***REVISION SHEET FOR MY LEETCODE SOLUTIONS***

|  |  |  |
| --- | --- | --- |
| ***TITLE*** | ***TOPICS*** | ***APPROACH/REMARK*** |
| [1. Two Sum](https://leetcode.com/problems/two-sum/) | Hash table |  |
| [2. Add Two Numbers](https://leetcode.com/problems/add-two-numbers/) | LL |  |
| [3. Longest Substring Without Repeating Characters](https://leetcode.com/problems/longest-substring-without-repeating-characters/) | Hash table, two pointer |  |
| [4. Median of Two Sorted Arrays](https://leetcode.com/problems/median-of-two-sorted-arrays/) | Binary search, Divide & conquer | ***Approach 1 -*** ***BINARY SEARCH*** ***O(log(min(n, m)))***  *perform BS to find the correct partition in array1,*  *according to it only, partition in array2 is determined*  ***Approach 2 – Divide & conquer O(log(m+n))***  Also, we can use algo of *find kth element in 2 sorted arrays.*  *We divide both arr in 2 parts each.*  *-If k lies in 1st part of merged sorted arr, eliminate second half of any one array based on their medians.*  *-else k lies in second half, so eliminate first half of any one array based on their medians.* |
| [5. Longest Palindromic Substring](https://leetcode.com/problems/longest-palindromic-substring/) (return the substring) | DP, LCS | dp[i][j] = T/F tells if substr from index [i,j] is palindrome.  Start from substrings of len1 and go till substrs of length L. |
| [11. Container With Most Water](https://leetcode.com/problems/container-with-most-water/) | 2 pointer | For every bar we encounter, we find the max area container we can make if this baar is taken.  Move i, j from left, right to inside  Evaluate the capacity of container having the smaller height, and advance that ptr |
| [12. Integer to Roman](https://leetcode.com/problems/integer-to-roman/) | String | We first find the upper bound of N -> (i).  Main idea is to handle cases like 40, 90, where we need to subtract prev pow(10) number. For this, we find prev val.  If val[i] is power of 10, prevVal is val[i-2] else [i-1] |
| [13. Roman to Integer](https://leetcode.com/problems/roman-to-integer/) | string | Scan from back. Subtract ith val if it is less thn (i+i)th |
| [15. 3Sum](https://leetcode.com/problems/3sum/) | sort | Sort the array, then take one ele, and find a pair having target-ele in remaining array.  Use ***if (i > 0 and nums[i] == nums[i-1]) continue;***  To deal with duplicates. |
| [16. 3Sum Closest](https://leetcode.com/problems/3sum-closest/) | sort | Same as above. Just keep updating the ans wrt abs diffrence from target. |
| [17. Letter Combinations of a Phone Number](https://leetcode.com/problems/letter-combinations-of-a-phone-number/) | backtracking | Basic backtracking |
| [18. 4Sum](https://leetcode.com/problems/4sum/) | Recursion, 2sum | make a k-sum template  used recursion to break k-sum to 2-sum problem |
| [19. Remove Nth Node From End of List](https://leetcode.com/problems/remove-nth-node-from-end-of-list/) | LL | Take 2 ptr. Move fast ptr n-1 steps ahead of slow.  Then move both until fast reaches the end. Now, slow is the answer |
| [21. Merge Two Sorted Lists](https://leetcode.com/problems/merge-two-sorted-lists/) | LL | if (l1->val < l2->val)  l1->next = mergeTwoLists(l1->next, l2);  else  l2->next = mergeTwoLists(l1, l2->next); |
| [22. Generate Parentheses](https://leetcode.com/problems/generate-parentheses/) | Recursion, backtracking | only add ‘(’, ‘)’ when it will remain a valid sequence. We can do this by keeping track of the number of opening and closing brackets we have placed so far.  We can start an opening bracket if we still have one (of n) left to place. And we can start a closing bracket if it would not exceed the number of opening brackets. |
| [23. Merge k Sorted Lists](https://leetcode.com/problems/merge-k-sorted-lists/) | Pq, LL | Put all heads in pq. Pop from pq one by one, and add the next of the popped ele in the pq again until pq is not empty. |
| [25. Reverse Nodes in k-Group](https://leetcode.com/problems/reverse-nodes-in-k-group/) | Recursion, LL | Reverse first k nodes, then get next part from recursion on k+1th node |
| [26. Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array/) | Array, 2 ptr | Keep one ptr at the start, and put only unique ele when it is encountered first, and increment it. |
| [29. Divide Two Integers](https://leetcode.com/problems/divide-two-integers/) | Bit shifting | Use bit shifting, and subtract from A  While B <= A, keep left shifting B  Let B is shifted k times, so 1 << k will need to be added to answer, and shifted B is subtracted from A |
| [30. Substring with Concatenation of All Words](https://leetcode.com/problems/substring-with-concatenation-of-all-words/) | Hashtable, rolling hash, 2 ptr, string | we check for every possible position of i.  starting from every position, use a hashmap to record the times we have seen.  We can prune on finding an unneeded word, or if a word occurs more than what it should. |
| [31. Next Permutation](https://leetcode.com/problems/next-permutation/) | Array, sort | Find the last decreasing pair (i, j) where i < j.  Now find the last ele k that is greater than i.  Swap i, k.  Sort [i+1, end] |
| [32. Longest Valid Parentheses](https://leetcode.com/problems/longest-valid-parentheses/) | Stack, dp | Use stack to store indices of opening braces  When a closing brace is found that doesn’t have its opening brace in stack, we mark the starting of a new substring.  Else if its pair is found, pop update the max length as i-stk.top() or i-left accordingly if stk becomes empty after popping.  *stk.pop();*  *if (stk.empty()) len = max(len, i-left);*  *else len = max(len, i-stk.top());* |
| [33. Search in Rotated Sorted Array](https://leetcode.com/problems/search-in-rotated-sorted-array/) | Binary search | Rotated sorted array has 2 non-dec subsrrays.  Make a custom comparator  If mid and target are in same subarray, comp is A[mid],  Else, we don’t need mid now as mid, target are in different subarr. So, compare with A[lo] to find if we should discard left subarr, or right subarr. |
| [81. Search in Rotated Sorted Array II](https://leetcode.com/problems/search-in-rotated-sorted-array-ii/) | Binary search | **Arr may contain dups**  All same as above, just  Keep incrementing lo while A[lo]==A[mid]  Keep decrementing hi while A[hi]==A[mid] |
| [34. Find First and Last Position of Element in Sorted Array](https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/) | Binary search | Use lower, upper bounds |
| [37. Sudoku Solver](https://leetcode.com/problems/sudoku-solver/) | Backtracking | **Key thing: *denote every cell by a single number from 0 to n\*n-1.***  If we reach cell n\*n, backtracking is complaeted.  If some digit is placed, move to next cell.  Else try placing every possible digit in cur cell, and backtrack on next cell.  Have a helper fn to check if placing cur digit is possible in cell i, j.  It’ll check if same digit exists in cur row, col, and block. |
| [40. Combination Sum II](https://leetcode.com/problems/combination-sum-ii/) | backtrack | ***Each ele can be taken just once***  Normal backtracking  Sort the arr, then take one ele, backtrack on rest part.  Get rid of dups by  *if (i>start and cand[i]==cand [i-1]) continue;* |
| [39. Combination Sum](https://leetcode.com/problems/combination-sum/) | backtracking | Same as above.  ***But, each ele can be taken any # of times***  *Let x can be used count times, use it count times.*  *Every time, after pushing* ***x,*** *backtrack on remaining part.*  *When count* ***x*** *are used, remove all of them from sum and cur array.* |
| [41. First Missing Positive](https://leetcode.com/problems/first-missing-positive/) | array | *Store nums in indices by making val at that index as –ve.*  *One main thing is to get rid of all –ve nums at first. So, make all n <= 0 as size+1*  *Then iterate over array, and make A[abs(A[i])-1] as negative.*  *At last scan the array. If A[i] is +ve, it means i+1 is the answer.* |
| [42. Trapping Rain Water](https://leetcode.com/problems/trapping-rain-water/) | Stack, 2 ptr | ***---- Using stack ---***  store next greater and prev greater for ever index.  Then for every index sum *min(next[i], prev[i])-h[i]*  ***---- Using 2 pointer ----***  Keep track of maxOnLeft, maxOnRight  If maxOnLeft <= maxOnRight: sum water at left ptr,  Else sum water at right ptr. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |