1. 2nd largest ele in arr

```
int getSecondLargest(vector<int> &arr) {
    // Code Here
   /* int min=INT_MAX;
    int min2=INT_MAX;
    int n=sizeof(arr)/sizeof(arr[0]);
    if(n==0||n==1)
    return -1;
    for(int i=0;i<n;i++)
    {
        if(arr[i]<min&&arr[i]<min2)
        {
        min2=min;
        min=arr[i];
        }
        else if(arr[i]>min&&arr[i]<min2)
         min2=arr[i];
    }

  return min2;*/
   int max=INT_MIN;
    int max2=INT_MIN;
    int n=arr.size();
    if(n==0||n==1)
    return -1;
    for(int i=0;i<n;i++)
    {
        if(arr[i]>max&&arr[i]>max2)
        {
        max2=max;
```

```
            max=arr[i];
          }
          else if(arr[i]<max&&arr[i]>max2)
           max2=arr[i];
       }
       if(max2==INT_MIN)
       return -1;

    return max2;
   }
};
```

2. Remove duplicates in sorted array inplace

```
3.  class Solution {
4.  public:
5.      int removeDuplicates(vector<int>& nums) {
6.          int n=nums.size();
7.          int temp=nums[0];
8.          int count=1;
```

```
9.          if(n<2)
10.         return n;
11.         for(int i=1;i<n;i++)
12.         {
13.             if(nums[i]!=temp)
14.             {
15.             nums[count]=nums[i];
16.             count++;
17.             temp=nums[i];
18.             }
19.         }
20.         return count;
21.     }
    };
```

3.rotate array by k positions to left or right

Method1:using temp array:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   //to the right
```

```
    //use a temp array:
    int arr[7]={1,2,3,4,5,6,7};
    int n=7;
    int k=3;
    int temp[n];
    int j=0;
    for(int i=(n-k);i<n;i++)
    {
        temp[j++]=arr[i];
    }
    for(int i=0;i<=k;i++)
    {
        temp[j++]=arr[i];
    }
     for(int i=0;i<n;i++)
    {
        cout<<temp[i]<<" ";
    }

    return 0;
}
```

Method 2 : using reversal algo

Clean, clear, concise code:

```
class Solution {
public:
 void reverse(int s,int e,vector<int>& nums)
    {
        while(s<e)
```

```
        {
            swap(nums[s],nums[e]);
            s++;
            e--;
        }

    }
    void rotate(vector<int>& nums, int k) {


    int n=nums.size();
    if(k>=n)
    k=k%n;


    reverse(0,n-1,nums);
    reverse(0,k-1,nums);
    reverse(k,n-1,nums);
    }
};
```

4.Move 0s to end of array: using additional temp array:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int arr[5]={0,1,0,3,12};
```

```
    int temp[5];
    int n=5;
    int j=0;
    for(int i=0;i<5;i++)
    {
        if(arr[i]!=0)
        temp[j++]=arr[i];
    }
     for(int i=j;i<5;i++)
     {
         temp[i]=0;

     }
     for(int i=0;i<5;i++)
     {
         cout<<temp[i]<<" ";

     }
    return 0;
}
```

LC CODE:

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
    int n=nums.size();
    vector<int>temp (n);
    int j=0;
    for(int i=0;i<n;i++)
```

```
    {
        if(nums[i]!=0)
        temp[j++]=nums[i];
    }
     for(int i=j;i<n;i++)
     {
         temp[i]=0;

     }
     for(int i=0;i<n;i++)
     {
         nums[i]=temp[i];
     }

    }
};
```

OPTIMIZED APPROACH: LC

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
     int n=nums.size();
     int count=0;
    for(int i=0;i<n;i++)
     {
```

```
    if(nums[i]!=0)
    {
    nums[count]=nums[i];
    count++;
    }
}
for(int i=count;i<n;i++)
nums[i]=0;

    }
};
```

5. Find union of 2 vectors code:

GFG Soln:

```
    vector<int> findUnion(vector<int> &a, vector<int> &b) {
        int n=a.size();
    int m=b.size();
    vector<int> v;
    int p=0;
    int q=0;
```

```
  while(p<n&&q<m)
  {
      if(a[p]==b[q])
      {
      if(v.size()==0||v.back()!=a[p])
      v.push_back(a[p]);
      p++;
      q++;
      }
      else if(a[p]<b[q])
      {
      if(v.size()==0||v.back()!=a[p])
      v.push_back(a[p]);
      p++;
      }
      else
      {
      if(v.size()==0||v.back()!=b[q])
      v.push_back(b[q]);
      q++;
      }
  }
  if(p<n)
  {
      for(int i=p;i<n;i++)
      {
      if(v.size()==0||v.back()!=a[i])
      v.push_back(a[i]);
      }
  }
  if(q<m)
```

```
    {
        for(int i=q;i<m;i++)
        {
        if(v.size()==0||v.back()!=b[i])
        v.push_back(b[i]);
        }
    }

    return v;
    }
};
```

6.MAX CONSECUTIVE 1's

A) USING COUNT APPROACH:

```
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int count1=0;
        // int count0=0;
```

```
        int max=INT_MIN;
        int n=nums.size();
        for(int i=0;i<n;i++)
        {
            if(nums[i]==1)
            count1++;
            else
            count1=0;
            if(count1>max)
            max=count1;
        }

        return max;
    }
};
```

7.LONGEST SUBARRAY SUM:

A) HASHING APPROACH:

```
class Solution {
  public:
    int lenOfLongestSubarr(vector<int>& arr, int k) {
        // code here
```

```
        int n=arr.size();
        int rem;
        int len;
        map<long long,int> preSumMap;
        int maxLen=INT_MIN;
        int sum=0;
        for(int i=0;i<n;i++)
        {
            sum+=arr[i];
            if(sum==k)
            {
                maxLen=max(maxLen,(i+1));
                preSumMap[sum]=i;
            }
            rem=sum-k;
            if(preSumMap.find(rem)!=preSumMap.end())  //rem part is found
            {
                len=i-preSumMap[rem];
                maxLen=max(len,maxLen);
            }
            if(preSumMap.find(sum)==preSumMap.end())
            {
                preSumMap[sum]=i;
            }

        }
        return maxLen;

    }
};
```

B) 2 POINTER APPROACH:

8. 2 sum problem:

A) HASHING APPROACH:

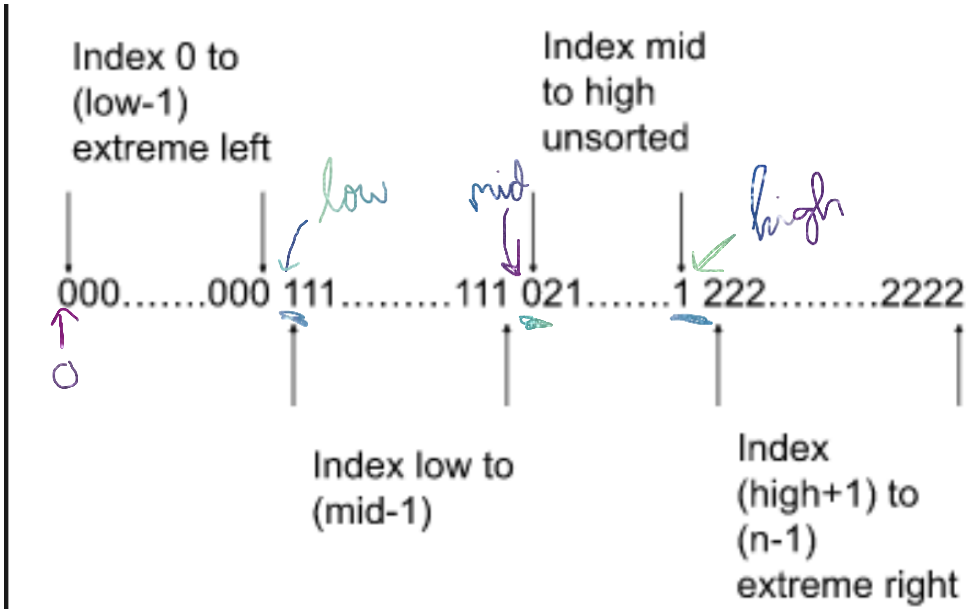
```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        //hashmap approach
        map<int,int> mp;
        vector<int> v;
        int temp;
        int rem;
        int n=nums.size();
        for(int i=0;i<n;i++)
        {
            mp[nums[i]]=i;
        }
        for(int i=0;i<n;i++)
        {
            rem=target-nums[i];
            if(mp.find(rem)!=mp.end())
            {
                auto it=mp.find(rem);
                temp=it->second;
                v.push_back(i);
```

```
            if(v.back()!=temp)
            {
            v.push_back(temp);
            break;
            }
            else
            v.clear();
        }

      }
      return v;
   }
};
```

9.sort array of 0s,1s and 2s

1.DNF ALGO:



10.Majority element:

1) LC SOLN HASHMAP:

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        //hashing approach:
        map<int,int> mp;
        int n=nums.size();
        int m;
        int c;
        int t;
        for(int i=0;i<n;i++)
        {
            mp[nums[i]]++;
        }
```

```
    m=mp.size();
    for(auto it:mp)
    {
        c=it.second;
        if(c>(n/2))
        {
            t=it.first;    USED AS SCOPE OF IT IS ONLY WITHIN THE LOOP
            break;
        }
    }
    return t;

  }
};
```

2) LC soln moore's voting algo optimal:

```
3) class Solution {
4) public:
5)     int majorityElement(vector<int>& nums) {
6)         //optimized approach Moore's voting algo:
7)         int n=nums.size();
8)         int ele;
9)         int c=0;
10)        int c2=0;
11)        for(int i=0;i<n;i++)
12)        {
13)            if(c==0)
14)            {
15)                c=1;
16)                ele=nums[i];
17)            }
18)            else if(nums[i]==ele)
19)            {
20)                c++;
21)            }
22)            else
23)            {
24)                c--;
25)            }
26)        }
27)        for(int i=0;i<n;i++)
```

```
28)         {
29)             if(nums[i]==ele)
30)             c2++;
31)         }
32)         if(c2<=(n/2))
33)         ele=-1;
34)         return ele;
35)     }
36) };
```

11) Kadane's algorithm:

1) almost optimal:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int n=nums.size();
        int s=0;
        int m=INT_MIN;
        int mn=INT_MIN;
        int count=0;
        for(int i=0;i<n;i++)
        {
            if(nums[i]<0)
            { count++;
            mn=max(mn,nums[i]);
            }
            s+=nums[i];
            if(s<0)
            s=0;
            else
            {
                m=max(s,m);
            }
```

```
        }
        if(count==n)
        m=mn;
        return m;
    }
};
```

2) Actual Kadane's logic: optimal soln:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int n=nums.size();
        int s=0;
        int m=INT_MIN;
        for(int i=0;i<n;i++)
        {
            s+=nums[i];
            m=max(s,m);
            if(s<0)
            s=0;
        }
        return m;
    }
};
```

3) Kadane's Algorithm with printing:

```
#include<bits/stdc++.h>
        using namespace std;
        int main()
```

```
    {
    int nums[9]={-2,1,-3,4,-1,2,1,-5,4};
    int n=9;
    int s=0;
    int start=-1;
    int startIndex=-1;
    int endIndex=-1;
    int m=INT_MIN;
    for(int i=0;i<n;i++)
    {
        if(s==0)
        start=i;
        s+=nums[i];
        if(s>m)
        {
            m=s;
            startIndex=start;
            endIndex=i;
        }
        if(s<0)
        s=0;
    }
    for(int i=startIndex;i<=endIndex;i++)
    {
        cout<<nums[i]<<" ";
    }
    return 0;
  }
```

12) BEST TIME TO BUY AND SELL STOCK:

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n=prices.size();
        int maxP=INT_MIN;
        int min=prices[0];
        for(int i=0;i<n;i++)
        {
            if(prices[i]<min)
            min=prices[i];
            maxP=max(maxP,(prices[i]-min));
        }
        return maxP;
    }
};
```

13) Rearrange positive and negative nums in array:

1) Brute Force Soln:

```
class Solution {
public:
    vector<int> rearrangeArray(vector<int>& nums) {

        int n=nums.size();
        int k=0;
        int j=1;
        vector<int> v(n,0);
        for(int i=0;i<n;i++)
        {
            if(nums[i]>0)
            {
            v[k]=nums[i];
```

```
            k+=2;
        }
        else
        {
        v[j]=nums[i];
        j+=2;
        }
    }
    return v;

  }
};
```

14) Next permutation code:

Optimized approach:

```
class Solution {
public:
  void nextPermutation(vector<int>& nums) {
    //optimized:
    int n=nums.size();
    int ind=-1;
    //step1: identifying the break point:
    for(int i=n-2;i>=0;i--)
    {
        if(nums[i]<nums[i+1])
        {
        ind=i;
        break;
```

```
            }
        }
        //edge case last perm:
        if(ind==-1)
        {
            reverse(nums.begin(),nums.end());
            return;
        }
        //step2: swapping the elements
        for(int i=n-1;i>=0;i--)
        {
            if(nums[i]>nums[ind])
            {
                swap(nums[i],nums[ind]);
                break;
            }
        }
        //step3: reversing the remaining part:
        reverse(nums.begin()+ind+1,nums.end());
    }
};
```

15) Leaders in an array code:

OPTIMIZED APPROACH:

```
class Solution {
   // Function to find the leaders in the array.
  public:
    vector<int> leaders(vector<int>& arr) {
        // Code here
        int n=arr.size();
        int t=arr[n-1];
        vector<int> v;
        v.push_back(t);
        for(int i=n-2;i>=0;i--)
        {
            if(arr[i]>=t)
            {
            t=arr[i];
           v.insert(v.begin(),t);    //IMPO
            }
        }
        return v;
    }
};
```

16) Longest consecutive subsequence:

BRUTE FORCE:

```
class Solution {
public:
    bool linearSearch(vector<int> nums,int num,int n)
    {
```

```
        for(int i=0;i<n;i++)
        {
            if(nums[i]==num)
            return true;
        }
        return false;
    }
    int longestConsecutive(vector<int>& nums) {
    //BRUTE FORCE:
    int x;
    int longest=1;
    int n=nums.size();
    if(n==0||n==1)
    return n;
    int count;
    for(int i=0;i<n;i++)
    {
        x=nums[i];
        count=1;
        while(linearSearch(nums,(x+1),n)==true)
        {x++;
        count++;
        }
        longest=max(longest,count);
    }
    return longest;
    }
```

BETTER APPROACH USING SORTING:

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        int n=nums.size();
        if(n==0)
        return n;
        sort(nums.begin(),nums.end());
        int lastSmaller=INT_MIN;
        int longest=1;
        int cnt=0;
        for(int i=0;i<n;i++)
        {
            if(nums[i]-1==lastSmaller)
            {
                cnt++;
                lastSmaller=nums[i];
            }
            else if(nums[i]!=lastSmaller)
            {
                cnt=0;
                lastSmaller=nums[i];
            }
            longest=max(longest,(cnt+1));
        }
        return longest;

    }
};
```

OPTIMIZED APPROACH USING SETS:

```
class Solution {
```

```
public:
    int longestConsecutive(vector<int>& nums) {
        int n=nums.size();
        int cnt=0;
        int m=INT_MIN;
        if(n==0)
        return n;
        int longest=1;
        unordered_set<int> s;
        for(int i=0;i<n;i++)
        {
            s.insert(nums[i]);
        }
        for(auto it:s)
        {
            if(s.find(it-1)==s.end())  //it is 1st element
            {
                auto x=it;
                cnt=0;
                while(s.find(x)!=s.end())
                {
                    x++;
                    cnt++;
                }
                m=max(cnt,m);
            }
        }
        return m;
    }
};
```

17) Set Matrix 0s:

1) Brute force soln:

```
class Solution {
public:
   void markrow(int i,int n,vector<vector<int>> &matrix)
   {
      for(int j=0;j<n;j++)
      {
         if(matrix[i][j]!=0)
         matrix[i][j]=-1;
      }
   }

   void markcol(int j,int m,vector<vector<int>> &matrix)
   {
      for(int i=0;i<m;i++)
      {
         if(matrix[i][j]!=0)
         matrix[i][j]=-1;
      }
   }

   void setZeroes(vector<vector<int>>& matrix) {
      //BRUTE FORCE Approach:
      int n=matrix.size();
      int m=matrix[0].size();

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

```
            for(int j=0;j<m;j++)
            {
                if(matrix[i][j]==0)
                {
                    markrow(i,n,matrix);
                    markcol(j,m,matrix);
                }
            }
        }

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

    }
};
```

2) Better soln:

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        //approach 2
        int n=matrix.size();
```

```
        int m=matrix[0].size();
        vector<int> row(n,0);
        vector<int> column(m,0);
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
            if(matrix[i][j]==0)
            {
                row[i]=1;
                column[j]=1;
            }
            }

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

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

    }
};
```

3) OPTIMIZED APPROACH:

```
class Solution {
```

```
public:
    void setZeroes(vector<vector<int>>& matrix) {
        //optimized soln:
        int col0=1;
        int n=matrix.size();
        int m=matrix[0].size();
        //step1: mark the row and column
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                if(matrix[i][j]==0)
                {
                    matrix[i][0]=0;
                    if(j!=0)
                    matrix[0][j]=0;
                    else
                    col0=0;
                }
            }
        }
        //step2: mark the elements from row 1 and column 1
        for(int i=1;i<n;i++)
        {
            for(int j=1;j<m;j++)
            {
                if(matrix[i][0]==0||matrix[0][j]==0)
                {
                    matrix[i][j]=0;
                }
```

```
            }
        }
        //step3: mark the first column followed by the first row:
        //to mark column1 which corresponds to the 1st row
        if(matrix[0][0]==0)
        {
            for(int j=0;j<m;j++)
            {
                matrix[0][j]=0;
            }
        }
        //to mark row1 which corresponds to the 1st column
        if(col0==0)
        {
            for(int i=0;i<n;i++)
            {
                matrix[i][0]=0;
            }
        }

    }
};
```

18) Rotate a matrix clockwise

1) BRUTE FORCE Using temp matrix:

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        //using another temp matrix
        int n=matrix.size();
        vector<vector<int>> temp(n,vector<int>(n,0));
```

```
        int tempy=n-1;
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                temp[j][i]=matrix[tempy][j];
            }
            tempy--;
        }
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                matrix[i][j]=temp[i][j];
            }
        }

    }
};
```

2) OPTIMIZED: By using transpose row reversal:

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
      //transpose row reversal optimized approach:
      int n=matrix.size();
      //step1: find the transpose of the matrix:
      for(int i=0;i<n;i++)
      {
        for(int j=0;j<i;j++)
        {
        swap(matrix[i][j],matrix[j][i]);
        }
```

```
    }
    //step2: reverse each row of the transposed matrix:
     for(int i=0;i<n;i++)
    {
      reverse(matrix[i].begin(),matrix[i].end());
    }

   }
};
```

19) Spiral matrix traversal:

Optimized approach:

```
class Solution {
public:
   vector<int> spiralOrder(vector<vector<int>>& matrix) {
      int n=matrix.size();
      int m=matrix[0].size();
      vector<int> v;
      int left=0;
      int top=0;
      int right=m-1;
      int bottom=n-1;
      while(left<=right&&top<=bottom)
      {
      //1st row:
      for(int j=left;j<=right;j++)
      {
         v.push_back(matrix[top][j]);
      }
       top++;
      //last column:
      for(int i=top;i<=bottom;i++)
      {
         v.push_back(matrix[i][right]);

      }
        right--;
      //last row:
      if(top<=bottom)
      {
      for(int j=right;j>=left;j--)
      {
         v.push_back(matrix[bottom][j]);
      }
      bottom--;
      }
      //1st column:
```

```
    if(left<=right)
    {
    for(int i=bottom;i>=top;i--)
    {
        v.push_back(matrix[i][left]);

    }
     left++;
    }
    }
    return v;
  }
};
```

20) Sum of subarray equals k:

1) Improved Brute Force approach(O(n^2)):

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        //BRUTE FORCE:
        int n=nums.size();
        int sum=0;
        int count=0;
        for(int i=0;i<n;i++)
        {
            sum=0;
            for(int j=i;j<n;j++)
            {
                sum+=nums[j];
                if(sum==k)
                count++;
            }
        }
        return count;
    }
};
```

2) Optimized approach using hashing:

```
class Solution {
public:
  int subarraySum(vector<int>& nums, int k) {
    //hashing:
    int n=nums.size();
    int preSum=0;
    int rem;
```

```
        int count=0;
        map<int,int> mpp;
        mpp[0]=1;
        for(int i=0;i<n;i++)
        {
            preSum+=nums[i];
            rem=preSum-k;
            count+=mpp[rem];
            mpp[preSum]+=1;
        }
        return count;
    }
};
```

22) Pascal's triangle: LC(Variant 3 soln 😊):

BRUTE FORCE APPROACH:

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

class Solution {
public:
    // Function to calculate factorial of a number.
    int factorial(int n) {
        int fact = 1;
        for (int i = 1; i <= n; i++) {
            fact *= i;
        }
        return fact;
    }

    // Function to calculate binomial coefficient C(n, k).
    int generateAns(int col, int row) {
        int rowFact = factorial(row);       // row!
        int colFact = factorial(col);       // col!
        int diffFact = factorial(row - col); // (row - col)!
        return rowFact / (colFact * diffFact);
    }

    // Function to generate a single row using brute force.
    vector<int> generateRow(int row) {
        vector<int> roww;
        for (int col = 0; col <= row; col++) {
            roww.push_back(generateAns(col, row));
        }
        return roww;
```

```
    }

    // Function to generate Pascal's Triangle up to numRows.
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> triangle;
        for (int i = 0; i < numRows; i++) {
            triangle.push_back(generateRow(i));
        }
        return triangle;
    }
};

int main() {
    Solution sol;
    int numRows = 5;
    vector<vector<int>> triangle = sol.generate(numRows);

    // Print Pascal's Triangle.
    for (const auto& row : triangle) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Variant 3 using variant 2(better approach):

```
class Solution {
public:
    //variation 3 using variation 2:
    vector<int> generateRow(int row)
    {
        vector<int> roww;
        int ans=1;
        roww.push_back(1);
        for(int col=1;col<row;col++)
        {
            ans=ans*(row-col);
            ans=ans/col;
            roww.push_back(ans);
        }
        return roww;

    }
    vector<vector<int>> generate(int numRows)
    {
```

```
        vector<vector<int>> v;
        for(int i=1;i<=numRows;i++)
        {
            v.push_back(generateRow(i));
        }
        return v;
    }
};
```

23) Majority element part 2 (>N/3):

OPTIMIZED APPROACH:

```
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
        //for N/3 ADV moore's voting algo:
        int n=nums.size();
        int ele1;
        int ele2;
        int c1=0;
        int c2=0;
        int count1=0;
        int count2=0;
        vector<int> v;
        for(int i=0;i<n;i++)
        {
            if(c1==0&&ele2!=nums[i])
            {
                c1=1;
                ele1=nums[i];
            }
            else if(c2==0&&ele1!=nums[i])
            {
```

```
            c2=1;
            ele2=nums[i];
        }
        else if(nums[i]==ele1)
        {
            c1++;
        }
        else if(nums[i]==ele2)
        c2++;
        else
        {
            c1--;
            c2--;
        }
    }
    for(int i=0;i<n;i++)
    {
        if(nums[i]==ele1)
        count1++;
        else if(nums[i]==ele2)
        count2++;
    }
    if(count1>=(((int)(n/3))+1))
    v.push_back(ele1);
    if(count2>=(((int)(n/3))+1))
    v.push_back(ele2);
    sort(v.begin(),v.end());
    return v;

  }
};
```

24) 3 sum:

1) BRUTE FORCE:

```
class Solution {
public:
   vector<vector<int>> threeSum(vector<int>& nums) {
       int n=nums.size();
      // vector<int> v;
       vector<vector<int>> V;
       set<vector<int>> s;
       for(int i=0;i<n;i++)
       {
           for(int j=i+1;j<n;j++)
           {

               for(int k=j+1;k<n;k++)
               {
                   vector<int> v;
                   if(((nums[i]+nums[j]+nums[k])==0))
                   {
                       v.push_back(nums[i]);
                       v.push_back(nums[j]);
                       v.push_back(nums[k]);
                       sort(v.begin(),v.end());

                   }
                    if(!v.empty())
                     s.insert(v);
               }

           }
       }
       int m=s.size();
       for(auto it:s)
       {
           V.push_back(it);
       }
       return V;
   }
};
```

2) Better Approach:

```
class Solution {
public:
   vector<vector<int>> threeSum(vector<int>& nums) {
      //better approach: hashmap approach O(n^2):
      int n=nums.size();
      set<vector<int>> s; //set s is used to store the unique triplets
      for(int i=0;i<n;i++)
      {
         set<int> hashset;  //to store third ele
         for(int j=i+1;j<n;j++)
         {
            int third=-(nums[i]+nums[j]);
            if(hashset.find(third)!=hashset.end())  //triplet exists
            {
               vector<int> v; //to store each triplet
               v={nums[i],nums[j],third};
               sort(v.begin(),v.end());
               s.insert(v);
            }
            hashset.insert(nums[j]);
         }
      }
      vector<vector<int>> V(s.begin(),s.end());
      return V;
   }
};
```

3) Optimized approach(modified 2 pointers):

```
class Solution {
public:
   vector<vector<int>> threeSum(vector<int>& nums) {
      //optimized approach:
      int n=nums.size();
      sort(nums.begin(),nums.end());
      int sum;
      vector<vector<int>> V;
      int j=-1;
      int k=-1;
```

```
        if(n<3)
        {
            return V;
        }
        for(int i=0;i<n;i++)
        {
            if(i>0)
            {
                if(nums[i]==nums[i-1])
                {
                    continue;
                }
            }
            j=i+1;
            k=n-1;
            while(j<k)
            {
                sum=nums[i]+nums[j]+nums[k];
                if(sum<0)
                j++;
                else if(sum>0)
                k--;
                else
                {
                    vector<int> v={nums[i],nums[j],nums[k]};
                    V.push_back(v);
                    j++;
                    k--;
                    while(j<k&&nums[j]==nums[j-1])
                    {
                        j++;
```

```
            }
                while(j<k&&nums[k]==nums[k+1])
            {
                k--;
            }
          }


        }
      }
      return V;
   }
};



//difference between continue and manual increment:
#include <iostream>
using namespace std;
int main()
{
for (int i = 0; i < 5; i++) {
   cout<<i<<" ";
   if (i == 2) {
      i+=2;  // Manually increment
   }
   cout << i << " ";
}
cout<<" "<<endl;
for (int i = 0; i < 5; i++) {
   cout<<i<<" ";
   if (i == 2) {
      //i+=2;  // Manually increment
```

```
        continue;
    }
    cout << i << " ";
}
    return 0;
}
```

25) Largest subarray with 0 sum:

1) ABSOLUTE BRUTE FORCE O(n^3) :

```
class Solution {
  public:
    int maxLen(vector<int>& arr) {
        // code here
        //abs Brute Force using 3 loops:
        int sum=0;
        int len=INT_MIN;
        int n=arr.size();
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                sum=0;
                for(int k=i;k<=j;k++)
                {
                  sum+=arr[k];

                }
                if(sum==0)
                len=max(len,(j-i+1));
            }

        }
        if(len==INT_MIN)
        len=0;
```

```
        return len;
    }
};
```

2) BETTER BRUTE FORCE(O(n^2) soln):

```
class Solution {
  public:
    int maxLen(vector<int>& arr) {
        // code here
        //better Brute Force using 2 loops:
        int n=arr.size();
        int sum;
        int len=INT_MIN;
        for(int i=0;i<n;i++)
        {
            sum=0;
            for(int j=i;j<n;j++)
            {
                sum+=arr[j];
                if(sum==0)
                len=max(len,(j-i+1));
            }
        }
        if(len==INT_MIN)
        len=0;
        return len;
    }
};
```

3) OPTIMIZED APPROACH USING HASHING:

```
class Solution {
  public:
    int maxLen(vector<int>& arr) {
```

```
        //using hashing:
         int n=arr.size();
        map<int,int> m;
        int len=INT_MIN;
        int sum=0;
        int rem;
        for(int i=0;i<n;i++)
        {
            sum+=arr[i];
            rem=sum;
            if(sum==0)
            {
                len=max(len,(i+1));
            }
            if(m.find(rem)!=m.end())
            {
                len=max(len,(i-m[rem]));

            }
            if(m.find(sum)==m.end())
            {
                m[sum]=i;
            }
        }
        if(len==INT_MIN)
        len=0;
        return len;

    }
};
```

LENGTH OF LONGEST SUBARRAY WITH SUM k:

OPTIMIZED APPROACH:

```
class Solution {
  public:
    int lenOfLongestSubarr(vector<int>& arr, int k) {
        // code here
        int left=0;
        int right=0;
        int sum=arr[0];
        int n=arr.size();
        int temp;
        int maxLen=0;
        while(right<n)
        {
            while(left<=right&&sum>k)
            {
                sum=sum-arr[left];
                left++;
            }

            if(sum==k)
            {
                temp=(right-left+1);
                maxLen=max(temp,maxLen);
            }

            if(right<len)
            {
            sum+=arr[right];
            right++;
            }

        }
        return maxLen;
    }
```

```
};
```

26) Overlapping subintervals:

1)BRUTE FORCE:

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        //BRUTE FORCE:
        sort(intervals.begin(),intervals.end());
      vector<vector<int>> ans;
      int n=intervals.size();
      int start;
      int end;
      for(int i=0;i<n;i++)
      {
          start=intervals[i][0];
          end=intervals[i][1];
          if(!ans.empty()&&(end<=ans.back()[1]))
          {
              continue;
          }
          else
          {
              for(int j=i+1;j<n;j++)
              {
                  if(intervals[j][0]<=end)
                  end=max(end,intervals[j][1]);
                  else
                  break;
              }
          }
          ans.push_back({start,end});
```

```
        }
        return ans;


    }
};
```

2) OPTIMIZED APPROACH:

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        //Optimized approach:
        int n=intervals.size();
        sort(intervals.begin(),intervals.end());
        vector<vector<int>> ans;
        for(int i=0;i<n;i++)
        {
            //creation of a new interval:
            if(ans.empty()||intervals[i][0]>ans.back()[1])
            {
                ans.push_back(intervals[i]);
            }
            //merge the intervals:
            else if(intervals[i][0]<=ans.back()[1])
            {
                ans.back()[1]=max(ans.back()[1],intervals[i][1]);
            }
        }
        return ans;
    }
};
```

27) MERGE SORTED ARRAY:

BRUTE FORCE:

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int temp=0;
       for(int i=m;i<(m+n);i++)
       {
           nums1[i]=nums2[temp];
           temp++;
       }
       sort(nums1.begin(),nums1.end());

    }
};
```

STRIVER Q BRUTE FORCE:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   vector<int> arr1={1,3,5,7};
   vector<int> arr2={0,2,6,8,9};
   int m=arr1.size();
   int n=arr2.size();
   int i=0;
   int j=0;
   //vector<int> V(m+n);
   vector<int> V;
   while(i<m&&j<n)
   {
       if(arr1[i]<arr2[j])
       {
```

```
            V.push_back(arr1[i]);
            i++;
        }
        else
        {
            V.push_back(arr2[j]);
            j++;
        }
    }
    while(i<m)
    {
    V.push_back(arr1[i]);
    i++;
    }
    while(j<n)
    {
    V.push_back(arr2[j]);
    j++;
    }
    j=0;
    for(int i=0;i<(m+n);i++)
    {
        if(i<m)
        arr1[i]=V[i];
        else
        {
        arr2[j]=V[i];
        j++;
        }
    }
    for(int i=0;i<m;i++)
    cout<<arr1[i]<<" ";
    cout<<" "<<endl;
```

```
    for(int j=0;j<n;j++)
    cout<<arr2[j]<<" ";
    cout<<" "<<endl;
    return 0;
}
```

GFG BRUTE FORCE SOLN(SAME AS STRIVER):

```
class Solution {

  public:
    void mergeArrays(vector<int>& a, vector<int>& b)
{
        int m=a.size();
    int n=b.size();
    int i=0;
    int j=0;
    vector<int> V;
    while(i<m&&j<n)
    {
        if(a[i]<b[j])
        {
            V.push_back(a[i]);
            i++;
        }
        else
        {
            V.push_back(b[j]);
            j++;
        }
    }
    while(i<m)
    {
    V.push_back(a[i]);
    i++;
```

```
    }
    while(j<n)
    {
    V.push_back(b[j]);
    j++;
    }
    j=0;
    for(int i=0;i<(m+n);i++)
    {
        if(i<m)
        a[i]=V[i];
        else
        {
        b[j]=V[i];
        j++;
        }
    }
    }
};
```

GFG OPTIMIZED SOLN1 (SAME AS STRIVER):

```
class Solution {

    public:
      void mergeArrays(vector<int>& a, vector<int>& b) {
        // code here
        //optimized approach 1:
        int n=a.size();
        int m=b.size();
        int i1=n-1;
        int i2=0;
        while(i1>=0&&i2<m)
        {
            if(a[i1]>b[i2])
            {
```

```
                swap(a[i1],b[i2]);
            }
            i1--;
            i2++;
        }
        sort(a.begin(),a.end());
        sort(b.begin(),b.end());
    }
};
```

MAX SUBARRAY PRODUCT:

OPTIMIZED APPROACH LC:

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int n=nums.size();
        int pre=1;
        int suf=1;
        int ans=INT_MIN;
        for(int i=0;i<n;i++)
        {
            if(pre==0)
            pre=1;
            if(suf==0)
            suf=1;
            pre=pre*nums[i];
            suf=suf*nums[n-i-1];
            ans=max(ans,max(pre,suf));
        }
        return ans;
    }
};
```

STRINGS:

1) REMOVE OUTERMOST PARANTHESES:

A) STACK APPROACH:

```
B) class Solution {
C) public:
D)     string removeOuterParentheses(string s) {
E)         //stack approach:
F)         int n=s.size();
G)         stack<char> st;
H)         string str;
I)         char ch;
J)         char chr;
K)         for(int i=0;i<n;i++)
L)         {
M)             ch=s[i];
N)             if(ch=='(')
O)             {
P)                 if(st.empty())
Q)                 {
R)                     st.push(ch);
S)                 }
T)                 else
U)                 {
V)                     st.push(ch);
W)                     str+=ch;
X)                 }
Y)             }
Z)             else if(ch==')')
AA)            {
BB)                if(!st.empty())
CC)                {
DD)                st.pop();
EE)                }
FF)                if(!st.empty())
GG)                {
HH)                    str+=ch;
II)                }
JJ)            }
KK)        }
LL)        return str;
MM)    }
NN)};
```

B)COUNTING APPROACH(OPTIMIZED APPROACH):

```
2) class Solution {
3) public:
4)     string removeOuterParentheses(string s) {
5)         int n=s.size();
6)         int count=0;
7)         string str="";
8)         for(int i=0;i<n;i++)
9)         {
10)            if(s[i]=='(')
11)            {
12)                count++;
13)                if(count>1)
14)                {
15)                    str+="(";
16)                }
17)            }
18)            else if(s[i]==')')
19)            {
20)                count--;
21)                if(count>0)
22)                {
23)                    str+=")";
24)                }
25)            }
26)        }
27)        return str;
28)    }
29)};
```

2) REVERSE THE ORDER OF WORDS IN A STRING:

BRUTE FORCE APPROACH USING A STACK:

```
class Solution {

public:

    string reverseWords(string s) {

        string str="";

        int n=s.size();

        s+=' ';

        stack<string> st;

        string stri="";

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

        {

            if(s[i]!=' ')

            {

                str+=s[i];

            }

            else if(s[i]==' ')

            {

                if(str.size()>0)

                st.push(str);

                str="";

            }

        }

        //until stack becomes empty keep popping elements
```

```
        //then add popped elements to a string
        while(st.size()>1)
        {
            stri+=st.top()+' ';
            st.pop();
        }
        stri+=st.top();
        return stri;
    }
};
```

OPTIMIZED APPROACH LC SOLN(CHATGPT):

```
class Solution {
public:
    string reverseWords(string s) {
        s+=" ";
        string ans="";
        string temp="";
        int n=s.size();
        int left=0;
        while(left<n)
        {
            if(s[left]!=' ')
            {
```

```
            temp+=s[left];
        }
        else if(s[left]==' '&&!temp.empty())
        {
            if(ans.empty())
            {
                ans=temp;
            }
            else
            {
                ans=temp+' '+ans;
            }
            temp="";
        }
        left++;
      }
      return ans;
   }
};
```

3)LARGEST ODD NUMBER IN A STRING:

WORKS FOR many CASES BUT NOT ALL DUE TO INT LIMIT EXCEEDED

1.BRUTE FORCE:

```
class Solution {
public:
   string largestOddNumber(string num) {
```

```
        string lOdd;
        int n=num.size();
        int m=INT_MIN;
        string str;
        int numm;
        for(int i=0;i<n;i++)
        {
            str="";
            for(int j=0;j<=i;j++)
            {
                str+=num[j];
            }
            numm=stoi(str);
            if(numm%2!=0)
            {
                m=max(numm,m);
            }
        }
        if(m==INT_MIN)
        {
            lOdd="";
        }
        else
        lOdd=to_string(m);
```

```
        return lOdd;
    }
};
```

2.OPTIMIZED APPROACH:

```
class Solution {
public:
    string largestOddNumber(string num) {
        //OPTIMIZED APPROACH:
        int n=num.size();
        char ch;
        int d;
        string lOdd="";
        for(int i=n-1;i>=0;i--)
        {
            ch=num[i];
            d=ch-'0';
            if(d%2!=0)
                {  lOdd=num.substr(0,i+1);
            break;}
        }
        return lOdd;
    }
};
```

4)LONGEST COMMON PREFIX:

BRUTE FORCE:

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        //BRUTE FORCE:
        string cpre;   //to store the common prefix
        int n=strs.size(); //to find the no.of strings in the list
        string cp="";
        string fincp="";
        int flag=0;
        int m=INT_MAX;
        //find the length of the shortest string:
        for(int i=0;i<n;i++)
        {
            if(strs[i].size()<m)
            m=strs[i].size();
        }
        for(int i=0;i<m;i++)
        {
            cpre=strs[0].substr(0,i+1);
            flag=0;
            for(int j=1;j<n;j++)
            {
                cp=strs[j].substr(0,i+1);
```

```
            if(cp!=cpre)
            {
            flag=1;
            break;
            }
        }
        if(flag==0)
        {
            fincp=cpre;
        }
      }
      return fincp;
   }
};
```

OPTIMIZED APPROACH BY SORTING:

```
class Solution {
public:
   string longestCommonPrefix(vector<string>& strs)
   {
     sort(strs.begin(),strs.end());
     int n=strs.size();
     string lpre="";
     //now compare 1st and last words
     //first find min length among the 2 words:
```

```
    int mini=min(strs[0].size(),strs[n-1].size());
    for(int i=0;i<mini;i++)
    {
    // if(strs[0][i]==strs[1][i])
     if(strs[0].substr(0,i+1)==strs[n-1].substr(0,i+1))
     {
        lpre=strs[0].substr(0,i+1);
     }
     else
     break;
    }
    return lpre;
  }
};
```

MORE OPTIMIZED APPROACH(SORTING WITHOUT substr):

```
class Solution {
public:
  string longestCommonPrefix(vector<string>& strs)
  {
    sort(strs.begin(),strs.end());
    int n=strs.size();
    string lpre="";
    if(n==1)
    {
```

```
        return strs[0];
      }
      int mini=min(strs[0].size(),strs[n-1].size());
      for(int i=0;i<mini;i++)
      {
      if(strs[0][i]==strs[n-1][i])
       {
           lpre+=strs[0][i];
       }
       else
       break;
      }
      return lpre;
    }
};
```

5)ISOMORPHIC STRINGS:

1)BRUTE FORCE APPROACH USING HASHMAP:

```
class Solution {
public:
    bool isIsomorphic(string s,string t) {
        //hashmap app:
        map<char,char> m1;
       map<char,char> m2;
       int l1=s.size();
```

```
        int l2=t.size();
        if(l1!=l2)
        return false;
        char ch1;
        char ch2;
        for(int i=0;i<l1;i++)
        {
            ch1=s[i];
            ch2=t[i];
            if(m1.count(ch1))
            { if(m1[ch1]!=ch2)
                return false;
            }
            else
            {
                m1[ch1]=ch2;
            }
            if(m2.count(ch2))
            {
                if(m2[ch2]!=ch1)
                return false;
            }
            else
            {
                m2[ch2]=ch1;
```

```
        }
    }
    return true;
  }
};
```

2) BETTER APPROACH(USING HASHING/HASHARRAY):

```
3) class Solution {
4) public:
5)     bool isIsomorphic(string s,string t) {
6)         //hashing or hasharray approach:
7)         int l1=s.size();
8)         int l2=t.size();
9)         if(l1!=l2)
10)        return false;
11)        vector<int> v1(256,-1);
12)        vector<int> v2(256,-1);
13)        char ch1;
14)        char ch2;
15)        for(int i=0;i<l1;i++)
16)        {
17)            ch1=s[i];
18)            ch2=t[i];
19)            if(v1[ch1]==-1)
20)            {
21)                v1[ch1]=ch2;
22)            }
23)            else
24)            {
25)                if(v1[ch1]!=ch2)
26)                return false;
27)            }
28)            if(v2[ch2]==-1)
29)            {
30)                v2[ch2]=ch1;
31)            }
32)            else
33)            {
34)                if(v2[ch2]!=ch1)
35)                return false;
36)            }
37)        }
38)        return true;
39)    }
40)};
```

6) CHECK IF A STRING IS ROTATION OF ANOTHER STRING:

1)BRUTE FORCE USING substr

```
class Solution {

public:

    bool rotateString(string s, string goal) {

        int l1=s.size();

        int l2=goal.size();

        if(s==goal)

        return true;

        if(l1!=l2)

        return false;

        //try the below code without substr function:

        string str1;

        string str2;

        string str;

        for(int i=0;i<l1-1;i++)

        {

            str1=s.substr(0,i+1);

            str2=s.substr(i+1,l1);

            str=str2+str1;

            if(str==goal)

            return true;

        }

        return false; }
```

```
};
```

TC: O(N^2)

SC: O(N) due to substr method

2)BETTER APPROACH without substr method:

```
class Solution {
public:
    bool rotateString(string s, string goal)
    {
        int l1=s.size();
        int l2=goal.size();
        if(s==goal)
        return true;
        if(l1!=l2)
        return false;
        char first;
        for(int i=0;i<l1;i++)
        {
            first=s[0];  //recheck
            for(int j=0;j<l1-1;j++)
            {
                s[j]=s[j+1];
            }
            s[l1-1]=first;
            if(s==goal)
            return true;
```

```
        }
        return false;
    }
};
```

TC: O(N^2)

SC: O(1) due to in-place rotation

3) OPTIMIZED SIMPLEST APPROACH:

```
class Solution {
public:
    bool rotateString(string s, string goal)
    {
        int l1=s.size();
        int l2=goal.size();
//impo edge case
        if(l1!=l2)
        {
            return false;
        }
        //optimized approach:
        string ds=s+s;
        return(ds.find(goal)!=std::string::npos);
    }
};
```

7)CHECK IF 2 WORDS ARE ANAGRAMS:

1) HASHMAP APPROACH:

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        int l1=s.size();
        int l2=t.size();
        map<int,int> m1;
        map<int,int> m2;
        if(l1!=l2)
        return false;
        for(int i=0;i<l1;i++)
        {
            m1[s[i]]++;
            m2[t[i]]++;
        }
        for(char i='a';i<='z';i++)
        {
            if(m1[i]!=m2[i])
            return false;
        }
        return true;
    }
};
```

2) SORTING APPROACH:

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        int l1=s.size();
        int l2=t.size();
        if(l1!=l2)
        return false;
        //sorting approach:
        sort(s.begin(),s.end());
        sort(t.begin(),t.end());
        for(int i=0;i<l1;i++)
        {
            if(s[i]!=t[i])
            return false;
        }
        return true;
    }
};
```

3)HASHARRAY OPTIMIZED APPROACH:

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        int l1=s.size();
        int l2=t.size();
        if(l1!=l2)
        return false;
        int count[26]={0};
        for(int i=0;i<l1;i++)
        {
            count[s[i]-'a']++;
            count[t[i]-'a']--;
        }
        for(int i=0;i<26;i++)
        {
            if(count[i]>0)
            {
                return false;
            }
        }
        return true;
    }
};
```

8) SORTING A STRING BASED ON FREQUENCY OF CHARACTERS:

1)BRUTE FORCE CODE:

```
class Solution {
public:
    string frequencySort(string s) {
        string str="";
        //1.create a hashmap
        unordered_map<char,int> m;
        for(char c:s)
        {
            m[c]++;
        }
        //2.creating a vector of pairs:
        vector<pair<char,int>> v(m.begin(),m.end());
        //3.sort the vector in desc order using lambda func:
        sort(v.begin(),v.end(),[](pair<char,int>& a,pair<char,int>& b)
        {
            return a.second>b.second;
        }
        );
        //4.reconstruct the freq sorted string:
        for(auto pair:v)
        {
            str+=string(pair.second,pair.first);
        }
```

```
        return str;
    }
};
```

2) OPTIMIZED APPROACH: BUCKET SORT:

```
class Solution {
public:
    string frequencySort(string s) {
        string res="";
        int n=0;
        char ch;
        int freq;
        //create a map of frequencies
        map<char,int> m;
        for(char c:s)
        {
            m[c]++;
        }
        int size=m.size();
        //find the max freq
        for(auto& pair:m)
        n =max(n,pair.second);
        //create the bucket (vector of strings):
        vector<string> buckets(n+1);
        //filling the bucket from map values
```

```
        for(auto& pair:m)
        {
            ch=pair.first;
            freq=pair.second;
            buckets[freq]+=string(freq,ch);
        }
        //building the res string from bucket vector:
        for(int i=n;i>0;i--)
        {
            res+=buckets[i];
        }
        return res;
    }
};
```

9) MAXIMUM NESTING DEPTH OF PARANTHESES:

1)BRUTE FORCE APPROACH USING STACKS:

```
class Solution {
public:
    int maxDepth(string s) {
        //try using stacks
        int n=s.size();
        char ch;
        stack<char> st;
        size_t m=0;
        for(int i=0;i<n;i++)
```

```
        {
            ch=s[i];
            if(ch=='(')
            {
                st.push(ch);
            }
            else if(ch==')')
            {
                if(!st.empty())
                st.pop();
            }
            m=max(m,st.size());
        }
        return m;
    }
};
```

TC: O(n)

SC: O(n/2) = O(n)

2)OPTIMIZED COUNTING APPROACH:

```
class Solution {
public:
    int maxDepth(string s) {
        //optimized counting approach:
        int count=0;
```

```
        int n=s.size();
        char ch;
        int m=0;
        for(int i=0;i<n;i++)
        {
            ch=s[i];
            if(ch=='(')
            {
                count++;
            }
            else if(ch==')')
            {
                count--;
            }
            m=max(count,m);
        }
        return m;
    }
};
```

TC: O(n)

SC: O(1)

10)ROMAN TO INT:

OPTIMIZED APPROACH:

```
class Solution {
public:
    int value(char c)
    {
        int val;
        switch(c)
        {
            case 'I':
            val=1;
            break;
            case 'V':
            val=5;
            break;
            case 'X':
             val=10;
            break;
            case 'L':
             val=50;
            break;
            case 'C':
             val=100;
            break;
            case 'D':
```

```
            val=500;
            break;
            case 'M':
            val=1000;
            break;
        }
        return val;
    }
    int romanToInt(string s) {
        int n=s.size();
        int ans=0;
        char c;
        for(int i=0;i<n;i++)
        {
            c=s[i];
            //special cases:
            if(i<(n-1)&&value(c)<value(s[i+1]))
                ans-=value(c);
            else
                ans+=value(c);
        }
        return ans;   }
    };
```

TC: O(n)

SC: O(1)

11) STRING TO INTEGER (ATOI):

SOLN:

```
class Solution {
public:
    int myAtoi(string s) {
        int n = s.size();
        int i = 0; // Pointer to traverse the string
        long ans = 0; // Use long to handle overflow temporarily
        int sign = 1; // Default sign is positive
        // 1. Ignore leading whitespace
        while (i < n && s[i] == ' ')
            i++;
        }
        // 2. Check for optional '+' or '-' sign
        if (i < n && (s[i] == '+' || s[i] == '-')) {
            if (s[i] == '-') {
                sign = -1; // Set sign to negative
            }
            i++;
        }
        // 3. Convert digits to integer, stopping at the first non-digit
        while (i < n && isdigit(s[i])) {
            int digit = s[i] - '0';
            // Check for overflow before updating `ans`
```

```
            if (ans > (INT_MAX - digit) / 10) {
                return sign == 1 ? INT_MAX : INT_MIN;
            }
            ans = ans * 10 + digit;
            i++;
        }
        // 4. Apply the sign and return the result
        ans = ans * sign;
        return (int)ans; // Cast to int since the result fits within 32-bit
    }
};
```

12) COUNT THE NUMBER OF SUBSTRINGS:

1) BRUTE FORCE:

```
class Solution {
  public:
    int countSubstr(string& s, int k) {
        //BRUTE FORCE:
        int n=s.size();
        string sub="";
        set<char> st;
        int count=0;
        for(int i=0;i<n;i++)
        {
            sub="";
```

```
            for(int j=i;j<n;j++)
            {
                sub+=s[j];
                //create set here and check:
                for(char c:sub)
                st.insert(c);
                if(st.size()==k)
                count++;
                st.clear();
            }
        }
        return count;
    }
};
```

TC: O(n^3) worst case due to 3 loops

SC: O(n) (for substring storage)

2) BETTER APPROACH:

```
class Solution {
  public:
    int countSubstr(string& s, int k) {
     //code2:
    int n = s.size();
    int count = 0;
    // Iterate over all possible substrings
```

```
    for (int i = 0; i < n; i++) {
        set<char> st;
        for (int j = i; j < n; j++) {
            st.insert(s[j]);  // Add current character to the set
            if (st.size() == k) {
                count++;  // Found a valid substring
            } else if (st.size() > k) {
                break;  // No need to continue if distinct characters exceed k
            }
        }
    }
    return count;
    }
};
```

Time Complexity: O(n^2)

Space Complexity :O(1)

2)OPTIMIZED APPROACH: SLIDING WINDOW+HASHMAP:

```
class Solution {
public:
    int countSubstr(string& s, int k) {
        return countSubstrWithAtmostk(s,k)-countSubstrWithAtmostk(s,k-1);
    }
private:
    int countSubstrWithAtmostk(string &s,int k)
    {
        int count=0;
        int n=s.size();
        map<char,int> m;
        int left=0;
        for(int right=0;right<n;right++)
        {
            m[s[right]]++;
            while(m.size()>k)
            {
                m[s[left]]--;
                if(m[s[left]]==0)
                {
                    m.erase(s[left]);
                }
                left++;
```

```
        }
            count+=(right-left+1);
        }
        return count;
    }
};
```

IMPO:

Time Complexity: O(n)

Space Complexity :O(1)

13)SUM OF BEAUTY OF ALL SUBSTRINGS:

1)HASHMAP SUBSTRING APPROACH:

```
class Solution {
public:
    int beautySum(string s) {
        int totB=0;
        int n=s.size();
        for(int i=0;i<n;i++)
        {
            vector<int> freq(26,0);
            for(int j=i;j<n;j++)
            {
                freq[s[j]-'a']++;
                int mini=INT_MAX;
                int maxi=0;
                for(int f:freq)
                {
                    if(f>0)
                    {
                        mini=min(mini,f);
                        maxi=max(maxi,f);
                    }
                }
                totB+=(maxi-mini);
```

```
        }
      }
      return totB;
    }
};
```

TC: O(n^2)

SC: O(1)

LL: new

1.MIDDLE OF LL:

1) BRUTE FORCE : clean code:

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
      //BRUTE FORCE approach:
      ListNode* temp=head;
      int count=0;
      int mid;
      int count2=0;
      while(temp!=nullptr)
      {
       count++;
       temp=temp->next;
      }
```

```
        temp=head;
        mid=(count/2)+1;
        while(temp!=nullptr)
        {
         count2++;
         if(count2==mid)
         {
             break;
         }
         temp=temp->next;
        }
        return temp;
    }
};
```

2)OPTIMIZED APPROACH: TORTOISE HARE ALGO:

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        //optimized approach:tortoise and hare algo:
        ListNode* fast=head;
        ListNode* slow=head;
        while(fast!=nullptr&&fast->next!=nullptr)
        {
         slow=slow->next;
         fast=fast->next->next;
```

```
        }
        return slow;
    }
};
```

2)REVERSE A LL(ITERATIVE):

BRUTE FORCE:

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
   ListNode* temp = head;
   ListNode* prev = NULL;
   while(temp != NULL){
        ListNode* front = temp->next;
        temp->next = prev;
        prev = temp;
        temp = front;
   }
   return prev;
    }
};
```

2)OPTIMIZED ITERATIVE APPROACH:

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev=nullptr;
        ListNode* cur=head;
        ListNode* temp=nullptr;
        if(cur!=nullptr)
        temp=cur->next;;
        while(cur!=nullptr)
        {
            //temp=cur->next;
            cur->next=prev;
            prev=cur;
            cur=temp;
            if(temp!=nullptr)
            temp=temp->next;

        }
        return prev;
    }
};
```

3)

```
class Solution {
```

```
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* newhead;
        ListNode* front;
       if(head==nullptr||head->next==nullptr)
       return head;
       else
       {
        newhead=reverseList(head->next);
        front=head->next;
        head->next=nullptr;
        front->next=head;
        return newhead;
       }
    }
};
```

3)DETECT CYCLE IN LL:

1)BRUTE FORCE USING MAP:

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode* temp=head;
        map<ListNode*,int> m;
        while(temp!=nullptr)
        {
```

```
            if(m.find(temp)==m.end())
            m[temp]=1;
            else
            return true;
            temp=temp->next;
        }
        return false;
    }
};
```

4)FIND STARTING POINT OF LL LOOP:

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode* temp=head;
        map<ListNode*,int> m;
        while(temp!=nullptr)
        {
            if(m.find(temp)==m.end())
            m[temp]=1;
            else
            return temp;
            temp=temp->next;
        }
        return nullptr;
```

```
    }
};
```

LC 500: STRINGS Q: -

```
class Solution {
public:
    vector<string> findWords(vector<string>& words) {
    unordered_set<char> row1 = {'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p'};
    unordered_set<char> row2 = {'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l'};
    unordered_set<char> row3 = {'z', 'x', 'c', 'v', 'b', 'n', 'm'};
    vector<string> ans;
    for(auto &word:words)
    {
        set<int> res;
        string temp=word;
        for(auto &c:word)
        {
            c=tolower(c);
            if(row1.find(c)!=row1.end())
            {
                res.insert(1);
            }
            if(row2.find(c)!=row2.end())
            {
                res.insert(2);
```

```
        }
            if(row3.find(c)!=row3.end())
        {
            res.insert(3);
        }
      }
      if(res.size()==1)
      {
          ans.push_back(temp);
      }
   }
   return ans;
   }
};
```

LL CONTD:

5) LL:SEGREGATE ODD AND EVEN NODES:

BRUTE FORCE:

```
class Solution {
public:
   ListNode* oddEvenList(ListNode* head) {
      //BRUTE FORCE approach:
      vector<int> v;
      ListNode* temp=head;
      if(temp==nullptr)
      return temp;
```

```
    ListNode* temp2=head->next;
    while(temp!=nullptr&&temp->next!=nullptr)
    {
        v.push_back(temp->val);
        temp=temp->next->next;
    }
    if(temp!=nullptr)  //check for even no.of ele
    v.push_back(temp->val);
    temp=head->next;
    while(temp!=nullptr&&temp->next!=nullptr) //check
    {
        v.push_back(temp->val);
        temp=temp->next->next;
    }
     if(temp!=nullptr)  //check
    v.push_back(temp->val);
    temp=head;
    int t=0;
    while(temp!=nullptr)
    {
        temp->val=v[t];
        t++;
        temp=temp->next;
    }

```

```
    return head;
    }
};
```

OPTIMIZED APPROACH:

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        //optimized approach:
        if(head==nullptr)
        return head;
        ListNode* odd=head;
        ListNode* even=odd->next;
        ListNode* evenHead=even;
        while(even!=nullptr&&even->next!=nullptr)
        {
            odd->next=odd->next->next;
            even->next=even->next->next;
            odd=odd->next;
            even=even->next;
        }
        odd->next=evenHead;
        return head;
    }
};
```

6) REMOVE Nth NODE FROM THE END:

BRUTE FORCE:

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        //bruteforce:
        int count=0;
        int count2=0;
        ListNode* temp=head;
        ListNode* q=head;
        while(temp!=nullptr)
        {
            temp=temp->next;
            count++;
        }
        temp=head;
        ListNode* p=nullptr;
        for(int i=0;i<count;i++)
        {
            count2++;
            if(count2==(count-n+1))
            {
            if(p!=nullptr)
```

```
        {
        p->next=temp->next;
        delete(temp);
        return head;
        }
        else   //head node deletion case:
        {
            head=head->next;
            delete(q);
            return head;
        }
        }
        p=temp;
        temp=temp->next;
      }
      return head;
   }
};
```

OPTIMIZED APPROACH:

```
class Solution {
public:
   ListNode* removeNthFromEnd(ListNode* head, int n) {
     //optimized approach:
     ListNode* slow=head;
```

```
        ListNode* fast=head;
        for(int i=0;i<n;i++)
        {
         fast=fast->next;
        }
        if(fast==nullptr)  //head deletion case
        {
         return head->next;
        }
        while(fast->next!=nullptr)
        {
         slow=slow->next;
         fast=fast->next;
        }
        ListNode* delNode=slow->next;
        slow->next=slow->next->next;
        delete(delNode);
        return head;
    }
};
```

7) DELETE THE MIDDLE NODE OF LL:

BRUTE FORCE APPROACH:

```
class Solution {
public:
```

```
ListNode* deleteMiddle(ListNode* head) {
    //Brute Force
    ListNode* temp=head;
    int count=0;
    while(temp!=nullptr)
    {
        temp=temp->next;
        count++;
    }
    int mid=count/2;  //assume 0 based indexing
    int count2=-1;
    temp=head;
    ListNode* p=nullptr;
    //spl case when middle mode is the head node
    for(int i=0;i<count;i++)
    {
        count2++;

        if(count2==mid)
        {
            if(p!=nullptr)
            {
                p->next=temp->next;
                delete(temp);
                return head;
```

```
            }
            else
            {
                return head->next;
            }
        }
        p=temp;
        temp=temp->next;
    }
    return head;
  }
};
```

OPTIMIZED APPROACH:

```
class Solution {
public:
  ListNode* deleteMiddle(ListNode* head) {
    //optimized approach:
    ListNode* slow=head;
    ListNode* fast=head;
    ListNode* temp=nullptr;
    if(head->next==nullptr)
    return nullptr;
    while(fast!=nullptr&&fast->next!=nullptr)
    {
```

```
        temp=slow;
        slow=slow->next;
        fast=fast->next->next;
      }
      temp->next=temp->next->next;
      delete(slow);
      return head;
    }
};
```

8) MERGE 2 SORTED LISTS:

NOTE : FOR DUMMY NODE APPROACH USE POINTER APPROACH

1)BRUTE FORCE: *uses additional LL and additional vector*

```
class Solution {
public:
   ListNode* mergeTwoLists(ListNode* list1, ListNode* list2)
   {
      vector<int> v;
      ListNode* t1=list1;
      ListNode* t2=list2;
      while(t1!=nullptr)
      {
          v.push_back(t1->val);
          t1=t1->next;
      }
```

```
        while(t2!=nullptr)
        {
            v.push_back(t2->val);
            t2=t2->next;
        }
        sort(v.begin(),v.end());
        ListNode dummy;
        ListNode* temp=&dummy;
        for(auto val:v)
        {
            temp->next=new ListNode(val);
            temp=temp->next;
        }
        return dummy.next;
    }
};
```

2)BETTER APPRAOCH: *uses additional LL*

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2)
    {
        //BETTER APPROACH : using only LL extra space:
        ListNode* t1=list1;
        ListNode* t2=list2;
```

```
    ListNode dummy;
    ListNode* t=&dummy;
    while(t1!=nullptr&&t2!=nullptr)
    {
        if(t1->val<t2->val)
        {
            t->next=new ListNode(t1->val);
            t1=t1->next;
        }
        else if(t1->val>t2->val)
        {
            t->next=new ListNode(t2->val);
            t2=t2->next;
        }
        else
        {
            t->next = new ListNode(t1->val);
            t = t->next;
            t->next = new ListNode(t2->val);
            t1 = t1->next;
            t2 = t2->next;
        }
        t=t->next;
    }
    if(t1!=nullptr)
```

```
        {
            t->next=t1;
        }
        else if(t2!=nullptr)
        {
            t->next=t2;
        }
        return dummy.next;
    }
};
```

2) ALTERNATIVE BETTER APPROACH USING POINTERS:

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2)
    {
        //BETTER APPROACH : using only LL extra space:
        ListNode* t1=list1;
        ListNode* t2=list2;
        ListNode* dummy=new ListNode(-1);
        ListNode* t=dummy;
        while(t1!=nullptr&&t2!=nullptr)
        {
            if(t1->val<t2->val)
            {
```

```
            t->next=new ListNode(t1->val);
            t1=t1->next;
        }
        else if(t1->val>t2->val)
        {
            t->next=new ListNode(t2->val);
            t2=t2->next;
        }
        else
        {
            t->next = new ListNode(t1->val);
            t = t->next;
            t->next = new ListNode(t2->val);
            t1 = t1->next;
            t2 = t2->next;
        }
        t=t->next;
    }
    if(t1!=nullptr)
    {
        t->next=t1;
    }
    else if(t2!=nullptr)
    {
        t->next=t2;
```

```
        }
        return dummy->next;
    }
};
```

3)OPTIMIZED APPROACH: *no extra space*

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2)
    {
        //OPTIMIZED APPROACH :
        ListNode* dummy=new ListNode(-1);
        ListNode* t=dummy;
        ListNode* t1=list1;
        ListNode* t2=list2;
        while(t1!=nullptr&&t2!=nullptr)
        {
        if(t1->val<t2->val)
        {
            t->next=t1;
            t1=t1->next;
            t=t->next;
        }
        else //impo: works for equal case as well :)
        {
```

```
            t->next=t2;
            t2=t2->next;
            t=t->next;
        }
        }
        //for remaining nodes:
        if(t1!=nullptr)
        {
            t->next=t1;
        }
        else if(t2!=nullptr)
        {
            t->next=t2;
        }
        return dummy->next;
    }
};
```

9) SORT A LL USING MERGE SORT:

```
class Solution {
public:
    //optimized approach using merge sort:
    ListNode* mergeII(ListNode* left,ListNode* right)
    {
        //OPTIMIZED APPROACH :
```

```
    ListNode* dummy=new ListNode(-1);
    ListNode* t=dummy;
    ListNode* t1=left;
    ListNode* t2=right;
    while(t1!=nullptr&&t2!=nullptr)
    {
    if(t1->val<t2->val)
    {
        t->next=t1;
        t1=t1->next;
        t=t->next;
    }
    else //impo: works for equal case as well :)
    {
        t->next=t2;
        t2=t2->next;
        t=t->next;
    }
    }
    //for remaining nodes:
    if(t1!=nullptr)
    {
        t->next=t1;
    }
    else if(t2!=nullptr)
```

```
        {
            t->next=t2;
        }
        return dummy->next;
    }
    ListNode* findMiddle(ListNode* head)
    {
        if (head == nullptr)
        return head;
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast != nullptr && fast->next != nullptr)
        {
        slow = slow->next;
        fast = fast->next->next;
        }
        return slow;
    }
    ListNode* sortList(ListNode* head)
    {
        if (head==nullptr||head->next == nullptr) //base case for recursion
        {
        return head;
        }
        ListNode* middle=findMiddle(head);
```

```
        ListNode* right=middle->next;
        middle->next=nullptr;
        ListNode* left=head;
        left=sortList(left);
        right=sortList(right);
        return mergell(left,right);
    }
};
```

10) SORT 0s,1s and 2s:

BRUTE FORCE:

```
class Solution {
  public:
    // Function to sort a linked list of 0s, 1s and 2s.
    Node* segregate(Node* head) {
        int count0=0;
        int count1=0;
        int count2=0;
        Node* temp=head;
        while(temp!=nullptr)
        {
            if(temp->data==0)
            count0++;
            else if(temp->data==1)
            count1++;
```

```
        else
        count2++;
        temp=temp->next;
    }
    temp=head;
    for(int i=0;i<count0;i++)
    {
        temp->data=0;
        temp=temp->next;
    }
    for(int i=0;i<count1;i++)
    {
        temp->data=1;
        temp=temp->next;
    }
    for(int i=0;i<count2;i++)
    {
        temp->data=2;
        temp=temp->next;
    }
    return head;
  }
};
```

OPTIMIZED APPROACH:

```
class Solution {
  public:
    Node* segregate(Node* head) {
       Node* zeroHead=new Node(-1);
       Node* oneHead=new Node(-1);
       Node* twoHead=new Node(-1);
       Node* zero=zeroHead;
       Node* one=oneHead;
       Node* two=twoHead;
       Node* temp=head;
       if(temp==nullptr||temp->next==nullptr)
       return temp;
       while(temp!=nullptr)
       {
           if(temp->data==0)
           {
               zero->next=temp;
               zero=temp;
               temp=temp->next;
           }
           else if(temp->data==1)
           {
               one->next=temp;
```

```
            one=temp;
            temp=temp->next;
        }
        else if(temp->data==2)
        {
            two->next=temp;
            two=temp;
            temp=temp->next;
        }
    }
    zero->next = (oneHead->next != nullptr) ? oneHead->next : twoHead->next;
    one->next = twoHead->next;
    two->next=nullptr;
    Node* newHead=zeroHead->next;
    delete(zeroHead);
    delete(oneHead);
    delete(twoHead);
    return newHead;
    }
};
```

11) INTERSECTION POINT OF Y LL:

1)ABS BRUTE FORCE:

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        //BRUTE FORCE:
        ListNode* t1=headA;
        ListNode* t2;
        ListNode* temp;
        while(t1!=nullptr)
        {
            t2=headB;
            while(t2!=nullptr)
            {
                if(t1==t2)
                return t1;
                t2=t2->next;
            }
            t1=t1->next;
        }
        return nullptr;
    }
};
```

2) BETTER BRUTE FORCE: using hashmap:

```
class Solution {
public:
   ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
      //best soln using hash
      map<ListNode*,int> m;
      ListNode* t1=headA;
      ListNode* t2=headB;
      while(t1!=nullptr)
      {
          m[t1]++;
          t1=t1->next;
      }
      while(t2!=nullptr)
      {
          if(m.find(t2)!=m.end())
          return t2;
          t2=t2->next;
      }
     return nullptr;
   }
};
```

TC AND SC:

Approach	Time Complexity	Space Complexity
Ordered `map`	O((m + n) log m)	O(m)
Unordered `map`	O(m + n) (on average)	O(m)

3)BETTER APPROACH: using difference in lengths:

```
class Solution {
public:
  ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
     //better approach: using difference in length of 2 LL:
     ListNode* t1=headA;
     ListNode* t2=headB;
     int len1=0;
     int len2=0;
     int d;
     while(t1!=nullptr)
     {
        len1++;
        t1=t1->next;
     }
     while(t2!=nullptr)
     {
        len2++;
        t2=t2->next;
```

```
    }
    t1=headA;
    t2=headB;
    if(len2>len1)
    {
        d=len2-len1;
        for(int i=0;i<d;i++)
        {
            t2=t2->next;
        }
    }
    else if(len1>len2)
    {
        d=len1-len2;
        for(int i=0;i<d;i++)
        {
            t1=t1->next;
        }
    }
    while(t1!=nullptr)
    {
        if(t1==t2)
        return t1;
        t1=t1->next;
        t2=t2->next;
```

```
        }
        return nullptr;
    }
};
```

4)OPTIMIZED APPROACH:

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        //optimized approach
        if(headA==nullptr||headB==nullptr)
        return nullptr;
        ListNode* t1=headA;
        ListNode* t2=headB;
        while(t1!=t2)
        {
            t1=t1->next;
            t2=t2->next;
            if(t1==t2)
            return t1;
            if(t1==nullptr)
            t1=headB;
            if(t2==nullptr)
            t2=headA;
        }
        return t1;
```

```
    }
};
```

12) ADD 1 TO A NUMBER:

BRUTE FORCE APPROACH:

```
class Solution {
  public:
  //BRUTE FORCE:reversal method:
       Node* rev(Node* head)
       {
           Node* temp=head;
           Node* prev=nullptr;
           Node* nxt=temp->next;
           while(temp!=nullptr)
           {
               temp->next=prev;
               prev=temp;
               temp=nxt;
               if(nxt!=nullptr)
               nxt=nxt->next;
           }
           return prev;
       }
       //addition logic:
    Node* addOne(Node* head) {
```

```
    head=rev(head);
    Node* temp=head;
    int carry=1;
    int sum;
    int flag=0;
    while(temp!=nullptr)
    {
        sum=temp->data+carry;
        if(sum<10)
        {
            temp->data=sum;
            flag=1;
            break;
        }
        else if(sum==10)
        {
            temp->data=0;
            carry=1;
        }
        temp=temp->next;
    }
    if(flag==0)   //additional node needed case
    {
        head=rev(head);
```

```
        Node* newHead=new Node(1);
        newHead->next=head;
        head=newHead;
    }
    else if(flag==1)
    {
        head=rev(head);
    }
     return head;
  }
};
```

OPTIMIZED APPROACH:

```
class Solution {
 public:
  int helper(Node* temp)
  {
    if(temp==nullptr)
    return 1;
    int carry=helper(temp->next);
    int sum=carry+temp->data;
    if(sum<10)
    {
        temp->data=sum;
        return 0;
```

```
        }
        else if(sum==10)
        {
            temp->data=0;
            return 1;
        }
    }
        //addition logic:
    Node* addOne(Node* head) {
        int carry=helper(head);
        if(carry==1)
        {
            Node* newNode=new Node(1);
            newNode->next=head;
            head=newNode;
        }
        return head;
    }
};
```

13) ADD 2 NUMBERS USING LL:

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* t1=l1;
        ListNode* t2=l2;
        int sum;
        int carry=0;
        ListNode* dummy=new ListNode(-1);
        ListNode* curr=dummy;
        //basic case:
        while(t1!=nullptr||t2!=nullptr)
        {
            sum=carry;
            if(t1)
            sum=sum+t1->val;
            if(t2)
            sum=sum+t2->val;
            if(sum<10)
            {
            ListNode* newNode=new ListNode(sum);
            curr->next=newNode;
            curr=curr->next;
            carry=0;
```

```
        }
        else if(sum>=10)
        {
        ListNode* newNode=new ListNode(sum%10);
        curr->next=newNode;
        curr=curr->next;
        carry=1;
        }
        if(t1)
        t1=t1->next;
        if(t2)
        t2=t2->next;
      }
     if(carry==1)
     {
        ListNode* newNode=new ListNode(1);
        curr->next=newNode;
     }
     return dummy->next;
   }
};
```

DLL:

1)DELETE ALL OCCURRENCES OF A NUMBER

CODE:

```
class Solution {
  public:
    void deleteAllOccurOfX(struct Node** head_ref, int x) {
        struct Node* temp=*head_ref;
        struct Node* prevN;
        struct Node* nextN;
        while(temp!=nullptr)
        {
        if(temp->data==x)
        {
            if(temp==*head_ref)
            {
                *head_ref=(*head_ref)->next;
                if(*head_ref)
                {
                    (*head_ref)->prev=nullptr;
                    free(temp);
                    temp=*head_ref;
                }
            }
            else
```

```
        {
            prevN=temp->prev;
            nextN=temp->next;
            if(prevN)
            prevN->next=nextN;
            if(nextN)
            nextN->prev=prevN;
            free(temp);
            temp=nextN;
        }
    }
    else
    temp=temp->next;
    }
  }
};
```

2) 2 SUM IN DLL:

```
class Solution
{
public:
  vector<pair<int, int>> findPairsWithGivenSum(Node *head, int target)
  {
    vector<pair<int, int>> v;
    Node* l=head;
```

```
        Node* t=head;
        while(t->next!=nullptr)
        {
            t=t->next;
        }
        Node* r=t;
        while(l->data<r->data)
        {
            if(l->data+r->data==target)
            {
            v.push_back({l->data,r->data});
            l=l->next;
            r=r->prev;
            }
            else if(l->data+r->data>target)
            r=r->prev;
            else if(l->data+r->data<target)
            l=l->next;
        }
        return v;
    }
};
```

STACKS AND QUEUES:

1) BALANCED BRACKETS:

```
2) class Solution {
3) public:
4)     bool isValid(string s) {
5)         stack<int> st;
6)         for(auto it:s)
7)         {
8)             if(it=='('||it=='{'||it=='[')
9)             {
10)                st.push(it);
11)            }
12)            else if(it==')'||it=='}'||it==']')
13)            {
14)                if(st.size()==0)
15)                return false;
16)                char ch=st.top();
17)                st.pop();
18)              //  if(ch!=it)
19)                if((ch=='{'&&it=='}')||(ch=='('&&it==')')||(ch=='['&&it==']
   '))
20)                continue;
21)                else
22)                return false;
23)
24)            }
25)        }
26)        return(st.empty());
27)
28)    }
29)};
```

FIX CONVERSIONS:

1) INFIX TO POSTFIX:

```
class Solution {
  public:
  int priority(char c)
    {
        if(c=='+'||c=='-')
        return 1;
        else if(c=='*'||c=='/')
        return 2;
        else if(c=='^')
        return 3;
        else
        return -1;
    }
    string infixToPostfix(string& s)
    {
        // code here
        string ans="";
        stack<char> st;
        char ch;
        for(auto it:s)
        {
            if((it=='+')||(it=='-')||(it=='*')||(it=='/')||(it=='^'))
            {
                while((!st.empty())&&(priority(it)<=priority(st.top())))
```

```
            {
                ans+=st.top();
                st.pop();
            }
            st.push(it);
        }
        else if(it=='(')
        {
            st.push(it);
        }
        else if(it==')')
        {
            while(!st.empty()&&st.top()!='(')
            {
            ans+=st.top();
            st.pop();
            }
            st.pop();
        }
        else
        {
            ans+=it;
        }
    }
    while (!st.empty())
```

```
    {
        ans += st.top();
        st.pop();
    }
    return ans;
  }
};
```

- **FOR INFIX TO PREFIX REF TO STRIVER VIDEO**

2) POSTFIX TO INFIX:

```
class Solution {
  public:
    string postToInfix(string exp) {
        // Write your code here
        stack<string> st;
        string op1;
        string op2;
        string ans="";
        string t="";
        for(auto it:exp)
        {
            if(it=='*'||it=='/'||it=='-'||it=='+'||it=='^')
            {
                op1=st.top();
                st.pop();
                op2=st.top();
                st.pop();
                ans="("+op2+it+op1+")";
                st.push(ans);
            }
            else
```

```
        {
            t+=it;
            st.push(t);
            t="";
        }
    }
    return ans;
  }
};
```

3) PREFIX TO INFIX:

```
class Solution {
  public:
   string preToInfix(string pre_exp) {
      // Write your code here
      int n=pre_exp.size();
      string ans="";
      n--;
      stack<string> st;
      string t="";
      string op1;
      string op2;
      while(n>=0)
      {
        if(pre_exp[n]=='*'||pre_exp[n]=='/'||pre_exp[n]=='+'||pre_exp[n]=='-'||pre_exp[n]=='^')
        {
            op1=st.top();
            st.pop();
```

```
            op2=st.top();
            st.pop();
            ans="("+op1+pre_exp[n]+op2+")";
            st.push(ans);
        }
        else
        {
            t+=pre_exp[n];
            st.push(t);
            t="";
        }
        n--;
    }
    return ans;
}
};
```

Monotonic stack/queue:

1) NEXT GREATER ELEMENT: (NGE1 LC #496):

BRUTE FORCE:

```
class Solution {

public:

  vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

    vector<int> ans;

    int n1=nums1.size();

    int n2=nums2.size();

    int temp;

    int temp2;

    int t=0;

    for(int i=0;i<n1;i++)

    {

      temp=nums1[i];

      t=0;

      for(int j=0;j<n2;j++)

      {

        if(nums1[i]==nums2[j])

        {

          temp2=j;

        }

      }

      for(int k=temp2+1;k<n2;k++)

      {

        if(nums2[k]>nums2[temp2])
```

```
            {
                t=1;
                ans.push_back(nums2[k]);
                break;
            }
        }
        if(t==0)
        {
            ans.push_back(-1);
        }
    }
    return ans;
  }
};
```

TO DO: OPTIMIZED APPROACH

2) NGE 1: TUF VERSION:

BRUTE FORCE: Ref TUF YT Video

OPTIMIZED APPROACH:

```
class Solution {
public:
  vector<int> nextLargerElement(vector<int> arr) {
      int n=arr.size();
      vector<int> ans(n);
      stack<int> st;
      int ele;
      int l;
      for(int i=n-1;i>=0;i--)
      {
          while(!st.empty()&&st.top()<=arr[i])
```

```
            {
                st.pop();
            }
            if(st.empty())
            {
                ans[i]=-1;
            }
            else
            {
                ans[i]=st.top();
            }
            st.push(arr[i]);
        }
        return ans;
    }
};
```

3)NGE 2:

BRUTE FORCE:

```
class Solution {
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        //BRUTE FORCE
        int n=nums.size();
        vector<int> ans(n);
        int ele;
        int flag=0;
        for(int i=0;i<n;i++)
        {
            flag=0;
            ele=nums[i];
            for(int j=i+1;j<n;j++)
            {
                if(nums[j]>ele)
```

```
            {
                ans[i]=nums[j];
                flag=1;
                break;
            }
        }
        if(flag==0)
        {
            for(int k=0;k<i;k++)
            {
                if(nums[k]>nums[i])
                {
                    ans[i]=nums[k];
                    flag=1;
                    break;
                }
            }
        }
        if(flag==0)
        ans[i]=-1;
    }
    return ans;
  }
};
```

ALTERNATIVE APPROACH: NOT recommended

```
class Solution {
public:
  vector<int> nextGreaterElements(vector<int>& nums) {
    int n=nums.size();
    vector<int> ans(n);
    stack<int> st;
    int ele;
    int l;
```

```
        int flag=0;
        for(int i=n-1;i>=0;i--)
        {
                flag=0;
                while(!st.empty()&&st.top()<=nums[i])
                {
                    st.pop();
                }
                if(st.empty())   //check
                {
                    for(int k=0;k<i;k++)
                    {
                        if(nums[k]>nums[i])
                        {
                            ans[i]=nums[k];
                            flag=1;
                            break;
                        }
                    }
                    if(flag==0)
                    {
                        ans[i]=-1;
                    }
                    st.push(nums[i]);
                }
                else
                {
                    ans[i]=st.top();
                    st.push(nums[i]);
                }
        }
        return ans;
    }
};
```

OPTIMIZED APPROACH:

```
class Solution {
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        int n=nums.size();
        vector<int> ans(n);
        stack<int> st;
        int ind;
        for(int i=(2*(n-1));i>=0;i--)
        {
            ind=i%n;
            if(i<n)
            {
                while(!st.empty()&&nums[i]>=st.top())
                {
                    st.pop();
                }
                if(!st.empty())
                {
                ans[i]=st.top();
                }
                else
                {
                    ans[i]=-1;
                }
                st.push(nums[i]);
            }
            else
            {
            if(st.empty())
            {
                st.push(nums[ind]);
            }
```

```
            else
            {
                if (nums[ind]>st.top())
                {
                    while(!st.empty()&&nums[ind]>=st.top())
                    {
                        st.pop();
                    }
                }
                st.push(nums[ind]);
            }
            }
        }
        return ans;
    }
};
```

3)SMALLER ELEMENT ON LEFT (GFG): previous smaller ele

https://www.geeksforgeeks.org/problems/smallest-number-on-left3403/0

OPTIMIZED APPROACH:

```
class Solution {
  public:
    vector<int> leftSmaller(vector<int> arr) {
        int n=arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i=0;i<n;i++)
        {
            while(!st.empty()&&st.top()>=arr[i])
            {
                st.pop();
            }
            if(st.empty())
            {
```

```
            ans[i]=-1;
        }
        else
        {
            ans[i]=st.top();
        }
        st.push(arr[i]);
    }
    return ans;
  }
};
```

4) MIN SUBARRAY SUM: correct approach but gives TLE for large inputs

```
class Solution {
public:
  int sumSubarrayMins(vector<int>& arr) {
    //BRUTE FORCE:
    long long sum=0;
    int ans;
    int n=arr.size();
    int min=INT_MAX;
    long long val;
    for(int i=0;i<n;i++)
    {
        min=INT_MAX;
        for(int j=i;j<n;j++)
        {
            if(arr[j]<min)
            min=arr[j];
            sum+=min;
        }
    }
    val=1000000007;
    ans=sum%val;
```

```
        return ans;
    }
};
```