DSA MOST LIKELY QUESTIONS

1-Set Matrix zeros:

***Problem Statement:****Given a matrix if an element in the matrix is 0 then you will have to set its entire column and row to 0 and then return the matrix.*

void setZeros(vector<vector<int>> &m)

{

    map<int, int> row, col;

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

    {

        for (int j = 0; j < m[0].size(); j++)

        {

            if (m[i][j] == 0)

            {

                row[i]++;

                col[j]++;

            }

        }

    }

    for (auto x : row)

    {

        for (int i = 0; i < m[0].size(); i++)

        {

            m[x.first][i] = 0;

        }

    }

    for (auto x : col)

    {

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

        {

            m[i][x.first] = 0;

        }

    }

}

2-Pascals Triangle:

***Problem Statement:****Given an integer****N****, return the first****N****rows of Pascal’s triangle.*

vector<vector<int>> generate(int numRows)

{

    vector<vector<int>> ans(numRows);

    for (int i = 0; i < numRows; i++)

    {

        ans[i].resize(i + 1);

        ans[i][0] = 1;

        ans[i][i] = 1;

        for (int j = 1; j < i; j++)

        {

            ans[i][j] = ans[i - 1][j - 1] + ans[i - 1][j];

        }

    }

    return ans;

}

3-Next Permutation

***Problem Statement:****Given an array Arr[] of integers, rearrange the numbers of the given array into the lexicographically next greater permutation of numbers.*

*If such an arrangement is not possible, it must rearrange it as the lowest possible order (i.e., sorted in ascending order).*

void nextPermutation(vector<int> &nums)

{

    int i = nums.size() - 2;

    while (i >= 0 && nums[i + 1] <= nums[i])

    {

        i--;

    }

    if (i >= 0)

    {

        int j = nums.size() - 1;

        while (nums[j] <= nums[i])

        {

            j--;

        }

        swap(nums[i], nums[j]);

    }

    reverse(nums.begin() + i + 1, nums.end());

}

4-Kadane’s Algorithm

***Problem Statement****: Given an integer array arr, find the contiguous sub-array (containing at least one number) which  
has the largest sum and return its sum and print the sub-array.*

int Kandane(vector<int> &nums)

{

    int ans = 0;

    int cur = 0;

    int mx = INT\_MIN;

    if (nums.size() == 1)

        return nums[0];

    for (auto x : nums)

    {

        cur += x;

        mx = max(mx, cur);

        if (cur > 0)

            ans = max(ans, cur);

        else

            cur = 0;

    }

    if (ans == 0)

        return mx;

    else

        return ans;

}

5-Sort an Array of 0’s 1’s & 2’s

***Problem Statement:****Given an array consisting of only 0s, 1s and 2s. Write a program to in-place sort the array without using inbuilt sort functions. ( Expected: Single pass-O(N) and constant space)*

void sortColors(vector<int> &nums)

{

    int i = 0;

    for (int v = 0; v <= 1; v++)

    {

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

        {

            if (nums[j] == v)

            {

                swap(nums[j], nums[i]);

                i++;

            }

        }

    }

}

*6-Stock buy and sell problem*

**Problem Statement:** You are given an array of prices where prices[i] is the price of a given stock on an ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

int maxProfit(vector<int> &p)

{

    int curr = INT\_MAX;

    int ans = 0;

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

    {

        curr = min(curr, p[i]);

        int pro = p[i] - curr;

        ans = max(ans, pro);

    }

    return ans;

}

*7-Merge overlapping intervals*

***Problem Statement:****Given an array of intervals, merge all the overlapping intervals and return an array of non-overlapping intervals.*

vector<vector<int>> merge(vector<vector<int>> &inter)

{

    sort(inter.begin(), inter.end());

    vector<vector<int>> ans;

    ans.push\_back({inter[0][0], inter[0][1]});

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

    {

        vector<int> bc = ans.back();

        if (inter[i][0] >= bc[0] && inter[i][0] <= bc[1])

        {

            ans.pop\_back();

            vector<int> temp = {min(bc[0], inter[i][0]), max(bc[1], inter[i][1])};

            ans.push\_back(temp);

        }

        else

        {

            ans.push\_back({inter[i][0], inter[i][1]});

        }

    }

    return ans;

}

*8-merge two sorted array without extra space*

***Problem statement:****Given two sorted arrays****arr1[]****and****arr2[]****of**sizes****n****and****m****in non-decreasing order. Merge them in sorted order. Modify arr1 so that it contains the first N elements and modify arr2 so that it contains the last M elements.*

void merge(vector<int> &nums1, int m, vector<int> &nums2, int n)

{

    vector<int> ans;

    int i = 0, j = 0;

    while (i < m and j < n)

    {

        if (nums1[i] < nums2[j])

        {

            ans.push\_back(nums1[i]);

            i++;

        }

        else

        {

            ans.push\_back(nums2[j]);

            j++;

        }

    }

    while (i < m)

    { ans.push\_back(nums1[i]);

        i++;

}

    while (j < n)

{

        ans.push\_back(nums2[j]);

        j++;

    }

    nums1 = ans;

}

*9-Find Duplicate*

***Problem Statement:****Given an array of N + 1 size, where each element is between 1 and N. Assuming there is only one duplicate number, your task is to find the duplicate number.*

int findDuplicate(vector<int> &nums)

{

    int fast = nums[0];

    int slow = nums[0];

    do

    {

        fast = nums[nums[fast]];

        slow = nums[slow];

    } while (slow != fast);

    fast = nums[0];

    while (slow != fast)

    {

        fast = nums[fast];

        slow = nums[slow];

    }

    return slow;

}

*10-Rotate Image by 90deg*

***Problem Statement:****Given a matrix, your task is to rotate the matrix 90 degrees clockwise*.

void rotate(vector<vector<int>> &mat)

{

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

    {

        for (int j = i + 1; j < mat[0].size(); j++)

        {

            swap(mat[i][j], mat[j][i]);

        }

    }

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

    {

        reverse(mat[i].begin(), mat[i].end());

    }

}

*11-Repeating and Missing Element*

***Problem Statement:****You are given a read-only array of N integers with values also in the range [1, N] both inclusive. Each integer appears exactly once except A which appears twice and B which is missing. The task is to find the repeating and missing numbers A and B where A repeats twice and B is missing*.

vector<int> Solution::repeatedNumber(const vector<int> &A)

{

    long long int len = A.size();

    long long int S = (len \* (len + 1)) / 2;

    long long int P = (len \* (len + 1) \* (2 \* len + 1)) / 6;

    long long int missingNumber = 0, repeating = 0;

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

    {

        S -= (long long int)A[i];

        P -= (long long int)A[i] \* (long long int)A[i];

    }

    missingNumber = (S + P / S) / 2;

    repeating = missingNumber - S;

    vector<int> ans;

    ans.push\_back(repeating);

    ans.push\_back(missingNumber);

    return ans;

}

*12-Count Inversions in an Array*

***Problem Statement:****Given an array of N integers, count the inversion of the array (using*[*merge-sort*](https://takeuforward.org/data-structure/merge-sort-algorithm/)*).What is an inversion of an array? Definition: for all i & j < size of array, if i < j then you have to find pair (A[i],A[j]) such that A[j] < A[i].*

long long merge(vector<long long> &arr, vector<long long> &temp, int l, int mid, int r)

{

    int inver = 0;

    int i = l;

    int j = mid;

    int k = l;

    while (i <= mid - 1 && j <= r)

    {

        if (arr[i] < arr[j])

        {

            temp[k++] = arr[i++];

        }

        else

        {

            temp[k++] = arr[j++];

            inver += mid - i;

        }

    }

    while (i <= mid - 1)

        temp[k++] = arr[i++];

while (j <= r)

        temp[k++] = arr[j++];

    for (int i = l; i <= r; i++)

        arr[i] = temp[i];

    return inver;

}

long long mergeSort(vector<long long> &arr, vector<long long> &temp, int l, int r)

{

    int inver = 0;

    if (l < r)

    {

        int mid = l + (r - l) / 2;

        inver += mergeSort(arr, temp, l, mid);

        inver += mergeSort(arr, temp, mid + 1, r);

        inver += merge(arr, temp, l, mid + 1, r);

    }

    return inver;

}

long long getInversions(vector<long long> &arr, int n)

{

    vector<long long> temp(n);

    return mergeSort(arr, temp, 0, n - 1);

}

*13-Search in a 2-D sorted Array*

***Problem Statement:****Given an m\*n 2D matrix and an integer, write a program to find if the given integer exists in the matrix*

*Given matrix has the following properties:*

*1-Integers in each row are sorted from left to right and the first integer of each row is greater than the last integer of the previous row*

bool searchMatrix(vector<vector<int>> &mat, int t)

{

    int m = mat.size();

    int n = mat[0].size();

    int r = 0, c = n - 1;

    bool found = false;

    while (r < m && c >= 0)

    {

        if (mat[r][c] > t)

            c--;

        else if (mat[r][c] < t)

            r++;

        else

        {

            found = true;

            break;

        }

    }

    return found;

}

*14-Power(x,n)*

***Problem Statement:****Given a double x and integer n, calculate x raised to power n. Basically Implement pow(x, n).*

double bin(double a, long b)

{

    double ans = 1;

    if (b < 0)

    {

        b = -b;

        a = 1 / a;

    }

    while (b)

    {

        if (b & 1)

        {

            ans \*= a;

        }

        a = (a \* a);

        b >>= 1;

    }

    return ans;

}

double myPow(double x, int n)

{

    double ans = bin(x, n);

    return ans;

}

*15-Find the majority element that occurs more than n/2 times*

***Problem Statement:****Given an array of****N integers****, write a program to return an element that occurs more than****N/2****times in the given array. You may consider that such an element always exists in the array.*

int majorityElement(vector<int> &nums)

{

    // Moore’s Voting Algorithm

    int cnt = 0;

    int ele = 0;

    for (auto x : nums)

    {

        if (cnt == 0)

        {

            ele = x;

        }

        if (x == ele)

            cnt++;

        else

            cnt--;

    }

    return ele;

}

*16- Find the majority element that occurs more than n/3 times*

***Problem Statement:****Given an array of N integers. Find the elements that appear more than****N/3****times in the array. If no such element exists, return an empty vector.*

vector<int> majorityElement(vector<int> &nums)

{

    int c1 = 0;

    int c2 = 0;

    int n1 = -1;

    int n2 = -1;

    int n = nums.size();

    for (auto x : nums)

    {

        if (x == n1)

            c1++;

        else if (x == n2)

            c2++;

        else if (c1 == 0)

        {

            c1 = 1;

            n1 = x;

        }

        else if (c2 == 0)

        {

            c2 = 1;

            n2 = x;

        }

        else

        {

            c1--;

            c2--;

        }

    }

    c1 = 0;

    c2 = 0;

    for (auto x : nums)

    {

        if (x == n1)

            c1++;

        else if (x == n2)

            c2++;

    }

    vector<int> ans;

    if (c1 > n / 3)

        ans.push\_back(n1);

    if (c2 > n / 3)

        ans.push\_back(n2);

    return ans;

}

*17-Count Unique Paths*

***Problem Statement:****Given a matrix****m X n****, count paths from left-top to the right bottom of a matrix with the constraints that from each cell you can either only move to the rightward direction or the downward direction. The test cases are generated so that the answer will be less than or equal to 2 \* 109.*

int Paths(int i, int j, int n, int m, vector<vector<int>> &dp)

{

    if (i > n - 1 || j > m - 1)

        return 0;

    if (dp[i][j] != -1)

        return dp[i][j];

    if (i == n - 1 && j == m - 1)

        return 1;

    int ans = Paths(i + 1, j, n, m, dp) + Paths(i, j + 1, n, m, dp);

    return dp[i][j] = ans;

}

int uniquePaths(int n, int m)

{

    vector<vector<int>> dp(n + 1, vector<int>(m, -1));

    return Paths(0, 0, n, m, dp);

}

*18-Count Reverse Pairs*

***Problem Statement:****Given an array of numbers, you need to return the count of reverse pairs.****Reverse Pairs****are those pairs where i<j and arr[i]>2\*arr[j].*

int Merge(int l, int mid, int r, vector<int> &v, vector<int> &temp)

{

    int total = 0;

    int j = mid + 1;

    for (int i = l; i <= mid; i++)

    {

        while (j <= r && v[i] > 2LL \* v[j])

        {

            j++;

        }

        total += (j - (mid + 1));

    }

    int i = l;

    j = mid + 1;

    int k = l;

    while (i <= mid && j <= r)

    {

        if (v[i] <= v[j])

        {

            temp[k++] = v[i++];

        }

        else

        {

            temp[k++] = v[j++];

        }

    }

    while (i <= mid)

        temp[k++] = v[i++];

    while (j <= r)

        temp[k++] = v[j++];

    for (int i = l; i <= r; i++)

        v[i] = temp[i];

    return total;

}

int mergeSort(int l, int r, vector<int> &v, vector<int> &temp)

{

    int pairs = 0;

    if (l < r)

    {

        int mid = l + (r - l) / 2;

        pairs += mergeSort(l, mid, v, temp);

        pairs += mergeSort(mid + 1, r, v, temp);

        pairs += Merge(l, mid, r, v, temp);

    }

    return pairs;

}

int ReversePairs(vector<int> &v, int n)

{

    vector<int> temp(n);

    return mergeSort(0, n - 1, v, temp);

}

*19-Two Sum*

*Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.*

vector<int> twoSum(vector<int> &nums, int target)

{

    vector<pair<int, int>> p;

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

    {

        p.push\_back({nums[i], i});

    }

    vector<int> ans;

    sort(p.begin(), p.end());

    int i = 0, j = nums.size() - 1;

    while (i < j)

    {

        int sum = p[i].first + p[j].first;

        if (sum == target)

        {

            ans.push\_back(p[i].second);

            ans.push\_back(p[j].second);

            break;

        }

        else if (sum > target)

        {

            j--;

        }

        else

        {

            i++;

        }

    }

    return ans;

}

*20-4-Sum*

***Problem Statement:****Given an array of N integers, your task is to find unique quads that add up to give a target value. In short, you need to return*an array of all the unique quadruplets*[arr[a], arr[b], arr[c], arr[d]] such that their sum is equal to a given*target*.*

vector<vector<int>> fourSum(vector<int> &nums, long long target)

{

    int n = nums.size();

    sort(nums.begin(), nums.end());

    set<vector<int>> sv;

    for (int i = 0; i < n; i++)

    {

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

        {

            for (int k = j + 1; k < n; k++)

            {

                long long x = (long long)target -

                              (long long)nums[i] -

                              (long long)nums[j] - (long long)nums[k];

                if (binary\_search(nums.begin() + k + 1, nums.end(), x))

                {

                    vector<int> v;

                    v.push\_back(nums[i]);

                    v.push\_back(nums[j]);

                    v.push\_back(nums[k]);

                    v.push\_back(x);

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

                    sv.insert(v);

                }

            }

        }

    }

    vector<vector<int>> res(sv.begin(), sv.end());

    return res;

}

*21-Longest Consecutive Sequence in an Array*

***Problem Statement:****You are given an array of ‘N’ integers. You need to find the length of the longest sequence which contains the consecutive elements.*

int longestConsecutive(vector<int> &nums)

{

    set<int> st;

    for (auto x : nums)

        st.insert(x);

    int ans = 0;

    for (auto x : nums)

    {

        if (!st.count(x - 1))

        {

            int temp = 1;

            int curr = x;

            while (st.count(curr + 1))

            {

                temp++;

                curr++;

            }

            ans = max(ans, temp);

        }

    }

    return ans;

}

*22-Longest Subarray with Zero Sum*

***Problem Statement:****Given an array containing both positive and negative integers, we have to find the length of the longest subarray with the sum of all elements equal to zero.*

int maxLen(vector<int> &A, int n)

{

    unordered\_map<int, int> mp;

    int sum = 0;

    int ans = 0;

    for (int i = 0; i < n; i++)

    {

        sum += A[i];

        if (sum == 0)

            ans = max(ans, i + 1);

        else

        {

            if (mp.count(sum))

                ans = max(ans, i - mp[sum]);

            else

                mp[sum] = i;

        }

    }

    return ans;

}

*23-Count the no. of subarrays with given XOR ‘K’*

***Problem Statement:****Given an array of integers A and an integer B. Find the total number of subarrays having bitwise XOR of all elements equal to B.*

int solve(vector<int> &A, int B)

{

    unordered\_map<int, int> mp;

    mp[0] = 1;

    int ans = 0;

    int curr = 0;

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

    {

        curr ^= A[i];

        if (mp[curr ^ B])

            ans += mp[curr ^ B];

        mp[curr]++;

    }

    return ans;

}

*24-Longest substring without any repeating character*

***Problem Statement:****Given a String, find the length of longest substring without any repeating character*.

int lengthOfLongestSubstring(string s)

{

    unordered\_map<char, int> mp;

    int i = 0, j = 0;

    int ans = -1;

    while (j < s.length())

    {

        mp[s[j]]++;

        if (mp[s[j]] > 1)

        {

            ans = max(ans, j - i);

            while (i < j && mp[s[j]] != 1)

            {

                mp[s[i]]--;

                i++;

            }

        }

        j++;

    }

    ans = max(ans, j - i);

    return ans;

}

*25-Reverse a Linked List*

***Problem Statement:****Given the*head *of a singly linked list, write a program to reverse the linked list, and return*the head pointer to the reversed list*.*

ListNode \*reverseList(ListNode \*head)

{

    if (!head)

        return head;

    ListNode \*pre = NULL;

    while (head->next != NULL)

    {

        ListNode \*nxt = head->next;

        head->next = pre;

        pre = head;

        head = nxt;

    }

    head->next = pre;

    return head;

}

*26-Middle of a Linked List*

***Problem Statement:****Given the****head****of a singly linked list, return*the middle node of the linked list*. If there are two middle nodes, return the second middle node.*

ListNode \*middleNode(ListNode \*head)

{

    ListNode \*fast = head;

    ListNode \*slow = head;

    while (fast && fast->next)

    {

        fast = fast->next->next;

        slow = slow->next;

    }

    return slow;

}

*27-Merge Two Sorted Linked List*

***Problem Statement:****Given two singly linked lists that are sorted in increasing order of node values, merge two****sorted****linked lists and return them as a sorted list. The list should be made by splicing together the nodes of the first two lists.*

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

{

    if (!l1)

        return l2;

    if (!l2)

        return l1;

    if (l1->val > l2->val)

        swap(l1, l2);

    ListNode \*res = l1;

    while (l1 && l2)

    {

        ListNode \*temp = NULL;

        while (l1 && l1->val <= l2->val)

        {

            temp = l1;

            l1 = l1->next;

        }

        temp->next = l2;

        swap(l1, l2);

    }

    return res;

}

*28-Remove nth Node from the end of a LL*

***Problem Statement:****Given a*[*linked list*](https://takeuforward.org/linked-list/linked-list-introduction/)*, and a number N. Find the Nth node from the end of this linked list and delete it. Return the head of the new modified linked list.*

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

{

    ListNode \*start = new ListNode();

    start->next = head;

    ListNode \*fast = start;

    ListNode \*slow = start;

    for (int i = 1; i <= n; ++i)

        fast = fast->next;

    while (fast->next != NULL)

    {

        fast = fast->next;

        slow = slow->next;

    }

slow->next = slow->next->next;

    return start->next;

}

*29-Add two numbers represented as LL*

***Problem Statement****: Given the****heads****of two non-empty linked lists representing two non-negative integers. The digits are stored in****reverse order****, and each of their nodes contains a single digit. Add the two numbers and return the****sum****as a linked list.*

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

    {

        ListNode \*h1 = l1;

        ListNode \*h2 = l2;

        ListNode \*ans;

        int carry = 0;

        while (l1 && l2)

        {

            int sum = l1->val + l2->val + carry;

            carry = sum / 10;

            sum = sum % 10;

            l1->val = sum;

            l2->val = sum;

            l1 = l1->next;

            l2 = l2->next;

        }

        if (!l1 && l2)

        {

            while (l2)

            {

                int sum = l2->val + carry;

                carry = sum / 10;

                sum %= 10;

                l2->val = sum;

                l2 = l2->next;

            }

            if (carry)

            {

                ListNode \*node = new ListNode(carry);

                l2 = h2;

                while (l2->next)

                    l2 = l2->next;

                l2->next = node;

                l2->next->next = NULL;

            }

            ans = h2;

        }

        else if (l1 && !l2)

        {

            while (l1)

            {

                int sum = l1->val + carry;

                carry = sum / 10;

                sum %= 10;

                l1->val = sum;

                l1 = l1->next;

            }

            if (carry)

            {

                ListNode \*node = new ListNode(carry);

                l1 = h1;

                while (l1->next)

                    l1 = l1->next;

                l1->next = node;

                l1->next->next = NULL;

            }

            ans = h1;

        }

        else

        {

            if (carry)

            {

                ListNode \*node = new ListNode(carry);

                l1 = h1;

                while (l1->next)

                    l1 = l1->next;

                l1->next = node;

                l1->next->next = NULL;

            }

            ans = h1;

        }

        return ans;

    }

*30-Delete given node in a LL*

***Problem Statement:****Write a function to****delete a node****in a singly-linked list. You will****not****be given access to the head of the list instead, you will be given access to****the node to be deleted****directly. It is****guaranteed****that the node to be deleted is****not a tail node****in the list.*

void deleteNode(ListNode \*node)

{

    node->val = node->next->val;

    node->next = node->next->next;

    return;

}

*31-Find intersection of two LL*

***Problem Statement:****Given the heads of two singly*[*linked-lists*](https://takeuforward.org/linked-list/linked-list-introduction/)***headA****and****headB****, return****the node at which the two lists intersect****. If the two linked lists have no intersection at all, return****null****.*

ListNode \*getIntersectionNode(ListNode \*A, ListNode \*B)

{

    ListNode \*d1 = A;

    ListNode \*d2 = B;

    while (d1 != d2)

    {

        d1 = d1 == NULL ? B : d1 = d1->next;

        d2 = d2 == NULL ? A : d2 = d2->next;

    }

    return d1;

}

*32-Detect cycle in a LL*

***Problem Statement:****Given*head*, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer.*

*Return*true*if there is a cycle in the linked list. Otherwise, return*false*.*

bool hasCycle(ListNode \*head)

{

    ListNode \*fast = head;

    ListNode \*slow = head;

    if (!fast || !fast->next)

        return false;

    do

    {

        fast = fast->next->next;

        slow = slow->next;

    } while (fast != slow && fast && fast->next);

    if (fast == slow)

        return true;

    else

        return false;

}

*33-Reverse LL in groups of size k*

***Problem Statement:****Given the head of a linked list, reverse the nodes of the list k at a time, and return*the modified list*. k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.*

int Length(ListNode \*head)

{

    int len = 0;

    while (head)

    {

        len++;

        head = head->next;

    }

    return len;

}

ListNode \*reverseKGroup(ListNode \*head, int k)

{

    ListNode \*dummy = new ListNode(0);

    dummy->next = head;

    ListNode \*pre = dummy;

    ListNode \*cur;

    ListNode \*nex;

    int len = Length(head);

    while (len >= k)

    {

        cur = pre->next;

        nex = cur->next;

        for (int i = 1; i < k; i++)

        {

            cur->next = nex->next;

            nex->next = pre->next;

            pre->next = nex;

            nex = cur->next;

        }

        pre = cur;

        len -= k;

    }

    return dummy->next;

}

*34-Check for palindrome*

***Problem Statement:****Given the head of a singly linked list, return true if it is a palindrome.*

ListNode \*reverse(ListNode \*ptr)

{

    ListNode \*pre = NULL;

    ListNode \*nex = NULL;

    while (ptr != NULL)

    {

        nex = ptr->next;

        ptr->next = pre;

        pre = ptr;

        ptr = nex;

    }

    return pre;

}

bool isPalindrome(ListNode \*head)

{

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

        return true;

    ListNode \*slow = head;

    ListNode \*fast = head;

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

    {

        slow = slow->next;

        fast = fast->next->next;

    }

    slow->next = reverse(slow->next);

    slow = slow->next;

    ListNode \*dummy = head;

    while (slow != NULL)

    {

        if (dummy->val != slow->val)

            return false;

        dummy = dummy->next;

        slow = slow->next;

    }

    return true;

}

*35-Starting point of a loop of a LL*

***Problem Statement:****Given the head of a*[*linked list*](https://takeuforward.org/linked-list/linked-list-introduction/)*, return*the node where the cycle begins. If there is no cycle, return *null.*

*There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that the tail’s next pointer is connected to (0-indexed). It is -1 if there is no cycle. Note that pos is not passed as a parameter.*

ListNode \*detectCycle(ListNode \*head)

{

    ListNode \*fast = head;

    ListNode \*slow = head;

    if (!fast || !fast->next)

        return NULL;

    do

    {

        fast = fast->next->next;

        slow = slow->next;

    } while (fast != slow && fast && fast->next);

    if (fast != slow)

        return NULL;

    fast = head;

    while (fast != slow)

    {

        slow = slow->next;

        fast = fast->next;

    }

    return slow;

}

*36-Flattening a LL*

**Problem Statement:**Given a [Linked List](https://takeuforward.org/linked-list/linked-list-introduction/) of size N, where every node represents a sub-linked-list and contains two pointers:

(i) a next pointer to the next node,

(ii) a bottom pointer to a linked list where this node is head.

Each of the sub-linked-list is in sorted order.

Flatten the Link List such that all the nodes appear in a single level while maintaining the sorted order.

Note: The flattened list will be printed using the bottom pointer instead of the next pointer.

**Examples:**

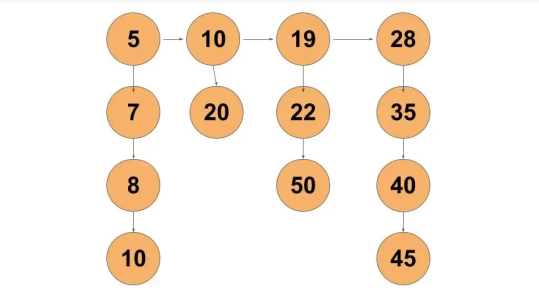
**Example 1:**

**Input:**

Number of head nodes = 4

Array holding length of each list with head and bottom = [4,2,3,4]

Elements of entire linked list = [5,7,8,30,10,20,19,22,50,28,35,40,45]

****

**Output:**

Flattened list = [5,7,8,10,19,20,22,28,30,35,40,45,50]

**Explanation:**

Flattened list is the linked list consisting entire elements of the given list in sorted order

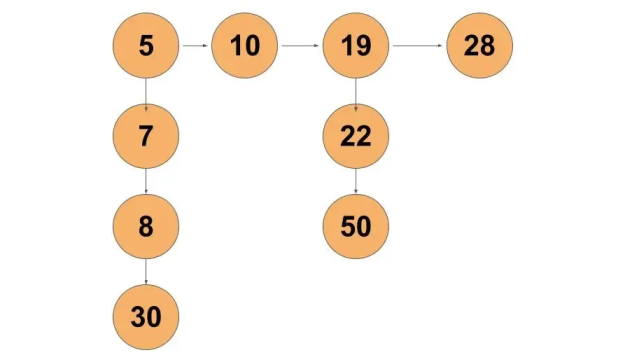
**Example 2:**

**Input:**

Number of head nodes = 4

Array holding length of each list with head and bottom = [4,1,3,1]

Elements of entire linked list = [5,7,8,30,10,19,22,50,28]

****

**Output:**

Flattened list = [5,7,8,10,19,22,28,30,50]

**Explanation:**

Flattened list is the linked list consisting entire elements of the given list in sorted order

Node \*merge(Node \*a, Node \*b)

{

    if (!a)

        return b;

    if (!b)

        return a;

    Node \*result;

    if (a->data <= b->data)

    {

        result = a;

        result->bottom = merge(a->bottom, b);

    }

    else

    {

        result = b;

        result->bottom = merge(a, b->bottom);

    }

    return result;

}

Node \*flatten(Node \*root)

{

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

        return root;

    return merge(root, flatten(root->next));

}

*37-Rotate a LL*

***Problem Statement:****Given the head of a*[*linked list*](https://takeuforward.org/linked-list/linked-list-introduction/)*, rotate the list to the right by k places.*

int length(ListNode \*head)

{

    int len = 0;

    while (head)

    {

        len++;

        head = head->next;

    }

    return len;

}

ListNode \*rotateRight(ListNode \*head, int k)

{

    if (!head || !head->next)

        return head;

    int len = length(head);

    k = k % len;

    k = len - k;

    k--;

    ListNode \*temp = head;

    while (k--)

    {

        temp = temp->next;

    }

    ListNode \*last = head;

    while (last->next)

        last = last->next;

    last->next = head;

    head = temp->next;

    temp->next = NULL;

    return head;

}

*38-Copy LL with Random pointers*

A linked list of length n is given such that each node contains an additional random pointer, which could point to any node in the list, or null.

Construct a [**deep copy**](https://en.wikipedia.org/wiki/Object_copying#Deep_copy) of the list. The deep copy should consist of exactly n **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list**.

For example, if there are two nodes X and Y in the original list, where X.random --> Y, then for the corresponding two nodes x and y in the copied list, x.random --> y.

Return *the head of the copied linked list*.

The linked list is represented in the input/output as a list of n nodes. Each node is represented as a pair of [val, random\_index] where:

* val: an integer representing Node.val
* random\_index: the index of the node (range from 0 to n-1) that the random pointer points to, or null if it does not point to any node.

Your code will **only** be given the head of the original linked list.

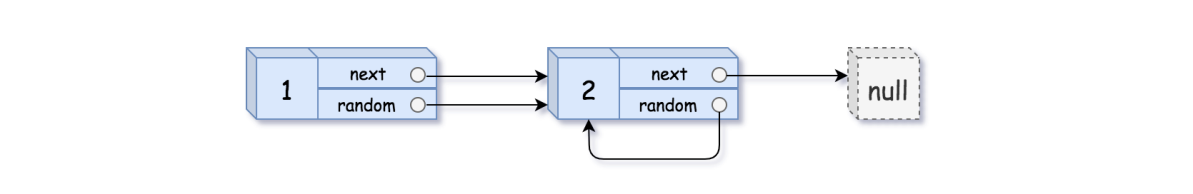
**Example 1:**



**Input:** head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

**Output:** [[7,null],[13,0],[11,4],[10,2],[1,0]]

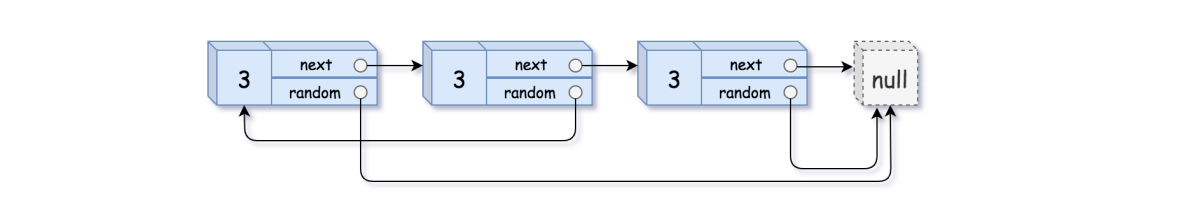
**Example 2:**



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

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

**Example 3:**

****

**Input:** head = [[3,null],[3,0],[3,null]]

**Output:** [[3,null],[3,0],[3,null]]

Node \*copyRandomList(Node \*head)

{

    Node \*iter = head;

    Node \*front = head;

    while (iter)

    {

        front = iter->next;

        Node \*copy = new Node(iter->val);

        iter->next = copy;

        copy->next = front;

        iter = front;

    }

    iter = head;

    while (iter)

    {

        if (iter->random)

            iter->next->random = iter->random->next;

        iter = iter->next->next;

    }

    iter = head;

    Node \*phead = new Node(0);

    Node \*copy = phead;

    while (iter)

    {

        front = iter->next->next;

        copy->next = iter->next;

        iter->next = front;

        copy = copy->next;

        iter = iter->next;

    }

    return phead->next;

}

*39-3 Sum*

***Problem Statement:****Given an array of N integers, your task is to find unique triplets that add up to give a sum of zero. In short, you need to return*an array of all the unique*triplets [arr[a], arr[b], arr[c]] such that i!=j, j!=k, k!=i, and their sum is equal to zero.*

vector<vector<int>> threeSum(vector<int> &nums)

{

    int n = nums.size();

    sort(nums.begin(), nums.end());

    vector<vector<int>> ans;

    for (int i = 0; i < n; i++)

    {

        if (nums[i] > 0)

            break;

        if (i > 0 && nums[i - 1] == nums[i])

            continue;

        int sum = -nums[i];

        int l = i + 1, r = n - 1;

        while (l < r)

        {

            int temp = nums[l] + nums[r];

            if (temp == sum)

            {

                vector<int> v;

                ans.push\_back({nums[i], nums[l], nums[r]});

                while (l < r && nums[l] == nums[l + 1])

                    l++;

                while (l < r && nums[r] == nums[r - 2])

                    r--;

                l++;

                r--;

            }

            else if (temp > sum)

                r--;

            else

                l++;

        }

    }

    return ans;

}

*40-Remove Duplicates in-place from a sorted array*

***Problem Statement:****Given an integer array sorted in non-decreasing order, remove the duplicates in place such that each unique element appears only once. The relative order of the elements should be kept the same.If there are k elements after removing the duplicates, then the first k elements of the array should hold the final result. It does not matter what you leave beyond the first k elements.*

***Note:****Return k after placing the final result in the first k slots of the array.*

int removeDuplicates(vector<int> &nums)

{

    int c = 1;

    int curr = nums[0];

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

    {

        if (nums[i] != curr)

        {

            curr = nums[i];

            nums[c] = curr;

            c++;

        }

    }

    return c;

}

*41-Count Maximum consecutive ones in an array*

***Problem Statement:****Given an array that contains****only 1 and 0****return the count of****maximum consecutive****ones in the array.*

int findMaxConsecutiveOnes(vector<int> &nums)

{

    int ans = 0;

    int cur = 0;

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

    {

        if (nums[i] == 1)

        {

            cur++;

        }

        else

        {

            ans = max(ans, cur);

            cur = 0;

        }

    }

    ans = max(ans, cur);

    return ans;

}

*42-*