EASY :

1. Climbing stairs

```
class Solution {
public:
    int climbStairs(int n)
    {
        int a=1;
        int b=2;
        if(n==1 || n==2)
        return n;
        else
        {
            int sum=0;
            for(int i=3;i<=n;i++)
            {
                sum=(a+b);
                a=b;
                b=sum;
            }
            return sum;
        }
    }
};
```

2. Best time to buy and sell stocks

```
class Solution {
public:
    int maxProfit(vector<int>& prices)
    {
        int minnn=INT_MAX;
        int diff=INT_MIN;
    for(int i=0;i<prices.size();i++)
    {
        if(minnn>prices[i])
        minnn=prices[i];
        if(prices[i]-minnn>diff)
            diff=prices[i]-minnn;
    }
        return diff;
    }
};
```

3. Best time to buy and sell stocks II

```
class Solution {
public:
   int maxProfit(vector<int>& prices)
   {
      int maxx=0;
      for(int i=1;i<prices.size();i++)
      maxx+=max(0,prices[i]-prices[i-1]);
      return maxx;
   }
};
```

MEDIUM :

1. Maximum path sum in matrix

```
class Solution {
public:
   int solve(vector <vector <int>> &grid )
   {
      int r = grid.size();
      int c = grid[0].size();

      vector <vector <int>> dp(r, vector <int>(c, 0));
      dp[0][0] = grid[0][0];

      for (int i = 1; i < c; i++)
         dp[0][i] = grid[0][i] + dp[0][i - 1];

      for (int i = 1; i < r; i++)
         dp[i][0] = grid[i][0] + dp[i - 1][0];

      for (int i = 1; i < r; i++)
         for (int j = 1; j < c; j++)
            dp[i][j] = grid[i][j] + min(dp[i - 1][j], dp[i][j - 1]);

      return dp[r - 1][c - 1];
   }
   int minPathSum(vector<vector<int>>& grid)
   {
       int ans1=solve(grid);
      return ans1;
   }
};
```

2. Decode ways

```
class Solution {
public:
    int numDecodings(string s) {

        vector<int>dp={1, s[0]>'0'};
        for(int i=1;i<s.size();i++)
        {

            if(s[i-1]=='2' && s[i]<'7' || s[i-1]=='1')
            dp[(i+1)%2]=dp[(i+1)%2];
            else
            dp[(i+1)%2]=0;
            if(s[i]>'0')
            dp[(i+1)%2]+=dp[i%2];

        }
            return dp[s.size()%2];
    }
};
```

3. Maximum product subarray

```
class Solution {
public:
    int maxProduct(vector<int>& nums)
    {
        long long int maxx=nums[0];
        long long int minn=nums[0];
        long long int ans=nums[0];
        int n=nums.size();

        for (int i=1;i<n;i++)
        {
            int max_val=maxx*nums[i];
            int min_val=minn*nums[i];

            maxx=max({nums[i],min_val,max_val});
            minn=min({nums[i],min_val,max_val});
            cout<<max_val<<" "<<min_val<<" "<<maxx<<" "<<minn<<"\n";
            ans=max(maxx,ans);
        }
        return ans;
    }
};
```

4. Longest common substring

```
class Solution {
public:
   int trap(vector<int>& height)
   {
      int n = height.size();
      int l=0;
      int r=n-1;
      int ans=0;
      int maxl=0;
      int maxr=0;

      while(l<=r)
      {

         if(height[l]<=height[r])
         {

            if(height[l]>=maxl)
            maxl=height[l];
            else
            ans+=maxl-height[l];

            l++;
         }
         else
         {

            if(height[r]>=maxr)
            maxr= height[r];
            else
            ans+=maxr-height[r];

            r--;
         }
      }
      return ans;
   }
};
```

5. Longest palindromic substring

```
class Solution {
public:
   string maxx_palindrome(string s)
   {
```

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

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

6. Longest palindromic subsequence

```
class Solution {
public:
   int lcs(string text1, string text2)
   {
      int n=text1.size();
      int m=text2.size();
      vector<vector<int>>dp(n+1,vector<int>(m+1));
      for(int i=1;i<=n;i++)
      for(int j=1;j<=m;j++)
      dp[i][j]=(text1[i-1]==text2[j-1])?dp[i-1][j-1]+1:max(dp[i-1][j],dp[i][j-1]);

      return dp[n][m];
   }
   int longestPalindromeSubseq(string s)
   {
      string str=s;
      reverse(s.begin(),s.end());
      return lcs(str,s);
   }
};
```

HARD :

1. Best time to buy and sell stocks III

```
class Solution {
public:
   int maxProfit(vector<int>& prices)
   {
      int k=2;
      int n=prices.size();
      if(n==0)
      return 0;
      else if(k==0)
      return 0;

      vector<vector<int>> dp(k+1,vector<int>(n,0));
      for(int i=1;i<=k;i++)
      {
         int val=dp[i-1][0]-prices[0];
         for(int j=1;j<n;j++)
         {
            int mx=max(dp[i][j-1],val+prices[j]);
            dp[i][j]=mx;
            val=max(val,dp[i-1][j]-prices[j]);
```

```
            }
        }
        return dp[k][n-1];
    }
};
```

2. Best time to buy and sell stocks IV

```
class Solution {
public:

    int maxProfit(int k, vector<int>& prices)
    {
        int n=prices.size();
        if(n==0)
        return 0;
        else if(k==0)
        return 0;

        vector<vector<int>> dp(k+1,vector<int>(n,0));
        for(int i=1;i<=k;i++)
        {
            int val=dp[i-1][0]-prices[0];
            for(int j=1;j<n;j++)
            {
                int mx=max(dp[i][j-1],val+prices[j]);
                dp[i][j]=mx;
                val=max(val,dp[i-1][j]-prices[j]);
            }
        }
        return dp[k][n-1];
    }
};
```

3. Trapping rain water

```
class Solution {
public:
    int trap(vector<int>& height)
    {
        int n = height.size();
        int l=0;
        int r=n-1;
        int ans=0;
        int maxl=0;
        int maxr=0;
```

```
        while(l<=r)
        {

            if(height[l]<=height[r])
            {

                if(height[l]>=maxl)
                maxl=height[l];
                else
                ans+=maxl-height[l];

                l++;
            }
            else
            {

                if(height[r]>=maxr)
                maxr= height[r];
                else
                ans+=maxr-height[r];

                r--;
            }
        }
        return ans;
    }
};
```