Table of Contents

[FindMinSortedArray 3](#_Toc510684786)

[FindMinSortedRotatedArray2 3](#_Toc510684787)

[SearchSortedRotatedArray 3](#_Toc510684788)

[SearchSortedRotatedArray2 4](#_Toc510684789)

[SquareRoot 4](#_Toc510684790)

[Find Peak Element 5](#_Toc510684791)

[First Bad Version 5](#_Toc510684792)

[GuessNumberHigherOrLower 5](#_Toc510684793)

[Buy Sell Stocks – One Transaction 5](#_Toc510684794)

[Buy Sell Stocks – Two Transaction 6](#_Toc510684795)

[Buy Sell Stocks – Infinite Transactions 6](#_Toc510684796)

[Buy Sell Stocks – K Transactions 7](#_Toc510684797)

[Buy Sell Stocks – Infinite Transactions + Cool Down 7](#_Toc510684798)

[Buy Sell Stocks – Infinite Transactions + Transaction Fee 8](#_Toc510684799)

[AddTwoNumbers (LinkedList) 8](#_Toc510684800)

[Binary Addition (2 Strings) 8](#_Toc510684801)

[Average of Binary Tree Levels 9](#_Toc510684802)

[HeightBalancedBinaryTree 9](#_Toc510684803)

[BinaryTreeLevelOrderTraversal 10](#_Toc510684804)

[Root-to-leaf paths 10](#_Toc510684805)

[CheckSubTree 11](#_Toc510684806)

[Diameter of a Binary Tree 11](#_Toc510684807)

[ClimbingStairs 12](#_Toc510684808)

[Delete LinkedList Node 12](#_Toc510684809)

[ExcelTitle 12](#_Toc510684810)

[ExcelTitleNumber 13](#_Toc510684811)

[FactorialTrailingZeros 13](#_Toc510684812)

[Find Disappeared Numbers 13](#_Toc510684813)

[FindAnagramMappings 14](#_Toc510684814)

[FindMissingNumber 14](#_Toc510684815)

[Single Number 15](#_Toc510684816)

[FirstUniqueCharacterInString 15](#_Toc510684817)

[HammingDistance 15](#_Toc510684818)

[Happy Number: 16](#_Toc510684819)

[HouseRobber 16](#_Toc510684820)

[IntegerToRoman 16](#_Toc510684821)

[Island Perimeter 17](#_Toc510684822)

[Judge Circle 17](#_Toc510684823)

[LetterCasePermutation 17](#_Toc510684824)

[LinkedListCycle 18](#_Toc510684825)

[LisenceKeyFormatting 18](#_Toc510684826)

[LongestCommonPrefix 18](#_Toc510684827)

[LongestUnivaluePath 19](#_Toc510684828)

[MajorityElement 20](#_Toc510684829)

[MaximumDepthOfBinaryTree 20](#_Toc510684830)

[MaximumSubarraySum 20](#_Toc510684831)

[MeetingRooms 21](#_Toc510684832)

[MergeSortedArray 21](#_Toc510684833)

[MinCostClimbingStairs 22](#_Toc510684834)

[MinimumAbsDifferenceInBST 22](#_Toc510684835)

[MinStack 23](#_Toc510684836)

[MoveZeros 23](#_Toc510684837)

[MovingAverage 24](#_Toc510684838)

[FindNthDigit: -- TODO 24](#_Toc510684839)

[Hamming Weight: 24](#_Toc510684840)

[PaintFence 25](#_Toc510684841)

[PaintHouses 25](#_Toc510684842)

[PalindromeNumber 25](#_Toc510684843)

[PathSumIII – Total Paths to a sum 26](#_Toc510684844)

[PlusOne 27](#_Toc510684845)

[Power of 3, 2 27](#_Toc510684846)

[RangeSumQueryImmutable 27](#_Toc510684847)

[ReadNCharsfromRead4 27](#_Toc510684848)

[RemoveDuplicatesInSortedArray 28](#_Toc510684849)

[RepeatedStringMatch 28](#_Toc510684850)

[ReverseInteger 29](#_Toc510684851)

[ReverseLinkedList 29](#_Toc510684852)

[ReverseVowelsOfString 30](#_Toc510684853)

[RomanToInteger 30](#_Toc510684854)

[SentenceSimilarity 30](#_Toc510684855)

[SortLinkedList – TODO 30](#_Toc510684856)

[Strobogrammatic number 30](#_Toc510684857)

[StrStr 31](#_Toc510684858)

[SumOfLeftLeaves 31](#_Toc510684859)

[Sum & Difference of Two Integers 32](#_Toc510684860)

[Toeplitz Matrix 32](#_Toc510684861)

[TwoSum 32](#_Toc510684862)

[TwoSumII 32](#_Toc510684863)

[TwoSumIII 33](#_Toc510684864)

[TwoSumIV 33](#_Toc510684865)

[ValidAnagram 33](#_Toc510684866)

[ValidPalindromeII 34](#_Toc510684867)

[Valid Square 34](#_Toc510684868)

[WellFormedParanthesis 35](#_Toc510684869)

# FindMinSortedArray

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).

Find the minimum element. You may assume no duplicate exists in the array.

**Solution:**

middle = left + (right - left) / 2;

if middle is greater than right, more left +1. Stay there and move the left until it hits left < right and exit.

# FindMinSortedRotatedArray2

"Find Minimum in Rotated Sorted Array": What if duplicates are allowed?

**Solution**:

Go until left <= right, with extra conditions while arr[left] == arr[right] and left <= right, left ++

If arr[left] <=arr[right], break out. Otherwise binary search as before

# SearchSortedRotatedArray

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).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

**Solution:**

Go until left <= right, if key == middle, return.

if left <= middle,

if left <= key < middle, then right = middle - 1

else left = middle + 1

else

if middle <= key < right, then left = middle + 1

else right = middle - 1

# SearchSortedRotatedArray2

Follow up for "Search in Rotated Sorted Array": What if duplicates are allowed?

**Solution:**

Extra condition to go one by one if equals. As in find min index

Go until left <= right, if key == middle, return.

If left < middle,

if left <= key < middle, then right = middle - 1

else left = middle + 1

else if left > middle

if middle <= key < right, then left = middle + 1

else right = middle – 1

else left == middle,

just go left ++

# SquareRoot

Implement int sqrt(int x). Compute and return the square root of x. x is guaranteed to be a non-negative integer.

**Solution:**

Middle = left + (Right – left)/2. Square = (double) middle \* (double) middle.

square > target, right = middle -1

square < target, left = middle + 1

# Find Peak Element

A peak element is an element that is greater than its neighbors.

**Solution**:

Left, right,

While(l<r)

middle = left + (right – left)/2

//converge on the peak.

If (mid > mid + 1)

Right = mid; // move right until the peak

Else if (mid < mid + 1)

Left = mid + 1; // move left until left == right

Return left;

# First Bad Version

n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

**Solution:**

Middle = left + (right – left) /2

If (middle is not bad)

left = middle +1;

Else

Right = middle

# GuessNumberHigherOrLower

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

**Solution**:

mid = left + (right – left) / 2;

if guess(mid) == 0 return;

if guess(mid) < 0, right = mid;

else left = mid +1

at the end return left;

# Buy Sell Stocks – One Transaction

Input: [7, 1, 5, 3, 6, 4]

Output: 5

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)

**Solution:**

Maintain three variables for max profit at the end of each day,

0TransactionHold0 = 0

1TransactionHold0 = 0

1TransactionHold1 = Integer.Min\_value.

Go through each price, and calculate the vals

1TransactionHold0 = max (1TransactionHold0 (hold), 1TransacctionHold1 + price (sell))

1TransacctionHold1 = max (1TransacctionHold1 (hold), 0TransactionHold0 - price (buy))

Return 1TransactionHold0

**NOTE:** Always start from higher transactions to lower transactions hold0 – to use previous values next time

# Buy Sell Stocks – Two Transaction

At most two transactions

**Solution:**

Maintain five variables for max profit at the end of each day,

0TransactionHold0 = 0

1TransactionHold0 = 0

1TransactionHold1 = Integer.Min\_value

2TransactionHold0 = 0

2TransactionHold1 = Integer.Min\_value

Go through each price, and calculate the vals

2TransactionHold0 = max (2TransactionHold0 (hold), 2TransactionHold1 – price(sell))

2TransactionHold1 = max (2TransactionHold1 (hold), 1TransactionHold0 + price (sell))

1TransactionHold0 = max (1TransactionHold0 (hold), 1TransacctionHold1 + price (sell))

1TransacctionHold1 = max (1TransacctionHold1 (hold), 0TransactionHold0 - price (buy))

Return 2TransactionHold0

# Buy Sell Stocks – Infinite Transactions

As many transactions as possible.

**Solution:**

If the hold1 takes the max of Prev hold0 – price (buy), then it accumulates the transactions.

prevTransactionHold0 = 0

TransactionHold0 = 0

TransactionHold1 = Integer.Min\_value.

Go through each price, and calculate the vals

prevTransactionHold0 = TransactionHold0;

TransactionHold0 = max (TransactionHold0 (hold), 1TransacctionHold1 + price (sell))

TransactionHold1 = max (TransactionHold1 (hold), prevTransactionHold0 - price (buy))

Return 1TransactionHold0

# Buy Sell Stocks – K Transactions

Specified number of transactions allowed.

**Solution:**

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

So, maintain two arrays for holding the values.

Hold0 array filled with 0 with k + 1 size.

Hold1 array filled with -1 with k + 1 size. To start with base condition

Go through each price, and calculate the vals

For each price, j = k, j until 0

Hold0[j] = max (hold0[j], hold1[j] + price) sell

Hold1[j] = max (hold1[j], hold0[j-1] - price) buy

# Buy Sell Stocks – Infinite Transactions + Cool Down

Wait one day before starting new transaction.

**Solution:**

If the hold1 takes the max of Prev hold0 – price (buy), then it accumulates the transactions.

prevTransactionHold0 = 0

TransactionHold0 = 0

TransactionHold1 = Integer.Min\_value.

Go through each price, and calculate the vals

prevPrevTransactionHold0 = prevTransactionHold0

prevTransactionHold0 = TransactionHold0;

TransactionHold0 = max (TransactionHold0 (hold), 1TransacctionHold1 + price (sell))

TransactionHold1 = max (TransactionHold1 (hold), prevPrevTransactionHold0 - price (buy))

Return TransactionHold0

# Buy Sell Stocks – Infinite Transactions + Transaction Fee

Pay fee for each time

**Solution:**

Reduce the fee from profit;

If the hold1 takes the max of Prev hold0 – price (buy), then it accumulates the transactions.

prevTransactionHold0 = 0

TransactionHold0 = 0

TransactionHold1 = Integer.Min\_value.

Go through each price, and calculate the vals

prevTransactionHold0 = TransactionHold0;

TransactionHold0 = max (TransactionHold0 (hold), 1TransacctionHold1 + price (sell))

TransactionHold1 = max (TransactionHold1 (hold), prevTransactionHold0 - price (buy) - fee)

Return 1TransactionHold0

**NOTE:** Take out while buying to avoid overflow…

# AddTwoNumbers (LinkedList)

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

**Solution:**

Create new list nodes to store result.Only one loop until either one is available, l1 or l2 or carry.

If l1 is null, val is zero. L2 is null, val is zero.

Sum = l1.val + l2.val + carry

currentVal = sum %10

carry = sum / 10

# Binary Addition (2 Strings)

a = "11"

b = "1"

Return "100"

**Solution:**

i = size of a – 1 (lsb)

j = size of b -1 (msb)

Only one loop until the end of a or b

Int sum = 0

If a is not done && a[i] == 1

Sum ++

If b is not done && b[i] == 1

Sum ++

Sum += carry

Carry = sum / 2;

Insert at beginning of result string (Char) (sum %2) + ‘0’

# Average of Binary Tree Levels

Return the average value of the nodes on each level in the form of an array.

Example 1:

Input:

3

/ \

9 20

/ \

15 7

Output: [3, 14.5, 11]

**Solution:**

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

Given a binary tree, determine if it is height-balanced.

**Solution:**

Condition: a node is balanced if left subtree and right subtree is balanced + number of nodes in right subtree – left subtree <= 1

Create a wrapper to hold isBalanced, height.

Report height from bottom with Max (left subtree height, right subtree height) + 1 (current level)

# BinaryTreeLevelOrderTraversal

For example:

Given binary tree [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

return its level order traversal as:

[

[3],

[9,20],

[15,7]

]

**Solution:**

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

/ \

2 3

\

5

All root-to-leaf paths are:

["1->2->5", "1->3"]

**Solution:**

Use recursive helper,

Base condition root == null,

Then add root, to a list of current nodes.

Do (required operation) when left == null && right == null.

Creating the string of paths.

If (left != null) {

Do left,

Remove current nodes.size() – 1.

}

If (right != null) {

Do right,

Remove current nodes.size() – 1.

}

# CheckSubTree

3

/ \

4 5

/ \

1 2

/

0

Given tree t:

4

/ \

1 2

Return false.

**Solution:**

Subtree:

If (t1.val is same t2.val && equals tree (t1, t2)

Return true;

Else

check subtree (t1.left, t2) OR check subtree(t1.right, t2)

Equals Tree:

Base condition: true

t1 == null && t2 == null

Base condition: False

t1 == null && t2 != null, t1 != null && t2 == null

T1.val == t2. Val && Equals Tree(t1.left, t2.left) && Equals Tree(t1.right, t2.right)

# Diameter of a Binary Tree

Given a binary tree

1

/ \

2 3

/ \

4 5

Return 3, which is the length of the path [4,2,1,3] or [5,2,1,3].

**Solution:**

Max of (Max height of left sub tree, Max height of right subtree)+ 1.

# ClimbingStairs

Input: 3

Output: 3

Explanation: There are three ways to climb to the top.

1 step + 1 step + 1 step

1 step + 2 steps

2 steps + 1 step

**Solution:**

Fibonacci sequence

0 step - 1

step – 1

step – 2

Loop through from 3 steps to k steps

3 steps = 2 step + 1 step

1step = 2 step;

2step = 3 step;

# Delete LinkedList Node

delete a node (except the tail) in a singly linked list, given only access to that node.

**Solution**:

Copy the value from the next node and modify the current pointer to the next one

# ExcelTitle

Positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

1-> A

2-> B

3-> C

...

26 -> Z

27 -> AA

28 -> AB

**Solution:**

While(n >0)

n--

Char c = (char) n %26 + ‘A’ 🡪 inorder to add from A, reduce n by 1 before

Next n = n / 26;

Reverse the chars formed

# ExcelTitleNumber

Given a title get the number.

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

**Solution:**

For each char c,

Result = Result \* 26 + ((int) c – ‘A’ + 1)

# FactorialTrailingZeros

integer n, return the number of trailing zeroes in n!

**Solution:**

Zero’s are always produced by 2 \* 5. Hence count how many 5’s are present. That many Zeros will be present.

So, n/5 + recursive (n/5) until zero

# Find Disappeared Numbers

Some elements appear twice and others appear once.

Input:

[4,3,2,7,8,2,3,1]

Output:

[5,6]

**Solution:**

Go through each element in the array and consider that as the index and make it negative. Offset for 1 for index position.

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.

for(int i=0;i<arr.length;i++)

int val = Math.abs(arr[i]) -1;

if(arr[val] > 0)

arr[val] = -arr[val];

for(int i=0;i<arr.length;i++)

if(arr[i] <0)

result.add(i + 1)

# FindAnagramMappings

For example, given

A = [12, 28, 46, 32, 50]

B = [50, 12, 32, 46, 28]

We should return

[1, 4, 3, 2, 0]

**Solution:**

Go through b and create a map of number to pos.

Then go through a, get the element from map and put the val in the result

# FindMissingNumber

Input: [3,0,1]

Output: 2

Example 2

Input: [9,6,4,2,3,5,7,0,1]

Output: 8

**Solution 1**: XOR

a^b^b =a.. hence

result = 0

For each element I in array from 0 postition

result = result ^ i ^ arr[i]

finally do the last index which would return the element that’s missing.

Return Result ^ arr len + 1 -1(for index)

**Solution 2:** Total Sum and subtraction from actual sum

N \* (n-1) / 2 – sum of all the elements in the array

# Single Number

an array of integers, every element appears twice except for one.

**Solution**:

XOR all the elements. Last result is the number.

# FirstUniqueCharacterInString

Examples:

s = "leetcode"

return 0.

s = "loveleetcode",

return 2.

**Solution**:

Use an array of size 26 and char integer val as the index.

Increment the frequency every time you see that char.

Go through the array and when a frequency is 1, return that char.

# HammingDistance

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

**Solution**:

A XOR B = no of bits that are different

Int result = 0

While (bitset > 0)

Result ++;

Bitset = bitset & (bitset – 1)

# Happy Number:

A happy number is a number defined by the following process:

Starting with any positive integer, replace the number by the sum of the squares of its digits,

and repeat the process until the number equals 1 (where it will stay),

**Solution**:

Necessary to maintain a hashset to track the seen numbers so that we don’t go into the infinite loop.

Get the digits and then calculate the square of sum

While(num > 0)

Digit = Num % 10

Num = num / 10;

Foreach (digit : digits)

Sum = sum + digit \* digit

If sum == num return happy;

Else go loop(sum)

# HouseRobber

constraint stopping you from robbing each of them is that adjacent houses have security system connected

**Solution**:

Dp array contains the max until that house.

Dp[0] = num[0]

Dp[1] = num[1]

I = 2

While(i<num.length)

MaxWithPrevprevhouse = num[i] + vals[i-2];

MaxWithprevprevprevhouse = nums[i] +( i-3 <= 0) ? 0 : vals[i-3]

currMax = max (MaxWithPrevprevhouse, MaxWithprevprevprevhouse)

totalMax = max (totalMax, currMax)

return totalMax

# IntegerToRoman

**Solution**:

X – 10, L – 50, C – 100, D – 500, M - 10000

int[] values = { 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };

String[] strs = { "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" };

Go from left to right in the values. If you can subtract the num, subtract and add the string.

For(Value : values)

while(num >= Value)

Num = num – value;

Result.append(stringofvalue);

# Island Perimeter

Example:

[

[0,1,0,0],

[1,1,1,0],

[0,1,0,0],

[1,1,0,0]

]

Answer: 16

**Solution**:

Go through each element, if zero then count how many ones you can see. left, right, up, down

If one, increase the count for boundaries. If I ==0, j==0, or I = length, j = length.

# Judge Circle

Example 1:

Input: "UD"

Output: true

Example 2:

Input: "LL"

Output: false

**Solution**:

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

Examples:

Input: S = "a1b2"

Output: ["a1b2", "a1B2", "A1b2", "A1B2"]

**Solution**:

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

**Solution**:

Slow, fast counter.

While(fast.next != null && fast.next.next != null)

If(slow == fast)

Break;

Slow = head

While(slow != fast)

Slow = slow.next;

Fast = fast.next;

Return slow;

# LisenceKeyFormatting

Example 2:

Input: S = "2-5g-3-J", K = 2

Output: "2-5G-3J"

**Solution**:

Take the string s, replace the – with “”

List<String> keys;

For (i = length; i – k >= 0; I = I -k)

Keys.add( s.substring(i-k, i))

Keys reverse

if(i>0)

mylist.add(S.substring(0, i));

String.join(“-”, keys);

# LongestCommonPrefix

Write a function to find the longest common prefix string amongst an array of strings.

Solution:

Take first word as lcp.

Iterate through the other words

For each char in lcp until the smaller length string.

If char doesn’t match, then substring until the index and make it lcp.

If index is 0, lcp is “” return.

If( lcp is longer than string)

Lcp = lcp.substring(0, index from previous exit)

Return lcp

# LongestUnivaluePath

Input:

1

/ \

4 5

/ \ \

4 4 5

Output:

2

**Solution**:

Pass in root value and current node. for root, it’s the same node and its same value.

But, first calculate what your sub trees tell you. Max is max of current max, left max + right max.

Then send above your height.

Let children decide to return the height. You just add them and find max.

getLengthRecursive(current, root.val)

if(current == null)

return 0;

left = getLengthRecursive(current.left, current.val)

right = getLengthRecursive(current.right, current.val)

len = max(len, left + right);

if(current.val == root.val)

return Math.max (left, right) +1;

else return 0;

# MajorityElement

The majority element is the element that appears more than ⌊ n/2 ⌋ times.

**Solution**:

If a number is greater than n/2 times, its count always going be greater than others.

For(int i: nums)

{

If(count == 0)

Result = n

Count++;

Else if(i== num)

Count ++;

Else

Count –

}

# MaximumDepthOfBinaryTree

3

/ \

9 20

/ \

15 7

return its depth = 3.

**Solution**:

Max depth is Max(left , right) + 1

Max(getMax(node.left), getMax(node.right)) + 1;

# MaximumSubarraySum

contiguous subarray within an array (containing at least one number) which has the largest sum.

[-2,1,-3,4,-1,2,1,-5,4],

the contiguous subarray [4,-1,2,1] has the largest sum = 6.

**Solution:**

When the sum is negative, and less than the current number, ignore the sum and use the current num as the sum

For (int i=0; i<nums.length; i++)

If(sum < 0 && sum < nums[i])

Sum = nums[i];

else

Sum = sum + nums[i];

maxSum = max(maxsum, sum)

# MeetingRooms

Determine if a person could attend all meetings.

[[0, 30],[5, 10],[15, 20]],

return false.

**Solution:**

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

Two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

**Solution**:

Start from the end. Get the end of arr1 = write index, m as I, n as j

Do until I and j and greater than 0.

While(j>0 && i>0)

{

If(arr[i] > arr[j])

{

Arr[writeIndex--] = arr[i--];

}

Else

{

Arr[writeIndex--] = arr[j--];

}

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.

While(j>0)

{

Arr[writeIndex--] = arr[j--];

}

# MinCostClimbingStairs

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.

Example 1:

Input: cost = [10, 15, 20]

Output: 15

Explanation: Cheapest is start on cost [1], pay that cost and go to the top.

Example 2:

Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]

Output: 6

**Solution**:

Create a dp array of input size + 1 to store min at the point.

// calculate how much cost to get to current. Either from -1 position or -2 position

Dp[0] = 0; to get to zero position – 0 cost.

Dp[1] = 0; to get to one position – 0 cost

For(int I = 2; i<input.length; i++)

{

Dp[i] = Min(dp[i-1] + input[i-1], dp[i-2] + input[i-2])

}

Return dp [input.length +1]

# MinimumAbsDifferenceInBST

find the minimum absolute difference between values of any two nodes.

Example:

Input:

1

\

3

/

2

Output:

1

**Solution**:

Use upperbound and lower bound and use the BST property.

At each node, you need the lowerbound and upperbound

Recurse(root.left, null, root)

Recurse(root.right, root, null)

Recurse(current, leftbound, rightbound)

If(current == null)

Return

If(leftBound!=null)

Calculate min

If(rightbound!=null)

Calculate min

Recurse (current.left, leftBound, current);

Recurse (current.right, current, rightBound);

# MinStack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

push(x) -- Push element x onto stack.

pop() -- Removes the element on top of the stack.

top() -- Get the top element.

getMin() -- Retrieve the minimum element in the stack.

**Solution**:

Maintain a min value.

Push x: when a value is less equal to min, push the min & update the min to x; push x anyway;

Pop: pop anyway. When the pop is equal to min, pop again and set in min.

Top: stack.top

Min: minval

# MoveZeros

nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

**Solution**:

Write index= 0

While(until end)

{

While(its zero && until end)

Count zeros

If(end)

Break

Arr[writeIndex++] = arr[readIndex++]

}

While(start from end, until length – zeros count)

{

Arr[index] = 0;

}

# MovingAverage

a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

**Solution**:

Maintain a queue.

When queue size is smaller than window size, sum = sum + num; avg = sum / size;

When queue size is large, remove element, subtract from sum, add newnum, calculate new sum and return avg

# FindNthDigit: -- TODO

# Hamming Weight:

the 32-bit integer ’11' has binary representation 00000000000000000000000000001011,

so the function should return 3.

**Solution**:

Int Bitset = 0;

While(num > 0)

{

Bitset ++;

num = num & (num – 1); // figures out the next bit

(or)

Bitset = bitset + (num & 1)

Num = num >> 1; //right shift by 1

}

# PaintFence

There is a fence with n posts, each post can be painted with one of the k colors.

You have to paint all the posts such that no more than two adjacent fence posts have the same color.

numWays(int n, int k)

**Solution**:

Dp[n]

Dp[0] – k

Dp[1] – k \* k

For(i=2 untill end)

Dp[i] – dp[ I - 1] \*( k -1) + dp(i- 2) \* (k -1)

Return dp[n]

# PaintHouses

There are a row of n houses, each house can be painted with one of the three colors:

red, blue or green. The cost of painting each house with a certain color is different.

You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a n x 3 cost matrix.

For example, costs[0][0] is the cost of painting house 0 with color red;

costs[1][2] is the cost of painting house 1 with color green, and so on...

Find the minimum cost to paint all houses.

Int minCost(int[][] costs)

**Solution**:

Min Cost of painting a house with one color = cost of that color + mincost ( previous house color1, previous house color2)

So, from second house, calculate the min.

Cost[i][0] = Cost[i][0] + min(Cost[i-1][1], Cost[i-1][2])

Cost[i][1] = Cost[i][1] + min(Cost[i-1][0], Cost[i-1][1])

Cost[i][2] = Cost[i][2] + min(Cost[i-1][0], Cost[i-1][1])

No extra dp array needed as the current one is used to store the dp vals.

# PalindromeNumber

whether an integer is a palindrome.

**Solution**:

Check if x < 0 or x % 10 == 0 return false.

While (x > rev) // doing until half in even, or one more time if odd. So rev is more than x.

Convert x into rev by

Rev = rev \* 10 + x /10;

X = x / 10;

Return x == rev (Even case) || x = rev /10 (oddcase)

# PathSumIII – Total Paths to a sum

root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

10

/ \

5 -3

/ \ \

3 2 11

/ \ \

3 -2 1

Return 3. The paths that sum to 8 are:

5 -> 3

5 -> 2 -> 1

-3 -> 11

**Solution**:

Total paths in tree:

Total path from node +

Total paths in tree(node.left, sum, targetsum) +

Totoal Paths in tree(node.right, sum, targetsum)

Totalpaths totalPathsfromNode(node, sum, targetsum)

if(node == null)

return 0;

totalpath = 0;

sum = sum + node.val

if(sum == targetsum)

totalPAth++;

// iterate further because of negative numbers.

totalpath = totalpath + totalPathsfromNode(node.left, sum, targetSum);

totalpath = totalpath + totalPathsfromNode(node.right, sum, targetSum);

return totalPath;

# PlusOne

Add plus one to integer represented in a array of digits

**Solution**:

Carry = 1

For(I = lengthofarr -1 ; i>0; i--)

{

Sum = sum + arr[i] + carry;

Arr[i] = sum %10;

Carry = sum /10;

}

If carry >0

Create a new array of length + 1

Copy carry to pos 0,

Copy rest of the arr from 1 position

# Power of 3, 2

**Solution**:

Power of 2 – n>0, n & n-1 == 0

Power of 3, n >0 while n >0, n %3 != 0 return false. Else loop with n = n / 3

# RangeSumQueryImmutable

Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1

sumRange(2, 5) -> -1

sumRange(0, 5) -> -3

**Solution**:

Create a new array and Calculate sum until that index.

GetSum:

if lowerbound == 0 return getVal[upperbound]

else getVal[upperbound] - getval[lowerbound - 1]

# ReadNCharsfromRead4

int read4(char \*buf) reads 4 characters at a time from a file.

public int read(char[] buf, int n)

**Solution**:

Boolean eof = false;

Int totalcount = 0;

Temp[] = new temp[4];

While(!eof &&totalCount == n)

{

Int localcount = read(temp);

If(localcount < 4)

Eof = true;

//localcount is minimum of chars available in file or max of possible to read remaining

int count = min (localcount, n-total)

//copy chars

for (int i = 0; i < count; i++)

buf[total++] = tmp[i];

}

# RemoveDuplicatesInSortedArray

Given a sorted array, remove the duplicates in-place such that each element appears only once and return the new length.

**Solution**:

Int writeIndex = 0;

Int readIndex = 0;

While(readIndex < arr.length)

{

arr[writeIndex] == arr[readIndex]

While(arr[writeIndex] == arr[readIndex])

readIndex++;

writeIndex++;

}

Return writeIndex

# RepeatedStringMatch

Two strings A and B, find the minimum number of times A has to be repeated such that B is a substring of it

A = "abcd" and B = "cdabcdab". Return 3

**Solution**:

Keep appending to the end, until the indexof is less than 0. Meaning no match. But, if the length of concat string minus A String is greater than B, exit

StringBuilder sb = new StringBuilder();

sb.append(A);

int count = 1;

while (sb.indexOf(B) < 0) {

if (sb.length() - A.length() > B.length()) {

return -1;

}

sb.append(A);

count++;

}

# ReverseInteger

Example 2:

Input: -123

Output: -321

**Solution**:

Use long to handle the overflow. But return 0, if it’s beyond the integer max and min value.

Use div and multiply to reverse

While(x>0)

{

Rev = rev \* 10 + x %10;

X = x % 10;

if (rev > int.max || rev < int.min)

Return 0;

}

# ReverseLinkedList

**Solution**:

1. Recursive:

if (head == null || head.next == null)

return head;

ListNode lasthead = reverseList(head.next);

head.next.next = head;

head.next = null;

return lasthead;

1. Iterative:

ListNode prev = null;

ListNode node = head;

while (node.next != null) {

ListNode next = node.next;

node.next = prev;

prev = node;

node = next;

}

node.next = prev;

return node;

# ReverseVowelsOfString

Given s = "leetcode", return "leotcede".

**Solution**:

More like palindrome. Two pointers one on left, one on right. Left ++ until you see a vowel in a map. Same as the otherside. Swap with temp variable and go left++, right -- finish until left < right pointer

# RomanToInteger

A roman numeral, convert it to an integer.

map.put('I', 1);

map.put('V', 5);

map.put('X', 10);

map.put('L', 50);

map.put('C', 100);

map.put('D', 500);

map.put('M', 1000);

**Solution**:

Declare prev = 0

Go from right to left.

If the value of char is less than prev, subtract from the sum.

Else sum = sum + val. Update the prev to current val.

# SentenceSimilarity

"great acting skills" and "fine drama talent" are similar, if the similar word pairs are pairs = [["great", "fine"], ["acting","drama"], ["skills","talent"]].

areSentencesSimilar(String[] words1, String[] words2, String[][] pairs)

**Solution**:

Put the pairs in k => v, v => k mapping. Then go through the words, see if they are same continue, check if both exists in the map, key -> value + value -> key

# SortLinkedList – TODO

# Strobogrammatic number

strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to determine if a number is strobogrammatic.

**Solution**:

map.put('6', '9');

map.put('9', '6');

map.put('0', '0');

map.put('1', '1');

map.put('8', '8');

Similar to palindrome. check if the numbers are present and the corresponding has the same value.

# StrStr

Implement strstr function

**Solution**:

Use indexOf. Also, Check if haystack is larger than needle, and needle is greater than 0.

# SumOfLeftLeaves

Find the sum of all left leaves in a given binary tree.

Example:

3

/ \

9 20

/ \

15 7

There are two left leaves in the binary tree, with values 9 and 15 respectively. Return 24.

**Solution**:

Inorder traversal with calculation in the middle. check if left!= null and its children are null.

Recurse(Node)

If(node == null)

Return;

Recurse(node.left);

If(node.left!=null && node.left.left == null && node.left.right == null)

Sum = sum + node.left.val;

Recurse(node.right)

# Sum & Difference of Two Integers

Without +, - operators

**Solution**:

**Sum**:

While(b>0) {

Int carry = a & b;

a = a ^ b;

b = carry << 1;

}

Return a;

**Difference**:

While(b>0) {

Int borrow = (-a) & b;

a = a ^ b;

b = borrow << 1;

}

Return a;

# Toeplitz Matrix

Toeplitz if every diagonal from top-left to bottom-right has the same element.

Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]

Output: True

Explanation:

1234

5123

9512

**Solution**:

Start with 0 and go until row – 1, columns – 1. Check if matrix[i][j] == matrix[i+1][j+1]. If anything, not the same, return false

# TwoSum

An array of integers, return indices of the two numbers such that they add up to a specific target.

**Solution**:

Put the diff in the map for each value. If the num is present then return true. Finally return false.

# TwoSumII

array of integers that is already sorted in ascending order

Solution:

Have two pointers, if sum greater than target, j-- else i++

# TwoSumIII

Design and implement a TwoSum class. It should support the following operations: add and find.

add - Add the number to an internal data structure.

find - Find if there exists any pair of numbers which sum is equal to the value.

For example,

add(1); add(3); add(5);

find(4) -> true

find(7) -> false

Solution:

Create a map of number -> count of how many integers seen.

# TwoSumIV

a Binary Search Tree and a target number, return true if there exist two elements in the BST such that their sum is equal to the given target.

Example 1:

Input:

5

/ \

3 6

/ \ \

2 4 7

Target = 9

**Solution**:

1. Inorder traversal and get the list. Use two pointers.
2. Use a hashset, put the diff’s when visiting a root. The next time see the nodes in the hashset.
3. Use search function to see if its there. Visit node. search if the diff exists

# ValidAnagram

Given two strings s and t, write a function to determine if t is an anagram of s.

For example,

s = "anagram", t = "nagaram", return true.

s = "rat", t = "car", return false.

**Solution**:

Create a hashmap / int[256], put the frequency count for one word. For the next word, reduce the frequency. Then go through the hasmap and see if any value is not 0. Return false.

# ValidPalindromeII

Given a non-empty string s, you may delete at most one character. Judge whether you can make it a palindrome.

Example 1:

Input: "aba"

Output: True

Example 2:

Input: "abca"

Output: True

**Solution**

Two pointers, left and right. If they are equal continue. If there is a mismatch, then have two options.

Return Increment left by one and calculate the palindrome. Or increment right by one and calculate the palindrome.

# Valid Square

Input:

[

"abcd",

"bnrt",

"crmy",

"dtye"

]

Output:

True

**Solution**:

Use it more like a matrix. Check if columns are not greater than rows, rows not greater than columns

And see if the char is not same

For(int i=0;i<words.length;i++)

For(int j=0;j<words.get(i).length;j++)

If(j>= words.length || words.get(j).length <= i || words.get(i).charAt(j) != words.get(j).charAt(i)

Return false;

# WellFormedParanthesis

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

**Solution**:

Use a stack to push the brackets only if its starting. If its ending braces, pop the stack and match if they are the same open and close.

If(isstarting(brace))

Push;

Else

If(stack.empty())

Return false;

!Ismactching(Brace, Stack.pop)

Retun false;

Return stack.isEmpty();