Table of Contents

[FindMinSortedArray 3](#_Toc510555341)

[FindMinSortedRotatedArray2 3](#_Toc510555342)

[SearchSortedRotatedArray 3](#_Toc510555343)

[SearchSortedRotatedArray2 4](#_Toc510555344)

[SquareRoot 4](#_Toc510555345)

[Find Peak Element 4](#_Toc510555346)

[First Bad Version 5](#_Toc510555347)

[GuessNumberHigherOrLower 5](#_Toc510555348)

[Buy Sell Stocks – One Transaction 5](#_Toc510555349)

[Buy Sell Stocks – Two Transaction 6](#_Toc510555350)

[Buy Sell Stocks – Infinite Transactions 7](#_Toc510555351)

[Buy Sell Stocks – K Transactions 7](#_Toc510555352)

[Buy Sell Stocks – Infinite Transactions + Cool Down 7](#_Toc510555353)

[Buy Sell Stocks – Infinite Transactions + Transaction Fee 8](#_Toc510555354)

[AddTwoNumbers (LinkedList) 9](#_Toc510555355)

[Binary Addition (2 Strings) 9](#_Toc510555356)

[Average of Binary Tree Levels 9](#_Toc510555357)

[HeightBalancedBinaryTree 10](#_Toc510555358)

[BinaryTreeLevelOrderTraversal 10](#_Toc510555359)

[Root-to-leaf paths 11](#_Toc510555360)

[CheckSubTree 11](#_Toc510555361)

[Diameter of a Binary Tree 12](#_Toc510555362)

[ClimbingStairs 13](#_Toc510555363)

[Delete LinkedList Node 13](#_Toc510555364)

[ExcelTitle 13](#_Toc510555365)

[ExcelTitleNumber 14](#_Toc510555366)

[FactorialTrailingZeros 14](#_Toc510555367)

[Find Disappeared Numbers 14](#_Toc510555368)

[FindAnagramMappings 15](#_Toc510555369)

[FindMissingNumber 15](#_Toc510555370)

[FirstUniqueCharacterInString 16](#_Toc510555371)

[HammingDistance 16](#_Toc510555372)

[Happy Number: 17](#_Toc510555373)

[HouseRobber 17](#_Toc510555374)

[IntegerToRoman 18](#_Toc510555375)

[Island Perimeter 18](#_Toc510555376)

[Judge Circle 18](#_Toc510555377)

[LetterCasePermutation 19](#_Toc510555378)

[LinkedListCycle 19](#_Toc510555379)

[LisenceKeyFormatting 20](#_Toc510555380)

[LongestCommonPrefix 20](#_Toc510555381)

# 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. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

**Solution:**

Fibonacci sequence

0 step - 1

1 step – 1

2 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

# 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