**Task 01)**Run the following code:

void countDown(int num)

 {

      if (num == 0) // test

                  System.out.println("Blastoff!");

       else {

                  System.out.println("...");

                    countDown(num-1); // recursive call

             }

}

what is the output of this code?

If my num was 7 the output would appear as follows

…

…

…

…

…

…

…

Blastoff!

- modify it so that it prints only the even numbers.

Public static void countDownOnlyEvens(int num) {

if (num == 0) {

System.out.println("Blastoff!");

}

else if (num % 2 == 0) {

System.out.println(num);

countDownOnlyEvens(num-1);

}

else {

System.out.println("...");

countDownOnlyEvens(num-1);// recursive call

}

}

- what is the time complexity of this algorithm and why?

I believe that the time complexity of this algorithm is O(n)

The method is called n number of times with recursive

The input for num is n

**Task 02) Run the following code:**

int gcd(int x, int y) {

      if (x % y == 0) //base case

             return y;

       else

             return gcd(y, x % y);

}

what is the output of this code?

If x was 27 and y was 63, the output would be 9.

This code allows you to find the greatest common divisor/factor of 2 integers

- modify it to find the gcd using subtraction instead of %.

Public static int gcdSubtraction(int x, int y) {

if (x % y == 0) { //base case

return y;

}

else {

if (x == y){

return x;

}

else if (x > y) {

return gcdSubtraction(x - y, y);

}

else {

return gcdSubtraction(x, y - x);

}

}

}

**Task 03) Run the following code:**

int fib(int n)

{

       if (n <= 0) // base case

            return 0;

      else if (n == 1) // base case

             return 1;

       else

             return fib(n – 1) + fib(n – 2);

}

what is the time complexity of this algorithm and why?

I believe the that the time complexity of the algorithm is O(2^n)

When neither of the base cases hit, 2 recursive calls occur