```java

/**
 * Top30ArrayProblems.java
 *
 * Implementations for Top 30 Medium Array Interview Problems (static methods).
 * Each method includes a short comment describing the approach and complexity.
 *
 * Compile & run:
 *   javac Top30ArrayProblems.java && java Top30ArrayProblems
 *
 * Author: ChatGPT
 */

import java.util.*;

public class Top30ArrayProblems {

    // 1. Two Sum (HashMap)
    public static int[] twoSum(int[] nums, int target) {
        Map<Integer,Integer> map = new HashMap<>();
        for (int i=0;i<nums.length;i++) {
            int need = target - nums[i];
            if (map.containsKey(need)) return new int[]{map.get(need), i};
            map.put(nums[i], i);
        }
        return new int[0];
    }

    // 2. 3Sum (sort + two pointers)
    public static List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        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 s = nums[i] + nums[l] + nums[r];
                if (s==0) {
                    res.add(Arrays.asList(nums[i], nums[l], nums[r]));
                    while (l<r && nums[l]==nums[l+1]) l++;
                    while (l<r && nums[r]==nums[r-1]) r--;
                    l++; r--;
                } else if (s<0) l++; else r--;
            }
        }
        return res;
    }

    // 3. 4Sum (generalize k-sum via recursion)
    public static List<List<Integer>> fourSum(int[] nums, int target) {
        Arrays.sort(nums);
        return kSum(nums, target, 4, 0);
    }
    private static List<List<Integer>> kSum(int[] nums, int target, int k, int start) {
        List<List<Integer>> res = new ArrayList<>();
        if (start >= nums.length) return res;
        if (k == 2) {
            int l = start, r = nums.length - 1;
            while (l < r) {
                int sum = nums[l] + nums[r];
                if (sum == target) {
                    res.add(Arrays.asList(nums[l], nums[r]));
                    while (l<r && nums[l]==nums[l+1]) l++;
                    while (l<r && nums[r]==nums[r-1]) r--;
                    l++; r--;
                } else if (sum < target) l++; else r--;
            }
        } else {
            for (int i = start; i < nums.length - (k - 1); i++) {
```

```
                if (i > start && nums[i] == nums[i-1]) continue;
                for (List<Integer> subset : kSum(nums, target - nums[i], k-1, i+1)) {
                    List<Integer> combined = new ArrayList<>(); combined.add(nums[i]);
                    combined.addAll(subset);
                    res.add(combined);
                }
            }
        }
        return res;
    }

    // 4. Longest Consecutive Sequence (HashSet)
    public static int longestConsecutive(int[] nums) {
        Set<Integer> set = new HashSet<>();
        for (int n: nums) set.add(n);
        int best = 0;
        for (int n: set) {
            if (!set.contains(n-1)) {
                int cur = n, len = 1;
                while (set.contains(cur+1)) { cur++; len++; }
                best = Math.max(best, len);
            }
        }
        return best;
    }

    // 5. Majority Element (Boyer-Moore)
    public static int majorityElement(int[] nums) {
        int cand = 0, cnt = 0;
        for (int n: nums) {
            if (cnt==0) { cand=n; cnt=1; }
            else if (n==cand) cnt++; else cnt--;
        }
        return cand;
    }

    // 6. Majority Element II (Boyer-Moore extended)
    public static List<Integer> majorityElementII(int[] nums) {
        int cand1=0,cand2=0,c1=0,c2=0;
        for (int n: nums) {
            if (n==cand1) c1++;
            else if (n==cand2) c2++;
            else if (c1==0) { cand1=n; c1=1; }
            else if (c2==0) { cand2=n; c2=1; }
            else { c1--; c2--; }
        }
        List<Integer> res = new ArrayList<>();
        int cnt1=0, cnt2=0;
        for (int n: nums) { if (n==cand1) cnt1++; else if (n==cand2) cnt2++; }
        if (cnt1 > nums.length/3) res.add(cand1);
        if (cnt2 > nums.length/3) res.add(cand2);
        return res;
    }

    // 7. Next Permutation
    public static void nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i>=0 && nums[i] >= nums[i+1]) i--;
        if (i>=0) {
            int j = nums.length - 1;
            while (nums[j] <= nums[i]) j--;
            swap(nums, i, j);
        }
        reverse(nums, i+1, nums.length-1);
    }

    // 8. Rotate Array (reverse method)
    public static void rotate(int[] nums, int k) {
        int n = nums.length; k %= n; if (k<0) k+=n;
```

```java
        reverse(nums, 0, n-1); reverse(nums, 0, k-1); reverse(nums, k, n-1);
    }

    // 9. Product of Array Except Self (prefix/suffix)
    public static int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] res = new int[n];
        int prefix = 1;
        for (int i=0;i<n;i++) { res[i] = prefix; prefix *= nums[i]; }
        int suffix = 1;
        for (int i=n-1;i>=0;i--) { res[i] *= suffix; suffix *= nums[i]; }
        return res;
    }

    // 10. Maximum Subarray Sum (Kadane)
    public static int maxSubArray(int[] nums) {
        int max = nums[0], cur = nums[0];
        for (int i=1;i<nums.length;i++) {
            cur = Math.max(nums[i], cur + nums[i]);
            max = Math.max(max, cur);
        }
        return max;
    }

    // 11. Maximum Product Subarray
    public static int maxProduct(int[] nums) {
        int max = nums[0], min = nums[0], res = nums[0];
        for (int i=1;i<nums.length;i++) {
            int a = nums[i];
            if (a < 0) { int tmp=max; max=min; min=tmp; }
            max = Math.max(a, max * a);
            min = Math.min(a, min * a);
            res = Math.max(res, max);
        }
        return res;
    }

    // 12. Find Minimum in Rotated Sorted Array
    public static int findMinRotated(int[] nums) {
        int l=0, r=nums.length-1;
        while (l<r) {
            int m = l + (r-l)/2;
            if (nums[m] > nums[r]) l = m+1;
            else r = m;
        }
        return nums[l];
    }

    // 13. Search in Rotated Sorted Array
    public static int searchRotated(int[] nums, int target) {
        int l=0, r=nums.length-1;
        while (l<=r) {
            int m = (l+r)/2;
            if (nums[m]==target) return m;
            if (nums[l] <= nums[m]) {
                if (nums[l] <= target && target < nums[m]) r = m-1;
                else l = m+1;
            } else {
                if (nums[m] < target && target <= nums[r]) l = m+1;
                else r = m-1;
            }
        }
        return -1;
    }

    // 14. Find the Duplicate Number (Floyd's cycle detection)
    public static int findDuplicate(int[] nums) {
        int slow = nums[0], fast = nums[0];
        do {
```

```java
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);
        slow = nums[0];
        while (slow != fast) { slow = nums[slow]; fast = nums[fast]; }
        return slow;
    }

    // 15. Set Matrix Zeroes (use first row/col as markers)
    public static void setZeroes(int[][] matrix) {
        boolean row0 = false, col0 = false;
        int m = matrix.length, n = matrix[0].length;
        for (int j=0;j<n;j++) if (matrix[0][j]==0) row0=true;
        for (int i=0;i<m;i++) if (matrix[i][0]==0) col0=true;
        for (int i=1;i<m;i++) {
            for (int j=1;j<n;j++) if (matrix[i][j]==0) { matrix[i][0]=0; matrix[0][j]=0;
            }
        }
        for (int i=1;i<m;i++) if (matrix[i][0]==0) Arrays.fill(matrix[i], 0);
        for (int j=1;j<n;j++) if (matrix[0][j]==0) for (int i=0;i<m;i++) matrix[i][j]=0;
        if (row0) Arrays.fill(matrix[0], 0);
        if (col0) for (int i=0;i<m;i++) matrix[i][0]=0;
    }

    // 16. Spiral Matrix
    public static List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> res = new ArrayList<>();
        if (matrix==null || 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 j=left;j<=right;j++) res.add(matrix[top][j]);
            top++;
            for (int i=top;i<=bottom;i++) res.add(matrix[i][right]);
            right--;
            if (top<=bottom) for (int j=right;j>=left;j--) res.add(matrix[bottom][j]);
            bottom--;
            if (left<=right) for (int i=bottom;i>=top;i--) res.add(matrix[i][left]);
            left++;
        }
        return res;
    }

    // 17. Merge Intervals
    public static int[][] mergeIntervals(int[][] intervals) {
        if (intervals==null || intervals.length==0) return new int[0][];
        Arrays.sort(intervals, (a,b)->Integer.compare(a[0], b[0]));
        List<int[]> out = new ArrayList<>();
        for (int[] it: intervals) {
            if (out.isEmpty() || out.get(out.size()-1)[1] < it[0]) out.add(it);
            else out.get(out.size()-1)[1] = Math.max(out.get(out.size()-1)[1], it[1]);
        }
        return out.toArray(new int[out.size()][]);
    }

    // 18. Insert Interval
    public static int[][] insertInterval(int[][] intervals, int[] newInterval) {
        List<int[]> res = new ArrayList<>();
        int i=0, n = intervals.length;
        while (i<n && intervals[i][1] < newInterval[0]) res.add(intervals[i++]);
        while (i<n && intervals[i][0] <= newInterval[1]) {
            newInterval[0] = Math.min(newInterval[0], intervals[i][0]);
            newInterval[1] = Math.max(newInterval[1], intervals[i][1]);
            i++;
        }
        res.add(newInterval);
        while (i<n) res.add(intervals[i++]);
        return res.toArray(new int[res.size()][]);
    }

```

```java
    // 19. Interval List Intersections
    public static int[][] intervalIntersection(int[][] A, int[][] B) {
        List<int[]> res = new ArrayList<>();
        int i=0,j=0;
        while (i<A.length && j<B.length) {
            int lo = Math.max(A[i][0], B[j][0]);
            int hi = Math.min(A[i][1], B[j][1]);
            if (lo <= hi) res.add(new int[]{lo,hi});
            if (A[i][1] < B[j][1]) i++; else j++;
        }
        return res.toArray(new int[res.size()][]);
    }

    // 20. Subarray Sum Equals K (prefix sum hashmap)
    public static int subarraySum(int[] nums, int k) {
        Map<Integer,Integer> count = new HashMap<>();
        count.put(0,1);
        int sum=0, res=0;
        for (int x: nums) {
            sum += x;
            res += count.getOrDefault(sum - k, 0);
            count.put(sum, count.getOrDefault(sum,0)+1);
        }
        return res;
    }

    // 21. Minimum Size Subarray Sum (sliding window)
    public static int minSubArrayLen(int s, int[] nums) {
        int left=0, sum=0, res=Integer.MAX_VALUE;
        for (int i=0;i<nums.length;i++) {
            sum += nums[i];
            while (sum >= s) {
                res = Math.min(res, i-left+1);
                sum -= nums[left++];
            }
        }
        return (res==Integer.MAX_VALUE)?0:res;
    }

    // 22. Longest Substring Without Repeating Characters (sliding window)
    public static int lengthOfLongestSubstring(String s) {
        Map<Character,Integer> last = new HashMap<>();
        int left=0, res=0;
        for (int i=0;i<s.length();i++) {
            char c = s.charAt(i);
            if (last.containsKey(c)) left = Math.max(left, last.get(c)+1);
            last.put(c, i);
            res = Math.max(res, i-left+1);
        }
        return res;
    }

    // 23. Container With Most Water (two pointers)
    public static int maxArea(int[] height) {
        int l=0, r=height.length-1, ans=0;
        while (l<r) {
            ans = Math.max(ans, Math.min(height[l], height[r]) * (r-l));
            if (height[l] < height[r]) l++; else r--;
        }
        return ans;
    }

    // 24. Trapping Rain Water (two pointers)
    public static int trap(int[] height) {
        int l=0, r=height.length-1;
        int leftMax=0, rightMax=0, trapped=0;
        while (l<r) {
            if (height[l] < height[r]) {
                if (height[l] >= leftMax) leftMax = height[l];
```

```
                else trapped += leftMax - height[l];
                l++;
            } else {
                if (height[r] >= rightMax) rightMax = height[r];
                else trapped += rightMax - height[r];
                r--;
            }
        }
        return trapped;
    }

    // 25. Jump Game (greedy max reach)
    public static boolean canJump(int[] nums) {
        int reach = 0;
        for (int i=0;i<nums.length;i++) {
            if (i > reach) return false;
            reach = Math.max(reach, i + nums[i]);
        }
        return true;
    }

    // 26. Jump Game II (greedy BFS-like)
    public static int jump(int[] nums) {
        int jumps=0, curEnd=0, curFarthest=0;
        for (int i=0;i<nums.length-1;i++) {
            curFarthest = Math.max(curFarthest, i + nums[i]);
            if (i == curEnd) {
                jumps++;
                curEnd = curFarthest;
            }
        }
        return jumps;
    }

    // 27. Sort Colors (Dutch National Flag)
    public static void sortColors(int[] nums) {
        int low=0, mid=0, high=nums.length-1;
        while (mid <= high) {
            if (nums[mid]==0) swap(nums, low++, mid++);
            else if (nums[mid]==1) mid++;
            else swap(nums, mid, high--);
        }
    }

    // 28. Find First and Last Position of Element in Sorted Array
    public static int[] searchRange(int[] nums, int target) {
        int left = findBound(nums, target, true);
        int right = findBound(nums, target, false);
        return new int[]{left, right};
    }
    private static int findBound(int[] nums, int target, boolean left) {
        int l=0, r=nums.length-1, bound=-1;
        while (l<=r) {
            int m = (l+r)/2;
            if (nums[m]==target) { bound=m; if (left) r=m-1; else l=m+1; }
            else if (nums[m] < target) l = m+1; else r = m-1;
        }
        return (bound==-1)? -1 : bound;
    }

    // 29. Find Peak Element
    public static int findPeakElement(int[] nums) {
        int l=0, r=nums.length-1;
        while (l<r) {
            int m=(l+r)/2;
            if (nums[m] > nums[m+1]) r = m;
            else l = m+1;
        }
        return l;
```

```java
    }

    // 30. Minimum Swaps to Group All 1's Together (sliding window count)
    public static int minSwaps(int[] nums) {
        int k=0;
        for (int n: nums) if (n==1) k++;
        if (k==0) return 0;
        int bad=0;
        for (int i=0;i<k;i++) if (nums[i]==0) bad++;
        int ans = bad;
        for (int i=k;i<nums.length;i++) {
            if (nums[i-k]==0) bad--;
            if (nums[i]==0) bad++;
            ans = Math.min(ans, bad);
        }
        return ans;
    }

      /**
     * 1. Two Sum
     * Find indices of the two numbers that add up to target.
     * Time: O(n), Space: O(n)
     */
    public static 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[]{};
    }

    /**
     * 2. Best Time to Buy and Sell Stock (Single Transaction)
     * Time: O(n), Space: O(1)
     */
    public static int maxProfit(int[] prices) {
        int minPrice = Integer.MAX_VALUE, maxProfit = 0;
        for (int price : prices) {
            minPrice = Math.min(minPrice, price);
            maxProfit = Math.max(maxProfit, price - minPrice);
        }
        return maxProfit;
    }

    /**
     * 3. Maximum Subarray (Kadane's Algorithm)
     * Time: O(n), Space: O(1)
     */
    public static int maxSubArray(int[] nums) {
        int maxSoFar = nums[0], curr = nums[0];
        for (int i = 1; i < nums.length; i++) {
            curr = Math.max(nums[i], curr + nums[i]);
            maxSoFar = Math.max(maxSoFar, curr);
        }
        return maxSoFar;
    }

    /**
     * 4. Find Duplicate Number
     * Using Floyd's cycle detection.
     * Time: O(n), Space: O(1)
     */
    public static int findDuplicate(int[] nums) {
        int slow = nums[0], fast = nums[0];
        do {
```

```java
            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. Merge Intervals
     * Time: O(n log n), Space: O(n)
     */
    public static int[][] mergeIntervals(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> merged = new ArrayList<>();
        for (int[] interval : intervals) {
            if (merged.isEmpty() || merged.get(merged.size() - 1)[1] < interval[0]) {
                merged.add(interval);
            } else {
                merged.get(merged.size() - 1)[1] = Math.max(merged.get(merged.size() - 1
                )[1], interval[1]);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }

    /**
     * 6. Product of Array Except Self
     * Time: O(n), Space: O(1) extra
     */
    public static int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] res = new int[n];
        Arrays.fill(res, 1);
        int prefix = 1;
        for (int i = 0; i < n; i++) {
            res[i] = prefix;
            prefix *= nums[i];
        }
        int suffix = 1;
        for (int i = n - 1; i >= 0; i--) {
            res[i] *= suffix;
            suffix *= nums[i];
        }
        return res;
    }

    /**
     * 7. Longest Consecutive Sequence
     * Time: O(n), Space: O(n)
     */
    public static int longestConsecutive(int[] nums) {
        Set<Integer> set = new HashSet<>();
        for (int num : nums) set.add(num);
        int longest = 0;
        for (int num : set) {
            if (!set.contains(num - 1)) {
                int curr = num, streak = 1;
                while (set.contains(curr + 1)) {
                    curr++;
                    streak++;
                }
                longest = Math.max(longest, streak);
            }
        }
        return longest;
```

```
    }

    // ... (Implement problems 8 to 40 with similar structure: Javadoc + code + complexity)

    public static void main(String[] args) {
        // Simple test runs
        System.out.println("Two Sum: " + Arrays.toString(twoSum(new int[]{2,7,11,15}, 9)));
        System.out.println("Max Profit: " + maxProfit(new int[]{7,1,5,3,6,4}));
        System.out.println("Max Subarray: " + maxSubArray(new int[]{-2,1,-3,4,-1,2,1,-5,4}));
        System.out.println("Find Duplicate: " + findDuplicate(new int[]{1,3,4,2,2}));
        int[][] merged = mergeIntervals(new int[][]{{1,3},{2,6},{8,10},{15,18}});
        for (int[] inter : merged) System.out.println(Arrays.toString(inter));
        System.out.println("Product Except Self: " + Arrays.toString(productExceptSelf(new int[]{1,2,3,4})));
        System.out.println("Longest Consecutive: " + longestConsecutive(new int[]{100,4,200,1,3,2}));
    }
}


Books problems:
1.
 public static boolean arrayAdvance(List<Integer> A) {
    int furthestReachSoFar = 0;
    int lastIdx = A.size() - 1;
    for (int i = 0; i <= furthestReachSoFar && furthestReachSoFar < lastIdx; i++) {
      furthestReachSoFar = Math.max(furthestReachSoFar, i + A.get(i));
    }
    return furthestReachSoFar >= lastIdx;
  }

2.

public static double maxProfit(List<Double> prices) {
    double minPrice = Double.MAX_VALUE, maxProfit = 0.0;
    for (Double price : prices) {
        minPrice = Math.min(minPrice, price);
        maxProfit = Math.max(maxProfit, price - minPrice);
    }
    return maxProfit;
}

3.

static int maxProfitOptimized(int[] price) {
    int firstBuy = Integer.MIN_VALUE;
    int firstSell = 0;
    int secondBuy = Integer.MIN_VALUE;
    int secondSell = 0;

    for (int p : price) {
        firstBuy = Math.max(firstBuy, -p);
        firstSell = Math.max(firstSell, firstBuy + p);
        secondBuy = Math.max(secondBuy, firstSell - p);
        secondSell = Math.max(secondSell, secondBuy + p);
    }

    return secondSell;
}

4.
public static int deleteDuplicates(List<Integer> A) {
    int slow = 0;
    int fast = 0;
    int n = A.size();

```

```
    while (fast < n) {
      int val = A.get(fast);
      while (fast < n && A.get(fast) == val) {
        fast++;
      }
      A.set(slow++, val);
    }
    int curr = slow;
    while (curr < n) {
      A.set(curr++, null);
    }
    return slow;
  }

5.
 public static void dutchNationalFlag(int p, List<Integer> A) {
    int replacementIdx = 0;
    int pivotVal = A.get(p);
    // Travel from left to right and keep swapping with replacement index if less than
    pivot found
    for (int i = 0; i < A.size(); i++) {
      if (A.get(i) < pivotVal) {
        Collections.swap(A, i, replacementIdx);
        replacementIdx++;
      }
    }
    replacementIdx = A.size() - 1;
    // Travel from right to left and keep swapping with replacement index if greater
    than pivot found
    for (int i = A.size() - 1; i >= 0; i--) {
      if (A.get(i) > pivotVal) {
        Collections.swap(A, i, replacementIdx);
        replacementIdx--;
      }
    }
  }

6.
 public static List<Integer> nextPermutation(List<Integer> permutation) {
    int rightToLeft = permutation.size() - 1;
    boolean flag = false;
    while (rightToLeft > 0 && !flag) {
      if (permutation.get(rightToLeft) > permutation.get(rightToLeft - 1)) {
        int leftToRight = rightToLeft;
        int smallerNum = permutation.get(rightToLeft - 1);
        int smallestNumGreater = Integer.MAX_VALUE;
        int smallestNumIdx = -1;
        while (leftToRight < permutation.size()) {
          if (permutation.get(leftToRight) > smallerNum && smallestNumGreater >
          permutation
              .get(leftToRight)) {
            smallestNumGreater = permutation.get(leftToRight);
            smallestNumIdx = leftToRight;
          }
          leftToRight++;
        }
        if (smallestNumIdx == -1) {
          break;
        }
        permutation.set(rightToLeft - 1, smallestNumGreater);
        permutation.set(smallestNumIdx, smallerNum);
        List<Integer> temp = new ArrayList<>();
        int tempIdx = rightToLeft;
        while (tempIdx < permutation.size()) {
          temp.add(permutation.get(tempIdx++));
        }
        Collections.sort(temp);
        tempIdx = 0;
        int queryIdx = rightToLeft;
```

```
          while (queryIdx < permutation.size()) {
            permutation.set(queryIdx++, temp.get(tempIdx++));
          }
          flag = true;
        }
        rightToLeft--;
      }
      return flag ? permutation : Collections.emptyList();
    }

7.
 public static int findLongestSubarrayLength(int[] arr) {
        if (arr.length == 0) return 0;

        int maxLength = 1;
        int currentLength = 1;

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] == arr[i - 1]) {
                currentLength++;
                maxLength = Math.max(maxLength, currentLength);
            } else {
                currentLength = 1;
            }
        }
        return maxLength;
    }

8.

public static int minJumps(int[] arr) {
    if (arr.length <= 1) return 0;
    if (arr[0] == 0) return -1;

    int maxReach = arr[0];
    int steps = arr[0];
    int jumps = 1;

    for (int i = 1; i < arr.length; i++) {
        if (i == arr.length - 1) return jumps;

        maxReach = Math.max(maxReach, i + arr[i]);
        steps--;

                  jumps++;
            if (i >= maxReach) return -1;
            steps = maxReach - i;
        }
    }

    return -1;
}

9.Remove Duplicates from a Sorted List

public static int removeDuplicates(List<Integer> A) {
    if (A.isEmpty()) return 0;
    int write = 1;
    for (int i = 1; i < A.size(); i++)
        if (!A.get(i).equals(A.get(write - 1)))
            A.set(write++, A.get(i));
    return write;
}

10.

public static void leftRotate(int[] arr, int k) {
    int n = arr.length;
    k %= n;
```

```
        reverse(arr, 0, k - 1);
        reverse(arr, k, n - 1);
        reverse(arr, 0, n - 1);
    }


    // -------------------- Helpers --------------------
    private static void swap(int[] a, int i, int j) { int t=a[i]; a[i]=a[j]; a[j]=t; }
    private static void reverse(int[] a, int l, int r) { while (l<r) swap(a, l++, r--); }

    // -------------------- Main: quick tests --------------------
    public static void main(String[] args) {
        System.out.println("1 Two Sum: " + Arrays.toString(twoSum(new int[]{2,7,11,15}, 9
        )));
        System.out.println("2 3Sum: " + threeSum(new int[]{-1,0,1,2,-1,-4}));
        System.out.println("3 4Sum: " + fourSum(new int[]{1,0,-1,0,-2,2}, 0));
        System.out.println("4 Longest Consecutive: " + longestConsecutive(new int[]{100,4
        ,200,1,3,2}));
        System.out.println("5 Majority Element: " + majorityElement(new int[]{3,2,3}));
        System.out.println("6 Majority II: " + majorityElementII(new int[]{1,1,1,3,3,2,2,
        2}));
        int[] p={1,2,3}; nextPermutation(p); System.out.println("7 Next Permutation: " +
        Arrays.toString(p));
        int[] r = {1,2,3,4,5}; rotate(r, 2); System.out.println("8 Rotate: " + Arrays.
        toString(r));
        System.out.println("9 Product Except Self: " + Arrays.toString(productExceptSelf(
        new int[]{1,2,3,4})));
        System.out.println("10 Max Subarray: " + maxSubArray(new int[]{-2,1,-3,4,-1,2,1,-
        5,4}));
        System.out.println("11 Max Product Subarray: " + maxProduct(new int[]{2,3,-2,4
        }));
        System.out.println("12 Find Min Rotated: " + findMinRotated(new int[]{3,4,5,1,2
        }));
        System.out.println("13 Search Rotated: " + searchRotated(new int[]{4,5,6,7,0,1,2
        }, 0));
        System.out.println("14 Find Duplicate: " + findDuplicate(new int[]{1,3,4,2,2}));
        int[][] mat = {{1,1,1},{1,0,1},{1,1,1}}; setZeroes(mat); System.out.println("15
        Set Matrix Zeroes: " + Arrays.deepToString(mat));
        int[][] spiral = {{1,2,3},{4,5,6},{7,8,9}}; System.out.println("16 Spiral
        Matrix: " + spiralOrder(spiral));
        System.out.println("17 Merge Intervals: " + Arrays.deepToString(mergeIntervals(
        new int[][]{{1,3},{2,6},{8,10},{15,18}})));
        System.out.println("18 Insert Interval: " + Arrays.deepToString(insertInterval(
        new int[][]{{1,3},{6,9}}, new int[]{2,5})));
        int[][] A = {{0,2},{5,10},{13,23},{24,25}}; int[][] B = {{1,5},{8,12},{15,24},{25
        ,26}}; System.out.println("19 Interval Intersection: " + Arrays.deepToString(
        intervalIntersection(A,B)));
        System.out.println("20 Subarray Sum Equals K: " + subarraySum(new int[]{1,1,1}, 2
        ));
        System.out.println("21 Min Subarray Len: " + minSubArrayLen(7, new int[]{2,3,1,2,
        4,3}));
        System.out.println("22 Longest Substring Without Repeating: " +
        lengthOfLongestSubstring("abcabcbb"));
        System.out.println("23 Container With Most Water: " + maxArea(new int[]{1,8,6,2,5
        ,4,8,3,7}));
        System.out.println("24 Trapping Rain Water: " + trap(new int[]{0,1,0,2,1,0,1,3,2,
        1,2,1}));
        System.out.println("25 Jump Game: " + canJump(new int[]{2,3,1,1,4}));
        System.out.println("26 Jump Game II: " + jump(new int[]{2,3,1,1,4}));
        int[] colors = {2,0,2,1,1,0}; sortColors(colors); System.out.println("27 Sort
        Colors: " + Arrays.toString(colors));
        System.out.println("28 Search Range: " + Arrays.toString(searchRange(new int[]{5,
        7,7,8,8,10}, 8)));
        System.out.println("29 Find Peak Element: " + findPeakElement(new int[]{1,2,1,3,5
        ,6,4}));
        System.out.println("30 Min Swaps to Group All 1's: " + minSwaps(new int[]{1,0,1,0
        ,1}));
    }
}
```