Table of Contents

1. [FindMinSortedArray 3](#_Toc510597276)
2. [FindMinSortedRotatedArray2 3](#_Toc510597277)
3. [SearchSortedRotatedArray 3](#_Toc510597278)
4. [SearchSortedRotatedArray2 4](#_Toc510597279)
5. [SquareRoot 4](#_Toc510597280)
6. [Find Peak Element 4](#_Toc510597281)
7. [First Bad Version 5](#_Toc510597282)
8. [GuessNumberHigherOrLower 5](#_Toc510597283)
9. [Buy Sell Stocks – One Transaction 5](#_Toc510597284)
10. [Buy Sell Stocks – Two Transaction 6](#_Toc510597285)
11. [Buy Sell Stocks – Infinite Transactions 6](#_Toc510597286)
12. [Buy Sell Stocks – K Transactions 7](#_Toc510597287)
13. [Buy Sell Stocks – Infinite Transactions + Cool Down 7](#_Toc510597288)
14. [Buy Sell Stocks – Infinite Transactions + Transaction Fee 8](#_Toc510597289)
15. [AddTwoNumbers (LinkedList) 9](#_Toc510597290)
16. [Binary Addition (2 Strings) 9](#_Toc510597291)
17. [Average of Binary Tree Levels 9](#_Toc510597292)
18. [HeightBalancedBinaryTree 10](#_Toc510597293)
19. [BinaryTreeLevelOrderTraversal 10](#_Toc510597294)
20. [Root-to-leaf paths 11](#_Toc510597295)
21. [CheckSubTree 11](#_Toc510597296)
22. [Diameter of a Binary Tree 12](#_Toc510597297)
23. [ClimbingStairs 13](#_Toc510597298)
24. [Delete LinkedList Node 13](#_Toc510597299)
25. [ExcelTitle 13](#_Toc510597300)
26. [ExcelTitleNumber 14](#_Toc510597301)
27. [FactorialTrailingZeros 14](#_Toc510597302)
28. [Find Disappeared Numbers 14](#_Toc510597303)
29. [FindAnagramMappings 15](#_Toc510597304)
30. [FindMissingNumber 15](#_Toc510597305)
31. [FirstUniqueCharacterInString 16](#_Toc510597306)
32. [HammingDistance 16](#_Toc510597307)
33. [Happy Number: 17](#_Toc510597308)
34. [HouseRobber 17](#_Toc510597309)
35. [IntegerToRoman 18](#_Toc510597310)
36. [Island Perimeter 18](#_Toc510597311)
37. [Judge Circle 18](#_Toc510597312)
38. [LetterCasePermutation 19](#_Toc510597313)
39. [LinkedListCycle 19](#_Toc510597314)
40. [LisenceKeyFormatting 20](#_Toc510597315)
41. [LongestCommonPrefix 20](#_Toc510597316)
42. [LongestUnivaluePath 20](#_Toc510597317)
43. [MajorityElement 21](#_Toc510597318)
44. [MaximumDepthOfBinaryTree 22](#_Toc510597319)
45. [MaximumSubarraySum 22](#_Toc510597320)
46. [MeetingRooms 23](#_Toc510597321)
47. [MergeSortedArray 23](#_Toc510597322)
48. [MinCostClimbingStairs 23](#_Toc510597323)
49. [MinimumAbsDifferenceInBST 24](#_Toc510597324)
50. [MinStack 25](#_Toc510597325)
51. [MoveZeros 25](#_Toc510597326)
52. [MovingAverage 26](#_Toc510597327)
53. [FindNthDigit: -- TODO 26](#_Toc510597328)
54. [Hamming Weight: 26](#_Toc510597329)
55. [PaintFence 27](#_Toc510597330)
56. [PaintHouses 27](#_Toc510597331)
57. [PalindromeNumber 28](#_Toc510597332)
58. [PathSumIII – Total Paths to a sum 28](#_Toc510597333)
59. [PlusOne 29](#_Toc510597334)
60. [Power of 3, 2 30](#_Toc510597335)
61. [RangeSumQueryImmutable 30](#_Toc510597336)

# FindMinSortedArray

1. Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
2. Find the minimum element. You may assume no duplicate exists in the array.
3. **Solution:**
4. middle = left + (right - left) / 2;
5. if middle is greater than right, more left +1. Stay there and move the left until it hits left < right and exit.

# FindMinSortedRotatedArray2

1. "Find Minimum in Rotated Sorted Array": What if duplicates are allowed?
2. **Solution:**
3. Go until left <= right, with extra conditions while arr[left] == arr[right] and left <= right, left ++
4. If arr[left] <=arr[right], break out. Otherwise binary search as before

# SearchSortedRotatedArray

1. Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
2. You are given a target value to search. If found in the array return its index, otherwise return -1.
3. You may assume no duplicate exists in the array.
4. **Solution:**
5. Go until left <= right, if key == middle, return.
6. if left <= middle,
   1. if left <= key < middle, then right = middle - 1
   2. else left = middle + 1
7. else
   1. if middle <= key < right, then left = middle + 1
   2. else right = middle - 1

# SearchSortedRotatedArray2

1. Follow up for "Search in Rotated Sorted Array": What if duplicates are allowed?
2. **Solution:**
3. Extra condition to go one by one if equals. As in find min index
4. Go until left <= right, if key == middle, return.
5. if left < middle,
   1. if left <= key < middle, then right = middle - 1
   2. else left = middle + 1
6. else if left > middle
   1. if middle <= key < right, then left = middle + 1
   2. else right = middle – 1
7. else left == middle,
   1. just go left ++

# SquareRoot

1. Implement int sqrt(int x). Compute and return the square root of x. x is guaranteed to be a non-negative integer.
2. **Solution:**
3. Middle = left + (Right – left)/2. Square = (double) middle \* (double) middle.
4. square > target, right = middle -1
5. square < target, left = middle + 1

# Find Peak Element

1. A peak element is an element that is greater than its neighbors.
2. **Solution**:
3. Left, right,
4. While(l<r)
   1. middle = left + (right – left)/2
   2. //converge on the peak.
   3. If (mid > mid + 1)
      1. Right = mid; // move right until the peak
   4. Else if (mid < mid + 1)
      1. Left = mid + 1; // move left until left == right
5. Return left;

# First Bad Version

1. n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.
2. **Solution:**
3. Middle = left + (right – left) /2
4. If (middle is not bad)
   1. left = middle +1;
5. Else
   1. Right = middle

# GuessNumberHigherOrLower

1. I pick a number from 1 to n. You have to guess which number I picked.
2. Every time you guess wrong, I'll tell you whether the number is higher or lower.
3. **Solution**:
   1. mid = left + (right – left) / 2;
   2. if guess(mid) == 0 return;
   3. if guess(mid) < 0, right = mid;
   4. else left = mid +1
4. at the end return left;

# Buy Sell Stocks – One Transaction

1. Input: [7, 1, 5, 3, 6, 4]
2. Output: 5
3. max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)
4. **Solution:**
5. Maintain three variables for max profit at the end of each day,
6. 0TransactionHold0 = 0
7. 1TransactionHold0 = 0
8. 1TransactionHold1 = Integer.Min\_value.
9. Go through each price, and calculate the vals
   1. 1TransactionHold0 = max (1TransactionHold0 (hold), 1TransacctionHold1 + price (sell))
   2. 1TransacctionHold1 = max (1TransacctionHold1 (hold), 0TransactionHold0 - price (buy))
10. Return 1TransactionHold0
11. **NOTE:** Always start from higher transactions to lower transactions hold0 – to use previous values next time

# Buy Sell Stocks – Two Transaction

1. At most two transactions
2. **Solution:**
3. Maintain five variables for max profit at the end of each day,
4. 0TransactionHold0 = 0
5. 1TransactionHold0 = 0
6. 1TransactionHold1 = Integer.Min\_value
7. 2TransactionHold0 = 0
8. 2TransactionHold1 = Integer.Min\_value
9. Go through each price, and calculate the vals
   1. 2TransactionHold0 = max (2TransactionHold0 (hold), 2TransactionHold1 – price(sell))
   2. 2TransactionHold1 = max (2TransactionHold1 (hold), 1TransactionHold0 + price (sell))
   3. 1TransactionHold0 = max (1TransactionHold0 (hold), 1TransacctionHold1 + price (sell))
   4. 1TransacctionHold1 = max (1TransacctionHold1 (hold), 0TransactionHold0 - price (buy))
10. Return 2TransactionHold0

# Buy Sell Stocks – Infinite Transactions

1. As many transactions as possible.
2. **Solution:**
3. If the hold1 takes the max of Prev hold0 – price (buy), then it accumulates the transactions.
4. prevTransactionHold0 = 0
5. TransactionHold0 = 0
6. TransactionHold1 = Integer.Min\_value.
7. Go through each price, and calculate the vals
   1. prevTransactionHold0 = TransactionHold0;
   2. TransactionHold0 = max (TransactionHold0 (hold), 1TransacctionHold1 + price (sell))
   3. TransactionHold1 = max (TransactionHold1 (hold), prevTransactionHold0 - price (buy))
8. Return 1TransactionHold0

# Buy Sell Stocks – K Transactions

1. Specified number of transactions allowed.
2. **Solution:**
3. After, n/2 you cannot do more transactions as its always buy at one day and sell at one day. So, for if k > n/2, do infinite transactions, for k < n/2 do array’s
4. So, maintain two arrays for holding the values.
5. Hold0 array filled with 0 with k + 1 size.
6. Hold1 array filled with -1 with k + 1 size. To start with base condition
7. Go through each price, and calculate the vals
   1. For each price, j = k, j until 0
      1. Hold0[j] = max (hold0[j], hold1[j] + price) sell
      2. Hold1[j] = max (hold1[j], hold0[j-1] - price) buy

# Buy Sell Stocks – Infinite Transactions + Cool Down

1. Wait one day before starting new transaction.
2. **Solution:**
3. If the hold1 takes the max of Prev hold0 – price (buy), then it accumulates the transactions.
4. prevTransactionHold0 = 0
5. TransactionHold0 = 0
6. TransactionHold1 = Integer.Min\_value.
7. Go through each price, and calculate the vals
8. prevPrevTransactionHold0 = prevTransactionHold0
9. prevTransactionHold0 = TransactionHold0;
10. TransactionHold0 = max (TransactionHold0 (hold), 1TransacctionHold1 + price (sell))
11. TransactionHold1 = max (TransactionHold1 (hold), prevPrevTransactionHold0 - price (buy))
12. Return TransactionHold0

# Buy Sell Stocks – Infinite Transactions + Transaction Fee

1. Pay fee for each time
2. **Solution:**
3. Reduce the fee from profit;
4. If the hold1 takes the max of Prev hold0 – price (buy), then it accumulates the transactions.
5. prevTransactionHold0 = 0
6. TransactionHold0 = 0
7. TransactionHold1 = Integer.Min\_value.
8. Go through each price, and calculate the vals
9. prevTransactionHold0 = TransactionHold0;
10. TransactionHold0 = max (TransactionHold0 (hold), 1TransacctionHold1 + price (sell))
11. TransactionHold1 = max (TransactionHold1 (hold), prevTransactionHold0 - price (buy) - fee)
12. Return 1TransactionHold0
13. **NOTE:** Take out while buying to avoid overflow…

# AddTwoNumbers (LinkedList)

1. Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
2. Output: 7 -> 0 -> 8
3. **Solution:**
4. Create new list nodes to store result.Only one loop until either one is available, l1 or l2 or carry.
5. If l1 is null, val is zero. L2 is null, val is zero.
6. Sum = l1.val + l2.val + carry
7. currentVal = sum %10
8. carry = sum / 10

# Binary Addition (2 Strings)

1. a = "11"
2. b = "1"
3. Return "100"
4. **Solution:**
5. i = size of a – 1 (lsb)
6. j = size of b -1 (msb)
7. Only one loop until the end of a or b
   1. Int sum = 0
   2. If a is not done && a[i] == 1
      1. Sum ++
   3. If b is not done && b[i] == 1
      1. Sum ++
   4. Sum += carry
   5. Carry = sum / 2;
   6. Insert at beginning of result string (Char) (sum %2) + ‘0’

# Average of Binary Tree Levels

1. Return the average value of the nodes on each level in the form of an array.
2. Example 1:
3. Input:
4. 3
5. / \
6. 9 20
7. / \
8. 15 7
9. Output: [3, 14.5, 11]
10. **Solution:**
11. Create a queue to hold the nodes in a level. Check until queue is empty – outer loop. Get all the elements at a time in the queue. 🡪 empty it by calculating the n. Loop through all the elements and find the average. At the same time, add the left and rights to the same queue for next level.

# HeightBalancedBinaryTree

1. Given a binary tree, determine if it is height-balanced.
2. **Solution:**
3. Condition: a node is balanced if left subtree and right subtree is balanced + number of nodes in right subtree – left subtree <= 1
4. Create a wrapper to hold isBalanced, height.
5. Report height from bottom with Max (left subtree height, right subtree height) + 1 (current level)

# BinaryTreeLevelOrderTraversal

1. For example:
2. Given binary tree [3,9,20,null,null,15,7],
3. 3
4. / \
5. 9 20
6. / \
7. 15 7
8. return its level order traversal as:
9. [
10. [3],
11. [9,20],
12. [15,7]
13. ]
14. **Solution:**
15. Same as average of levels. Get the queue size at each iteration. Empty until the size and add the next levels into the same queue. Calculate result when iterating through the inner n elements of the queue

# Root-to-leaf paths

1. 1
2. / \
3. 2 3
4. \
5. 5
6. All root-to-leaf paths are:
7. ["1->2->5", "1->3"]
8. **Solution:**
9. Use recursive helper,
10. Base condition root == null,
11. Then add root, to a list of current nodes.
12. Do (required operation) when left == null && right == null.
    1. Creating the string of paths.
13. If (left != null) {
    1. Do left,
    2. Remove current nodes.size() – 1.
14. }
15. If (right != null) {
    1. Do right,
    2. Remove current nodes.size() – 1.
16. }

# CheckSubTree

1. 3
2. / \
3. 4 5
4. / \
   * + 1. 2
5. /
6. 0
7. Given tree t:
8. 4
9. / \
   * + 1. 2
10. Return false.
11. **Solution:**
12. Subtree:
    1. If (t1.val is same t2.val && equals tree (t1, t2)
       1. Return true;
    2. Else
       1. check subtree (t1.left, t2) OR check subtree(t1.right, t2)
13. Equals Tree:
    1. Base condition: true
    2. t1 == null && t2 == null
    3. Base condition: False
    4. t1 == null && t2 != null, t1 != null && t2 == null
    5. T1.val == t2. Val && Equals Tree(t1.left, t2.left) && Equals Tree(t1.right, t2.right)

# Diameter of a Binary Tree

1. Given a binary tree
2. 1
3. / \
4. 2 3
5. / \
6. 4 5
7. Return 3, which is the length of the path [4,2,1,3] or [5,2,1,3].
8. **Solution:**
9. Max of (Max height of left sub tree, Max height of right subtree)+ 1.

# ClimbingStairs

1. Input: 3
2. Output: 3
3. Explanation: There are three ways to climb to the top.
4. 1 step + 1 step + 1 step
5. 1 step + 2 steps
6. 2 steps + 1 step
7. **Solution:**
8. Fibonacci sequence
9. 0 step - 1
   * + 1. step – 1
       2. step – 2
10. Loop through from 3 steps to k steps
    1. 3 steps = 2 step + 1 step
    2. 1step = 2 step;
    3. 2step = 3 step;

# Delete LinkedList Node

1. delete a node (except the tail) in a singly linked list, given only access to that node.
2. **Solution**:
3. Copy the value from the next node and modify the current pointer to the next one

# ExcelTitle

1. Positive integer, return its corresponding column title as appear in an Excel sheet.
2. For example:
   * + 1. -> A
       2. -> B
       3. -> C
3. ...
4. 26 -> Z
5. 27 -> AA
6. 28 -> AB
7. **Solution:**
8. While(n >0)
   1. n--
   2. Char c = (char) n %26 + ‘A’ 🡪 inorder to add from A, reduce n by 1 before
   3. Next n = n / 26;
9. Reverse the chars formed

# ExcelTitleNumber

1. Given a title get the number.
2. A -> 1
3. B -> 2
4. C -> 3
5. ...
6. Z -> 26
7. AA -> 27
8. AB -> 28
9. **Solution:**
10. For each char c,
11. Result = Result \* 26 + ((int) c – ‘A’ + 1)

# FactorialTrailingZeros

1. integer n, return the number of trailing zeroes in n!
2. **Solution:**
3. Zero’s are always produced by 2 \* 5. Hence count how many 5’s are present. That many Zeros will be present.
4. So, n/5 + recursive (n/5) until zero

# Find Disappeared Numbers

1. Some elements appear twice and others appear once.
2. Input:
3. [4,3,2,7,8,2,3,1]
4. Output:
5. [5,6]
6. **Solution:**
7. Go through each element in the array and consider that as the index and make it negative. Offset for 1 for index position.
8. Then go through each element again and see which index is not negative. That is the number not present in the array. + 1 for index.
9. for(int i=0;i<arr.length;i++)
   1. int val = Math.abs(arr[i]) -1;
   2. if(arr[val] > 0)
      1. arr[val] = -arr[val];
10. for(int i=0;i<arr.length;i++)
    1. if(arr[i] <0)
       1. result.add(i + 1)

# FindAnagramMappings

1. For example, given
2. A = [12, 28, 46, 32, 50]
3. B = [50, 12, 32, 46, 28]
4. We should return
5. [1, 4, 3, 2, 0]
6. **Solution:**
7. Go through b and create a map of number to pos.
8. Then go through a, get the element from map and put the val in the result

# FindMissingNumber

1. Input: [3,0,1]
2. Output: 2
3. Example 2
4. Input: [9,6,4,2,3,5,7,0,1]
5. Output: 8
6. **Solution 1**: XOR
7. a^b^b =a.. hence
8. result = 0
9. For each element I in array from 0 postition
   1. result = result ^ i ^ arr[i]
10. finally do the last index which would return the element that’s missing.
11. Return Result ^ arr len + 1 -1(for index)
12. **Solution 2:** Total Sum and subtraction from actual sum
13. N \* (n-1) / 2 – sum of all the elements in the array

# FirstUniqueCharacterInString

1. Examples:
2. s = "leetcode"
3. return 0.
4. s = "loveleetcode",
5. return 2.
6. **Solution**:
7. Use an array of size 26 and char integer val as the index.
8. Increment the frequency every time you see that char.
9. Go through the array and when a frequency is 1, return that char.

# HammingDistance

1. The Hamming distance between two integers is the number of positions at which the corresponding bits are different.
2. **Solution**:
3. A XOR B = no of bits that are different
4. Int result = 0
5. While (bitset > 0)
   1. Result ++;
   2. Bitset = bitset & (bitset – 1)

# Happy Number:

1. A happy number is a number defined by the following process:
2. Starting with any positive integer, replace the number by the sum of the squares of its digits,
3. and repeat the process until the number equals 1 (where it will stay),
4. **Solution**:
5. Necessary to maintain a hashset to track the seen numbers so that we don’t go into the infinite loop.
6. Get the digits and then calculate the square of sum
7. While(num > 0)
   1. Digit = Num % 10
   2. Num = num / 10;
8. Foreach (digit : digits)
   1. Sum = sum + digit \* digit
9. If sum == num return happy;
10. Else go loop(sum)

# HouseRobber

1. constraint stopping you from robbing each of them is that adjacent houses have security system connected
2. **Solution**:
3. Dp array contains the max until that house.
4. Dp[0] = num[0]
5. Dp[1] = num[1]
6. I = 2
7. While(i<num.length)
   1. MaxWithPrevprevhouse = num[i] + vals[i-2];
   2. MaxWithprevprevprevhouse = nums[i] +( i-3 <= 0) ? 0 : vals[i-3]
   3. currMax = max (MaxWithPrevprevhouse, MaxWithprevprevprevhouse)
   4. totalMax = max (totalMax, currMax)
8. return totalMax

# IntegerToRoman

1. **Solution**:
2. X – 10, L – 50, C – 100, D – 500, M - 10000
3. int[] values = { 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };
4. String[] strs = { "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" };
5. Go from left to right in the values. If you can subtract the num, subtract and add the string.
6. For(Value : values)
   1. while(num >= Value)
      1. Num = num – value;
      2. Result.append(stringofvalue);

# Island Perimeter

1. Example:
2. [
3. [0,1,0,0],
4. [1,1,1,0],
5. [0,1,0,0],
6. [1,1,0,0]
7. ]
8. Answer: 16
9. **Solution**:
10. Go through each element, if zero then count how many ones you can see. left, right, up, down
11. If one, increase the count for boundaries. If I ==0, j==0, or I = length, j = length.

# Judge Circle

1. Example 1:
2. Input: "UD"
3. Output: true
4. Example 2:
5. Input: "LL"
6. Output: false
7. **Solution**:
8. Have x and y co-ordinates. Go left, x decrease, go right x increase, go up y increase, go down y decrease. Check x and y == 0.

# LetterCasePermutation

1. Examples:
2. Input: S = "a1b2"
3. Output: ["a1b2", "a1B2", "A1b2", "A1B2"]
4. **Solution**:
5. Use recursion. Do two recursions. One for normal. One for uppercase or lowercase. Pass in the currentIndex. If val is 97 – 122 then do uppercase, if 65 – 90 do lowercase

# LinkedListCycle

1. **Solution**:
2. Slow, fast counter.
3. While(fast.next != null && fast.next.next != null)
   1. If(slow == fast)
      1. Break;
4. Slow = head
5. While(slow != fast)
   1. Slow = slow.next;
   2. Fast = fast.next;
6. Return slow;

# LisenceKeyFormatting

1. Example 2:
2. Input: S = "2-5g-3-J", K = 2
3. Output: "2-5G-3J"
4. **Solution**:
5. Take the string s, replace the – with “”
6. List<String> keys;
7. For (i = length; i – k >= 0; I = I -k)
   1. Keys.add( s.substring(i-k, i))
8. Keys reverse
9. if(i>0)
   1. mylist.add(S.substring(0, i));
10. String.join(“-”, keys);

# LongestCommonPrefix

1. Write a function to find the longest common prefix string amongst an array of strings.
2. Solution:
3. Take first word as lcp.
4. Iterate through the other words
   1. For each char in lcp until the smaller length string.
      1. If char doesn’t match, then substring until the index and make it lcp.
      2. If index is 0, lcp is “” return.
   2. If( lcp is longer than string)
      1. Lcp = lcp.substring(0, index from previous exit)
5. Return lcp

# LongestUnivaluePath

1. Input:
   1. 1
2. / \
3. 4 5
4. / \ \
5. 4 4 5
6. Output:
7. 2
8. **Solution**:
9. Pass in root value and current node. for root, it’s the same node and its same value.
10. But, first calculate what your sub trees tell you. Max is max of current max, left max + right max.
11. Then send above your height.
12. Let children decide to return the height. You just add them and find max.
13. getLengthRecursive(current, root.val)
14. if(current == null)
    1. return 0;
15. left = getLengthRecursive(current.left, current.val)
16. right = getLengthRecursive(current.right, current.val)
17. len = max(len, left + right);
18. if(current.val == root.val)
    1. return Math.max (left, right) +1;
19. else return 0;

# MajorityElement

1. The majority element is the element that appears more than ⌊ n/2 ⌋ times.
2. **Solution**:
3. If a number is greater than n/2 times, its count always going be greater than others.
4. For(int i: nums)
5. {
   1. If(count == 0)
      1. Result = n
      2. Count++;
   2. Else if(i== num)
      1. Count ++;
   3. Else
      1. Count –
6. }

# MaximumDepthOfBinaryTree

1. 3
2. / \
3. 9 20
4. / \
5. 15 7
6. return its depth = 3.
7. **Solution**:
8. Max depth is Max(left , right) + 1
9. Max(getMax(node.left), getMax(node.right)) + 1;

# MaximumSubarraySum

1. contiguous subarray within an array (containing at least one number) which has the largest sum.
2. [-2,1,-3,4,-1,2,1,-5,4],
3. the contiguous subarray [4,-1,2,1] has the largest sum = 6.
4. **Solution:**
5. When the sum is negative, and less than the current number, ignore the sum and use the current num as the sum
6. For (int i=0; i<nums.length; i++)
   1. If(sum < 0 && sum < nums[i])
      1. Sum = nums[i];
   2. else
      1. Sum = sum + nums[i];
   3. maxSum = max(maxsum, sum)

# MeetingRooms

1. Determine if a person could attend all meetings.
2. [[0, 30],[5, 10],[15, 20]],
3. return false.
4. **Solution:**
5. Sort the meeting times with comparator. Sort based on start time. If they are equal compare end times. Then see if any of the start times are before the end times of previous meetings, return false.

# MergeSortedArray

1. Two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.
2. **Solution**:
3. Start from the end. Get the end of arr1 = write index, m as I, n as j
4. Do until I and j and greater than 0.
5. While(j>0 && i>0)
6. {
7. If(arr[i] > arr[j])
8. {
9. Arr[writeIndex--] = arr[i--];
10. }
11. Else
12. {
13. Arr[writeIndex--] = arr[j--];
14. }
15. At the very end, copy the rest of j elements into arr1. If there are more I,its okay as its already in the right place.
16. While(j>0)
17. {
18. Arr[writeIndex--] = arr[j--];
19. }

# MinCostClimbingStairs

1. Once you pay the cost, you can either climb one or two steps. You need to find minimum cost to reach the top of the floor, and you can either start from the step with index 0, or the step with index 1.
2. Example 1:
3. Input: cost = [10, 15, 20]
4. Output: 15
5. Explanation: Cheapest is start on cost [1], pay that cost and go to the top.
6. Example 2:
7. Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]
8. Output: 6
9. **Solution**:
10. Create a dp array of input size + 1 to store min at the point.
11. // calculate how much cost to get to current. Either from -1 position or -2 position
12. Dp[0] = 0; to get to zero position – 0 cost.
13. Dp[1] = 0; to get to one position – 0 cost
14. For(int I = 2; i<input.length; i++)
15. {
    1. Dp[i] = Min(dp[i-1] + input[i-1], dp[i-2] + input[i-2])
16. }
17. Return dp [input.length +1]

# MinimumAbsDifferenceInBST

1. find the minimum absolute difference between values of any two nodes.
2. Example:
3. Input:
4. 1
5. \
6. 3
7. /
8. 2
9. Output:
10. 1
11. **Solution**:
12. Use upperbound and lower bound and use the BST property.
13. At each node, you need the lowerbound and upperbound
14. Recurse(root.left, null, root)
15. Recurse(root.right, root, null)
16. Recurse(current, leftbound, rightbound)
    1. If(current == null)
       1. Return
    2. If(leftBound!=null)
       1. Calculate min
    3. If(rightbound!=null)
       1. Calculate min
    4. Recurse (current.left, leftBound, current);
    5. Recurse (current.right, current, rightBound);

# MinStack

1. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
2. push(x) -- Push element x onto stack.
3. pop() -- Removes the element on top of the stack.
4. top() -- Get the top element.
5. getMin() -- Retrieve the minimum element in the stack.
6. **Solution**:
7. Maintain a min value.
8. Push x: when a value is less equal to min, push the min & update the min to x; push x anyway;
9. Pop: pop anyway. When the pop is equal to min, pop again and set in min.
10. Top: stack.top
11. Min: minval

# MoveZeros

1. nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].
2. **Solution**:
3. Write index= 0
4. While(until end)
5. {
   1. While(its zero && until end)
      1. Count zeros
   2. If(end)
      1. Break
   3. Arr[writeIndex++] = arr[readIndex++]
6. }
7. While(start from end, until length – zeros count)
8. {
9. Arr[index] = 0;
10. }

# MovingAverage

1. a stream of integers and a window size, calculate the moving average of all integers in the sliding window.
2. **Solution**:
3. Maintain a queue.
4. When queue size is smaller than window size, sum = sum + num; avg = sum / size;
5. When queue size is large, remove element, subtract from sum, add newnum, calculate new sum and return avg

# FindNthDigit: -- TODO

# Hamming Weight:

1. the 32-bit integer ’11' has binary representation 00000000000000000000000000001011,
2. so the function should return 3.
3. **Solution**:
4. Int Bitset = 0;
5. While(num > 0)
6. {
   1. Bitset ++;
   2. num = num & (num – 1); // figures out the next bit
   3. (or)
   4. Bitset = bitset + (num & 1)
   5. Num = num >> 1; //right shift by 1
7. }

# PaintFence

1. There is a fence with n posts, each post can be painted with one of the k colors.
2. You have to paint all the posts such that no more than two adjacent fence posts have the same color.
3. numWays(int n, int k)
4. **Solution**:
5. Dp[n]
6. Dp[0] – k
7. Dp[1] – k \* k
8. For(i=2 untill end)
   1. Dp[i] – dp[ I - 1] \*( k -1) + dp(i- 2) \* (k -1)
9. Return dp[n]

# PaintHouses

1. There are a row of n houses, each house can be painted with one of the three colors:
2. red, blue or green. The cost of painting each house with a certain color is different.
3. You have to paint all the houses such that no two adjacent houses have the same color.
4. The cost of painting each house with a certain color is represented by a n x 3 cost matrix.
5. For example, costs[0][0] is the cost of painting house 0 with color red;
6. costs[1][2] is the cost of painting house 1 with color green, and so on...
7. Find the minimum cost to paint all houses.
8. Int minCost(int[][] costs)
9. **Solution**:
10. Min Cost of painting a house with one color = cost of that color + mincost ( previous house color1, previous house color2)
11. So, from second house, calculate the min.
12. Cost[i][0] = Cost[i][0] + min(Cost[i-1][1], Cost[i-1][2])
13. Cost[i][1] = Cost[i][1] + min(Cost[i-1][0], Cost[i-1][1])
14. Cost[i][2] = Cost[i][2] + min(Cost[i-1][0], Cost[i-1][1])
15. No extra dp array needed as the current one is used to store the dp vals.

# PalindromeNumber

1. whether an integer is a palindrome.
2. **Solution**:
3. Check if x < 0 or x % 10 == 0 return false.
4. While (x > rev) // doing until half in even, or one more time if odd. So rev is more than x.
5. Convert x into rev by
   1. Rev = rev \* 10 + x /10;
   2. X = x / 10;
6. Return x == rev (Even case) || x = rev /10 (oddcase)

# PathSumIII – Total Paths to a sum

1. root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8
2. 10
3. / \
4. 5 -3
5. / \ \
6. 3 2 11
7. / \ \
8. 3 -2 1
9. Return 3. The paths that sum to 8 are:
10. 5 -> 3
11. 5 -> 2 -> 1
12. -3 -> 11
13. **Solution**:
14. Total paths in tree:
    1. Total path from node +
    2. Total paths in tree(node.left, sum, targetsum) +
    3. Totoal Paths in tree(node.right, sum, targetsum)
15. Totalpaths totalPathsfromNode(node, sum, targetsum)
    1. if(node == null)
       1. return 0;
    2. totalpath = 0;
    3. sum = sum + node.val
    4. if(sum == targetsum)
    5. totalPAth++;
16. // iterate further because of negative numbers.
17. totalpath = totalpath + totalPathsfromNode(node.left, sum, targetSum);
    1. totalpath = totalpath + totalPathsfromNode(node.right, sum, targetSum);
    2. return totalPath;

# PlusOne

1. Add plus one to integer represented in a array of digits
2. **Solution**:
3. Carry = 1
4. For(I = lengthofarr -1 ; i>0; i--)
5. {
6. Sum = sum + arr[i] + carry;
7. Arr[i] = sum %10;
8. Carry = sum /10;
9. }
10. If carry >0
    1. Create a new array of length + 1
    2. Copy carry to pos 0,
11. Copy rest of the arr from 1 position

# Power of 3, 2

1. **Solution**:
2. Power of 2 – n>0, n & n-1 == 0
3. Power of 3, n >0 while n >0, n %3 != 0 return false. Else loop with n = n / 3

# RangeSumQueryImmutable

1. Given nums = [-2, 0, 3, -5, 2, -1]
2. sumRange(0, 2) -> 1
3. sumRange(2, 5) -> -1
4. sumRange(0, 5) -> -3
5. **Solution**:
6. Create a new array and Calculate sum until that index.
7. GetSum:
8. if lowerbound == 0 return getVal[upperbound]
9. else getVal[upperbound] - getval[lowerbound - 1]