The Most Important Topic In DSA : Dynamic Programming.

Dynamic Programming is mainly an optimization over plain recursion.
The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

SNo. Problem Statement

1. Easy Level: Climbing Stairs.

Code:

```
Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps
```

```
int climbStairs(int n) {
        int t[n+1];
        t[0] =1;
        t[1] = 1;
        for(int i=2; i<n+1; i++) t[i] = t[i-1] + t[i-2];
        return t[n];

    }
```

2. Easy Level: Maximum Product Subarray.

Code:

```
Input: nums = [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.
```

```
int maxProduct(vector<int>& nums) {

    /*    int n=nums.size();
        int dp[n][2];

        dp[0][0]=nums[0];
        dp[0][1]=nums[0];
        int ans=dp[0][0];
        for(int i=0;i<=n;i++)
        {
            for(int j=0;j<=n;j++)
            {
                if(i==0 || j==0)
                {
                    dp[i][j]=0;
                }
            }
        }
        for(int i=1;i<n;i++){
            dp[i][0] = max(nums[i],max(dp[i-1][0]*nums[i],dp[i-
1][1]*nums[i]));
            dp[i][1] = min(nums[i],min(dp[i-1][0]*nums[i],dp[i-
1][1]*nums[i]));
            ans = max(ans,dp[i][0]);
        }
        return ans;*/
```

Solution-2 :)

```
     int r=nums[0];
    int maxi=r;
    int mini=r;
        for(int i=1;i<nums.size();i++)
        {
            if(nums[i]<0)
            {
                int temp=maxi;
                maxi=mini;
                mini=temp;
            }
            maxi=max(nums[i],nums[i]*maxi);
            mini=min(nums[i],nums[i]*mini);
            r=max(r,maxi);
        }
        return r;


        //brute force due to time limit did not accepted
```

```
   // int maxProduct(vector<int>& A) {
    /*int ans = INT_MIN;
    for(int i = 0; i < nums.size(); i++) {
        int curProd = 1;
        for(int j = i; j < nums.size(); j++)
            curProd *= nums[j],
            ans = max(ans, curProd);
    }
    return ans;*/
}
```

3.

Easy Level: Ones and Zeroes.
Code:

```
Input: strs = ["10","0001","111001","1","0"], m = 5, n = 3

Output: 4

Explanation: The largest subset with at most 5 0's and 3 1's is {"10",
"0001", "1", "0"}, so the answer is 4.

Other valid but smaller subsets include {"0001", "1"} and {"10", "1",
"0"}.

{"111001"} is an invalid subset because it contains 4 1's, greater than
the maximum of 3.
```

```
vector<vector<vector<int>>>dp;
    int maxOneandZero(vector<string>& strs,int i,int m,int n)
    {
        if(m<0 || n<0)
            return -1e9;
        if(m==0 and n==0)
            return 0;
        if(i==strs.size())
            return 0;
        if(dp[i][m][n]!=-1)
            return dp[i][m][n];

        int c1=0;
        int c2=0;

        for(auto it:strs[i])
        {
            if(it=='1')
                c1++;
            else
                c2++;
        }
        return dp[i][m][n]=max(1+maxOneandZero(strs,i+1,m-c2,n-c1),maxOneandZero(strs,i+1,m,n));
    }
    int findMaxForm(vector<string>& strs, int m, int n) {
        int s = strs.size();
        dp.resize(s, vector<vector<int>>(m+1, vector<int>(n+1,-1)));

        return max(maxOneandZero(strs,0,m,n),0);
    }
```

4. **Easy Level: Counting Bits.**

Code:

```
Input: n = 2

Output: [0,1,1]

Explanation:

0 --> 0

1 --> 1

2 --> 10
```

```
vector<int> countBits(int n) {
    /*   vector<int>arr(n+1);
        arr[0]=0;
        for(int i=0;i<=n;i++)
        {
            if(i & 1)//condition for n=odd
                arr[i]=1;
            else
                arr[i]=0;
            arr[i]+=arr[i/2];
        }
        return arr;*/
        vector<int>res;
        for(int i=0;i<=n;i++)
        {
            int c=0;
            int num=i;
            while(num)
            {
                c++;
                num=num & (num-1);
            }
            res.push_back(c);
        }
        return res;
    }
```