Instituto Politécnico Nacional

Escuela Superior de Cómputo

Análisis de Algoritmos

Ejercicios 01

Calcular el número de impresiones

Alumno:
Luis Fernando RESÉNDIZ CHÁVEZ

23 de octubre de 2020

1. Código 01

1.1. Código implementado por el profesor

```c
#include <stdio.h>
#include <math.h>

int main() {

  int i, n;
  scanf("%d", &n);

  for(i = 10; i < 5 * n; i *= 2) {
    printf("Algoritmo\n");
  }

  return 0;
}
```

1.2. Análisis del código

Partimos con la incógnita del problema, ¿Cuántas veces se imprime la la palabra algoritmo?
Las primeras observaciones al código fueron que únicamente existe una variable, en este caso N, por lo tanto, hay que encontrar dada una N, cuantos valores toma la variable i antes de que la condición $i < n * 5$ deje de cumplirse.
Como en el vídeo donde se explicó la complejidad de los algoritmos, hay que comenzar a darle valores a N, e ir anotando los resultados, para posteriormente ir encontrando el patrón y poder llegar a una conclusión.
Sin embargo, hay que ver como se comporta nuestra variable i:
Al principio del for inicia con un valor de 10, y por cada iteración, el valor se va multiplicando por 2, por lo tanto es fácil deducir que la función es $10 * 2^i$.
Entonces haciendo pruebas ahora si, con diferentes valores para N, solo hay que calcular para cuantos valores de i se cumple la condición del for:

$$10 * 2^i < 5 * N \tag{1}$$

Resolviendo la desigualdad para encontrar el valor de i obtenemos:

$$0 \leq i < \log_2(\frac{n}{2}) \tag{2}$$

El 0 acota a la i porque los exponentes empiezan desde 0.

Dentro del for la variable i siempre es un número entero, por lo tanto, los resultados que arroje esta función hay que redondear hacia el entero mas cercano. Después de probar la función con las N dadas por el profesor, me di cuenta que la función solo es válida para números mayores o iguales a 3 y que debe redondearse hacia arriba, es decir aplicar la función techo para obtener únicamente los valores enteros a los que i podría llegar a tomar, ya que en los resultados para $N = \{-1, 0, 1, 2\}$ el logaritmo no existía o me daba 0.

1.3. La función f(n)

$$f(n) = \begin{cases} 0 & si \quad n < 3 \\ \\ \lceil \log_2(\frac{n}{2}) \rceil & si \quad n \geq 3 \end{cases} \tag{3}$$

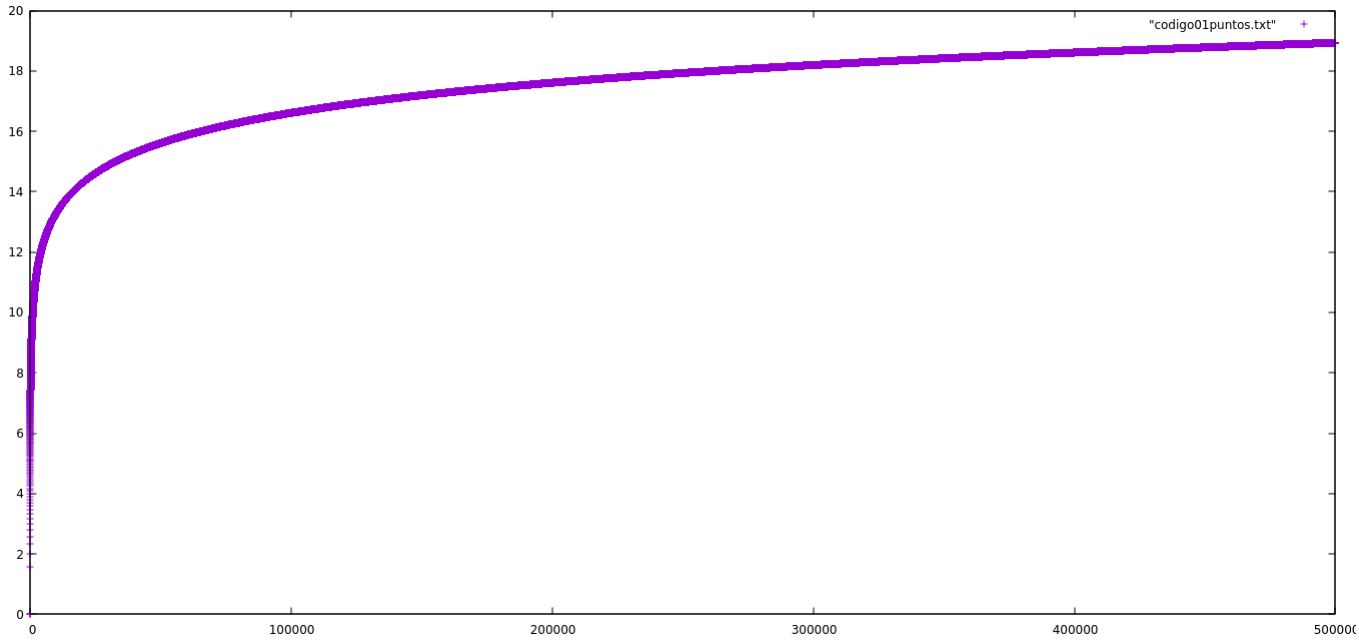
La función quedó de esta manera, los valores para N menores a 3, imprimirán 0 veces la palabra algoritmo y los mayores o iguales a 3 la función techo descrita.

1.4. Tabla comparativa de los resultados

Cuadro 1: Tabla comparativa del código 1

n	f(n)	Conteo Teórico	Conteo Empírico
-1	0.00000000000000000000	0	0
0	0.00000000000000000000	0	0
1	0.00000000000000000000	0	0
2	0.00000000000000000000	0	0
3	0.58496250072115629770	1	1
5	1.32192809488736218171	2	2
15	2.90689059560851870145	3	3
20	3.32192809488736262580	4	4
100	5.64385618977472525160	6	6
409	7.67595703294174924736	8	8
500	7.96578428466208698921	8	8
593	8.21188829454600366375	9	9
1000	8.96578428466208698921	9	9
1471	9.52258153125483097767	10	10
1500	9.55074678538324306487	10	10
2801	10.45172626807450733111	11	11
3000	10.55074678538324306487	11	11
5000	11.28771237954945050319	12	12
10000	12.28771237954945050319	13	13
20000	13.28771237954944872683	14	14

1.5. Gráfica muestral sobre 500,000 puntos



1.5.1. Código de la función f(n)

```
double f(int n) {
  return (n < 3) ? 0 : log2(n/2);
}
```

2. Código 02

2.1. Código implementado por el profesor

```c
#include <stdio.h>
#include <math.h>

int main() {

  int i, j, n;
  scanf("%d", &n);

  for(j = n; j > 1; j /= 2) {
    if(j < (n / 2)) {
      for(i = 0; i < n; i += 2) {
        printf("Algoritmo\n");
      }
    }
  }

  return 0;
}
```

2.2. Análisis del código

En este código hay que notar que tenemos mas de un ciclo for, por lo tanto nuestra función a encontrar que calcule en número de veces que se imprime la palabra algoritmo será polinomial. Entonces, comencemos con el primer for, donde la variable j toma de primer instancia el valor de la n dada y por cada iteración se va dividiendo entre 2 mientras se cumpla la condición de que sea mayor que 1. Primero calculamos los posibles valores que puede tomar j dada una n, rapidamente podemos darnos cuenta que el patrón es el siguiente: $\frac{n}{2^k}$, donde k son todos los valores que toma j, ahora generamos una desigualdad tal que $\frac{n}{2^k} > 1$, puesto que esa es la condición para que se siga cumpliendo el for, resolviendo dicha igualdad obtenemos que los posibles valores reales para j estan dentro del intervalo abierto $(1, \log_2 n)$. Al probar varios valores para n, desde 0 hasta 20, me di cuenta que de 0 a 1 j no cumple con la condición de ser mayor a 1, sino hasta $n \geq 2$, con dichas pruebas me di cuenta que la complejidad del primer for es $\lfloor \log_2 n \rfloor$, es decir, obtenemos solo los valores enteros a los que puede valer j.

La condicional if nos indica que se va a ejecutar el siguiente for, si y solo si, $j \leq \frac{n}{2}$ por lo tanto, la pregunta es ¿Cuántos valores toma j antes de que se comience a cumplir la condición de if ?, la respuesta es 2,$\{n, \frac{n}{2}\}$, por lo tanto lo único que tenemos que modificar a la función del for anterior es restarle 2, queda de la siguiente forma:

$$\lfloor \log_2 n \rfloor - 2 \tag{4}$$

Finalmente calculemos la complejidad del tercer for para poder encontrar cuantas veces se imprime la palabra Algoritmo, vemos que utiliza una variable diferente de j, la variable i, es decir, i es independiente de las complejidades calculadas anteriormente. La variable i comienza desde 0 hasta $n-1$ y en cada iteración se le va sumando 2. Haciendo pruebas con las mismos 21 valores para n (de 0 a 20), me di cuenta que la complejidad de ese for es $\lfloor \frac{n}{2} \rfloor$, ya que con números pares la división es exacta, pero con números impares i no alcanza a tomar el valor exacto de n, por lo tanto es función piso.

Después de probar la función con las N dadas por el profesor, me di cuenta que la función solo es válida para $n \geq 8$ ya que para valores menores a 8, no se cumplen las condiciones anteriores y el algoritmo no imprime ni una vez la palabra algoritmo.

2.3. La función f(n)

$$f(n) = \begin{cases} 0 & si \quad n < 8 \\ \\ \lceil \log_2(n) \rceil - 2 \cdot \lfloor \frac{n}{2} \rfloor & si \quad n \geq 8 \end{cases} \tag{5}$$

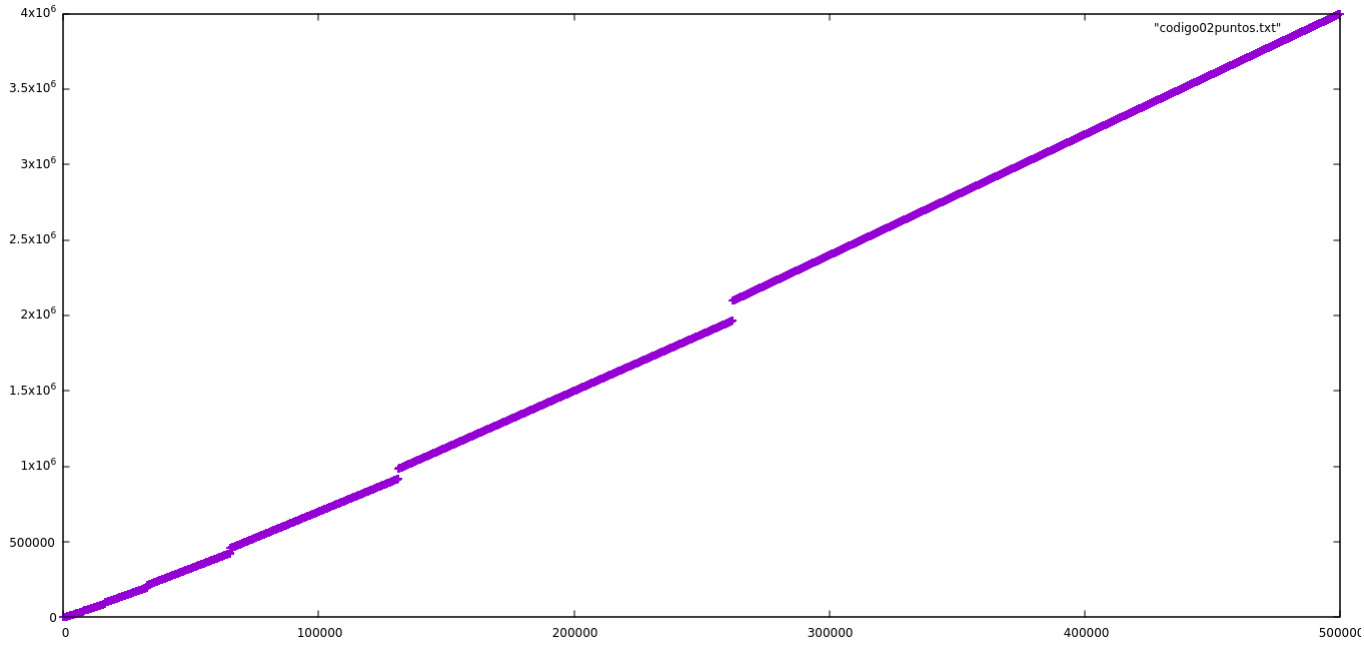
La función quedó de esta manera, los valores para n menores a 8, imprimirán 0 veces la palabra algoritmo y los mayores o iguales a 8 la función descrita.

2.4. Tabla comparativa de los resultados

Cuadro 2: Tabla comparativa del código 2

n	f(n)	Conteo Teórico	Conteo Empírico
-1	0.0000000000	0	0
0	0.0000000000	0	0
1	0.0000000000	0	0
2	0.0000000000	0	0
3	0.0000000000	0	0
5	0.0000000000	0	0
15	8.0000000000	8	8
20	20.0000000000	20	20
100	200.0000000000	200	200
409	1230.0000000000	1230	1230
500	1500.0000000000	1500	1500
593	2079.0000000000	2079	2079
1000	3500.0000000000	3500	3500
1471	5888.0000000000	5888	5888
1500	6000.0000000000	6000	6000
2801	12609.0000000000	12609	12609
3000	13500.0000000000	13500	13500
5000	25000.0000000000	25000	25000
10000	55000.0000000000	55000	55000
20000	120000.0000000000	120000	120000

2.5. Gráfica muestral sobre 500,000 puntos



2.5.1. Código de la función f(n)

```
double f(int n) {
  return (n < 8) ? 0 : (floor(log2(n))-2.0) * ceil(n/2.0);
}
```

3. Código 03

3.1. Código implementado por el profesor

```c
#include <stdio.h>
#include <math.h>

int main() {

  int i, j, k, n;
  scanf("%d", &n);

  for(i = 0; i < 5 * n; i += 2) {
    for(j = 0; j < 2 * n; ++j) {
      for(k = j; k < n; ++k) {
        printf("Algoritmos\n");
      }
    }
  }

  return 0;
}
```

3.2. Análisis del código

En este tercer algoritmo, nos podemos dar cuenta de que existen 3 variables distintas, $\{i, j, k\}$, cada una de ellas controlan un for, por lo tanto es necesario calcular la complejidad de cada ciclo, para encontrar una función en términos de n que nos diga cuantas veces se imprime la palabra algoritmo. Comencemos por el primer ciclo que controla la variabe i, partimos con la observación de que i inicia en 0 y se le va sumando 2 hasta que su valor sea menor que $5(n)$, por lo tanto la función de complejidad en primera instancia es $\frac{5n}{2}$, ahora hay que probar con algunos valores de n, y ver su comportamiento
Una vez realizado el experimento con valores para n en el rango $[0, 100]$, me di cuenta que los valores impares de n, la función debe redondearse hacia arriba para obtener las veces exactas en que se repite éste ciclo for, por lo tanto la función final para este ciclo es:

$$\lceil \frac{5n}{2} \rceil \tag{6}$$

Continuando con el segundo for, aparece una variable independiente del for anterior, j, por lo tanto podemos calcular su complejidad de forma independiente, de primera instancia nos damos cuenta que la variable j inicia en 0 y en cada iteración incrementa en una unidad, esto se repite mientras que su valor sea menor a $2n$, fácilmente podemos deducir que la función de este ciclo es:

$$2(n) \tag{7}$$

Finalmente, el tercer y último for, en él se involucran dos variables, j y k, donde k depende del valor que vaya tomando j hasta n, esto se convierte en una suma de Gauss. Por lo tanto al multiplicarse con la ecuación anterior, obtenemos:

$$\frac{n}{2}(n+1) \tag{8}$$

3.3. La función f(n)

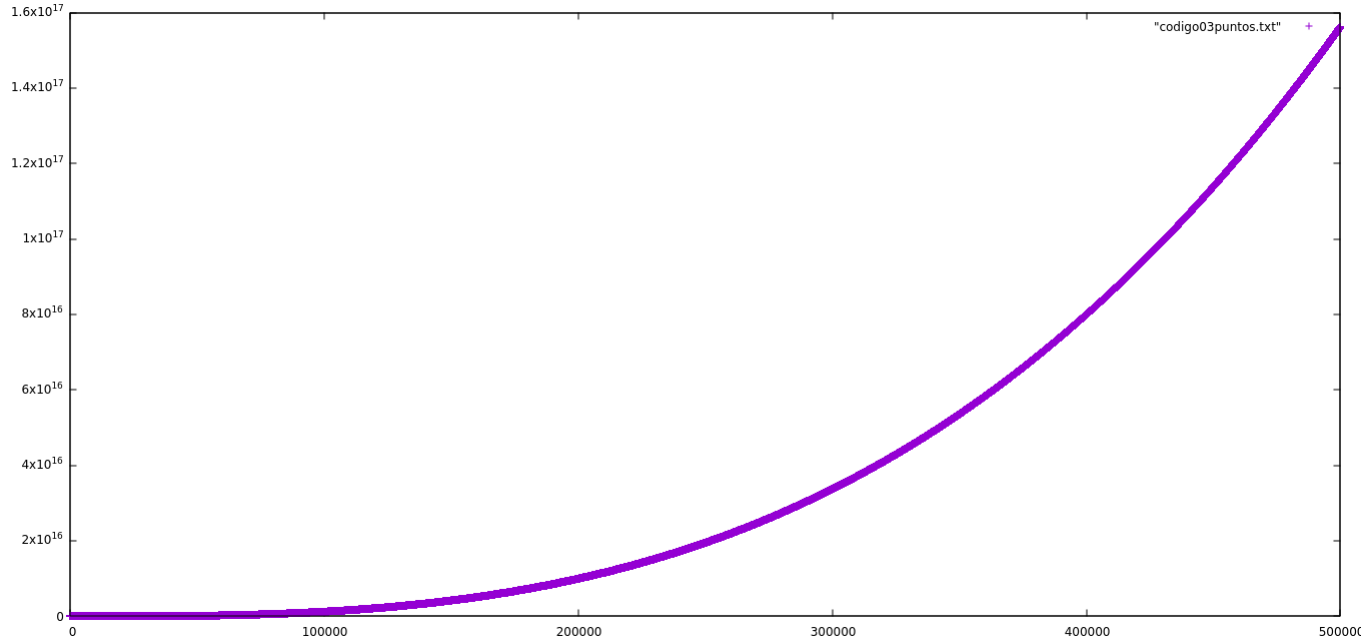
$$f(n) = \begin{cases} 0 & si \quad n \leq 0 \\ \\ \lceil \frac{5n}{2} \rceil \cdot \frac{n}{2}(n+1) & si \quad n > 0 \end{cases} \tag{9}$$

3.4. Tabla comparativa de los resultados

Cuadro 3: Tabla comparativa del código 2

n	f(n)	Conteo Teórico	Conteo Empírico
-1	0.0000000000	0	0
0	0.0000000000	0	0
1	3.0000000000	3	3
2	15.0000000000	15	15
3	48.0000000000	48	48
5	195.0000000000	195	195
15	4560.0000000000	4560	4560
20	10500.0000000000	10500	10500
100	1262500.0000000000	1262500	1262500
409	85773435.0000000000	85773435	85773435
500	156562500.0000000000	156562500	156562500
593	261187443.0000000000	261187443	261187443
1000	1251250000.0000000000	1251250000	1251250000
1471	3982008768.0000000000	3982008768	3982008768
1500	4221562500.0000000000	4221562500	4221562500
2801	27481179603.0000000000	27481179603	27481179603
3000	33761250000.0000000000	33761250000	33761250000
5000	156281250000.0000000000	156281250000	156281250000
10000	1250125000000.0000000000	1250125000000	1250125000000
20000	10000500000000.0000000000	10000500000000	10000500000000

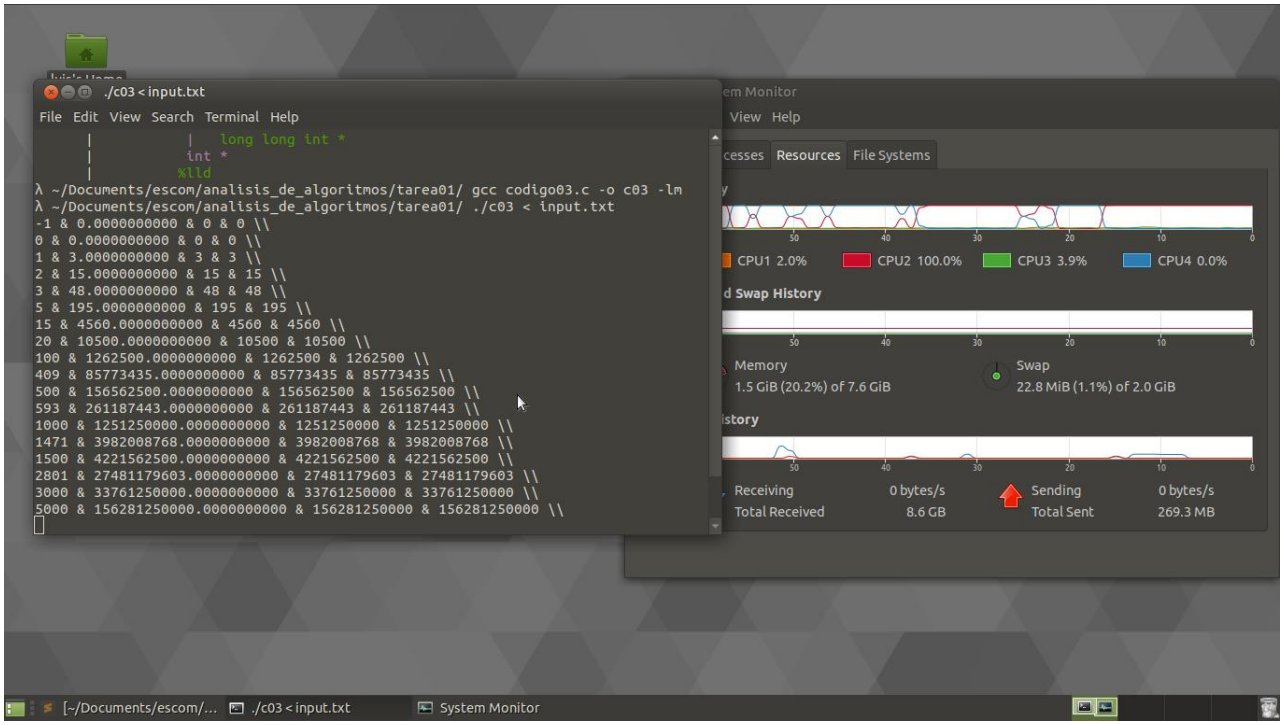
3.5. Gráfica muestral sobre 500,000 puntos



3.5.1. Código de la función f(n)

```
long double f(int x) {
  long double n = (double)x;
  return ceil(5.0*n/2.0)*(n/2.0*(n+1.0));
}
```

3.5.2. Pruebas de testeo



4. Código 04

4.1. Código implementado por el profesor

```c
#include <stdio.h>
#include <math.h>

int main() {

  int i, j, n;
  scanf("%d", &n);

  i = n;
  while(i >= 0) {
    for(j = n; i < j; i -= 2, j /= 2) {
      printf("Algoritmo\n");
    }
  }

  return 0;
}
```

4.2. Análisis del código

Al parecer en éste código nunca se imprime la palabra algoritmo, primero la variable i toma el valor de n, después el ciclo *while* tiene la condición de que i nunca puede ser negativa, por lo tanto la primer función es: $f(n) = 0, n < 0$.

Únicamente con $n \geq 0$, la condición del *while* se cumple y pasa al for la variable j toma el valor de n y el ciclo se cumple mientras $i < j$, pero en el primer instante en el que se entra al for: i es igual a n y j es igual n, entonces i es igual a j, por lo tanto nunca entra al for, hace un salto falso y no hay otra instrucción que controle el comportamiento de la variable i dentro del *while*, por lo tanto el programa queda en un ciclo infinito y nunca imprime la palabra algoritmo.

4.3. La función f(n)

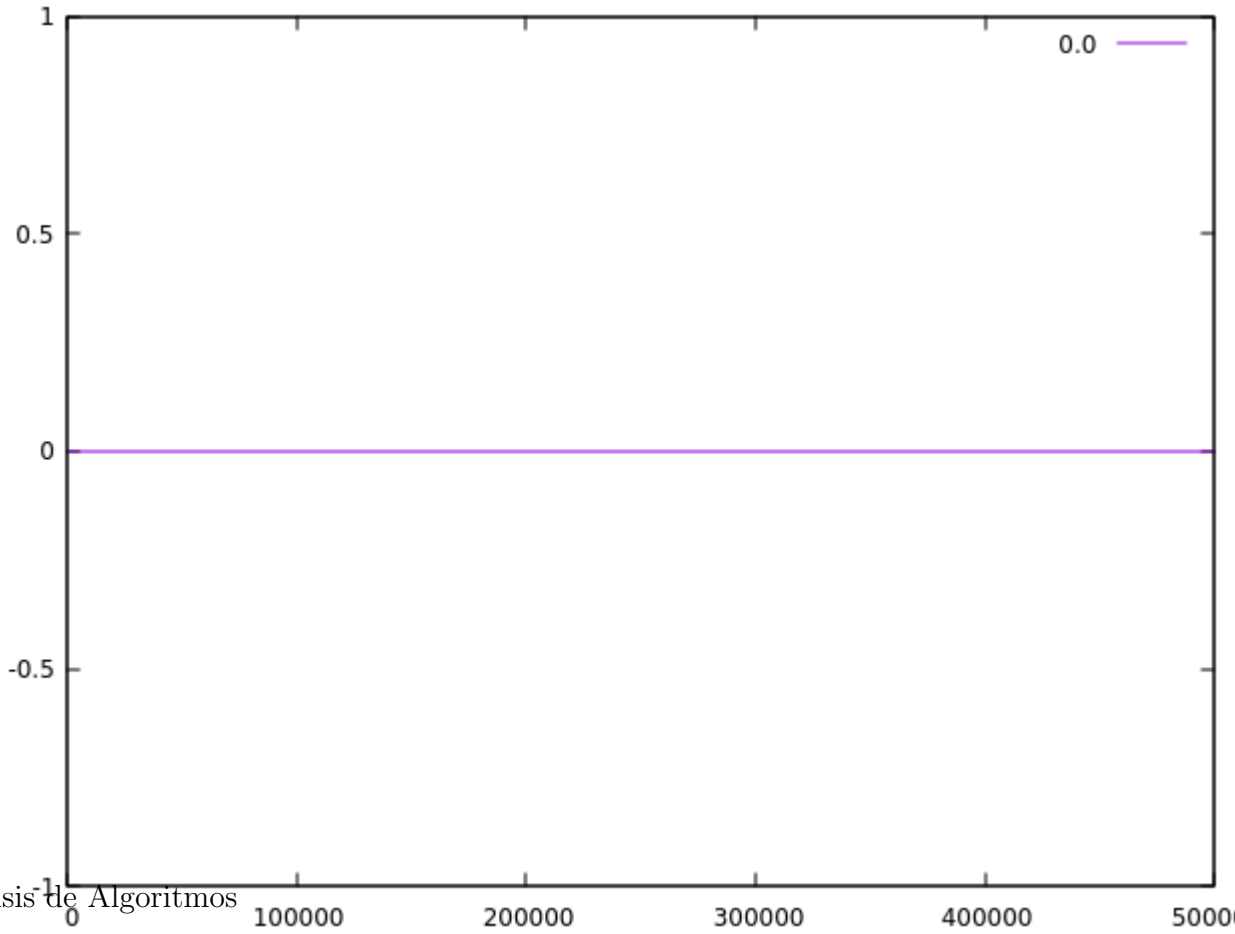
$$f(n) = 0 \tag{10}$$

Cuadro 4: Tabla comparativa del código 2

n	f(n)	Conteo Teórico	Conteo Empírico
-1	0.0000000000	0	0
0	0.0000000000	0	...
1	0.0000000000	0	...
2	0.0000000000	0	...
3	0.0000000000	0	...
5	0.0000000000	0	...
15	0.0000000000	0	...
20	0.0000000000	0	...
100	0.0000000000	0	...
409	0.0000000000	0	...
500	0.0000000000	0	...
593	0.0000000000	0	...
1000	0.0000000000	0	...
1471	0.0000000000	0	...
1500	0.0000000000	0	...
2801	0.0000000000	0	...
3000	0.0000000000	0	...
5000	0.0000000000	0	...
10000	0.0000000000	0	...
20000	0.0000000000	0	...

4.4. Tabla comparativa de los resultados

4.5. Gráfica muestral sobre 500,000 puntos



4.5.1. Código de la función f(n)

```
int f(int n) {
    return 0;
}
```

5. Código 05

5.1. Código implementado por el profesor

```c
#include <stdio.h>
#include <math.h>

int main() {

  int i, j, n;
  scanf("%d", &n);

  for(i = 1; i < 4 * n; i *= 2) {
    for(j = i; j < 5 * n; j += 3) {
      printf("Algoritmo\n");
    }
  }

  return 0;
}
```

5.2. Análisis del código

El primer for es controlado por la variable i, la cual itera desde 1 hasta $4n$ y por cada iteración se duplica, por lo tanto, la función que calcula su complejidad queda en términos de logaritmo base 2, de la siguiente manera:

$$\log_2 4n \tag{11}$$

Después, el siguiente for es controlado por la variable j pero que depende del valor que vaya tomando i, el for se cumple mientras $j < 5n$ y se incrementa de 3 en 3, por lo tanto se forma una suma de Gauss:

$$\frac{5n(5n-1)}{3} \tag{12}$$

Ahora hay que darnos cuenta que entonces nuestra función se vuelve una sumatoria para el valor que vaya tomando j. Hasta este punto mi función final iba a ser:

$$f(n) = \lfloor \frac{5n\log_2 n + 2}{3} \rfloor \tag{13}$$

Evaluando las $n's$ de prueba que nos dió, algunos valores eran correctos pero la mayoría no, por lo tanto hay que volver a replantear el problema.

Los valores hasta donde pueden llegar las variables i y j, son $4n$ y $5n$ respectivamente. Entonces podemos verlos como rangos, la i va desde 1 hasta $4n$ y j desde i hasta $5n$, cada vez que j avanza (se multiplica por 2) obviamente la i tambien empieza desde una posición mayor, pero hay que darnos cuenta que los rangos $4n$ y $5n$ nunca cambian, ambas variables tiene que terminar sus respectivos rangos, por lo tanto, cada vez que la j se multiplica, esa cantidad se la estamos quitando a la variable i, sucede el siguiente patrón: $5n-1, 5n-2, 5n-4, ...$, comienza desde $5n-1$ porque el for nos está indicando que no puede tocar el $5n$, generalizando la idea, queda:

$$5n - 2^{t-1} \tag{14}$$

Donde t, es cada iteración de la variable i, solo falta ver el caso en que para una siguiente iteración, el rango $4n$ ya no es suficiente, por lo tanto hay que agregarle la función piso:

$$\lfloor 5n - 2^{t-1} - 1 \rfloor \tag{15}$$

Ahora, la variable j en el segundo for va aumentando de 3 en 3 por lo tanto hay que dividir entre 3 y sumar 1, porque el valor inicial de j ya es válido:

$$g(n,t) = \lfloor \frac{5n - 2^{t-1} - 1}{3} \rfloor + 1 \tag{16}$$

Con esta función ya podemos encontrar cuantas veces se imprime la palabra algoritmo, dada una n y en un tiempo t, sin embargo, ahora tenemos que calcular el número de segmentos que nos dará la variable i en el primer for, por lo tanto se convierte en una suma de los resultados de ésta función, la variable t se convierte en el conteo de iteraciones de la i, no su valor, el conteo de cuantas veces es válida. La n se vuelve constante y queda de la siguiente manera:

$$g(t) = \lfloor \frac{5n - 2^{t-1} - 1}{3} \rfloor + 1 \tag{17}$$

La función suma es:

$$f(n) = g(1) + g(2) + g(3) + ... + g(n-1) + g(n) \tag{18}$$

Pero la variable i no llega solo hasta n, llega hasta $4n$ y su aumento va multiplicandose por 2, entonces sería:

$$f(n) = g(1) + g(2) + g(4) + g(8)... + g(4n-1) \tag{19}$$

Entonces la cota que nos dice cuantos valores de i son validos es:

$$\lceil \log_2 4n \rceil \tag{20}$$

Finalmente, la suma de éstas funciones es la que nos dirá cuantas veces se imprime la palabra algoritmo en el programa:

5.3. La función f(n)

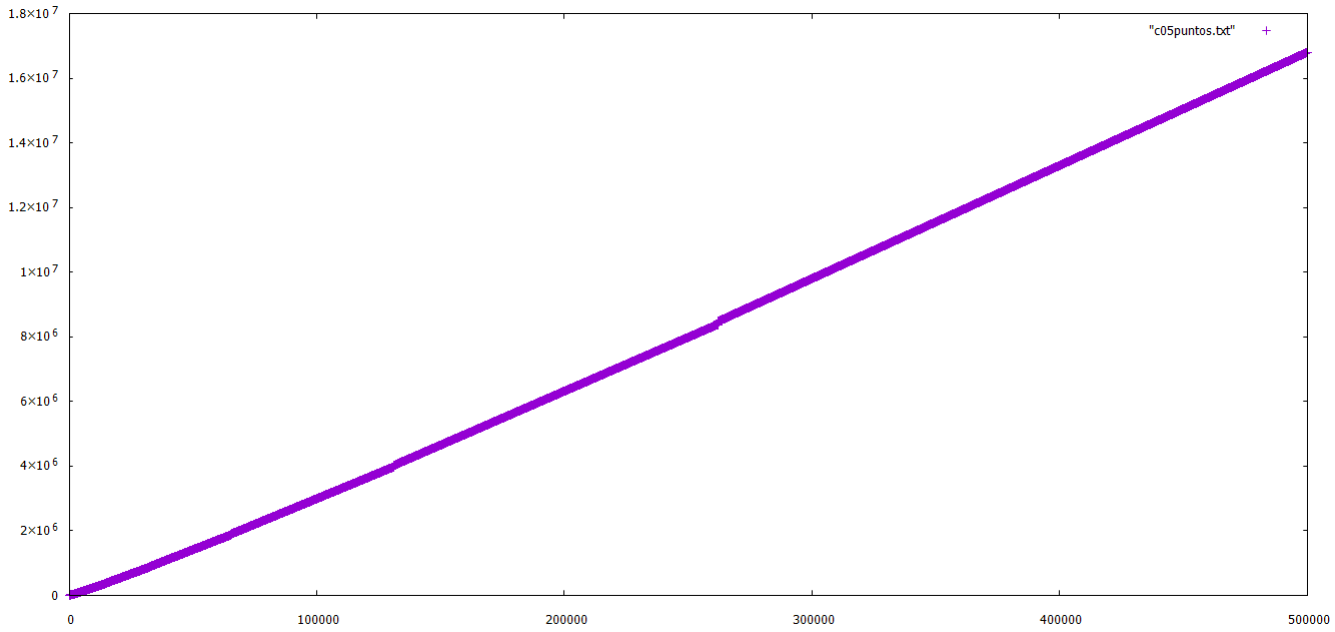
$$f(n) = \begin{cases} 0 & si \quad n \leq 0 \\ \sum_{t=1}^{\lceil \log_2 4n \rceil} \lfloor \frac{5n - 2^{t-1} - 1}{3} \rfloor + 1 & si \quad n > 0 \end{cases} \tag{21}$$

Cuadro 5: Tabla comparativa del código 2

n	f(n)	Conteo Teórico	Conteo Empírico
-1	0.0000000000	0	0
0	0.0000000000	0	0
1	3.0000000000	3	3
2	8.0000000000	8	8
3	17.0000000000	17	17
5	32.0000000000	32	32
15	132.0000000000	132	132
20	192.0000000000	192	192
100	1333.0000000000	1333	1333
409	6820.0000000000	6820	6820
500	8486.0000000000	8486	8486
593	10497.0000000000	10497	10497
1000	18639.0000000000	18639	18639
1471	29146.0000000000	29146	29146
1500	29776.0000000000	29776	29776
2801	59898.0000000000	59898	59898
3000	64546.0000000000	64546	64546
5000	114080.0000000000	114080	114080
10000	244827.0000000000	244827	244827
20000	522979.0000000000	522979	522979

5.4. Tabla comparativa de los resultados

5.5. Gráfica muestral sobre 500,000 puntos



5.5.1. Código de la función f(n)

```
double f(int n) {
    int conteoTeorico = 0, t, dosALaT;

    for(t = 1, dosALaT = 1; t <= ceil(log2(4.0 * (double)n)); t++, dosALaT *= 2) {
        conteoTeorico += (double)floor((5.0 * n - dosALaT - 1) / (3.0)) + 1;
    }

    return conteoTeorico;
}
```