**ANKUSH SAINI**

DSA 150 DAYS CHALLENGE 450 QUESTIONS

* **DAY 16 :MATHS**

**1.SQRT(X)**

class Solution {

public:

int mySqrt(int x) {

long long l=0,h=x;

int result;

while(l<=h)

{

long long mid=(l+h)/2;

if(mid\*mid==x)

{

return mid;

}

else if(mid\*mid<x)

{

l=mid+1;

}

else

{

h=mid-1;

}

if(mid\*mid<x)

{

result=mid;

}

}

return result;

}

};

**2.PLUS ONE**

**NAÏVE SOLUTION:**

**Example 1:**

**Input:** digits = [1,2,3]

**Output:** [1,2,4]

**Explanation:** The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1,2,4].

**Example 2:**

**Input:** digits = [4,3,2,1]

**Output:** [4,3,2,2]

**Explanation:** The array represents the integer 4321.

Incrementing by one gives 4321 + 1 = 4322.

Thus, the result should be [4,3,2,2].

**Example 3:**

**Input:** digits = [9]

**Output:** [1,0]

**Explanation:** The array represents the integer 9.

Incrementing by one gives 9 + 1 = 10.

Thus, the result should be [1,0].

class Solution {

public:

vector<int> plusOne(vector<int>& digits) {

vector<int>v;

int i=digits.size()-1;

int carry=1;

while(i>=0)

{

v.push\_back((digits[i]+carry)%10);

carry=(digits[i]+carry)/10;

i--;

}

if(carry==1)

{

v.push\_back(1);

}

return vector<int>(v.rbegin(),v.rend());

}

};

**EFFICIENT SOLUTION :**

class Solution {

public:

vector<int> plusOne(vector<int>& digits) {

int i=digits.size()-1;

while(i>=0)

{

if(digits[i]<9)

{

digits[i]++;

return digits;

}

digits[i]=0;

i--;

}

digits.insert(digits.begin(),1);

return digits;

}

};

**3.ROMAN TO INTEGER**

class Solution {

public:

int romanToInt(string s) {

int result=0,prev\_result=0;

int i=s.length()-1;

while(i>=0)

{

int value=getValue(s[i]);

if(prev\_result>value)

{

result-=value;

}

else

{

result+=value;

}

prev\_result=value;

i--;

}

return result;

}

private:

int getValue(char s)

{

if(s=='I')

return 1;

else if(s=='V')

return 5;

else if(s=='L')

return 50;

else if(s=='X')

return 10;

else if(s=='C')

return 100;

else if(s=='D')

return 500;

else

return 1000;

}

};

**4.HAPPY NUMBER**

**Happy number doesn’t contain 4 while non happy number always contain 4 after some iteration**

Example 1:

Input: n = 19

Output: true

Explanation:

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

Example 2:

Input: n = 2

Output: false

class Solution {

public:

bool isHappy(int n) {

int sum=0;

while(sum!=4)

{

sum=0;

while(n>0)

{

sum+=pow((n%10),2);

n/=10;

}

n=sum;

if(sum==1)

return true;

}

return false;

}

};

* **DAY 17:ARRAYS**

**1.Count Odd Number in an range Interval**

Example 1:

Input: low = 3, high = 7

Output: 3

**Explanation:** The odd numbers between 3 and 7 are [3,5,7].

Example 2:

Input: low = 8, high = 10

Output: 1

**Explanation:** The odd numbers between 8 and 10 are [9].

class Solution {

public:

int countOdds(int low, int high) {

int count=0;

if(low%2==0)

low++;

for(int i=low;i<=high;i+=2)

{

count++;

}

return count;

}

};

**2.Rectangle overlap**

An axis-aligned rectangle is represented as a list [x1, y1, x2, y2], where (x1, y1) is the coordinate of its bottom-left corner, and (x2, y2) is the coordinate of its top-right corner. Its top and bottom edges are parallel to the X-axis, and its left and right edges are parallel to the Y-axis.

Two rectangles overlap if the area of their intersection is positive. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two axis-aligned rectangles rec1 and rec2, return true if they overlap, otherwise return false.

Example 1:

Input: rec1 = [0,0,2,2], rec2 = [1,1,3,3]

Output: true

Example 2:

Input: rec1 = [0,0,1,1], rec2 = [1,0,2,1]

Output: false

Example 3:

Input: rec1 = [0,0,1,1], rec2 = [2,2,3,3]

Output: false

class Solution {

public:

bool isRectangleOverlap(vector<int>& rec1, vector<int>& rec2) {

return (rec2[1]>=rec1[3]||rec1[1]>=rec2[3]||rec2[0]>=rec1[2]||rec1[0]>=rec2[2])?false:true;

}

};

**3.Add Digits**

Example 1:

Input: num = 38

Output: 2

Explanation: The process is

38 --> 3 + 8 --> 11

11 --> 1 + 1 --> 2

Since 2 has only one digit, return it.

class Solution {

public:

int addDigits(int num) {

int sum=0;

do

{

sum=0;

while(num>0)

{

sum+=(num%10);

num/=10;

}

num=sum;

}while(sum>=10);

return sum;

}

};

**4.Maximum Product of three numbers**

class Solution {

public:

int maximumProduct(vector<int>& nums) {

int n = nums.size();

sort(nums.begin(), nums.end());

int max\_product = nums[n - 1] \* nums[n - 2] \* nums[n - 3];

int alternate\_max\_product = nums[0] \* nums[1] \* nums[n - 1];

return max(max\_product, alternate\_max\_product);

}

};

**5.Excel Sheet Column Number**

Example 1:

Input: columnTitle = "A"

Output: 1

Example 2:

Input: columnTitle = "AB"

Output: 28

Example 3:

Input: columnTitle = "ZY"

Output: 701

class Solution {

public:

int titleToNumber(string s) {

int res=0;

for(int i=0;i<s.size();i++)

{

int value=s[i]-'A'+1;

res=res\*26+value;

}return res;

}

};

* **DAY 18 : BITS**

**1.Add Binary**

Example 1:

Input: a = "11", b = "1"

Output: "100"

class Solution {

public:

string addBinary(string a, string b) {

string sum="";

int i=a.size()-1,j=b.size()-1;

int carry=0;

int digit\_sum=0;

while(i>=0 || j>=0 || carry)

{

digit\_sum=carry;

if(i>=0)

{

digit\_sum+=a[i--]-'0';

}

if(j>=0)

{

digit\_sum+=b[j--]-'0';

}

sum=to\_string(digit\_sum%2)+sum;

carry=digit\_sum/2;

}

return sum;

}

};

**2.Counting Bits**

Input: n = 2

Output: [0,1,1]

Explanation:

0 --> 0

1 --> 1

2 --> 10

Example 2:

Input: n = 5

Output: [0,1,1,2,1,2]

Explanation:

0 --> 0

1 --> 1

2 --> 10

3 --> 11

4 --> 100

5 --> 101

**NAÏVE APPROACH :**

class Solution {

public:

vector<int> countBits(int n) {

// vector<int>v(n+1,0);

// for(int i=1;i<=n;i++)

// {

// int num=i;

// while(num>1)

// {

// if(num%2==1)

// v[i]++;

// num=num/2;

// }

// ++v[i];

// }

// return v;

}

};

**EFFICIENT APPROACH: O(n)**

class Solution {

public:

vector<int> countBits(int n) {

vector<int>v(n+1);

v[0]=0;

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

{

if(i%2!=0)

{

v[i]=v[i/2]+1;

}

else

{

v[i]=v[i/2];

}

}

return v;

}

};

**4.Single Number**

Example 1:

Input: nums = [2,2,1]

Output: 1

Example 2:

Input: nums = [4,1,2,1,2]

Output: 4

Example 3:

Input: nums = [1]

Output: 1

class Solution {

public:

int singleNumber(vector<int>& nums) {

sort(nums.begin(),nums.end());

for(int i=1;i<nums.size();i=i+2)

{

if(nums[i]!=nums[i-1])

return nums[i-1];

}

return nums[nums.size()-1];

}

};

**5.Missing Number**

**NAÏVE APPROACH :**

Example 1:

Input: nums = [3,0,1]

Output: 2

Example 3:

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

Output: 8

class Solution {

public:

int missingNumber(vector<int>& nums) {

sort(nums.begin(),nums.end());

for(int i=0;i<=nums.size()-1;i++)

{

if(nums[i]!=i)

{

return i;

}

}

return nums.size();

}

};

**EFFICIENT APPROACH:**

class Solution {

public:

int missingNumber(vector<int>& nums) {

int missing=nums.size();

for(int i=0;i<nums.size();i++)

{

missing^=i^nums[i];

}

return missing;

}

};

**ANOTHER SOLUTION :**

class Solution {

public:

int missingNumber(vector<int>& nums) {

int n=nums.size();

int sum=(n\*(n+1))/2;

for(int i=0;i<nums.size();i++)

{

sum=sum-nums[i];

}

return sum;

}

};

**6.Number of 1 bits**

Example 1:

Input: n = 00000000000000000000000000001011

Output: 3

Explanation: The input binary string 00000000000000000000000000001011 has a total of three '1' bits.

**Method 1:**

class Solution {

public:

int hammingWeight(uint32\_t n) {

int res=0;

while(n>0)

{

if(n%2)

res++;

n/=2;

}

return res;

}

};

**Method 2: Bitwise Right Shift >>**

class Solution {

public:

int hammingWeight(uint32\_t n) {

int res=0;

while(n!=0)

{

res+=n&1;

n=n>>1;

}

return res;

}

};

**7.Hamming Distance**

Example 1:

Input: x = 1, y = 4

Output: 2

Explanation:

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

class Solution {

public:

int hammingDistance(int x, int y) {

int s=x^y;

int res=0;

while(s!=0)

{

res+=s&1;

s=s>>1;

}

return res;

}

};

**8.Reverse Bits**

Example 1:

Input: n = 00000010100101000001111010011100

Output: 964176192 (00111001011110000010100101000000)

Explanation: The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 00111001011110000010100101000000.

Example 2:

Input: n = 11111111111111111111111111111101

Output: 3221225471 (10111111111111111111111111111111)

Explanation: The input binary string 11111111111111111111111111111101 represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is 10111111111111111111111111111111.

**NAIVE APPRACH :**

class Solution {

public:

uint32\_t reverseBits(uint32\_t n) {

uint32\_t result1=0;

for(int i=31;i>=0;i--)

{

uint32\_t result2=n&1;

result2<<=i;

result1|=result2;

n=n>>1;

}

return result1;

}

};

**EFFICIENT APPROACH :**

class Solution {

public:

uint32\_t reverseBits(uint32\_t n) {

n=((n&0xffff0000)>>16)|((n&0x0000ffff)<<16);

n=((n&0xff00ff00)>>8)|((n&0x00ff00ff)<<8);

n=((n&0xf0f0f0f0)>>4)|((n&0x0f0f0f0f)<<4);

n=((n&0xcccccccc)>>2)|((n&0x33333333)<<2);

n=((n&0xaaaaaaaa)>>1)|((n&0x55555555)<<1);

return n;

}

};

* **DAY 19 : BS**

**1.Binary Search :**

**4.Valid Perfect Square :**

Example 1:

Input: num = 16

Output: true

Explanation: We return true because 4 \* 4 = 16 and 4 is an integer.

Example 2:

Input: num = 14

Output: false

Explanation: We return false because 3.742 \* 3.742 = 14 and 3.742 is not an integer.

class Solution {

public:

bool isPerfectSquare(int num) {

if(num==1 || num==0)

return true;

long long l=0,h=num/2;

while(l<=h)

{

long long m=(l+h)/2;

if(m\*m==num){return true;}

else if(m\*m>num){h=m-1;}

else{l=m+1;}

}

return false;

}

};

**5.Kth Missing Positive Number :**

Example 1:

Input: arr = [2,3,4,7,11], k = 5

Output: 9

Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th missing positive integer is 9.

Example 2:

Input: arr = [1,2,3,4], k = 2

Output: 6

Explanation: The missing positive integers are [5,6,7,...]. The 2nd missing positive integer is 6.

class Solution {

public:

int findKthPositive(vector<int>& arr, int k) {

int n=arr[arr.size()-1];

int i=0,j=0;

while(i<n)

{

if((i+1)!=arr[j])

{

k--;

if(k==0)

return i+1;

}

else

{

j++;

}

i++;

}

return arr[arr.size()-1]+k;

}

};

* **DAY 20 :Hashing**

**2.Ransom Note:**

Given two strings ransomNote and magazine, return true if ransomNote can be constructed by using the letters from magazine and false otherwise.

Each letter in magazine can only be used once in ransomNote.

Example 1:

Input: ransomNote = "a", magazine = "b"

Output: false

Example 2:

Input: ransomNote = "aa", magazine = "ab"

Output: false

Example 3:

Input: ransomNote = "aa", magazine = "aab"

Output: true

class Solution {

public:

bool canConstruct(string ransomNote, string magazine) {

int character[26]={0};

for(int i=0;i<magazine.size();i++)

{

character[magazine[i]-'a']++;

}

for(int i=0;i<ransomNote.size();i++)

{

if(character[ransomNote[i]-'a']==0)

{

return false;

}

else

{

character[ransomNote[i]-'a']--;

}

}

return true;

}

};

**3.Contain Duplicate :**

**Using Sorting (Naïve Approach):**

Example 1:

Input: nums = [1,2,3,1]

Output: true

Example 2:

Input: nums = [1,2,3,4]

Output: false

Example 3:

Input: nums = [1,1,1,3,3,4,3,2,4,2]

Output: true

class Solution {

public:

bool containsDuplicate(vector<int>& nums) {

sort(nums.begin(),nums.end());

for(int i=1;i<nums.size();i++)

{

if(nums[i]==nums[i-1])

return true;

}

return false;

}

};

**HashTable (Efficient Approach):**

class Solution {

public:

bool containsDuplicate(vector<int>& nums) {

unordered\_set<int>s;

for(auto n:nums)

{

if(s.find(n)!=s.end())

{

return true;

}

else

{

s.insert(n);

}

}

return false;

}

};

**5.Jewels and Stones :**

Example 1:

Input: jewels = "aA", stones = "aAAbbbb"

Output: 3

Example 2:

Input: jewels = "z", stones = "ZZ"

Output: 0

class Solution {

public:

int numJewelsInStones(string jewels, string stones) {

unordered\_map<char,int>um;

for(auto a:stones)

{

um[a]++;

}

int count=0;

for(auto a:jewels)

{

if(um.find(a)!=um.end())

{

count+=um[a];

}

}

return count;

}

};

**6.Unique Number of Occurrences :**

Example 1:

Input: arr = [1,2,2,1,1,3]

Output: true

Explanation: The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.

Example 2:

Input: arr = [1,2]

Output: false

Example 3:

Input: arr = [-3,0,1,-3,1,1,1,-3,10,0]

Output: true

class Solution {

public:

bool uniqueOccurrences(vector<int>& arr) {

unordered\_map<int,int>m;

for(auto a:arr)

{

m[a]++;

}

unordered\_set<int>s;

for(auto a:m)

{

if(s.find(a.second)!=s.end())

{

return false;

}

else

{

s.insert(a.second);

}

}

return true;

}

};

* **DAY 21 :String**

* **DAY 23 :Array**

**1.Fibbonacci Series :**

The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

Example 1:

Input: n = 2

Output: 1

Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.

Example 2:

Input: n = 3

Output: 2

Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.

Example 3:

Input: n = 4

Output: 3

Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.

class Solution {

public:

int fib(int n) {

// RECURSION

// if(n==0)

// {

// return 0;

// }

// if(n==1)

// {

// return 1;

// }

// return fib(n-1)+fib(n-2);

if(n==0 || n==1)

{

return n;

}

int first=0,second=1;

while(n>1)

{

int temp=first;

first=second;

second=temp+second;

n--;

}

return second;

}

};

**2.Min Cost Climbing Stairs :**

**3.Climbing Stairs :**

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2

Output: 2

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

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 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

class Solution {

public:

int climbStairs(int n) {

// DIRECTLY CAN DO USING THE CONCEPT OF FIBBONACCI SERIES

if(n==0||n==1)

{

return 1;

}

vector<int>v(n+1,0);

v[0]=v[1]=1;

for(int i=2;i<=n;i++)

{

v[i]=v[i-1]+v[i-2];

}

return v[n];

}

};

**5.Can Place Flower**

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

Example 1:

Input: flowerbed = [1,0,0,0,1], n = 1

Output: true

Example 2:

Input: flowerbed = [1,0,0,0,1], n = 2

Output: false

class Solution {

public:

bool canPlaceFlowers(vector<int>& flowerbed, int n){

int size=flowerbed.size();

for(int i=0;i<flowerbed.size() && n>0;i++)

{

if(flowerbed[i]==0 && (i==0 || flowerbed[i-1]==0) && (i==size-1 || flowerbed[i+1]==0))

{

flowerbed[i]=1;

n--;

}

}

return n<=0;

}

};

* **DAY 24 :Array**

1. **3sum :**

**3.Non Decreasing array :**

Given an array nums with n integers, your task is to check if it could become non-decreasing by modifying at most one element.

We define an array is non-decreasing if nums[i] <= nums[i + 1] holds for every i (0-based) such that (0 <= i <= n - 2).

Example 1:

Input: nums = [4,2,3]

Output: true

Explanation: You could modify the first 4 to 1 to get a non-decreasing array.

Example 2:

Input: nums = [4,2,1]

Output: false

Explanation: You cannot get a non-decreasing array by modifying at most one element.

class Solution {

public:

bool checkPossibility(vector<int>& nums) {

int atmost=0;

for(int i=1;i<nums.size();i++)

{

if(nums[i]<nums[i-1])

{

if(atmost==1)

{

return false;

}

if(i==1 || nums[i]>=nums[i-2])

{

nums[i-1]=nums[i];

}

else

{

nums[i]=nums[i-1];

}

atmost++;

}

}

return atmost<=1;

}

};

**4. Product of Array Except Self :**

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

Example 1:

Input: nums = [1,2,3,4]

Output: [24,12,8,6]

Example 2:

Input: nums = [-1,1,0,-3,3]

Output: [0,0,9,0,0]

**NAIVE APPROACH :**

class Solution {

public:

vector<int> productExceptSelf(vector<int>& nums) {

int n=nums.size();

vector<int>preffix\_product(n,1);

vector<int>suffix\_product(n,1);

vector<int>answer(n,0);

for(int i=1;i<nums.size();i++)

{

preffix\_product[i]=preffix\_product[i-1]\*nums[i-1];

}

for(int i=n-2;i>=0;i--)

{

suffix\_product[i]=suffix\_product[i+1]\*nums[i+1];

}

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

{

answer[i]=preffix\_product[i]\*suffix\_product[i];

}

return answer;

}

};

**EFFICIENT APPROACH :**

* **DAY 25: Array**

**1.Merge Intervals :**

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Example 2:

Input: intervals = [[1,4],[4,5]]

Output: [[1,5]]

Explanation: Intervals [1,4] and [4,5] are considered overlapping.

class Solution {

public:

vector<vector<int>> merge(vector<vector<int>>& intervals) {

if(intervals.empty())

return intervals;

sort(intervals.begin(),intervals.end());

vector<vector<int>>result;

// result.push\_back({intervals[0][0],intervals[0][1]});

// or

result.push\_back(intervals[0]);

for(int i=1;i<intervals.size();i++)

{

if(result.back()[1]>=intervals[i][0])

{

result.back()[1]=max(result.back()[1],intervals[i][1]);

}

else

{

result.push\_back({intervals[i][0],intervals[i][1]});

}

}

return result;

}

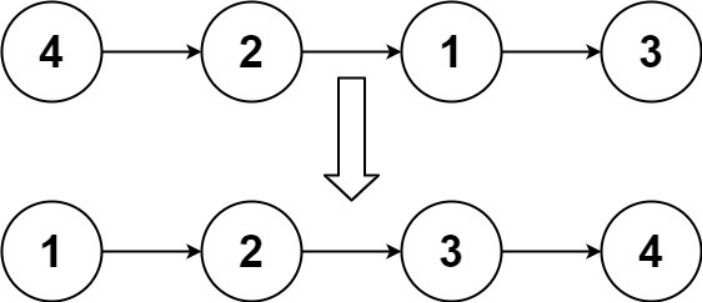
};

* **DAY 34 : LIST**

**1.Sort List :**

Given the head of a linked list, return the list after sorting it in *ascending order*.

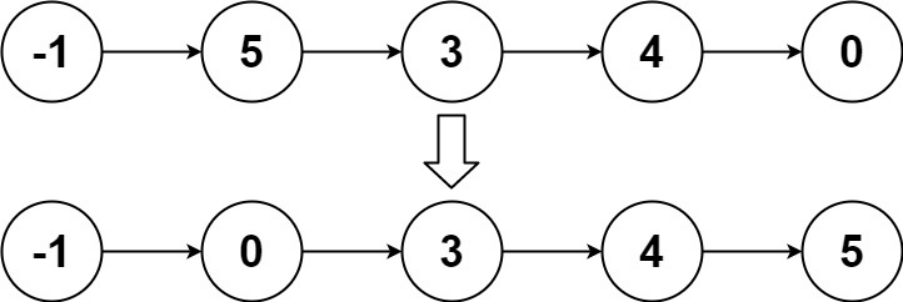
Example 1:



Input: head = [4,2,1,3]

Output: [1,2,3,4]

Example 2:



Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Example 3:

Input: head = []

Output: []

**EFFICIENT APPROACH :( Method -1) Time : O(n logn) Space : O(logn)**

class Solution {

public:

ListNode\* sortList(ListNode\* head){

ListNode\*a=NULL,\*b=NULL;

if(head==NULL || head->next==NULL )

{

return head;

}

divideList(head,a,b);

a=sortList(a);

b=sortList(b);

return merge(a,b);

}

ListNode\*merge(ListNode\*l1, ListNode\*l2)

{

if(l1==NULL)

{

return l2;

}

if(l2==NULL)

{

return l1;

}

ListNode\*result=NULL;

if(l1->val<=l2->val)

{

result=l1;

result->next=merge(l1->next,l2);

}

else

{

result=l2;

result->next=merge(l1,l2->next);

}

return result;

}

void divideList(ListNode\*head,ListNode\* &a,ListNode\* &b)

{

ListNode\*slow=head;

ListNode\*fast=head;

while(fast->next!=NULL && fast->next->next!=NULL)

{

slow=slow->next;

fast=fast->next->next;

}

a=head;

b=slow->next;

slow->next=NULL;

}

};

**EFFICIENT APPROACH : Time: O(N log N)   
Space : O(1)**

**2.Reorder List :**

You are given the head of a singly linked-list. The list can be represented as:

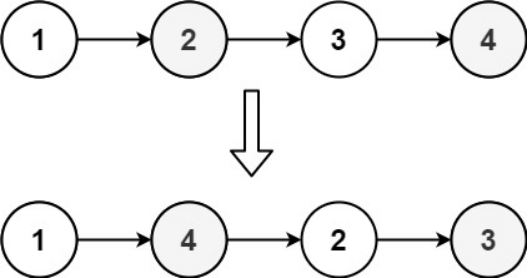
L0 → L1 → … → Ln - 1 → Ln

Reorder the list to be on the following form:

L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

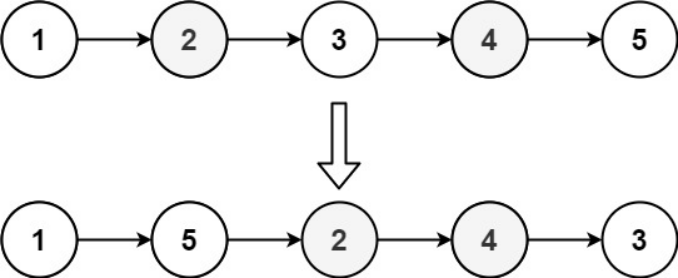
Example 1:



Input: head = [1,2,3,4]

Output: [1,4,2,3]

Example 2:



Input: head = [1,2,3,4,5]

Output: [1,5,2,4,3]

class Solution {

public:

void reorderList(ListNode\* head) {

if(!head || !head->next || !head->next->next)

{

return;

}

ListNode\*slow=middle(head);

ListNode\*reversed=reverse(slow->next);

slow->next=NULL;

merge(head,reversed);

}

public:

ListNode\*middle(ListNode\*head)

{

ListNode\*slow=head;

ListNode\*fast=head;

while(fast!=NULL && fast->next!=NULL)

{

slow=slow->next;

fast=fast->next->next;

}

return slow;

}

ListNode\*reverse(ListNode\*slow)

{

ListNode\*prev=NULL;

while(slow!=NULL)

{

ListNode\*nextNode=slow->next;

slow->next=prev;

prev=slow;

slow=nextNode;

}

return prev;

}

void merge(ListNode\*l1,ListNode\*l2)

{

while(l1 && l2)

{

ListNode\*l1\_next=l1->next;

ListNode\*l2\_next=l2->next;

l1->next=l2;

l2->next=l1\_next;

l1=l1\_next;

l2=l2\_next;

}

}

};

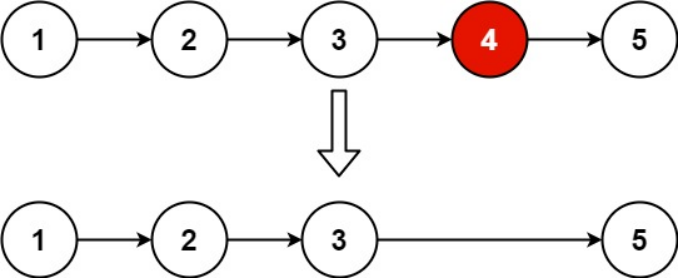
**3.Rotate List :**

* **DAY 31 : LIST**

**1.Remove Nth node from the list**

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Example 1:



Input: head = [1,2,3,4,5], n = 2

Output: [1,2,3,5]

Example 2:

Input: head = [1], n = 1

Output: []

Example 3:

Input: head = [1,2], n = 1

Output: [1]

class Solution {

public:

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

if(head->next==NULL && n==1 || head==NULL)

{

return nullptr;

}

int count=length(head);

if(count-n==0)

{

return head->next;

}

ListNode\*fast=head;

for(int i=1;i<count-n;i++)

{

fast=fast->next;

}

ListNode\*temp=fast->next;

if(temp!=NULL)

{

fast->next=temp->next;

delete temp;

}

else

{

fast->next=NULL;

}

return head;

}

int length(ListNode\*fast)

{

int count=0;

while(fast->next!=NULL && fast->next->next!=NULL)

{

fast=fast->next->next;

count+=2;

}

if(fast->next==NULL)

{

count+=1;

}

else

{

count+=2;

}

return count;

}

};

**2.Delete Node in a Linked List:**

**3.Remove Duplicates from sorted list II :**

Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list *sorted* as well.

Example 1:



Input: head = [1,2,3,3,4,4,5]

Output: [1,2,5]

Example 2:



Input: head = [1,1,1,2,3]

Output: [2,3]

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

if(head==NULL || head->next==NULL)

{

return head;

}

ListNode\*dummy=new ListNode(0);

dummy->next=head;

ListNode\*curr=head;

ListNode\*prev=dummy;

while(curr!=NULL && curr->next!=NULL)

{

if(curr->val==curr->next->val)

{

int duplicate = curr->val;

while(curr!=NULL && curr->val==duplicate)

{

ListNode\*temp=curr;

curr=curr->next;

delete temp;

}

prev->next=curr;

}

else

{

prev=curr;

curr=curr->next;

}

}

return dummy->next;

}

};

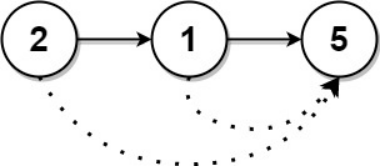
**4. Next Greater Node in linked list :**

You are given the head of a linked list with n nodes.

For each node in the list, find the value of the next greater node. That is, for each node, find the value of the first node that is next to it and has a strictly larger value than it.

Return an integer array answer where answer[i] is the value of the next greater node of the ith node (1-indexed). If the ith node does not have a next greater node, set answer[i] = 0.

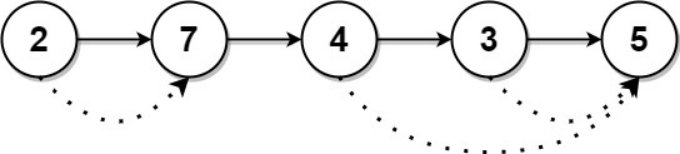
Example 1:



Input: head = [2,1,5]

Output: [5,5,0]

Example 2:



Input: head = [2,7,4,3,5]

Output: [7,0,5,5,0]

**NAIVE APPROACH :**

class Solution {

public:

vector<int> nextLargerNodes(ListNode\* head) {

vector<int>result;

ListNode\* curr=head;

while(curr->next!=NULL)

{

ListNode\*temp=curr->next;

while(temp!=NULL && temp->val<=curr->val)

{

temp=temp->next;

}

if(temp==NULL)

{

result.push\_back(0);

}

else

{

result.push\_back(temp->val);

}

curr=curr->next;

}

result.push\_back(0);

return result;

}

};

**EFFICIENT APPROACH :**

* **DAY 35 : Stack**

**1.Reverse polish Notation :**

**2.Min stack :**

* **DAY 36 : Stack**

1. **Next Greater Element :**

Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return the *next greater number* for every element in nums.

The next greater number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number.

Example 1:

Input: nums = [1,2,1]

Output: [2,-1,2]

Explanation: The first 1's next greater number is 2;

The number 2 can't find next greater number.

The second 1's next greater number needs to search circularly, which is also 2.

Example 2:

Input: nums = [1,2,3,4,3]

Output: [2,3,4,-1,4]

**(Method : 1)**

class Solution {

public:

vector<int> nextGreaterElements(vector<int>& nums) {

stack<int>st;

vector<int>result(nums.size());

st.push(nums[nums.size()-1]);

for(int i=nums.size()-2;i>=0;i--)

{

while(!st.empty() && st.top()<=nums[i])

{

st.pop();

}

st.push(nums[i]);

}

for(int i=nums.size()-1;i>=0;i--)

{

while(!st.empty() && st.top()<=nums[i])

{

st.pop();

}

int res=(st.empty())?-1:st.top();

st.push(nums[i]);

result[i]=res;

}

return result;

}

};

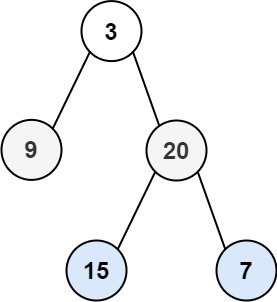
**(Method : 2) (more efficient)**

* **DAY 46:B\_Tree**

**1.Binary Tree Level Order Traversal :**

Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

**Example 1:**



**Input:** root = [3,9,20,null,null,15,7]

**Output:** [[3],[9,20],[15,7]]

**Example 2:**

**Input:** root = [1]

**Output:** [[1]]

**Example 3:**

**Input:** root = []

**Output:** []

class Solution {

public:

    vector<vector<int>> levelOrder(TreeNode\* root) {

        vector<vector<int>>v;

        if(root==NULL)

        {

            return v;

        }

        queue<TreeNode\*>q;

        q.push(root);

        while(!q.empty())

        {

            vector<int>v1;

            int count=q.size();

            while(count>0)

            {

                root=q.front();

                v1.push\_back(root->val);

                q.pop();

                if(root->left)

                {

                    q.push(root->left);

                }

                if(root->right)

                {

                    q.push(root->right);

                }

                count--;

            }

            if(!v1.empty())

            {

                v.push\_back(v1);

                v1.clear();

            }

        }

        return v;

    }

};

**2.Binary Tree ZigZag Level Order Traversal :**

Given the root of a binary tree, return *the zigzag level order traversal of its nodes' values*. (i.e., from left to right, then right to left for the next level and alternate between).

**Example 1:**

A diagram of a triangle

Description automatically generated

**Input:** root = [3,9,20,null,null,15,7]

**Output:** [[3],[20,9],[15,7]]

**Example 2:**

**Input:** root = [1]

**Output:** [[1]]

**Example 3:**

**Input:** root = []

**Output:** []

class Solution {

public:

    vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

        if(root==NULL)

        {

            return {};

        }

        vector<vector<int>>v;

        queue<TreeNode\*>q;

        int ZigZag=1;

        q.push(root);

        while(!q.empty())

        {

            int size=q.size();

            vector<int>v1;

            for(int i=0;i<size;i++)

            {

                root=q.front();

                v1.push\_back(root->val);

                q.pop();

                if(root->left)

                {

                    q.push(root->left);

                }

                if(root->right)

                {

                    q.push(root->right);

                }

            }

            if(ZigZag%2==0)

            {

                reverse(v1.begin(),v1.end());

            }

            v.push\_back(v1);

            v1={};

            ZigZag++;

        }

        return v;

    }

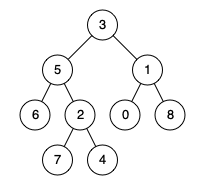
};

**3.Lowest Common Ancestor of a Binary Tree :**

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the [definition of LCA on Wikipedia](https://en.wikipedia.org/wiki/Lowest_common_ancestor): “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

**Example 1:**

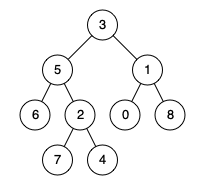


**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

**Output:** 3

**Explanation:** The LCA of nodes 5 and 1 is 3.

**Example 2:**



**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

**Output:** 5

**Explanation:** The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

**Example 3:**

**Input:** root = [1,2], p = 1, q = 2

**Output:** 1

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if(root==NULL)

        {

            return NULL;

        }

        if(root==p || root==q)

        {

            return root;

        }

        TreeNode\* leftN=lowestCommonAncestor(root->left,p,q);

        TreeNode\* rightN=lowestCommonAncestor(root->right,p,q);

        if(leftN!=NULL && rightN!=NULL)

        {

            return root;

        }

        if(leftN!=NULL)

        {

            return leftN;

        }

        return rightN;

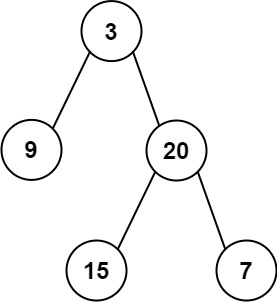
    }

};

**4.Construct Binary Tree from Inorder & Preorder Travesal :**

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return *the binary tree*.

**Example 1:**



**Input:** preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

**Output:** [3,9,20,null,null,15,7]

**Example 2:**

**Input:** preorder = [-1], inorder = [-1]

**Output:** [-1]