1. Find Duplicate characters in a string

Method 1:For each element traverse entire string
comp=O(n^2);

Method 2:By using sorting,comp=O(nlogn)
Create curr_max,curr_cahr,max_count,max_char;

Method 3:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	string str;
	cin>>str;
	int count[256]={0};
	int i=0;
	while(str[i++]!=NULL){
		count[str[i]]++;
	}
	for(int i=0;i<256;i++){
		if(count[i]>1){
			cout<<char(i)<<" "<<count[i]<<" ";
			//printf("%c and %d",i,count[i]);
		}
	}
	return 0;
}
```

complexity=O(n);

2.Write a Code to check whether one string is a rotation of another

Method 1:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	string str1,str2;
	cin>>str1>>str2;
	if(str1.length()!=str2.length()){
		cout<<"No"<<endl;
		return 0;
	}
	int i=0,j=0;
	queue<char> q1;
	queue<char> q2;
	while(str1[i]!='\0')q1.push(str1[i++]);
	while(str2[j]!='\0')q2.push(str2[j++]);
	int size=str2.length();
	while(size--){
		char temp=q2.front();
		cout<<temp<<endl;
		q2.pop();
		q2.push(temp);
		if(q1==q2){
			cout<<"Yes"<<endl;
			return 0;
		}
	}
```

```
        cout<<"No"<<endl;
        return 0;

}
```

Complexity=O(N^2);

Method 2: KMP

```
#include<bits/stdc++.h>
using namespace std;
int main(){
        string s1,s2;
        cin>>s1>>s2;
        string s3=s1+s1;
        int len3=s3.length(),len2=s2.length();
        int j=0;
        for(int i=0;i<len3;i++){
                if(s3[i]==s2[j])j++;
                if(j==len2){
                cout<<"Yes"<<endl;
                return 0;}
        }
        cout<<"NO"<<endl;
        return 0;
}
```

Complexity=O(2n)
Space comp=O(n);

3.Write a Program to check whether a string is a valid shuffle of two strings or not

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	string s1,s2;
	cin>>s1>>s2;
	string s3;
	cin>>s3;
	if(s1.length()+s2.length() !=s3.length()){
		cout<<"No"<<endl;
		return 0;
	}
	int k=s3.length(),i=0,j=0,l=0;
	while(k--)
	{
		if(i<s1.length() && s1[i]==s3[l])i++;
		else if (j<s2.length() && s2[j]==s3[l])j++;
		else {
			cout<<"No"<<endl;
			return 0;
		}
		l++;
	}
	cout<<"Yes"<<endl;
	return 0;
}
Complexity=O(n);
```

4.Count and Say problem

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    string str;
    cin>>str;
    string ans="";
    int count=1;
    int i=1;
    int l=str.length();
    while(i<l){
        if(str[i]==str[i-1])count++;
        else {
            ans+=to_string(count);
            ans+=str[i-1];
            count=1;
        }
        i++;
    }
        ans+=to_string(count);
        ans+=str[i-1];
    cout<<ans;
    return 0;
```

}

5.Longest palindrome in string

Method-1 :Brute Force.

Approach: The simple approach is to check each substring whether the substring is a palindrome or not. To do this first, run three nested loops, the outer two loops pick all substrings one by one by fixing the corner characters, the inner loop checks whether the picked substring is palindrome or not.

compl=O(n^3)

Method-2:Dynamic approach
Compl=O(n^2) space comp=O(n^2)

Method-3:Optimal

```
int longestPalSubstr(char* str)
{
        int maxLength = 1;

    int start = 0;
```

```
    int len = strlen(str);

    int low, high

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

      //EVEN LENGTH PALINDROMES

        low = i - 1;
        high = i;
        while (low >= 0 && high < len
                && str[low] == str[high]) {
            --low;
            ++high;
        }
        ++low; --high;
          if (str[low] == str[high] && high
- low + 1 > maxLength) {
            start = low;
              maxLength = high - low + 1;
        }

       //ODD LENGTH PALINDROMES

        low = i - 1;
        high = i + 1;
        while (low >= 0 && high < len
```

```
                && str[low] == str[high]) {
            --low;
            ++high;
        }

        ++low; --high;
          if (str[low] == str[high] && high
- low + 1 > maxLength) {
            start = low;
              maxLength = high - low + 1;
        }
    }

    cout << "Longest palindrome substring
is: ";
    int ans=maxlength;
    while(ans--)
      cout<<str[start++];

      return maxLength;
}
Complexity:O(n^2)    space comp=O(1).
```

6.Total number of subsequences

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

```
using namespace std;
void solve(string ip,string op){
    if(ip.length()==0){
        cout<<op<<" ";
        return;
    }
    solve(ip.substr(1),op+ip[0]);
    solve(ip.substr(1),op);
    return;
}
int main(){
    string s;
    cin>>s;
    solve(s,"");
    return 0;
}
Complexity=O(2^n)
```

8.SPLIT BINARY STRING INTO SUBSTRINGS HAVING EQUAL NO. OF 0s AND 1s

Initialize count = 0 and traverse the string character by character and keep track of the number of 0s and 1s so far, whenever the count of 0s and 1s become equal increment the count. As in the given question, if it is not possible to split string then on that time count of 0s must not be equal to

count of 1s then return -1 else print the value of count after the traversal of the complete string.
Below is the implementation of the above approach:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	string s;
	cin>>s;
	int count=0,count_0=0,count_1=0;
	for(int i=0;i<s.length();i++){
		if(s[i]=='0')count_0++;
		else count_1++;
		if(count_0==count_1)
		{
		count++;
		}
	}
	cout<<count<<endl;
	return 0;
}
```

Comp=O(N);
Space=O(N);

9:Next Permutation

Bruteforce:Find all permutations comp=O(N!N)

Optimal:

```
class Solution{
public:
   vector<int> nextPermutation(int N, vector<int> arr){
       int idx=-1;
       for(int i=N-1;i>0;i--){
           if(arr[i]>arr[i-1]){
               idx=i;
               break;
           }
       }
       int prev=idx;
       if(idx==-1){
           reverse(arr.begin(),arr.end());
           return arr;
       }
       else{
           int prev=idx;
       for(int k=idx+1;k<N;k++){
           if(arr[k]>arr[idx-1] && arr[k]<=arr[prev]){
               prev=k;
           }
       }
       swap(arr[idx-1],arr[prev]);
       reverse(arr.begin()+idx,arr.end());
       return arr;
   }}
```

};

Compl=O(N);

10:Number of permutation of string

```
#include<bits/stdc++.h>
using namespace std;
void permute(string s,int l,int r){
    if(l==r){
        cout<<s<<" ";
        return ;
    }
    else
    for(int i=l;i<=r;i++)
    {
    swap(s[l],s[i]);
    permute(s,l+1,r);
    swap(s[l],s[i]);   //backtracking
    }
}
int main(){
    string s;
    cin>>s;
    permute(s,0,s.length()-1);
```

```
    return 0;
}
```

Complexity=O(n*n!).
Space=O(n)

11:PERMUTATION WITH DUPLICATES

```
#include<bits/stdc++.h>
using namespace std;
void permute(string s,int l,int r){
    if(l==r){
        cout<<s<<" ";
        return ;
    }
    unordered_set <int> o;
    for(int i=l;i<=r;i++)
    {
        if(o.find(s[i])!=o.end())continue;
        o.insert(s[i]);
    swap(s[l],s[i]);
    permute(s,l+1,r);
    swap(s[l],s[i]);    //backtracking
    }
}
int main(){
```

```
    string s;
    cin>>s;
    permute(s,0,s.length()-1);
    return 0;
}
```

```
Complexity=O(N*N!)
Space=O(N)
```

12:REMOVE ADJACENT DUPLICATES FROM STRING

```
Method-1:Using stack---time comp=O(n)
space=O(n)
```

```
Method-2:Using recursion
```

```
#include<bits/stdc++.h>
using namespace std;
string remove_ad(string s){
    int n=s.size();
    if(n==0 || n==1)return s;

    if(s[0]==s[1]){
        int idx=0;
```

```
        while(s[0]==s[idx] && idx<n)idx++;
        return [0]+remove_ad(s.substr(idx));
    }
    return s[0]+remove_ad(s.substr(1));
}
int main(){
    string s;
    cin>>s;
    cout<<remove_ad(s);
    return 0;
}
```

Comp=O(n) Space=O(n)

Method 3:Recursion

```
class Solution{
    public:
    string rdup(string s,string &op){
        if(s.size()==0)return op;
        else {
            if(s[0]!=s[1])op+=s[0];
            return rdup(s.substr(1),op);
        }
    }
    string
removeConsecutiveCharacter(string S)
```

```
    {
       string op="";
       return rdup(S,op);
    }
};
```

13:Find first duplicate word in a string

```
void first_duplicatestr(string s)
{
    unordered_map<string,int> m;
    string t="",ans="";
    for(int i=s.length()-1;i>=0;i--){
   if(s[i]!=' '){
        t+=s[i];

    }
   else{
        m[t]++;
        if(m[t]>1)ans=t;

        t="";
    }
  }
  m[t]++;
```

```
  if(m[t]>1)ans=t;

  if(ans!=""){
    reverse(ans.begin(),ans.end());
  cout<<ans<<" ";}
  else cout<<"No repitition"<<" ";
}
```

14:Check if string is rotated by 2 places

```
bool rotate_2places(string s1,string s2){
    //clockwise
if(s1.length()!=s2.length())return false;
    string
b1=s2.substr(s2.length()-2,s2.length());
    string b2=s2.substr(0,s2.length()-2);
    if(b1+b2==s1)return true;
//Anti_clockwise
    b1=s2.substr(2,s2.length());
    b2=s2.substr(0,2);
    if(b1+b2==s1)return true;
    return false;
}
```

15:Anagram

An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, “abcd” and “dabc” are an anagram of each other.

```
bool anagram(string s1,string s2){
if(s1.length()!=s2.length())return false;
int count[256];
for(int i=0;i<s1.length();i++)
      Count[s1[i]]++;
for(int i=0;i<s2.length();i++)
      Count[s2[i]]--;
fro(int i=0;i<256;i++)
  if(count[i]>1)return false;

return true;
}
```

16:Longest Common Prefix

```
string longestcommon_prefix(vector<string>
&v){
    if(v.size()==0)return " ";
    string t=v[0];
    int mi=INT_MIN;
    for(int i=1;i<v.size();i++){
        int j=0,k=0,a=0;
```

```
        while(j<t.size() && k<v[i].size()){
            if(t[j]==v[i][k])a++;
            else break;
            k++;
            j++;
        }
        mi=max(a,mi);
    }
    return t.substr(0,mi);
}
Complexity=O(n)
space=O(1)
```

17:Rabin karp:

```
Input:  txt[] = "THIS IS A TEST TEXT"
        pat[] = "TEST"
Output: Pattern found at index 10
```

```
void rabin_karp(char m[],char n[]){
    int q=101;
    int N=strlen(m);
    int M=strlen(n);
    int p=0,t=0,h=1,i,j;
    for(i=0;i<M-1;i++)h=(h*d)%q;

    for(i=0;i<M;i++){
```

```
        p=(p*d+n[i])%q;
        t=(t*d+m[i])%q;
    }
    for(i=0;i<=N-M;i++){
            if(p==t){
            bool flag=true;
            for(j=0;j<M;j++){
                if(n[j]!=m[i+j]){
                    flag=false;
                    break;
                }
                if(flag)
                    cout<<i<<" ";
            }
            if(j==M)cout<<i<<endl;
        }
        if(i<N-M){
        t=(d*(t-m[i]*h))+m[i+M]%q;
        if(t<0)t=t+q;}
    }
    return;
}
```

18:Generate substrings

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

```
using namespace std;
void getsubstr(string s,int t,int i){
    //int i=s.size();
    if(i==t)return;
    cout<<s.substr(0,t)<<endl;
    getsubstr(s,++t,i);
    return ;
}
void substrings(string s){
    if(s.size()==0)return;
    int t=1;
    getsubstr(s,t,s.size()+1);
    substrings(s.substr(1));
    return;
}
int main(){
    string s;
    cin>>s;
    substrings(s);
    return 0;
}
```

19:COUNT AND SAY

```
 string countAndSay(int n) {
        if(n==1)return "1";
```

```
        if(n==2)return "11";

        string str="11";
        for(int i=3;i<=n;i++){
            str +="$";
            string temp="";
            int len=str.length();
            int count=1;
            for(int j=1;j<len;j++)
            {
                if(str[j]!=str[j-1]){
                    temp +=count+'0';
                    temp +=str[j-1];
                    count=1;
                }
                else count++;
            }
            str=temp;
        }
        return str;

    }
```

20. Parenthesis Checker:

```
class Solution
{
```

```
    public:
    //Function to check if brackets are balanced or not.
    bool ispar(string x)
    {
        stack<char> s;
        for(int i=0;i<x.size();i++){
            if(x[i]=='[' || x[i]=='{' || x[i]=='('
){s.push(x[i]);continue;}
            if(s.size()<1)return false;
            char x1;
            switch(x[i]){
                case ']':
                    x1=s.top();
                    s.pop();
                    if(x1=='(' || x1=='{')return false;
                    break;
                case ')':
                    x1=s.top();
                    s.pop();
                    if(x1=='{' || x1=='[')return false;
                    break;
                case '}':
                    x1=s.top();
                    s.pop();
                    if(x1=='(' || x1=='[')return false;
                    break;
            }
```

```
        }
            if(s.size()>0)return false;
            else return true;

    }

};
```