1.Spiral of matrices:

```
class Solution
{
   public:
   //Function to return a list of integers denoting spiral
traversal of matrix.
   vector<int> spirallyTraverse(vector<vector<int> > matrix,
int r, int c)
   {
      vector<int> v;
   int idx=0;
   int left=0,right=c-1,top=0,bottom=r-1;
   while(left<=right && top<=bottom)
   {
      for(int i=left;i<=right;i++)v.push_back(matrix[top][i]);
      top++;
      for(int i=top;i<=bottom;i++)v.push_back(matrix[i][right]);
      right--;
      if(top<=bottom)
      {
      for(int i=right;i>=left;i--)v.push_back(matrix[bottom][i]);
      bottom--;
      }
      if(left<=right)
      {
      for(int i=bottom;i>=top;i--)v.push_back(matrix[i][left]);
      left++;
      }
   }
```

```
    //vector<int> v(arr,arr+n);
    return v;
    }
};
```

2.Max number of rows in row sorted array:

Method 1:Brute force

Method 2:Using binary search

Method 3:

```
class Solution{
public:
        int rowWithMax1s(vector<vector<int> > arr, int n, int m) {
            int count=0,ans=-1;
            int idx=m-1;
            for(int i=0;i<n;i++)
            {
                while(arr[i][idx]==1)
                {
                    idx--;
                    count+=1;
                    ans=i;
                }
            }
            return ans;
        }
};
```

3.Rotate a matrix by 90 degree in clockwise direction without using any extra space:

```
void rotate90clockwise(int mat[n][n])
{
    // Transpose of matrix
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            swap(mat[i][j], mat[j][i]);
    // Reverse individual rows
    for (int i = 0; i < n; i++) {
        int low = 0, high = n - 1;
        while (low < high) {
            swap(mat[i][low], mat[i][high]);
            low++;
            high--;
        }
    }
}
```

4.Common elements in all rows of a given matrix:

```
void printCommonElements(int mat[M][N])
{
    unordered_map<int, int> mp;
    // initialize 1st row elements with value 1
    for (int j = 0; j < N; j++)
        mp[mat[0][j]] = 1;

    // traverse the matrix
    for (int i = 1; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            // If element is present in the map and
            // is not duplicated in current row.
            if (mp[mat[i][j]] == i)
            {
               // we increment count of the element
               // in map by 1
                mp[mat[i][j]] = i + 1;

                // If this is last row
                if (i==M-1 && mp[mat[i][j]]==M)
                  cout << mat[i][j] << " ";
            }
        }
    }
}
```

STACKS

1.Next greater element on right side:

Method 1:Brute force

Check one by one

Method 2:Stacks

```
class Solution
{
    public:
    //Function to find the next greater element for each
element of the array.
    vector<long long> nextLargerElement(vector<long long>
arr, int n){
        stack<long> s;
        vector<long long> v;

        v.push_back(-1);
        s.push(arr[n-1]);
        for(int i=n-2;i>=0;i--)
        {
            while(!s.empty() && s.top()<=arr[i])s.pop();

            if(s.empty())v.push_back(-1);
```

```
      else if(s.top()>arr[i])v.push_back(s.top());

      s.push(arr[i]);
    }
    reverse(v.begin(),v.end());
    return v;
  }
};
```

2.Next smaller number on right side

```
class Solution{

  public:
  vector<int> help_classmate(vector<int> arr, int n)
  {
   stack<int> s;
   vector<int> v;

   s.push(arr[n-1]);
   v.push_back(-1);

   for(int i=n-2;i>=0;i--)
   {
     while(!s.empty() && s.top() >= arr[i])s.pop();

     if(s.empty())v.push_back(-1);
     else if(s.top() < arr[i])v.push_back(s.top());

     s.push(arr[i]);
```

```
      }
      reverse(v.begin(),v.end());

      return v;
    }
};
```

3. Online Stock Span

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

int main()
{
    int arr[]={100,80,60,70,60,75,85};
    int n=sizeof(arr)/sizeof(arr[0]);
    stack<int> s;
    vector<int> v;

    v.push_back(1);
    s.push(0);

    for(int i=1;i<n;i++)
    {
        while(!s.empty() && arr[s.top()] <= arr[i])s.pop();
        if(s.empty())v.push_back(1);
        else if(arr[s.top()] > arr[i])v.push_back(i-s.top());

        s.push(i);
    }

    for(auto i:v)cout<<i<<" ";
    return 0;
}
```

4.Max area of Histogram

```
class Solution
{
    public:
    //Function to find largest rectangular area possible in a given histogram.
    long long getMaxArea(long long arr[], int n)
    {
        stack<long int> s;
        vector<int> v1;
        vector<int> v2;

        s.push(0);
        v1.push_back(-1);

        for(int i=1;i<n;i++)
        {
            while(!s.empty() && arr[s.top()]>= arr[i])s.pop();
            if(s.empty())v1.push_back(-1);
            else if(arr[s.top()] < arr[i])v1.push_back(s.top());

            s.push(i);
```

```
         }

         while(!s.empty())s.pop();

         s.push(n-1);
         v2.push_back(n);


        for(int i=n-2;i>=0;i--)
        {
            while(!s.empty() && arr[s.top()] >= arr[i])s.pop();
             if(s.empty())v2.push_back(n);
             else if(arr[s.top()] < arr[i])v2.push_back(s.top());

             s.push(i);
        }
        reverse(v2.begin(),v2.end());
        //for(int i=0;i<n;i++)cout<<v2[i]<<" ";
        long int area=0;
        long int max_area=INT_MIN;
        for(int i=0;i<n;i++)
        {
            area=(v2[i]-v1[i]-1)*arr[i];
            max_area=max(max_area,area);
        }
        return max_area;
    }
};
```

5.Max area of rectangle:

```
class Solution{
  public:
     int getMaxArea(int arr[], int n)
    {
        stack<int> s;
        vector<int> v1;
        vector<int> v2;

        s.push(0);
        v1.push_back(-1);

        for(int i=1;i<n;i++)
        {
            while(!s.empty() && arr[s.top()]>= arr[i])s.pop();
            if(s.empty())v1.push_back(-1);
            else if(arr[s.top()] < arr[i])v1.push_back(s.top());

            s.push(i);
        }

        while(!s.empty())s.pop();

        s.push(n-1);
        v2.push_back(n);


       for(int i=n-2;i>=0;i--)
       {
```

```
       while(!s.empty() && arr[s.top()] >= arr[i])s.pop();
        if(s.empty())v2.push_back(n);
        else if(arr[s.top()] < arr[i])v2.push_back(s.top());

        s.push(i);
   }
   reverse(v2.begin(),v2.end());
   //for(int i=0;i<n;i++)cout<<v2[i]<<" ";
   int area=0;
   int max_area=INT_MIN;
   for(int i=0;i<n;i++)
   {
       area=(v2[i]-v1[i]-1)*arr[i];
       max_area=max(max_area,area);
   }
   return max_area;
}

int maxArea(int M[MAX][MAX], int n, int m) {
int arr[m]={0};
int max_area=INT_MIN;
for(int i=0;i<n;i++)
  {
    for(int j=0;j<m;j++)
        {
            if(M[i][j]==0)arr[j]=0;
            else arr[j]+=1;
        }
    max_area=max(max_area,getMaxArea(arr,m));
  }
```

```
      return max_area;
    }
};
```

6.Celebrity Problem:

Method 1:Using in and out array

In[j]++→if i knows j
Out[i]++→if i knows j

Celebrity:in[c]==n-1 && out[c]==0 return c;

Complexity:Time=O(N^2) space=O(N)

Method 2:

```
class Solution
{
    public:
    //Function to find if there is a celebrity in the party or not.
    int celebrity(vector<vector<int> >& M, int n)
    {

        int c=0;
        for(int i=1;i<n;i++)
        {
            if(M[c][i]==1)c=i;
        }
        for(int i=0;i<n;i++)
        {
            if(c!=i && (M[c][i]==1 or M[i][c]==0))return -1;
        }
        return c;
    }
};
```

Complexity:O(n)

7.Circular Queue:

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

class Queue{
    public:
    int *arr;
    int r;
    int f;
    int n;
    Queue()
    {
        arr=new int[3];
        r=f=-1;
        n=3;
    }
    void enqueue(int val)
    {
        if((r+1)%n==f)cout<<"overflow"<<endl;
        if(f==-1)f=0;
        r=(r+1)%n;
        arr[r]=val;
    }
    int dequeu()
    {
        if(f==-1)return -1;
        int val=arr[f];
        if(r==f)r=f=-1;
        f=(f+1)%n;
        return val;
    }
};
int main(){
    Queue ob;
    ob.enqueue(1);
    ob.enqueue(2);
    ob.enqueue(3);
    ob.enqueue(4);
    cout<<ob.dequeu();
    return 0;
```

```
}
```

BIT MANIPULATION

1.Lcm and Gcd

```
class Solution {
  public:
  int gcd_(long int a,long int b)
  {
      if(b==0)return a;
      return gcd_(b,a%b);
  }
    vector<long long> lcmAndGcd(long long A , long long B) {
        vector<long long> v;
        long int gcd=gcd_(A,B);
        v.push_back((A*B)/gcd);
        v.push_back(gcd);
       return v;
    }
};
```

Time Complexity: O(log(min(a,b))

Auxiliary Space: O(log(min(a,b))

2.Count the number of bits:

```
class Solution {
  public:
    int setBits(int N) {
        int c=0;
        while(N)
        {
            if(N&1)c++;
            N=N>>1;
        }
        return c;
    }
};
```

13.Find the two non-repeating elements in an array of repeating elements:

```
class Solution
{
public:
    vector<int> singleNumber(vector<int> nums)
    {
        int xor1=nums[0];
        for(int i=1;i<nums.size();i++)xor1^=nums[i];

        int right_set_bit=xor1 & ~(xor1-1);
        int x=0,y=0;
        for(int i=0;i<nums.size();i++)
        {
            if(nums[i]&right_set_bit)x^=nums[i];
            else y^=nums[i];
        }
        vector<int> v;
        v.push_back(x);
        v.push_back(y);
        sort(v.begin(),v.end());
        return v;
    }
};
```

14.count the number of bits needed to be flipped to convert A to B.

```
class Solution{
    public:
    // Function to find number of bits needed to be flipped to
convert A to B
    int countBitsFlip(int a, int b){

        int xor1=a^b;
        int c=0;
        while(xor1)
        {
            if(xor1&1)c++;
            xor1=xor1>>1;
        }
        return c;

    }
};
```

15.Power of 2:

```
class Solution{
    public:
    // Function to check if given number n is a power of two.
    bool isPowerofTwo(long long n){
        if(n==0)return 0;

      return(!(n&(n-1)));

    }
};
```

15.Find position of set bit:

```
class Solution {
  public:
    int findPosition(int N) {
        if(N==0)return -1;
        int i=1;
        while(N)
        {
            if(N&1) break;
            i++;
            N=N>>1;
        }
       N=N>>1;
       if(N>0)return -1;
```

```
      else return i;
   }
};
```

16.Divide two integers without using multiplication, division and mod operator:

```
class Solution
{
   public:
   long long divide(long long dividend, long long divisor)
   {
      int sign=((dividend < 0)^(divisor < 0))?-1:1;

      dividend=abs(dividend);
      divisor=abs(divisor);
      //int temp=divisor;
      int ans=0;
      while(dividend-divisor>=0)
      {
         int count=0;
         while(dividend-(divisor<<1<<count) >=0)count++;


         ans+=1<<count;
         dividend=dividend-(divisor<<count);
      }
      return ans*sign;
   }
};
```