From our recursive function,

public static void getFibonacci(long a, long b, int n, int ctr){

if(n==1){

System.out.println(a);

System.out.println("\nTotal function call/s: " + ctr);

}

else

getFibonacci(b, a+b, n-1, ctr+1);

}

the recurrence of our function is:

**T(n) = T(n-1) + 4 [1st call]**

**T(n-2) + 4 + 4 [2nd call]**

**T(n-3) + 4 + 4 + 4 [3rd call]**

**ith = T(n-i) + 4i**

Since our base case states that n should be equal to 1, then

**n – i = 1**

**n = i + 1**

**i = n – 1**

We then substitute n with i + 1 and we obtain,

**T(i + 1 – i) + 4i**

**T(1) + 4i**

We then substitute I with n – 1 and also T(1) with 3 since if n is 1, the frequency count is 3

**3 + 4(n-1)**

**3+4n – 4**

**4n – 1**

We have obtained the closed-form of our recurrence relation which is **4n - 1** and the big-Oh is O(n).