

```

1
2 /**
3  * Top30ArrayProblems.java
4  *
5  * Implementations for Top 30 Medium Array Interview Problems (static methods).
6  * Each method includes a short comment describing the approach and complexity.
7  *
8  * Compile & run:
9  *   javac Top30ArrayProblems.java && java Top30ArrayProblems
10 *
11 * Author: ChatGPT
12 */
13
14 import java.util.*;
15
16 public class Top30ArrayProblems {
17
18     // 1. Two Sum (HashMap)
19     public static int[] twoSum(int[] nums, int target) {
20         Map<Integer,Integer> map = new HashMap<>();
21         for (int i=0;i<nums.length;i++) {
22             int need = target - nums[i];
23             if (map.containsKey(need)) return new int[]{map.get(need), i};
24             map.put(nums[i], i);
25         }
26         return new int[0];
27     }
28
29     // 2. 3Sum (sort + two pointers)
30     public static List<List<Integer>> threeSum(int[] nums) {
31         List<List<Integer>> res = new ArrayList<>();
32         Arrays.sort(nums);
33         for (int i=0;i<nums.length-2;i++) {
34             if (i>0 && nums[i]==nums[i-1]) continue;
35             int l=i+1, r=nums.length-1;
36             while (l<r) {
37                 int s = nums[i] + nums[l] + nums[r];
38                 if (s==0) {
39                     res.add(Arrays.asList(nums[i], nums[l], nums[r]));
40                     while (l<r && nums[l]==nums[l+1]) l++;
41                     while (l<r && nums[r]==nums[r-1]) r--;
42                     l++; r--;
43                 } else if (s<0) l++; else r--;
44             }
45         }
46         return res;
47     }
48
49     // 3. 4Sum (generalize k-sum via recursion)
50     public static List<List<Integer>> fourSum(int[] nums, int target) {
51         Arrays.sort(nums);
52         return kSum(nums, target, 4, 0);
53     }
54     private static List<List<Integer>> kSum(int[] nums, int target, int k, int start) {
55         List<List<Integer>> res = new ArrayList<>();
56         if (start >= nums.length) return res;
57         if (k == 2) {
58             int l = start, r = nums.length - 1;
59             while (l < r) {
60                 int sum = nums[l] + nums[r];
61                 if (sum == target) {
62                     res.add(Arrays.asList(nums[l], nums[r]));
63                     while (l<r && nums[l]==nums[l+1]) l++;
64                     while (l<r && nums[r]==nums[r-1]) r--;
65                     l++; r--;
66                 } else if (sum < target) l++; else r--;
67             }
68         } else {
69             for (int i = start; i < nums.length - (k - 1); i++) {

```

```

70         if (i > start && nums[i] == nums[i-1]) continue;
71         for (List<Integer> subset : kSum(nums, target - nums[i], k-1, i+1)) {
72             List<Integer> combined = new ArrayList<>(); combined.add(nums[i]);
73             combined.addAll(subset);
74             res.add(combined);
75         }
76     }
77     return res;
78 }
79
80 // 4. Longest Consecutive Sequence (HashSet)
81 public static int longestConsecutive(int[] nums) {
82     Set<Integer> set = new HashSet<>();
83     for (int n: nums) set.add(n);
84     int best = 0;
85     for (int n: set) {
86         if (!set.contains(n-1)) {
87             int cur = n, len = 1;
88             while (set.contains(cur+1)) { cur++; len++; }
89             best = Math.max(best, len);
90         }
91     }
92     return best;
93 }
94
95 // 5. Majority Element (Boyer-Moore)
96 public static int majorityElement(int[] nums) {
97     int cand = 0, cnt = 0;
98     for (int n: nums) {
99         if (cnt==0) { cand=n; cnt=1; }
100         else if (n==cand) cnt++; else cnt--;
101     }
102     return cand;
103 }
104
105 // 6. Majority Element II (Boyer-Moore extended)
106 public static List<Integer> majorityElementII(int[] nums) {
107     int cand1=0, cand2=0, c1=0, c2=0;
108     for (int n: nums) {
109         if (n==cand1) c1++;
110         else if (n==cand2) c2++;
111         else if (c1==0) { cand1=n; c1=1; }
112         else if (c2==0) { cand2=n; c2=1; }
113         else { c1--; c2--; }
114     }
115     List<Integer> res = new ArrayList<>();
116     int cnt1=0, cnt2=0;
117     for (int n: nums) { if (n==cand1) cnt1++; else if (n==cand2) cnt2++; }
118     if (cnt1 > nums.length/3) res.add(cand1);
119     if (cnt2 > nums.length/3) res.add(cand2);
120     return res;
121 }
122
123 // 7. Next Permutation
124 public static void nextPermutation(int[] nums) {
125     int i = nums.length - 2;
126     while (i>=0 && nums[i] >= nums[i+1]) i--;
127     if (i>=0) {
128         int j = nums.length - 1;
129         while (nums[j] <= nums[i]) j--;
130         swap(nums, i, j);
131     }
132     reverse(nums, i+1, nums.length-1);
133 }
134
135 // 8. Rotate Array (reverse method)
136 public static void rotate(int[] nums, int k) {
137     int n = nums.length; k %= n; if (k<0) k+=n;

```

```

138         reverse(nums, 0, n-1); reverse(nums, 0, k-1); reverse(nums, k, n-1);
139     }
140
141     // 9. Product of Array Except Self (prefix/suffix)
142     public static int[] productExceptSelf(int[] nums) {
143         int n = nums.length;
144         int[] res = new int[n];
145         int prefix = 1;
146         for (int i=0;i<n;i++) { res[i] = prefix; prefix *= nums[i]; }
147         int suffix = 1;
148         for (int i=n-1;i>=0;i--) { res[i] *= suffix; suffix *= nums[i]; }
149         return res;
150     }
151
152     // 10. Maximum Subarray Sum (Kadane)
153     public static int maxSubArray(int[] nums) {
154         int max = nums[0], cur = nums[0];
155         for (int i=1;i<nums.length;i++) {
156             cur = Math.max(nums[i], cur + nums[i]);
157             max = Math.max(max, cur);
158         }
159         return max;
160     }
161
162     // 11. Maximum Product Subarray
163     public static int maxProduct(int[] nums) {
164         int max = nums[0], min = nums[0], res = nums[0];
165         for (int i=1;i<nums.length;i++) {
166             int a = nums[i];
167             if (a < 0) { int tmp=max; max=min; min=tmp; }
168             max = Math.max(a, max * a);
169             min = Math.min(a, min * a);
170             res = Math.max(res, max);
171         }
172         return res;
173     }
174
175     // 12. Find Minimum in Rotated Sorted Array
176     public static int findMinRotated(int[] nums) {
177         int l=0, r=nums.length-1;
178         while (l<r) {
179             int m = l + (r-l)/2;
180             if (nums[m] > nums[r]) l = m+1;
181             else r = m;
182         }
183         return nums[l];
184     }
185
186     // 13. Search in Rotated Sorted Array
187     public static int searchRotated(int[] nums, int target) {
188         int l=0, r=nums.length-1;
189         while (l<=r) {
190             int m = (l+r)/2;
191             if (nums[m]==target) return m;
192             if (nums[l] <= nums[m]) {
193                 if (nums[l] <= target && target < nums[m]) r = m-1;
194                 else l = m+1;
195             } else {
196                 if (nums[m] < target && target <= nums[r]) l = m+1;
197                 else r = m-1;
198             }
199         }
200         return -1;
201     }
202
203     // 14. Find the Duplicate Number (Floyd's cycle detection)
204     public static int findDuplicate(int[] nums) {
205         int slow = nums[0], fast = nums[0];
206         do {

```

```

207         slow = nums[slow];
208         fast = nums[nums[fast]];
209     } while (slow != fast);
210     slow = nums[0];
211     while (slow != fast) { slow = nums[slow]; fast = nums[fast]; }
212     return slow;
213 }
214
215 // 15. Set Matrix Zeroes (use first row/col as markers)
216 public static void setZeroes(int[][] matrix) {
217     boolean row0 = false, col0 = false;
218     int m = matrix.length, n = matrix[0].length;
219     for (int j=0;j<n;j++) if (matrix[0][j]==0) row0=true;
220     for (int i=0;i<m;i++) if (matrix[i][0]==0) col0=true;
221     for (int i=1;i<m;i++) {
222         for (int j=1;j<n;j++) if (matrix[i][j]==0) { matrix[i][0]=0; matrix[0][j]=0; }
223     }
224     for (int i=1;i<m;i++) if (matrix[i][0]==0) Arrays.fill(matrix[i], 0);
225     for (int j=1;j<n;j++) if (matrix[0][j]==0) for (int i=0;i<m;i++) matrix[i][j]=0;
226     if (row0) Arrays.fill(matrix[0], 0);
227     if (col0) for (int i=0;i<m;i++) matrix[i][0]=0;
228 }
229
230 // 16. Spiral Matrix
231 public static List<Integer> spiralOrder(int[][] matrix) {
232     List<Integer> res = new ArrayList<>();
233     if (matrix==null || matrix.length==0) return res;
234     int top=0, bottom=matrix.length-1, left=0, right=matrix[0].length-1;
235     while (top<=bottom && left<=right) {
236         for (int j=left;j<=right;j++) res.add(matrix[top][j]);
237         top++;
238         for (int i=top;i<=bottom;i++) res.add(matrix[i][right]);
239         right--;
240         if (top<=bottom) for (int j=right;j>=left;j--) res.add(matrix[bottom][j]);
241         bottom--;
242         if (left<=right) for (int i=bottom;i>=top;i--) res.add(matrix[i][left]);
243         left++;
244     }
245     return res;
246 }
247
248 // 17. Merge Intervals
249 public static int[][] mergeIntervals(int[][] intervals) {
250     if (intervals==null || intervals.length==0) return new int[0][];
251     Arrays.sort(intervals, (a,b)->Integer.compare(a[0], b[0]));
252     List<int[]> out = new ArrayList<>();
253     for (int[] it: intervals) {
254         if (out.isEmpty() || out.get(out.size()-1)[1] < it[0]) out.add(it);
255         else out.get(out.size()-1)[1] = Math.max(out.get(out.size()-1)[1], it[1]);
256     }
257     return out.toArray(new int[out.size()][]);
258 }
259
260 // 18. Insert Interval
261 public static int[][] insertInterval(int[][] intervals, int[] newInterval) {
262     List<int[]> res = new ArrayList<>();
263     int i=0, n = intervals.length;
264     while (i<n && intervals[i][1] < newInterval[0]) res.add(intervals[i++]);
265     while (i<n && intervals[i][0] <= newInterval[1]) {
266         newInterval[0] = Math.min(newInterval[0], intervals[i][0]);
267         newInterval[1] = Math.max(newInterval[1], intervals[i][1]);
268         i++;
269     }
270     res.add(newInterval);
271     while (i<n) res.add(intervals[i++]);
272     return res.toArray(new int[res.size()][]);
273 }
274

```

```

275 // 19. Interval List Intersections
276 public static int[][] intervalIntersection(int[][] A, int[][] B) {
277     List<int[]> res = new ArrayList<>();
278     int i=0,j=0;
279     while (i<A.length && j<B.length) {
280         int lo = Math.max(A[i][0], B[j][0]);
281         int hi = Math.min(A[i][1], B[j][1]);
282         if (lo <= hi) res.add(new int[]{lo,hi});
283         if (A[i][1] < B[j][1]) i++; else j++;
284     }
285     return res.toArray(new int[res.size()][2]);
286 }
287
288 // 20. Subarray Sum Equals K (prefix sum hashmap)
289 public static int subarraySum(int[] nums, int k) {
290     Map<Integer,Integer> count = new HashMap<>();
291     count.put(0,1);
292     int sum=0, res=0;
293     for (int x: nums) {
294         sum += x;
295         res += count.getOrDefault(sum - k, 0);
296         count.put(sum, count.getOrDefault(sum,0)+1);
297     }
298     return res;
299 }
300
301 // 21. Minimum Size Subarray Sum (sliding window)
302 public static int minSubArrayLen(int s, int[] nums) {
303     int left=0, sum=0, res=Integer.MAX_VALUE;
304     for (int i=0;i<nums.length;i++) {
305         sum += nums[i];
306         while (sum >= s) {
307             res = Math.min(res, i-left+1);
308             sum -= nums[left++];
309         }
310     }
311     return (res==Integer.MAX_VALUE)?0:res;
312 }
313
314 // 22. Longest Substring Without Repeating Characters (sliding window)
315 public static int lengthOfLongestSubstring(String s) {
316     Map<Character,Integer> last = new HashMap<>();
317     int left=0, res=0;
318     for (int i=0;i<s.length();i++) {
319         char c = s.charAt(i);
320         if (last.containsKey(c)) left = Math.max(left, last.get(c)+1);
321         last.put(c, i);
322         res = Math.max(res, i-left+1);
323     }
324     return res;
325 }
326
327 // 23. Container With Most Water (two pointers)
328 public static int maxArea(int[] height) {
329     int l=0, r=height.length-1, ans=0;
330     while (l<r) {
331         ans = Math.max(ans, Math.min(height[l], height[r]) * (r-l));
332         if (height[l] < height[r]) l++; else r--;
333     }
334     return ans;
335 }
336
337 // 24. Trapping Rain Water (two pointers)
338 public static int trap(int[] height) {
339     int l=0, r=height.length-1;
340     int leftMax=0, rightMax=0, trapped=0;
341     while (l<r) {
342         if (height[l] < height[r]) {
343             if (height[l] >= leftMax) leftMax = height[l];

```

```

344         else trapped += leftMax - height[l];
345         l++;
346     } else {
347         if (height[r] >= rightMax) rightMax = height[r];
348         else trapped += rightMax - height[r];
349         r--;
350     }
351 }
352 return trapped;
353 }
354
355 // 25. Jump Game (greedy max reach)
356 public static boolean canJump(int[] nums) {
357     int reach = 0;
358     for (int i=0;i<nums.length;i++) {
359         if (i > reach) return false;
360         reach = Math.max(reach, i + nums[i]);
361     }
362     return true;
363 }
364
365 // 26. Jump Game II (greedy BFS-like)
366 public static int jump(int[] nums) {
367     int jumps=0, curEnd=0, curFarthest=0;
368     for (int i=0;i<nums.length-1;i++) {
369         curFarthest = Math.max(curFarthest, i + nums[i]);
370         if (i == curEnd) {
371             jumps++;
372             curEnd = curFarthest;
373         }
374     }
375     return jumps;
376 }
377
378 // 27. Sort Colors (Dutch National Flag)
379 public static void sortColors(int[] nums) {
380     int low=0, mid=0, high=nums.length-1;
381     while (mid <= high) {
382         if (nums[mid]==0) swap(nums, low++, mid++);
383         else if (nums[mid]==1) mid++;
384         else swap(nums, mid, high--);
385     }
386 }
387
388 // 28. Find First and Last Position of Element in Sorted Array
389 public static int[] searchRange(int[] nums, int target) {
390     int left = findBound(nums, target, true);
391     int right = findBound(nums, target, false);
392     return new int[]{left, right};
393 }
394 private static int findBound(int[] nums, int target, boolean left) {
395     int l=0, r=nums.length-1, bound=-1;
396     while (l<=r) {
397         int m = (l+r)/2;
398         if (nums[m]==target) { bound=m; if (left) r=m-1; else l=m+1; }
399         else if (nums[m] < target) l = m+1; else r = m-1;
400     }
401     return (bound==-1)? -1 : bound;
402 }
403
404 // 29. Find Peak Element
405 public static int findPeakElement(int[] nums) {
406     int l=0, r=nums.length-1;
407     while (l<r) {
408         int m=(l+r)/2;
409         if (nums[m] > nums[m+1]) r = m;
410         else l = m+1;
411     }
412     return l;

```

```

413     }
414
415     // 30. Minimum Swaps to Group All 1's Together (sliding window count)
416     public static int minSwaps(int[] nums) {
417         int k=0;
418         for (int n: nums) if (n==1) k++;
419         if (k==0) return 0;
420         int bad=0;
421         for (int i=0;i<k;i++) if (nums[i]==0) bad++;
422         int ans = bad;
423         for (int i=k;i<nums.length;i++) {
424             if (nums[i-k]==0) bad--;
425             if (nums[i]==0) bad++;
426             ans = Math.min(ans, bad);
427         }
428         return ans;
429     }
430
431     /**
432     * 1. Two Sum
433     * Find indices of the two numbers that add up to target.
434     * Time: O(n), Space: O(n)
435     */
436     public static int[] twoSum(int[] nums, int target) {
437         Map<Integer, Integer> map = new HashMap<>();
438         for (int i = 0; i < nums.length; i++) {
439             int complement = target - nums[i];
440             if (map.containsKey(complement)) {
441                 return new int[]{map.get(complement), i};
442             }
443             map.put(nums[i], i);
444         }
445         return new int[]{};
446     }
447
448     /**
449     * 2. Best Time to Buy and Sell Stock (Single Transaction)
450     * Time: O(n), Space: O(1)
451     */
452     public static int maxProfit(int[] prices) {
453         int minPrice = Integer.MAX_VALUE, maxProfit = 0;
454         for (int price : prices) {
455             minPrice = Math.min(minPrice, price);
456             maxProfit = Math.max(maxProfit, price - minPrice);
457         }
458         return maxProfit;
459     }
460
461     /**
462     * 3. Maximum Subarray (Kadane's Algorithm)
463     * Time: O(n), Space: O(1)
464     */
465     public static int maxSubArray(int[] nums) {
466         int maxSoFar = nums[0], curr = nums[0];
467         for (int i = 1; i < nums.length; i++) {
468             curr = Math.max(nums[i], curr + nums[i]);
469             maxSoFar = Math.max(maxSoFar, curr);
470         }
471         return maxSoFar;
472     }
473
474     /**
475     * 4. Find Duplicate Number
476     * Using Floyd's cycle detection.
477     * Time: O(n), Space: O(1)
478     */
479     public static int findDuplicate(int[] nums) {
480         int slow = nums[0], fast = nums[0];
481         do {

```

```

482         slow = nums[slow];
483         fast = nums[nums[fast]];
484     } while (slow != fast);
485     slow = nums[0];
486     while (slow != fast) {
487         slow = nums[slow];
488         fast = nums[fast];
489     }
490     return slow;
491 }
492
493 /**
494  * 5. Merge Intervals
495  * Time: O(n log n), Space: O(n)
496  */
497 public static int[][] mergeIntervals(int[][] intervals) {
498     Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
499     List<int[]> merged = new ArrayList<>();
500     for (int[] interval : intervals) {
501         if (merged.isEmpty() || merged.get(merged.size() - 1)[1] < interval[0]) {
502             merged.add(interval);
503         } else {
504             merged.get(merged.size() - 1)[1] = Math.max(merged.get(merged.size() - 1)
505                 [1], interval[1]);
506         }
507     }
508     return merged.toArray(new int[merged.size()][]);
509 }
510
511 /**
512  * 6. Product of Array Except Self
513  * Time: O(n), Space: O(1) extra
514  */
515 public static int[] productExceptSelf(int[] nums) {
516     int n = nums.length;
517     int[] res = new int[n];
518     Arrays.fill(res, 1);
519     int prefix = 1;
520     for (int i = 0; i < n; i++) {
521         res[i] = prefix;
522         prefix *= nums[i];
523     }
524     int suffix = 1;
525     for (int i = n - 1; i >= 0; i--) {
526         res[i] *= suffix;
527         suffix *= nums[i];
528     }
529     return res;
530 }
531
532 /**
533  * 7. Longest Consecutive Sequence
534  * Time: O(n), Space: O(n)
535  */
536 public static int longestConsecutive(int[] nums) {
537     Set<Integer> set = new HashSet<>();
538     for (int num : nums) set.add(num);
539     int longest = 0;
540     for (int num : set) {
541         if (!set.contains(num - 1)) {
542             int curr = num, streak = 1;
543             while (set.contains(curr + 1)) {
544                 curr++;
545                 streak++;
546             }
547             longest = Math.max(longest, streak);
548         }
549     }
550     return longest;

```



```

550     }
551
552     // ... (Implement problems 8 to 40 with similar structure: Javadoc + code +
    complexity)
553
554     public static void main(String[] args) {
555         // Simple test runs
556         System.out.println("Two Sum: " + Arrays.toString(twoSum(new int[]{2,7,11,15}, 9
    ))) );
557         System.out.println("Max Profit: " + maxProfit(new int[]{7,1,5,3,6,4}));
558         System.out.println("Max Subarray: " + maxSubArray(new int[]{-2,1,-3,4,-1,2,1,-5,4
    }));
559         System.out.println("Find Duplicate: " + findDuplicate(new int[]{1,3,4,2,2}));
560         int[][] merged = mergeIntervals(new int[][]{{1,3},{2,6},{8,10},{15,18}});
561         for (int[] inter : merged) System.out.println(Arrays.toString(inter));
562         System.out.println("Product Except Self: " + Arrays.toString(productExceptSelf(
    new int[]{1,2,3,4})));
563         System.out.println("Longest Consecutive: " + longestConsecutive(new int[]{100,4,
    200,1,3,2}));
564     }
565 }
566
567
568 Books problems:
569 1.
570     public static boolean arrayAdvance(List<Integer> A) {
571         int furthestReachSoFar = 0;
572         int lastIdx = A.size() - 1;
573         for (int i = 0; i <= furthestReachSoFar && furthestReachSoFar < lastIdx; i++) {
574             furthestReachSoFar = Math.max(furthestReachSoFar, i + A.get(i));
575         }
576         return furthestReachSoFar >= lastIdx;
577     }
578
579 2.
580
581     public static double maxProfit(List<Double> prices) {
582         double minPrice = Double.MAX_VALUE, maxProfit = 0.0;
583         for (Double price : prices) {
584             minPrice = Math.min(minPrice, price);
585             maxProfit = Math.max(maxProfit, price - minPrice);
586         }
587         return maxProfit;
588     }
589
590 3.
591
592     static int maxProfitOptimized(int[] price) {
593         int firstBuy = Integer.MIN_VALUE;
594         int firstSell = 0;
595         int secondBuy = Integer.MIN_VALUE;
596         int secondSell = 0;
597
598         for (int p : price) {
599             firstBuy = Math.max(firstBuy, -p);
600             firstSell = Math.max(firstSell, firstBuy + p);
601             secondBuy = Math.max(secondBuy, firstSell - p);
602             secondSell = Math.max(secondSell, secondBuy + p);
603         }
604
605         return secondSell;
606     }
607
608 4.
609     public static int deleteDuplicates(List<Integer> A) {
610         int slow = 0;
611         int fast = 0;
612         int n = A.size();
613

```

```

614 while (fast < n) {
615     int val = A.get(fast);
616     while (fast < n && A.get(fast) == val) {
617         fast++;
618     }
619     A.set(slow++, val);
620 }
621 int curr = slow;
622 while (curr < n) {
623     A.set(curr++, null);
624 }
625 return slow;
626 }
627
628 5.
629 public static void dutchNationalFlag(int p, List<Integer> A) {
630     int replacementIdx = 0;
631     int pivotVal = A.get(p);
632     // Travel from left to right and keep swapping with replacement index if less than
633     // pivot found
634     for (int i = 0; i < A.size(); i++) {
635         if (A.get(i) < pivotVal) {
636             Collections.swap(A, i, replacementIdx);
637             replacementIdx++;
638         }
639     }
640     replacementIdx = A.size() - 1;
641     // Travel from right to left and keep swapping with replacement index if greater
642     // than pivot found
643     for (int i = A.size() - 1; i >= 0; i--) {
644         if (A.get(i) > pivotVal) {
645             Collections.swap(A, i, replacementIdx);
646             replacementIdx--;
647         }
648     }
649 }
650
651 6.
652 public static List<Integer> nextPermutation(List<Integer> permutation) {
653     int rightToLeft = permutation.size() - 1;
654     boolean flag = false;
655     while (rightToLeft > 0 && !flag) {
656         if (permutation.get(rightToLeft) > permutation.get(rightToLeft - 1)) {
657             int leftToRight = rightToLeft;
658             int smallerNum = permutation.get(rightToLeft - 1);
659             int smallestNumGreater = Integer.MAX_VALUE;
660             int smallestNumIdx = -1;
661             while (leftToRight < permutation.size()) {
662                 if (permutation.get(leftToRight) > smallerNum && smallestNumGreater >
663                     permutation
664                         .get(leftToRight)) {
665                     smallestNumGreater = permutation.get(leftToRight);
666                     smallestNumIdx = leftToRight;
667                 }
668                 leftToRight++;
669             }
670             if (smallestNumIdx == -1) {
671                 break;
672             }
673             permutation.set(rightToLeft - 1, smallestNumGreater);
674             permutation.set(smallestNumIdx, smallerNum);
675             List<Integer> temp = new ArrayList<>();
676             int tempIdx = rightToLeft;
677             while (tempIdx < permutation.size()) {
678                 temp.add(permutation.get(tempIdx++));
679             }
680             Collections.sort(temp);
681             tempIdx = 0;
682             int queryIdx = rightToLeft;

```

```

680         while (queryIdx < permutation.size()) {
681             permutation.set(queryIdx++, temp.get(tempIdx++));
682         }
683         flag = true;
684     }
685     rightToLeft--;
686 }
687 return flag ? permutation : Collections.emptyList();
688 }
689
690 7.
691 public static int findLongestSubarrayLength(int[] arr) {
692     if (arr.length == 0) return 0;
693
694     int maxLength = 1;
695     int currentLength = 1;
696
697     for (int i = 1; i < arr.length; i++) {
698         if (arr[i] == arr[i - 1]) {
699             currentLength++;
700             maxLength = Math.max(maxLength, currentLength);
701         } else {
702             currentLength = 1;
703         }
704     }
705     return maxLength;
706 }
707
708 8.
709
710 public static int minJumps(int[] arr) {
711     if (arr.length <= 1) return 0;
712     if (arr[0] == 0) return -1;
713
714     int maxReach = arr[0];
715     int steps = arr[0];
716     int jumps = 1;
717
718     for (int i = 1; i < arr.length; i++) {
719         if (i == arr.length - 1) return jumps;
720
721         maxReach = Math.max(maxReach, i + arr[i]);
722         steps--;
723
724         jumps++;
725         if (i >= maxReach) return -1;
726         steps = maxReach - i;
727     }
728
729     return -1;
730 }
731
732
733 9.Remove Duplicates from a Sorted List
734
735 public static int removeDuplicates(List<Integer> A) {
736     if (A.isEmpty()) return 0;
737     int write = 1;
738     for (int i = 1; i < A.size(); i++)
739         if (!A.get(i).equals(A.get(write - 1)))
740             A.set(write++, A.get(i));
741     return write;
742 }
743
744 10.
745
746 public static void leftRotate(int[] arr, int k) {
747     int n = arr.length;
748     k %= n;

```

```

749     reverse(arr, 0, k - 1);
750     reverse(arr, k, n - 1);
751     reverse(arr, 0, n - 1);
752 }
753
754
755 // ----- Helpers -----
756 private static void swap(int[] a, int i, int j) { int t=a[i]; a[i]=a[j]; a[j]=t; }
757 private static void reverse(int[] a, int l, int r) { while (l<r) swap(a, l++, r--); }
758
759 // ----- Main: quick tests -----
760 public static void main(String[] args) {
761     System.out.println("1 Two Sum: " + Arrays.toString(twoSum(new int[]{2,7,11,15}, 9
762     ));
763     System.out.println("2 3Sum: " + threeSum(new int[]{-1,0,1,2,-1,-4}));
764     System.out.println("3 4Sum: " + fourSum(new int[]{1,0,-1,0,-2,2}, 0));
765     System.out.println("4 Longest Consecutive: " + longestConsecutive(new int[]{100,4
766     ,200,1,3,2}));
767     System.out.println("5 Majority Element: " + majorityElement(new int[]{3,2,3}));
768     System.out.println("6 Majority II: " + majorityElementII(new int[]{1,1,1,3,3,2,2,
769     2}));
770     int[] p={1,2,3}; nextPermutation(p); System.out.println("7 Next Permutation: " +
771     Arrays.toString(p));
772     int[] r = {1,2,3,4,5}; rotate(r, 2); System.out.println("8 Rotate: " + Arrays.
773     toString(r));
774     System.out.println("9 Product Except Self: " + Arrays.toString(productExceptSelf(
775     new int[]{1,2,3,4}));
776     System.out.println("10 Max Subarray: " + maxSubArray(new int[]{-2,1,-3,4,-1,2,1,-
777     5,4}));
778     System.out.println("11 Max Product Subarray: " + maxProduct(new int[]{2,3,-2,4
779     }));
780     System.out.println("12 Find Min Rotated: " + findMinRotated(new int[]{3,4,5,1,2
781     }));
782     System.out.println("13 Search Rotated: " + searchRotated(new int[]{4,5,6,7,0,1,2
783     }, 0));
784     System.out.println("14 Find Duplicate: " + findDuplicate(new int[]{1,3,4,2,2}));
785     int[][] mat = {{1,1,1},{1,0,1},{1,1,1}}; setZeroes(mat); System.out.println("15
786     Set Matrix Zeroes: " + Arrays.deepToString(mat));
787     int[][] spiral = {{1,2,3},{4,5,6},{7,8,9}}; System.out.println("16 Spiral
788     Matrix: " + spiralOrder(spiral));
789     System.out.println("17 Merge Intervals: " + Arrays.deepToString(mergeIntervals(
790     new int[][]{{1,3},{2,6},{8,10},{15,18}}));
791     System.out.println("18 Insert Interval: " + Arrays.deepToString(insertInterval(
792     new int[][]{{1,3},{6,9}}, new int[]{2,5}));
793     int[][] A = {{0,2},{5,10},{13,23},{24,25}}; int[][] B = {{1,5},{8,12},{15,24},{25
794     ,26}}; System.out.println("19 Interval Intersection: " + Arrays.deepToString(
795     intervalIntersection(A,B));
796     System.out.println("20 Subarray Sum Equals K: " + subarraySum(new int[]{1,1,1}, 2
797     ));
798     System.out.println("21 Min Subarray Len: " + minSubArrayLen(7, new int[]{2,3,1,2,
799     4,3}));
800     System.out.println("22 Longest Substring Without Repeating: " +
801     lengthOfLongestSubstring("abcabcbb"));
802     System.out.println("23 Container With Most Water: " + maxArea(new int[]{1,8,6,2,5
803     ,4,8,3,7}));
804     System.out.println("24 Trapping Rain Water: " + trap(new int[]{0,1,0,2,1,0,1,3,2,
805     1,2,1}));
806     System.out.println("25 Jump Game: " + canJump(new int[]{2,3,1,1,4}));
807     System.out.println("26 Jump Game II: " + jump(new int[]{2,3,1,1,4}));
808     int[] colors = {2,0,2,1,1,0}; sortColors(colors); System.out.println("27 Sort
809     Colors: " + Arrays.toString(colors));
810     System.out.println("28 Search Range: " + Arrays.toString(searchRange(new int[]{5,
811     7,7,8,8,10}, 8));
812     System.out.println("29 Find Peak Element: " + findPeakElement(new int[]{1,2,1,3,5
813     ,6,4}));
814     System.out.println("30 Min Swaps to Group All 1's: " + minSwaps(new int[]{1,0,1,0
815     ,1}));
816 }
817 }

```