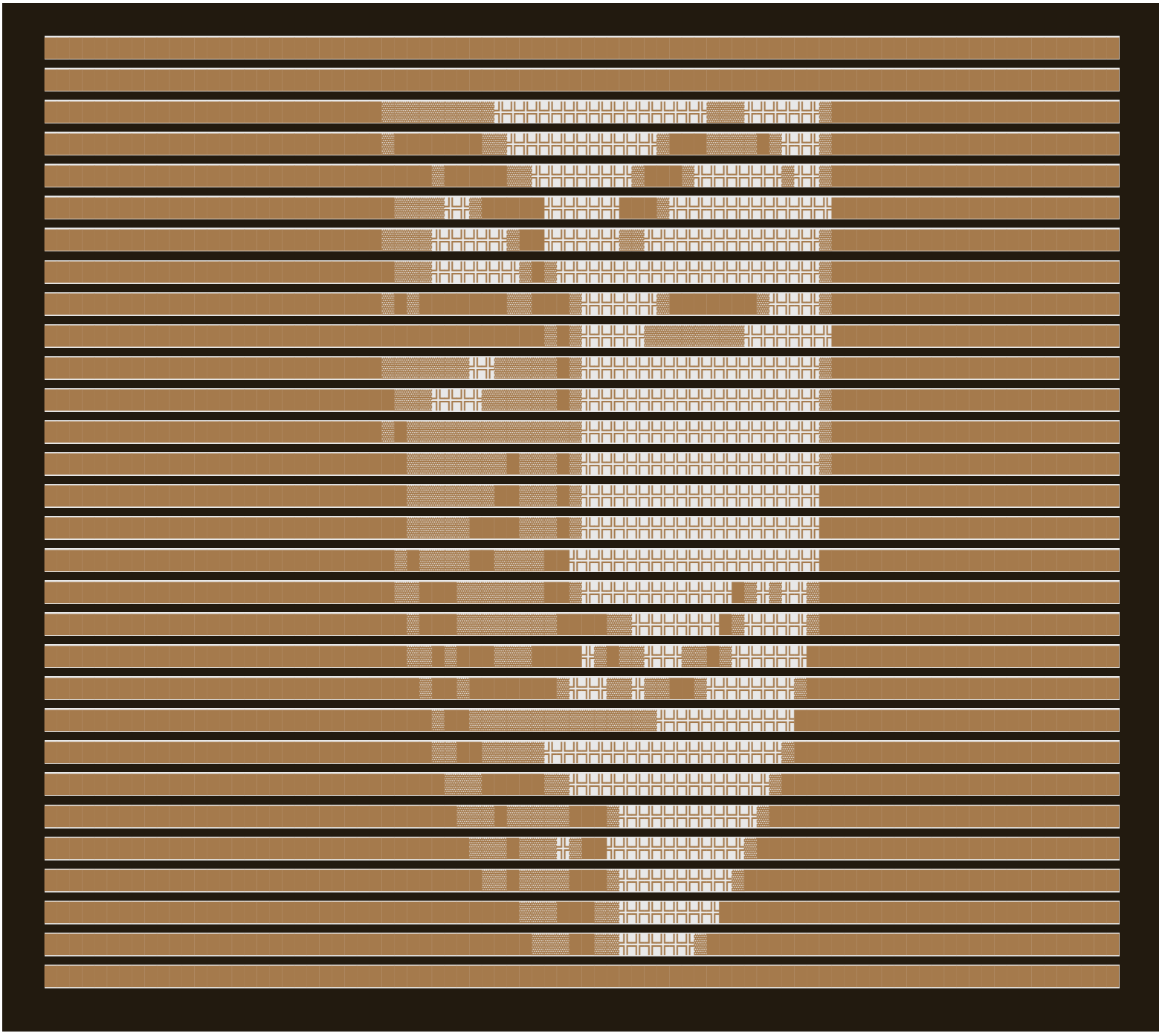
Strivers A2Z Sheet

-By Ashish



Contents

Step 1: Learn the basics

Step 1.1: Things to Know in C++

If Else statements

```
string compareNM(int n, int m){
    if (n < m) return "lesser";
    else if (n == m) return "equal";
    return "greater";
}
```

Step 1.4: Know Basic Maths

Count Digits

```
int evenlyDivides(int N){
    int M = N, ans = 0;
    while (M){
        if (M % 10 != 0 && N % (M % 10) == 0) ans++;
        M /= 10;
    }
    return ans;
}
```

Reverse a Number

```
int reverse(int x) {
    int ans=0;
    while(x){
       if (ans>INT_MAX/10 || (ans==INT_MAX/10 && x%10>INT_MAX%10)) return 0;
       if (ans<INT_MIN/10 || (ans==INT_MIN/10 && x%10<INT_MIN%10)) return 0;
        ans=ans*10+x%10;
        x/=10;
    }
    return ans;
}
```

Check Palindrome

```
bool isPalindrome(int x) {
    if(x<0) return false;
    string s=to_string(x);
    string t=s;
    reverse(s.begin(),s.end());
    return s==t;
}
```

GCD Or HCF

```
long long gcd(long long a, long long b){
    if (b == 0) return a;
    return gcd(b, a % b);
}

vector<long long> lcmAndGcd(long long A, long long B){
    long long a = gcd(A, B);
    vector<long long> ans = {A / a * B, a};
    return ans;
}
```

Armstrong Numbers

```
string armstrongNumber(int n){
    int m = n;
    int curr = 0;
    while (m){
        curr += pow(m % 10, 3);
        m /= 10;
    }
    if (curr == n) return "Yes";
    else return "No";
}
```

Sum of all divisors from 1 to n

```
long long sumOfDivisors(int N){
    long long ans = 0;
    for (int i = 1; i <= N; i++) ans += (i * (N / i));
    return ans;
}
```

Minimum number of jumps

```
int minJumps(int arr[], int n){
    if (n == 1) return 0;
    int step = 1, idx = arr[0], nxt_idx = arr[0];

    for (int i = 1; i < n; i++){
        if (i > idx){
            idx = nxt_idx;
            step++;
        }
        if (i > idx) return -1;
        nxt_idx = max(nxt_idx, i + arr[i]);
    }
    return step;
}
```

Step 1.5: Learn Basic Recursion

Print name N times using recursion

```
void printGfg(int N){
    if (N == 0) return;
    cout << "GFG ";
    printGfg(N - 1);
}
```

Print 1 To N Without Loop

```
void printNos(int N){
    if (N == 0) return;
    printNos(N - 1);
    cout << N << " ";
}
```

Print N to 1 without loop

```
void printNos(int N){
    if (N == 0) return;
    cout << N << " ";
    printNos(N - 1);
}
```

Sum of first n terms

```
long long sumOfSeries(long long n){
    if (n == 1) return 1;
    return n * n * n + sumOfSeries(n - 1);
}
```

```
long long sumOfSeries(long long n){
    long long ans = n * (n + 1) / 2;
    return ans * ans;
}
```

Factorial of N numbers

```
void solve(long long i, long long curr, long long N, vector<long long> &ans){
    if (curr > N) return;
    ans.push_back(curr);
    solve(i + 1, curr * i, N, ans);
}

vector<long long> factorialNumbers(long long N){
    vector<long long> ans;
    solve(2, 1, N, ans);
    return ans;
}
```

```
vector<long long> factorialNumbers(long long N){
    vector<long long> ans;
    long long val = 1, i = 1;
    while (val <= N){
        ans.push_back(val);
        i++;
        val *= i;
    }
    return ans;
}
```

Check if a string is palindrome or not

```
    bool isPalindrome(string s) {
        int i=0,j=s.size()-1;

        while(i<j){
            if(s[i]>='A' && s[i]<='Z') s[i]+='a'-'A';
            else if((s[i]<'a' || s[i]>'z')&&(s[i]<'0'|| s[i]>'9'))
              {i++;continue;}

            if(s[j]>='A' && s[j]<='Z') s[j]+='a'-'A';
            else if((s[j]<'a' || s[j]>'z') && (s[j]<'0'|| s[j]>'9'))
              {j--;continue;}

            if(s[i]!=s[j]) return false;
            i++;
            j--;
        }
        return true;
    }

bool isPalindrome(string s) {
    int i=0,j=s.size()-1;
    while(i<j){
        if(!isalnum(s[i])) i++;
        else if(!isalnum(s[j])) j--;
```

```
        else if(tolower(s[i])!=tolower(s[j])) return false;
        else {
            i++;
            j--;
        }
    }
    return true;
}
```

Fibonacci Number

```
void solve(long long a, long long b, int n, vector<long long> &ans){
    ans.push_back(a + b);
    if (ans.size() == n) return;
    solve(b, a + b, n, ans);
}

vector<long long> printFibb(int n){
    vector<long long> ans = {1};
    if (n == 1) return ans;
    solve(0, 1, n, ans);
}
```

Counting frequencies of array elements**** O(1) space ***

```
void frequencyCount(vector<int> &arr, int N, int P){
    int curr = 0, next;

    while (curr < N){
        next = arr[curr] - 1;

        if (next >= N) arr[curr++] = 0;
        else if (arr[curr] < 0) curr++;
        else if (arr[next] > 0){
            arr[curr] = arr[next];
            arr[next] = -1;
            if (curr == next) curr++;
        }
        else{
```

```
            arr[next]--;
            arr[curr++] = 0;
        }
    }

    for (int i = 0; i < N; i++) arr[i] = abs(arr[i]);
}
```

Top K Frequent Elements in Array

```
map<int,int> mp;
static bool comp(pair<int,int> a,pair<int,int> b){
    if(a.second==b.second) return a.first>b.first;
    return a.second>b.second;
}

vector<int> topK(vector<int>& nums, int k) {
    int n=nums.size();
    for(int i=0;i<n;i++) mp[nums[i]]++;

    vector<pair<int,int>> temp;
    for(auto it:mp) temp.push_back(it);
    sort(temp.begin(),temp.end(),comp);

    vector<int> ans;
    for(int i=0;i<k;i++) ans.push_back(temp[i].first);
    return ans;
}
```

Step 2: Learn Important Sorting Techniques

Step 2.1: Sorting-I

Selection Sort

```
int select(int arr[], int i, int n){
    int idx = i;
    for (int k = i + 1; k < n; k++){
        if (arr[k] < arr[idx]) idx = k;
    }
    return idx;
}

void selectionSort(int arr[], int n){
    for (int i = 0; i < n; i++){
        int j = select(arr, i, n);
        swap(arr[i], arr[j]);
    }
}
```

Bubble Sort

```
void bubbleSort(int arr[], int n){
    for(int i=n-1;i>=0;i--){
        for(int j=0;j<i;j++){
            if(arr[j]>arr[j+1]) swap(arr[j],arr[j+1]);
        }
    }
}
```

Insertion Sort

```
void insert(int arr[], int i){
    int val = arr[i];
    i--;
    while (i >= 0 && arr[i] > val){
        arr[i + 1] = arr[i];
        i--;
```

```
    }
    arr[i + 1] = val;
}

void insertionSort(int arr[], int n){
    for (int i = 1; i < n; i++) insert(arr, i);
}
```

Step 2.2: Sorting-II

[Merge Sort]

```
void merge(int arr[], int l, int m, int r){
    int temp[r - l + 1];
    int i = 0, j = l, k = m + 1;

    while (j <= m && k <= r){
        if (arr[j] < arr[k]) temp[i++] = arr[j++];
        else temp[i++] = arr[k++];
    }

    while (j <= m) temp[i++] = arr[j++];
    while (k <= r) temp[i++] = arr[k++];

    for (int i = 0; i <= r - l; i++) arr[i + l] = temp[i];
}

void mergeSort(int arr[], int l, int r)
{
    if (l >= r) return;
    int mid = l + (r - l) / 2;
    mergeSort(arr, l, mid);
    mergeSort(arr, mid + 1, r);
    merge(arr, l, mid, r);
}
```

Bubble Sort

```
void bubbleSort(int arr[], int n){
    if(n==1) return;
    for(int j=0;j<n-1;j++){
        if(arr[j]>arr[j+1]) swap(arr[j],arr[j+1]);
    }
    bubbleSort(arr,n-1);
}
```

Insertion Sort

```
void insert(int arr[], int i,int n){
    if(i==n) return;

    int val=arr[i],j=i-1;
    while(j>=0 && arr[j]>val){
        arr[j+1]=arr[j];
        j--;
    }
    arr[j+1]=val;

    insert(arr,i+1,n);
}

void insertionSort(int arr[], int n){
    insert(arr,1,n);
}
```

Quick Sort

```
int partition (int arr[], int low, int high){
    int pivot=arr[high];
    int curr=low,next=high-1;

    while(curr<=next){
        if(arr[curr]>pivot) swap(arr[curr],arr[next--]);
        else curr++;
    }
    swap(arr[curr],arr[high]);
    return curr;
}

void quickSort(int arr[], int low, int high){
    if(low>=high) return;
    int i=partition(arr,low,high);
    quickSort(arr,low,i-1);
    quickSort(arr,i+1,high);
}
```

Step 3: Solve Problems on Arrays [Easy -> Medium -> Hard]

Step 3.1: Easy

Largest Element in Array [TC:O(n) & SC:O(1)]

```
int largest(vector<int> &arr, int n){
    int mx=0;
    for(int i=0;i<n;i++) mx=max(mx,arr[i]);
    return mx;
}
```

Second Largest Element in an array Without Sorting [TC:O(n) & SC:O(1)]

```
int print2largest(int arr[], int n) {
    int mx1=-1,mx2=-1;
    for(int i=0;i<n;i++){
        if(arr[i]>mx1){
            mx2=mx1;
            mx1=arr[i];
        }
        else if (arr[i]<mx1 && arr[i]>mx2){
            mx2=arr[i];
        }
    }
    return mx2;
}
```

Check if Array Is Sorted and Rotated [TC:O(n) & SC:O(1)]

```
bool check(vector<int>& nums) {
    int n=nums.size(),ct=0;

    for(int i=1;i<n;i++){
        if(nums[i-1]>nums[i]) {
            if(nums[0]<nums[n-1] || ct) return false;
            ct++;
        }
    }
    return true;
}
```

Remove Duplicates from Sorted Array [TC:O(n) & SC:O(1)]

```
int removeDuplicates(vector<int>& nums) {
    int n=nums.size();
    int idx=1;
    for(int i=1;i<n;i++){
        if(nums[i]!=nums[i-1]) nums[idx++]=nums[i];
    }
    return idx;
}
```

Rotate Array by K place [TC:O(n) & SC:O(1)]

```
void reverse(int i,int j,vector<int> & nums){
    while(i<j) swap(nums[i++],nums[j--]);
}

void rotate(vector<int>& nums, int k) {
    int n=nums.size();
    k%=n;
    reverse(0,n-k-1,nums);
    reverse(n-k,n-1,nums);
    reverse(0,n-1,nums);
}
```

Move Zeroes [TC:O(n) & SC:O(1)]

```
void moveZeroes(vector<int>& nums) {
    int n=nums.size();
    int idx=0;
    for(int i=0;i<n;i++){
        if(nums[i]) nums[idx++]=nums[i];
    }
    while(idx<n) nums[idx++]=0;
}
```

Searching an element in a sorted array [TC:O(log(n)) & SC:O(1)]

```
int searchInSorted(int arr[], int N, int K){
    int l=0,h=N-1;
    while(l<=h){
        int mid=(l+h)/2;
        if(arr[mid]==K) return 1;
        else if(arr[mid]<K) l=mid+1;
        else h=mid-1;
    }
    return -1;
}
```

Union of Two Sorted Arrays [TC:O(n+m) & SC:O(n+m)]

```
vector<int> findUnion(int arr1[], int arr2[], int n, int m){
    vector<int> ans;
    int i=0,j=0;
    while(i<n || j<m){
        if(i<n && j<m){
            if(arr1[i]<arr2[j]) {
                if(ans.empty() || ans.back()!=arr1[i]) ans.push_back(arr1[i]);
                i++;
            }
            else {
                if(ans.empty() || ans.back()!=arr2[j])ans.push_back(arr2[j]);
                j++;
            }
        }
        else if(i<n) {
            if(ans.empty() || ans.back()!=arr1[i]) ans.push_back(arr1[i]);
            i++;
        }
        else {
            if(ans.empty() || ans.back()!=arr2[j]) ans.push_back(arr2[j]);
            j++;
        }
    }
    return ans;
}
```

Intersection of Two Sorted Arrays [TC:O(n+m) & SC:O(min(n,m))]

```
vector<int> findIntersection(vi arr1, vi arr2, int n, int m){
    vector<int> ans;
    int i=0,j=0;
    while(i<n && j<m){
        if(arr1[i]<arr2[j]) i++;
        else if(arr1[i]>arr2[j]) j++;
        else {
            if(ans.empty() || ans.back()!=arr2[j]) ans.push_back(arr2[j]);
            i++;
            j++;
        }
    }
    return ans;
}
```

Missing Number [TC:O(n) & SC:O(1)]

```
int missingNumber(vector<int>& nums) {
    int n=nums.size();
    int ans=n;
    for(int i=0;i<n;i++) ans^=i^nums[i];
    return ans;
}
```

```
int missingNumber(vector<int>& nums) {
    int n=nums.size();
    int ans=n*(n+1)/2;
    for(int i=0;i<n;i++) ans-=nums[i];
    return ans;
}
```

Max Consecutive Ones [TC:O(n) & SC:O(1)]

```
int findMaxConsecutiveOnes(vector<int>& nums) {
    int n=nums.size();
    int curr=0,ans=0;
    for(int i=0;i<n;i++){
        if(nums[i]) curr++;
        else{
            ans=max(ans,curr);
            curr=0;
        }
    }
    ans=max(ans,curr);
    return ans;
}
```

Find the number that appears once & all other twice [TC:O(n) & SC:O(1)]

```
int singleNumber(vector<int>& nums) {
    int ans=0;
    for(auto it:nums) ans^=it;
    return ans;
}
```

Longest subarray with given sum (Positives) [TC:O(n) & SC:O(1)]

```
int subarray(vector<int>& nums,int target) {
    int n=nums.size();
    int ans=0,sum=0,prv=0;

    for(int i=0;i<n;i++){
        sum+=nums[i];
        while(sum>target){
            sum-=nums[prv];
            prv++;
        }
        if(sum==target) ans=max(ans,i-prv+1);
    }
    return ans;
}
```

Longest subarray with given sum(Positives + Negatives) [TC:O(n) & SC:O(1)]

```
int lenOfLongSubarr(int arr[],  int N, int K) {
    map<ll,int> mp;
    ll sum=0;
    int ans=0;

    for(int i=1;i<=N;i++){
        sum+=arr[i-1];
        if(sum==K) ans=max(ans,i);
        if(mp[sum-K]) ans=max(ans,i-mp[sum-K]);
        if(!mp[sum]) mp[sum]=i;
    }
    return ans;
}
```

Step 3.2: Medium

Two Sum [TC:O(n*log(n)) & SC:O(n)]

```
vector<int> twoSum(vector<int>& nums, int target) {
    map<int,int> mp;
    vector<int> ans;
    int n=nums.size();

    for(int i=1;i<=n;i++){
        if(mp[target-nums[i-1]]) {
            ans={i-1,mp[target-nums[i-1]]-1};
            break;
        }
        mp[nums[i-1]]=i;
    }
    return ans;
}
```

Sort an array of 0's 1's and 2's [TC:O(n) & SC:O(1)]

```
void sortColors(vector<int>& nums) {
    int n=nums.size();
    int i=0,j=0,k=n-1;

    while(j<=k){
        if(nums[j]==0) {
            swap(nums[i],nums[j]);
            i++;
            j++;
        }
        else if(nums[j]==2){
            swap(nums[k],nums[j]);
            k--;
        }
        else j++;
    }
}
```

Majority Element (>n/2 times) [TC:O(n) & SC:O(1)]

```
int majorityElement(vector<int>& nums) {
    int n=nums.size();
    int curr,ct=0;
    for(int i=0;i<n;i++){
        if(ct){
            if(curr==nums[i]) ct++;
            else ct--;
        }
        else{
            ct++;
            curr=nums[i];
        }
    }
    return curr;
}
```

Kadane's Algorithm, maximum subarray sum [TC:O(n) & SC:O(1)]

```
int maxSubArray(vector<int>& nums) {
    int n=nums.size();
    int sum=0,ans=INT_MIN;

    for(int i=0;i<n;i++){
        sum+=nums[i];
        ans=max(ans,sum);
        if(sum<0) sum=0;
    }
    return ans;
}
```

Print the subarray with maximum sum [TC:O(n) & SC:O(1)]

```
int maxSubArray(vector<int>& nums) {
    int n=nums.size();
    pair<int,int> idx;
    int sum=0,ans=INT_MIN,j=0;

    for(int i=0;i<n;i++){
        if(sum == 0) j=i;
        sum+=nums[i];

        if(sum>ans) {
            ans=sum;
            idx={j,i};
        }

        if(sum<0) sum=0;
    }

    for(int i=idx.first;i<=idx.second;i++) cout<<nums[i]<<" ";
    cout<<endl;
    return ans;
}
```

Best Time to Buy and Sell Stock [TC:O(n) & SC:O(1)]

```
int maxProfit(vector<int>& prices) {
    int n=prices.size();
    int mn=prices[0];
    int ans=0;

    for(int i=1;i<n;i++){
        ans=max(ans,prices[i]-mn);
        mn=min(mn,prices[i]);
    }
    return ans;
}
```

Rearrange the array in alternating +ve and -ve items [TC:O(n) & SC:O(n)]

```
vector<int> rearrangeArray(vector<int>& nums) {
    int n=nums.size();
    vector<int> ans(n);
    int pos=0,neg=1;

    for(int i=0;i<n;i++){
        if(nums[i]>0) {ans[pos]=nums[i];pos+=2;}
        else if(nums[i]<0) {ans[neg]=nums[i];neg+=2;}
    }
    return ans;
}
```

Next Permutation [TC:O(n) & SC:O(1)]

```
void nextPermutation(vector<int>& nums) {
    int n=nums.size();
    int i=n-1;

    while(i && nums[i-1]>=nums[i]) i--;
    int idx=i-1;

    if(i){
        while(i<n && nums[i]>nums[idx]) i++;
        swap(nums[idx],nums[i-1]);
    }
```

```
    reverse(nums.begin()+idx+1,nums.end());
    return ;
}
```

Leaders in an array [TC:O(n) & SC:O(1)]

```
vector<int> leaders(int a[], int n){
    int mx=0;
    vector<int> ans;

    for(int i=n-1;i>=0;i--){
        if(a[i]>=mx){
            ans.push_back(a[i]);
            mx=a[i];
        }
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

Longest Consecutive Sequence [TC:O(n*log(n)) & SC:O(n)]

```
int longestConsecutive(vector<int>& nums) {
    set<int> s;
    for(auto it:nums) s.insert(it);

    int prv=INT_MIN,ct=0,ans=0;
    for(auto it:s){
        if(prv+1==it) ct++;
        else {
            ans=max(ans,ct);
            ct=1;
        }
        prv=it;
    }
    ans=max(ans,ct);
    return ans;
}
```

Set Matrix Zeroes [TC:O(n*m) & SC:O(1)]

```
void setZeroes(vector<vector<int>>& matrix) {
    int n=matrix.size();
    int m=matrix[0].size();
    int row0=1;
    //matrix[0][0] store value of col0

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(matrix[i][j]==0){
                if(i==0) row0=0;
                else matrix[i][0]=matrix[0][j]=0;
            }
        }
    }


    for(int i=1;i<n;i++){
        for(int j=1;j<m;j++){
            if(matrix[i][0]==0 || matrix[0][j]==0) matrix[i][j]=0;
        }
    }

    if(matrix[0][0]==0) for(int i=1;i<n;i++) matrix[i][0]=0;
    if(row0==0) for(int j=0;j<m;j++) matrix[0][j]=0;
}
```

Rotate Matrix by 90 degrees [TC:O(n*m) & SC:O(1)]

```
/*
By observation, we see that the first column of the original matrix Is
the reverse of the first row of the rotated matrix, so that's why we
transpose the matrix and then reverse each row, and since we are
making
changes in the matrix itself space complexity gets reduced to O(1).
*/
```

```
void rotate(vector<vector<int>>& matrix) {
    int n = matrix.size();

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
          swap(matrix[i][j], matrix[j][i]);
        }
    }

    for (int i = 0; i < n; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    }
}
```

Print the matrix in spiral manner [TC:O(n*m) & SC:O(n*m)]

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    int r1=0,r2=matrix.size()-1,c1=0,c2=matrix[0].size()-1;
    vector<int> ans;
    while(r1<=r2 && c1<=c2){
        for(int j=c1;j<=c2;j++) ans.push_back(matrix[r1][j]);
        r1++;
        if(r1>r2) break;
        for(int i=r1;i<=r2;i++) ans.push_back(matrix[i][c2]);
        c2--;
        if(c1>c2) break;
        for(int j=c2;j>=c1;j--) ans.push_back(matrix[r2][j]);
        r2--;
        for(int i=r2;i>=r1;i--) ans.push_back(matrix[i][c1]);
        c1++;
    }
    return ans;
}
```

Find number of subarrays with sum K [TC:O(n*log(n)) & SC:O(n)]

```
int subarraySum(vector<int>& nums, int k) {
    map<int,int> mp;
    int n=nums.size();
    int sum=0,ans=0;
    mp[0]=1;              // if(sum==k) ans++;

    for(int i=0;i<n;i++){
        sum+=nums[i];
        ans+=mp[sum-k];
        mp[sum]++;
    }
    return ans;
}
```

Step 3.3: Hard

Pascal's Triangle [TC:O(n^2) & SC:O(n^2)]

```
vector<vector<int>> generate(int numRows) {
    vector<vector<int>> ans;

    for(int i=0;i<numRows;i++){
        vector<int> curr(i+1,1);

        for(int j=1;j<i;j++){
            curr[j]=ans[i-1][j-1]+ans[i-1][j];
        }
        ans.push_back(curr);
    }
    return ans;
}
```

Majority Element (n/3 times) [TC:O(n) & SC:O(1)]

```
vector<int> majorityElement(vector<int>& nums) {
    int n=nums.size();
    int a,b,ct1=0,ct2=0;

    for (int i = 0; i < n; i++) {
        if (nums[i] == a) ct1++;
        else if (nums[i] == b) ct2++;
        else if (ct1 == 0) {
            a = nums[i];
            ct1 = 1;
        }
        else if (ct2 == 0) {
            b = nums[i];
            ct2 = 1;
        }
        else {
            ct1--;
            ct2--;
        }
    }

    ct1=ct2=0;
    for(int i=0;i<n;i++){
        if(a==nums[i]) ct1++;
        else if(b==nums[i]) ct2++;
    }

    vector<int> ans;
    if(ct1>n/3) ans.push_back(a);
    if(ct2>n/3) ans.push_back(b);
    return ans;
}
```

3-Sum Problem [TC:O(n^2) & SC:O(3*k)]

```
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++){
        int j=i+1,k=n-1;
        while(j<k){
            if(nums[i]+nums[j]+nums[k]==0) {
                ans.push_back({nums[i],nums[j],nums[k]});
                while(j<k && nums[j]==nums[j+1]) j++;
                while(j<k && nums[k-1]==nums[k]) k--;
                j++;k--;
            }
            else if(nums[i]+nums[j]+nums[k]>0) k--;
            else j++;
        }
        while(i<n-1 && nums[i]==nums[i+1]) i++;
    }
    return ans;
}
```

4-Sum Problem [TC:O(n^3) & SC:O(4*k)]

```
vector<vector<int>> fourSum(vector<int>& nums, int target) {
    int n=nums.size();
    sort(nums.begin(),nums.end());

    vector<vector<int>> ans;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            int k=j+1,l=n-1;
            while(k<l){
                long sum=nums[i]+0ll+nums[j]+nums[k]+nums[l];
                if(sum==target) {
                    ans.push_back({nums[i],nums[j],nums[k],nums[l]});
                    while(k<l && nums[k]==nums[k+1]) k++;
                    while(k<l && nums[l-1]==nums[l]) l--;
                    k++;l--;
                }
                else if(sum>target) l--;
                else k++;
            }
            while(j<n-1 && nums[j]==nums[j+1]) j++;
        }
        while(i<n-1 && nums[i]==nums[i+1]) i++;
    }
    return ans;
}
```

Count number of subarrays with given xor K [TC:O(n*log(n)) & SC:O(n)]

```
int subarrayXOR(vector<int> arr, int N, int K) {
    map<int,int> mp;
    int xr=0,ans=0;
    mp[0]=1;              //if(xr==K) ans++;

    for(int i=0;i<N;i++){
        xr^=arr[i];
        ans+=mp[xr^K];
        mp[xr]++;
    }
    return ans;
}
```

Count number of subset with given xor K [TC:O(n*m) & SC:O(n*m)]

```
int subsetXOR(vector<int> arr, int N, int K) {
    vector<vector<int>> dp(N+1,vector<int>(128,0));
    dp[0][0]=1;

    for(int i=1;i<=N;i++){
        for(int j=0;j<128;j++){
            dp[i][j]=dp[i-1][j]+dp[i-1][j^arr[i-1]];
        }
    }

    return dp[N][K];
}
```

Merge Overlapping Subintervals [TC:O(n*log(n)) & SC:O(n)]

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    int n=intervals.size();
    sort(intervals.begin(),intervals.end());
    vector<vector<int>> ans;

    vector<int> curr=intervals[0];
    for(int i=1;i<n;i++){
        if(intervals[i][0]<=curr[1]) curr[1]=max(curr[1],intervals[i][1]);
        else{
            ans.push_back(curr);
            curr=intervals[i];
        }
    }
    ans.push_back(curr);
    return ans;
}
```

```
bool comp(vector<int> a,vector<int> b){
    if(a[1]==b[1]) return a[0]<b[0];
    return a[1]<b[1];
}

vector<vector<int>> merge(vector<vector<int>>& intervals) {
    int n=intervals.size();
    sort(intervals.begin(),intervals.end(),comp);
    vector<vector<int>> ans;

    vector<int> curr=intervals[n-1];
    for(int i=n-2;i>=0;i--){
       if(intervals[i][1]>=curr[0]) curr[0]=min(curr[0],intervals[i][0]);
        else{
            ans.push_back(curr);
            curr=intervals[i];
        }
    }
    ans.push_back(curr);
    return ans;
}
```

Merge two sorted arrays without extra space [TC:O(n+m) & SC:O(1)]

```
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    int i=m-1,j=n-1,k=m+n-1;
    while(i>=0 && j>=0){
        if(nums2[j]>=nums1[i]) nums1[k--]=nums2[j--];
        else nums1[k--]=nums1[i--];
    }
    while(j>=0) nums1[k--]=nums2[j--];
}
```

GFG [TC:O(n*m) & SC:O(1)]

```
void merge(long long arr1[], long long arr2[], int n, int m) {
    for (int i = 0; i < n; i++){
        if (arr1[i] > arr2[0]) swap(arr1[i],arr2[0]);

        int k, first = arr2[0];
        for (k = 1; k < m && arr2[k] < first; k++) arr2[k - 1] = arr2[k];
        arr2[k - 1] = first;
    }
}
```

[TC:O(n+m+n*log(n)+m*log(m)) & SC:O(1)]

```
void merge(long long arr1[], long long arr2[], int n, int m) {
    int i=n-1,j=0;
    while(i>=0 && j<m){
        if(arr1[i] > arr2[j])swap(arr1[i--],arr2[j++]);
        else break;
    }
    sort(arr1,arr1+n);
    sort(arr2,arr2+m);
}
```

➡️ Gap Method******* [TC:O((n+m)*log(m+n)) & SC:O(1)]

```
void merge(long long arr1[], long long arr2[], int n, int m) {
    int gap=(m+n+1)/2;

    while(gap){
        for(int i=0;i+gap<n+m;i++){
            if(i+gap<n){
                if(arr1[i]>arr1[i+gap]) swap(arr1[i],arr1[i+gap]);
            }
            else if(i>=n){
                if(arr2[i-n]>arr2[i+gap-n]) swap(arr2[i-n],arr2[i+gap-n]);
            }
            else{
                if(arr1[i]>arr2[i+gap-n]) swap(arr1[i],arr2[i+gap-n]);
            }
        }
        if(gap==1) break;
        gap=(gap+1)/2;
    }
}
```

Find the repeating and missing number [TC:O(n) & SC:O(1)]

```
int *findTwoElement(int *arr, int n) {
    int xr=0;
    for(int i=1;i<=n;i++) xr^=arr[i-1]^i;

    int lsbn=(xr & (-xr));
    int a=0,b=0;

    for(int i=1;i<=n;i++){
        if(lsbn & arr[i-1]) a^=arr[i-1];
        else b^=arr[i-1];

        if(lsbn & i) a^=i;
        else b^=i;
    }
```

```
    for(int i=0;i<n;i++){
        if(arr[i]==b) break;
        if(arr[i]==a){
            swap(a,b);
            break;
        }
    }

    int*ans = new int[2] {b,a};
    return ans;
}
```

[TC:O(n) & SC:O(1)]

```
int *findTwoElement(int *arr, int n) {
    ll sum=0,sumsq=0;
    for(ll i=1;i<=n;i++){
        sum+=i-arr[i-1];
        sumsq+=i*i-arr[i-1]*1ll*arr[i-1];
    }

    ll div=sumsq/sum;
    //sumsq = a^2 - b^2
    //sum   = a - b
    //div   = a + b
    int a = (sum+div)/2;
    int b = div - a;
    int *ans = new int[2]{b,a};
    return ans;
}
```

Count Inversions [TC:O(n*log(n)) & SC:O(n)]

```
ll merge(int st,int mid,int en,ll arr[]){
    ll ans=0,temp[en-st+1];
    int i=st,j=mid+1,k=0;

    while(i<=mid && j<=en){
        if(arr[i]<=arr[j]) temp[k++]=arr[i++];
        else{
            ans+=(mid-i+1);
            temp[k++]=arr[j++];
        }
    }

    while(i<=mid) temp[k++]=arr[i++];
    while(j<=en)  temp[k++]=arr[j++];

    for(int k=0;k<=en-st;k++) arr[k+st]=temp[k];
    return ans;
}

ll mergeSort(int st,int en, ll arr[]){
    if(st>=en) return 0;
    int mid=(st+en)/2;
    ll ans=0;

    ans+=mergeSort(st,mid,arr);
    ans+=mergeSort(mid+1,en,arr);
    ans+=merge(st,mid,en,arr);
    return ans;
}

long long int inversionCount(long long arr[], long long N){
    return mergeSort(0,N-1,arr);
}
```

Count Reverse Pairs [TC:O(n*log(n)) & SC:O(n)]

```
int merge(int st,int mid,int en,vector<int> &arr){
    int ans=0,temp[en-st+1];
    int i=st,j=mid+1,k=0,curr=st;

    while(i<=mid && j<=en){
        if(arr[i]<=arr[j]) temp[k++]=arr[i++];
        else{
            while(curr<=mid && arr[curr] <= 2*arr[j]) curr++;
            ans+=(mid-curr+1);
            temp[k++]=arr[j++];
        }
    }

    while(i<=mid) temp[k++]=arr[i++];
    while(j<=en)  temp[k++]=arr[j++];

    for(int k=0;k<=en-st;k++) arr[k+st]=temp[k];
    return ans;
}

int mergeSort(int st,int en, vector<int> &arr){
    if(st>=en) return 0;
    int mid=(st+en)/2;
    int ans=0;

    ans+=mergeSort(st,mid,arr);
    ans+=mergeSort(mid+1,en,arr);
    ans+=merge(st,mid,en,arr);
    return ans;
}

int countRevPairs(int n, vector<int> arr) {
    return mergeSort(0,n-1,arr);
}
```

Maximum Product Subarray [TC:O(n) & SC:O(1)]

```
long long maxProduct(vector<int> nums, int n) {
    long long prod =1 ,ans = nums[0];

    //leftMaximum
    for(int i=0;i<n;i++) {
        prod *= nums[i];
        ans = max(ans,prod);
        if(nums[i] == 0) prod = 1;
    }

    prod = 1;

    //rightMaximum
    for(int i=n-1;i>=0;i--) {
        prod *= nums[i];
        ans = max(ans,prod);
        if(nums[i] == 0) prod=1;
    }

    return ans;
}
```

➡️ Like Kadane's Algorithm****[TC:O(n) & SC:O(1)]

```
long long maxProduct(vector<int> arr, int n) {
    long long pos = arr[0],neg = arr[0],ans = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] < 0) swap(pos, neg);

        pos = max(1ll * arr[i], pos * arr[i] );
        neg = min(1ll * arr[i], neg * arr[i] );

        ans = max(ans, pos);
    }
    return ans;
}
```

Step 4: Binary Search [1D, 2D Arrays, Search Space]

Step 4.1: Learning BS on 1D Arrays

Binary Search [TC:O(log(n)) & SC:O(1)]

```
int binarySearch(vector<int>& nums, int target) {
    int l=0,h=nums.size()-1,mid;
    while(l<=h){
        mid=(l+h)/2;
        if(nums[mid]==target) return mid;
        else if(nums[mid]<target) l=mid+1;
        else h=mid-1;
    }
    return -1;
}
```

Find the row with maximum number of 1's [TC:O(n*log(m)) & SC:O(1)]

```
int rowWithMax1s(vector<vector<int> > arr, int n, int m) {
    int l=0,h=m-1,mid,ans=-1;
    while(l<=h){
        mid=(l+h)/2;
        int curr=-1;

        for(int i=0;i<n;i++){
            if(arr[i][mid]) {
                ans=curr=i;
                break;
            }
        }

        if(curr==-1) l=mid+1;
        else h=mid-1;
    }
    return ans;
}
```

[TC:O(n+m) & SC:O(1)]

```
int rowWithMax1s(vector<vector<int> > arr, int n, int m) {
    int ans=-1,j=m;
    for(int i=0;i<n;i++){
        if(arr[i][j-1]){
            while(j && arr[i][j-1]) j--;
            ans=i;
        }
        if(j==0) break;
    }
    return ans;
}
```

Floor in a Sorted Array [TC:O(log(n)) & SC:O(1)]

```
int findFloor(vector<long long> v, long long n, long long x){
    int low=0,high=n-1;

    while(low<=high){
        int mid = (low+high)/2;
        if(v[mid]>x) high = mid -1;
        else low = mid + 1;
    }

    return high ;
}
```

Ceil The Floor **[TC:O(n) & SC:O(1)]**

```
pair<int, int> getFloorAndCeil(int arr[], int n, int x) {
    int mn=-1,mx=INT_MAX;
    for(int i=0;i<n;i++){
        if(arr[i]>=x) mx=min(mx,arr[i]);
        if(arr[i]<=x) mn=max(mn,arr[i]);
    }
    if(mx==INT_MAX) mx=-1;
    return {mn,mx} ;
}
```

Lower Bound [TC:O(log(n)) & SC:O(1)]

```
int searchInsert(vector<int>& arr, int target) {
    int low = 0,high = arr.size()-1;
    while(low<=high){
        int mid = (low+high)/2;

        if(arr[mid]==target) return mid;
        else if(arr[mid]<target) low=mid+1;
        else high = mid-1;
    }
    return low;
}
```

Check if Input array is sorted [TC:O(n) & SC:O(1)]

```
bool arraySortedOrNot(int arr[], int n) {
    for(int i=1;i<n;i++){
        if(arr[i-1]>arr[i]) return false;
    }
    return true;
}
```

Find First and Last Position of Element in Sorted Array [TC:O(log(n)) & SC:O(1)]

```
int lower_idx(vector<int>& nums, int target){
    int n=nums.size();
    int low = 0, high = n-1;

    while(low<=high){
        int mid = (low+high)/2;
        if(nums[mid]>=target) high = mid-1;
        else low=mid+1;
    }
    if(low<n && nums[low]==target) return low;
    return -1;
}

int upper_idx(vector<int>& nums, int target){
```

```
    int low = 0, high = nums.size()-1;

    while(low<=high){
        int mid = (low+high)/2;
        if(nums[mid]<=target) low=mid+1;
        else high = mid-1;
    }
    if(high>=0 && nums[high]==target) return high;
    return -1;
}

vector<int> searchRange(vector<int>& nums, int target) {
    vector<int> ans(2);
    ans[0] = lower_idx(nums,target);
    ans[1] = upper_idx(nums,target);
    return ans;
}
```

➡️ Using STL

```
vector<int> searchRange(vector<int>& nums, int target) {
    vector<int> ans={-1,-1};
    int idx =
lower_bound(nums.begin(),nums.end(),target)-nums.begin();
    if(idx==nums.size() || nums[idx]!=target) return ans;

    ans[0] = idx;
    ans[1] = upper_bound(nums.begin(),nums.end(),target)-nums.begin()-1;
    return ans;
}
```

Number of occurrence [TC:O(log(n)) & SC:O(1)]

```
int count(int arr[], int n, int x) {
    return upper_bound(arr,arr+n,x)-lower_bound(arr,arr+n,x);
}
```

**Find Peak Element [TC:O(log(n)) & SC:O(1)]

```
int findPeakElement(vector<int>& nums) {
    int low=0,high=nums.size()-1;

    // Two Parts [low:mid] and [mid+1,high]
    // If (mid) ele is smaller than (mid+1) ele,then (mid+1)
    // may be the pick element so ignore 1st half of the elements
    // else ignore 2nd half of the elements

    while(low<high){
        int mid=(low+high)/2;
        if(nums[mid]<nums[mid+1]) low=mid+1;
        else high=mid;
    }

    return low;
}
```

Search in Rotated Sorted Array [TC:O(log(n)) & SC:O(1)]

```
int search(vector<int>& nums, int target) {
    int low=0,high=nums.size()-1;

    while(low<=high){
        int mid = (low+high)/2;
        if(nums[mid]==target) return mid;

        if(nums[low]<=nums[mid]){ //1st part is sorted [low:mid]
            if(target>=nums[low] && target<nums[mid]) high=mid-1;
            else low = mid+1;
        }
        else{  //2nd part is sorted [mid+1:high]
            if(target>=nums[mid+1] && target<=nums[high]) low=mid+1;
            else high = mid-1;
        }
    }
    return -1;
}
```

Search in Rotated Sorted Array II [TC:O(log(n)) & SC:O(1)]

```
bool search(vector<int>& nums, int target) {
    int low=0,high=nums.size()-1;

    while(low<=high){
        int mid = (low+high)/2;
        if(nums[mid]==target) return true;

        if(nums[low]==nums[high] && nums[low]==nums[mid]) {
            low++;
            high--;
        }
        else if(nums[low]<=nums[mid]){ //1st part is sorted [low:mid]
            if(target>=nums[low] && target<nums[mid]) high=mid-1;
            else low = mid+1;
        }
        else{  //2nd part is sorted [mid+1:high]
            if(target>=nums[mid+1] && target<=nums[high]) low=mid+1;
            else high = mid-1;
        }
    }
    return false;
}
```

Step 4.2: Applying BS on 2D Arrays

Search in a 2 D matrix [TC:O(log(n*m)) & SC:O(1)]

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int n=(int)matrix.size();
    int m=(int)matrix[0].size();

    int l=0,h=n*m-1;

    while(l<=h){
        int mid=l+(h-l)/2;
        int r=mid/m;
        int c=mid%m;

        if(matrix[r][c]==target) return true;
        else if(matrix[r][c]<target) l=mid+1;
        else h=mid-1;
    }
    return false;
}
```

Find a Peak Element II **[TC:O(log(n*m)) & SC:O(1)]**

//

//

Step 5: Strings [Basic and Medium]

Step 5.1: Basic and Easy String Problems

//

Step 5.2: Medium String Problems

//

Implement Atoi [TC:O(n) & SC:O(1)] (Iterative)

For Recursive Click here

```
int myAtoi(string s) {
    int n=s.size();
    int ans=0,i=0;
    bool neg=false;

    while(i<n && s[i]==' ') i++;
    if(i==n) return ans;

    if(s[i]=='-') {neg=true;i++;}
    else if(s[i]=='+') i++;
    while(i<n && s[i]>='0' && s[i]<='9'){
        int curr = s[i] - '0' ;
        if(neg){if(-ans<INT_MIN/10 ||
                (-ans==INT_MIN/10 && -curr<=INT_MIN%10)) return
INT_MIN;}
        else if (ans>INT_MAX/10 ||
                (ans==INT_MAX/10 && curr>=INT_MAX%10)) return INT_MAX;

        ans = ans * 10 + curr ;
        i++;
    }
    if(neg) ans*=-1;

    return ans;
}
```

//

Step 6: Learn LinkedList [Single/Double LL, Medium, Hard]

Step 6.1: Learn 1D LinkedList

Introduction to LinkedList [TC:O(n) & SC:O(n)]

👉 Method 1

```
Node* constructLL(vector<int>& arr) {
    int n=arr.size();

    Node* head = new Node(arr[0]);
    Node* temp = head;

    for(int i=1;i<n;i++){
        temp->next = new Node(arr[i]);
        temp=temp->next;
    }
    return head;
}
```

👉 Method 2

```
Node* constructLL(vector<int>& arr) {
    Node* head = NULL, * temp ;

    for(auto it:arr){
        if(!head){
            head = new Node(it);
            temp = head;
        }
        else {
            temp->next = new Node(it);
            temp=temp->next;
        }
    }
    return head;
}
```

Inserting a node in LinkedList[TC: {O(1) & O(n)} & SC:O(1)]

```
//Function to insert a node at the beginning of the linked list.
Node *insertAtBegining(Node *head, int x) {
   Node* new_node = new Node(x);
   new_node->next = head;
   head = new_node;
   return head;
}

//Function to insert a node at the end of the linked list.
Node *insertAtEnd(Node *head, int x)  {
   Node* new_node = new Node(x);
   if(!head) return new_node;

   Node* temp = head;
   while(temp->next) temp = temp->next;
   temp->next = new_node;
   return head;
}
```

Deleting a node in LinkedList[TC:O(n) & SC:O(1)]

```
Node *deleteNode(Node *head, int x){
    Node *temp = head;

    if (x == 1){
        head = head->next;
        delete temp;
        return head;
    }

    for (int i = 2; i < x; i++) temp = temp->next;

    Node *to_delete = temp->next;
    temp->next = temp->next->next;
    delete to_delete;
    return head;
}
```

Find the length of the linkedlist[TC:O(n) & SC:O(1)]

```
int getCount(struct Node* head){
    int ct = 0;
    while(head) {
        ct++;
        head = head->next;
    }
    return ct;
}
```

Search an element in the LL [TC:O(n) & SC:O(1)]

```
bool searchKey(int n, struct Node* head, int key) {
    while(head){
        if(head->data==key) return true;
        head = head -> next;
    }
    return false;
}
```

Step 6.2: Learn Doubly LinkedList

Introduction to DLL, learn about struct, and how is node represented [TC:O(n) & SC:O(n)]

```
Node* constructDLL(vector<int>& arr) {
    int n=arr.size();

    Node* head = new Node(arr[0]);
    Node* temp = head;

    for(int i=1;i<n;i++){
        temp->next = new Node(arr[i]);
        temp->next->prev = temp;
        temp=temp->next;
    }
    return head;
}
```

Insert a node in DLL [TC:O(n) & SC:O(1)]

```
void addNode(Node *head, int pos, int data){
   Node *temp = new Node(data);

    for (int i = 0; i < pos; i++) head = head->next;

    temp -> next = head->next;
    head -> next = temp;

    if (temp->next) temp -> next -> prev = temp;
    head -> next -> prev = head;
}
```

Delete a node in DLL [TC:O(n) & SC:O(1)]

```
Node* deleteNode(Node *head, int x){
    Node *temp = head;

    if (x == 1){
        head = head->next;
        if(head) head->prev = NULL ;
        delete temp;
        return head;
    }

    for (int i = 1; i < x; i++) temp = temp->next;

    temp->prev->next = temp->next;
    if(temp->next) temp->next->prev = temp->prev;
    delete temp;
    return head;
}
```

Reverse a DLL

➡️ Iterative **[TC:O(n) & SC:O(1)]**

```
Node* reverseDLL(Node * head){
    Node* before = NULL;

    while(head){
        head -> prev = head -> next;
        head -> next = before;

        before = head;
        head = head-> prev;
    }
    return before;
}
```

➡️ Recursive **[TC:O(n) & SC:O(n)]**

```
Node* reverseDLL(Node * head){
    if(!head->next) {
        head->prev = NULL;
        return head;
    }

    Node* new_head = reverseDLL(head->next);

    head -> prev = head -> next ;
    head -> next -> next = head;
    head -> next = NULL;

    return new_head;
}
```

Step 6.3: Medium Problems of LL

Middle of a LinkedList [TC:O(n) & SC:O(1)]

```
int getMiddle(Node *head){
    Node* slow = head;
    Node* fast = head;

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

    return slow->data;
}
```

Reverse a LinkedList

➡️ Iterative [TC:O(n) & SC:O(1)]

```
struct Node* reverseList(struct Node *head){
    Node* before = NULL,*after = NULL ;

    while(head){
        after = head->next;
        head->next = before;
        before = head;
        head = after;
    }
    return before;
}
```

➡️ Recursive [TC:O(n) & SC:O(n)]

```
struct Node* reverseList(struct Node *head){
    if(!head->next) return head;

    Node* new_head = reverseList(head->next);
    head->next->next = head;
    head->next = NULL;
    return new_head;
}
```

Detect a loop in LL [TC:O(n) & SC:O(1)]

```
bool detectLoop(Node* head){
    Node* slow = head;
    Node* fast = head;

    while(fast && fast->next){
        slow = slow->next;
        fast = fast->next->next;
        if(slow==fast) return true;
    }
    return false;
}
```

Remove loop in Linked List [TC:O(n) & SC:O(1)]

👉 Method 1

```
void removeLoop(Node* head){
    Node* slow = head,* fast = head;

    while(fast && fast->next){
        slow = slow->next;
        fast = fast->next->next;
        if(slow==fast) break;
    }

    if(!fast || !fast->next) return;

    if(fast==head){  // if loop is at head
        while(slow->next!=head) slow=slow->next;
    }
    else{
        fast=head;
        while(slow->next!=fast->next){
            slow = slow->next;
            fast = fast->next;
        }
    }
    slow->next=NULL;
}
```

👉 Method 2

```
void removeLoop(Node* head){
    Node* slow = head,* fast = head;

    while(fast && fast->next){
        slow = slow->next;
        fast = fast->next->next;
        if(slow==fast) break;
    }

    if(!fast || !fast->next) return;

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

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

    slow->next=NULL;
}
```

Length of Loop in LL [TC:O(n) & SC:O(1)]

```
int countNodesinLoop(struct Node *head){
    Node* slow = head,* fast = head;

    while(fast && fast->next){
        slow = slow->next;
        fast = fast->next->next;
        if(slow==fast) break;
    }

    if(!fast || !fast->next) return 0;

    int count = 1;
    while(slow->next!=fast) {
        count++;
        slow = slow->next;
    }

    return count;
}
```

Check if LL is palindrome or not [TC:O(n) & SC:O(1)]

👉 Method 1

```
struct Node* reverseList(struct Node *head){
    Node* before = NULL,*after = NULL ;

    while(head){
        after = head->next;
        head->next = before;
        before = head;
        head = after;
    }
    return before;
}

bool isPalindrome(Node *head){
    Node* slow = head,* fast = head;

    while(fast->next && fast->next->next){
        fast = fast->next->next;
        slow = slow->next;
    }
    //slow point first middle element in even

    slow->next = reverseList(slow->next);
    slow = slow -> next;
    fast = head;

    while(slow) {
        if(slow->data != fast->data) return false;
        slow = slow->next;
        fast = fast->next;
    }

    return true;
}
```

👉 Method 2

```
bool isPalindrome(Node *head){
    Node* before = NULL,*after;
    Node* slow = head,* fast = head;

    while(fast && fast->next){
        fast = fast->next->next;
        after = slow->next;          //slow = slow->next

        slow->next = before;
        before = slow;
        slow = after;
    }
    //slow point second middle element in even

    if(!fast) fast = slow;    //even
    else fast = slow->next;   //odd
    slow = before;

    while(slow) {
        if(slow->data != fast->data) return false;
        slow = slow->next;
        fast = fast->next;
    }

    return true;
}
```

Segregate odd and even nodes in LL[TC:O(n) & SC:O(1)]

▶️ Leetcode

```
ListNode* oddEvenList(ListNode* head) {
    if(!head) return NULL;

    ListNode* odd = head;
    ListNode* even = head->next;
    ListNode* evenStart = even;

    while(even && even->next){
        odd->next = odd->next->next;
        even->next = even->next->next;

        odd = odd->next;
        even = even->next;
    }

    odd->next = evenStart;
    return head;
}
```

▶️ GFG

```
Node* divide(int N, Node *head){
    Node* odd = NULL,* even = NULL;
    Node * evenStart = NULL ,*oddStart = NULL;

    while(head){
        if(head->data%2){
            if(odd){
                odd->next = head;
                odd = odd->next;
            }
            else {
                odd = head;
                oddStart = odd;
            }
        }
        else{
            if(even){
                even->next = head;
                even = even->next;
            }
            else {
                even = head;
                evenStart = even;
            }
        }

        head = head->next;
    }

    if(!evenStart) return oddStart;

    even->next = oddStart;
    if(odd) odd ->next = NULL;

    return evenStart;
}
```

Remove Nth node from the back of LL[TC:O(n) & SC:O(1)]

▶️ LeetCode

```
ListNode* removeNthFromEnd(ListNode* head, int n) {
    ListNode * start = new ListNode();
    start -> next = head;
    ListNode* fast = start,* slow = start;

    while(n--) fast = fast->next;

    while(fast->next){
        fast = fast->next;
        slow = slow->next;
    }

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

    return start->next;
}
```

▶️ GFG

```
int getNthFromLast(Node *head, int n){
    Node* slow = head;
    Node* fast = head;

    while(n--) {
        if(!fast) return -1;
        fast = fast->next;
    }

    while(fast){
        fast = fast->next;
        slow = slow->next;
    }
    return slow->data;
}
```

Delete the middle node of LL[TC:O(n) & SC:O(1)]

```
Node* deleteMid(Node* head){
    if(!head->next){
        delete head;
        return NULL;
    }

    Node* before = NULL;
    Node* slow = head;
    Node* fast = head;

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

    before->next = before->next->next;
    delete slow;
    return head;
}
```

Sort LL that is Sorted Alternatingly [TC:O(n) & SC:O(1)]

```
struct Node* reverseList(struct Node *head){
    Node* before = NULL,*after = NULL ;

    while(head){
        after = head->next;
        head->next = before;
        before = head;
        head = after;
    }
    return before;
}
struct Node* merge_sort(struct Node* head1,struct Node* head2){
```

```
    Node* dummy = new Node(-1);
    Node* temp = dummy;

    while(head1 && head2){
        if(head1->data<head2->data) {
            temp->next = head1;
            head1 = head1->next;
        }
        else{
            temp->next = head2;
            head2 = head2->next;
        }
        temp = temp->next;
    }

    if(head1) temp->next = head1;
    else temp->next = head2;
    return dummy->next;
}

void sort(Node **head){
    Node* odd = *head;
    Node* even = (*head)->next;
    Node* evenStart = even;

    while(even && even->next){
        odd->next = odd->next->next;
        even->next = even->next->next;

        odd = odd->next;
        even = even->next;
    }
    odd->next = NULL;

    evenStart = reverseList(evenStart);
    *head = merge_sort(*head,evenStart);
}
```

Sort LL [TC:O(n*log(n)) & SC:O(1)]

```
ListNode* merge_sort(ListNode* head1,ListNode* head2){
    ListNode* dummy = new ListNode();
    ListNode* temp = dummy;

    while(head1 && head2){
        if(head1->val<head2->val) {
            temp->next = head1;
            head1 = head1->next;
        }
        else{
            temp->next = head2;
            head2 = head2->next;
        }
        temp = temp->next;
    }

    if(head1) temp->next = head1;
    else temp->next = head2;
    return dummy->next;
}

ListNode* sortList(ListNode* head) {
    if(!head || !head->next) return head;

    ListNode* before = NULL;
    ListNode* slow = head;
    ListNode* fast = head;

    while(fast && fast->next){
        before = slow;
        slow = slow->next;
        fast = fast->next->next;
    }
    before->next = NULL;

    ListNode* h1 = sortList(head);
    ListNode* h2 = sortList(slow);
    return merge_sort(h1,h2);
}
```

Sort a LL of 0's 1's and 2's [TC:O(n) & SC:O(1)]

```
Node* segregate(Node *head) {
    Node* zero = new Node(-1);
    Node* one = new Node(-1);
    Node* two = new Node(-1);
    Node* zeroStart = zero;
    Node* oneStart = one;
    Node* twoStart = two;

    while(head){
        if(head->data == 0) {
            zero->next = head;
            zero = zero->next;
        }
        else if(head->data == 1) {
            one->next = head;
            one = one->next;
        }
        else {
            two->next = head;
            two = two->next;
        }
        head = head -> next;
    }

    two->next = NULL;
    one->next = twoStart->next;
    zero->next = oneStart->next;

    return zeroStart->next;
}
```

Intersection Point of Two LL [TC:O(n+m) & SC:O(1)]

```
ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
    ListNode *t1=headA,*t2=headB;
    while(t1!=t2){
        t1=(t1==NULL?headB:t1->next);
        t2=(t2==NULL?headA:t2->next);
    }
    return t1;
}
```

Intersection of Two LL [TC:O(n*log(n)) & SC:O(1)]

```
Node* findIntersection(Node* head1, Node* head2){
    Node* dummy = new Node(-1);
    Node* temp = dummy;
    map<int,int> mp;

    while(head2){
        mp[head2->data]=1;
        head2 = head2->next;
    }

    while(head1){
        if(mp[head1->data]){
            temp->next = head1;
            temp = temp->next;
        }
        head1 = head1->next;
    }

    temp->next = NULL;
    return dummy->next;
}
```

Add 1 to a number represented as linked list [TC:O(n) & SC:O(1)]

```
struct Node* reverseList(struct Node *head){
    Node* before = NULL,*after = NULL ;

    while(head){
        after = head->next;
        head->next = before;
        before = head;
        head = after;
    }
    return before;
}

Node* addOne(Node *head) {
    head = reverseList(head);
    int carry = 1;
    Node* temp = head;

    while(carry){
        temp->data += 1;

        if(temp->data == 10) temp->data = 0;
        else carry = 0;

        if(!temp->next) break;
        temp = temp->next;
    }
    if(carry) temp->next = new Node(carry);

    return reverseList(head);
}
```

Add two numbers represented by LL [TC:O(max(n,m)) & SC:O(1)]

```
struct Node* reverseList(struct Node *head){
    Node* before = NULL,*after = NULL ;

    while(head){
        after = head->next;
        head->next = before;
        before = head;
        head = after;
    }
    return before;
}

struct Node* addTwoLists(struct Node* first, struct Node* second){
    first = reverseList(first);
    second = reverseList(second);
    Node* head = first;
    int carry = 0;

    while(first){
        first->data += carry;
        if(second) first->data += second->data;
        carry = first->data/10;
        first->data%=10;

        if(!first->next) {
            if(!second || !second->next){
                if(carry) first->next = new Node(carry);
                break;
            }
            first->next = second->next;
            second->next = NULL;
        }

        if(second) second = second->next;
        first = first->next;
    }
    return reverseList(head);
}
```

Step 6.4: Medium Problems of DLL

Delete all occurrences of a given key in a DLL [TC:O(n) & SC:O(1)]

```
void deleteAllOccurOfX(struct Node** head_ref, int x) {
    Node* dummy = (Node*)malloc(sizeof(Node));
    dummy->next = NULL;
    Node* curr = dummy;
    Node* temp = *head_ref;

    while(temp){
        if(temp->data!=x){
            curr->next = temp;
            temp->prev = curr;
            curr = curr->next;
        }
        temp = temp->next;
    }
    curr->next = NULL;
    *head_ref = dummy->next;
    if(*head_ref) (*head_ref)->prev=NULL;
}
```

Find pairs with given sum in DLL [TC:O(n) & SC:O(1)]

```
vector<pair<int, int>> findPairsWithGivenSum(Node *head, int target){
    vector<pair<int,int>> ans;
    Node* temp = head;
    while(temp->next) temp = temp->next;

    while(head->data<temp->data){
        int sum = head->data+temp->data;
        if(sum == target) {
            ans.push_back({head->data,temp->data});
            head=head->next;
            temp=temp->prev;
        }
        else if(sum > target) temp=temp->prev;
        else head=head->next;
    }
    return ans;
}
```

Remove duplicates from a sorted DLL [TC:O(n) & SC:O(1)]

```
Node * removeDuplicates(struct Node *head){
    Node* curr=head;
    Node* temp=head->next;

    while(temp){
        if(curr->data!=temp->data){
            curr->next=temp;
            temp->prev=curr;
            curr=curr->next;
        }
        temp=temp->next;
    }
    curr->next=NULL;
    return head;
}
```

Step 6.5: Hard Problems of LL

Reverse LL in group of given size K

➡️Recursive [TC:O(n) & SC:O(n/k)]

```
struct node *reverse (struct node *head, int k){
    node* before = NULL;
    node* temp = head;
    node* after = NULL;

    for(int i=0;i<k && temp;i++){
        after = temp->next;
        temp->next=before;
        before = temp;
        temp=after;
    }

    if(temp) head->next = reverse(temp,k);
    return before;
}
```

➡️Iterative** [TC:O(n) & SC:O(1)]

```
struct node *reverse (struct node *head, int k){
    node* before = NULL;
    node* temp = head;
    node* after = NULL;
    node* last = NULL;

    while(temp){
        before = NULL;
        node* start=temp;

        for(int i=0;i<k && temp;i++){
            after = temp->next;
            temp->next=before;
            before = temp;
            temp=after;
        }
        if(last) last->next = before;
        else head = before;
        last = start;
    }
    return head;
}
```

Rotate a LL [TC:O(n) & SC:O(1)]

```
Node* rotate(Node* head, int k){
    Node* temp = head;
    for(int i=1;i<k;i++) temp=temp->next;
    if(!temp->next) return head;

    Node* new_head = temp->next;
    temp->next = NULL;
    temp = new_head;
    while(temp->next) temp=temp->next;
    temp->next = head;
    return new_head;
}
```

Flattening a LL [TC:O(n*n*m) & SC:O(n*m)]

```
Node* mergeTwoLists(Node* a, Node* b) {
    Node *dummy = new Node(-1);
    Node *temp = dummy;

    while(a && b) {
        if(a->data < b->data) {
            temp->bottom = a;
            a = a->bottom;
        }
        else {
            temp->bottom = b;
            b = b->bottom;
        }
        temp = temp->bottom;
    }

    if(a) temp->bottom = a;
    else temp->bottom = b;

    return dummy -> bottom;
}

Node *flatten(Node *root){
    if (!root || !root->next) return root;
    root->next = flatten(root->next);
    root = mergeTwoLists(root, root->next);
    return root;
}
```

Clone a LL with next and random pointer [TC:O(n) & SC:O(n)]

```
Node *copyList(Node *head){
    Node* temp = head;
    while(temp){
        Node* curr = new Node(temp->data);
        curr->next = temp->next;
        temp->next = curr;
        temp = temp->next->next;
    }

    temp = head;
    while(temp){
        if(temp->arb) temp->next->arb = temp->arb->next;
        temp = temp->next->next;
    }

    temp = head;
    Node* new_head = head->next;
    Node* new_temp = new_head;

    while(new_temp->next){
        temp->next = temp->next->next;
        new_temp->next = new_temp->next->next;

        temp = temp->next;
        new_temp = new_temp->next;
    }
    temp->next = NULL;
    return new_head;
}
```

Step 7: Recursion [PatternWise]

Step 7.1: Get a Strong Hold

Implement Atoi [TC:O(n) & SC:O(n)] (Recursive)

For Iterative Click here

```
void solve(int i,int &ans, string s){
    if(i==s.size()) return;
    if(s[i]<'0' || s[i]>'9') {
        ans=-1;
        return;
    }

    int curr = s[i]-'0';
    ans = ans * 10 + curr;
    solve(i+1,ans,s);
}

int atoi(string str) {
    int ans=0,i=0,sign=1;
    if(str[0]=='-') {sign = -1;i++;}
    solve(i,ans,str);
    if(ans==-1) return ans;
    return sign * ans;
}
```

Pow(x,n)

▶️GFG [TC:O(log(R)) & SC:O(log(R))]

For Iterative Click here

```
long long power(int N,int R){
    if(R==0) return 1;

    if(R%2) return N*power(N,R-1)%M;
    return power(N*1ll*N%M, R/2)%M;
}
```

▶️Leetcode [TC:O(log(n)) & SC:O(log(n))]

```
double myPow(double x, int n) {
    if(n<0) {n=abs(n);x=1/x;}
    if(n==0) return 1;

    if(n%2) return x*myPow(x,n-1);
    return myPow(x*x,n/2);
}
```

Count Good numbers[TC:O(log(n)) & SC:O(log(n))]

```
int power(int x,long long n){
    if(n==0) return 1;

    if(n%2) return x*1ll*power(x,n-1)%M;
    return power(x*1ll*x%M, n/2)%M;
}

int countGoodNumbers(long long N) {
    int even = power(5,(N+1)/2);
    int odd = power(4,N/2);
    return even*1ll*odd%M;
}
```

Sort a stack

👉Recursive [TC:O(n^2) & SC:O(n)]

```
void insertElement(stack<int> &st, int val){
    if (st.empty() || st.top() <= val){
        st.push(val);
        return;
    }
    int temp = st.top();
    st.pop();
    insertElement(st, val);
    st.push(temp);
}
```

```
void sortStack(stack<int> &st){
    if (st.empty())return;

    int temp = st.top();
    st.pop();
    sortStack(st);

    insertElement(st, temp);
}
```

👉Iterative **[TC:O(n^2) & SC:O(n)]**

```
void sortStack(stack<int> &st){
    stack<int> tempSt;

    while (!st.empty()){
        int temp = st.top();
        st.pop();

        while (!tempSt.empty() && tempSt.top() > temp){
            st.push(tempSt.top());
            tempSt.pop();
        }
        tempSt.push(temp);
    }
    st = tempSt;
}
```

Reverse a Stack

👉Iterative **[TC:O(n^2) & SC:O(n)]**

```
void Reverse(stack<int> &st){
    stack<int> tempSt;
    while (!st.empty()){
        tempSt.push(st.top());
        st.pop();
    }
    st = tempSt;
}
```

👉Recursive **[TC:O(n^2) & SC:O(n)]**

```
void insertElement(stack<int> &st, int val){
    if (st.empty()){
        st.push(val);
        return;
    }
    int temp = st.top();
    st.pop();
    insertElement(st, val);
    st.push(temp);
}

void Reverse(stack<int> &st){
    if (st.empty())return;

    int temp = st.top();
    st.pop();
    Reverse(st);

    insertElement(st, temp);
}
```

Step 7.2: Subsequences Pattern

Generate all binary strings **[TC:O(2^n) & SC:O(n)]**

```
void solve(int n, string s){
    if(n==0){
        cout<<s<<" ";
        return;
    }
    solve(n-1, s+'0');
    if(s.empty() || s.back()=='0') solve(n-1, s+'1');
}

void generateBinaryStrings(int num){
    solve(num,"");
}
```

Generate Parentheses [TC:O(2^n) & SC:O(n)]

```
void solve(int open,int n,string s, vector<string> &ans){
    if(n==0){
        while(open--) s+=')';
        ans.push_back(s);
        return;
    }
    solve(open+1,n-1,s+'(',ans);
    if(open) solve(open-1,n,s+')',ans);
}

vector<string> AllParenthesis(int n) {
    vector<string> ans;
    solve(0,n,"",ans);
    return ans;
}
```

Power Set [TC:O(n*2^n) & SC:O(n)]

For Iterative Click here

```
void solve(int i, string curr, string s, vector<string> &ans){
    if (i == s.size()){
        if (!curr.empty()) ans.push_back(curr);
        return;
    }
    solve(i + 1, curr, s, ans);
    solve(i + 1, curr + s[i], s, ans);
}

vector<string> AllPossibleStrings(string s){
    vector<string> ans;
    solve(0, "", s, ans);
    sort(ans.begin(), ans.end());
    return ans;
}
```

Number of distinct subsequences

👉Method1 **[TC:O(n*2^n) & SC:O(2^n)]**

```
void solve(int i, string curr, string s, set<string> &ans){
    if (i == s.size()){
        ans.insert(curr);
        return;
    }
    solve(i + 1, curr, s, ans);
    solve(i + 1, curr + s[i], s, ans);
}

int distinctSubsequences(string s){
    set<string> st;
    solve(0, "", s, st);
    return st.size();
}
```

👉Method2 **[TC:O(n*2^n) & SC:O(n)]**

For Tabulation click here

```
int solve(int i, string &s, vector<int> &count){
    if (i < 0) return 1;
    int curr = solve(i - 1, s, count);
    int ans =  2*curr - count[s[i]-'a'];
    count[s[i]-'a'] = curr;
    return (ans%M+M)%M;
}

int distinctSubsequences(string s){
    vector<int> count(26,0);
    return solve(s.size()-1, s, count);
}
```

Step 7.3: Trying out all Combos / Hard

//

Step 8: Bit Manipulation [Concepts & Problems]

Step 8.1: Learn Bit Manipulation

Check whether K-th bit is set or not [TC:O(1) & SC:O(1)]

```
bool checkKthBit(int n, int k){
    return (1<<k)&n ;
    return (n>>k)&1 ;
}
```

Odd or Even [TC:O(1) & SC:O(1)]

```
string oddEven(int N){
    if (N&1) return "odd" ;
    else return "even" ;
}
```

Power of 2 [TC:O(1) & SC:O(1)]

```
bool isPowerofTwo(long long n){
    if(n==0) return false;
    return (n&(n-1))==0 ;
}
```

Count total set bits [TC:O(log(n)) & SC:O(1)]

```
int countSetBits(int n){
    n++;
    int ans = 0, curr = 1;
    for(int i=0;i<30;i++){
        curr*=2;
        ans+=n/curr*(curr/2);
        int left = n%curr;
        if(left>curr/2) ans+=left-curr/2;
    }
    return ans;
}
```

Set the rightmost unset bit

👉Method 1 **[TC:O(log(n)) & SC:O(1)]**

```
int setBit(int N){
    int val=1;
    while(val<N){
        if((N&val)==0) return N|val;
        val<<=1;
    }
    return N;
}
```

👉Method 2 **[TC:O(1) & SC:O(1)]**

```
int getRightmostSetBit(int n){
    return n & ~(n-1) ;
    return n & -n;
}

int setBit(int N){
    if((N & (N+1)) == 0) return N;                // if All Bit Set
    int val = getRightmostSetBit(~N);
    return val | N ;
}
```

Swap two numbers **[TC:O(1) & SC:O(1)]**

```
pair<int, int> get(int a, int b){
    // a = a^b^(b=a);
    // a = a + b - (b=a);
    // a = a * 1ll * b / (b=a);

    a ^= b;
    b ^= a;
    a ^= b;
    return {a,b};
}
```

Division without using multiply, division and mod operator

👉Method 1 **[TC:O(log(n)) & SC:O(1)]**

```
long long divide(long long dividend, long long divisor){
    int q_sign = (dividend < 0) ^ (divisor < 0) ? -1 : 1;
    int r_sign = (dividend < 0) ? -1 : 1;
    dividend = abs(dividend);
    divisor = abs(divisor);

    long long q = 0;
    for (int i = 31; i >= 0; i--){
        if (divisor << i <= dividend){
            dividend -= (divisor << i);
            q |= (1 << i);
        }
    }
    long long rem = dividend * r_sign;
    return q * q_sign;
}
```

👉Method 2 **[TC:O(1) & SC:O(1)]**

```
//  x = e ^ (ln(x))
//  a/b = e ^ (ln(a/b))
//  a/b = e ^ (ln(a) - ln(b))

long long divide(long long dividend, long long divisor){
    if (dividend == 0) return 0;

    int sign = (dividend < 0) ^ (divisor < 0) ? -1 : 1;
    dividend = abs(dividend);
    divisor = abs(divisor);

    long long int q = exp(log(dividend) - log(divisor)) + 0.01;

    return q * sign;
}
```

Step 8.2: Interview Problems

Bit Difference

👉Method 1 **[TC:O(log(n)) & SC:O(1)]**

```
int countBitsFlip(int a, int b){
    a^=b;
    int ct=0;
    for(int i=0;i<30;i++) {
        if (a & (1<<i)) ct++;
    }
    return ct;
}
```

👉Method 2 **[TC:O(log(n)) & SC:O(1)]**

```
int countBitsFlip(int a, int b){
    a^=b;
    int ct=0;
    while(a) {
        a &= (a-1);  //unset the rightmost set bit
        ct++;
    }
    return ct;
}
```

👉Method 3 **[TC:O(1) & SC:O(1)]**

```
int countBitsFlip(int a, int b){
    return __builtin_popcount(a^b);
}
```

Find the number that appears odd number of times [TC:O(n) & SC:O(1)]

```
int getOddOccurrence(int arr[], int n) {
    int ans = 0;
    for(int i=0;i<n;i++) ans^=arr[i];
    return ans;
}
```

Power Set [TC:O(n*2^n) & SC:O(1)]

For Recursive Click here

```
vector<string> AllPossibleStrings(string s){
    int n=s.size();
    vector<string> ans;

    for(int i=1;i<(1<<n);i++){
        string curr="";
        for(int j=0;j<n;j++){
            if(i&(1<<j)) curr+=s[j];
        }
        ans.push_back(curr);
    }
    sort(ans.begin(),ans.end());
    return ans;
}
```

Find XOR of numbers from L to R

👉Method 1 [TC:O(log(n)) & SC:O(1)]

```
int XorUptoN(int N){
    N++;
    int ans = 0, curr = 1;

    while (curr<N) {
        int pair = N/curr;
        int count = N/curr/2*curr;
        if(pair%2) count+=N%curr;

        if(count%2) ans |= curr ;
        curr <<= 1;
    }
    return ans;
}

int findXOR(int l, int r) {
    return XorUptoN(r) ^ XorUptoN(l-1);
}
```

👉Method 2 **[TC:O(1) & SC:O(1)]**

```
int XorUptoN(int N){
    if(N%4==0) return N;
    if(N%4==1) return 1;
    if(N%4==2) return N+1;
    if(N%4==3) return 0;
}
```

Two numbers with odd occurrences [TC:O(n) & SC:O(1)]

```
vector<long long int> twoOddNum(long long int Arr[], long long int N){
    long long int xr=0,a=0,b=0;
    for(int i=0;i<N;i++) xr^=Arr[i];
    int val = xr&-xr;

    for(int i=0;i<N;i++){
        if(val&Arr[i]) a^=Arr[i];
        else b^=Arr[i];
    }

    if(a>b) return {a,b};
    else return {b,a};
}
```

Step 8.3: Advanced Maths

Prime Factors [TC:O(n) & SC:O(n)]

```
vector<int>AllPrimeFactors(int N) {
    vector<int> ans;
    for(int i=2;i<=N;i++){
        if(N%i==0) ans.push_back(i);
        while(N%i==0) N/=i;
    }
    return ans;
}
```

All divisors of a Number [TC:O(sqrt(n)) & SC:O(sqrt(n))]

```
void print_divisors(int n) {
    vector<int> res;
    for(int i=1;i*i<=n;i++){
        if(n%i==0){
            cout<<i<<" ";
            res.push_back(n/i);
        }
    }

    if(res.back()*res.back()==n) res.pop_back();
    for(int i=res.size()-1;i>=0;i--){
        cout<<res[i]<<" ";
    }
}
```

Sieve of Eratosthenes [TC:O(n*log(log(n))) & SC:O(n)]

```
vector<int> sieveOfEratosthenes(int N){
    vector<int> prime(N+1,1),ans;

    for(int i=2;i<=N;i++){
        if(prime[i]){
            ans.push_back(i);

            for(int j=i*i;j<=N;j+=i){
                prime[j]=0;
            }
        }
    }

    return ans;
}
```

Prime Factorization using Sieve [TC:O(n*log(log(n))) & SC:O(n)]

```
int spf[int(2e5)+1];

void sieve() {
    for(int i=0;i<=2e5;i++) spf[i]=i;

    for(int i=2;i*i<=2e5;i++){
        if(spf[i]==i){
            for(int j=i*i;j<=2e5;j+=i){
                if(spf[j]==j) spf[j]=i;
            }
        }
    }
}

vector<int> findPrimeFactors(int N) {
    vector<int> ans;

    while(N!=1){
        ans.push_back(spf[N]);
        N/=spf[N];
    }
    return ans;
}
```

Power Of Numbers [TC:O(log(R)) & SC:O(1)]

For Recursive Click here

```
long long power(int N,int R){
   int M = 1e9+7;
   long long res = 1,temp=N;

    while(R){
        if(R&1) res = res*N%M;
        N = N*1ll*N%M;
        R>>=1;
    }
    return res;
}
```

Step 9: Stack and Queues

Step 9.1: Learning

//

Step 9.2: Prefix, Infix, PostFix Conversion Problems

//

Step 9.3: Monotonic Stack/Queue Problems [VVV. Imp]

Next Greater Element (NGE) [TC:O(n) & SC:O(n)]

```
vector<long long> nextLargerElement(vector<long long> arr, int n){
    vector<long long> ans(n,-1);
    stack<int> s;

    for (int i = 0; i < n; i++) {
        while (!s.empty() && arr[s.top()] < arr[i]) {
            ans[s.top()]=arr[i];
            s.pop();
        }
        s.push(i);
    }

    return ans;
}
```

Next Greater Element 2 (Circular Array) [TC:O(2*n) & SC:O(n)]

```
vector<int> nextGreaterElement(int N, vector<int>& arr) {
    vector<int> ans(N,-1);
    stack<int> st;

    for(int i=0;i<2*N;i++){
        while(!st.empty() && arr[st.top()]<arr[i%N]){
            ans[st.top()] = arr[i%N];
            st.pop();
        }
        st.push(i%N);
    }

    return ans;
}
```

Next Smaller Element (NSE) [TC:O(n) & SC:O(n)]

```
vector<long long> nextSmallerElement(vector<long long> arr, int n){
    vector<long long> ans(n,-1);
    stack<int> s;

    for (int i = 0; i < n; i++) {
        while (!s.empty() && arr[s.top()] > arr[i]) {
            ans[s.top()]=arr[i];
            s.pop();
        }
        s.push(i);
    }

    return ans;
}
```

Number of NGEs to the right[TC:O(q*n) & SC:O(1)]

```
vector<int> count_NGE(int n, vector<int> &arr, int queries, vector<int> &indices){
    vector<int> ans(queries);

    for(int i=0;i<queries;i++){
        int curr = arr[indices[i]];
        int ct=0;

        for(int j=indices[i]+1;j<n;j++){
            if( curr < arr[j]) ct++;
        }

        ans[i] = ct;
    }

    return ans;
}
```

Trapping Rain Water [TC:O(n) & SC:O(n)]

👉Method1 [TC:O(3*n) & SC:O(2*n)]

```
long long trappingWater(int arr[], int n){
    vector<int> Lmax(n),Rmax(n);
    Lmax[0]=arr[0];
    Rmax[n-1]=arr[n-1];

    for(int i=1;i<n;i++) Lmax[i]=max(Lmax[i-1],arr[i]);
    for(int i=n-2;i>=0;i--) Rmax[i]=max(Rmax[i+1],arr[i]);

    long long ans=0;
    for(int i=0;i<n;i++){
        int H=min(Lmax[i],Rmax[i]);
        if(H>arr[i]) ans+=H-arr[i];
    }
    return ans;
}
```

👉Method2 [TC:O(n) & SC:O(n)]

```
long long trappingWater(int arr[], int n){
    stack<int> s;
    long long ans=0;

    for(int i=0;i<n;i++){
        while(!s.empty() && arr[s.top()]<=arr[i] ){
            int curr=s.top();
            s.pop();
            if(s.empty()) break;

            int len=i-s.top()-1;
            int h = min(arr[s.top()],arr[i])-arr[curr];

            ans+=(h*1ll*len);
        }
        s.push(i);
    }
    return ans;
}
```

👉Method3 **[TC:O(n) & SC:O(1)]**

```
long long trappingWater(int arr[], int n){
    int Lmax=0,Rmax=0;
    int left=0,right=n-1;
    long long ans=0;

    while(left<=right){
        if(arr[left]<arr[right]){
            if(arr[left]>Lmax) Lmax=arr[left];
            else ans+=(Lmax-arr[left]);
            left++;
        }
        else{
            if(arr[right]>Rmax) Rmax=arr[right];
            else ans+=(Rmax-arr[right]);
            right--;
        }
    }
    return ans;
}
```

Sum of subarray minimum **[TC:O(n) & SC:O(n)]**

```
int sumSubarrayMins(int N, vector<int> &arr) {
    stack<int> st;
    st.push(-1);
    ll ans=0;

    for(int i=0;i<=N;i++){
        while(st.top()!=-1 && (i==N || arr[st.top()]>=arr[i])){
            int curr=st.top();
            st.pop();
            ll left = curr-st.top();
            ll right = i-curr;
            ans= (ans+left*right*arr[curr])%M;
        }
        st.push(i);
    }
    return int(ans);
}
```

Asteroid Collision [TC:O(n) & SC:O(n)]

```
vector<int> asteroidCollision(int N, vector<int> &asteroids) {
    stack<int> st;

    for(int i=0;i<N;i++){
        if(asteroids[i]>0 || st.empty() || st.top()<0) st.push(asteroids[i]);
        else{
            while(!st.empty()){
                if(st.top()>abs(asteroids[i])) break;
                if(st.top()==abs(asteroids[i])) {st.pop();break;}

                st.pop();
                if(st.empty() || st.top()<0)
{st.push(asteroids[i]);break;}
            }
        }
    }

    vector<int> ans;
    while(st.size()){
        ans.push_back(st.top());
        st.pop();
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

Sum of subarray ranges [TC:O(2*n) & SC:O(n)]

```
long long subarrayRanges(int N, vector<int> &arr) {
    stack<int> st;
    st.push(-1);
    ll ans=0;

    //Max
    for(int i=0;i<=N;i++){
        while(st.top()!=-1 && (i==N || arr[st.top()]<=arr[i])){
            int curr=st.top();
            st.pop();
            ll left = curr-st.top();
            ll right = i-curr;
            ans+=(left*right*arr[curr]);
        }
        st.push(i);
    }
    st.pop();                                   //pop N

    //Min
    for(int i=0;i<=N;i++){
        while(st.top()!=-1 && (i==N || arr[st.top()]>=arr[i])){
            int curr=st.top();
            st.pop();
            ll left = curr-st.top();
            ll right = i-curr;
            ans-=(left*right*arr[curr]);
        }
        st.push(i);
    }
    return ans;
}
```

Remove K Digits [TC:O(n) & SC:O(1)]

```
string removeKdigits(string S, int k) {
    int n=S.size();
    string ans;

    for(int i=0;i<=n;i++){
        while(k && ans.size() && (i==n || ans.back()>S[i])) {
            ans.pop_back();
            k--;
        }
        if(i!=n && (ans.size() || S[i]!='0')) ans+=S[i];
    }

    return ans.size()?ans:"0";
}
```

Maximum Rectangular Area in a Histogram

👉Method1 [TC:O(n) & SC:O(n)]

```
long long getMaxArea(long long arr[], int n){
    stack<int> st;
    st.push(-1);
    ll ans=0;

    for(int i=0;i<=n;i++){
        while(st.top()!=-1 && (i==n || arr[st.top()]>=arr[i])){
            ll h=arr[st.top()];
            st.pop();

            ll len = i-st.top()-1;
            ans=max(ans,h*len);
        }
        st.push(i);
    }
    return ans;
}
```

👉Method2 **[TC:O(n) & SC:O(3*n)]**

```
long long getMaxArea(long long arr[], int n){
    vector<int> leftsmall(n,-1),rightsmall(n,n);

    stack<int> s;
    for(int i=0;i<n;i++){
        while(!s.empty() && arr[s.top()]>=arr[i]) s.pop();
        if(s.size()) leftsmall[i]=s.top();
        s.push(i);
    }

    while(s.size()) s.pop();
    for(int i=n-1;i>=0;i--){
        while(!s.empty() && arr[s.top()]>=arr[i]) s.pop();
        if(s.size()) rightsmall[i]=s.top();
        s.push(i);
    }

    long long ans=0;
    for(int i=0;i<n;i++){
        long long width = rightsmall[i]-leftsmall[i]-1 ;
        long long curr=arr[i]*width;
        ans=max(ans,curr);
    }
    return ans;
}
```

Max rectangle[TC:O(n*m) & SC:O(m)]

```
int maxArea(int M[MAX][MAX], int n, int m) {
    int ans = getMaxArea(M[0],m);

    for(int i=1;i<n;i++){
        for(int j=0;j<m;j++) if(M[i][j]) M[i][j]+=M[i-1][j];
        ans=max(ans,getMaxArea(M[i],m));
    }
    return ans;
}
```

Step 9.4: Implementation Problems

Sliding Window Maximum

👉Method1 **[TC:O(n) & SC:O(k)]**

```
vector<int> max_of_subarrays(vector<int> arr, int n, int k) {
    deque<int> q;
    vector<int> ans;

    for(int i=0;i<k-1;i++){
        while(!q.empty() && arr[q.back()]<=arr[i]) q.pop_back();
        q.push_back(i);
    }

    for(int i=k-1;i<n;i++){
        while(!q.empty() && q.front()<=i-k) q.pop_front();
        while(!q.empty() && arr[q.back()]<=arr[i]) q.pop_back();
        q.push_back(i);
        ans.push_back(arr[q.front()]);
    }
    return ans;
}
```

👉Method2 **[TC:O(n) & SC:O(K)]**

```
vector<int> max_of_subarrays(vector<int> arr, int n, int k) {
    priority_queue<pair<int,int>> q;
    vector<int> ans;

    for(int i=0;i<k-1;i++) q.push({arr[i],i});

    for(int i=k-1;i<n;i++){
        while(!q.empty() && q.top().second<=i-k) q.pop();
        q.push({arr[i],i});
        ans.push_back(q.top().first);
    }
    return ans;
}
```

Stock span problem (PGE) [TC:O(n) & SC:O(1)]

```
vector <int> calculateSpan(int price[], int n){
    stack<int> st;
    st.push(-1);
    vector<int> ans(n);

    for(int i=0;i<n;i++){
        while(st.top()!=-1 && price[st.top()]<=price[i]) st.pop();
        ans[i]=i-st.top();
        st.push(i);
    }
    return ans;
}
```

The Celebrity Problem[TC:O(2*n) & SC:O(1)]

```
int celebrity(vector<vector<int> >& M, int n){
    int c = 0;                                  // let celebrity be 0
    for(int i=1;i<n;i++) if(M[c][i]) c=i;       // possible celebrity be i

    //Now check both condition for c
    for(int i=0;i<n;i++){
        if(i!=c && (M[c][i]==1 || M[i][c]==0)) return -1;
    }
    return c;
}
```

Maximum of minimum for every window size[TC:O(2*n) & SC:O(n)]

```
vector <int> maxOfMin(int arr[], int n){
    stack<int> s;
    s.push(-1);
    vector<int> ans(n,0);

    for(int i=0;i<=n;i++){
        while(s.top()!=-1 && (i==n || arr[s.top()]>=arr[i])){
            int j=s.top();
            s.pop();

            int len = i-s.top()-1;
            ans[len-1]=max(ans[len-1],arr[j]);
        }
        s.push(i);
    }

    for(int i=n-2;i>=0;i--) ans[i]=max(ans[i],ans[i+1]);
    return ans;
}
```

LRU Cache

Node Class for doubly linked list

```
class node{
public:
    int key;
    int val;
    node* prev;
    node* next;

    node(int key,int val){
        this->key = key;
        this->val = val;
        prev = next = NULL;
    }
};
```

```
class LRUCache{
    int cap;
    node* head;
    node* tail;
    unordered_map<int,node*> mp;
public:
    LRUCache(int cap){
        this->cap = cap;
        head = new node(-1,-1);
        tail = new node(-1,-1);
        head->next=tail;
        tail->prev=head;
    }

    void remove(node* root){
        root->prev->next = root->next;
        root->next->prev = root->prev;
    }
```

```
    void add(node* root){
        root->next = head->next;
        head->next = root;
        root->prev = head;
        root->next->prev = root;
    }

    int GET(int key){
        if(mp.find(key)==mp.end()) return -1;
        remove(mp[key]);
        add(mp[key]);
        return mp[key]->val;                                    //key to node
    }

    void SET(int key, int value){
        if(mp.find(key)!=mp.end()) {
            mp[key]->val=value;
            remove(mp[key]);
        }
        else {
            if(mp.size()==cap) {
                node* todelete = tail->prev;
                mp.erase(todelete->key);                        //node to key
                remove(todelete);
                delete todelete;
            }
            mp[key] = new node(key,value);
        }
        add(mp[key]);
    }
};
```

LFU Cache

```
class node{
public:
    int key;
    int val;
    int count;
    node* prev;
    node* next;

    node(int key,int val){
        this->key = key;
        this->val = val;
        count=1;
        prev = next = NULL;
    }
};
```

```
class List{
public:
    node* head;
    node* tail;
    int size;

    List(){
        size = 0;
        head = new node(-1,-1);
        tail = new node(-1,-1);
        head->next=tail;
        tail->prev=head;
    }

    void remove(node* root){
        size--;
        root->prev->next = root->next;
        root->next->prev = root->prev;
    }
```

```
    void add(node* root){
        root->next = head->next;
        head->next = root;
        root->prev = head;
        root->next->prev = root;
        size++;
    }
};
```

```
class LFUCache {
    int cap;
    int minFreq;
    unordered_map<int,node*> mpNode;
    unordered_map<int,List*> mpList;
public:
    LFUCache(int cap){
        this->cap = cap;
        minFreq=0;
    }

    void updateFreq(node* root){
        mpList[root->count]->remove(root);
        if(minFreq==root->count && mpList[root->count]->size==0)
minFreq++;
        root->count++;
        if(!mpList[root->count]) mpList[root->count] = new List();
        mpList[root->count]->add(root);
    }

    int get(int key) {
        if(mpNode.find(key)==mpNode.end()) return -1;
        updateFreq(mpNode[key]);
        return mpNode[key]->val;                    //key to node
```

```
    }
    void put(int key, int value) {
        if(cap==0) return;
        if(mpNode.find(key)!=mpNode.end()) {
            mpNode[key]->val=value;
            updateFreq(mpNode[key]);
        }
        else {
            if(mpNode.size()==cap) {
                node* todelete = mpList[minFreq]->tail->prev;
                mpList[minFreq]->remove(todelete);
                mpNode.erase(todelete->key);         //node to key
                delete todelete;
            }

            mpNode[key] = new node(key,value);
            if(!mpList[1]) mpList[1] = new List();
            mpList[1]->add(mpNode[key]);
            minFreq=1;
        }
    }
};
```

Step 10: Sliding Window & Two Pointer Combined Problems

Step 10.1: Medium Problems

Length of the longest substring

👉Method 1 **[TC:O(2*n) & SC:O(n)]**

```
int longestUniqueSubsttr(string S){
    vector<int> arr(26,0);
    int i=0,ans=0;

    for(int j=0;j<nums.size();j++){
        arr[S[j]-'a']++;

        while(arr[S[j]-'a']>1){
            arr[S[i]-'a']--;
            i++;
        }
        ans = max(ans,j-i+1);
    }
    return ans;
}
```

👉Method 2 **[TC:O(n) & SC:O(n)]**

```
int longestUniqueSubsttr(string S){
    vector<int> arr(26,-1);
    int i=0,ans=0;

    for(int j=0;j<nums.size();j++){
        if(arr[S[j]-'a']!=-1) i=max(i,arr[S[j]-'a']+1);
        arr[S[j]-'a'] = j;
        ans = max(ans,j-i+1);
    }
    return ans;
}
```

Max Consecutive Ones III [TC:O(n) & SC:O(1)]

```
int longestOnes(vector<int>& nums, int k) {
    int i=0,zero_ct=0,ans=0;

    for(int j=0;j<nums.size();j++){
        zero_ct+=(nums[j]==0);
        while(zero_ct>k){
            zero_ct-=(nums[i]==0);
            i++;
        }
        ans = max(ans,j-i+1);
    }
    return ans;
}
```

Fruit Into Baskets [TC:O(n) & SC:O(1)]

👉Method 1

```
int totalFruit(vector<int>& fruits) {
    pair<int,int> one={-1,-1},two={-1,-1};
    int i=0,ans=0;

    for(int j=0;j<fruits.size();j++){
        if(fruits[j]==one.first) one.second++;
        else if(fruits[j]==two.first) two.second++;
        else if(one.first==-1) one = {fruits[j],1};
        else if(two.first==-1) two = {fruits[j],1};
        else {
            if(two.first == fruits[j-1]) swap(one,two);
            while(two.second){
                if(fruits[i]==one.first) one.second--;
                else if(fruits[i]==two.first) two.second--;
                i++;
            }
            two = {fruits[j],1};
        }
        ans=max(ans,j-i+1);
    }
    return ans;
}
```

👉Method 2

```
int totalFruit(vector<int>& fruits) {
    unordered_map<int,int> mp;
    int i=0,ans=0;

    for(int j=0;j<fruits.size();j++){
        mp[fruits[j]]++;
        while(mp.size()>2){
            mp[fruits[i]]--;
            if(mp[fruits[i]]==0) mp.erase(fruits[i]);
            i++;
        }
        ans=max(ans,j-i+1);
    }
    return ans;
}
```

Longest Repeating Character Replacement

//

Step 10.2: Hard Problems

//

Step 11: Heaps [Learning, Medium, Hard Problems]

Step 11.1: Learning

//

Step 11.2: Medium Problems

Kth Largest Element [TC:O(n*log(k)) & SC:O(K)]

```
int kthLargest(int arr[], int n, int k) {
    priority_queue<int,vector<int>,greater<int>> pq;

    for(int i=0;i<n;i++){
        pq.push(arr[i]);
        if(pq.size()>k) pq.pop();
    }
    return pq.top();
}
```

See Partition Function Implementation [TC:O(n) & SC:O(1)]

```
int partition (int arr[], int low, int high){
    int pivot = arr[low] ;
    int l = low + 1 ;
    int r = high;

    while (l <= r) {
        if (arr[l] < pivot && arr[r] > pivot) {
            swap(arr[l], arr[r]);
            l++ ;
            r-- ;
        }
        if (arr[l] >= pivot) l++;
        if (arr[r] <= pivot) r--;
    }
    swap(arr[low], arr[r]);
    return r;
}

int kthLargest(int arr[], int n, int k){
    int left = 0, right = n - 1;
    while (1) {
        int idx = partition(arr, left, right);
        if (idx == k - 1) return arr[idx];

        if (idx < k - 1) left = idx + 1;
        else right = idx - 1;
    }
}
```

Kth Smallest Element [TC:O(n*log(k)) & SC:O(K)]

```
int kthSmallest(int arr[], int n, int k) {
    priority_queue<int> pq;

    for(int i=0;i<n;i++){
        pq.push(arr[i]);
        if(pq.size()>k) pq.pop();
    }
    return pq.top();
}
```

[TC:O(n) & SC:O(log(n))] SC : O(1) using a while loop as above.

```
int partition (int arr[], int low, int high){
    int pivot=arr[high];
    int curr=low,next=high-1;

    while(curr<=next){
        if(arr[curr]>pivot) swap(arr[curr],arr[next--]);
        else curr++;
    }
    swap(arr[curr],arr[high]);
    return curr;
}

int kthSmallest(int arr[], int l, int r, int k) {
    int idx = partition(arr,l,r);
    if(idx==k-1) return arr[idx];
    if(idx>k-1) return kthSmallest(arr,l,idx-1,k);
    return kthSmallest(arr,idx+1,r,k);
}
```

Sort K sorted array [TC:O(n*log(k)) & SC:O(k)]

```
#define pi pair<int,int>
vector <int> nearlySorted(int arr[], int num, int k){
    priority_queue<pi,vector<pi>,greater<pi>> pq;

    for(int i=0;i<k;i++) pq.push({arr[i],i});

    vector<int> ans;
    for(int i=0;i<num;i++){
        if(i+k<num) pq.push({arr[i+k],i+k});
        ans.push_back(pq.top().first);
        pq.pop();
    }
    return ans;
}
```

Merge k Sorted Arrays [TC:O(k*k*log(k)) & SC:O(k)]

```
#define pi pair<int,pair<int,int>>
vector<int> mergeKArrays(vector<vector<int>> arr, int K){
    priority_queue<pi,vector<pi>,greater<pi>> pq;
    for(int i=0;i<K;i++) pq.push({arr[i][0],{i,0}});

    vector<int> ans;
    while(!pq.empty()){
        ans.push_back(pq.top().first);
        int x=pq.top().second.first;
        int y=pq.top().second.second;
        pq.pop();
        if(y+1<K) pq.push({arr[x][y+1],{x,y+1}});
    }
    return ans;
}
```

Replace elements by its rank in the array [TC:O(n*log(n)) & SC:O(n)]

```
vector<int> replaceWithRank(vector<int> &arr, int N){
    vector<pair<int,int>> v;
    for(int i=0;i<N;i++) v.push_back({arr[i],i});

    sort(v.begin(),v.end());
    int idx=1;
    vector<int> ans(N);

    for(int i=0;i<N;i++){
        if(i && v[i-1].first!=v[i].first) idx++;
        ans[v[i].second] = idx;
    }

    return ans;
}
```

**Task Scheduler [TC:O(n) & SC:O(26)]

```
int leastInterval(int N, int K, vector<char> &tasks) {
    vector<int> arr(26,0);
    int maxFreq = 0;

    for(int i = 0; i < N; i++){
        arr[tasks[i]-'A']++;
        maxFreq = max(maxFreq, arr[tasks[i]-'A']);
    }

    int ans = (maxFreq - 1) * (K + 1);         //1 less as it will add
below

    for(int i=0;i<26;i++)  if(arr[i] == maxFreq) ans++;

    return max((int)tasks.size(), ans);
}
```

Hands of Straights[TC:O(n*log(n)) & SC:O(n)]

```cpp
bool isStraightHand(int n, int k, vector<int> &hand) {
    map<int,int> mp;
    for(int i=0;i<n;i++) mp[hand[i]]++;

    while(!mp.empty()){
        int st=(*mp.begin()).first;

        for(int i=st;i<st+k;i++){
            if(mp.find(i)==mp.end()) return 0;
            if(mp[i]==1) mp.erase(i);
            else mp[i]--;
        }
    }
    return 1;
}
```

Step 11.3: Hard Problems

//

Step 13: Binary Trees [Traversals, Medium and Hard Problems]

Step 13.1: Traversals

Introduction to Trees [TC:O(1) & SC:O(1)]

```
int countNodes(int i) {
    return 1<<(i-1) ;
}
```

Binary Tree Representation [TC:O(n) & SC:O(n)]

```
void create_tree(node* root0, vector<int> &vec){
    root0->left = newNode(vec[1]);
    root0->right = newNode(vec[2]);
    root0->left->left = newNode(vec[3]);
    root0->left->right = newNode(vec[4]);
    root0->right->left = newNode(vec[5]);
    root0->right->right = newNode(vec[6]);
}
```

Preorder Traversal (Recursive) [TC:O(n) & SC:O(n)]

```
void preOrder(Node* root,vector<int> &v){
    if(!root) return;
    v.push_back(root->data);
    preOrder(root->left,v);
    preOrder(root->right,v);
}
```

Inorder Traversal (Recursive) [TC:O(n) & SC:O(n)]

```
void inorder(Node* root,vector<int> &v){
    if(!root) return;
    inorder(root->left,v);
    v.push_back(root->data);
    inorder(root->right,v);
}
```

Postorder Traversal (Recursive) [TC:O(n) & SC:O(n)]

```
void postorder(Node* root,vector<int> &v){
    if(!root) return;
    postorder(root->left,v);
    postorder(root->right,v);
    v.push_back(root->data);
}
```

Level order traversal

▶️GFG [TC:O(n) & SC:O(n)]

```
vector<int> levelOrder(Node *root){
    vector<int> ans;
    queue<Node *> q;
    q.push(root);

    while (q.size()){
        auto curr = q.front();
        q.pop();
        ans.push_back(curr->data);
        if (curr->left) q.push(curr->left);
        if (curr->right) q.push(curr->right);
    }
    return ans;
}
```

▶️LeetCode [TC:O(n) & SC:O(n)]

👉Method 1

```
vector<vector<int>> levelOrder(TreeNode* root) {
    queue<TreeNode*> q;
    vector<vector<int>> v;
    if(!root) return v;
    q.push(root);

    while(q.size()){
        int n=q.size();
        vector<int> lvl;
```

```
        while(n--){
            TreeNode* curr=q.front();
            lvl.push_back(curr->val);
            q.pop();
            if(curr->left) q.push(curr->left);
            if(curr->right) q.push(curr->right);
        }
        v.push_back(lvl);
    }
    return v;
}
```

👉Method 2

```
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> v;
    if(!root) return v;

    queue<TreeNode*> q;
    vector<int> lvl;
    q.push(root);
    q.push(NULL);

    while(q.size()){
        TreeNode* curr=q.front();
        q.pop();

        if(curr){
            lvl.push_back(curr->val);
            if(curr->left) q.push(curr->left);
            if(curr->right) q.push(curr->right);
        }
        else {
            v.push_back(lvl);
            lvl.clear();
            if(q.size()) q.push(NULL);
        }
    }
    return v;
}
```

Level order traversal in spiral form [TC:O(n) & SC:O(n)]

👉 Method 1

```
vector<int> findSpiral(Node *root){
    vector<int> ans;
    stack<Node*> s_left,s_right;
    if(root) s_right.push(root);
    int left = 0;

    while(s_left.size() || s_right.size()){
        if(left){
            if(s_left.empty()) {
                left = 0;
                continue;
            }
            auto curr = s_left.top();
            s_left.pop();
            ans.push_back(curr->data);
            if(curr->left) s_right.push(curr->left);
            if(curr->right) s_right.push(curr->right);
        }
        else{
            if(s_right.empty()) {
                left = 1;
                continue;
            }
            auto curr = s_right.top();
            s_right.pop();
            ans.push_back(curr->data);
            if(curr->right) s_left.push(curr->right);
            if(curr->left) s_left.push(curr->left);
        }
    }
    return ans;
}
```

👉 Method 2

```
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>>ans;
    if(!root) return ans;
    stack<TreeNode*>s1,s2;
    s1.push(root);

    while(s1.size() || s2.size()){
        if(s1.size()){
            int n=s1.size();
            vector<int> v;
            while(n--){
                TreeNode* curr=s1.top();
                s1.pop();
                v.push_back(curr->val);
                if(curr->left) s2.push(curr->left);
                if(curr->right) s2.push(curr->right);
            }
            ans.push_back(v);
        }
        else{
            int n=s2.size();
            vector<int> v;
            while(n--){
                TreeNode* curr=s2.top();
                s2.pop();
                v.push_back(curr->val);
                if(curr->right) s1.push(curr->right);
                if(curr->left) s1.push(curr->left);
            }
            ans.push_back(v);
        }
    }
    return ans;
}
```

👉 Method 3

Preorder Traversal (Iterative) [TC:O(n) & SC:O(n)]

👉 Method1

```
vector <int> preorder(Node* root){
    vector<int> ans;
    stack<Node*> st;
    st.push(root);

    while(st.size()){
        auto curr = st.top();
        st.pop();
        ans.push_back(curr->data);
        if(curr->right) st.push(curr->right);
        if(curr->left) st.push(curr->left);
    }
    return ans;
}
```

👉 Method2

```
vector <int> preorder(Node* root){
    vector<int> ans;
    stack<Node*> st;

    while(root || st.size()){
        if(root){
            ans.push_back(root->data);
            if(root->right) st.push(root->right);
            root = root->left;
        }
        else{
            root = st.top();
            st.pop();
        }
    }
    return ans;
}
```

Inorder Traversal (Iterative) [TC:O(n) & SC:O(n)]

```
vector<int> inOrder(Node* root) {
    vector<int> ans;
    stack<Node*> st;

    while(root || st.size()){
        if(root){
            st.push(root);
            root = root->left;
        }
        else{
            auto curr = st.top();
            st.pop();
            ans.push_back(curr->data);
            root = curr->right;
        }
    }
    return ans;
}
```

Postorder Traversal (Iterative) (2 Stack) [TC:O(n) & SC:O(n)]

👉 Method1

```
vector <int> postOrder(Node* root){
    vector<int> ans;
    stack<Node*> st;
    st.push(root);

    while(st.size()){
        auto curr = st.top();
        st.pop();
        ans.push_back(curr->data);
        if(curr->left) st.push(curr->left);
        if(curr->right) st.push(curr->right);
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

👉 Method 2

```
vector <int> postOrder(Node* root){
    vector<int> ans;
    stack<Node*> st;

    while(root || st.size()){
        if(root){
            ans.push_back(root->data);
            if(root->left) st.push(root->left);
            root = root->right;
        }
        else{
            root = st.top();
            st.pop();
        }
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

**Postorder Traversal (Iterative) (1 Stack) [TC:O(n) & SC:O(n)]

```
vector <int> postOrder(Node* root){
    vector<int> ans;
    stack<Node*> st;

    while(root || st.size()){
        if(root){
            st.push(root);
            root = root->left;
        }
        else{
            root = st.top()->right;

            if(!root){
                while(st.size() && st.top()->right==root){
                    root = st.top();
                    ans.push_back(root->data);
                    st.pop();
                }
                root = NULL;
            }
        }
    }
    return ans;
}
```

Preorder, Inorder, and Postorder Traversal in one Traversal

[TC:O(n) & SC:O(n)]

```
void allTraversal(Node *root){
    if (!root) return;
    vector <int> preorder, inorder, postorder;
    stack<pair<Node *, int>> st;
    st.push({root,1});

    while (st.size()){
        auto it = st.top();
        st.pop();

        if (it.second == 1){
            preorder.push_back(it.first->data);
            it.second++;
            st.push(it);
            if (it.first->left)
st.push({it.first->left,1});
        }
        else if (it.second == 2){
            inorder.push_back(it.first->data);
            it.second++;
            st.push(it);
            if(it.first->right) st.push({it.first->right,1});
        }
        else postorder.push_back(it.first->data);
    }
}
```

Step 13.2: Medium Problems

Height of Binary Tree [TC:O(n) & SC:O(h)]

```
int height(struct Node* node){
    if(!node) return 0;

    int lh=height(node->left);
    int rh=height(node->right);

    return 1+max(lh,rh);
}
```

Check for Balanced Tree [TC:O(n) & SC:O(h)]

```
bool check(Node* root,int &h){
    if(!root){
        h = 0;
        return true;
    }

    int lh,rh;
    if(!check(root->left,lh)) return false;
    if(!check(root->right,rh)) return false;
    if(abs(lh-rh)>1) return false;

    h = 1+max(lh,rh);
    return true;
}

bool isBalanced(Node *root){
    int h;
    return check(root,h);
}
```

Diameter of a Binary Tree [TC:O(n) & SC:O(h)]

👉 Pass height as reference and return ans

```
int solve(Node* root,int &h){
    if(!root) {
        h=0;
        return 0;
    }

    int lh,rh;
    int ld = solve(root->left,lh);
    int rd = solve(root->right,rh);

    h=1+max(lh,rh);
    return max(1+lh+rh,max(ld,rd));
}

int diameter(Node* root) {
    int h;
    return solve(root,h);
}
```

👉 Pass ans as reference and return height

```
int height(Node*root, int &dm){
    if(!root) return 0;

    int lh=height(root->left,dm);
    int rh=height(root->right,dm);
    dm=max(dm,1+lh+rh);

    return 1+max(lh,rh);
}

int diameter(Node* root) {
    int dm=0;
    height(root,dm);
    return dm;
}
```

Maximum path sum from any node [TC:O(n) & SC:O(h)]

```
int solve(Node*root, int &sum){
    if(!root) return 0;

    int l_sum=solve(root->left,sum);          // >=0
    int r_sum=solve(root->right,sum);         // >=0
    sum=max(sum,root->data+l_sum+r_sum);

    return max(0,root->data+max(l_sum,r_sum));
}

int findMaxSum(Node* root){
    int sum=INT_MIN;
    solve(root,sum);
    return sum;
}
```

Determine if Two Trees are Identical [TC:O(n) & SC:O(h)]

```
bool isIdentical(Node *r1, Node *r2){
    if(!r1 && !r2) return true;
    if(!r1 || !r2) return false;
    if(r1->data != r2->data) return false;
    if(!isIdentical(r1->left,r2->left)) return false;
    if(!isIdentical(r1->right,r2->right)) return false;
    return true;
}
```

ZigZag Tree Traversal [TC:O(n) & SC:O(n)]

Same as spiral level order traversal (leetcode)

```
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>>ans;
    if(!root) return ans;

    queue<TreeNode*> q;
    q.push(root);
    int left = 1;

    while(q.size()){
        int n = q.size();
        vector<int> temp(n);

        for(int i=0; i<n; i++){
            TreeNode* node = q.front();
            q.pop();

            int index = left ? i : n-i-1;
            temp[index] = node->val;

            if(node->left) q.push(node->left);
            if(node->right) q.push(node->right);
        }

        left = !left;
        ans.push_back(temp);
    }
    return ans;
}
```

Boundary Traversal of binary tree [TC:O(n) & SC:O(n)]

```
bool isLeaf(Node* root) {
    return !root -> left && !root -> right;
}

void left_boundary(Node* root,vector<int> &v){
    while(root){
        if(!isLeaf(root)) v.push_back(root->data);
        if(root->left) root = root->left;
        else root = root->right;
    }
}

void add_leaf(Node* root,vector<int> &v){
    if(isLeaf(root)) {
        v.push_back(root->data);
        return;
    }
    if(root->left) add_leaf(root->left,v);
    if(root->right) add_leaf(root->right,v);
}

void right_boundary(Node* root,vector<int> &v){
    vector<int> temp;
    while(root){
        if(!isLeaf(root)) temp.push_back(root->data);
        if(root->right) root = root->right;
        else root = root->left;
    }
    v.insert(v.end(),temp.rbegin(),temp.rend());
}

vector <int> boundary(Node *root){
    vector<int> ans;
    ans.push_back(root->data);
    if(isLeaf(root)) return ans;

    left_boundary(root->left,ans);
    add_leaf(root,ans);
    right_boundary(root->right,ans);

    return ans;
}
```

Vertical Traversal of Binary Tree [TC:O(n*log(n)) & SC:O(n)]

👉 BFS

```
vector<int> verticalOrder(Node *root){
    vector<int> ans;
    map<int,vector<int>> mp;
    queue<pair<int,Node*>> q;
    q.push({0,root});

    while(q.size()){
        auto it=q.front();
        q.pop();

        mp[it.first].push_back(it.second->data);
        if(it.second->left) q.push({it.first-1,it.second->left});
        if(it.second->right) q.push({it.first+1,it.second->right});
    }

    for(auto it:mp){
        ans.insert(ans.end(),it.second.begin(),it.second.end());
    }
    return ans;
}
```

👉 DFS

```
void dfs(int col,int row, Node* root,map<pair<int,int>,vector<int>> &mp){
    mp[{col,row}].push_back(root->data);
    if(root->left) dfs(col-1,row+1,root->left,mp);
    if(root->right) dfs(col+1,row+1,root->right,mp);
}
vector<int> verticalOrder(Node *root){
    vector<int> ans;
    map<pair<int,int>,vector<int>> mp;
    dfs(0,0,root,mp);
    for(auto it:mp){
        ans.insert(ans.end(),it.second.begin(),it.second.end());
    }
    return ans;
}
```

Top View of Binary Tree [TC:O(n*log(n)) & SC:O(w)]

👉 BFS

```
vector<int> topView(Node *root){
    vector<int> ans;
    map<int,int> mp;
    queue<pair<int,Node*>> q;
    q.push({0,root});

    while(q.size()){
        auto it=q.front();
        q.pop();

        if(!mp[it.first]) mp[it.first] = it.second->data;
        if(it.second->left) q.push({it.first-1,it.second->left});
        if(it.second->right) q.push({it.first+1,it.second->right});
    }

    for(auto it:mp) ans.push_back(it.second);
    return ans;
}
```

👉 DFS

```
void dfs(Node *root, int col,int row, map<int,pair<int,int>> &mp){
    if (!root) return;
    if(mp.find(col)==mp.end()) mp[col] = {row,root->data};
    else if(mp[col].first>row) mp[col] = {row,root->data};
    dfs(root->left, col - 1,row+1, mp);
    dfs(root->right, col + 1,row+1, mp);
}

vector<int> topView(Node *root){
    vector<int> ans;
    map<int,pair<int,int>> mp;
    dfs(root,0,0,mp);

    for(auto it:mp) ans.push_back(it.second.second);
    return ans;
}
```

Bottom View of Binary Tree[TC:O(n*log(n)) & SC:O(w)]

👉 BFS

```
vector <int> bottomView(Node *root) {
    vector<int> ans;
    map<int,int> mp;
    queue<pair<int,Node*>> q;
    q.push({0,root});

    while(q.size()){
        auto it=q.front();
        q.pop();

        mp[it.first] = it.second->data;
        if(it.second->left) q.push({it.first-1,it.second->left});
        if(it.second->right) q.push({it.first+1,it.second->right});
    }

    for(auto it:mp) ans.push_back(it.second);
    return ans;
}
```

👉 DFS

```
void dfs(Node *root, int col,int row, map<int,pair<int,int>> &mp){
    if (!root) return;
    if(mp.find(col)==mp.end()) mp[col] = {row,root->data};
    else if(mp[col].first<=row) mp[col] = {row,root->data};
    dfs(root->left, col - 1,row+1, mp);
    dfs(root->right, col + 1,row+1, mp);
}
vector <int> bottomView(Node *root) {
    vector<int> ans;
    map<int,pair<int,int>> mp;
    dfs(root,0,0,mp);

    for(auto it:mp) ans.push_back(it.second.second);
    return ans;
}
```

Left View of Binary Tree [TC:O(n*log(n)) & SC:O(w)]

👉 BFS

```
vector<int> leftView(Node *root){
    vector<int> ans;
    if (!root) return ans;
    queue<Node *> q;
    q.push(root);

    while (!q.empty()){
        int n = q.size();
        Node *curr = q.front();
        ans.push_back(curr->data);

        while (n--){
            curr = q.front();
            q.pop();
            if (curr->left) q.push(curr->left);
            if (curr->right) q.push(curr->right);
        }
    }
    return ans;
}
```

👉 DFS

```
void dfs(Node *root, int level, vector<int> &ans){
    if(!root) return ;
    if(ans.size()==level) ans.push_back(root->data);
    dfs(root->left, level+1, ans);
    dfs(root->right, level+1, ans);
}

vector<int> leftView(Node *root){
    vector<int> ans;
    dfs(root, 0, ans);
    return ans;
}
```

Right View of Binary Tree [TC:O(n*log(n)) & SC:O(w)]

👉 BFS

```
vector<int> rightView(Node *root){
    vector<int> ans;
    if (!root) return ans;
    queue<Node *> q;
    q.push(root);

    while (!q.empty()){
        int n = q.size();

        while (n--){
            Node *curr = q.front();
            q.pop();
            if (!n) ans.push_back(curr->data);
            if (curr->left) q.push(curr->left);
            if (curr->right) q.push(curr->right);
        }
    }
    return ans;
}
```

👉 DFS

```
void dfs(Node *root, int level, vector<int> &ans){
    if (!root) return;
    if (ans.size() == level) ans.push_back(root->data);
    dfs(root->right, level + 1, ans);
    dfs(root->left, level + 1, ans);
}

vector<int> rightView(Node *root){
    vector<int> ans;
    dfs(root, 0, ans);
    return ans;
}
```

Symmetric Tree [TC:O(n) & SC:O(h)]

Similar to -> determine if two trees are identicals

```
bool check(Node* l,Node* r){
    if(!l || !r) return l==r;
    if(l->data!=r->data) return false;
    if(!check(l->left,r->right)) return false;
    if(!check(l->right,r->left)) return false;
    return true;
}

bool isSymmetric(struct Node* root){
    if(!root) return true;
    return check(root->left,root->right);
}
```

Step 13.3: Hard Problems

Root to Leaf Paths [TC:O(n) & SC:O(h)] (DFS)

```
void dfs(Node* root,vector<int> &temp,vector<vector<int>> &ans){
    if(!root) return;
    temp.push_back(root->data);

    if(!root->left && !root->right) ans.push_back(temp);
    else {
        dfs(root->left,temp,ans);
        dfs(root->right,temp,ans);
    }

    temp.pop_back();
}

vector<vector<int>> Paths(Node* root){
    vector<vector<int>> ans;
    vector<int> temp;
    dfs(root,temp,ans);
    return ans;
}
```

Lowest Common Ancestor in a Binary Tree [TC:O(n) & SC:O(h)] (DFS)

```
Node* lca(Node* root ,int n1 ,int n2 ){
   if(!root || root->data==n1 || root->data==n2) return root;

   Node* l = lca(root->left,n1,n2);
   Node* r = lca(root->right,n1,n2);

   if(l && r) return root;
   if(l) return l;
   return r;
}
```

Maximum Width of Tree [TC:O(n) & SC:O(w)] (BFS)

▶️GFG

```
int getMaxWidth(Node* root) {
    int ans=1;
    queue<Node*> q;
    q.push(root);

    while(q.size()){
        int n=q.size();
        ans=max(ans,n);

        while(n--){
            root=q.front();
            q.pop();
            if(root->left) q.push(root->left);
            if(root->right) q.push(root->right);
        }
    }
    return ans;
}
```

▶️LeetCode *** (Overflow)

```
int widthOfBinaryTree(TreeNode* root) {
    int ans=1;
    queue<pair<TreeNode*,long>> q;
    q.push({root,1});

    while(q.size()){
        int n=q.size();
        int mn=q.front().second;
        int l,r;

        for(int i=0;i<n;i++){
            TreeNode* curr=q.front().first;
            int idx=q.front().second-mn;
            q.pop();

            if(i==0)    l=idx;
            if(i==n-1)  r=idx;

            if(curr->left)  q.push({curr->left,idx*2ll});
            if(curr->right) q.push({curr->right,idx*2ll+1});
        }
        ans=max(ans,r-l+1);
    }
    return ans;
}
```

Children Sum Parent [TC:O(n) & SC:O(h)]

▶️ GFG (Using BFS SC:O(w))

```
int isSumProperty(Node *root){
    if(!root->left && !root->right) return 1;

    int sum = 0;
    if(root->left){
        if(!isSumProperty(root->left)) return 0;
        sum+=root->left->data;
    }
    if(root->right){
        if(!isSumProperty(root->right)) return 0;
        sum+=root->right->data;
    }
    return sum==root->data;
}
```

▶️ CodeStudio

```
void changeTree(BinaryTreeNode < int > * root) {
    if(!root) return;
    int sum = 0;

    if(root->left) sum+=root->left->data;
    if(root->right) sum+=root->right->data;

    if(sum<root->data){
        if(root->left) root->left->data=root->data;
        else if(root->right) root->right->data=root->data;
    }

    sum=0;
    if (root->left) {
      changeTree(root->left);
      sum += root->left->data;
    }
    if (root->right) {
      changeTree(root->right);
      sum += root->right->data;
    }
    if(sum) root->data=sum;
}
```

** Nodes at distance K in binary tree [TC:O(n*log(n)) & SC:O(h)]

```
void solve_below (Node* root,int k, vector<int> &ans){
     if(!root || k<0) return;
     if(k==0) {
         ans.push_back(root->data);
         return;
     }
     solve_below(root->left,k-1,ans);
     solve_below(root->right,k-1,ans);
}
int solve(Node* root,int target,int k, vector<int> &ans){
    if(!root) return -1;
    if(root->data==target){
        solve_below(root,k,ans);
        return 1;
    }
    int l = solve(root->left,target,k,ans);
    if(l!=-1){
        if(l==k) {
            ans.push_back(root->data);
            return -1;
        }
        solve_below(root->right,k-l-1,ans);
        return l+1;
    }
    int r = solve(root->right,target,k,ans);
    if(r!=-1){
        if(r==k) {
            ans.push_back(root->data);
            return -1;
        }
        solve_below(root->left,k-r-1,ans);
        return r+1;
    }
    return -1;
}
vector <int> KDistanceNodes(Node* root, int target , int k){
    vector<int> ans;
    solve(root,target,k,ans);
    sort(ans.begin(),ans.end());
    return ans;
}
```

**Burning Tree [TC:O(n) & SC:O(h)]

```
int ans=0;
bool height(Node* node, int target, int &h){
    if(!node) {
        h=0;
        return 0;
    }

    int lh,rh;
    int l = height(node->left,target,lh);
    int r = height(node->right,target,rh);

    if(node->data==target){
        ans=max(ans,max(lh,rh));
        h = 1;
        return 1;
    }

    if(l) h = lh+1;
    if(r) h = rh+1;

    if(l || r){
        ans=max(ans,lh+rh);
        return 1;
    }

    h = 1+max(lh,rh);
    return 0;
}

int minTime(Node* root, int target){
    int h;
    height(root,target,h);
    return ans;
}
```

Count Number of Nodes in a Binary Tree [TC:O(log(n)^2) & SC:O(log(n))]

```
int left_height(Node* root){
    int ct=0;
    while(root){
        root=root->left;
        ct++;
    }
    return ct;
}

int right_height(Node* root){
    int ct=0;
    while(root){
        root=root->right;
        ct++;
    }
    return ct;
}

int countNodes(Node* root) {
    if(!root) return 0;

    int l = left_height(root);
    int r = right_height(root);

    if(l==r) return (1<<l)-1;

    int ans = 1;
    ans+=countNodes(root->left);
    ans+=countNodes(root->right);

    return ans;
}
```

Unique Binary Tree Requirements [TC:O(1) & SC:O(1)]

```
bool isPossible(int a,int b){
    return (a+b)%2 ;
}
```

Tree from Inorder & Preorder [TC:O(n^2) & SC:O(n)]

Reduce time complexity to O(n*log(n)) by storing index in map

👉 Method 1

```
int search(int l,int r,int target,int in[]){
    for(int i=l;i<=r;i++){
        if(in[i]==target) return i;
    }
}
Node* build(int idx, int l, int r, int in[], int pre[]){
    if(l>r) return NULL;

    int curr = search(l,r,pre[idx],in);

    Node* root = new Node(pre[idx]);
    root->left = build(idx+1,l,curr-1,in,pre);
    root->right = build(idx+1+(curr-l),curr+1,r,in,pre);
    return root;
}
Node* buildTree(int in[],int pre[], int n){
    return build(0,0,n-1,in,pre);
}
```

👉 Method 2

```
Node* build(int &idx, int l, int r, int in[], int pre[]){
    if(l>r) return NULL;

    int curr = search(l,r,pre[idx],in);
    Node* root = new Node(pre[idx]);
    idx++;

    root->left = build(idx,l,curr-1,in,pre);
    root->right = build(idx,curr+1,r,in,pre);
    return root;
}
Node* buildTree(int in[],int pre[], int n){
    int idx=0;
    return build(idx,0,n-1,in,pre);
}
```

Tree from Postorder and Inorder [TC:O(n^2) & SC:O(n)]

👉 Method 1

```
int search(int l,int r,int target,int in[]){
    for(int i=l;i<=r;i++){
        if(in[i]==target) return i;
    }
}
Node* build(int idx, int l, int r, int in[], int pre[]){
    if(l>r) return NULL;

    int curr = search(l,r,pre[idx],in);

    Node* root = new Node(pre[idx]);
    root->left = build(idx-1-(r-curr),l,curr-1,in,pre);
    root->right = build(idx-1,curr+1,r,in,pre);
    return root;
}
Node *buildTree(int in[], int post[], int n) {
    return build(n-1,0,n-1,in,post);
}
```

👉 Method 2 (Call Right first)

```
Node* build(int &idx, int l, int r, int in[], int post[]){
    if(l>r) return NULL;

    int curr = search(l,r,post[idx],in);
    Node* root = new Node(post[idx]);
    idx--;

    root->right = build(idx,curr+1,r,in,post);
    root->left = build(idx,l,curr-1,in,post);
    return root;
}

Node *buildTree(int in[], int post[], int n) {
    int idx=n-1;
    return build(idx,0,n-1,in,post);
}
```

Tree from Preorder and Postorder Traversal

Serialize and Deserialize a Binary Tree [TC:O(n) & SC:O(n)]

```
vector<int> serialize(Node *root){
    vector<int> ans;
    queue<Node*> q;
    q.push(root);
    while(q.size()){
        root = q.front();
        q.pop();
        if(root) {
            ans.push_back(root->data);
            q.push(root->left);
            q.push(root->right);
        }
        else ans.push_back(-1);
    }
    return ans;
}
Node * deSerialize(vector<int> &A){
    Node* root = new Node(A[0]);
    queue<Node*> q;
    q.push(root);
    for(int i=1;i<A.size();i+=2){
        Node* curr = q.front();
        q.pop();

        if(A[i]!=-1) {
            Node* temp = new Node(A[i]);
            curr->left = temp;
            q.push(temp);
        }
        if(A[i+1]!=-1) {
            Node* temp = new Node(A[i+1]);
            curr->right = temp;
            q.push(temp);
        }
    }
    return root;
}
```

Morris Preorder Traversal of a Binary Tree [TC:O(n) & SC:O(1)]

```
vector <int> preorder(Node* root){
    vector<int> ans;

    while(root){
        if(!root->left){
            ans.push_back(root->data);
            root = root->right;
            continue;
        }

        Node* temp = root->left;
        while(temp->right && temp->right!=root){
            temp = temp->right;
        }

        if(temp->right){
            temp->right = NULL;
            root = root->right;
        }
        else{
            ans.push_back(root->data);
            temp->right = root;
            root = root->left;
        }
    }
    return ans;
}
```

Morris Inorder Traversal of a Binary Tree [TC:O(n) & SC:O(1)]

```
vector<int> inOrder(Node* root) {
    vector<int> ans;

    while(root){
        if(!root->left){
            ans.push_back(root->data);
            root = root->right;
            continue;
        }

        Node* temp = root->left;
        while(temp->right && temp->right!=root){
            temp = temp->right;
        }

        if(temp->right){
            ans.push_back(root->data);
            temp->right = NULL;
            root = root->right;
        }
        else{
            temp->right = root;
            root = root->left;
        }
    }
    return ans;
}
```

Morris Postorder Traversal of a Binary Tree [TC:O(n) & SC:O(1)]

//Do Reverse of PreOrder

```
vector <int> postOrder(Node* root){
    vector<int> ans;

    while(root){
        if(!root->right){
            ans.push_back(root->data);
            root = root->left;
            continue;
        }

        Node* temp = root->right;
        while(temp->left && temp->left!=root){
            temp = temp->left;
        }

        if(temp->left){
            temp->left = NULL;
            root = root->left;
        }
        else{
            ans.push_back(root->data);
            temp->left = root;
            root = root->right;
        }
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

**Flatten binary tree to linked list [TC:O(n) & SC:O(1)]

```
Node* solve(Node* root){
    if(!root->right && !root->left) return root;

    Node* l_last = NULL,*r_last = NULL;

    if(root->left) l_last = solve (root->left);
    if(root->right) r_last = solve (root->right);

    if(root->left) {
        l_last->right = root->right;
        root->right = root->left;
    }
    if(!r_last) r_last = l_last;

    root->left = NULL;
    return r_last;
}

void flatten(Node *root){
    solve(root);
}
```

Step 14: Binary Search Trees [Concept and Problems]

Step 14.1: Concepts

Introduction to Binary Search Tree [TC:O(n) & SC:O(1)]

```
bool isBSTTraversal(vector<int>& nums) {
    for(int i=1;i<nums.size();i++){
        if(nums[i-1]>=nums[i]) return false;
    }
    return true;
}
```

Search a node in BST [TC:O(h) & SC:O(h) //SC:O(1) by using iterative

```
bool search(Node* root, int x) {
    if(!root) return false;

    if(root->data==x) return true;
    if(root->data<x) return search(root->right,x);
    return search(root->left,x);
}
```

Minimum element in BST [TC:O(h) & SC:O(h)] //SC:O(1) by using iterative

```
int minValue(Node* root) {
    if(!root) return -1;
    if(root->left) return minValue(root->left);
    return root->data;
}
```

Step 14.2: Practice Problems

Ceil in BST [TC:O(h) & SC:O(h)] //SC:O(1) by using iterative

```
int findCeil(Node* root, int input) {
    if (root == NULL) return -1;

    if(root->data==input) return input;
    if(root->data < input) return findCeil(root->right,input);

    int curr = findCeil(root->left,input);
    if(curr == -1) return root->data;
    return curr;
}
```

Floor in BST [TC:O(h) & SC:O(h)] //SC:O(1) by using iterative

```
int floor(Node* root, int x) {
    if (root == NULL) return -1;

    if(root->data==x) return x;
    if(root->data > x) return floor(root->left,x);

    int curr = floor(root->right,x);
    if(curr == -1) return root->data;
    return curr;
}
```

Floor and ceil by Iterative [TC:O(h) & SC:O(1)]

```
void floorCeilBST(Node* root, int key){
    int floor = -1, ceil = -1;

    while (root) {
        if (root->data == key) {
            ceil = floor = root->data;
            root = NULL;
        }
        else if (root->data < key) {
            floor = root->data;
            root = root->right;
        }
        else {
            ceil = root->data;
            root = root->left;
        }
    }

    cout << key << ' ' << floor << ' ' << ceil << '\n';
}
```

Insert a node in a BST

➡️ Recursive **[TC:O(h) & SC:O(h)]**

```
Node* insert(Node* root, int key) {
    if(!root) return new Node(key);

    if(root->data < key) root->right = insert(root->right,key);
    else if(root->data > key) root->left = insert(root->left,key);

    return root;
}
```

➡️ Iterative **[TC:O(h) & SC:O(1)]**

```
Node* insert(Node* root, int key) {
    Node* prev = NULL;
    Node* temp = root;

    while (temp) {
        if (temp->data > key) {
            prev = temp;
            temp = temp->left;
        }
        else if (temp->data < key) {
            prev = temp;
            temp = temp->right;
        }
        else return root;
    }

    if (prev->data > key) prev->left = new Node(key);
    else prev->right = new Node(key);

    return root;
}
```

*Delete a node from BST

➡️ Recursive **[TC:O(h) & SC:O(h)]**

```
Node* delete_root(Node* root){
    if(!root->left) return root->right;
    if(!root->right) return root->left;

    Node*temp = root->left;
    while(temp->right) temp = temp->right;
    temp->right = root->right;
    return root->left ;
}
Node *deleteNode(Node *root, int x) {
    if(!root) return NULL;

    if(root->data==x) return delete_root(root);
    else if(root->data < x) root->right = deleteNode(root->right,x);
    else root->left = deleteNode(root->left,x);

    return root;
}
```

➡️ Iterative **[TC:O(h) & SC:O(1)]**

```
Node *deleteNode(Node *root, int x) {
    if(root->data==x) return delete_root(root);

    Node* temp = root;
    while(temp){
        if(temp->data < x) {
            if(temp->right && temp->right->data==x){
                temp->right = delete_root(temp->right);
                break;
            }
            temp = temp->right;
        }
        else {
            if(temp->left && temp->left->data==x){
                temp->left = delete_root(temp->left);
                break;
            }
            temp = temp->left;
        }
    }
    return root;
}
```

k-th smallest element in BST

➡️ Recursive **[TC:O(n) & SC:O(h)]**

```
int KthSmallestElement(Node *root, int &k) {
    if(!root) return -1;
    int val = KthSmallestElement(root->left,k);
    if(val!=-1) return val;
    if(k==1) return root->data;
    k--;
    return KthSmallestElement(root->right,k);
}
```

➡️ Iterative **[TC:O(n) & SC:O(1)] (Morris Traversal)**

```
int KthSmallestElement(Node *root, int k) {
    if(!root) return -1;
    int ans = -1;
    while(root){
        if(root->left){
            Node* temp = root->left;
            while(temp->right && temp->right!=root) temp = temp->right;
            if(temp->right){
                temp->right = NULL;
                if(k==1) ans = root->data;
                k--;
                root = root -> right;
            }
            else{
                temp->right = root;
                root = root -> left ;
            }
        }
        else{
            if(k==1) ans = root->data;
            k--;
            root = root -> right;
        }
    }
    return ans;
}
```

Kth largest element in BST

➡️ Recursive **[TC:O(n) & SC:O(h)]**

```
int kthLargest(Node *root, int &K){
    if(!root) return -1;

    int val = kthLargest(root->right,K);
    if(val!=-1) return val;
    if(K==1) return root->data;
    K--;

    return kthLargest(root->left,K);
}
```

➡️ Iterative **[TC:O(n) & SC:O(1)] (Morris Traversal)**

```
int kthLargest(Node *root, int ){
    if(!root) return -1;

    int ans = -1;
    while(root){
        if(root->right){
            Node* temp = root->right;
            while(temp->left && temp->left!=root) temp = temp->left;

            if(temp->left){
                temp->left = NULL;
                if(k==1) ans = root->data;
                k--;
                root = root -> left;
            }
            else{
                temp->left = root;
                root = root -> right ;
            }
        }
        else{
            if(k==1) ans = root->data;
            k--;
            root = root -> left;
        }
    }
    return ans;
}
```

Check for BST

➡️ Recursive **[TC:O(n) & SC:O(h)]**

```
bool check(Node* root,int mn,int mx){
    if(!root) return true;
    if(root->data<mn || root->data>mx) return false;

    if(!check(root->left,mn,root->data-1)) return false;
    return check(root->right,root->data+1,mx);
}

bool isBST(Node* root) {
    return check(root,INT_MIN,INT_MAX);
}
```

➡️ Recursive **[TC:O(n) & SC:O(h)] (Inorder)**

```
bool inOrder(Node* root,int &prv){
    if(!root) return true;
    if(!inOrder(root->left,prv)) return false;
    if(root->data<=prv) return false;
    prv = root->data;
    return inOrder(root->right,prv);
}

bool isBST(Node* root) {
    int prv = INT_MIN;
    return inOrder(root,prv);
}
```

➡️ Iterative **[TC:O(n) & SC:O(1)] (Morris Traversal)**

```
bool isBST(Node* root) {
    int prv = INT_MIN;

    while(root){
        if(!root->left){
            if(root->data<=prv) return false;
            prv = root->data;
            root = root->right;
            continue;
        }

        Node* temp = root->left;
        while(temp->right && temp->right!=root){
            temp = temp->right;
        }

        if(temp->right){
            if(root->data<=prv) return false;
            prv = root->data;
            temp->right = NULL;
            root = root->right;
        }
        else{
            temp->right = root;
            root = root->left;
        }
    }
    return true;
}
```

Lowest Common Ancestor in a BST

➡️ Recursive **[TC:O(h) & SC:O(h)]**

```
Node* LCA(Node *root, int n1, int n2){
   if(root->data > n1 && root->data > n2) return LCA(root->left,n1,n2);
   if(root->data < n1 && root->data < n2) return LCA(root->right,n1,n2);
   return root;
}
```

➡️ Iterative **[TC:O(h) & SC:O(1)]**

```
Node* LCA(Node *root, int n1, int n2){
    while(root){
        if(root->data > n1 && root->data > n2) root = root->left;
        else if(root->data < n1 && root->data < n2) root = root->right;
        else break;
    }
    return root;
}
```

Construct BST from Preorder

👉 Method 1 **[TC:O(n^2) & SC:O(n)]**

Make InOrder array by Sorting the PreOrder and Applying this.

👉 Method 2 **(Recursion)** **[TC:O(n) & SC:O(h)]**

```
Node* build(int &idx,int mx, int pre[], int size){
    if(idx==size || pre[idx]>mx) return NULL;

    Node* root = newNode(pre[idx]);
    idx++;

    root->left = build(idx,root->data,pre,size);
    root->right = build(idx,mx,pre,size);
    return root;
}

Node* post_order(int pre[], int size){
    int idx=0;
    return build(idx,INT_MAX,pre,size);
}
```

👉 Method 3 **(Using Stack) [TC:O(n) & SC:O(h)]**

```
Node* post_order(int pre[], int size){
    Node* root = newNode(pre[0]);
    Node* temp = root;
    stack<Node*> st;

    for(int i=1;i<size;i++){
        if(pre[i]<temp->data){
            temp->left = newNode(pre[i]);
            st.push(temp);
            temp = temp->left;
        }
        else{
            while(st.size() && st.top()->data<pre[i]){
                temp = st.top();
                st.pop();
            }
            temp->right = newNode(pre[i]);
            temp = temp->right;
        }
    }
    return root;
}
```

Construct BST from Postorder

👉 Method 1 **[TC:O(n^2) & SC:O(n)]**

Make InOrder array by Sorting the PreOrder and Apply this.

👉 Method 2 **(Recursion) [TC:O(n) & SC:O(h)]**

👉 Method 3 **(Using Stack)** **[TC:O(n) & SC:O(h)]**

```
Node *constructTree (int post[], int size){
    Node* root = new Node(post[size-1]);
    Node* temp = root;
    stack<Node*> st;

    for(int i=size-2;i>=0;i--){
        if(post[i]>temp->data){
            temp->right = new Node(post[i]);
            st.push(temp);
            temp = temp->right;
        }
        else{
            while(st.size() && st.top()->data>post[i]){
                temp = st.top();
                st.pop();
            }
            temp->left = new Node(post[i]);
            temp = temp->left;
        }
    }
    return root;
}
```

Inorder Successor/Predecessor in BST **[TC:O(h) & SC:O(1)]**

```
Node *predecessor(Node *root, int key){
    Node *pre = NULL;
    while (root){
        if (root->key < key){
            pre = root;
            root = root->right;
        }
        else root = root->left;
    }
    return pre;
}
```

```
Node *sucessor(Node *root, int key){
    Node *suc = NULL;
    while (root){
        if (root->key > key){
            suc = root;
            root = root->left;
        }
        else root = root->right;
    }
    return suc;
}
```

```
void findPreSuc(Node *root, int key){
    Node *pre = NULL,*suc = NULL;

    while (root){
        if (root->key == key){
            if (root->left){
                Node *tmp = root->left;
                while (tmp->right) tmp = tmp->right;
                pre = tmp;
            }
            if (root->right){
                Node *tmp = root->right;
                while (tmp->left) tmp = tmp->left;
                suc = tmp;
            }
            return;
        }
        else if (root->key > key){
            suc = root;
            root = root->left;
        }
        else{
            pre = root;
            root = root->right;
        }
    }
}
```

Merge two BST 's [TC:O(n+m) & SC:O(h1+h2)]

```
vector<int> merge(Node *root1, Node *root2){
    vector<int> ans;
    stack<Node*> st1,st2;

    while(root1 || root2 || st1.size() || st2.size()){
        while(root1){
            st1.push(root1);
            root1=root1->left;
        }
        while(root2){
            st2.push(root2);
            root2=root2->left;
        }

        if(st2.empty() || (st1.size() &&
st1.top()->data<st2.top()->data)){
            root1 = st1.top();
            st1.pop();
            ans.push_back(root1->data);
            root1 = root1->right;
        }
        else {
            root2 = st2.top();
            st2.pop();
            ans.push_back(root2->data);
            root2 = root2->right;
        }

    }
    return ans;
}
```

Find a pair with given target in BST [TC:O(n) & SC:O(2*h)]

```
int isPairPresent(struct Node *root, int target){
    Node *temp1 = root, *temp2 = root;
    stack<Node*> st1,st2;

    while(temp1 || temp2 || st1.size() || st2.size()){
        while(temp1){
            st1.push(temp1);
            temp1 = temp1->left;
        }
        while(temp2){
            st2.push(temp2);
            temp2=temp2->right;
        }

        if(st1.top()->data >= st2.top()->data) break;

        int sum = st1.top()->data + st2.top()->data;

        if(sum == target) return 1;
        else if(sum < target) {
            temp1 = st1.top()->right;
            st1.pop();
        }
        else {
            temp2 = st2.top()->left;
            st2.pop();
        }
    }
    return 0;
}
```

Fixing Two nodes of a BST [TC:O(n) & SC:O(1)]

```
void correctBST( struct Node* root ){
    Node* first = NULL,*second = NULL,*prv = NULL;
    stack<Node*> st;

    while(root || st.size()){
        if(root){
            st.push(root);
            root = root->left;
        }
        else{
            root = st.top();
            st.pop();
            if(prv && prv->data>root->data){
                if(first) {
                    second = root;
                    break;
                }
                else {
                    first = prv;
                    second = root;
                }
            }
            prv = root;
            root = root->right;
        }
    }
    swap(first->data,second->data);
}
```

Largest BST in BT **[TC:O(n) & SC:O(h)]**

//

Step 15: Graphs [Concepts & Problems]

Step 15.1: Learning

Graph Representation [TC:O(V+E) & SC:O(V+E)]

```
vector<vector<int>> printGraph(int V, vector<int> adj[]) {
    vector<vector<int>> new_adj(V);
    for(int i=0;i<V;i++) {
        new_adj[i].push_back(i);
        new_adj[i].insert(new_adj[i].end(),adj[i].begin(),adj[i].end());
    }
    return new_adj;
}
```

BFS of a Graph [TC:O(V+2*E) & SC:O(V+V)] (visited + queue)

```
void BFS(int node,vector<int> &vis,vector<int> adj[]){
    queue<int> q;
    q.push(node);
    vis[node]=1;

    while(q.size()){
        node=q.front();
        q.pop();

        for(auto it:adj[node]){
            if(!vis[it]) {
                q.push(it);
                vis[it]=1;
            }
        }
    }
}
```

DFS of a Graph [TC:O(V+2*E) & SC:O(V+V)] (visited + stack)

```
void DFS(int node,vector<int> &vis,vector<int> adj[]){
    vis[node]=1;
    for(auto it:adj[node]){
        if(!vis[it]) DFS(it,vis,adj);
    }
}
```

Step 15.2: Problems on BFS/DFS

Number of Provinces

👉 Method 1 **[TC:O(V+2*E) & SC:O(V+V)] + [TC:O(V^2) & SC:O(V+E)]**

```
int numProvinces(vector<vector<int>> matrix, int V) {
    vector<int> adj[V],vis(V,0);

    for(int i=0;i<V;i++){
        for(int j=0;j<V;j++){
            if(matrix[i][j]){
                adj[i].push_back(j);
                adj[j].push_back(i);
            }
        }
    }

    int ans = 0;

    for(int i=0;i<V;i++){
        if(!vis[i]){
            ans++;
            // DFS(i,vis,adj);
            BFS(i,vis,adj);
        }
    }

    return ans;
}
```

👉 Method 2 **[TC:O(V^2) & SC:O(V+V)]** (visited + queue/stack)

```cpp
void DFS(int node,vector<int> &vis,vector<vector<int>> &matrix){
    vis[node] = 1;
    for(int i=0;i<matrix.size();i++){
        if(matrix[node][i] && !vis[i]) DFS(i,vis,matrix);
    }
}

void BFS(int node,vector<int> &vis,vector<vector<int>> &matrix){
    queue<int> q;
    q.push(node);
    vis[node]=1;
    while(q.size()){
        node=q.front();
        q.pop();
        for(int i=0;i<matrix.size();i++){
            if(matrix[node][i] && !vis[i]){
                q.push(i);
                vis[i]=1;
            }
        }
    }
}

int numProvinces(vector<vector<int>> matrix, int V) {
    vector<int> vis(V,0);
    int ans = 0;

    for(int i=0;i<V;i++){
        if(!vis[i]){
            ans++;
            // DFS(i,vis,matrix);
            BFS(i,vis,matrix);
        }
    }
    return ans;
}
```

Rotten Oranges [TC:O(n*m) & SC:O(n*m)]

```
int orangesRotting(vector<vector<int>>& grid) {
    int n=grid.size();
    int m=grid[0].size();
    int dx[]={1,-1,0,0};
    int dy[]={0,0,1,-1};
    vector<vector<int>> vis(n,vector<int>(m,0));
    queue<pair<int,int>> q;

    int ans=0,fresh=0;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(grid[i][j]==2) q.push({i,j});
            else if(grid[i][j]==1) fresh++;
        }
    }

    while(q.size()){
        int x=q.front().first;
        int y=q.front().second;
        q.pop();

        for(int i=0;i<4;i++){
            int nx=x+dx[i];
            int ny=y+dy[i];

            if(nx>=0 && ny>=0 && nx<n && ny<m && grid[nx][ny]==1 &&

vis[nx][ny]==0){
                vis[nx][ny]=vis[x][y]+1;
                ans=max(ans,vis[nx][ny]);
                fresh--;
                q.push({nx,ny});
            }
        }
    }

    if(fresh) return -1;
    return ans;
}
```

Flood fill Algorithm [TC:O(n*m) & SC:O(n*m)]

👉 BFS

```
vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc,
int newColor) {
    vector<vector<int>> ans = image;
    if(image[sr][sc]==newColor) return ans;

    int n=image.size();
    int m=image[0].size();
    int dx[]={1,-1,0,0};
    int dy[]={0,0,1,-1};

    queue<pair<int,int>> q;
    ans[sr][sc]=newColor;
    q.push({sr,sc});

    while(q.size()){
        int x=q.front().first;
        int y=q.front().second;
        q.pop();

        for(int i=0;i<4;i++){
            int nx=x+dx[i];
            int ny=y+dy[i];

            if(nx>=0 && ny>=0 && nx<n && ny<m &&
            image[nx][ny]==image[sr][sc] && ans[nx][ny]!=newColor){
                ans[nx][ny]=newColor;
                q.push({nx,ny});
            }
        }
    }

    return ans;
}
```

👉 DFS

```
int dx[4]={1,-1,0,0};
int dy[4]={0,0,1,-1};

void dfs(int x,int y,vector<vector<int>>& image, int newColor,
vector<vector<int>>& ans){
    int n=image.size();
    int m=image[0].size();

    ans[x][y] = newColor;

    for(int i=0;i<4;i++){
        int nx=x+dx[i];
        int ny=y+dy[i];

        if(nx>=0 && ny>=0 && nx<n && ny<m &&
        image[nx][ny]==image[x][y] && ans[nx][ny]!=newColor){
            dfs(nx,ny,image,newColor,ans);
        }
    }
}

vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc,
int newColor) {
    vector<vector<int>> ans = image;
    if(image[sr][sc]==newColor) return ans;
    dfs(sr,sc,image,newColor,ans);
    return ans;
}
```

Detect cycle in an undirected graph

//DFS

Step 16: Dynamic Programming [Patterns and Problems]

//

Number of distinct subsequences [TC:O(n) & SC:O(n)] SC:O(1) by storing only prev

For recursive Click here

```
int distinctSubsequences(string s){
    int n=s.size();
    vector<int> count(26,0);
    vector<int> dp(n+1,0);
    dp[0] = 1;

    for(int i=1;i<=n;i++){
        dp[i] = ((2*dp[i - 1] - count[s[i-1]-'a'])%M + M )  % M;
        count[s[i-1]-'a'] = dp[i - 1];
    }
    return dp[n];
}
```

//

Step 17: Tries

//

Step 18: Strings

//

To Solve

Construct Binary Tree from Preorder and Postorder Traversal

Construct BST from Postorder

My Profile

- ➢ Linkedin
- ➢ Github
- ➢ Codeforces
- ➢ Codechef
- ➢ LeetCode
- ➢ Gfg
- ➢ HackerRank
- ➢ HackerEarth
- ➢ AtCoder
- ➢ Instagram
- ➢ Gmail : ashishk6285@gmail.com