FIBONACCI AND GCD

NAME: K SASANK PRABHATH
REG NO:19BCE7564

Fibonacci

Using Naïve 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));
}}
```

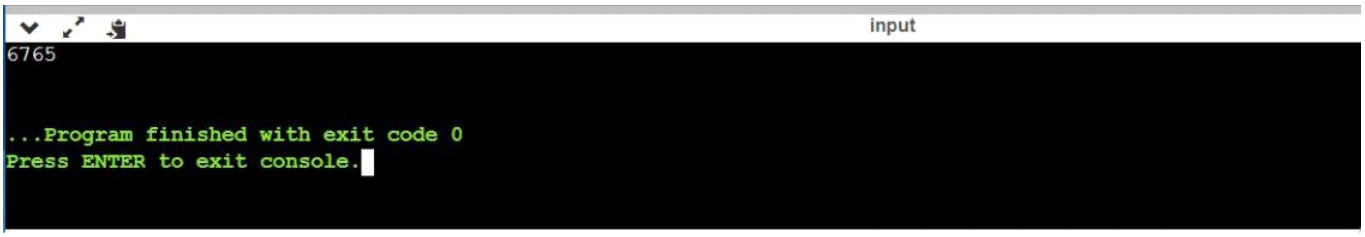
OUTPUT:

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));
}}
```

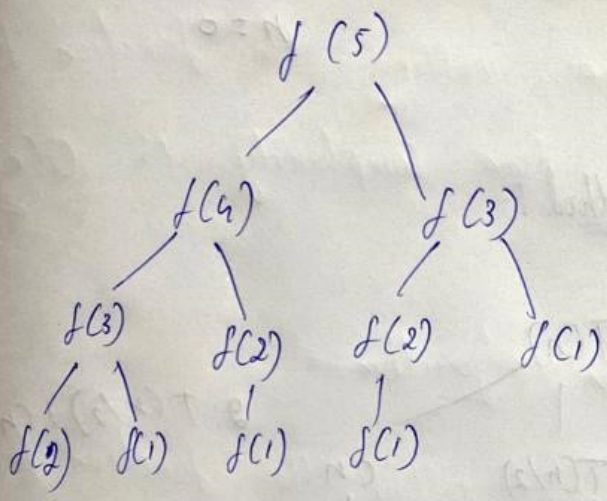
OUTPUT:



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.

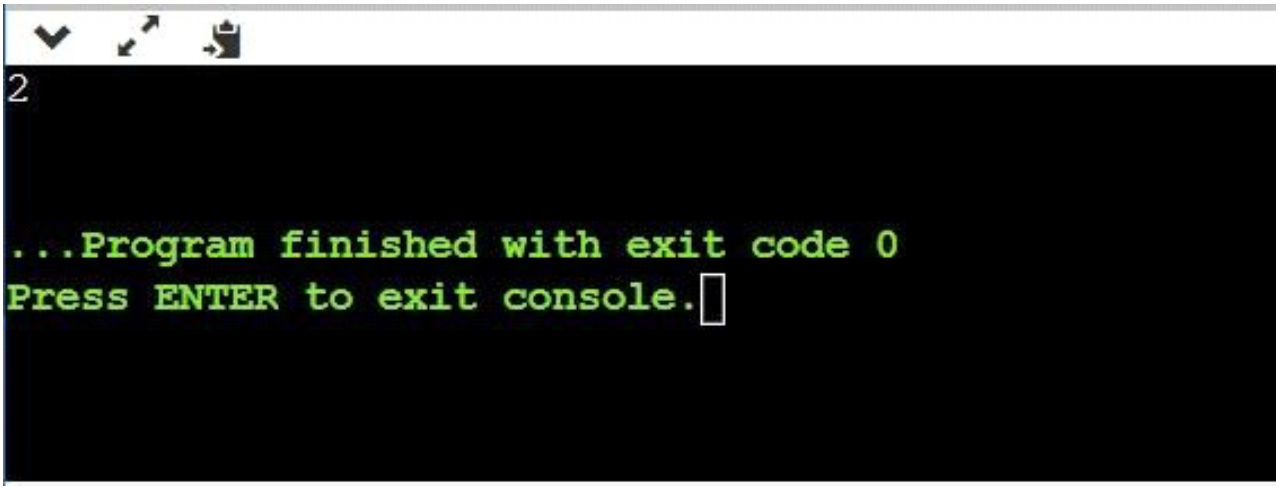
GCD

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:

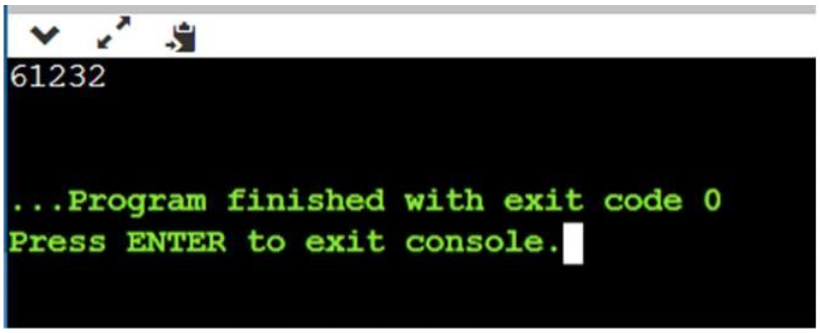


Using Euclidean Algorithm:

```
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:



ANALYSIS:

GCD:

1)Using naïve algorithm:

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

2)Using Evelieden Algorithm:

Here, we are calling the function for every 6 degree log remainder. So, time complexity is O(log n).