DSA SHEET BY ARSH GOYAL

SOLUTION BY SHIVANI PATEL

Problem Statement

1. **Easy Level-Find the Duplicate Number.**

Code:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

int dupl(vector<int>&num)
{
    int n=num.size();
    unordered_map<int,int>m;
    for(int i=0;i<n;i++)
    {
        m[num[i]]++;
        if(m[num[i]]>1)
        return num[i];
    }
return 0;



}
int main()
{
   vector<int>num={1,3,4,2,2};

   cout<<dupl(num)<<endl;
   return 0;
}
```

122. **1.Easy level- Sort an array of 0s, 1s and 2s**

Code:

```
#include <bits/stdc++.h>
using namespace std;


void sort(int a[], int n)
```

```
{
   int lo = 0;
   int hi = n - 1;
   int mid = 0;


   while (mid <= hi) {
      switch (a[mid]) {


      case 0:
         swap(a[lo++], a[mid++]);
         break;


      case 1:
         mid++;
         break;

      case 2:
         swap(a[mid], a[hi--]);
         break;
      }
   }
}


void printArray(int arr[], int n)
{

   for (int i = 0; i < n; i++)
      cout << arr[i] << " ";
}


int main()
{
   int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
   int n = sizeof(arr) / sizeof(arr[0]);
```

```
    sort(arr, n);



    printArray(arr, n);

    return 0;
}
```

3.

Easy level-3 Remove Duplicates from Sorted Array.

Code:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

int removeDuplicates(vector<int>& nums)
{
    set<int>s;
    for(int i=0;i<nums.size();i++)
    {
        s.insert(nums[i]);
    }
    int k=0;
    int p=s.size();
    for(auto it:s)
    {
        nums[k]=it;
        k++;
    }
    return p;
}
int main()
{
    vector<int>nums={0,0,1,1,1,2,2,3,3,4};
    cout<<removeDuplicates(nums)<<endl;


    return 0;
}
```

4.

Easy level-4 Set Matrix Zeroes

Code:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

void setZeroes(vector<vector<int>>& matrix) {
        int m=matrix.size(), n=matrix[0].size();

       bool col=true, row=true;
        for(int i=0; i<m; i++)
            for(int j=0; j<n; j++)
                if(matrix[i][j]==0){
                    if(i==0)
                        row = false;
                    if(j==0)
                        col = false;
                    matrix[0][j]=0;
                    matrix[i][0]=0;
                }

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

        if(col==false)
            for(int i=0; i<m; i++)
                matrix[i][0]=0;
         if(row==false)
            for(int j=0; j<n; j++)
                matrix[0][j]=0;
    }
int main()
{
    vector<vector<int>>matrix={{1,1,1},{1,0,1},{1,1,1}};
    setZeroes(matrix);
    for (int i = 0; i < matrix.size(); i++) {
    for (int j = 0; j < matrix[0].size(); j++) {
      cout << matrix[i][j] << " ";
    }
    cout<<"\n";

}
    return 0;
}
```

5.

Easy level-5 Move Zeroes

Code:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

 void reorder(int A[], int n)
{

    int k = 0;


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

        if (A[i] != 0) {
            A[k++] = A[i];
        }
    }


    for (int i = k; i < n; i++) {
        A[i] = 0;
    }
}

int main(void)
{
    int A[] = { 6, 0, 8, 2, 3, 0, 4, 0, 1 };
    int n = sizeof(A) / sizeof(A[0]);
    reorder(A, n);

    for (int i = 0; i < n; i++) {
        printf("%d ", A[i]);
    }

    return 0;
}
```

6. **Best Time to Buy and Sell Stock**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
int maxprofit(int a[],int n)
{
    int pro=0;
    for(int i=0;i<n-1;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            int profit=a[j]-a[i];
            if(profit>pro)
            pro=profit;
        }
    }
    return pro;
}
int main()
{
    int a[]={7,1,5,3,6,4};
    int n=sizeof(a)/sizeof(a[0]);
    cout<<maxprofit(a,n);
    return 0;
}
```

7.

Chocolate Distribution Problem

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
int minimumdistribution(int a[],int n,int m)
{
    if(m==0 || n==0)
    return 0;
    sort(a,a+n);
    if(n<m)
    return -1;
    int mini=INT_MAX;
    for(int i=0;i+m-1<n;i++)
    {
       int diff=a[i+m-1]-a[i];
       if(diff<mini)
       mini=diff;
    }
    return mini;
}
int main()
{
    int a[]={7, 3, 2, 4, 9, 12, 56};

    int n=sizeof(a)/sizeof(a[0]);
    int m=3;
    cout<<minimumdistribution(a,n,m);
    return 0;
}
```

8.

Two Sum
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
int sumoftwo(int a[],int n,int target)
{
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(a[i]+a[j]==target)
            cout<<"a[i]= "<<i<<" "<<"a[j]= "<<j<<endl;
        }
    }
    return 0;
}
int main()
{
    int a[]={2,7,11,15};
    int n=sizeof(a)/sizeof(a[0]);
    int target=9;
    cout<<sumoftwo(a,n,target);
    return 0;
}
```

9.

Best Time to Buy and Sell Stock II
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
 int maxProfit(int prices[],int n) {
        int diff=0;
        for(int i=1;i<n;i++)
        {

            if(prices[i]>prices[i-1])
            {
                diff=diff+prices[i]-prices[i-1];
            }
        }
        return diff;

    }

int main()

{

    int prices[]={7,1,5,3,6,4};

    int n=sizeof(prices)/sizeof(prices[0]);


    cout<<maxProfit(prices,n);

    return 0;

}
```

SNo. **Problem Statement**

1. **Medium Level-Subarray Sums Divisible by K**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int subarraysDivByK(vector<int>& A, int K) {
        vector<int> counts(K, 0);
        int sum = 0;
        for(int x: A){
            sum += (x%K + K)%K;
            counts[sum % K]++;
        }
        int result = counts[0];
        for(int c : counts)
            result += (c*(c-1))/2;
        return result;
    }
int main()
{
    vector<int>A={ 4, 5, 0, -2, -3, 1};
    int n=A.size();
    int K=5;
    cout<<subarraysDivByK(A,K);
    return 0;
}
```

2. **Medium Level-Find All Duplicates in an Array**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int findalldupl(int a[],int n)
{
    unordered_map<int,int>m;
    for(int i=0;i<n;i++)
    {
        m[a[i]]++;
    }
    for(auto it:m)
    {
        if(it.second>1)
        {
            cout<<it.first<<" ";
        }
    }
    cout<<"\n";
    return 0;
}
int main()
{
    int a[]={4,3,2,7,8,2,3,1};
    int n=sizeof(a)/sizeof(a[0]);
    cout<<findalldupl(a,n);
    return 0;
}
```

3. **Medium Level-Container With Most Water**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int maxwater(vector<int>&v)
{
    int left=0;
    int right=v.size()-1;
    int maxarea=0;
    while(left<right){
        int area=min(v[left],v[right])*(right-left);
        maxarea=max(maxarea,area);
        if(v[left]<v[right])
        left++;
        else
        right--;
    }
    return maxarea;
}
int main()
{
    vector<int>v={1,8,6,2,5,4,8,3,7};
    int n=v.size();
    cout<<maxwater(v);
    return 0;
}
```

4. **3Sum (Brute as well as Optimal)**

Code:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

 void triplets(int a[],int n){
      /*bool have=false;
      for (int i=0; i<n-2; i++)
    {
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (a[i]+a[j]+a[k] == 0)
                {
                    cout << a[i] << " "<< a[j] << " "<< a[k] <<endl;

                         have = true;
                }
            }
        }
    }*/
    bool have = false;

    for (int i=0; i<n-1; i++)
    {

        unordered_set<int> s;
        for (int j=i+1; j<n; j++)
        {
            int x = -(a[i] + a[j]);
            if (s.find(x) != s.end())
            {
                printf("%d %d %d\n", x, a[i], a[j]);
                have = true;
            }
            else
                s.insert(a[j]);
        }
    }
    if(have==false)
    cout<<"triplet not exist"<<endl;

}
int main()
{

    int a[] = {0, -1, 2, -3, 1};
    int n = sizeof(a)/sizeof(a[0]);
    triplets(a, n);
    return 0;

}
```

5. **Medium Level-Maximum Points You Can Obtain from Cards**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
int findpoint(int a[],int n,int k)
{
        int sum=0;

        int ans=0;
        for(int i=0;i<k;i++){
            sum+=a[i];
        }
        ans=sum;
        int i=k-1,j=n-1;
        while(i>=0 && j>=n-k){
            sum-=a[i];
            sum+=a[j];
            i--;
            j--;
            ans=max(sum,ans);
        }
        return ans;
}
int main()
{
    int a[]={1,2,3,4,5,6,1};
    int n=sizeof(a)/sizeof(a[0]);
    int k=3;
    cout<<findpoint(a,n,k);
    return 0;
}
```

6. **Medium Level-Subarray Sum Equals K**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
 int subarraySum(int nums[],int n, int k) {


        int count=0;
        unordered_map<int,int>prevSum;
        int sum=0;
        for(int i=0;i<n;i++){
        sum+=nums[i];
        if(sum==k)
        count++;
        if(prevSum.find(sum-k)!=prevSum.end()){
        count+=prevSum[sum-k];
        }
        prevSum[sum]++;
        }
```

```
  return count;
 }
int main()
{
  int nums[]={1,1,1};
  int n=sizeof(nums)/sizeof(nums[0]);
  int k=2;
  cout<<subarraySum(nums,n,k);
  return 0;
}
```

7. **Medium Level-Spiral Matrix**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
 vector<int> spiralOrder(vector<vector<int>>& matrix) {

     int T,B,L,R,dir;
     T=0;
     B=matrix.size()-1;
     L=0;
     R=matrix[0].size()-1;
     dir=0;

    vector<int>res;
    while(T<=B and L<=R)
    {
      if(dir==0)
      {
        for(int i=L;i<=R;i++)
          res.push_back(matrix[T][i]);
        T++;
      }
      else if(dir==1)
      {
        for(int i=T;i<=B;i++)
          res.push_back(matrix[i][R]);
        R--;
      }
```

```
            else if(dir==2)
             {
                for(int i=R;i>=L;i--)
                    res.push_back(matrix[B][i]);
                B--;
             }
              else if(dir==3)
             {
                for(int i=B;i>=T;i--)
                    res.push_back(matrix[i][L]);
                L++;
             }
             dir=(dir+1)%4;
         }
         return res;

     }
int main()
{
    vector<vector<int>> matrix{{1, 2, 3, 4},
                          {5, 6, 7, 8},
                          {9, 10, 11, 12},
                          {13, 14, 15, 16}};
    for(int x:spiralOrder(matrix))
    {
        cout << x << " ";
    }

    return 0;
}
```

8. **Medium Level-Word Search**

Code:

```
bool dfs(vector<vector<char>>& board, string &word,int i,int j){

        //base case
        if(word.size()==0) return true;
        if(i<0 || j<0 || i>=board.size() || j>= board[0].size() ||
board[i][j]!=word[0]) return false;

        char c = board[i][j];
```

```
     board[i][j] ='X';
    string s = word.substr(1);

    //dfs call
    bool res = dfs(board,s,i+1,j)||dfs(board,s,i-
1,j)||dfs(board,s,i,j+1)||dfs(board,s,i,j-1);

    //backtrack
    board[i][j] =c;
    return res;
  }
  bool exist(vector<vector<char>>& board, string word) {
    int m = board.size();
    int n = board[0].size();

    for(int i=0;i<m;i++){
      for(int j=0;j<n;j++){
        if(dfs(board,word,i,j)) return true;
      }
    }
    return false;

  }
```

9. **Medium Level-Jump Game**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
bool canJump(int a[],int n)
{
  int reach=0;
    for(int i=0;i<n;i++)
    {
      if(reach < i)

        return false;
        reach=max(reach,i+a[i]);
        }
      return true;
}
int main()
{
   int a[]={2,3,1,1,4};
   int n=sizeof(a)/sizeof(a[0]);
   cout<<canJump(a,n)<<endl;
   return 0;
}
```

10. **Medium Level-Merge Sorted Array.**

Code:

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;


void mergeArrays(int arr1[], int arr2[], int n1,
                              int n2, int arr3[])
{
   int i = 0, j = 0, k = 0;


   while (i<n1 && j <n2)
   {

      if (arr1[i] < arr2[j])
         arr3[k++] = arr1[i++];
      else
         arr3[k++] = arr2[j++];
   }


   while (i < n1)
      arr3[k++] = arr1[i++];


   while (j < n2)
      arr3[k++] = arr2[j++];
}
```

```
int main()
{
   int arr1[] = {1, 3, 5, 7};
   int n1 = sizeof(arr1) / sizeof(arr1[0]);

   int arr2[] = {2, 4, 6, 8};
   int n2 = sizeof(arr2) / sizeof(arr2[0]);

   int arr3[n1+n2];
   mergeArrays(arr1, arr2, n1, n2, arr3);


   for (int i=0; i < n1+n2; i++)
      cout << arr3[i] << " ";

   return 0;
}
```

11. **Medium Level-Majority Element.**

Code:

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;


 int majorityElement(vector<int>& nums) {

     unordered_map<int,int>m;
     int n=nums.size();
     for(int i=0;i<nums.size();i++)
     {
        m[nums[i]]++;
        if(m[nums[i]]>(n/2))
        return nums[i];
     }
     return 0;
 }
int main()
{
    vector<int>nums={3,2,3};
    int n=nums.size();
    cout<<majorityElement(nums);
    return 0;
}
```

12. **Medium Level-Reverse Pairs.**

Code:

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
class Solution
{
    public:
     void mergeArray(vector<int> &arr, int low, int mid, int high, int
&cnt)
{

       int l = low, r = mid + 1;
        while(l <= mid && r <= high){
            if((long)arr[l] > (long) 2 * arr[r]){
                cnt += (mid - l + 1);
                r++;
            }else{
                l++;
            }
        }
 sort(arr.begin()+low, arr.begin()+high+1 );
}

void mergeSort(vector<int> &arr, int low, int high, int &cnt)
{
    if (low < high)
    {
        int mid = low + (high - low) / 2;
        mergeSort(arr, low, mid, cnt);
        mergeSort(arr, mid + 1, high,cnt);

        mergeArray(arr, low, mid, high, cnt);
    }
}
```

```
    int reversePairs(vector<int>& arr) {
        int cnt = 0;
         mergeSort(arr, 0, arr.size() - 1, cnt);
        return cnt;

    }
};


int main()
{
    Solution ob;
    vector<int> v = {2,8,7,7,2};
    cout << (ob.reversePairs(v));

}
```

13. **Medium Level-Print all possible combinations of r elements in a given array of size n.**

Code:

```
#include <bits/stdc++.h>

using namespace std;

void comUtil(int arr[], int n, int r,

                    int index, int data[], int i);



void printCom(int arr[], int n, int r)

{



    int data[r];
    comUtil(arr, n, r, 0, data, 0);
}



void comUtil(int arr[], int n, int r,
                    int index, int data[], int i)
{

    if (index == r)
    {
        for (int j = 0; j < r; j++)
            cout << data[j] << " ";
        cout << endl;
        return;
    }



    if (i >= n)
        return;
    data[index] = arr[i];
    comUtil(arr, n, r, index + 1, data, i + 1);



    comUtil(arr, n, r, index, data, i+1);
}



int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCom(arr, n, r);
    return 0;
}
```

14. **Medium Level-Game Of Life.**

Code:

```
class Solution {
public:

    int life(vector<vector<int>>& board,int i,int j)
    {
        if(i<0||j<0||i>=board.size()||j>=board[0].size()||board[i][j]==0)
        {
            return 0;
        }
        return 1;
    }

    int checklive(vector<vector<int>>& board,int i,int j)
    {
        int k=0;

        if(life(board,i-1,j)==1)
        {
            k++;
        }
        if(life(board,i,j-1)==1)
        {
            k++;
        }
        if(life(board,i+1,j+1)==1)
        {
            k++;
        }
        if(life(board,i+1,j)==1)
        {
            k++;
        }
        if(life(board,i-1,j-1)==1)
        {
            k++;
        }
        if(life(board,i,j+1)==1)
        {
            k++;
        }
        if(life(board,i+1,j-1)==1)
        {
            k++;
        }
        if(life(board,i-1,j+1)==1)
        {
            k++;
        }
        if(board[i][j]==0 and k==3)
        {
            return 1;
        }
        if(board[i][j]==1 and (k==2||k==3))
        {
            return 1;
        }
        return 0;
    }
    void gameOfLife(vector<vector<int>>& board) {


vector<vector<int>>a(board.size(),vector<int>(board[0].size(),0));
        for(int i=0;i<board.size();i++){
            for(int j=0;j<board[0].size();j++){
                a[i][j]=checklive(board,i,j);
            }
        }
        board=a;
    }
};
```

SNo.	PROBLEM STATEMENT
1.	**Easy Level-Valid Parentheses** **Code:**

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
bool isValid(string s)
{
    stack<int>st;




           //stack st;
         for (int i=0;i<s.size();i++){
                 if(s[i]=='('||s[i]=='{'||s[i]=='['){
                     st.push(s[i]);
                      }
                         else{
                                if(st.size()==0) return false;
                                     if(s[i]==')'&& st.top()=='('||s[i]=='}'&&
st.top()=='{'||s[i]==']'&&
st.top()=='['){
                                                        st.pop();
                  }
                                                       else {return false;}
    }
}
                    if(st.size()==0) {return true;}
                                  return false;
}
int main()
{
   string s="()[]{}";
   if(isValid(s))
   cout<<"Valid";
   else
   cout<<"Not valid";

    return 0;
}
```

2. **Easy Level-Print all the duplicates in the input string.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
void dupl(string s)
{
    unordered_map<char,int>m;
    for(int i=0;i<s.size();i++)
    {
        m[s[i]]++;
    }
    for(auto it:m)
    {
        if(it.second>1)
         cout << it.first << ", count = " << it.second<< "\n";

    }
}
int main()
{
   string s="shivanishivi";
   dupl(s);
   return 0;
}
```

3. **Easy Level- Implement strStr()**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int impstr(string haystack, string needle)
{
    if(haystack.size()==0 and needle.size()==0)
   return 0;
   return haystack.find(needle);
}
```

```
int main()
{
  string haystack = "hello", needle = "ll";
  int res = impstr(haystack, needle);
    if (res == -1)
        cout << "Not present";
    else
        cout << "Present at index " << res;
        return 0;
}
```

4. **Easy Level- Longest Common Prefix.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

string longestCommonPrefix(vector<string>& s)
{
    if(s.size()==0)
    return " ";
    else
    {
      string s1=s[0];
      for(int i=1;i<s.size();i++)
      {
          for(int j=0;j<s1.size();j++)
          {
              if(j==s[i].size() or s1[j]!=s[i][j])
              {
                   s1=s1.substr(0,j);
                        break;
              }
          }
      }
      return s1;
    }

}
int main()
```

```
{
  vector<string>s={"flower","flow","flight"};
  string res=longestCommonPrefix(s);
  cout<<res;
  return 0;
}
```

5. **Easy Level- Valid Palindrome II**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

 bool check(int start,int end,string s)
    {

        while(start<end)
        {


            if(s[start]==s[end])
            {
                start++;
                end--;
            }
            else
                return false;
        }
        return true;
    }
    bool validPalindrome(string s) {

        int start=0;
        int end=s.length()-1;

        while(s[start]==s[end] && start<end)
        {
            start++;
            end--;
        }
        return check(start+1,end,s)|| check(start,end-1,s);
    }
int main()
{
  string s="mom";
  int start=0;
  int end=s.size()-1;
  if(check(start,end,s))
  cout<<"Palindrome";
  else
  cout<<"Not Palindrome";
  return 0;
}
```

SNo.	Problem Statement
1.	**Easy Level: Minimum Cost Tree From Leaf Values.** **Code:**

```
Input: arr = [6,2,4]

Output: 32

Explanation: There are two possible trees shown.

The first has a non-leaf node sum 36, and the second has non-leaf node
sum 32.
```

```
int mctFromLeafValues(vector<int>& arr) {

        stack<int>s;
        int sum=0;
        int t;
        for(int a:arr)
        {
            while(!s.empty() and a>s.top())
            {
                t=s.top();
                s.pop();
                if(s.empty())
                    sum+=t*a;
                else
                    sum+=t*min(s.top(),a);
            }
            s.push(a);
        }
        while(!s.empty())
        {
            t=s.top();
            s.pop();
            if(!s.empty())
                sum+=s.top()*t;
        }
        return sum;
    }
```

2.	**Easy Level: Daily Temperatures.**

Code:

```
Input: temperatures = [73,74,75,71,69,72,76,73]

Output: [1,1,4,2,1,1,0,0]

Input: temperatures = [30,40,50,60]

Output: [1,1,1,0]
```

```
vector<int> dailyTemperatures(vector<int>& temperatures) {


        int n=temperatures.size();
        stack<int>s;
        vector<int>ans(n,0);
        for(int i=0;i<n;i++)
        {
            while(s.size() and temperatures[s.top()]<temperatures[i])
            {
                ans[s.top()]=i-s.top();
                s.pop();
            }
            s.push(i);
        }
        return ans;
    }
```

3. **Medium Level: Distance of nearest cell having 1.**
Code:

```
Input: grid = {{0,1,1,0},{1,1,0,0},{0,0,1,1}}

Output: {{1,0,0,1},{0,0,1,1},{1,1,0,0}}

Explanation: The grid is-

0 1 1 0

1 1 0 0
```

```
0 0 1 1
0's at (0,0), (0,3), (1,2), (1,3), (2,0) and
(2,1) are at a distance of 1 from 1's at (0,1),
(0,2), (0,2), (2,3), (1,0) and (1,1)
respectively.
```

```
vector<vector<int>>nearest(vector<vector<int>>grid)
        {
            // Code here

            int n=grid.size();
            int m=grid[0].size();
            vector<vector<int>>ans(n,vector<int>(m,INT_MAX));
            queue<pair<int,int>>q;
            for(int i=0;i<n;i++)
            {
                for(int j=0;j<m;j++)
                {
                    if(grid[i][j]==1)
                    {
                        ans[i][j]=0;
                        q.push({i,j});
                    }
                }
            }
            while(!q.empty())
            {
                int i=q.front().first;
                int j=q.front().second;
                if((i-1)>=0 and ans[i][j]+1 < ans[i-1][j])
                {
                    ans[i-1][j]=ans[i][j]+1;
                    q.push({i-1,j});
                }
                if((j-1)>=0 and ans[i][j]+1 < ans[i][j-1])
                {
                    ans[i][j-1]=ans[i][j]+1;
                    q.push({i,j-1});
                }
                if((i+1)<n and ans[i][j]+1 < ans[i+1][j])
                {
                    ans[i+1][j]=ans[i][j]+1;
                    q.push({i+1,j});
                }
                if((j+1)<m and ans[i][j]+1 < ans[i][j+1])
                {
                    ans[i][j+1]=ans[i][j]+1;
                    q.push({i,j+1});
                }
                q.pop();
            }
            return ans;
        }
```

4. **Medium Level : Online Stock Span.**

Code:

```
Input

["StockSpanner", "next", "next", "next", "next", "next", "next",
"next"]

[[], [100], [80], [60], [70], [60], [75], [85]]

Output

[null, 1, 1, 1, 2, 1, 4, 6]



Explanation

StockSpanner stockSpanner = new StockSpanner();

stockSpanner.next(100); // return 1

stockSpanner.next(80);  // return 1

stockSpanner.next(60);  // return 1

stockSpanner.next(70);  // return 2

stockSpanner.next(60);  // return 1
stockSpanner.next(75);  // return 4, because the last 4 prices
(including today's price of 75) were less than or equal to today's
price.

stockSpanner.next(85);  // return 6
```

```
stack<pair<int,int>>s;
    int index=-1;
    StockSpanner() {



    }

    int next(int price) {

        index+=1;
        while(!s.empty() and s.top().second<=price)//previous greater
element
            s.pop();
        //if no previous greater
        if(s.empty())
        {
            s.push({index,price});
            return index+1;
        }
            int res=s.top().first;
            s.push({index,price});
            return index-res;

    }
```

5. **Medium Level:** **Rotten Oranges.**
Code:

```
Input: grid = {{0,1,2},{0,1,2},{2,1,1}}

Output: 1

Explanation: The grid is-

0 1 2

0 1 2
```

```
2 1 1
Oranges at positions (0,2), (1,2), (2,0)
will rot oranges at (0,1), (1,1), (2,2) and
(2,1) in unit time.
```

```
int orangesRotting(vector<vector<int>>& grid) {
        // Code here
        queue<pair<int, int>> rotten;
        int r = grid.size(), c = grid[0].size(), fresh = 0, t = 0;
        for(int i = 0; i < r; ++i){
            for(int j = 0; j < c; ++j){
                if(grid[i][j] == 2) rotten.push({i, j});
                else if(grid[i][j] == 1) fresh++;
            }
        }

        while(!rotten.empty()){
            int num = rotten.size();
            for(int i = 0; i < num; ++i){
                int x = rotten.front().first, y = rotten.front().second;
                rotten.pop();
                if(x > 0 && grid[x-1][y] == 1)
                {
                    grid[x-1][y] = 2;
                    fresh--;
                    rotten.push({x-1, y});
                };
                if(y > 0 && grid[x][y-1] == 1)
                {
                    grid[x][y-1] = 2;
                    fresh--;
                    rotten.push({x, y-1});
                };
                if(x < r-1 && grid[x+1][y] == 1)
                {
                    grid[x+1][y] = 2;
                    fresh--;
                    rotten.push({x+1, y});
                };
                if(y < c-1 && grid[x][y+1] == 1)
                {
                    grid[x][y+1] = 2;
                    fresh--;
                    rotten.push({x, y+1});
                };
            }
            if(!rotten.empty()) t++;
        }
        return (fresh == 0) ? t : -1;
    }
```

6. **Medium Level: sum-of-subarray-minimums.**

Code:

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

Output: 17

Explanation:

Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2],
[1,2,4], [3,1,2,4].

Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.

Sum is 17.
```

```
int sumSubarrayMins(vector<int>& arr) {


        int n = arr.size(), mod = 1e9+7;
        long sum = 0;

        stack<pair<int,long>> st;


        for(int i=n-1; i>=0; i--){

            while(!st.empty() && arr[i] <= arr[st.top().first]){
                st.pop();
            }
            if(st.empty()){
                st.push({i, (arr[i] * (n-i) % mod)});
            }



            else {
                st.push({i, (arr[i] * (st.top().first - i) % mod +
st.top().second)});
            }


            sum = (sum + st.top().second) % mod;
        }
        return sum;
    }
```

7.

Medium Level: Evaluate Reverse Polish Notation.

Code:

```
Input: tokens = ["2","1","+","3","*"]

Output: 9

Explanation: ((2 + 1) * 3) = 9
```

```
int evalRPN(vector<string>& tokens) {

        stack<int>s;
        int i=0;
        while(i<tokens.size())
        {
            if(tokens[i]=="+" || tokens[i]=="-" || tokens[i]=="*" ||
tokens[i]=="/" )
            {
                int a=s.top();
                s.pop();
                int b=s.top();
                s.pop();
                if(tokens[i]=="+")
                {
                    s.push(a+b);
                }
                if(tokens[i]=="-")
                {
                    s.push(b-a);
                }
                if(tokens[i]=="*")
                {
                    s.push(a*b);
                }
                if(tokens[i]=="/")
                {
                    int x=b/a;
                    s.push(x);
                }
                i++;
            }
            else
            {
                s.push(stoi(tokens[i]));
                i++;
            }
        }
        return s.top();
    }
```

8. **Medium Level: Circular tour .**
Code:

```
Input:
N = 4
Petrol = 4 6 7 4
Distance = 6 5 3 5
Output: 1
Explanation: There are 4 petrol pumps with
amount of petrol and distance to next
petrol pump value pairs as {4, 6}, {6, 5},
{7, 3} and {4, 5}. The first point from
where truck can make a circular tour is
2nd petrol pump. Output in this case is 1
(index of 2nd petrol pump).
```

```
int tour(petrolPump p[],int n)
    {
       //Your code here
       int totSum=0,currSum=0,j=0;
       for(int i=0;i<n;i++)
       {
           totSum+=p[i].petrol-p[i].distance;
           currSum+=p[i].petrol-p[i].distance;
           if(currSum<0)
           {
               j=i+1;
               currSum=0;
           }
       }
       return totSum<0?-1:j;
    }
```

9. **Medium Level: Flatten Nested List Iterator.**

Code:

```
Input: nestedList = [[1,1],2,[1,1]]

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

Explanation: By calling next repeatedly until hasNext returns false,
the order of elements returned by next should be: [1,1,2,1,1].
```

```
vector<int> flattenList;
    int index;

    NestedIterator(vector<NestedInteger> &nestedList)
    {
        index = 0;
        doDFS(nestedList);
    }
```

```
int next()
{
   return flattenList[index++];
}

bool hasNext()
{
   if (index < flattenList.size())
      return true;

   return false;
}

void doDFS(vector<NestedInteger> &nestedList)
{

   for (auto nestedInt : nestedList)
   {
      if (nestedInt.isInteger())
      {
         flattenList.push_back(nestedInt.getInteger());
      }
      else
      {
         auto list = nestedInt.getList();
         doDFS(list);
      }
   }
}
```

SNo.	Problem Statement
1.	**Medium Level-Maximum size rectangle binary sub-matrix with all 1s.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;



#define R 4
#define C 4




int maxHist(int row[])
{

    stack<int> res;

    int tval;

    int max_area = 0;

    int area = 0;
    int i = 0;
    while (i < C) {

        if (res.empty() || row[res.top()] <= row[i])
            res.push(i++);

        else {

            tval = row[res.top()];
            res.pop();
            area = tval * i;

            if (!res.empty())
```

```
                area = tval * (i - res.top() - 1);
            max_area = max(area, max_area);
        }
    }


    while (!res.empty()) {
        tval = row[res.top()];
        res.pop();
        area = tval * i;
        if (!res.empty())
            area = tval * (i - res.top() - 1);

        max_area = max(area, max_area);
    }
    return max_area;
}


int maxRectangle(int A[][C])
{

    int res = maxHist(A[0]);


    for (int i = 1; i < R; i++) {

        for (int j = 0; j < C; j++)


            if (A[i][j])
                A[i][j] += A[i - 1][j];


        res = max(res, maxHist(A[i]));
    }

    return res;
}
```

```
int main()
{
    int A[][C] = {
        { 0, 1, 1, 0 },
        { 1, 1, 1, 1 },
        { 1, 1, 1, 1 },
        { 1, 1, 0, 0 },
    };

    cout << "Area of maximum rectangle is "
         << maxRectangle(A);

    return 0;
}
```

2. **Medium Level:Find the number of islands**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

void dfs(vector<vector<int>>&mat,int i,int j,int r,int c)
{
    if(i<0 || j<0 || i>(r-1) || j>(c-1) || mat[i][j]!=1)
    {
        return;
    }
    if(mat[i][j]==1)
    {
        mat[i][j]=0;
        dfs(mat,i+1,j,r,c);
        dfs(mat,i-1,j,r,c);
         dfs(mat,i,j+1,r,c);
          dfs(mat,i,j-1,r,c);
           dfs(mat,i-1,j-1,r,c);
            dfs(mat,i+1,j+1,r,c);
             dfs(mat,i-1,j+1,r,c);
              dfs(mat,i+1,j-1,r,c);
    }
}

int countIslands(vector<vector<int>> &mat)
{
    int r = mat.size();
    int c = mat[0].size();
    int cnt = 0;
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
        {
            if (mat[i][j] == 1)
            {
                mat[i][j] = 0;
                cnt++;
                dfs(mat, i + 1, j, r, c);
                dfs(mat, i - 1, j, r, c);
                dfs(mat, i, j + 1, r, c);
                dfs(mat, i, j - 1, r, c);
                dfs(mat, i + 1, j + 1, r, c);
                dfs(mat, i - 1, j - 1, r, c);
                dfs(mat, i + 1, j - 1, r, c);
                dfs(mat, i - 1, j + 1, r, c);
            }
        }
    }
    return cnt;
}
int main()
{
    vector<vector<int>> mat = {{1, 1, 0, 0, 0},
                               {0, 1, 0, 0, 1},
                               {1, 0, 0, 1, 1},
                               {0, 0, 0, 0, 0},
                               {1, 0, 1, 0, 1}};

    cout << "Number of islands is: " << countIslands(mat);
    return 0;
}
```

3.

Medium Level:Given a matrix of ‘O’ and ‘X’, replace ‘O’ with ‘X’ if surrounded by ‘X’

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
#define M 6
#define N 6
void flood(char mat[][N],int x,int y,char pre,char newP)
{
     if (x < 0 || x >= M || y < 0 || y >= N)
        return;
    if (mat[x][y] != pre)
        return;


    mat[x][y] = newP;


    flood(mat, x+1, y, pre, newP);
    flood(mat, x-1, y, pre, newP);
    flood(mat, x, y+1, pre, newP);
    flood(mat, x, y-1, pre, newP);
}

int replace(char mat[][N])
{
     for (int i=0; i<M; i++)
      for (int j=0; j<N; j++)
          if (mat[i][j] == 'O')
             mat[i][j] = '-';


   for (int i=0; i<M; i++)
      if (mat[i][0] == '-')
        flood(mat, i, 0, '-', 'O');
   for (int i=0; i<M; i++)
      if (mat[i][N-1] == '-')
```

```
        flood(mat, i, N-1, '-', 'O');
    for (int i=0; i<N; i++)
      if (mat[0][i] == '-')
        flood(mat, 0, i, '-', 'O');
    for (int i=0; i<N; i++)
      if (mat[M-1][i] == '-')
        flood(mat, M-1, i, '-', 'O');


    for (int i=0; i<M; i++)
      for (int j=0; j<N; j++)
        if (mat[i][j] == '-')
            mat[i][j] = 'X';

}
int main()
{
    char mat[][N] =  {{'X', 'O', 'X', 'O', 'X', 'X'},
                     {'X', 'O', 'X', 'X', 'O', 'X'},
                     {'X', 'X', 'X', 'O', 'X', 'X'},
                     {'O', 'X', 'X', 'X', 'X', 'X'},
                     {'X', 'X', 'X', 'O', 'X', 'O'},
                     {'O', 'O', 'X', 'O', 'O', 'O'},
                    };

        replace(mat);
        for (int i=0; i<M; i++)
    {
      for (int j=0; j<N; j++)
          cout << mat[i][j] << " ";
      cout << endl;
    }


    return 0;
}
```

4. **Medium Level:Spiral Matrix**

Code:

```
#include <bits/stdc++.h>
#include <iostream>
```

```
#define M 3
#define N 3
using namespace std;
 vector<int> spiralOrder(vector<vector<int>>& matrix) {

        int T,B,L,R,dir;
        T=0;
        B=matrix.size()-1;
        L=0;
        R=matrix[0].size()-1;
        dir=0;

       vector<int>res;
       while(T<=B and L<=R)
       {
           if(dir==0)
           {
               for(int i=L;i<=R;i++)
                   res.push_back(matrix[T][i]);
               T++;
           }
           else if(dir==1)
           {
               for(int i=T;i<=B;i++)
                   res.push_back(matrix[i][R]);
               R--;
           }
          else if(dir==2)
           {
               for(int i=R;i>=L;i--)
                   res.push_back(matrix[B][i]);
               B--;
           }
            else if(dir==3)
           {
               for(int i=B;i>=T;i--)
                   res.push_back(matrix[i][L]);
               L++;
           }
           dir=(dir+1)%4;
        }
        return res;

    }
int main()
{
   vector<vector<int>>matrix={{1,2,3},{4,5,6},{7,8,9}};
       for (int x : spiralOrder(matrix))
    {
        cout << x << " ";
    }


    return 0;
}
```

5. **Medium Level:Rotate Image**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

#define N 4
using namespace std;
void rotate(int arr[N][N])
{

    for (int j = 0; j < N; j++)
    {
        for (int i = N - 1; i >= 0; i--)
            cout << arr[i][j] << " ";
        cout << '\n';
    }
}


int main()
{
    int arr[N][N] = { { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 },
                      { 9, 10, 11, 12 },
                      { 13, 14, 15, 16 } };
    rotate(arr);
    return 0;
}
```

<table>
<tr><th>SNo.</th><th>Problem Statement</th></tr>
<tr><td>1.</td><td>

Easy Level: Minimum Moves to Equal Array Elements.
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
int minmove(vector<int>&nums,int n)
{
    int c=0;
    int mini=*min_element(nums.begin(),nums.end());
    for(int i=0;i<n;i++)
    {
        if(nums[i]!=mini)
        c+=nums[i]-mini;
    }
    return c;
}
int main()
{
    vector<int>nums={1,2,3};
    int n=nums.size();
    cout<<minmove(nums,n);
    return 0;
}
```

</td></tr>
<tr><td>2.</td><td>

Easy Level: Add Binary.
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
string addBinary(string a, string b,int n1,int n2) {

        string res;

        int carry=0;
        while(n1>=0 || n2>=0)
        {
            int sum=carry;
            if(n1>=0)
```

</td></tr>
</table>

```
                sum+=a[n1--]-'0';
            if(n2>=0)
                sum+=b[n2--]-'0';
            carry=sum>1?1:0;
            res+=to_string(sum%2);
        }
        if(carry)
            res+=to_string(carry);
        reverse(res.begin(),res.end());
        return res;
    }
int main()
{
    string a="11";
    string b="1";
     int n1=a.size()-1;
     int n2=b.size()-1;

    cout<<addBinary(a,b,n1,n2);
    return 0;
}
```

3. **Easy Level:Maximum Product of Three Numbers.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
int maxProduct(vector<int>&nums,int n)
{
    int maxi=INT_MIN;
    if(n<3)
    return -1;
    for(int i=0;i<n-2;i++)
      for(int j=i+1;j<n-1;j++)
        for(int k=j+1;k<n;k++)
        maxi=max(maxi,nums[i]*nums[j]*nums[k]);
        return maxi;
}
int main()
{
    vector<int>nums={1,2,3};
    int n=nums.size();
    cout<<maxProduct(nums,n);
    return 0;
}
```

4. **Easy Level: Excel Sheet Column Title.**
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
string convertToTitle(int colnum)
{
    string res="";
    while(colnum)
    {
        char c='A'+(colnum-1)%26;
        res=c+res;
        colnum=(colnum-1)/26;
    }
    return res;
}
int main()
{
   int colnum=5;
   cout<<convertToTitle(colnum);
    return 0;
}
```

5. **Easy Level: Happy Number.**
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
 bool isHappy(int n) {
        if(n<9)
        {
            n=n*n;
        }
        while(n>9)
        {
            long long sum=0;
            while(n)
            {
                sum=sum+pow(n%10,2);
                n=n/10;
            }
            n=sum;
        }
        if(n==1 || n==7)
        {
            return true;
        }
        else {
            return false;
        }
    }
int main()
{
   int n=19;
   if(isHappy(n))
   cout<<"Yes";
   else
   cout<<"No";
    return 0;
}
```

6.

Easy Level: Palindrome Number.

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

bool palindrome(int x)
{
    int rem,a;
    long long int sum=0;
    a=x;
    while(x!=0)
    {
        rem=x%10;
        sum=sum*10+rem;
        x=x/10;
    }
    if(a>=0 and sum==a)
    {
        return true;
    }
    return false;
}

int main()
{
    int x=121;
    if(palindrome(x))
    cout<<"True";
    else
    cout<<"False";
    return 0;
}
```

7.

Easy Level : Missing Number.
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int missing(int a[],int n)
{
    int sum=0;
    int p=(n*(n+1)/2);
    for(int i=0;i<n;i++)
    {
        sum+=a[i];
    }
    return p-sum;
}
```

```
int main()
{
  int a[]={3,0,1};
  int n=sizeof(a)/sizeof(a[0]);
  cout<<missing(a,n);
  return 0;
}
```

8. **Easy Level : Reverse Integer.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int reverse(int x) {
        int rev=0;
        while(x!=0)
        {
            int p=x%10;
            x/=10;
            if(rev>INT_MAX/10||(rev==INT_MAX/10&&p>7))
                return 0;
            if(rev<INT_MIN/10||(rev==INT_MIN/10&&p<-8))
                return 0;
            rev=rev*10+p;
        }
        return rev;

    }

int main()
{
  int x=123;

  cout<<reverse(x);
  return 0;
}
```

9. **Easy Level : Power of Two**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

 bool isPowerOfTwo(int n) {

        if(n==0)
        {
            return false;
        }
        while(n!=0)
        {
            if(n==1)
                return true;

                if(n%2!=0)
                    return false;
                 else
                    n=n/2;
            }
                    return true;
        }


int main()
{
   int n=1;
   if(isPowerOfTwo(n))
   cout<<"YES";
   else
   cout<<"NO";
   return 0;
}
```

SNo. **Problem Statement**

1. **Easy Level : Permute two arrays such that sum of every pair is greater or equal to K.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
bool permute(int a[],int n, int b[], int m,int k)
{
    for(int i=0;i<n;i++)

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

            if(a[i]+b[j]>=k)
            return true;
            else
            return false;


}
int main()
{
    int a[]={2, 1, 3};
    int n=sizeof(a)/sizeof(a[0]);
    int b[]={7, 8, 9};
    int m=sizeof(b)/sizeof(b[0]);
    int k=10;
    if(permute(a,n,b,m,k))
    cout<<"YES";
    else
    cout<<"NO";
    return 0;
}
```

2. **Easy Level:Ceiling in a sorted array.**

Code:

```
#include <bits/stdc++.h>

#include <iostream>


using namespace std;

int findceil(int a[],int low,int high,int x)

{

    int i;

     if(x <= a[low])

        return low;

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

    {

        if(a[i] == x)

        return i;




        if(a[i] < x && a[i+1] >= x)

        return i+1;

    }
    return -1;
}
int main()
{
    int a[]={1, 2, 8, 10, 10, 12, 19};
    int n=sizeof(a)/sizeof(a[0]);

    int x=3;
    int p=findceil(a,0,n-1,x);
    if(p==-1)
    cout<<x;
    else
    cout<<x  <<" -> is : "<<  a[p];
    return 0;
}
```

3. **Easy Level : Find a pair with the given difference.**
Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
```

```
bool findpair(int a[],int n,int diff)

{
    int i=0;
    int j=1;
    while(i<n and j<n)
    {
        if(i!=j and (abs(a[i]-a[j])==diff))
        {
        cout<<a[i]<<" "<<a[j];
        return true;
        }
        else if(abs(a[i]-a[j])<diff)
        {
        j++;
        }
        else
        i++;
    }
    cout << "No such pair";
    return false;
}
int main()
{
    int a[]={1, 8, 30, 40, 100};
    int n=sizeof(a)/sizeof(a[0]);


    int diff=60;
    findpair(a,n,diff);


    return 0;
}
```

Medium Level Problem

SNo.	Problem Statement
1.	**Medium Level: Check if reversing a sub array make the array sorted.** **Code:**

```
#include<bits/stdc++.h>
using namespace std;


bool checkReverse(int a[], int n)
{
    if (n == 1)
        return true;
    int i;
    for (i=1; i < n && a[i-1] < a[i]; i++);
    if (i == n)
        return true;


    int j = i;
    while (j < n && a[j] < a[j-1])
    {
        if (i > 1 && a[j] < a[i-2])
            return false;
        j++;
    }

    if (j == n)
        return true;


    int k = j;


    if (a[k] < a[i-1])
       return false;

    while (k > 1 && k < n)
    {
        if (a[k] < a[k-1])
            return false;
        k++;
    }
    return true;
}



int main()
{
    int a[] = {1, 3, 4, 10, 9, 8};
    int n = sizeof(a)/sizeof(a[0]);
    checkReverse(a, n)? cout << "Yes" : cout << "No";
    return 0;
}
```

3. **Medium Level : Product of Array except itself**

Code:

```
#include <bits/stdc++.h>
using namespace std;


void productArray(int arr[], int n)
{

    if (n == 1) {
        cout << 0;
        return;
    }

    int i, temp = 1;


    int* prod = new int[(sizeof(int) * n)];

    memset(prod, 1, n);


    for (i = 0; i < n; i++) {
        prod[i] = temp;
        temp *= arr[i];
    }


    temp = 1;


    for (i = n - 1; i >= 0; i--) {
        prod[i] *= temp;
        temp *= arr[i];
    }
    for (i = 0; i < n; i++)
        cout << prod[i] << " ";

    return;
}


int main()
{
    int arr[] = { 10, 3, 5, 6, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    productArray(arr, n);
}
```

4. **Medium Level : Make all array elements equal with minimum cost.**

```
#include <bits/stdc++.h>
using namespace std;


int minCostToMakeElementEqual(int a[], int n)
{
    int o;
    if(n%2==1)
    o=a[n/2];

    else
    o=(a[n/2]+a[(n-2)/2])/2;
    int sum=0;
    for(int i=0;i<n;i++)
    sum+=abs(a[i]-o);
    return sum;
}


int main()
```

```
{
    int a[] = { 1, 100, 101 };
   int n = sizeof(a) / sizeof(a[0]);

   cout << (minCostToMakeElementEqual(a, n));
}
```

5. **Medium Level : Find Peak Element**

Code:

```
#include <bits/stdc++.h>
using namespace std;


 int findPeakElement(vector<int>& nums) {

    int left=0,right=nums.size()-1;
    while(left<right)
    {
       int mid=(left+right)/2;
       if(nums[mid]>nums[mid+1])
       right=mid;
       else
       left=mid+1;
    }
    return left;
 }


int main()
{
    vector<int>nums={1,2,3,1};
    int n=nums.size();

    cout<<findPeakElement(nums);
    return 0;
}
```

<table>
<tr><th>SNo.</th><th>Problem Statement</th></tr>
<tr><td>

1.

</td><td>

Easy LeveL : Middle of the Linked List.
Code:

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

Output: [3,4,5]

Explanation: The middle node of the list is node 3.
```

```
ListNode* middle(ListNode* head)
{
   ListNode* slow=head;
   ListNode* fast=head;
   if(head!=NULL)
   while(fast!=NULL and fast->next!=NULL)
   {
      fast=fast->next->next;
      slow=slow->next;
   }
   return slow;
}
```

</td></tr>
<tr><td>

2.

</td><td>

Easy Level : Linked List Cycle
Code:

```
Input: head = [3,2,0,-4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where the tail
connects to the 1st node (0-indexed).
```

```
 bool hasCycle(ListNode *head) {

      ListNode*slow=head;
      ListNode*fast=head;
      while(fast!=NULL && fast->next!=NULL){
         slow=slow->next;
         fast=fast->next->next;
```

</td></tr>
</table>

```
            if(fast==slow){
                return true;
            }
        }
        return false;
    }
```

3.

Easy Level : Convert Binary Number in a Linked List to Integer.
Code:

```
Input: head = [1,0,1]

Output: 5

Explanation: (101) in base 2 = (5) in base 10
```

```
int getDecimalValue(ListNode* head) {

        int num=head->val;
        while(head->next!=NULL)
        {
            num=num*2+head->next->val;
            head=head->next;
        }
        return num;
    }
```

4.

Easy Level : Remove Duplicates from Sorted List.
Code:

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

Output: [1,2]
```

```
ListNode* removeduplicate(ListNode* head){

    if(head==NULL)
    return head;
    ListNode* tmp=head;
    while(tmp->next!=NULL)
    {
        if(tmp->next->val==tmp->next->val)
        tmp->next=tmp->next->next;
        else
        tmp=tmp->next;
    }
    return head;
}
```

5.

Easy Level : Sort a linked list of 0s, 1s and 2s.

Code:

Input: 1 -> 1 -> 2 -> 0 -> 2 -> 0 -> 1 -> NULL
Output: 0 -> 0 -> 1 -> 1 -> 1 -> 2 -> 2 -> NULL
Input: 1 -> 1 -> 2 -> 1 -> 0 -> NULL
Output: 0 -> 1 -> 1 -> 1 -> 2 -> NULL

```
ListNode* sortList(ListNode* head)
{
    vector<int>v;
    if(head==NULL || head->next==NULL)
    return head;
    while(head!=NULL)
    {
        v.push_back(head->val);
        head=head->next;
    }
    sort(v.begin(),v.end());
    ListNode* node=new ListNode(v[0]);
    ListNode* start=node;
    for(int i=1;i<v.size();i++)
    {
        node->next=new ListNode(v[i]);
        node=node->next;
    }
    return start;
}
```

6.

Easy Level : Remove Linked List Elements.

Code:

```
Input: head = [1,2,6,3,4,5,6], val = 6

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

```
ListNode* removeElements(ListNode* head, int val) {
        if(head==NULL)
            return NULL;
        head->next=removeElements(head->next,val);
        if(head->val==val)
            return head->next;
        return head;
    }
```

7. **Easy Level : Merge Two Sorted Lists.**

Code:

```
Input: list1 = [1,2,4], list2 = [1,3,4]

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

```
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {

        ListNode *ans=NULL;
        if(!l1)
            return l2;
        else if(!l2)
            return l1;
        if(l1->val <= l2->val)
        {
            ans=l1;
            ans->next=mergeTwoLists(l1->next,l2);
        }
        else
        {
            ans=l2;
            ans->next=mergeTwoLists(l1,l2->next);
        }
        return ans;
    }
```

8. **Easy Level : Multiply two numbers represented by Linked Lists.**

Code:

```
Input : 9->4->6

        8->4

Output : 79464
```

```
Input : 3->2->1
        1->2
Output : 3852
```

```
long long multiplyTwoLists (Node* l1, Node* l2)
{
    long long N= 1000000007;
    long long num1 = 0, num2 = 0;
    while (l1 || l2){

        if(l1){
            num1 = ((num1)*10)%N + l1->data;
            l1 = l1->next;
        }

        if(l2)
        {
            num2 = ((num2)*10)%N + l2->data;
            l2 = l2->next;
        }

    }
    return ((num1%N)*(num2%N))%N;
}
```

9.

Easy Level : Intersection of Two Linked Lists.

Code:

```
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5],
skipA = 2, skipB = 3

Output: Intersected at '8'
```

```
ListNode *getIntersectionNode(ListNode *headA, ListNode *headB)
{


    if(headA == NULL || headB == NULL)
    return NULL;
    ListNode* a=headA;
    ListNode* b=headB;
    while(a!=b)
    {
        a = a == NULL? headB : a->next;
        b = b == NULL ? headA : b->next;
    }
    return a;
}
```

10. **Easy Level : Given only a pointer/reference to a node to be deleted in a singly linked list, how do you delete it?**

Code:

```
void deleteNode(Node* node)
{
    Node* prev;
    if(prev==NULL)
      return;
    else
    {
       while(node->next!=NULL)
       {
          node->data=node->next->data;
          prev=node;
          node=node->next;
       }
       prev->next=NULL;
    }
}
```

11. **Easy Level : Palindrome Linked List.**

Code:

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

Output: true
```

```
bool isPalindrome(ListNode* head)
{
    stack<int>s;
    ListNode* slow=head;
    ListNode* fast=head;
    while(fast and fast->next)
    {
       s.push(slow->data);
       slow=slow->next;
       fast=fast->next->next;

    }
    if(fast!=NULL)
     slow=slow->next;
      while(!s.empty() and slow)
       {
          if(s.top()!=slow->val)
             return false;
          s.pop();
          slow=slow->next;
       }
    return true;
}
```

12. **Easy Level : Reverse Linked List.**

Code:

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

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

```
ListNode* reverseList(ListNode* head) {
    ListNode* cur=head;
    ListNode* prev=NULL;
    while(cur!=NULL)
    {
       ListNode* tmp=cur->next;
       cur->next=prev;
       prev=cur;
       cur=tmp;
    }
    return prev;
}
```

SNo.	Problem Statement
1.	**Medium Level : Add Two Numbers.** **Code:**

```
Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.
```

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2)
{
    ListNode* dummy=new ListNode(0);
    ListNode* tmp=dummy;
    int carry=0;
    while(l1!=NULL || l2!=NULL || carry)
    {
        int sum=0;
        if(l1!=NULL)
        {
            sum+=l1->val;
            l1=l1->next;
        }
        if(l2!=NULL)
        {
            sum+=l2->val;
            l2=l2->next;
        }
        sum+=carry;
        carry=sum/10;
        ListNode* node=new ListNode(sum%10);
        tmp->next=node;
        tmp=tmp->next;
    }
    return dummy->next;
}
```

SNo.	Problem Statement
2.	**Medium Level : Copy List with Random Pointer.** **Code :**

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

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

```
class Solution {
public:
    Node* copyRandomList(Node* head) {

       Node *curr=head,*front=head;

    while(curr!=NULL)
    {
        front=curr->next;
        Node *copy=new Node(curr->val);
        curr->next=copy;
        copy->next=front;
        curr=front;
    }
    curr=head;
    while(curr!=NULL)
    {
        if(curr->random!=NULL)
        {
            curr->next->random=curr->random->next;
        }
        curr=curr->next->next;
    }
    curr=head;
    Node *dummy=new Node(0);
    Node *copy=dummy;
    while(curr!=NULL)
    {
        front=curr->next->next;
        copy->next=curr->next;
        curr->next=front;
        copy=copy->next;
        curr=curr->next;
    }
    return dummy->next;
    }
};
```

3. **Medium Level : Add Two Numbers II.**

Code:

```
Input: l1 = [7,2,4,3], l2 = [5,6,4]

Output: [7,8,0,7]
```

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2)
{
    stack<int>s1;
    stack<int>s2;

    ListNode* ans=new ListNode(0);
    while(l1)
    {
        st1.push(l1->val);
        l1=l1->next;
    }
    while(l2)
    {
        st2.push(l2->val);
        l2=l2->next;
    }
    int sum=0;
    while(!st1.empty() || !st2.empty())
    {
        if(!st1.empty())
        {
            sum+=st1.top();
            st1.pop();
        }

        if(!st2.empty())
        {
            sum+=st2.top();
            st2.pop();
        }
        ans->val=sum%10;
        sum/=10;
        ListNode* head=new ListNode(sum);
        head->next=ans;
        ans=head;

    }
    return ans->val==0?ans->next:ans;
}
```

4.

Medium Level : Reverse Linked List II.
Code:

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

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

```
ListNode* reverse(ListNode* head){

        ListNode* prev = NULL, *next = NULL, *current = head;
        while(current != NULL){
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;

        }

        return prev;
    }

    ListNode* reverseBetween(ListNode* head, int left, int right) {

        if(head == NULL || left == right){
            return head;
        }
        ListNode* prev, *tail = NULL, *temp = NULL;
        ListNode dummy(NULL);
        prev = &dummy;
        dummy.next = head;
        for(int i=0; i < left-1; i++){
            prev = prev->next;
        }
        tail = prev->next;
        for(int i=0; i< right - left;i++){
            temp = prev->next;
            prev->next = tail->next;
            tail->next = tail->next->next;
            prev->next->next = temp;
        }

        return dummy.next;
    }
```

5.

Medium Level : Reorder List.

Code:

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

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

```
void reorderList(ListNode* head)
{
    stack<int>s;
    ListNode* curr=head;
    while(curr)
    {
        s.push(curr);
        curr=curr->next;
    }
    curr=head;
    int n=s.size();
    ListNode* next;
    for(int i=0;i<n/2;i++)

    {
        next=curr->next;
        curr->next=s.top();
        s.pop();
        curr=curr->next;
        curr->next=next;
        curr=curr->next;
    }
    curr->next=NULL;
}
```

6. **Medium Level : Remove Nth Node From End of List.**

Code :

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

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

```
ListNode* removeNthFromEnd(ListNode* head, int n)
 {
     ListNode* dummy=neew ListNode();
     dummy->next=head;
     int c=0;
    while(dummy->next!=NULL)
    {
       dummy=dummy->next;
       c++;
    }
    int num=c-n;
    ListNode* tmp=new ListNode();
    tmp->next=head;
    while(num!=0)
    {
       tmp=tmp->next;
       num--;
    }
    if(c!=n)
       {
          tmp->next=tmp->next->next;
          return head;
       }
       else
       {
          head=head->next;
          return head;
       }
 }
```

7.

Medium Level : Flatten a Multilevel Doubly Linked List.
Code :

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

Output: [1,3,2]

Explanation: The multilevel linked list in the input is shown.

After flattening the multilevel linked list it becomes:
```

```
Node* flatten(Node* head) {
        Node* final = head;
        stack<Node*> s;
        Node* temp;
        while(head != nullptr){
            if(head->child != nullptr){
                if(head->next != nullptr){
                    temp = head->next;
                    s.push(temp);
                }
                head->child->prev = head;
                head->next = head->child;
                head->child = nullptr;

            }
            if(!s.empty() && head->next == nullptr){
                head->next = s.top();
                head->next->prev = head;
                s.pop();
            }
            head = head->next;

        }
        return final;
    }
```

8.

Medium Level : Partition List.
Code:

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

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

```
    ListNode* partition(ListNode* head, int x) {


        ListNode *small_head=new ListNode(0);
        ListNode *small=small_head;
        ListNode *high_head=new ListNode(0);
        ListNode *high=high_head;

        while(head!=NULL)
        {
            if(head->val<x)
            {
                small->next=head;
                small=small->next;
            }
            else
            {
                high->next=head;
                high=high->next;
            }
            head=head->next;
        }
        high->next=NULL;
        small->next=high_head->next;
        return small_head->next;

    }
```

9. **Medium Level : Remove Duplicates from Sorted List II.**
Code:

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

Output: [1,2,5]
```

```
ListNode* deleteDuplicates(ListNode* head)
 {
     if(head==NULL)
```

```
    return NULL;
    unordered_map<int,int>m;
    ListNode* tmp=head;
    while(tmp)
    {
       m[tmp->val]++;
       tmp=tmp->next;
    }
    ListNode* ans=new ListNode(-1);
    ListNode* tmp2=ans;
    for(auto i:m)
    {
       if(i.second==1)
       temp2->next = new ListNode(i.first);
       temp2 = temp2->next;
    }
    return ans->next;
 }
```

10. **Medium Level: Rearrange a Linked List in Zig-Zag fashion**
Code:

```
Input:  1->2->3->4

Output: 1->3->2->4

Explanation : 1 and 3 should come first before 2 and 4 in
zig-zag fashion, So resultant linked-list will be 1->3->2-
>4.


Input:  11->15->20->5->10

Output: 11->20->5->15->10
```

```
Node* zigzag(Node* head, bool flag)
 {
    if(!head || !head->next)
    return head;
    if(flag==1)
    {
       if(head->data > head->next->data)
```

```
          {
              swap(head->data,head->next->data);
              return zigzag(head->next,!flag);
          }
      }
      else {
         if (head->data < head->next->data)
             swap(head->data, head->next->data);
         return zigzag(head->next, !flag);
      }
   }
```

11. **Medium Level: Sort List.**
Code:

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

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

```
ListNode* sortList(ListNode* head)
{
    if(head==NULL || head->next==NULL)
    return head;
    vector<int>v;
    while(head!=NULL)
    {
        v.push_back(head->val);
        head=head->next;
    }
    sort(v.begin(),v.end());
    ListNode* ans=new ListNode(v[0]);
    ListNode* start=ans;
    for(int i=1;i<v.size();i++)
    {
        ans->next=new ListNode(v[i]);
        ans=ans->next;
    }
    return start;
}
```

12. **Medium Level:** **Sort List.**

Code:

```
Input: 17->15->8->12->10->5->4->1->7->6->NULL
Output: 8->12->10->4->6->17->15->5->1->7->NULL

Input: 8->12->10->5->4->1->6->NULL
Output: 8->12->10->4->6->5->1->NULL
```

```
ListNode* sortList(ListNode* head)
{
    if(head==NULL || head->next==NULL)
    return head;
    vector<int>v;
    while(head!=NULL)
    {
        v.push_back(head->val);
        head=head->next;
    }
    sort(v.begin(),v.end());
    ListNode* ans=new ListNode(v[0]);
    ListNode* start=ans;
    for(int i=1;i<v.size();i++)
    {
        ans->next=new ListNode(v[i]);
        ans=ans->next;
    }
    return start;
}
```

13. **Medium Level: Rearrange a given linked list in-place.**
Code:

```
Input:  1 -> 2 -> 3 -> 4

Output: 1 -> 4 -> 2 -> 3


Input:  1 -> 2 -> 3 -> 4 -> 5

Output: 1 -> 5 -> 2 -> 4 -> 3
```

```
void rearrange(Node* head)
```

```
{
    if (head == NULL)
        return;


    Node *prev = head, *curr = head->next;

    while (curr) {

        if (prev->data > curr->data)
            swap(prev->data, curr->data);


        if (curr->next && curr->next->data > curr->data)
            swap(curr->next->data, curr->data);

        prev = curr->next;

        if (!curr->next)
            break;
        curr = curr->next->next;
    }
}
```

SNo.	PROBLEM STATEMENT
1.	**Easy Level-Valid Parentheses** **Code:**

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
bool isValid(string s)
{
    stack<int>st;




           //stack st;
         for (int i=0;i<s.size();i++){
                 if(s[i]=='('||s[i]=='{'||s[i]=='['){
                     st.push(s[i]);
                      }
                         else{
                                if(st.size()==0) return false;
                                     if(s[i]==')'&& st.top()=='('||s[i]=='}'&&
st.top()=='{'||s[i]==']'&&
st.top()=='['){
                                                  st.pop();
                  }
                                                 else {return false;}
    }
}
                    if(st.size()==0) {return true;}
                                  return false;
}
int main()
{
   string s="()[]{}";
   if(isValid(s))
   cout<<"Valid";
   else
   cout<<"Not valid";

    return 0;
}
```

2.

Easy Level-Print all the duplicates in the input string.

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;
void dupl(string s)
{
    unordered_map<char,int>m;
    for(int i=0;i<s.size();i++)
    {
        m[s[i]]++;
    }
    for(auto it:m)
    {
        if(it.second>1)
         cout << it.first << ", count = " << it.second<< "\n";

    }
}
int main()
{
   string s="shivanishivi";
   dupl(s);
   return 0;
}
```

3.

Easy Level- Implement strStr()

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int impstr(string haystack, string needle)
{
    if(haystack.size()==0 and needle.size()==0)
   return 0;
   return haystack.find(needle);
}
```

```
int main()
{
  string haystack = "hello", needle = "ll";
  int res = impstr(haystack, needle);
    if (res == -1)
        cout << "Not present";
    else
        cout << "Present at index " << res;
        return 0;
}
```

4. **Easy Level- Longest Common Prefix.**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

string longestCommonPrefix(vector<string>& s)
{
    if(s.size()==0)
    return " ";
    else
    {
      string s1=s[0];
      for(int i=1;i<s.size();i++)
      {
          for(int j=0;j<s1.size();j++)
          {
              if(j==s[i].size() or s1[j]!=s[i][j])
              {
                   s1=s1.substr(0,j);
                        break;
              }
          }
      }
      return s1;
    }

}
int main()
```

```
{
  vector<string>s={"flower","flow","flight"};
  string res=longestCommonPrefix(s);
  cout<<res;
  return 0;
}
```

5. **Easy Level- Valid Palindrome II**

Code:

```
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

 bool check(int start,int end,string s)
    {

        while(start<end)
        {


            if(s[start]==s[end])
            {
                start++;
                end--;
            }
            else
                return false;
        }
        return true;
    }
    bool validPalindrome(string s) {

        int start=0;
        int end=s.length()-1;

        while(s[start]==s[end] && start<end)
        {
            start++;
            end--;
        }
        return check(start+1,end,s)|| check(start,end-1,s);
    }
int main()
{
  string s="mom";
  int start=0;
  int end=s.size()-1;
  if(check(start,end,s))
  cout<<"Palindrome";
  else
  cout<<"Not Palindrome";
  return 0;
}
```

SNo.	Problem Statement
1.	**Easy Level: Minimum Cost Tree From Leaf Values.** **Code:**

```
Input: arr = [6,2,4]

Output: 32

Explanation: There are two possible trees shown.

The first has a non-leaf node sum 36, and the second has non-leaf node
sum 32.
```

```
int mctFromLeafValues(vector<int>& arr) {

    stack<int>s;
    int sum=0;
    int t;
    for(int a:arr)
    {
        while(!s.empty() and a>s.top())
        {
            t=s.top();
            s.pop();
            if(s.empty())
                sum+=t*a;
            else
                sum+=t*min(s.top(),a);
        }
        s.push(a);
    }
    while(!s.empty())
    {
        t=s.top();
        s.pop();
        if(!s.empty())
            sum+=s.top()*t;
    }
    return sum;
  }
```

SNo.	Problem Statement
2.	**Easy Level: Daily Temperatures.**

Code:

```
Input: temperatures = [73,74,75,71,69,72,76,73]

Output: [1,1,4,2,1,1,0,0]

Input: temperatures = [30,40,50,60]

Output: [1,1,1,0]
```

```
vector<int> dailyTemperatures(vector<int>& temperatures) {



        int n=temperatures.size();
        stack<int>s;
        vector<int>ans(n,0);
        for(int i=0;i<n;i++)
        {
            while(s.size() and temperatures[s.top()]<temperatures[i])
            {
                ans[s.top()]=i-s.top();
                s.pop();
            }
            s.push(i);
        }
        return ans;
    }
```

3. **Medium Level: Distance of nearest cell having 1.**
Code:

```
Input: grid = {{0,1,1,0},{1,1,0,0},{0,0,1,1}}

Output: {{1,0,0,1},{0,0,1,1},{1,1,0,0}}

Explanation: The grid is-

0 1 1 0

1 1 0 0
```

```
0 0 1 1
0's at (0,0), (0,3), (1,2), (1,3), (2,0) and
(2,1) are at a distance of 1 from 1's at (0,1),
(0,2), (0,2), (2,3), (1,0) and (1,1)
respectively.
```

```
vector<vector<int>>nearest(vector<vector<int>>grid)
      {
          // Code here
          
          int n=grid.size();
          int m=grid[0].size();
          vector<vector<int>>ans(n,vector<int>(m,INT_MAX));
          queue<pair<int,int>>q;
          for(int i=0;i<n;i++)
          {
              for(int j=0;j<m;j++)
              {
                  if(grid[i][j]==1)
                  {
                      ans[i][j]=0;
                      q.push({i,j});
                  }
              }
          }
          while(!q.empty())
          {
              int i=q.front().first;
              int j=q.front().second;
              if((i-1)>=0 and ans[i][j]+1 < ans[i-1][j])
              {
                  ans[i-1][j]=ans[i][j]+1;
                  q.push({i-1,j});
              }
              if((j-1)>=0 and ans[i][j]+1 < ans[i][j-1])
              {
                  ans[i][j-1]=ans[i][j]+1;
```

```
                q.push({i,j-1});
            }
            if((i+1)<n and ans[i][j]+1 < ans[i+1][j])
            {
                ans[i+1][j]=ans[i][j]+1;
                q.push({i+1,j});
            }
            if((j+1)<m and ans[i][j]+1 < ans[i][j+1])
            {
                ans[i][j+1]=ans[i][j]+1;
                q.push({i,j+1});
            }
            q.pop();
        }
        return ans;
    }
```

4. **Medium Level : Online Stock Span.**

Code:

```
Input

["StockSpanner", "next", "next", "next", "next", "next", "next",
"next"]

[[], [100], [80], [60], [70], [60], [75], [85]]

Output

[null, 1, 1, 1, 2, 1, 4, 6]



Explanation

StockSpanner stockSpanner = new StockSpanner();

stockSpanner.next(100); // return 1

stockSpanner.next(80);  // return 1

stockSpanner.next(60);  // return 1

stockSpanner.next(70);  // return 2

stockSpanner.next(60);  // return 1
stockSpanner.next(75);  // return 4, because the last 4 prices
(including today's price of 75) were less than or equal to today's
price.

stockSpanner.next(85);  // return 6
```

```
stack<pair<int,int>>s;
    int index=-1;
    StockSpanner() {




    }

    int next(int price) {

        index+=1;
        while(!s.empty() and s.top().second<=price)//previous greater
element
            s.pop();
        //if no previous greater
        if(s.empty())
        {
            s.push({index,price});
            return index+1;
        }
            int res=s.top().first;
            s.push({index,price});
            return index-res;

    }
```

5. **Medium Level: Rotten Oranges.**

Code:

```
Input: grid = {{0,1,2},{0,1,2},{2,1,1}}

Output: 1

Explanation: The grid is-

0 1 2

0 1 2
```

```
2 1 1
Oranges at positions (0,2), (1,2), (2,0)
will rot oranges at (0,1), (1,1), (2,2) and
(2,1) in unit time.
```

```
int orangesRotting(vector<vector<int>>& grid) {
        // Code here
        queue<pair<int, int>> rotten;
        int r = grid.size(), c = grid[0].size(), fresh = 0, t = 0;
        for(int i = 0; i < r; ++i){
            for(int j = 0; j < c; ++j){
                if(grid[i][j] == 2) rotten.push({i, j});
                else if(grid[i][j] == 1) fresh++;
            }
        }

        while(!rotten.empty()){
            int num = rotten.size();
            for(int i = 0; i < num; ++i){
                int x = rotten.front().first, y = rotten.front().second;
                rotten.pop();
                if(x > 0 && grid[x-1][y] == 1)
                {
                    grid[x-1][y] = 2;
                    fresh--;
                    rotten.push({x-1, y});
                };
                if(y > 0 && grid[x][y-1] == 1)
                {
                    grid[x][y-1] = 2;
                    fresh--;
                    rotten.push({x, y-1});
                };
                if(x < r-1 && grid[x+1][y] == 1)
                {
                    grid[x+1][y] = 2;
                    fresh--;
                    rotten.push({x+1, y});
                };
                if(y < c-1 && grid[x][y+1] == 1)
                {
                    grid[x][y+1] = 2;
                    fresh--;
                    rotten.push({x, y+1});
                };
            }
            if(!rotten.empty()) t++;
        }
        return (fresh == 0) ? t : -1;
    }
```

6.

Medium Level: sum-of-subarray-minimums.
Code:

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

Output: 17

Explanation:

Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2],
[1,2,4], [3,1,2,4].

Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.

Sum is 17.
```

```
int sumSubarrayMins(vector<int>& arr) {


        int n = arr.size(), mod = 1e9+7;
        long sum = 0;

        stack<pair<int,long>> st;


        for(int i=n-1; i>=0; i--){

            while(!st.empty() && arr[i] <= arr[st.top().first]){
                st.pop();
            }
            if(st.empty()){
                st.push({i, (arr[i] * (n-i) % mod)});
            }



            else {
                st.push({i, (arr[i] * (st.top().first - i) % mod + 
st.top().second)});
            }


            sum = (sum + st.top().second) % mod;
        }
        return sum;
    }
```

7.

Medium Level: Evaluate Reverse Polish Notation.

Code:

```
Input: tokens = ["2","1","+","3","*"]

Output: 9

Explanation: ((2 + 1) * 3) = 9
```

```
int evalRPN(vector<string>& tokens) {

        stack<int>s;
        int i=0;
        while(i<tokens.size())
        {
            if(tokens[i]=="+" || tokens[i]=="-" || tokens[i]=="*" || 
tokens[i]=="/" )
            {
                int a=s.top();
                s.pop();
                int b=s.top();
                s.pop();
                if(tokens[i]=="+")
                {
                    s.push(a+b);
                }
                if(tokens[i]=="-")
                {
                    s.push(b-a);
                }
                if(tokens[i]=="*")
                {
                    s.push(a*b);
                }
                if(tokens[i]=="/")
                {
                    int x=b/a;
                    s.push(x);
                }
                i++;
            }
            else
            {
                s.push(stoi(tokens[i]));
                i++;
            }
        }
        return s.top();
    }
```

8. **Medium Level: Circular tour .**

Code:

```
Input:
N = 4
Petrol = 4 6 7 4
Distance = 6 5 3 5
Output: 1
Explanation: There are 4 petrol pumps with
amount of petrol and distance to next
petrol pump value pairs as {4, 6}, {6, 5},
{7, 3} and {4, 5}. The first point from
where truck can make a circular tour is
2nd petrol pump. Output in this case is 1
(index of 2nd petrol pump).
```

```
int tour(petrolPump p[],int n)
    {
       //Your code here
       int totSum=0,currSum=0,j=0;
       for(int i=0;i<n;i++)
       {
           totSum+=p[i].petrol-p[i].distance;
           currSum+=p[i].petrol-p[i].distance;
           if(currSum<0)
           {
               j=i+1;
               currSum=0;
           }
       }
       return totSum<0?-1:j;
    }
```

9. **Medium Level: Flatten Nested List Iterator.**

Code:

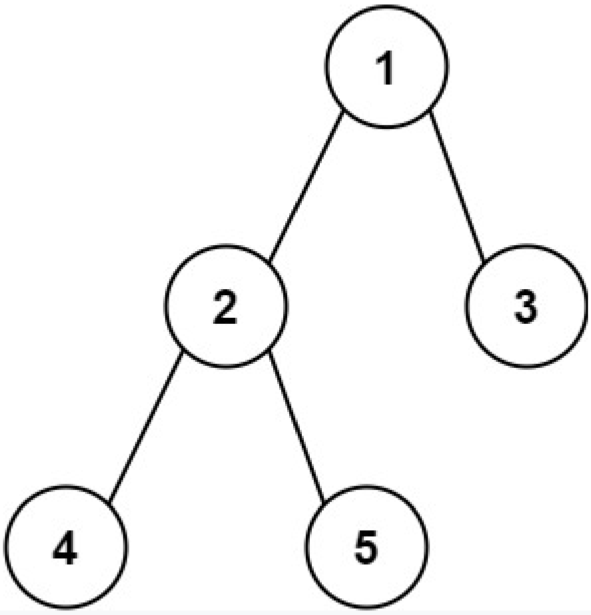
```
Input: nestedList = [[1,1],2,[1,1]]

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

Explanation: By calling next repeatedly until hasNext returns false,
the order of elements returned by next should be: [1,1,2,1,1].
```

```
vector<int> flattenList;
    int index;

    NestedIterator(vector<NestedInteger> &nestedList)
    {
        index = 0;
        doDFS(nestedList);
    }
```

```
int next()
{
    return flattenList[index++];
}

bool hasNext()
{
    if (index < flattenList.size())
        return true;

    return false;
}

void doDFS(vector<NestedInteger> &nestedList)
{

    for (auto nestedInt : nestedList)
    {
        if (nestedInt.isInteger())
        {
            flattenList.push_back(nestedInt.getInteger());
        }
        else
        {
            auto list = nestedInt.getList();
            doDFS(list);
        }
    }
}
```

SNo.	Problem Statement
1.	**Easy Level: Diameter of Binary Tree.**

Code:

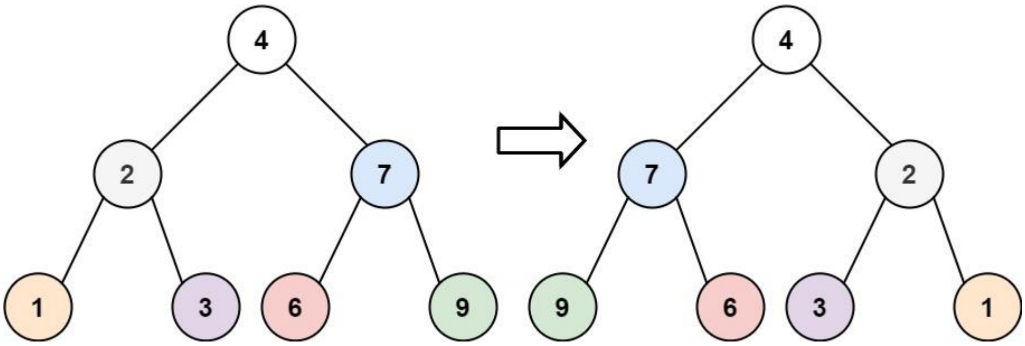


```
Input: root = [1,2,3,4,5]

Output: 3

Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].
```

```
int diameterOfBinaryTree(TreeNode* root) {

        int diameter=0;
        height(root,diameter);
        return diameter;
    }

    int height(TreeNode* node,int &diameter)
    {
        if(node==NULL)
            return 0;
        int lh=height(node->left,diameter);
        int rh=height(node->right,diameter);
        diameter=max(diameter,lh+rh);
        return 1+max(lh,rh);
    }
```

2. **Easy Level: Invert Binary Tree.**

Code:

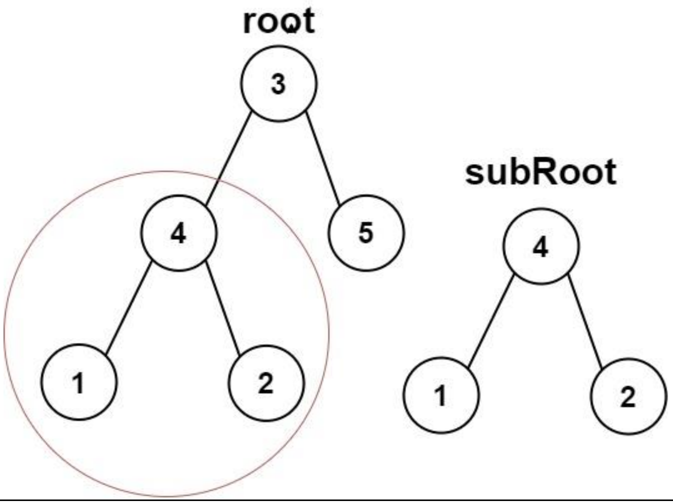


```
Input: root = [4,2,7,1,3,6,9]

Output: [4,7,2,9,6,3,1]
```

```
TreeNode* invertTree(TreeNode* root)
 {
    if(root==NULL)
    return 0;
    TreeNode* left=invertTree(root->left);
    TreeNode* right=invertTree(root->right);
    root->left=right;
    root->right=left;
    return root;
 }
```

3. **Easy Level: Subtree of Another Tree.**

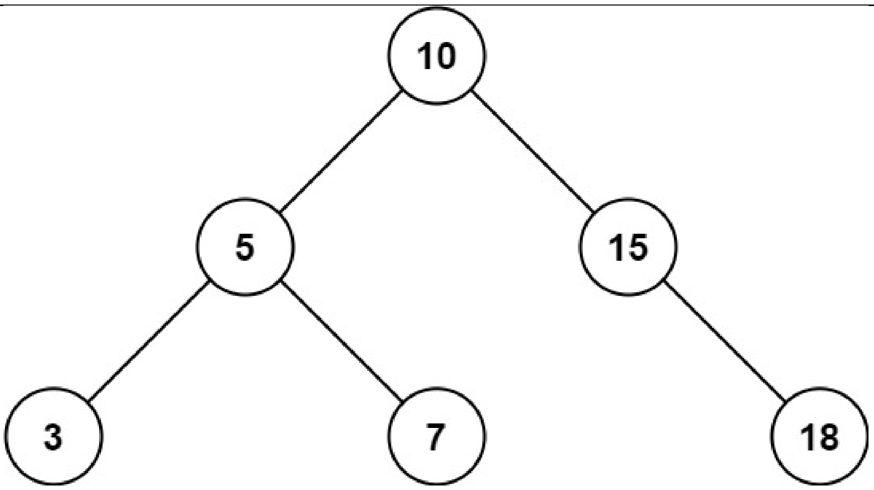
Code:



```
Input: root = [3,4,5,1,2], subRoot = [4,1,2]

Output: true
```

```
bool dfs(TreeNode* root1,TreeNode* root2)
    {
        if(!root1 and !root2)
            return true;
        if(!root1 || !root2)
            return false;
        if(root1->val!=root2->val)
            return false;
        return dfs(root1->left,root2->left) and dfs(root1->right,root2->right);

    }
    bool isSubtree(TreeNode* root, TreeNode* subRoot) {
       if(!root)
           return false;
        if(root->val==subRoot->val)
        {
            if(dfs(root,subRoot))
                return true;
        }
        return isSubtree(root->left,subRoot)||isSubtree(root->right,subRoot);
    }
```

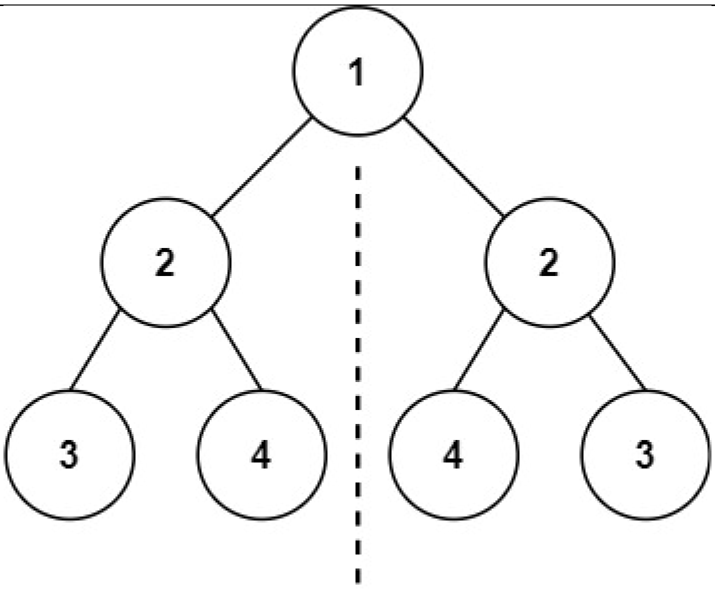
4. **Easy Level: Range Sum of BST.**
Code:



```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15

Output: 32

Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 =
32.
```

```
int rangeSumBST(TreeNode* root, int low, int high)
 {
    if(root==NULL)
    return 0;
    if(root->val>=low and root->val<=high)
    {
        return rangeSumBST(root->left,low,root->val)+rangeSumBST(root->right,root->val,high)+root->val;
    }
    else
    {
        return rangeSumBST(root->left,low,high)+rangeSumBST(root->right,low,high);
    }
    return 0;
 }
```

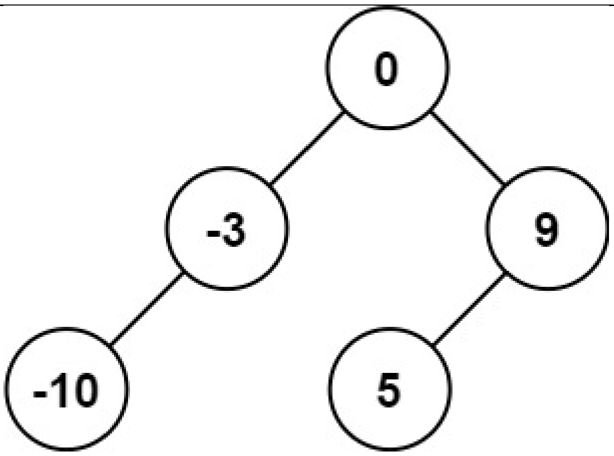
5. **Easy Level: Symmetric Tree.**
Code:



```
Input: root = [1,2,2,3,4,4,3]

Output: true
```

```
bool dfs(TreeNode* root1, TreeNode* root2)
{
   if(root1==NULL and root2==NULL)
   {
      return true;
   }
   if(root1==NULL or root2==NULL)
   {
      return false;
   }
   return ((root1->val==root2->val) and dfs(root1->left,root2->right)
and dfs(root1->right,root2->left));
}
bool isSymmetric(TreeNode* root) {

   return dfs(root->left,root->right);
}
```

6. **Easy Level: Convert Sorted Array to Binary Search Tree.**
Code:



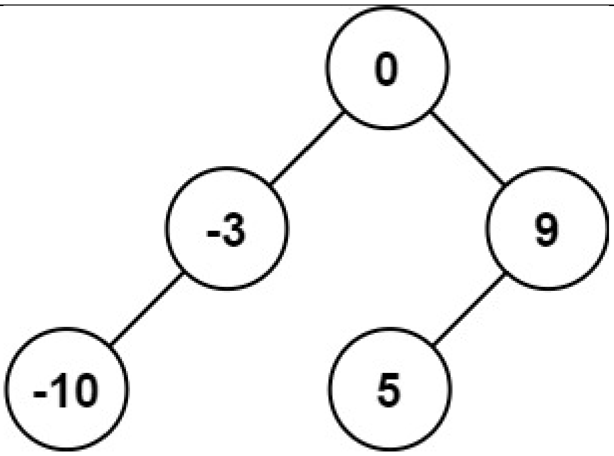
```
Input: nums = [-10,-3,0,5,9]

Output: [0,-3,9,-10,null,5]

Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```

```
TreeNode* binaryST(int s,int e,vector<int>nums)
    {
         if(s>e)
             return NULL;
        if(s==e)
        {
            return new TreeNode(nums[e]);
        }
         int mid=(e+s)/2;
        TreeNode* root=new TreeNode(nums[mid]);
        root->left=binaryST(s,mid-1,nums);
        root->right=binaryST(mid+1,e,nums);
        return root;
    }
    TreeNode* sortedArrayToBST(vector<int>& nums) {

       return binaryST(0,nums.size()-1,nums);
    }
```

7.

Easy Level: Merge Two Binary Trees.

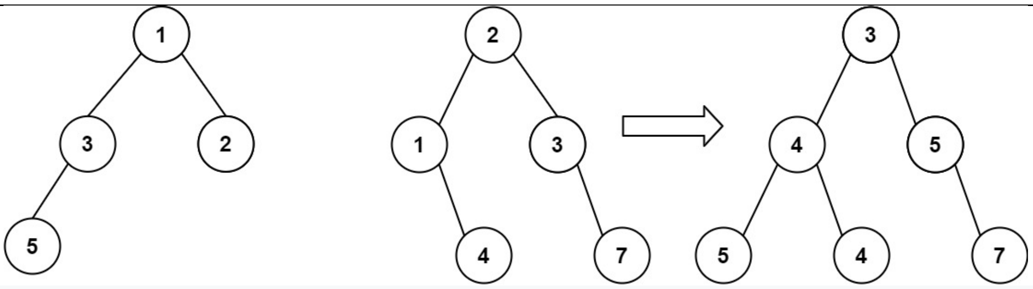
Code:



```
Input: nums = [-10,-3,0,5,9]

Output: [0,-3,9,-10,null,5]

Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```

```
TreeNode* mergeTrees(TreeNode* root1, TreeNode* root2) {

      if(root1==NULL )
         return root2;
      if(root2==NULL)
         return root1;
      root1->val=root1->val+root2->val;
      root1->left=mergeTrees(root1->left,root2->left);
      root1->right=mergeTrees(root1->right,root2->right);
      return root1;



   }
```

8. **Easy Level: Maximum Depth of Binary Tree.**
Code:



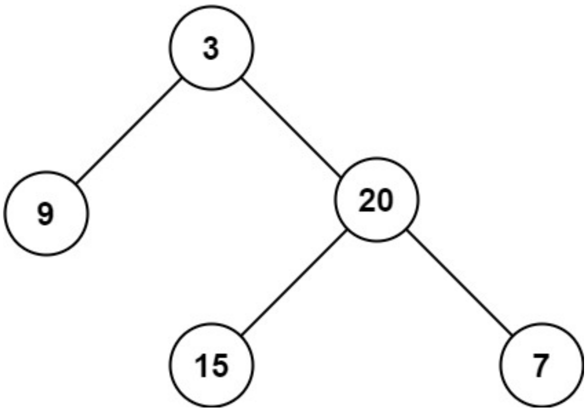
```
Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]

Output: [3,4,5,5,4,null,7]
```

```
int maxDepth(TreeNode* root) {

        if(root==NULL)
            return 0;
        return max(maxDepth(root->left),maxDepth(root->right))+1;
    }
```

9.

Easy Level: Same Tree.

Code:



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

Output: 3
```

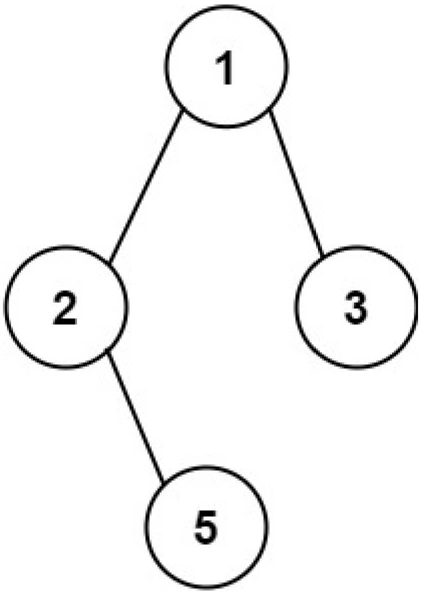
```
bool isSameTree(TreeNode* p, TreeNode* q) {

        if(p==NULL && q==NULL)
            return true;
        if(q==NULL || p==NULL)
```

```
          return false;
       if(p->val!=q->val)
          return false;
       return isSameTree(p->right,q->right) and isSameTree(p->left,q->left);


   }
```

10.

Easy Level: Lowest Common Ancestor of a Binary Search Tree.

Code:



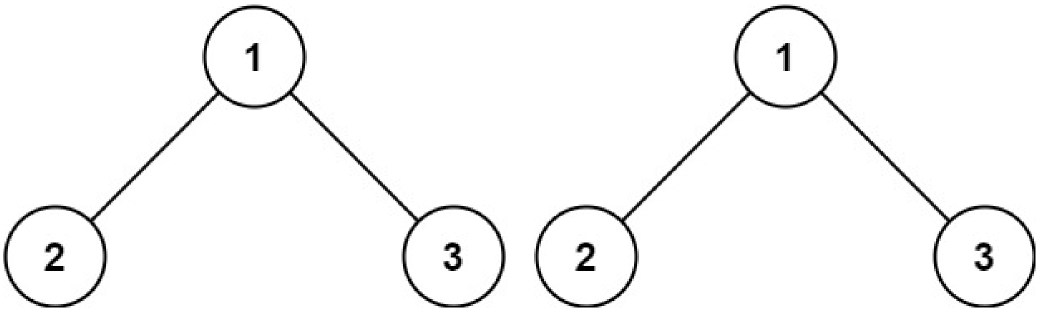
```
Input: root = [1,2,3,null,5]

Output: ["1->2->5","1->3"]
```

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

       if(root==NULL || p==root || q==root)
       {
          return root;
       }
     TreeNode* left=lowestCommonAncestor(root->left,p,q);
      TreeNode* right=lowestCommonAncestor(root->right,p,q);
```

```
        if(left==NULL)
            return right;
        if(right==NULL)
            return left;
        else
            return root;
    }
```

11. **Easy Level: Path Sum.**

Code:

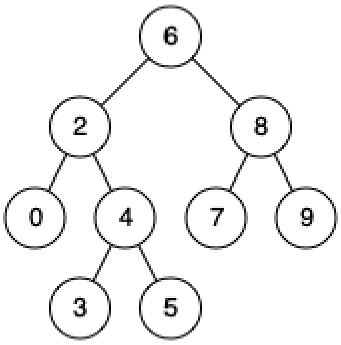


```
Input: p = [1,2,3], q = [1,2,3]

Output: true
```

```
bool hasPathSum(TreeNode* root, int targetSum) {
        if(root==NULL)
            return false;
        if(root->left==NULL and root->right==NULL)
        {
            return (targetSum - root->val==0);
        }
        return (hasPathSum(root->right, targetSum - root->val))||hasPathSum(root->left, targetSum - root->val));
    }
```

12. **Easy Level: Minimum Absolute Difference in BST.**

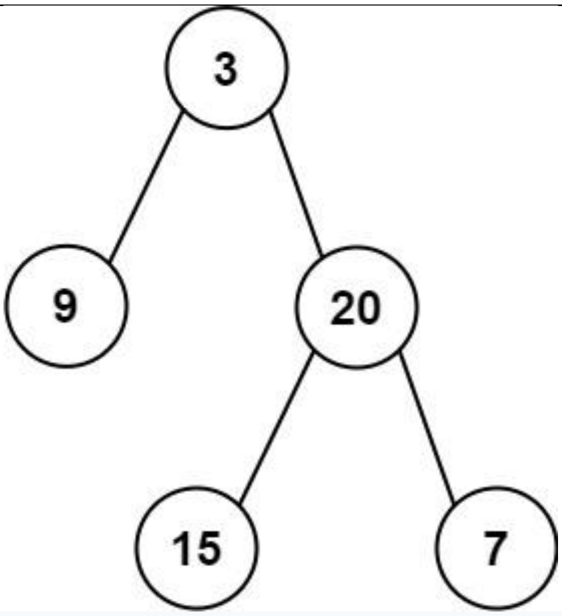
Code:



```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

Output: 6

Explanation: The LCA of nodes 2 and 8 is 6.
```

```
int diff = INT_MAX;
     TreeNode* prev = NULL;
    void dfs(TreeNode* root)
    {
        if(root==NULL)
            return;
        dfs(root->left);
        if(prev)
            diff = min(diff, abs(prev->val-root->val));
            prev = root;
        dfs(root->right);

    }
    int getMinimumDifference(TreeNode* root) {
        if(root==NULL)
            return 0;
        dfs(root);
        return diff;
    }
```

13. **Easy Level: Sum of Left Leaves.**
Code:

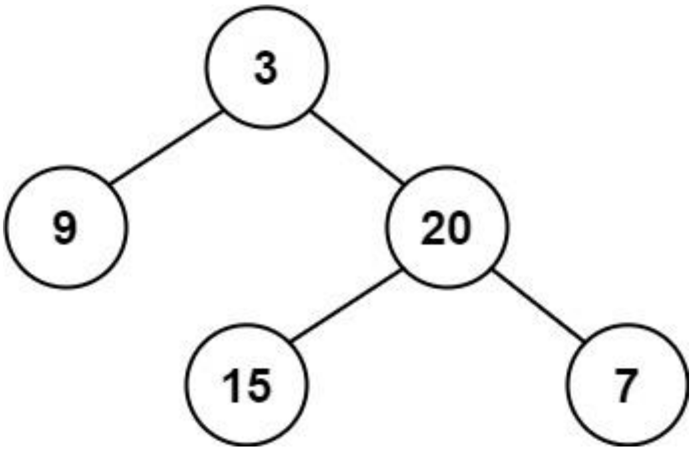


```
Input: root = [3,9,20,null,null,15,7]

Output: 24

Explanation: There are two left leaves in the binary tree, with values 9
and 15 respectively.
```

```
int sumOfLeftLeaves(TreeNode *root)
{
    int sum=0;
    std::queue<TreeNode* >q ;
    while(!q.empty())
    {
        q.push(root);
        TreeNode* node=q.front();
        TreeNode* p;
        if(node->left)
        {
            p=node->left;
            if(p->left==NULL and p->right==NULL)
            {
                sum+=node->left->val;
            }
        }
        q.pop();
        if(node->left)
```

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

14. **Easy Level: Balanced Binary Tree.**
Code:



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

Output: true
```

```
bool isBalanced(TreeNode* root) {

        return height(root)!=-1;

    }

    int height(TreeNode* root)
    {
        if(root==NULL)
            return 0;
        int leftHeight=height(root->left);
        if(leftHeight==-1)
            return -1;
        int rightHeight=height(root->right);
        if(rightHeight==-1)
            return-1;
        if(abs(leftHeight - rightHeight)>1)
            return -1;
        return max(leftHeight,rightHeight)+1;
    }
```

15. **Easy Level: Predecessor and Successor.**

Code:

Input:

```
2
6
50 30 L 30 20 L 30 40 R 50 70 R 70 60 L 70 80 R
65
6
50 30 L 30 20 L 30 40 R 50 70 R 70 60 L 70 80 R
100
```

Output:

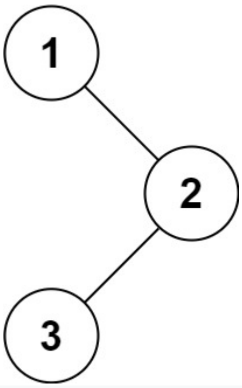
```
60 70
80 -1
```

```
void inorder_successor(Node* root, Node* &succ, int key)
{
    while(root!=NULL)
    {
        if(root->key<=key)
        {
            root=root->right;
        }
        else if(root->key>key)
        {
            succ=root;
            root=root->left;
        }
    }
}

void inorder_predecessor(Node* root, Node* &pred, int key)
{
    while(root!=NULL)
    {
        if(root->key>=key)
        {
            root=root->left;
        }
        else if(root->key<key)
        {
            pred=root;
            root=root->right;
        }
    }
}

void findPreSuc(Node* root, Node*& pre, Node*& suc, int key)
{

// Your code goes here

    inorder_successor(root,suc,key);
    inorder_predecessor(root,pre,key);

}
```

16. **Easy Level: Binary Tree Inorder Traversal.**

Code:

```
Input: root = [1,null,2,3]

Output: [1,3,2]
```



```
Input: root = [1,null,2,3]
```

```
Output: [1,3,2]
```

```
void tree(TreeNode* root,vector<int>&v)
    {
        if(root==NULL)
            return;
        tree(root->left,v);
        v.push_back(root->val);
        tree(root->right,v);

    }
    vector<int> inorderTraversal(TreeNode* root) {

        vector<int>v;
        tree(root,v);
        return v;
    }
```

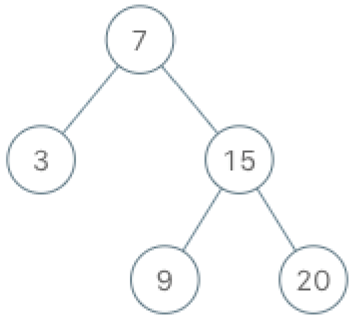
17. **Easy Level: Check whether BST contains Dead End.**

Code:

```
int c=0;
bool fun(Node* root,int lb,int ub)
{
    if(root==0&& abs(lb-ub)==1)
    return 1;
    if(root==0)
    return 0;

    bool l=fun(root->left,lb,root->data);
    bool r=fun(root->right,root->data,ub);

    if(l&&r)
    c=1;
    return 0;
}
bool isDeadEnd(Node *root)
{
    //Your code here
    c=0;
    fun(root,0,INT_MAX);
```

	return c;
	}

SNo.	Problem Statement
1.	**Medium Level: Binary Search Tree Iterator.** **Code:**

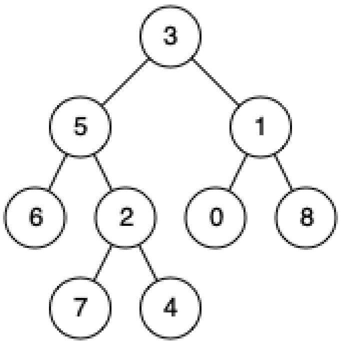


```
Input

["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next",
"hasNext", "next", "hasNext"]

[[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []]

Output

[null, 3, 7, true, 9, true, 15, true, 20, false]
```

```
class BSTIterator {
    stack<TreeNode*>s;
public:
    BSTIterator(TreeNode* root) {
        pushAll(root);
    }

    int next() {

        TreeNode* tmpNode=s.top();
        s.pop();
        pushAll(tmpNode->right);
        return tmpNode->val;
    }

    bool hasNext() {
        return !s.empty();
    }
    private:
```

```
    void pushAll(TreeNode* node)
    {
        for(;node!=NULL;s.push(node),node=node->left);
    }
};
```

2. **Medium Level: Lowest Common Ancestor of a Binary Tree.**
Code:

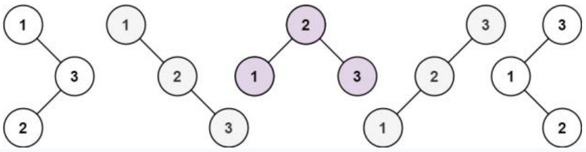


```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

Output: 3

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

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


        if(root==NULL || root==p || root==q)
        {
            return root;
        }
        TreeNode* left=lowestCommonAncestor(root->left,p,q);
        TreeNode* right=lowestCommonAncestor(root->right,p,q);

        if(left==NULL)
            return right;
        else if(right==NULL)
            return left;
        else
            return root;
    }
```

3. **Medium Level: Unique Binary Search Trees II.**

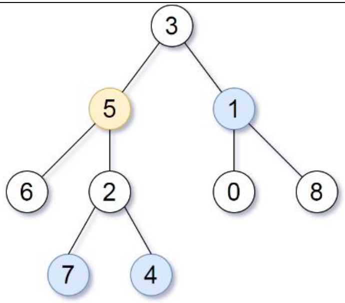
Code:



```
Input: n = 3

Output:
[[1,null,2,null,3],[1,null,3,2],[2,1,3],[3,1,null,null,2],[3,2,null,1]]
```

```
vector<TreeNode*> generateTrees(int n) {

        return solve(1,n);

    }
    vector<TreeNode*>solve(int s,int e)
    {
        if(s>e)
        {
            return vector<TreeNode*>(1,NULL);
        }
        vector<TreeNode*>res;
        for(int i=s;i<=e;i++)
        {
            vector<TreeNode*>left=solve(s,i-1);
            vector<TreeNode*>right=solve(i+1,e);
            for(int j=0;j<left.size();j++)
            {
                for(int k=0;k<right.size();k++)
                {
                    res.push_back(new TreeNode(i,left[j],right[k]));
                }
            }
        }
        return res;
    }
```

4. **Medium Level: All Nodes Distance K in Binary Tree.**
Code:



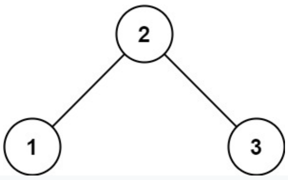
```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, k = 2

Output: [7,4,1]

Explanation: The nodes that are a distance 2 from the target node (with
value 5) have values 7, 4, and 1.
```

```
vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {

     vector<int>ans;
    if(root == nullptr) return ans;
    unordered_map<TreeNode*, TreeNode*>mp;
    queue<TreeNode *>q;
    mp[root] = nullptr;
    q.push(root);
    while(!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        if(node->left) {
            mp[node->left] = node;
            q.push(node->left);
        }
        if(node->right) {
            mp[node->right] = node;
            q.push(node->right);
        }
    }
    unordered_map<TreeNode*, bool>visited;
    q.push(target);
    visited[target] = true;
    while(!q.empty()) {
        int size = q.size();
        if(k == 0) {
            for(int i = 0; i < size; i++) {
                TreeNode* node = q.front();
```

```
                    q.pop();
                    ans.push_back(node->val);
                }
            } else {
                for(int i = 0; i < size; i++) {
                    TreeNode* node = q.front();
                    q.pop();
                    if(node->left && !visited[node->left]) {
                        q.push(node->left);
                        visited[node->left] = true;
                    }
                    if(node->right && !visited[node->right]) {
                        q.push(node->right);
                        visited[node->right] = true;
                    }
                    if(mp[node] && !visited[mp[node]]) {
                        q.push(mp[node]);
                        visited[mp[node]] = true;
                    }
                }
                k--;
            }
        }
        return ans;
    }
```

5.

Medium Level: Validate Binary Search Tree.
Code:

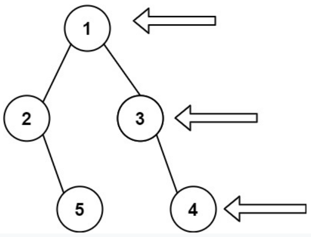


```
Input: root = [2,1,3]

Output: true
```

```
void inOrder(TreeNode* root,vector<int> &v)
    {
                    //Left SubTree
        if(root->left != NULL)
            inOrder(root->left,v);
                    //root
        v.push_back(root->val);
                    //right subtree
        if(root->right != NULL)
            inOrder(root->right,v);
    }
    bool isValidBST(TreeNode* root) {
        vector<int> v;
        inOrder(root,v);
                    //if array is sorted or not
        for(int i=0;i<v.size()-1;i++)
        {
            if(v[i]<v[i+1])
                continue;
            else
                return false;
        }
        return true;
}
```

6. **Medium Level: Binary Tree Right Side View.**

Code:



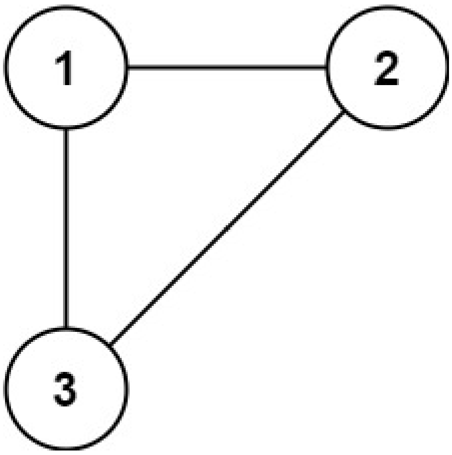
```
Input: root = [1,2,3,null,5,null,4]

Output: [1,3,4]
```

```
vector<int> rightSideView(TreeNode* root) {

        vector<int>level;
        if(root==NULL)
        {
            return {};
        }
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty())
        {
```

```
        int n=q.size();
        for(int i=1;i<=n;i++)
        {
            TreeNode* node=q.front();
            q.pop();
            if(i==n)
            {
                level.push_back(node->val);
            }
            if(node->left!=NULL){
                q.push(node->left);
            }
            if(node->right!=NULL)
            {
                q.push(node->right);
            }
        }
    }
    return level;
}
```

7.

Medium Level: Redundant Connection.
Code:

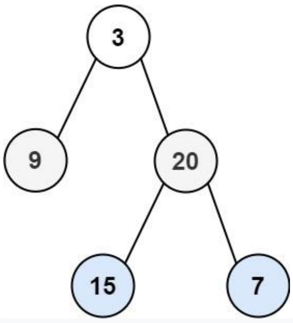


```
Input: edges = [[1,2],[1,3],[2,3]]

Output: [2,3]
```

```
int Root(int x,vector<int> &parent )
{
    if(parent[x] == -1)
        return x;
   return Root( parent[x],parent);
}

vector<int> findRedundantConnection(vector<vector<int>>& edges)
{
   int n = edges.size();
    vector<int> parent(n+1,-1);

    for( auto x:edges)
    {
        if( Root(x[0],parent) == Root( x[1],parent) )
            return x;

        int t = Root(x[0],parent);
        parent[t] = Root(x[1],parent);
    }
    return {-1};
}
```

8. **Medium Level: Binary Tree Level Order Traversal.**

Code:



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

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

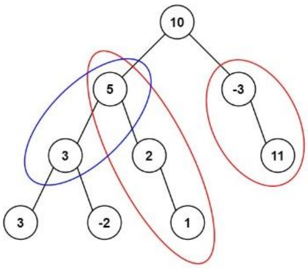
```
vector<vector<int>> levelOrder(TreeNode* root)
 {
     vector<vector<int>>res;
     if(root==NULL)
     return res;
     std::queue<TreeNode*>q ;
     q.push(root);
     while(!q.empty())
```

```
    {
        vector<int>level;
        int n=q.size();
        for(int i=0;i<n;i++)
        {
            TreeNode* node=q.front();
            q.pop();
            if(node->left!=NULL)
            q.push(node->left);
            if(node->right!=NULL)
            q.push(node->right);
            level.push_back(level);
        }
        ans.push_back(level);
    }
    return ans;
}
```

9.

Medium Level: Path Sum III.

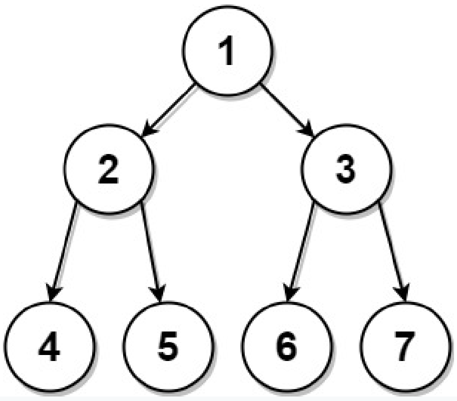
Code:



```
void dfs(TreeNode* root,int sum)
 {
    if(root==NULL)
          return;
       sum=sum-root->val;
       if(sum==0)
       {
          c++;
       }
       dfs(root->left,sum);
       dfs(root->right,sum);
 }
 int pathSum(TreeNode* root, int targetSum)
 {
    if(root==NULL)
    return 0;
    dfs(root,targetSum);
    pathSum(root->left,targetSum);
    pathSum(root->right,targetSum);
    return c;
 }
```

10. **Medium Level: Construct Binary Tree from Preorder and Postorder Traversal.**

Code:



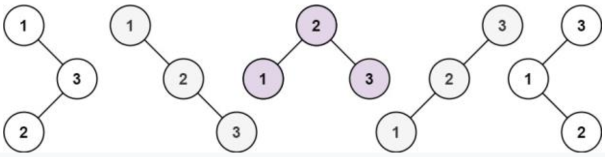
```
Input: preorder = [1,2,4,5,3,6,7], postorder = [4,5,2,6,7,3,1]

Output: [1,2,3,4,5,6,7]
```

```
   int pre_index=0, post_index = 0;
   TreeNode* constructFromPrePost(vector<int>& preorder,
vector<int>& postorder) {
       TreeNode* root=new TreeNode(preorder[pre_index++]);
       if(root->val!=postorder[post_index])
       {
           root->left=constructFromPrePost(preorder,postorder);
       }

      if(root->val!=postorder[post_index])
       {
           root->right=constructFromPrePost(preorder,postorder);
       }
       post_index++;
       return root;
   }
```

11. **Medium Level: Unique Binary Search Trees.**

<table>
<tr>
<td></td>
<td>

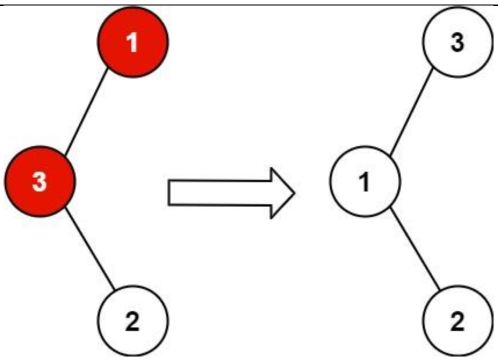
Code:



```
Input: n = 3

Output: 5
```

```
int unique(vector<int>&dp,int n)
    {
        if(n==1)
            return dp[n]=1;
        if(n==2)
            return dp[n]=2;
        if(dp[n]!=-1)
            return dp[n];
        int res=0;
        for(int i=1;i<=n;i++)
        {
            res += (unique(dp,i-1)*unique(dp,n-i));
        }
        return dp[n]=res;
    }
    int numTrees(int n) {

        vector<int>dp(n+1,-1);
        dp[0]=1;
        return unique(dp,n);



    }
```

</td>
</tr>
<tr>
<td><b>12.</b></td>
<td><b>Medium Level: Recover Binary Search Tree.</b>
<b>Code:</b></td>
</tr>
</table>



```
vector<TreeNode*>v;
    void inorder(TreeNode* root)
    {
        if(root==NULL)
            return;
        inorder(root->left);
        v.push_back(root);
        inorder(root->right);
    }
    void recoverTree(TreeNode* root) {



        inorder(root);
        int i,j;
        int n=v.size();
        for(int i=0;i<n-1;i++)
        {
            for(int j=0;j<n-i-1;j++)
            {
                if(v[j]->val>v[j+1]->val)
                    swap(v[j]->val,v[j+1]->val);
            }
        }

    }
```

13. **Medium Level: Populating Next Right Pointers in Each Node.**
Code:

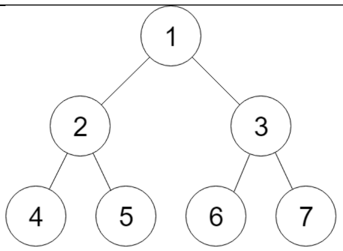


Figure A

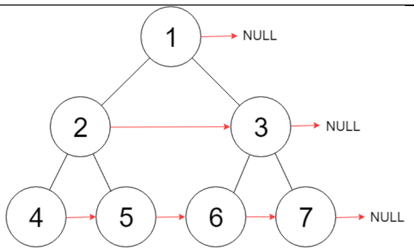


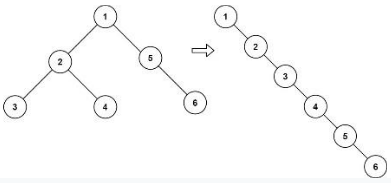
Figure B

```
Input: root = [1,2,3,4,5,6,7]

Output: [1,#,2,3,#,4,5,6,7,#]
```

```
  Node* connect(Node* root) {

       if(root==NULL)
           return NULL;
       if(root->left!=NULL)
       {
           root->left->next=root->right;
           root->left=connect(root->left);
       }
       if(root->right)
       {
           if(!root->next)
               root->right->next=NULL;
           else
               root->right->next=root->next->left;
           root->right=connect(root->right);
       }
       return root;
}
```

14. **Medium Level: Flatten Binary Tree to Linked List.**

Code:



```
Input: root = [1,2,5,3,4,null,6]

Output: [1,null,2,null,3,null,4,null,5,null,6]
```

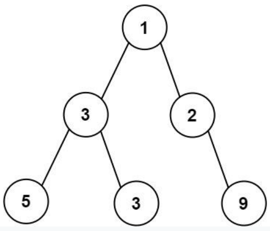
```
void flatten(TreeNode* root) {
  stack<TreeNode*>st;
```

```
    if(root==NULL)
        return;
    st.push(root);
    while(!st.empty())
    {
        TreeNode* cur=st.top();
        st.pop();
        if(cur->right!=NULL)
        {
            st.push(cur->right);
        }
        if(cur->left!=NULL)
        {
            st.push(cur->left);
        }
        if(!st.empty())
        {
            cur->right=st.top();
        }
        cur->left=NULL;
    }
```

15.

Medium Level: Maximum Width of Binary Tree.

Code:



Input: root = [1,3,2,5,3,null,9]

Output: 4

Explanation: The maximum width exists in the third level with length 4 (5,3,null,9).

```
int widthOfBinaryTree(TreeNode* root) {


    if(root==NULL)
        return 0;
    int res=0;
    queue<pair<TreeNode*,int>>q;
    q.push({root,0});
    while(!q.empty())
    {
        int n=q.size();
        int min=q.front().second;
        int first,last;
        for(int i=0;i<n;i++)
        {
            int cur=q.front().second-min;//subtract to prevent integer overflow
            TreeNode* node=q.front().first;
            q.pop();
            if(i==0)
                first=cur;
            if(i==n-1)
                last=cur;
            if(node->left)
            {
                q.push({node->left,cur*2+1});
            }
            if(node->right)
            {
                q.push({node->right,cur*2+2});
            }

        }
        res=max(res,last-first+1);
    }
    return res;
}
```

16.

Medium Level: Min distance between two given nodes of a Binary Tree .

Code:

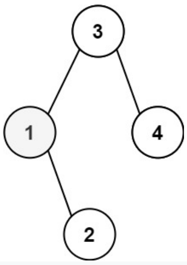
```
Input:
         1
       /   \
      2     3
a = 2, b = 3
Output: 2
```

```
Node* func(Node *root,int a,int b){
        if(root==NULL)
            return NULL;
        if(root->data==a or root->data==b)
            return root;

        Node *left=func(root->left,a,b);
```

```
    Node *right=func(root->right,a,b);

    if(left!=NULL and right!=NULL)
        return root;
    else if(left!=NULL)
        return left;
    else
        return right;
}

  int fun(Node *root,int x,int d){
    if(root==NULL)
        return INT_MAX;

    if(root->data==x)
        return d;
    int p=fun(root->left,x,d+1);
    int q=fun(root->right,x,d+1);
    return min(p,q);
}

int findDist(Node* root, int a, int b) {
    // Your code here
    Node *lca=func(root,a,b);
    int p=fun(lca,a,0);
    int q=fun(lca,b,0);
    return p+q;
 }
```

17. **Medium Level: Kth Smallest Element in a BST.**

Code:

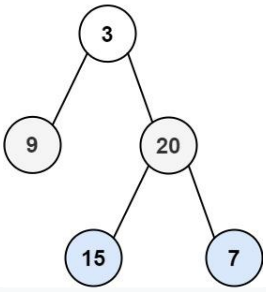


```
Input: root = [3,1,4,null,2], k = 1

Output: 1
```

```
void inorder(TreeNode* root,vector<int>&v)
   {
      if(root==NULL)
      return ;
      inorder(root->left,v);
      v.push_back(root->val);
      inorder(root->right,v);
   }
public:
   int kthSmallest(TreeNode* root, int k)
   {
      vector<int>v;
      inorder(root,v);
      sort(v.begin(),v.end(),greater<int>());
      int ans=v.size()-k;
      return v[ans];


   }
```

18. **Medium Level: Binary Tree Zigzag Level Order Traversal.**

Code:



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

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

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

      vector<vector<int>>res;
      if(root==NULL)
         return res;
      queue<TreeNode*>q;

      q.push(root);
```

```
        int flag=0;//left to right

        while(!q.empty())
        {
            int n=q.size();
            vector<int>zig;
            for(int i=0;i<n;i++)
            {
                TreeNode* node=q.front();
                q.pop();
                if(node->left!=NULL)
                {
                    q.push(node->left);
                }
                 if(node->right!=NULL)
                {
                    q.push(node->right);
                }
                zig.push_back(node->val);
            }

        if(flag==1)
        {
            reverse(zig.begin(),zig.end());
            flag=0;
        }
        else
        {
            flag=1;
        }

        res.push_back(zig);
    }
        return res;
    }
```

19.	**Medium Level: Count BST nodes that lie in a given range .** **Code:** `Input:`

```
       10
      /  \
     5    50
    /    /  \
   1    40  100
l = 5, h = 45
```

```
Output: 3
Explanation: 5 10 40 are the node in the
range
```

```
void solve(Node *root, int l, int h,int &c)
   {
       if(root==NULL)
       return;
       if(root->data<=h and root->data>=l)
       c++;
       solve(root->left,l,h,c);
       solve(root->right,l,h,c);
   }
   int getCount(Node *root, int l, int h)
   {
     // your code goes here
     int c=0;
     solve(root,l,h,c);
     return c;
   }
```

20. **Medium Level: Preorder to Postorder .**
Code:

```
Input:
N = 5
arr[]  = {40,30,35,80,100}
Output: 35 30 100 80 40
Explanation: PreOrder: 40 30 35 80 100
```

```
InOrder: 30 35 40 80 100

Therefore, the BST will be:

              40

           /      \

         30        80

           \         \

           35       100

Hence, the postOrder traversal will

be: 35 30 100 80 40
```

```
Node* pre_post(int start,int end, int pre[]){
      if(start>end){
         return NULL;;
      }
      Node* root=newNode(pre[start]);
      int mid=start+1;
      while(mid<=end and pre[mid]<pre[start]){
         mid++;
      }
      mid--;
      root->left=pre_post(start+1,mid,pre);
      root->right=pre_post(mid+1,end,pre);
      return root;
   }
   Node* post_order(int pre[], int size)
   {
      //code here
       return pre_post(0,size-1,pre);
   }
```

What is the difference between priority queue and heap?

The priority queue is the queue data structure and the heap is the tree data structure that operates and organizes data. The priority queue is based on a queue data structure working as a queue with a priority function. The heap is a tree data structure uses for sorting data in a specific order using an algorithm.

SNo. **Problem Statement**

1. **Easy/Medium Level: Top K Frequent Elements.**
Code:

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

Output: [1,2]
```

```
vector<int> topKFrequent(vector<int>& nums, int k) {

        vector<int>v;
        priority_queue<pair<int,int>>pq;
         unordered_map<int,int>mp;

        for(int i=0;i<nums.size();i++)
        {
            mp[nums[i]]++;
        }
        for(auto it:mp)
        {
            pq.push({it.second,it.first});
        }
        //this is for push the element in vector v and remove the element
from queue.
        while(k--)
        {
            v.push_back(pq.top().second);
            pq.pop();
        }
        return v;
    }
```

2. **Easy/Medium-Level: Kth Largest Element in an Array.**
Code:

```
Input: nums = [3,2,1,5,6,4], k = 2

Output: 5
```

```
int findKthLargest(vector<int>& nums, int k) {

        sort(nums.begin(),nums.end());
       // return (nums[k-1]);
        int n=nums.size();
        return (nums[n-k]);
    }
```

3. **Easy/Medium -Level: Ugly Number II.**
Code:

```
Input: n = 10

Output: 12

Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the
first 10 ugly numbers.
```

```
int nthUglyNumber(int n) {


        int dp[n];
        int a2=0,a3=0,a5=0;
        dp[0]=1;
        for(int i=1;i<n;i++)
        {
            dp[i] = min(dp[a2]*2,min(dp[a3]*3,dp[a5]*5));
            if(dp[i]==dp[a2]*2)
                a2++;
            if(dp[i]==dp[a3]*3)
                a3++;
            if(dp[i]==dp[a5]*5)
                a5++;
        }
        return dp[n-1];
    }
```

4. **Easy/Medium-Level: Furthest Building You Can Reach.**
Code:

```
Input: heights = [4,2,7,6,9,14,12], bricks = 5, ladders = 1

Output: 4
```

```
int furthestBuilding(vector<int>& heights, int bricks, int ladders) {

        int n=heights.size();
        priority_queue<int>p;
        for(int i=0;i<n-1;i++)
        {
            if(heights[i+1]>heights[i])
            {
                int gap=heights[i+1] - heights[i];
                bricks -= gap;
                p.emplace(gap);
                if(bricks<0)
                {
                    if(ladders--)
                    {
                        bricks+=p.top();
                        p.pop();
                    }
                    else return i;
                }
            }
        }
        return n-1;
    }
```

5.

Easy/Medium-Level: Kth Smallest Element in a Sorted Matrix.

Code:

```
Input: matrix = [[1,5,9],[10,11,13],[12,13,15]], k = 8
```

```
Output: 13

Explanation: The elements in the matrix are [1,5,9,10,11,12,13,13,15],
and the 8th smallest number is 13
```

```
//using max-heap
        priority_queue<int>pq;
        for(int i=0;i<matrix.size();i++)
        {
            for(int j=0;j<matrix.size();j++)
            {
                pq.push(matrix[i][j]);
                if(pq.size()>k)
                {
                    pq.pop();
                }
            }
        }
        return pq.top();
    }
```

6.

Easy/Medium Level: Reorganize String.

Code:

```
Input: s = "aab"

Output: "aba"
```

```
string reorganizeString(string s) {

       unordered_map<char,int>m;
        for(auto it:s)
            m[it]++;
        priority_queue<pair<int,char>>pq;
        string res="";
        for(auto it:m)
            pq.push({it.second,it.first});

        while(pq.size() > 1)
        {
            auto top1 = pq.top();
            pq.pop();
            auto top2 = pq.top();
            pq.pop();

            res+=top1.second;
            res+=top2.second;

            top1.first--;
            top2.first--;

            if (top1.first > 0)
                pq.push(top1);
            if (top2.first > 0)
                pq.push(top2);
        }
         if (!pq.empty())
            if (pq.top().first > 1)
                return "";
            else
                res += pq.top().second;

        return res;
    }
```

7.

Easy/Medium Level: Find the Most Competitive Subsequence.

Code:

```
Input: nums = [3,5,2,6], k = 2

Output: [2,6]

Explanation: Among the set of every possible subsequence: {[3,5], [3,2],
[3,6], [5,2], [5,6], [2,6]}, [2,6] is the most competitive.
```

```
vector<int> mostCompetitive(vector<int>& nums, int k) {


        //using vector as a stack and follow lexicographical order

        vector<int>res;
        for(int i=0;i<nums.size();i++)
        {
            while(res.size()>0 and res.back()>nums[i] and (k-(res.size()-1))<=(nums.size()-i))
                res.pop_back();
```

```
           if(res.size()<k)//this condition is when you are not able to push and pop like 1,2,3,4 so 4 doesnt pop 3
               res.push_back(nums[i]);

       }
       return res;
    }
```

8. **Easy/Medium Level: Smallest Positive missing number.**
Code:

```
Input:
N = 5
arr[] = {1,2,3,4,5}
Output: 6
Explanation: Smallest positive missing
number is 6.
```

```
int missingNumber(int arr[], int n)
    {
        // Your code here
         unordered_map<int,int>mapping;

      for(int i = 0; i < n; i++){
          mapping[arr[i]]++;
      }


      for(int i = 1; i <= n+1; i++){
          if(mapping[i] == 0){
              return i;
          }
      }
    }
```

9. **Easy/Medium Level: Largest subarray with 0 sum .**
Code:

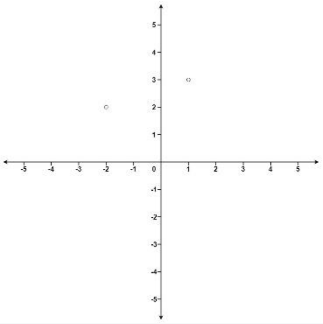
```
Input:
```

```
N = 8
A[] = {15,-2,2,-8,1,7,10,23}
Output: 5
Explanation: The largest subarray with
sum 0 will be -2 2 -8 1 7.
```

```
int maxLen(vector<int>&A, int n)
    {
        // Your code here
        
        unordered_map<int,int>m;
        int maxi=0;
        int sum=0;
        for(int i=0;i<n;i++)
        {
            sum=sum+A[i];
            if(sum==0)
            {
                maxi=i+1;
            }
            else
            {
                if(m.find(sum)!=m.end())
                {
                    maxi=max(maxi,i-m[sum]);//logic->store sum like first
0+15 =15 then
                                            //15-2=13 then 13+2=15 so 15 is already
preseent in
                                             //map so find index where it is find prev-
curr =max update
                }
                else
                {
                    m[sum]=i;
                }
            }
        }
        return maxi;
    }
```

10.

Easy/Medium Level: K Closest Points to Origin.
Code:



```
Input: points = [[1,3],[-2,2]], k = 1

Output: [[-2,2]]

Explanation:

The distance between (1, 3) and the origin is sqrt(10).

The distance between (-2, 2) and the origin is sqrt(8).

Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.

We only want the closest k = 1 points from the origin, so the answer is
just [[-2,2]].
```

```
vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {

         priority_queue<pair<int,pair<int,int>>>pq;

        for(auto i : points){
            pq.push(make_pair(i[0]*i[0]+i[1]*i[1], make_pair(i[0],i[1])));
            if(pq.size()>k){
                pq.pop();
            }
        }

        vector<vector<int>> ans;
        while(!pq.empty()){
            vector<int> temp;
            temp.push_back(pq.top().second.first);
            temp.push_back(pq.top().second.second);
            ans.push_back(temp);
            pq.pop();
        }

        return ans;

    }
```

SNo. **Problem Statement**

1.

Medium Level: **Sort Colors.**

Important Point to be noticed: (We sort it using stl library without a second thought but for particular this problem we can't use sort library . So there is an algorithm that is known by DNF so in this we set low and mid to zero and high is length -1. We only adjust 0 and 2 position and one is placed self in their place. For better understanding I showed you code .)

Code:

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

```
void sortColors(vector<int>& nums) {

        /*  int low = 0;
          int mid = 0;
          int high = nums.size() - 1;

    while(mid<=high){
        if(nums[mid] == 0){
            swap(nums[low], nums[mid]);
            low++;
            mid++;
        }
        else if(nums[mid] == 1){
            mid++;
        }
        else{
            swap(nums[high], nums[mid]);
            high--;
        }
    }*/
        sort(nums.begin(),nums.end());
}
```

2.

Medium Level: Longest Repeating Character Replacement.

You are given a string `s` and an integer `k`. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most `k` times.

Return *the length of the longest substring containing the same letter you can get after performing the above operations*.

Example 1:

```
Input: s = "ABAB", k = 2

Output: 4

Explanation: Replace the two 'A's with two 'B's or vice versa.
```

Approach:

First you read the problem statement for better understanding ,and When I read the PS then I get first point take map and store string in map.
Why I take map? -: because map sotre the count of char .
So after store in map ,I took two pointer start and end ,initialize to zero. Now I check the condition because I perform operation k times so for that I check end-start+1-c>k
Then we reduce it frequency and increment start.
And find max , increment end , return max.

Code:

```
int characterReplacement(string s, int k) {

        unordered_map<char,int>m;
        int maxi=INT_MIN;
        int c=0;
        int st=0,e=0;
        while(e < s.length())
        {
```

```
        m[s[e]]++;
        c=max(c,m[s[e]]);
        if(e-st+1-c>k)
        {
            m[s[st]]--;
            st++;
        }
        maxi=max(maxi,e-st+1);
        e++;
    }
    return maxi;
}
```

The Most Important Topic In DSA : Dynamic Programming.

Dynamic Programming is mainly an optimization over plain recursion.
The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

SNo.	Problem Statement

1.

Easy Level: Climbing Stairs.

Code:

```
Input: n = 2

Output: 2

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

1. 1 step + 1 step

2. 2 steps
```

```
int climbStairs(int n) {
        int t[n+1];
        t[0] =1;
        t[1] = 1;
        for(int i=2; i<n+1; i++) t[i] = t[i-1] + t[i-2];
        return t[n];

    }
```

2.

Easy Level: Maximum Product Subarray.

Code:

```
Input: nums = [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.
```

```
int maxProduct(vector<int>& nums) {

    /*    int n=nums.size();
        int dp[n][2];

        dp[0][0]=nums[0];
        dp[0][1]=nums[0];
        int ans=dp[0][0];
        for(int i=0;i<=n;i++)
        {
            for(int j=0;j<=n;j++)
            {
                if(i==0 || j==0)
                {
                    dp[i][j]=0;
                }
            }
        }
        for(int i=1;i<n;i++){
            dp[i][0] = max(nums[i],max(dp[i-1][0]*nums[i],dp[i-1][1]*nums[i]));
            dp[i][1] = min(nums[i],min(dp[i-1][0]*nums[i],dp[i-1][1]*nums[i]));
            ans = max(ans,dp[i][0]);
        }
        return ans;*/
```

Solution-2 :)

```
      int r=nums[0];
     int maxi=r;
     int mini=r;
        for(int i=1;i<nums.size();i++)
        {
            if(nums[i]<0)
            {
                int temp=maxi;
                maxi=mini;
                mini=temp;
            }
            maxi=max(nums[i],nums[i]*maxi);
            mini=min(nums[i],nums[i]*mini);
            r=max(r,maxi);
        }
        return r;


        //brute force due to time limit did not accepted
```

```
 // int maxProduct(vector<int>& A) {
  /*int ans = INT_MIN;
  for(int i = 0; i < nums.size(); i++) {
      int curProd = 1;
      for(int j = i; j < nums.size(); j++)
          curProd *= nums[j],
          ans = max(ans, curProd);
  }
  return ans;*/
}
```

3.

Easy Level: Ones and Zeroes.
Code:

```
Input: strs = ["10","0001","111001","1","0"], m = 5, n = 3

Output: 4

Explanation: The largest subset with at most 5 0's and 3 1's is {"10",
"0001", "1", "0"}, so the answer is 4.

Other valid but smaller subsets include {"0001", "1"} and {"10", "1",
"0"}.

{"111001"} is an invalid subset because it contains 4 1's, greater than
the maximum of 3.
```

```
vector<vector<vector<int>>>dp;
    int maxOneandZero(vector<string>& strs,int i,int m,int n)
    {
        if(m<0 || n<0)
            return -1e9;
        if(m==0 and n==0)
            return 0;
        if(i==strs.size())
            return 0;
        if(dp[i][m][n]!=-1)
            return dp[i][m][n];

        int c1=0;
        int c2=0;

        for(auto it:strs[i])
        {
           if(it=='1')
              c1++;
           else
              c2++;
        }
        return dp[i][m][n]=max(1+maxOneandZero(strs,i+1,m-c2,n-c1),maxOneandZero(strs,i+1,m,n));
     }
     int findMaxForm(vector<string>& strs, int m, int n) {
        int s = strs.size();
        dp.resize(s, vector<vector<int>>(m+1, vector<int>(n+1,-1)));

        return max(maxOneandZero(strs,0,m,n),0);
     }
```

4. **Easy Level: Counting Bits.**

Code:

```
Input: n = 2

Output: [0,1,1]

Explanation:

0 --> 0

1 --> 1

2 --> 10
```

```
vector<int> countBits(int n) {
     /*   vector<int>arr(n+1);
        arr[0]=0;
        for(int i=0;i<=n;i++)
        {
           if(i & 1)//condition for n=odd
              arr[i]=1;
           else
              arr[i]=0;
           arr[i]+=arr[i/2];
        }
        return arr;*/
        vector<int>res;
        for(int i=0;i<=n;i++)
```

```
    {
        int c=0;
        int num=i;
        while(num)
        {
            c++;
            num=num & (num-1);
        }
        res.push_back(c);
    }
    return res;
}
```