Fibonacci:

```
#include<stdio.h>


int main(){

    //runtime complexity:O(n)



    int fibo[46],n,i;

    fibo[0] = 0;

    fibo[1] = 1;

    for(int i = 2; i <= 45; i++)

        fibo[i] = fibo[i - 1] + fibo[i - 2];

    scanf("%d",&n);

    printf("%d",fibo[n]);



    return 0;



}
```

ANALYSIS:

1st number -> 0
2nd number -> 1
N th fibo = sum previous 2 number

3rd = 1st + 2nd
= 0 + 1

= 1

4^{th} = 3^{rd} + 2^{nd}

= 1 + 1

= 2

5^{th} = 4^{th} + 3^{rd}

=2+1

=3

```
int fibo[10]; //array size

fibo [ 0 ] = 0;     // [ array index ]

fibo [ 1 ] = 1;  //array index

fibo [ 2 ] = fibo[2 - 1] + fibo[2 - 2];
            =  1 + 0
            =  1
fibo[ 3 ] = fibo[3 - 1] + fibo[3 - 2]
            = fibo[2] + fibo[1]
            = 1 + 1
            = 2
fibo[ 4 ] = fibo[4 - 1] + fibo[4 - 2]
           = fibo[ 3 ] + fibo[ 2 ]
           = 2 + 1
           = 3



int fibo[11];
Fibo[ 0 ] = 0;

Fibo[ 1 ] = 1;

for(int i = 2; i < 11; i++)  // n = 10

        Fibo[ i ] = fibo[i - 1] + fibo[i - 2];


Fibo[ 3 ]     = fibo[3 - 1] + fibo[3 - 2]
                = fibo [2] + fibo[ 1 ]
                = 1 + 1
                = 2
```

Nth fibo number time complexity O(N)

```
#include<stdio.h>

int main(){
    //runtime complexity:O(n)
    int n;
    scanf("%d",&n);
    int fibo[n + 1];
    fibo[0] = 0;
    fibo[1] = 1;
    for(int i = 2; i <= n; i++)
        fibo[i] = (fibo[i - 1] + fibo[i - 2]) % 10;
    printf("%d\n", fibo[n]);
    return 0;
}
```

ANALYSIS:

Last digit:

```
int fibo[11];
Fibo[ 0 ] = 0;
Fibo[ 1 ] = 1;
for(int i = 2; i < 11; i++)                    // n time loop
        Fibo[ i ] = ( fibo[i - 1] + fibo[i - 2] ) % 10;
```

15 **%** 10 = 5 // % modulus

Time complexity: O(n)

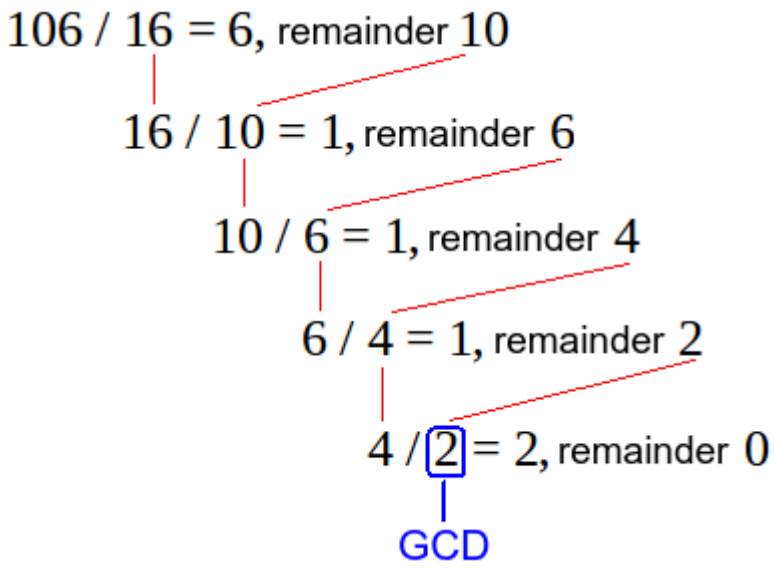
```
i = 2
Fibo[ 2 ]     = ( fibo[2 - 1] + fibo[2 - 2]) % 10;
              = ( fibo[1] + fibo[0] ) % 10
              = ( 1 + 0) % 10
              = 1 % 10
              = 1
Fibo[ 2 ] = 1


Fibo[ 7 ] = 13         //13 % 10 == 3
Fibo[ 8 ] =  21        // 21 % 10 == 1
Fibo[ 9 ] = Fibo[ 9 - 1 ] + Fibo[ 9 - 2 ]
          = Fibo[ 8 ] + Fibo[ 7 ]
```

= 21 + 13
= 34 // 34%10 ==4

Fibo[7] = 3
Fibo[8] = 1
Fibo[9] = 4

GCD



Log (min(a, b))

106 16
= 16

log(16)

Let,
gcd of a b = z log (min(a, b))

```c
#include<stdio.h>
int main(){
    //run time complexity:O(logn)
    int  a, b ,gcd, rem;
    scanf("%d %d",&a,&b);
    while(b!=0)
    {
       rem=a%b;
       a=b;
       b=rem;
    }
    gcd =a;
    printf("%d\n",gcd);
    return 0;
}
```

GCD algorithm…

Euclidean gcd algorithm based on subtraction. It's normally used for computing gcd of two numbers.

Algorithm:

```
gcd of a and b
        while b ≠ 0
                a:= b
                rem:= a mod b
                b= rem
        return a
```

Explain:

In this algorithm, it divide the big number by a small number. If the remainder is not equal to zero it goes to the next step. Divide the divisor by the remainder. Repeated that process until the remainder is equal to zero. When the remainder is zero that divisors are gcd.

Example: Find out gcd of number 24 and 18?

1: Divide the big number (24) by the small number (18).
24 18 =1 remainder 6
2: Divide divisor (18) by the remainder(6). 18 6 = 3 remainder 0.

The remainder is 0. So, gcd of 24 and 18 is 6.

Illustration:

Recursive Application:

```
int gcd(int a, int b)
 {
        if(a == 0) return b;
        return gcd(b % a, a);
}
```

Time complexity. log(min(a, b))

LCM

```c
#include<stdio.h>
                    int main(){
                        //run time complexity :O(logn)
                        int  a, b ,gcd, rem,lcm,n1,n2;
                        scanf("%d %d",&n1,&n2);
                        a=n1;
                        b=n2;
                        while(b!=0)
                        {
                           rem=a%b;
                           a=b;
                           b=rem;
                        }
                        gcd=a;


                        lcm= ((n1*n2)/gcd);
                        printf("%d\n",lcm);
                        return 0;
                    }
```

LCM:

We know that,

Lcm of two numbers = multiplication of two numbers gcd of two numbers.

lcm(a, b) = a * b / gcd(a, b).

Time complexity: O(log(min(a, b))) + O(1)

= O (log(min(a, b))).