

Solutions to Leetcode

Aayush Bajaj

2025-10-21T20:42:32+11:00

Contents

1	1. Two Sum	4
2	2. Add Two Numbers	4
3	3. Longest Substring Without Repeating Characters	5
4	4. Median of Two Sorted Arrays	5
5	11. Container With Most Water	6
6	12. Integer to Roman	6
7	15. 3Sum	7
8	16. 3Sum Closest	8
9	17. Letter Combinations of a Phone Number	9
10	20. Valid Parentheses	9
11	21. Merge Two Sorted Lists	9
12	36. Valid Sudoku	9
13	37. Sudoku Solver	10
14	41. First Missing Positive	11
15	42. Trapping Rain Water	12
16	45. Jump Game II	12
17	46. Permutations	12
18	49. Group Anagrams	12
19	53. Maximum Subarray	13
20	54. Spiral Matrix	13
21	56. Merge Intervals	14
22	70. Climbing Stairs	14
23	77. Combinations	14

24	78. Subsets	14
25	88. Merge Sorted Array	15
26	92. Reverse Linked List II	16
27	100. Same Tree	16
28	102. Binary Tree Level Order Traversal	16
29	104. Maximum Depth of Binary Tree	17
30	108. Convert Sorted Array to Binary Search Tree	17
31	111. Minimum Depth of Binary Tree	17
32	112. Path Sum	17
33	121. Best Time to Buy and Sell Stock	18
34	128. Longest Consecutive Sequence	18
35	133. Clone Graph	18
36	136. Single Number	19
37	141. Linked List Cycle	19
38	150. Evaluate Reverse Polish Notation	19
39	151. Reverse Words in a String	20
40	152. Maximum Product Subarray	20
41	153. Find Minimum in Rotated Sorted Array	20
42	155. Min Stack	20
43	167. Two Sum II - Input Array Is Sorted	20
44	169. Majority Element	20
45	200. Number of Islands	21
46	202. Happy Number	22
47	203. Remove Linked List Elements	22
48	206. Reverse Linked List	22
49	207. Course Schedule	23
50	209. Minimum Size Subarray Sum	23
51	215. Kth Largest Element in an Array	24
52	217. Contains Duplicate	24
53	219. Contains Duplicate II	24

54	225. Implement Stack using Queues	24
55	226. Invert Binary Tree	25
56	230. Kth Smallest Element in a BST	25
57	234. Palindrome Linked List	25
58	235. Lowest Common Ancestor of a Binary Search Tree	25
59	236. Lowest Common Ancestor of a Binary Tree	26
60	238. Product of Array Except Self	26
61	239. Sliding Window Maximum	27
62	260. Single Number III	28
63	268. Missing Number	28
64	303. Range Sum Query - Immutable	28
65	322. Coin Change	28
66	338. Counting Bits	29
67	347. Top K Frequent Elements	29
68	392. Is Subsequence	29
69	424. Longest Repeating Character Replacement	30
70	438. Find All Anagrams in a String	30
71	448. Find All Numbers Disappeared in an Array	31
72	450. Delete Node in a BST	31
73	530. Minimum Absolute Difference in BST	32
74	543. Diameter of Binary Tree	32
75	567. Permutation in String	32
76	621. Task Scheduler	33
77	637. Average of Levels in Binary Tree	33
78	647. Palindromic Substrings	34
79	653. Two Sum IV - Input is a BST	34
80	700. Search in a Binary Search Tree	34
81	701. Insert into a Binary Search Tree	35
82	784. Letter Case Permutation	35
83	787. Cheapest Flights Within K Stops	35

84	845. Longest Mountain in Array	36
85	876. Middle of the Linked List	36
86	973. K Closest Points to Origin	36
87	977. Squares of a Sorted Array	37
88	1200. Minimum Absolute Difference	37
89	1266. Minimum Time Visiting All Points	37
90	1365. How Many Numbers Are Smaller Than the Current Number	37
91	1366. Rank Teams by Voters	38
92	1382. Balance a Binary Search Tree	38
93	2034. Stock Price Fluctuation	39
94	2073. Time Needed to Buy Tickets	39
95	2933. High-Access Employee	40
96	3026. Maximum Good Subarray Sum	40

Here lie my implementations and notes for various [leetcode](#) problems.

1 1. Two Sum

```

1 class Solution:
2     def twoSum(self, nums: List[int], target: int) -> List[int]:
3         table = dict()
4         for i, num in enumerate(nums):
5             if target - num in table.keys():
6                 return [i, table[target-num]]
7             table[num] = i
8         return []

```

2 2. Add Two Numbers

```

1 # Definition for singly-linked list.
2 #class ListNode:
3 #    def __init__(self, val=0, next=None):
4 #        self.val = val
5 #        self.next = next
6 class Solution:
7     def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:
8         # get numbers out:
9         head = l1
10        l1_nums = [] # [2,4,3]
11        while head:
12            l1_nums.append(head.val)
13            head = head.next
14        print("l1_nums:", l1_nums)
15        head = l2
16        l2_nums = [] # [5,6,4]
17        while head:
18            l2_nums.append(head.val)
19            head = head.next

```

```

20     print("l2_nums:", l2_nums)
21     # reverse the lists and turn them into ints:
22     if l1_nums is not None:
23         l1_nums = list(reversed(l1_nums))
24     if l2_nums is not None:
25         l2_nums = list(reversed(l2_nums))
26     l1_num = int(''.join(map(str, l1_nums))) #342
27     l2_num = int(''.join(map(str, l2_nums))) #465
28     sum = l1_num + l2_num
29     sum_list = list(map(int, str(sum)))      # [8, 0, 7]
30     #if sum_list is not None:
31     #     sum_list = list(reversed(sum_list)) # [7, 0, 8]
32     print(sum_list)
33
34     # LN(7) <- LN(0) <- LN(8)
35
36     tail = None
37     for i in range(len(sum_list)):
38         next = ListNode(sum_list[i], tail)
39         i = i + 1
40         tail = next
41
42     return tail

```

3. Longest Substring Without Repeating Characters

```

1  class Solution:
2      def lengthOfLongestSubstring(self, s: str) -> int:
3          l = r = 0
4          longest_length = r - l
5          counts = collections.defaultdict(int)
6          while r < len(s):
7              if counts[s[r]] == 0:
8                  counts[s[r]] += 1
9                  r += 1
10             else:
11                 counts[s[l]] = 0
12                 l += 1
13             longest_length = max(r - l, longest_length)
14
15     return longest_length

```

4. Median of Two Sorted Arrays

```

1  double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int nums2Size){
2      int totalSize = nums1Size + nums2Size;
3      int merged[totalSize];
4      int i = 0, j = 0, k = 0;
5
6      while (i < nums1Size && j < nums2Size) {
7          if (nums1[i] < nums2[j]) {
8              merged[k++] = nums1[i++];
9          } else {
10             merged[k++] = nums2[j++];
11         }
12     }
13
14     while (i < nums1Size) {
15         merged[k++] = nums1[i++];
16     }
17
18     while (j < nums2Size) {

```

```

19     merged[k++] = nums2[j++];
20 }
21
22 if (totalSize % 2 == 0) {
23     return (double)(merged[totalSize/2] + merged[totalSize/2 - 1])/2;
24 } else {
25     return (double)merged[totalSize/2];
26 }
27 }
28

```

5 11. Container With Most Water

```

1 class Solution:
2     def maxArea(self, height: List[int]) -> int:
3         """
4         sorted_indices = sorted(range(len(height)),
5                                 key=lambda index: height[index], reverse=True)
6
7         maxArea = 0
8         print(sorted_indices)
9         for i, s in enumerate(sorted_indices):
10             for j, t in enumerate(sorted_indices):
11                 if j >= i:
12                     container_height = min(height[s], height[t])
13                     area = abs(s-t) * container_height
14                     if area > maxArea:
15                         maxArea = area
16
17         return maxArea
18
19         # Time Limit Exceeded | 54/65 testcases passed
20         """
21         # two pointer approach
22         maxArea = 0
23         leftPointer = 0
24         rightPointer = len(height)-1
25         while leftPointer != rightPointer:
26             area = min(height[leftPointer], height[rightPointer]) * abs(leftPointer - rightPointer)
27             if area > maxArea:
28                 maxArea = area
29             if height[leftPointer] < height[rightPointer]:
30                 leftPointer += 1
31             else:
32                 rightPointer -= 1
33
34         return maxArea

```

6 12. Integer to Roman

```

1 class Solution:
2     def intToRoman(self, num: int) -> str:
3         values = [1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000][::-1]
4         symbols = ["I", "IV", "V", "IX", "X", "XL", "L", "XC", "C", "CD", "D", "CM", "M"][::-1]
5
6         romans = list(zip(values, symbols))
7         num_as_str = str(num)
8         digits = len(num_as_str)
9
10        result = ""
11
12        for val, sym in romans:
13            multiplier = num // val

```

```

14         if multiplier:
15             result += sym * multiplier
16             num %= val
17
18     return result

```

7 15. 3Sum

```

1  #from itertools import combinations
2  """
3  class Solution:
4
5      def threeSum(self, nums: List[int]) -> List[List[int]]:
6          def twoSum(numbers: List[int], target: int) -> List[List[int]]:
7              res = []
8              if not numbers:
9                  return []
10             leftPointer, rightPointer = 0, len(numbers) - 1
11             while leftPointer < rightPointer:
12                 status = numbers[leftPointer] + numbers[rightPointer]
13                 if status < target:
14                     leftPointer += 1
15                 elif status > target:
16                     rightPointer -= 1
17                 else:
18                     res.append([leftPointer + 1, rightPointer + 1])
19
20             # skip duplicates on BOTH sides
21             lv, rv = numbers[leftPointer], numbers[rightPointer]
22             while leftPointer < rightPointer and numbers[leftPointer] == lv:
23                 leftPointer += 1
24             while leftPointer < rightPointer and numbers[rightPointer] == rv:
25                 rightPointer -= 1
26
27             return res
28         #combinations_of_3 = list(combinations(nums,3))
29         #print(len(combinations_of_3))
30         #out = []
31         #for c in combinations_of_3:
32             # if sum(c) == 0:
33             #     if sorted(c) not in out:
34             #         out.append(sorted(c))
35             #
36         #return out
37
38         nums.sort()
39         out = []
40         for i,n in enumerate(nums):
41             if n>0:
42                 break # sorted, so a positive number means we can never get to 0
43             if i>0 and n == nums[i-1]: # skip if same as previous
44                 continue
45             print(nums[i+1:])
46             idxs = twoSum(nums[i+1:], 0-n)
47             if idxs:
48                 for idx in idxs:
49                     out.append([n, nums[i+idx[0]], nums[i+idx[1]]])
50
51         return out
52     """
53
54 class Solution:
55     def threeSum(self, nums: List[int]) -> List[List[int]]:
56         def twoSum(numbers: List[int], target: int) -> List[List[int]]:

```

```

57     res: List[List[int]] = []
58     if not numbers:
59         return res
60     leftPointer, rightPointer = 0, len(numbers) - 1
61
62     while leftPointer < rightPointer:
63         status = numbers[leftPointer] + numbers[rightPointer]
64         if status < target:
65             leftPointer += 1
66         elif status > target:
67             rightPointer -= 1
68         else:
69             # record (keeping your 1-based indexing)
70             res.append([leftPointer + 1, rightPointer + 1])
71
72             # skip duplicates on BOTH sides
73             lv, rv = numbers[leftPointer], numbers[rightPointer]
74             while leftPointer < rightPointer and numbers[leftPointer] == lv:
75                 leftPointer += 1
76             while leftPointer < rightPointer and numbers[rightPointer] == rv:
77                 rightPointer -= 1
78     return res
79
80     nums.sort()
81     out: List[List[int]] = []
82
83     for i, n in enumerate(nums):
84         if n > 0:
85             break # remaining numbers are positive → no more triplets
86         if i > 0 and n == nums[i - 1]:
87             continue # skip duplicate anchors
88
89         idxs = twoSum(nums[i + 1:], -n) # search suffix
90         for l1, r1 in idxs: # l1/r1 are 1-based within the suffix
91             out.append([n, nums[i + l1], nums[i + r1]])
92
93     return out

```

8 16. 3Sum Closest

```

1 class Solution:
2
3     def threeSumClosest(self, nums: List[int], target: int) -> int:
4
5         nums = sorted(nums)
6
7         n = len(nums)
8         closest_sum = 9999999
9
10        for i in range(n):
11            if i > 0 and nums[i] == nums[i-1]:
12                continue
13
14            lo, hi = i + 1, n - 1
15
16            while lo < hi:
17                cur_sum = nums[i] + nums[lo] + nums[hi]
18
19                if abs(cur_sum - target) < abs(closest_sum - target):
20                    closest_sum = cur_sum
21
22                if cur_sum == target:
23                    return cur_sum
24                elif cur_sum < target:

```

```

25         lo += 1
26     else:
27         hi -= 1
28
29     return closest_sum

```

9 17. Letter Combinations of a Phone Number

10 20. Valid Parentheses

```

1 class Solution:
2     def isValid(self, s: str) -> bool:
3         stack = []
4         hashMap = {
5             ')': '(',
6             '}': '{',
7             ']': '['
8         }
9         for element in s:
10             if stack and (element in hashMap and stack[-1] == hashMap[element]):
11                 stack.pop()
12             else:
13                 stack.append(element)
14         return not stack

```

11 21. Merge Two Sorted Lists

```

1 class Solution:
2     def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) ->
3         ↪ Optional[ListNode]:
4         if not list1:
5             return list2
6         if not list2:
7             return list1
8
9         if list2.val < list1.val:
10             start, other = list2, list1
11         else:
12             start, other = list1, list2
13
14         curr = start
15         while curr and other:
16             if curr.next is None or curr.next.val > other.val:
17                 other_next = other.next
18                 other.next = curr.next
19                 curr.next = other
20                 other = other_next
21                 curr = curr.next
22             else:
23                 curr = curr.next
24         return start

```

12 36. Valid Sudoku

```

1 def checkGroup(group: List[str]) -> bool:
2     print(group)
3     uniques = list(set(group))
4     counts = {s: group.count(s) for s in uniques}

```

```

5  del counts['.']
6  print(counts.values())
7  if any(value > 1 for value in counts.values()):
8      print("tripped")
9      return False
10 if any(int(key) > 9 or int(key) < 1 for key in counts.keys()):
11     return False
12 return True
13 class Solution:
14     def isValidSudoku(self, board: List[List[str]]) -> bool:
15         # check rows:
16         n = len(board)
17         for row in board:
18             if not checkGroup(row):
19                 return False
20         # check columns:
21         for i in range(n):
22             col = [row[i] for row in board]
23             if not checkGroup(col):
24                 return False
25         # check grids:
26         for row_offset in range(0,n,3):
27             for col_offset in range(0,n,3):
28                 subgrid = []
29                 for r in range(row_offset, row_offset + 3):
30                     for c in range(col_offset, col_offset + 3):
31                         subgrid.append(board[r][c])
32                 if not checkGroup(subgrid):
33                     return False
34         return True

```

13 37. Sudoku Solver

```

1  from typing import List
2
3  class Solution:
4      def solveSudoku(self, board: List[List[str]]) -> None:
5          N = len(board)
6          empties = []
7
8          rows = [set() for _ in range(N)]
9          cols = [set() for _ in range(N)]
10         boxes = [set() for _ in range(N)]
11
12         def find_empties():
13             for i, row in enumerate(board):
14                 for j, val in enumerate(row):
15                     if val == '.':
16                         empties.append((i, j))
17                     else:
18                         rows[i].add(val)
19                         cols[j].add(val)
20                         boxes[(i // 3) * 3 + (j // 3)].add(val)
21
22         def validMove(pos):
23             r, c = pos
24             b = (r // 3) * 3 + (c // 3)
25             valid_moves = []
26             for d in "123456789":
27                 if d not in rows[r] and d not in cols[c] and d not in boxes[b]:
28                     valid_moves.append(d)
29             return valid_moves
30
31         def dfs_rec(idx):

```

```

32         if idx == len(empties):
33             return True
34         r, c = empties[idx]
35         b = (r // 3) * 3 + (c // 3)
36         for move in validMove((r, c)):
37             board[r][c] = move
38             rows[r].add(move)
39             cols[c].add(move)
40             boxes[b].add(move)
41             if dfs_rec(idx + 1):
42                 return True
43             board[r][c] = '.'
44             rows[r].remove(move)
45             cols[c].remove(move)
46             boxes[b].remove(move)
47         return False
48
49     find_empties()
50     dfs_rec(0)

```

14 41. First Missing Positive

```

1  class Solution:
2      def firstMissingPositive(self, nums: List[int]) -> int:
3          """shockingly this code is accepted despite the O(nlogn) tc and O(n) sc
4              # will remove this assumption later:
5              nums = sorted(list(set(nums)))
6
7              one = False
8              location = 0
9              for i, num in enumerate(nums):
10                 if num == 1:
11                     one = True
12                     location = i
13
14                 if one == False:
15                     return 1
16
17                 # check subsequent:
18                 j = location
19                 spi = 1
20                 while j < len(nums):
21                     if nums[j] == spi:
22                         spi += 1
23                         j += 1
24                         continue
25                 return spi
26             return spi
27             """
28
29         # cyclic sort:
30         n = len(nums)
31
32         # place each positive integer at the respective index within nums
33         for i in range(n):
34             while 1 <= nums[i] <= n and nums[nums[i] - 1] != nums[i]:
35                 nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1] # swap
36
37         # linear search for first discrepancy
38         for i in range(n):
39             if nums[i] != i + 1:
40                 return i + 1 # returns discrep
41
42         # or returns n + 1

```

```
43 |         return n + 1
```

15 42. Trapping Rain Water

```
1 class Solution:
2     def trap(self, height: List[int]) -> int:
3         l_wall = r_wall = 0
4         n = len(height)
5         max_left = [0] * n
6         max_right = [0] * n
7
8         for i in range(n):
9             j = -i - 1
10            max_left[i] = l_wall
11            max_right[j] = r_wall
12            l_wall = max(l_wall, height[i])
13            r_wall = max(r_wall, height[j])
14
15        summ = 0
16        for i in range(n):
17            pot = min(max_left[i], max_right[i])
18            summ += max(0, pot - height[i])
19
20        return summ
```

16 45. Jump Game II

17 46. Permutations

```
1 class Solution:
2     def permute(self, nums: List[int]) -> List[List[int]]:
3         def backtrack(start, end):
4             if start == end:
5                 result.append(nums[:])
6                 return
7             for i in range(start, end):
8                 nums[start], nums[i] = nums[i], nums[start]
9                 backtrack(start + 1, end)
10                nums[start], nums[i] = nums[i], nums[start]
11        result = []
12        backtrack(0, len(nums))
13        return result
```

18 49. Group Anagrams

```
1 class Solution:
2     def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
3         """
4         #O(mnlogn) sorting bruteforce
5         print(strs)
6         base = []
7         for string in strs:
8             base.append("".join(sorted(string)))
9
10        print(base)
11        # find indices that are all the same
12        idxs = []
13        marked = []
```

```

14     for i, word1 in enumerate(base):
15         i_likes = []
16         for j, word2 in enumerate(base):
17             if word1 == word2 and i <= j and j not in marked:
18                 marked.append(j)
19                 i_likes.append(j)
20         if i_likes:
21             idxs.append(i_likes)
22
23
24     print(idxs)
25     # replace indices with words:
26     ans = []
27     for tup in idxs:
28         sublist = []
29         for idx in tup:
30             sublist.append(strs[idx])
31         ans.append(sublist)
32
33     return ans
34     """
35
36     # hashmap: O(m*n)
37     hash = {}
38     for s in strs:
39
40         count = [0] * 26
41         for c in s:
42             count[ord(c)-ord("a")] += 1
43
44         key = tuple(count)
45         if key in hash:
46             hash[key].append(s)
47         else:
48             hash[key] = [s]
49
50     return list(hash.values())

```

19 53. Maximum Subarray

```

1 class Solution:
2     def maxSubArray(self, nums: List[int]) -> int:
3         dp = [0] * len(nums)
4         for i, n in enumerate(nums):
5             dp[i] = max(n, dp[i-1] + n)
6         return max(dp)

```

20 54. Spiral Matrix

```

1 class Solution:
2     def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
3         ret = []
4         while matrix:
5             ret += (matrix.pop(0))
6             if matrix and matrix[0]:
7                 for row in matrix:
8                     ret.append(row.pop())
9             if matrix:
10                 ret += (matrix.pop()[::-1])
11             if matrix and matrix[0]:
12                 for row in matrix[::-1]:
13                     ret.append(row.pop(0))

```

```
14     return ret
```

21 56. Merge Intervals

```
1 class Solution:
2     def merge(self, intervals: List[List[int]]) -> List[List[int]]:
3         intervals.sort()
4         merged = []
5
6         for start, end in intervals:
7             if not merged or start > merged[-1][1]:
8                 merged.append([start, end])
9             else:
10                 merged[-1][1] = max(merged[-1][1], end)
11
12         return merged
```

22 70. Climbing Stairs

```
1 class Solution:
2     def climbStairs(self, n: int) -> int:
3         if n == 1:
4             return 1
5         dp = [0] * (n + 1)
6         dp[0] = 0
7         dp[1] = 1
8         dp[2] = 2
9
10        for i in range(3, n + 1):
11            dp[i] = dp[i-1] + dp[i-2]
12        return dp[n]
```

23 77. Combinations

```
1 class Solution:
2     def combine(self, n: int, k: int) -> List[List[int]]:
3         def backtrack(start, path):
4             if len(path) == k:
5                 result.append(path[:])
6                 return
7             for i in range(start, n + 1):
8                 path.append(i)
9                 backtrack(i + 1, path)
10                path.pop()
11        result = []
12        backtrack(1, [])
13        return result
```

24 78. Subsets

```
1 class Solution:
2     def subsets(self, nums: List[int]) -> List[List[int]]:
3         def backtrack(start, path):
4             result.append(path[:])
5             for i in range(start, len(nums)):
6                 path.append(nums[i])
```

```

7         backtrack(i + 1, path)
8         path.pop()
9     result = []
10    backtrack(0, [])
11    return result

```

25 88. Merge Sorted Array

```

1  // function to insert value m at position n in array a by shifting the array
2  void insert(int *a, int m, int n, int l) {
3      printf("debug %d %d %d\n", m, n, l);
4      int temp = a[l-1];
5      for(int i=l-1; i>n; i--) {
6          a[i] = a[i-1];
7      }
8      a[0] = temp;
9      a[n] = m;
10 }
11
12 void merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size, int n){
13     int p1 = m - 1;
14     int p2 = n - 1;
15
16     for (int p = m + n - 1; p >= 0; p--) {
17         if (p2 < 0) {break;}
18         else {
19             nums1[p] = (p1 >= 0 && nums1[p1] > nums2[p2]) ? nums1[p1--] : nums2[p2--];
20         }
21     }
22 }
23
24 /*
25  int offset = 0;
26  for (int i = 0; i < m; i++) {
27      for (int j = 0 + offset; j < n; j++) {
28          // if less than first element
29          if (i == 0 && nums1[i] >= nums2[j]) {
30              printf("insert start\n");
31              insert(nums1, nums2[j], i, m + n);
32              offset++;
33              break;
34          }
35          // if greater than last element
36          else if (i == m - 1 && nums1[i] <= nums2[j]) {
37              printf("insert end\n");
38              insert(nums1, nums2[j], i, m + n);
39              offset++;
40              break;
41          }
42          else if (nums1[i] <= nums2[j] && (i + 1 < m && nums1[i+1] >= nums2[j])){ // belongs in
43              ↪ middle
44              printf("insert middle\n");
45              insert(nums1, nums2[j], i+1, m + n);
46              offset++;
47              break;
48          }
49      }
50  }
51 }
52 */

```

26 92. Reverse Linked List II

```
1 class Solution:
2     def reverseBetween(self, head: Optional[ListNode], left: int, right: int) ->
3         Optional[ListNode]:
4         if not head or left == right:
5             return head
6
7         dummy = ListNode(0, head)
8         leftprev = dummy
9
10        curr = head
11        for _ in range(left - 1):
12            leftprev = curr
13            curr = curr.next
14
15        prev = None
16        tail = curr
17        for _ in range(right - left + 1):
18            nxt = curr.next
19            curr.next = prev
20            prev = curr
21            curr = nxt
22
23        leftprev.next = prev
24        tail.next = curr
25        return dummy.next
```

27 100. Same Tree

```
1 class Solution:
2     def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:
3         stack = [(p, q)]
4
5         while stack:
6             node1, node2 = stack.pop()
7
8             if not node1 and not node2:
9                 continue
10
11            elif None in [node1, node2] or node1.val != node2.val:
12                return False
13
14            stack.append((node1.left, node2.left))
15            stack.append((node1.right, node2.right))
16
17        return True
```

28 102. Binary Tree Level Order Traversal

```
1 class Solution:
2     def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
3         if not root:
4             return []
5
6         queue = deque([root])
7         result = []
8
9         while queue:
10            level = []
11            for i in range(len(queue)):
```

```

12         node = queue.popleft()
13         level.append(node.val)
14         if node.left:
15             queue.append(node.left)
16         if node.right:
17             queue.append(node.right)
18         result.append(level)
19     return result

```

29 104. Maximum Depth of Binary Tree

```

1 class Solution:
2     def maxDepth(self, root: Optional[TreeNode]) -> int:
3         if not root:
4             return 0
5         return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1

```

30 108. Convert Sorted Array to Binary Search Tree

```

1 class Solution:
2     def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:
3         def convert(left, right):
4             if left > right:
5                 return None
6             mid = (left + right) // 2
7             node = TreeNode(nums[mid])
8             node.left = convert(left, mid - 1)
9             node.right = convert(mid + 1, right)
10            return node
11        return convert(0, len(nums) - 1)

```

31 111. Minimum Depth of Binary Tree

```

1 class Solution:
2     def minDepth(self, root: Optional[TreeNode]) -> int:
3         if not root:
4             return 0
5
6         queue = deque([(root, 1)])
7
8         while queue:
9             node, level = queue.popleft()
10
11             if not node.left and not node.right:
12                 return level
13
14             if node.left:
15                 queue.append((node.left, level + 1))
16
17             if node.right:
18                 queue.append((node.right, level + 1))
19
20        return 0

```

32 112. Path Sum

```

1 class Solution:

```

```

2  def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:
3      if not root:
4          return False
5
6      stack = [(root, root.val)]
7
8      while stack:
9          node, val = stack.pop()
10
11         if not node.left and not node.right and val == targetSum:
12             return True
13
14         if node.right:
15             stack.append((node.right, val + node.right.val))
16         if node.left:
17             stack.append((node.left, val + node.left.val))
18
19     return False

```

33 121. Best Time to Buy and Sell Stock

```

1  class Solution:
2      def maxProfit(self, prices: List[int]) -> int:
3          l, r = 0, 1
4          maxP = 0
5
6          while r != len(prices):
7              if prices[l] < prices[r]:
8                  prof = prices[r] - prices[l]
9                  maxP = max(maxP, prof)
10             else:
11                 l = r
12             r += 1
13
14     return maxP

```

34 128. Longest Consecutive Sequence

```

1  class Solution:
2      def longestConsecutive(self, nums: List[int]) -> int:
3          numSet = set(nums) # O(n) average time, O(n) space
4          longest = 0
5          for n in numSet: # ← iterate uniques to avoid duplicate re-walks
6              # check if it is the start of the sequence
7              if (n - 1) not in numSet:
8                  length = 0
9                  while (n + length) in numSet:
10                     length += 1
11                 longest = max(length, longest)
12     return longest

```

35 133. Clone Graph

```

1  class Solution:
2      def cloneGraph(self, node: Optional['Node']) -> Optional['Node']:
3          if not node:
4              return node
5
6          queue = deque([node])

```

```

7     clones = {node.val: Node(node.val)}
8
9     while queue:
10         curr = queue.popleft()
11         curr_clone = clones[curr.val]
12
13         for neighbor in curr.neighbors:
14             if neighbor.val not in clones:
15                 clones[neighbor.val] = Node(neighbor.val)
16                 queue.append(neighbor)
17                 curr_clone.neighbors.append(clones[neighbor.val])
18     return clones[node.val]

```

36 136. Single Number

```

1 class Solution:
2     def singleNumber(self, nums: List[int]) -> int:
3         xor = 0
4         for num in nums:
5             xor ^= num
6         return xor

```

37 141. Linked List Cycle

```

1 class Solution:
2     def hasCycle(self, head: Optional[ListNode]) -> bool:
3         curr = head
4         visited = set()
5         while curr:
6             if curr not in visited:
7                 visited.add(curr)
8             else:
9                 return True
10            curr = curr.next
11        return False

```

38 150. Evaluate Reverse Polish Notation

```

1 class Solution:
2     def evalRPN(self, tokens: List[str]) -> int:
3         stack = []
4         for token in tokens:
5             if token not in {'+', '-', '*', '/'}:
6                 stack.append(int(token))
7             else:
8                 op1 = stack.pop()
9                 op2 = stack.pop()
10                if token == '+':
11                    stack.append(op2 + op1)
12                elif token == '*':
13                    stack.append(op2 * op1)
14                elif token == '/':
15                    stack.append(int(op2 / op1))
16                elif token == '-':
17                    stack.append(op2 - op1)
18        return stack.pop()

```

39 151. Reverse Words in a String

```
1 import re
2 class Solution:
3     def reverseWords(self, s: str) -> str:
4         splt = re.split('\\s+', s)
5         splt.reverse()
6         return " ".join(splt).strip()
7
```

40 152. Maximum Product Subarray

41 153. Find Minimum in Rotated Sorted Array

42 155. Min Stack

```
1 class MinStack:
2
3     def __init__(self):
4         self.data = []
5
6     def push(self, val: int) -> None:
7         self.data.append(val)
8
9     def pop(self) -> None:
10        self.data.pop()
11
12    def top(self) -> int:
13        return self.data[-1]
14
15    def getMin(self) -> int:
16        return min(self.data)
```

43 167. Two Sum II - Input Array Is Sorted

```
1 class Solution:
2     def twoSum(self, numbers: List[int], target: int) -> List[int]:
3         leftPointer, rightPointer = 0, len(numbers) - 1
4         while leftPointer != rightPointer:
5             status = numbers[leftPointer] + numbers[rightPointer]
6             if status < target:
7                 leftPointer += 1
8             elif status > target:
9                 rightPointer -= 1
10            else:
11                return [leftPointer+1, rightPointer+1]
12
13        return []
14
```

44 169. Majority Element

```
1 class Solution:
2     def majorityElement(self, nums: List[int]) -> int:
3         """my naive soln"""
4         d = {x:nums.count(x) for x in nums}
5         a, b = d.keys(), d.values()
```

```

6         max_value = max(b)
7         max_index = list(b).index(max_value)
8         return (list(a)[max_index])
9
10        # o(n^2) because we run o(n) count on each x
11        """
12
13        """
14        candidate = 0
15        count = 0
16        # phase 1: find candidate
17        for num in nums:
18            if count == 0:
19                candidate = num
20            count += (1 if num == candidate else -1)
21
22        return candidate
23        """
24        count = {} # dictionary.
25        res, maxCount = 0, 0
26
27        for n in nums:
28            count[n] = 1 + count.get(n, 0)
29            res = n if count[n] > maxCount else res
30            maxCount = max(count[n], maxCount)
31
32        return res

```

45 200. Number of Islands

```

1 class Solution:
2     def numIslands(self, grid: List[List[str]]) -> int:
3         visited = set()
4         count = 0
5         m = len(grid)
6         n = len(grid[0])
7
8         def bfs(r, c):
9             q = deque()
10            visited.add((r, c))
11            q.append((r, c))
12
13            while q:
14                row, col = q.popleft()
15                directions = [[1, 0], [0, 1], [-1, 0], [0, -1]]
16                for dr, dc in directions:
17                    r, c = row + dr, col + dc
18                    if (r in range(m) and c in range(n) and grid[r][c] == "1" and (r, c) not in
19                        ↪ visited):
20                        q.append((r, c))
21                        visited.add((r, c))
22
23            for i in range(m):
24                for j in range(n):
25                    if grid[i][j] == '1' and (i, j) not in visited:
26                        count += 1
27                        bfs(i, j)
28
29            return count

```

46 202. Happy Number

```
1 class Solution:
2     def isHappy(self, n: int) -> bool:
3         """
4         # N is input size, n is number of digits of N
5         visited = set() # O(log n)
6         while n != 1:
7             m = 0
8             if n in visited: # O(1)
9                 return False
10            digits = [int(digit) for digit in str(n)] # O(log n)
11            for digit in digits: # O(log n)
12                m += digit*digit
13            visited.add(n)
14            n = m
15        return True
16
17        """
18        # Time Complexity: O(log n) - number of digits in n
19        # Space Complexity: O(log n) - size of visited set
20
21        visited: set[int] = set() # Track numbers we've seen to detect cycles
22
23        while n not in visited:
24            visited.add(n)
25
26            if n == 1:
27                return True
28
29            # Calculate sum of squared digits
30            current_sum: int = 0
31            while n > 0:
32                digit: int = n % 10
33                current_sum += digit * digit
34                n //= 10
35
36            n = current_sum
37
38        return False # We found a cycle, number is not happy
39
```

47 203. Remove Linked List Elements

```
1 class Solution:
2     def removeElements(self, head: Optional[ListNode], val: int) -> Optional[ListNode]:
3         curr = head
4         dummy = ListNode(-1, head)
5         prev = dummy
6         while curr:
7             if curr.val == val:
8                 prev.next = curr.next
9                 curr = curr.next
10            else:
11                prev = curr
12                curr = curr.next
13        return dummy.next
```

48 206. Reverse Linked List

```
1 /**
```

```

2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8  struct ListNode* reverseList(struct ListNode* head){
9      if (head == NULL || head->next == NULL) {
10         return head;
11     }
12
13     struct ListNode *new = NULL;
14     struct ListNode *curr = head;
15     while (curr != NULL) {
16         struct ListNode *tmp = curr->next;
17         curr->next = new;
18         new = curr;
19         curr = tmp;
20     }
21     return new;
22 }
23

```

49 207. Course Schedule

```

1  class Solution:
2      def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
3          adj = {course: [] for course in range(numCourses)}
4          for course, pre in prerequisites:
5              adj[course].append(pre)
6
7          for course in range(numCourses):
8              stack = [(course, set())]
9              while stack:
10                 cur_course, visited = stack.pop()
11                 if cur_course in visited:
12                     return False
13                 visited.add(cur_course)
14                 for pre in adj[cur_course]:
15                     stack.append((pre, visited.copy()))
16             adj[course] = []
17         return True

```

50 209. Minimum Size Subarray Sum

```

1  class Solution:
2      def minSubArrayLen(self, target: int, nums: List[int]) -> int:
3          l = 0
4          total = 0
5          res = float('inf')
6
7          for r in range(len(nums)):
8              total += nums[r]
9
10             while total >= target:
11                 res = min(res, r - l + 1)
12                 total -= nums[l]
13                 l += 1
14             if res == float('inf'):
15                 return 0
16             else:

```

```
17     return res
```

51 215. Kth Largest Element in an Array

```
1 class Solution:
2     def findKthLargest(self, nums: List[int], k: int) -> int:
3         heap = nums[:k]
4         heapq.heapify(heap)
5
6         for num in nums[k:]:
7             if num > heap[0]:
8                 heapq.heapreplace(heap, num)
9
10        return heap[0]
```

52 217. Contains Duplicate

```
1 class Solution:
2     def containsDuplicate(self, nums: List[int]) -> bool:
3         #return True if len(set(nums)) != len(nums) else False
4         return len(set(nums)) != len(nums)
```

53 219. Contains Duplicate II

```
1 class Solution:
2     def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
3         seen = set()
4         for i, num in enumerate(nums):
5             if num in seen:
6                 return True
7             seen.add(num)
8             if len(seen) > k:
9                 seen.remove(nums[i - k])
10        return False
```

54 225. Implement Stack using Queues

```
1 class MyStack:
2
3     def __init__(self):
4         self.stack = deque()
5
6     def push(self, x: int) -> None:
7         self.stack.append(x)
8
9     def pop(self) -> int:
10        return self.stack.pop()
11
12    def top(self) -> int:
13        return self.stack[-1]
14
15    def empty(self) -> bool:
16        return True if len(self.stack) == 0 else False
```

55 226. Invert Binary Tree

```
1 class Solution:
2     def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
3         stack = [root]
4
5         while stack:
6             curr = stack.pop()
7
8             if curr:
9                 curr.left, curr.right = curr.right, curr.left
10                stack.extend([curr.right, curr.left])
11        return root
```

56 230. Kth Smallest Element in a BST

```
1 class Solution:
2     def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
3         stack = []
4
5         while root or stack:
6             if root:
7                 stack.append(root)
8                 root = root.left
9             else:
10                root = stack.pop()
11                k -= 1
12
13                if k == 0:
14                    return root.val
15
16                root = root.right
```

57 234. Palindrome Linked List

```
1 class Solution:
2     def isPalindrome(self, head: Optional[ListNode]) -> bool:
3         vals = []
4         curr = head
5         while curr:
6             vals.append(curr.val)
7             curr = curr.next
8         return vals == vals[::-1]
```

58 235. Lowest Common Ancestor of a Binary Search Tree

```
1 class Solution:
2     def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
3         small = min(p.val, q.val)
4         large = max(p.val, q.val)
5
6         while root:
7             if root.val > large:
8                 root = root.left
9             elif root.val < small:
10                root = root.right
11            else:
12                return root
```

```
13
14     return None
```

59 236. Lowest Common Ancestor of a Binary Tree

```
1 class Solution:
2     def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
3         if root == None or root == p or root == q:
4             return root
5         left = self.lowestCommonAncestor(root.left, p, q)
6         right = self.lowestCommonAncestor(root.right, p, q)
7         if left != None and right != None:
8             return root
9         if left != None:
10            return left
11        return right
```

60 238. Product of Array Except Self

```
1 class Solution:
2     def productExceptSelf(self, nums: List[int]) -> List[int]:
3         """o(n^2)
4         res = [1] * len(nums)
5         for i,num in enumerate(nums):
6             for j in range(len(nums)):
7                 res[j] *= (num if i !=j else 1)
8
9         return res
10        """
11
12        """better code, but still not fast enough
13        # calculate prefix
14        prefix = [0] * (len(nums) + 2)
15        prefix[0], prefix[len(nums)+1] = 1,1
16        for i in range(len(nums)):
17            prefix[i+1] = (nums[i] * prefix[i])
18
19        print(prefix)
20
21        # calculate postfix
22        postfix = [0] * (len(nums) + 2)
23        postfix[0], postfix[len(nums)+1] = 1,1
24        print(postfix)
25        for i in reversed(range(len(nums))):
26            postfix[i+1] = (nums[i] * postfix[i+2])
27
28        print(postfix)
29        # multiply prefix with postfix for each n
30        res = [0] * len(nums)
31        for i in range(len((nums))):
32            print(res)
33            res[i] = prefix[i] * postfix[i+2]
34
35        return res
36        """
37
38        # the issue above was space complexity.
39        # we are going to update the result array for both prefix and postfix
40
41        res = [1] * len(nums)
42        # prefix loop:
43        for i in range(1, len(nums)):
```

```

44         res[i] = nums[i-1] * res[i-1]
45
46     postfix = 1
47     for j in reversed(range(len(nums)-1)):
48         postfix *= nums[j+1]
49         res[j] *= postfix
50
51     return res
52

```

61 239. Sliding Window Maximum

```

1  from collections import deque
2  #from typing import List
3  class Solution:
4      def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
5          """anki
6          q = deque()
7          left = right = 0
8
9          def slide_right():
10             nonlocal right
11             while q and nums[q[-1]] < nums[right]:
12                 q.pop()
13             q.append(right)
14             right += 1
15
16          def slide_left():
17             nonlocal left
18             left += 1
19             if q and left > q[0]:
20                 q.popleft()
21
22          result = []
23
24          while right < k:
25             slide_right()
26             result.append(nums[q[0]])
27
28          while right < len(nums):
29             slide_right()
30             slide_left()
31             result.append(nums[q[0]])
32
33          return result
34          """
35          output = []
36          l = r = 0
37          q = deque()
38          while r < len(nums):
39             while q and nums[q[-1]] < nums[r]:
40                 q.pop()
41             q.append(r)
42
43             # remove left val from window
44             if l > q[0]:
45                 q.popleft()
46
47             if (r+1) >= k:
48                 output.append(nums[q[0]])
49                 l += 1
50
51             r+= 1
52          return output

```

```

53
54
55     """naive
56     left = 0
57     right = k
58     result = []
59
60     N = len(nums)
61
62     while right <= N:
63         result.append(max(nums[left:right]))
64         left += 1
65         right += 1
66
67     return result
68     """

```

62 260. Single Number III

```

1  class Solution:
2      def singleNumber(self, nums: List[int]) -> List[int]:
3          nums_str = str(nums)
4          nums_set = set(nums)
5          ans = []
6          for i in nums_set:
7              if nums_str.count(str(i)) == 1:
8                  ans.append(i)
9
10
11     return ans

```

63 268. Missing Number

```

1  class Solution:
2      def missingNumber(self, nums: List[int]) -> int:
3          return sum(range(len(nums)+1))-sum(nums)

```

64 303. Range Sum Query - Immutable

```

1  class NumArray:
2
3      def __init__(self, nums: List[int]):
4          self.acc_nums = [0]
5          for num in nums:
6              self.acc_nums.append(self.acc_nums[-1] + num)
7
8      def sumRange(self, left: int, right: int) -> int:
9          return self.acc_nums[right + 1] - self.acc_nums[left]

```

65 322. Coin Change

```

1  class Solution:
2      def coinChange(self, coins: List[int], amount: int) -> int:
3          dp = [amount + 1] * (amount + 1)
4          dp[0] = 0
5

```

```

6     for i in range(1, amount + 1):
7         for c in coins:
8             if (i - c) >= 0:
9                 dp[i] = min(dp[i], 1 + dp[i - c])
10    return dp[amount] if (dp[amount] != amount + 1) else -1

```

66 338. Counting Bits

```

1 class Solution:
2     def countBits(self, n: int) -> List[int]:
3         ans = []
4         for i in range(n+1):
5             temp = bin(i)
6             ans.append(str(temp)[2:].count('1'))
7
8     return ans

```

67 347. Top K Frequent Elements

```

1 def topKFrequent(nums: List[int], k:int) -> List[int]:
2     """
3     d = defaultdict(int)
4     for item in nums:
5         d[item] += 1
6     l = list(sorted(d.items(), key = lambda x: x[1],reverse=True))
7     return [x[0] for x in l[:k]]
8     """
9     # O(nlogn), dominated by the sorting
10    # O(n)
11
12    #####
13    # O(n) solution via bucket sort:
14    # 1. count frequencies O(n)
15    frequencies = defaultdict(int) # lookup failures will be populated with a default int of 0
16    for item in nums:
17        frequencies[item] += 1
18
19    n = len(nums)
20
21    # 2. create buckets (index = frequency) O(n)
22    buckets = [[] for _ in range(n+1)]
23    for num, frequency in frequencies.items():
24        buckets[frequency].append(num)
25
26    # 3. collect k most frequent items O(n)
27    result = []
28    while n > -1 and k > 0:
29        if buckets[n]:
30            result.append(buckets[n].pop())
31            k -= 1
32        else:
33            n -= 1
34    return result

```

68 392. Is Subsequence

```

1 class Solution:
2     def isSubsequence(self, s: str, t: str) -> bool:
3         """reasonably good, but not duplicate resistant code

```

```

4     matched = 0
5     str_idx = -1
6     for s_char in s:
7         for curr_idx, t_char in enumerate(t):
8             if s_char == t_char:
9                 if curr_idx > str_idx:
10                    str_idx = curr_idx
11                    matched += 1
12            else:
13                return False
14    if matched == len(s):
15        return True
16    return False
17    """
18    matched = 0
19    match_idx = 0
20    for s_char in s:
21        for curr_idx, t_char in enumerate(t[match_idx:]):
22            if s_char == t_char:
23                matched += 1
24                match_idx += curr_idx + 1
25            break
26    if matched == len(s):
27        return True
28    return False

```

69 424. Longest Repeating Character Replacement

```

1  from collections import defaultdict
2  def maxRep(s: str, k: int) -> int:
3      count = defaultdict(int)
4      max_count = 0
5      left = right = 0
6      while right < len(s):
7          count[s[right]] += 1
8          max_count = max(max_count, count[s[right]])
9          right += 1
10         if right - left - max_count > k:
11             count[s[left]] -= 1
12             left += 1
13     return right - left

```

70 438. Find All Anagrams in a String

```

1  import itertools
2  class Solution:
3      def findAnagrams(self, s: str, p: str) -> List[int]:
4          """
5          positions = set()
6          perms = [''.join(q) for q in itertools.permutations(p)]
7
8          for perm in perms:
9              for i in range(len(s)):
10                 index = s.find(perm, i)
11                 if index == -1:
12                     continue
13                 if index not in positions:
14                     positions.add(index)
15                 i = index + 1
16         return list(positions)
17         """
18     if len(p) > len(s): return []

```

```

19     pCount, sCount = {}, {}
20     for i in range(len(p)):
21         pCount[p[i]] = 1 + pCount.get(p[i], 0)
22         sCount[s[i]] = 1 + sCount.get(s[i], 0)
23
24     res = [0] if sCount == pCount else []
25     l = 0
26
27     for r in range(len(p), len(s)):
28         sCount[s[r]] = 1 + sCount.get(s[r], 0)
29         sCount[s[l]] -= 1
30
31         if sCount[s[l]] == 0:
32             sCount.pop(s[l])
33             l += 1
34         if sCount == pCount:
35             res.append(l)
36     return res

```

71 448. Find All Numbers Disappeared in an Array

```

1 class Solution:
2     def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
3         missing = []
4         """
5         hashmap = {}
6         for num in nums:
7             hashmap[num] = 1
8         for i in range(1, len(nums)+1):
9             if i not in hashmap:
10                 missing.append(i)
11         return missing
12         """
13         uniques = set(nums)
14         for i in range(1, len(nums)+1):
15             if i not in uniques:
16                 missing.append(i)
17         return missing

```

72 450. Delete Node in a BST

```

1 class Solution:
2     def deleteNode(self, root: Optional[TreeNode], key: int) -> Optional[TreeNode]:
3         if not root:
4             return None
5
6         if key < root.val:
7             root.left = self.deleteNode(root.left, key)
8         elif key > root.val:
9             root.right = self.deleteNode(root.right, key)
10        else:
11            if not root.left:
12                return root.right
13            if not root.right:
14                return root.left
15
16            successor = root.right
17            while successor.left:
18                successor = successor.left
19
20            root.val = successor.val
21            root.right = self.deleteNode(root.right, successor.val)

```

```
22
23     return root
```

73 530. Minimum Absolute Difference in BST

```
1 class Solution:
2     def getMinimumDifference(self, root: Optional[TreeNode]) -> int:
3         min_diff = float('inf')
4         prev_val = float('-inf')
5
6         stack = []
7         while root or stack:
8             if root:
9                 stack.append(root)
10                root = root.left
11            else:
12                root = stack.pop()
13                min_diff = min(min_diff, root.val - prev_val)
14                prev_val = root.val
15                root = root.right
16
17         return min_diff
```

74 543. Diameter of Binary Tree

```
1 class Solution:
2     def diameterOfBinaryTree(self, root: Optional[TreeNode]) -> int:
3         self.diameter = 0
4
5         def depth(root):
6             if not root:
7                 return 0
8             left_depth = depth(root.left)
9             right_depth = depth(root.right)
10            self.diameter = max(self.diameter, left_depth + right_depth)
11            return 1 + max(left_depth, right_depth)
12
13         depth(root)
14         return self.diameter
```

75 567. Permutation in String

```
1 import itertools
2 class Solution:
3     def checkInclusion(self, s1: str, s2: str) -> bool:
4         """
5         if len(s1) == len(s2) and set(s1) != set(s2): return False
6         perms = [''.join(q) for q in itertools.permutations(s1)]
7
8         res = False
9         for perm in perms:
10             print(perm)
11             index = s2.find(perm, 0)
12             if index != -1:
13                 res = True
14         return res
15         """
16         n1 = len(s1)
17         n2 = len(s2)
```

```

18
19     if n1 > n2:
20         return False
21
22     s1_counts = [0] * 26
23     s2_counts = [0] * 26
24
25
26     for i in range(n1):
27         s1_counts[ord(s1[i]) - ord('a')] += 1
28         s2_counts[ord(s2[i]) - ord('a')] += 1
29
30     if s1_counts == s2_counts:
31         return True
32
33     for i in range(n1, n2):
34         s2_counts[ord(s2[i]) - ord('a')] += 1
35         s2_counts[ord(s2[i - n1]) - ord('a')] -= 1
36         if s1_counts == s2_counts:
37             return True
38     return False

```

76 621. Task Scheduler

```

1 class Solution:
2     def leastInterval(self, tasks: List[str], n: int) -> int:
3         task_counts = Counter(tasks)
4
5         max_heap = []
6         for count in task_counts.values():
7             max_heap.append(-count)
8
9         heapq.heapify(max_heap)
10
11         time = 0
12         wait_queue = deque()
13
14         while max_heap or wait_queue:
15             time += 1
16
17             if max_heap:
18                 current_task = heapq.heappop(max_heap)
19                 current_task += 1
20
21                 if current_task != 0:
22                     wait_queue.append((current_task, time + n))
23
24             if wait_queue and wait_queue[0][1] == time:
25                 heapq.heappush(max_heap, wait_queue.popleft()[0])
26
27         return time

```

77 637. Average of Levels in Binary Tree

```

1 class Solution:
2     def averageOfLevels(self, root: Optional[TreeNode]) -> List[float]:
3         queue = deque([root])
4         result = []
5
6         while queue:
7             level = []
8             for i in range(len(queue)):

```

```

9         node = queue.popleft()
10        level.append(node.val)
11        if node.left: queue.append(node.left)
12        if node.right: queue.append(node.right)
13
14        result.append(sum(level) / len(level))
15    return result

```

78 647. Palindromic Substrings

```

1  class Solution:
2      def countSubstrings(self, s: str) -> int:
3          res = 0
4          for i in range(len(s)):
5              res += self.countPali(s, i, i)
6              res += self.countPali(s, i, i+1)
7          return res
8
9      def countPali(self, s, l, r):
10         res = 0
11         while l >= 0 and r < len(s) and s[l] == s[r]:
12             res += 1
13             l -= 1
14             r += 1
15         return res

```

79 653. Two Sum IV - Input is a BST

```

1  class Solution:
2      def findTarget(self, root: Optional[TreeNode], k: int) -> bool:
3          queue = deque([root])
4          num_set = set()
5
6          while queue:
7              node = queue.popleft()
8
9              if (k - node.val) in num_set:
10                 return True
11             else:
12                 num_set.add(node.val)
13
14             if node.left:
15                 queue.append(node.left)
16             if node.right:
17                 queue.append(node.right)
18
19         return False

```

80 700. Search in a Binary Search Tree

```

1  class Solution:
2      def searchBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:
3          while root:
4              if root.val == val:
5                  return root
6              elif root.val < val:
7                  root = root.right
8              else:
9                  root = root.left

```

```
10 | return None
```

81 701. Insert into a Binary Search Tree

```
1 class Solution:
2     def insertIntoBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:
3         new_node = TreeNode(val)
4
5         if not root:
6             return new_node
7
8         current = root
9         while True:
10            if val < current.val:
11                if current.left:
12                    current = current.left
13                else:
14                    current.left = new_node
15                    break
16            else:
17                if current.right:
18                    current = current.right
19                else:
20                    current.right = new_node
21                    break
22        return root
```

82 784. Letter Case Permutation

```
1 class Solution:
2     def letterCasePermutation(self, s: str) -> List[str]:
3         output = [""]
4
5         for c in s:
6             tmp = []
7             if c.isalpha():
8                 for o in output:
9                     tmp.append(o + c.lower())
10                    tmp.append(o + c.upper())
11            else:
12                for o in output:
13                    tmp.append(o + c)
14            output = tmp
15        return output
```

83 787. Cheapest Flights Within K Stops

```
1 class Solution:
2     def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst: int, k: int) ->
3     ↪ int:
4         prices = [float('inf')] * n
5         prices[src] = 0
6
7         for i in range(k + 1):
8             tmpPrices = prices.copy()
9
10            for from_node, to_node, cost in flights:
11                if prices[from_node] == float('inf'):
12                    continue
```

```

12         if prices[from_node] + cost < tmpPrices[to_node]:
13             tmpPrices[to_node] = prices[from_node] + cost
14
15         prices = tmpPrices
16
17         return -1 if prices[dst] == float('inf') else prices[dst]

```

84 845. Longest Mountain in Array

```

1 class Solution:
2     def longestMountain(self, arr: List[int]) -> int:
3         max_length = 0
4
5         for i in range(1, len(arr) - 1):
6             if arr[i-1] < arr[i] > arr[i+1]:
7                 l = r = i
8                 while l > 0 and arr[l] > arr[l-1]:
9                     l -= 1
10                while r < len(arr) - 1 and arr[r] > arr[r+1]:
11                    r += 1
12
13                max_length = max(max_length, r - l + 1)
14
15         return max_length

```

85 876. Middle of the Linked List

```

1 class Solution:
2     def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
3         N = 0
4         counter = 0
5         curr = head
6         while curr:
7             N += 1
8             curr = curr.next
9         curr = head
10        stopAt = floor(N / 2)
11        while counter < stopAt:
12            curr = curr.next
13            counter += 1
14        return curr

```

86 973. K Closest Points to Origin

```

1 class Solution:
2     def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
3         heap = []
4
5         for (x, y) in points:
6             dist = -(x * x + y * y)
7             if len(heap) == k:
8                 heapq.heappushpop(heap, (dist, x, y))
9             else:
10                heapq.heappush(heap, (dist, x, y))
11
12        return [(x, y) for (dist, x, y) in heap]

```

87 977. Squares of a Sorted Array

```
1 class Solution:
2     def sortedSquares(self, nums: List[int]) -> List[int]:
3         answer = collections.deque()
4         l, r = 0, len(nums) - 1
5
6         while l <= r:
7             left, right = abs(nums[l]), abs(nums[r])
8             if left > right:
9                 answer.appendleft(left * left)
10                l += 1
11            else:
12                answer.appendleft(right * right)
13                r -= 1
14        return list(answer)
```

88 1200. Minimum Absolute Difference

```
1 class Solution:
2     def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
3         mins = []
4         arr.sort()
5         l, r = 0, 1
6         dif = max(arr) - min(arr)
7         while r < len(arr):
8             a, b = arr[l], arr[r]
9             cur_dif = b - a
10            if cur_dif < dif:
11                mins = []
12                mins.append([a, b])
13            elif cur_dif == dif:
14                mins.append([a, b])
15            dif = min(dif, cur_dif)
16            r += 1
17            l += 1
18        return mins
```

89 1266. Minimum Time Visiting All Points

```
1 class Solution:
2     def minTimeToVisitAllPoints(self, points: List[List[int]]) -> int:
3         res = 0
4         x1, y1 = points.pop()
5         while points:
6             x2, y2 = points.pop()
7             x_dist = abs(x1 - x2)
8             y_dist = abs(y1 - y2)
9             res += max(x_dist, y_dist)
10            x1, y1 = x2, y2
11        return res
```

90 1365. How Many Numbers Are Smaller Than the Current Number

```
1 class Solution:
2     def smallerNumbersThanCurrent(self, nums: List[int]) -> List[int]:
3         """
4         counts = []
```

```

5     for a in nums:
6         count = 0
7         for b in nums:
8             if a != b and b < a:
9                 count += 1
10            counts.append(count)
11    return counts
12    """
13
14    mynums = sorted(nums)
15    iteration = -1
16    hashMap = {}
17    for item in mynums:
18        iteration += 1
19        if item in hashMap:
20            continue
21        hashMap[item] = iteration
22    return [hashMap[item] for item in nums]
23

```

91 1366. Rank Teams by Voters

```

1  class Solution:
2      def rankTeams(self, votes: List[str]) -> str:
3          n = len(votes[0])
4          score = {team: [0] * n for team in votes[0]}
5
6          for vote in votes:
7              for i, team in enumerate(vote):
8                  score[team][i] -= 1
9
10         return "".join(sorted(votes[0], key=lambda x: (score[x], x)))

```

92 1382. Balance a Binary Search Tree

```

1  class Solution:
2      def balanceBST(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
3          nodes = self.inorder(root)
4          return self.build(nodes, 0, len(nodes) - 1)
5
6      def inorder(self, root):
7          result = []
8          stack = []
9          current = root
10         while current or stack:
11             while current:
12                 stack.append(current)
13                 current = current.left
14             current = stack.pop()
15             result.append(current.val)
16             current = current.right
17         return result
18
19     def build(self, nums, left, right):
20         if left > right:
21             return None
22         mid = (left + right) // 2
23         node = TreeNode(nums[mid])
24         node.left = self.build(nums, left, mid - 1)
25         node.right = self.build(nums, mid + 1, right)
26         return node

```

93 2034. Stock Price Fluctuation

```
1 class StockPrice:
2
3     def __init__(self):
4         self.timestamps = {}
5         self.max_time = 0
6         self.min_heap = []
7         self.max_heap = []
8
9     def update(self, timestamp: int, price: int) -> None:
10         self.timestamps[timestamp] = price
11         self.max_time = max(self.max_time, timestamp)
12
13         heapq.heappush(self.min_heap, (price, timestamp))
14         heapq.heappush(self.max_heap, (-price, timestamp))
15
16     def current(self) -> int:
17         return self.timestamps[self.max_time]
18
19     def maximum(self) -> int:
20         cur_price, timestamp = heapq.heappop(self.max_heap)
21
22         while -cur_price != self.timestamps[timestamp]:
23             cur_price, timestamp = heapq.heappop(self.max_heap)
24
25         heapq.heappush(self.max_heap, (cur_price, timestamp))
26
27         return -cur_price
28
29     def minimum(self) -> int:
30         cur_price, timestamp = heapq.heappop(self.min_heap)
31
32         while cur_price != self.timestamps[timestamp]:
33             cur_price, timestamp = heapq.heappop(self.min_heap)
34
35         heapq.heappush(self.min_heap, (cur_price, timestamp))
36
37         return cur_price
38
39 # Time
40 # Update: pushing onto a heap is  $\log(k)$ , where  $k$  is the number of items we have in the heap
41 # Current:  $O(1)$ 
42 # Maximum: rejug the heap takes  $\log(k)$ , and we might need to do this  $n$  times, hence:  $O(n\log k)$ 
43 # Minimum: rejug the heap takes  $\log(k)$ , and we might need to do this  $n$  times, hence:  $O(n\log k)$ 
44 # Space:  $O(n)$ 
```

94 2073. Time Needed to Buy Tickets

```
1 class Solution:
2     def timeRequiredToBuy(self, tickets: List[int], k: int) -> int:
3         result = 0
4         for i in range(len(tickets)):
5             if i <= k:
6                 result += min(tickets[i], tickets[k])
7             else:
8                 result += min(tickets[i], tickets[k] - 1)
9         return result
```

95 2933. High-Access Employee

```
1 class Solution:
2     def findHighAccessEmployees(self, access_times: List[List[str]]) -> List[str]:
3         n = len(access_times)
4         names = set(access_times[i][0] for i in range(n))
5         employees = defaultdict(list)
6         for i, access in enumerate(access_times):
7             employee, time = access[0], access[1]
8             employees[employee].append(time)
9         for k, v in employees.items():
10            new_v = [0] * len(v)
11            for i, hhmm in enumerate(v):
12                minutes = 60 * int(hhmm[:2]) + int(hhmm[2:])
13                new_v[i] = minutes
14            print(new_v)
15            employees[k] = sorted(new_v)
16
17        return [
18            name
19            for name, times in employees.items()
20            if any(times[i+2] - times[i] < 60 for i in range(len(times) - 2))
21        ]
```

96 3026. Maximum Good Subarray Sum

```
1 class Solution:
2     def maximumSubarraySum(self, nums: List[int], k: int) -> int:
3         """bruteforce:
4         n = len(nums)
5         list_of_lists = []
6         curr_max = float("-inf")
7         for i in range(n):
8             for j in range(n):
9                 if i < j:
10                    list_of_lists.append(nums[i:j+1])
11                    #print(nums[i:j], i, j)
12        print(list_of_lists)
13        for subarray in list_of_lists:
14            print(f"subarray: {subarray}")
15            if abs(subarray[0] - subarray[-1]) == k:
16                print(f"good array: {subarray}")
17                curr_max = max(curr_max, sum(subarray))
18        return 0 if curr_max == float("-inf") else curr_max
19        """
20        prefix = 0
21        best = float("-inf")
22
23        seen = {}
24
25        for x in nums:
26            if x - k in seen:
27                best = max(best, prefix + x - seen[x-k])
28            if x + k in seen:
29                best = max(best, prefix + x - seen[x+k])
30
31            if x not in seen:
32                seen[x] = prefix
33            else:
34                seen[x] = min(seen[x], prefix)
35
36        prefix += x
37
```

```
return 0 if best == float("-inf") else best
```