EASY :

1. Palindrome string

```
class Solution{
public:
        int isPalindrome(string S)
        {
         int j,i=0;
    j=S.size()-1;
   while(i<j)
   {
     if(S[i]!=S[j])
     return 0;
     i++;
     j--;
   }
   return 1;
  }
};
```

2. Roman to Integer

```
class Solution {
public:
   int romanToInt(string s)
   {
      map<char,int>mp;
      int sum=0;
      int I,V,X,L,C,D,M;
      mp.insert({'I',1});
      mp.insert({'V',5});
      mp.insert({'X',10});
      mp.insert({'L',50});
      mp.insert({'C',100});
      mp.insert({'D',500});
      mp.insert({'M',1000});

      char c=s[s.size()-1];
      sum+=mp[c];

      for(int i=s.size()-2;i>=0;i--)
      {
         char ch=s[i];
         char ch1=s[i+1];
         if(mp[ch]<mp[ch1])
              sum-=mp[ch];
              else
```

```
            sum+=mp[ch];
        }
        return sum;
    }
};
```

MEDIUM :

1. Count and say

```
class Solution {
public:
    string countAndSay(int n)
    {

        if(n==1)
        return "1";

        else
        {
            string s=countAndSay(n-1);
            string ans="";

            char ch=s[0];
            int t=0;

            for (int i=0;i<s.length();i++)
            {
                if(ch==s[i])
                t++;

                else
                {
                    ans+=to_string(t)+ch;
                    ch=s[i];
                    t=1;
                }
            }
            ans+=to_string(t)+ch;
            return ans;
        }
    }
};
```

2. Longest palindrome substring

```
class Solution {
```

```
public:
    string maxx_palindrome(string s)
    {
        int n=s.size(),size;
        if(n<=1)
        return s;
        int max_l=1;
        int l,r;
        int st=0;
        for(int i=0;i<n;i++)
        {
            l=i;
            r=i;
            while(l>=0&&r<n && s[l]==s[r])
            {
                l--;
                r++;
            }
            size=r-l-1;
            if(size>max_l)
            {
                max_l=size;
                st=l+1;
            }

            l=i;
            r=i+1;
            while(l>=0&&r<n && s[l]==s[r])
            {
                l--;
                r++;
            }
            size=r-l-1;
            if(size>max_l)
            {
                max_l=size;
                st=l+1;
            }
            cout<<max_l;
         }
        return s.substr(st,max_l);
    }
    string longestPalindrome(string s)
    {
        string ans=maxx_palindrome(s);
        return ans;
    }};
```

3. Longest repeating subsequence // this is almost same as LCS

```
class Solution {
        public:
                int LongestRepeatingSubsequence(string str)
                {
                   // Code here
                   string str1=str;
          int x=str.size();
          int y=x;
          vector<vector<int>>dp(x+1,vector<int>(y+1));
          for(int i=1;i<=x;i++)
          for(int j=1;j<=y;j++)
          dp[i][j]=(str1[i-1]==str[j-1] && i!=j)?dp[i-1][j-1]+1:max(dp[i-1][j],dp[i][j-1]);
           return dp[x][y];
       }
};
```

4. All permutation of a string

```
class Solution
{
        public:
        void solve(int i,string s,vector<string>&ans)
        {
          if(i==s.size())
          {
             ans.push_back(s);
             return ;
          }
         for(int k=i;k<s.size();k++)
          {
             swap(s[i],s[k]);
             solve(i+1,s,ans);
             swap(s[i],s[k]);
          }
   }
     vector<string>find_permutation(string s)
        {
           vector<string>ans;
           solve(0,s,ans);
           sort(ans.begin(),ans.end());
           return ans;
        }
};
```

5. Edit distance

```
class Solution {
  public:
    int editDistance(string s, string t) {

        int n=s.size();
        int m=t.size();
        int dp[n+1][m+1];

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

                else if(j==0)
                dp[i][j]=i;

                else if(s[i-1]==t[j-1])
                dp[i][j]=dp[i-1][j-1];

                else
                dp[i][j]=1+min({dp[i][j-1], dp[i-1][j], dp[i-1][j-1]});
            }
        }
        return dp[n][m];
    }
};
```

6. Next permutation

```
class Solution{
public:
    vector<int> nextPermutation(int N, vector<int> arr){
        // code here
         int i=0;
        if(N<=1)
        return arr;
        for(i=N-1;i>0;i--)
        {
            if(arr[i]>arr[i-1])
            break;
        }
        if(i==0)
        reverse(arr.begin(),arr.end());
        else
        {
```

```
            int x=arr[i-1];
            int s=i;
            for(int j=i+1;j<N;j++)
            {
                if(arr[j]>x && arr[j]<=arr[s])
                s=j;
            }

            swap(arr[i-1],arr[s]);
            sort(arr.begin()+i,arr.end());
        }
        return arr;
    }
};
```

7. Count reversals

```
int countRev (string s)
{
     int n=s.size();
    if(n%2==1)
    return -1;
    int c=0;
    int r=0;
    int l=0;
    for(int i=0; i<n; i++)
    {
        if(r>l)
        {
            r--;
            l++;
            c++;
        }
        if(s[i]=='{')
        l++;

        else if(s[i]=='}')
        r++;
    }
    l=l-r;
    c+=l/2;
    return c;
}
```

8. Count all palidromic subsequence

```
class Solution{
```

```
    public:
    long long int  countPS(string str)
    {
      long long int md=1e9+7;
      long long int mat[1001][1001];
      int n=str.size();
      for(int i=0;i<n;i++)
      for(int j=0;j<n;j++)
       {
          if(i==j)
          mat[i][j]=1;
      }
      for(int i=0;i<n;i++)
      for(int j=0;j<n;j++)
      {
          if(i==j-1 and str[i]==str[j])
          mat[i][j]=3;
          else if(i==j-1)
          mat[i][j]=2;
      }

      for(int k=2;k<n;k++)
      {
          int i=0;
          int j=k;
          while(i<n and j<n)
          {
              if(str[i]==str[j])
              mat[i][j]=1+mat[i+1][j]+mat[i][j-1];
              else
              mat[i][j]=mat[i+1][j]+mat[i][j-1]-mat[i+1][j-1];
              mat[i][j]+=md;
              mat[i][j]%=md;
              i++;
              j++;
          }
      }
      return mat[0][n-1];
    }
};
```