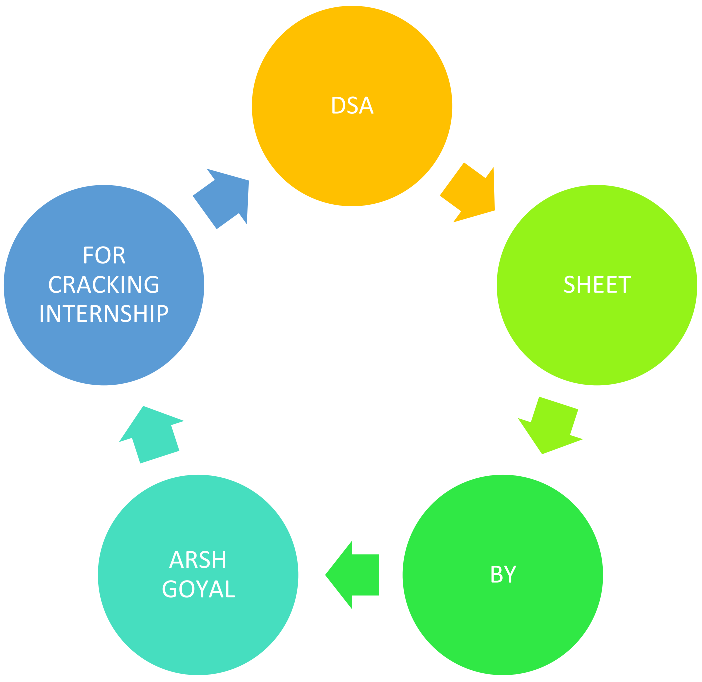
SOLUTION OF ARRAY, STRING, MATHEMATICAL,SORTING AND SEARCHING,LINKED LIST EASY AND MEDIUM LEVEL PEOBLRM

shivani patel



SNO.	LEVEL OF PROBLEM	TOPIC COVERED IN THIS DOCUMENT
1.	EASY	ARRAY
2.	MEDIUM	ARRAY
3.	EASY	STRING
4.	MEDIUM	STRING
5.	EASY	MATHEMATICAL
6.	MEDIUM	MATHEMATICAL
7.	EASY	SORTING AND SEARCHING
8.	MEDIUM	SORTING AND SEARCHING
9.	EASY	LINKED LIST
10.	MEDIUM	LINKED LIST

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:

```cpp
#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:
```cpp
#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;
}
```

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

Code:
```cpp
#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--;
            }
```

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

```
            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;
}
```

</td></tr>
<tr><td>8.</td><td>

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];
```

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

```
         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.	**Medium Level-Integer to Roman**
	Code:
	`#include <bits/stdc++.h>`
	`#include <iostream>`
	
	`using namespace std;`
	`string sTonum(int num)`
	`{`
	`    string ans="";`
	`    while(num >= 1000){`
	`            ans += "M";`
	`            num -= 1000;`
	`        }`
	`        if(num >= 900){`
	`            ans += "CM";`
	`            num -= 900;`
	`        }`
	`        while(num >= 500){`
	`            ans += "D";`
	`            num -= 500;`
	`        }`
	`        if(num >= 400){`
	`            ans += "CD";`
	`            num -= 400;`
	`        }`
	`        while(num >= 100){`
	`            ans += "C";`
	`            num -= 100;`
	`        }`
	`        if(num >= 90){`
	`            ans += "XC";`
	`            num -= 90;`
	`        }`
	`        while(num >= 50){`
	`            ans += "L";`
	`            num -= 50;`
	`        }`
	`        if(num >= 40){`
	`            ans += "XL";`

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

```
        num -= 40;
    }
    while(num >= 10){
        ans += "X";
        num -= 10;
    }
    if(num >= 9){
        ans += "IX";
        num -= 9;
    }
    while(num >= 5){
        ans += "V";
        num -= 5;
    }
    if(num >= 4){
        ans += "IV";
        num -= 4;
    }
    while(num >= 1){
        ans += "I";
        num -= 1;
    }
    return ans;

}
int main()
{
  int num=3;
  cout<<sTonum(num);

  return 0;
}
```

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

Medium Level-Generate Parentheses
Code:

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

void generateParenthesis(int n, int o, int c, string s, vector<string>
&ans){
```

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

```
    if(o==n && c==n){
        ans.push_back(s);
        return;
    }

    if(o<n){
        generateParenthesis(n, o+1, c, s+"{", ans);
    }

    if(c<o){
        generateParenthesis(n, o, c+1, s+"}", ans);
    }

}
int main() {
    int n = 3;

    vector<string> ans;

    generateParenthesis(n,0,0,"",ans);

    for(auto s:ans){
        cout<<s<<endl;
    }
    return 0;
}
```

3. **Medium Level-Simplify Path**

Code:

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

using namespace std;
 string simplifyPath(string path) {

        string res;
        stack<string>s;
        for(int i=0;i<path.size();i++)
        {
            if(path[i]=='/')
                continue;
            string tmp;
            while(i<path.size() and path[i]!='/')
            {
                tmp+=path[i];
                i++;
            }
            if(tmp==".")
                continue;
            else if(tmp=="..")
            {
                if(!s.empty())
                    s.pop();
            }
            else
                s.push(tmp);
        }
            while(!s.empty())
            {
                res="/"+s.top()+res;
                s.pop();
            }
            if(res.size()==0)
                return "/";
        return res;
    }
int main()
{
    string path="/home/";
    string p=simplifyPath(path);
    cout<<p;
    return 0;
}
```

4. **Medium Level-Smallest window in a string containing all the characters of another string**
Code:

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

using namespace std;
 string smallestWindow (string s, string p)
 {

    int i=0;
    int j=0;
    int count = 0;
    int reqcount = 0;
    string pans;
    string ans;
    unordered_map<char, int> mp1;
    unordered_map<char, int> mp2;
    for(int i=0 ; i<p.size() ; i++){
        mp1[p[i]]++;
    }
    reqcount = p.size();

    while(true){
        bool loop1 = false;
        bool loop2 = false;
        while(i<s.size() && count<reqcount){
            mp2[s[i]]++;
            if(mp2[s[i]] <= mp1[s[i]]){
                count++;
            }
            loop1 == true;
            i++;
        }
        while(j<i && count==reqcount){
            pans = s.substr(j, i-j);
            if(ans.size() == 0 || pans.size() < ans.size()){
                ans = pans;
            }

            if(mp2[s[j]] == 1){
                mp2.erase(s[j]);
            }
            else{
                mp2[s[j]]--;
            }
            if(mp2[s[j]] < mp1[s[j]]){
                count--;
                }
                j++;
                loop2 = true;
            }
            if(loop1 == false && loop2 == false){
                break;
            }
        }
        if(ans.size() == 0){
            return "-1";
        }
        return ans;
    }
int main()
{
    string s="timetopractice";
    string p="toc";
    string r=smallestWindow(s,p);
    cout<<r;
    return 0;
}
```

5. **Medium Level-Reverse Words in a String**

Code:

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

using namespace std;
 string reverse(string s)
 {
     string temp;
        string ans;
        int n = s.size();
        for(int i=n-1;i>=0;i--)
        {
            if(s[i]!=' ')
            {   temp="";
                while(i>=0 && s[i]!=' ')
                {
                    temp+=s[i];
                    i--;
            } reverse(temp.begin(),temp.end());
            ans+=temp; ans+=' ';
          }
       }
       ans.pop_back();
       return ans;
 }
int main()
{
   string s="the sky is blue";

   string r=reverse(s);
   cout<<r;
   return 0;
}
```

6. **Medium Level-Rabin-Karp Algorithm for Pattern Searching**

Code:

```
#include <bits/stdc++.h>
#include <iostream>
#define d 256
using namespace std;

void search(char pat[], char txt[], int q)
{
   int M = strlen(pat);
   int N = strlen(txt);
   int i, j;
   int p = 0;
   int t = 0;
   int h = 1;


   for (i = 0; i < M - 1; i++)
       h = (h * d) % q;


   for (i = 0; i < M; i++)
   {
       p = (d * p + pat[i]) % q;
       t = (d * t + txt[i]) % q;
```

```
    }


    for (i = 0; i <= N - M; i++)
    {


        if ( p == t )
        {
            bool flag = true;

            for (j = 0; j < M; j++)
            {
                if (txt[i+j] != pat[j])
                {
                  flag = false;
                  break;
                }
                  if(flag)
                  cout<<i<<" ";

            }



            if (j == M)
                cout<<"Pattern at index "<< i<<endl;
        }


        if ( i < N-M )
        {
            t = (d*(t - txt[i]*h) + txt[i+M])%q;


            if (t < 0)
            t = (t + q);
        }
    }
}
```

```
int main()
{
    char txt[] = "GEEKS FOR GEEKS";
    char pat[] = "GEEK";


    int q = 101;


      search(pat, txt, q);
    return 0;
}
```

7. **Medium Level-Group Anagrams.**
Code:

```
vector<vector<string>> groupAnagrams(vector<string>& strs) {


        vector<vector<string>>res;
         unordered_map<string,vector<string>>m;
        for(auto it:strs)
        {
            string curr=it;
            sort(curr.begin(),curr.end());
            m[curr].push_back(it);
        }
        for(auto i:m)
            res.push_back(i.second);
        return res;
    }
```

8. **Medium Level-Word Wrap.**
Code:

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


int solve(int ind ,int n , vector<int>& nums , int k , vector<int>&
dp){
       if(ind >= n )
```

	```
	        return 0;
	        if(dp[ind] != -1)
	        return dp[ind];
	        int ans = INT_MAX;
	        int sum = 0 ;
	        for(int i = ind ; i < n ; i++){
	            sum += nums[i];
	            if(sum + (i - ind) <=k){
	                int cost = 0;
	                if(i != n - 1){
	                    cost = pow(k - sum - i + ind , 2);
	                }
	                ans = min(ans ,cost + solve(i + 1, n , nums , k , dp));

	            }
	        }
	        return dp[ind] = ans;
	    }
	    int solveWordWrap(vector<int>nums, int k)
	    {
	        // Code here
	         int n = nums.size() ;
	         vector<int>dp(n , -1);
	         return solve(0 , n , nums , k , dp);
	    }

	int main()
	{
	    vector<int>nums={3, 2, 2, 5};
	    int k=6;

	    cout<<solveWordWrap(nums,k);
	    return 0;
	}
	```

9.	**Medium Level-Basic Calculator II**
	Code:
	`#include <bits/stdc++.h>`
	`using namespace std;`

```
int calculate(string s) {
        s += '+';
        stack<int>x;

        char sign='+';
        int curr=0;
        int ans=0;
        for(int i=0;i<s.size();i++)
        {
            if(isdigit(s[i]))
            {
                curr=10*curr+(s[i]-'0');
            }
            else if(s[i]=='+' || s[i]=='-' || s[i]=='*'  ||s[i]=='/' )
            {
                if(sign=='+')
                {
                    x.push(curr);

                }

                else if(sign=='-')
                {
                    x.push(-curr);

                }

                else if(sign=='*')
                {
                    int a=x.top();
                    x.pop();
                    int b=curr*a;
                    x.push(b);

                }

                else if(sign=='/')
                {
                    int a=x.top();
                    x.pop();
```

	```
	                    int b=a/curr;
	                    x.push(b);

	                }
	                curr=0;
	                sign=s[i];

	            }

	        }
	         while(!x.empty())
	            {
	                ans=ans+x.top();
	                x.pop();
	            }
	            return ans;
	    }
	int main()
	{
	    string s="3+2*2";
	    cout<<calculate(s);
	    return 0;
	}
	```

SNo.	Problem Statement

1. **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;
}
```

2. **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)
```

```
                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;
}
```

SNo.	Problem Statement
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;
}
```

SNo.	Problem Statement

1.	**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;`
	`}`

2.	**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;`

```
            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;
	}
	```

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;
    }
}
```