Give the order of growth (as a function of N ) of the running times of each of the following code fragments:

**Note:** use the following Notations for writing Answers: N, 1, log(N), N^2, N\*log(N), 2^N, N^3, log(N)

**Code-1:**

int count = 0;

for(int i = 0; i < N; i++)

{

         count++;

 }

Ans: N

because it is a for loop and in for loop it is repeated n times

**Code-2:**

int sum = 0;

    if(sum == 0){

    sum++;}

Ans: 1

Because if is executed for once here and it is a single statement. where loop is done only once

**Code-3:**

 for(int i = N; i > 0; i < N/2){

int sum = 0;}

Ans: log N

Because the value is half times

**Code-4:**

 for(int i = 0; i < N; i++){

                for(int j = 0; j < N; j++){

                    System.out.println(“Hello”);

                }

            }

Ans: N^2

Because it is nested loop- the outer loop is n times and inner loop is n times

total n\* n times repeating

O(n^2)

**Code-5:**

for(int i = 0; i < N; i++){

 for(int j = 0; j < N; j = j \* 2){

  System.out.println(“Hello”);

                }

            }

      Ans: N\*log(N)

because the outer loop is O(N) and inner loop is log(N)

**Code-6:**

public int fibonacci(int number) {

if (number <= 1) {

return number;

 }else {

      return fibonacci(number - 1) + fibonacci(number - 2);

   }

}

Ans: 1

because the recursion process is occurred ,so the constant value will be there.