continue;
            used[i] = true;
            curr.push_back(nums[i]);
            backtrack();
            curr.pop_back();
            used[i] = false;
        }
```

```
    };
    backtrack();
    return res;
}
```

## 37. Power Set

### Java

```
public List<List<Integer>> powerSet(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    res.add(new ArrayList<>());
    for(int n: nums) {
        int size = res.size();
        for(int i=0;i<size;i++) {
            List<Integer> subset = new ArrayList<>(res.get(i));
            subset.add(n);
            res.add(subset);
        }
    }
    return res;
}
```

### C++

```
vector<vector<int>> powerSet(vector<int>& nums) {
    vector<vector<int>> res(1);
    for(int n: nums) {
        int size = res.size();
        for(int i=0;i<size;i++) {
            auto tmp = res[i];
            tmp.push_back(n);
            res.push_back(tmp);
        }
    }
    return res;
}
```

## 38. Intersection of Two Linked Lists

### Java

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    ListNode a = headA, b = headB;
    while(a != b) {
        a = (a == null) ? headB : a.next;
        b = (b == null) ? headA : b.next;
    }
    return a;
}
```

### C++

```
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;
}
```

## 39. Print All Palindromic Substrings Count

### Java

```
public int countSubstrings(String s) {
    int n = s.length(), res = 0;
    for(int i=0;i<n;i++) {
        res += countPalin(s, i, i);
        res += countPalin(s, i, i+1);
    }
    return res;
}
private int countPalin(String s, int l, int r) {
    int cnt=0;
    while(l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)) {
        cnt++; l--; r++;
```

```
    }
    return cnt;
}
```

**C++**

```
int countPalindromicSubstrings(string s) {
    int n = s.length(), res = 0;
    auto count = [&](int l, int r) {
        int cnt = 0;
        while(l>=0 && r<n && s[l]==s[r]) { cnt++; l--; r++; }
        return cnt;
    };
    for(int i=0;i<n;i++) {
        res += count(i, i);
        res += count(i, i+1);
    }
    return res;
}
```

## 40. Detect Cycle in Undirected Graph using BFS

**Java**

```
public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj) {
    boolean[] visited = new boolean[V];
    for(int i=0; i<V; i++)
        if(!visited[i] && bfs(i, adj, visited)) return true;
    return false;
}
private boolean bfs(int src, ArrayList<ArrayList<Integer>> adj, boolean[] visited) {
    Queue<int[]> q = new LinkedList<>();
    q.add(new int[]{src, -1});
    visited[src] = true;
    while(!q.isEmpty()) {
        int[] node = q.poll();
        int u = node[0], parent = node[1];
        for(int v: adj.get(u)) {
            if(!visited[v]) {
```

```
                visited[v] = true; q.add(new int[]{v, u});
            } else if(v != parent) return true;
        }
    }
    return false;
}
```

**C++**

```
bool bfs(int src, vector<vector<int>>& adj, vector<bool>& visited) {
    queue<pair<int, int>> q;
    q.push({src, -1}); visited[src] = true;
    while(!q.empty()) {
        auto[node, parent] = q.front(); q.pop();
        for(int v: adj[node]) {
            if(!visited[v]) {
                visited[v]=true; q.push({v, node});
            } else if(v != parent) return true;
        }
    }
    return false;
}
bool isCycle(int V, vector<vector<int>>& adj) {
    vector<bool> visited(V, false);
    for(int i=0;i<V;i++)
        if(!visited[i] && bfs(i, adj, visited)) return true;
    return false;
}
```

## 41. Find All Palindromic Substrings

### Java

```
public int countSubstrings(String s) {
    int n = s.length(), res = 0;
    for (int i = 0; i < n; i++) {
        res += countAround(s, i, i);
        res += countAround(s, i, i + 1);
    }
    return res;
}
private int countAround(String s, int l, int r) {
    int cnt = 0;
    while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
        cnt++; l--; r++;
    }
    return cnt;
}
```

### C++

```
int countSubstrings(string s) {
    int n = s.size(), res = 0;
    for (int i = 0; i < n; i++) {
        for (int l = i, r = i; l >= 0 && r < n && s[l] == s[r]; l--, r++) res++;
        for (int l = i, r = i + 1; l >= 0 && r < n && s[l] == s[r]; l--, r++) res++;
    }
    return res;
}
```

## 42. Longest Palindromic Substring

### Java

```
public String longestPalindrome(String s) {
    int start = 0, maxLen = 1;
    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 > maxLen) {
            start = i - (len - 1) / 2;
            maxLen = len;
        }
    }
    return s.substring(start, start + maxLen);
}
private 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;
}
```

**C++**

```
string longestPalindrome(string s) {
    int start = 0, maxLen = 1;
    auto expand = [&](int l, int r) {
        while (l >= 0 && r < s.size() && s[l] == s[r]) { l--; r++; }
        return r - l - 1;
    };
    for (int i = 0; i < s.size(); i++) {
        int len1 = expand(i, i), len2 = expand(i, i + 1);
        int len = max(len1, len2);
        if (len > maxLen) {
            start = i - (len - 1) / 2;
            maxLen = len;
        }
    }
    return s.substr(start, maxLen);
}
```

## 43. Subsets with Duplicates

**Java**

```
public List<List<Integer>> subsetsWithDup(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> res = new ArrayList<>();
    backtrack(res, new ArrayList<>(), nums, 0);
    return res;
}
private void backtrack(List<List<Integer>> res, List<Integer> curr, int[] nums, int
start) {
    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(res, curr, nums, i + 1);
        curr.remove(curr.size() - 1);
    }
}
```

**C++**

```
vector<vector<int>> subsetsWithDup(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    vector<vector<int>> res; vector<int> curr;
    function<void(int)> dfs = [&](int start) {
        res.push_back(curr);
        for (int i = start; i < nums.size(); i++) {
            if (i > start && nums[i] == nums[i-1]) continue;
            curr.push_back(nums[i]);
            dfs(i+1);
            curr.pop_back();
        }
    };
    dfs(0);
    return res;
}
```

## 44. Find Duplicates in Array

**Java**

```
public List<Integer> findDuplicates(int[] nums) {
    List<Integer> res = new ArrayList<>();
    for (int n : nums) {
        int idx = Math.abs(n) - 1;
        if (nums[idx] < 0) res.add(idx+1);
        nums[idx] *= -1;
    }
    return res;
}
```

**C++**

```
vector<int> findDuplicates(vector<int>& nums) {
    vector<int> res;
    for (int n : nums) {
        int idx = abs(n) - 1;
        if (nums[idx] < 0) res.push_back(idx+1);
        nums[idx] *= -1;
    }
    return res;
}
```

## 45. Minimum Path Sum in a 2D Grid

**Java**

```
public int minPathSum(int[][] grid) {
    for (int i = 1; i < grid.length; i++) grid[i][0] += grid[i-1][0];
    for (int j = 1; j < grid[0].length; j++) grid[0][j] += grid[0][j-1];
    for (int i = 1; i < grid.length; i++)
        for (int j = 1; j < grid[0].length; j++)
            grid[i][j] += Math.min(grid[i-1][j], grid[i][j-1]);
    return grid[grid.length-1][grid[0].length-1];
}
```

**C++**

```
int minPathSum(vector<vector<int>>& grid) {
    int m = grid.size(), n = grid[0].size();
```

```
    for (int i = 1; i < m; i++) grid[i][0] += grid[i-1][0];
    for (int j = 1; j < n; j++) grid[0][j] += grid[0][j-1];
    for (int i = 1; i < m; i++)
        for (int j = 1; j < n; j++)
            grid[i][j] += min(grid[i-1][j], grid[i][j-1]);
    return grid[m-1][n-1];
}
```

## 46. Reverse Words in a String

### Java

```
public String reverseWords(String s) {
    String[] words = s.trim().split("\\s+");
    Collections.reverse(Arrays.asList(words));
    return String.join(" ", words);
}
```

### C++

```
string reverseWords(string s) {
    istringstream iss(s);
    vector<string> words;
    while (iss >> s) words.push_back(s);
    reverse(words.begin(), words.end());
    string res;
    for (string& w : words) res += w + " ";
    res.pop_back();
    return res;
}
```

## 47. Merge Intervals

### Java

```
public int[][] merge(int[][] intervals) {
    Arrays.sort(intervals, (a,b) -> a[0] - b[0]);
    List<int[]> res = new ArrayList<>();
    int[] prev = intervals[0];
```

```
    for (int i = 1; i < intervals.length; i++) {
        if (intervals[i][0] <= prev[1])
            prev[1] = Math.max(prev[1], intervals[i][1]);
        else {
            res.add(prev);
            prev = intervals[i];
        }
    }
    res.add(prev);
    return res.toArray(new int[res.size()][]);
}
```

**C++**

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    sort(intervals.begin(), intervals.end());
    vector<vector<int>> res;
    vector<int> prev = intervals[0];
    for (int i = 1; i < intervals.size(); i++) {
        if (intervals[i][0] <= prev[1])
            prev[1] = max(prev[1], intervals[i][1]);
        else {
            res.push_back(prev);
            prev = intervals[i];
        }
    }
    res.push_back(prev);
    return res;
}
```

## 48. Longest Consecutive Sequence

**Java**

```
public int longestConsecutive(int[] nums) {
    Set<Integer> set = new HashSet<>();
    for (int n : nums) set.add(n);
    int longest = 0;
    for (int n : set) {
```

```
        if (!set.contains(n - 1)) {
            int curr = n, streak = 1;
            while (set.contains(curr + 1)) {
                curr++; streak++;
            }
            longest = Math.max(longest, streak);
        }
    }
    return longest;
}
```

**C++**

```
int longestConsecutive(vector<int>& nums) {
    unordered_set<int> set(nums.begin(), nums.end());
    int longest = 0;
    for (int n : set) {
        if (!set.count(n - 1)) {
            int curr = n, streak = 1;
            while (set.count(curr + 1)) { curr++; streak++; }
            longest = max(longest, streak);
        }
    }
    return longest;
}
```

## 49. Search in Rotated Sorted Array

**Java**

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

```
            if (nums[mid] < target && target <= nums[right]) left = mid + 1;
            else right = mid - 1;
        }
    }
    return -1;
}
```

**C++**

```
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[left] <= nums[mid]) {
            if (nums[left] <= target && target < nums[mid]) right = mid - 1;
            else left = mid + 1;
        } else {
            if (nums[mid] < target && target <= nums[right]) left = mid + 1;
            else right = mid - 1;
        }
    }
    return -1;
}
```

## 50. Kth Largest Element in Array

**Java**

```
public int findKthLargest(int[] nums, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int n : nums) {
        pq.offer(n);
        if (pq.size() > k) pq.poll();
    }
    return pq.peek();
}
```

**C++**

```
int findKthLargest(vector<int>& nums, int k) {
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int n : nums) {
        pq.push(n);
        if (pq.size() > k) pq.pop();
    }
    return pq.top();
}
```