CSE 3004 DAA LAB TASK - 3

NAME : V.SAIKRISHNA
REG. NO :19BCE7638

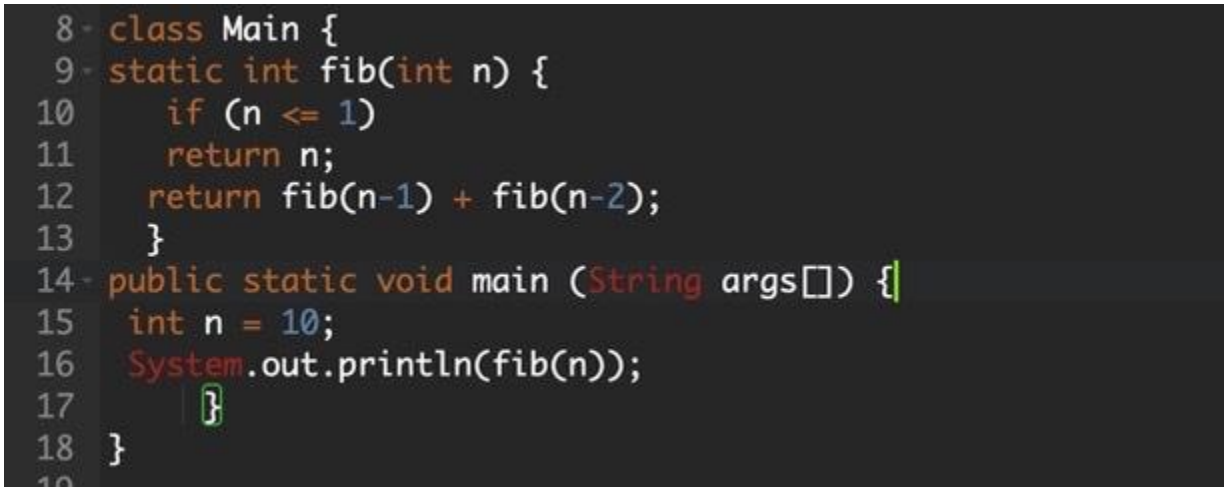
Fibonacci

Using Naive algorithm:

```
class Main {
static int fib(int n) {
   if (n <= 1)
   return n;
  return fib(n-1) + fib(n-2);
```

```
  }
public static void main (String args[]) {
 int n = 10;
 System.out.println(fib(n));
     }
}
```

OUTPUT:



```
class Main {
static int fib(int n) {
   if (n <= 1)
   return n;
  return fib(n-1) + fib(n-2);
  }
public static void main (String args[]) {
 int n = 10;
 System.out.println(fib(n));
     }
}
```

Using DP:

```
class Main
{
  static int fib(int n)
  {
   int f[] = new int[n+2];
   f[0] = 0;
   f[1] = 1;
```

```
   for (int i = 2; i <= n; i++) {
   f[i] = f[i-1] + f[i-2];
    }
   return f[n];
   }
public static void main (String args[]) {
 int n = 20;
 System.out.println(fib(n));
 }
}
```

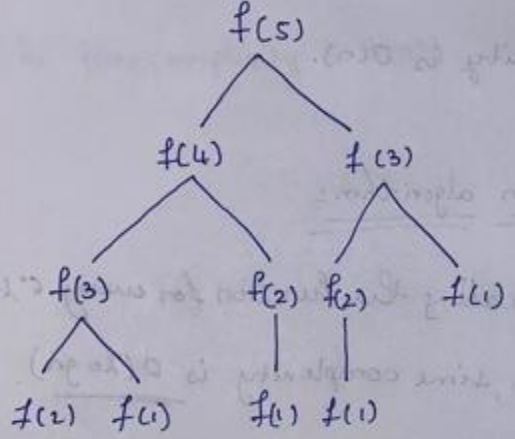
OUTPUT:

```
class Main
{
  static int fib(int n)
  {
   int f[] = new int[n+2];
   f[0] = 0;
   f[1] = 1;
   for (int i = 2; i <= n; i++) {
   f[i] = f[i-1] + f[i-2];
    }
   return f[n];
   }
public static void main (String args[]) {
 int n = 20;
 System.out.println(fib(n));
 }
}
```

ANALYSIS:

Fibonacci :-

(1.) Using Naive approach :-



Here it almost calling the method function at 2^n times.

$\therefore O(2^n)$

(2.) Using DP :-

Here, we are saving the value of f(5), f(4) etc. in an array. so at worst case we call it at O(n) times.

GCD

Using Navie Algorithm:

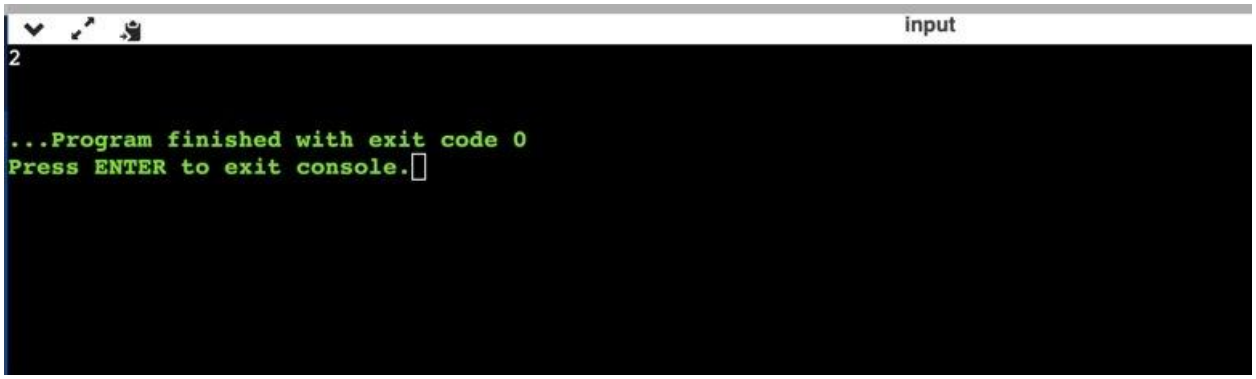
```
import java.lang.Math;
import java.util.Scanner;
public class Main {
static int GCD(int a,int b) {
int maximum=Math.max(a,b);
int currentNumber=maximum-1;
while(currentNumber>1){

if((a%currentNumber==0)&&(b%currentNumber==0)
){ return currentNumber;
 }
```

```
 else {
     currentNumber--;
     }
    }
  return 1;
  }
public static void main(String[] args){
System.out.println(GCD(20,42));
    }
}
```

OUTPUT:

```java
import java.lang.Math;
import java.util.Scanner;
public class Main {
static int GCD(int a,int b) {
int maximum=Math.max(a,b);
int currentNumber=maximum-1;
while(currentNumber>1){
    if((a%currentNumber==0)&&(b%currentNumber==0)){ return currentNumber;
 }
 else {
     currentNumber--;
     }
    }
  return 1;
  }
public static void main(String[] args){
System.out.println(GCD(20,42));
    }
}
```



Using Euclidean Algorithm:

```
import java.util.*;
import java.lang.*;
class Main {
 public static int gcd(int a, int b) {
 if (a == 0)
 return b;
 return gcd(b%a, a);
 }
 public static void main(String[] args) {
     System.out.println(gcd(3918848,1653264));
    }
}
```

OUTPUT:

```java
import java.util.*;
import java.lang.*;
class Main {
 public static int gcd(int a, int b) {
 if (a == 0)
 return b;
 return gcd(b%a, a);
 }
 public static void main(String[] args) {
     System.out.println(gcd(3918848,1653264));
    }
}
```

ANALYSIS:

GCD :-

(1.) Using naive algorithm :-

Here, we are calling the function for n times then, the complexity is O(n).

(2.) Using Euclieden algorithm :-

there, we are calling the function for every $\log$ reminder. so, time complexity is $O(\log n)$.