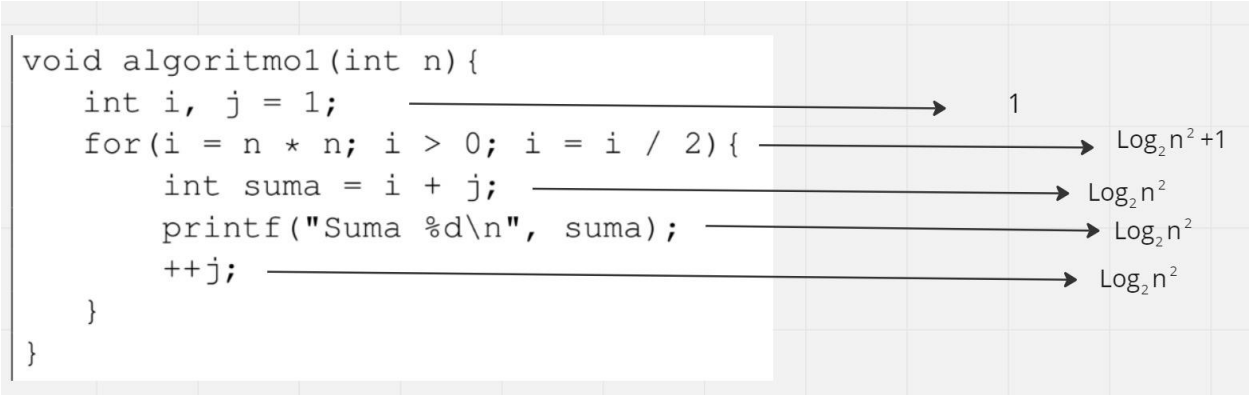
Tarea 2

Rui Yu Lei Wu

Marco Antonio Riascos

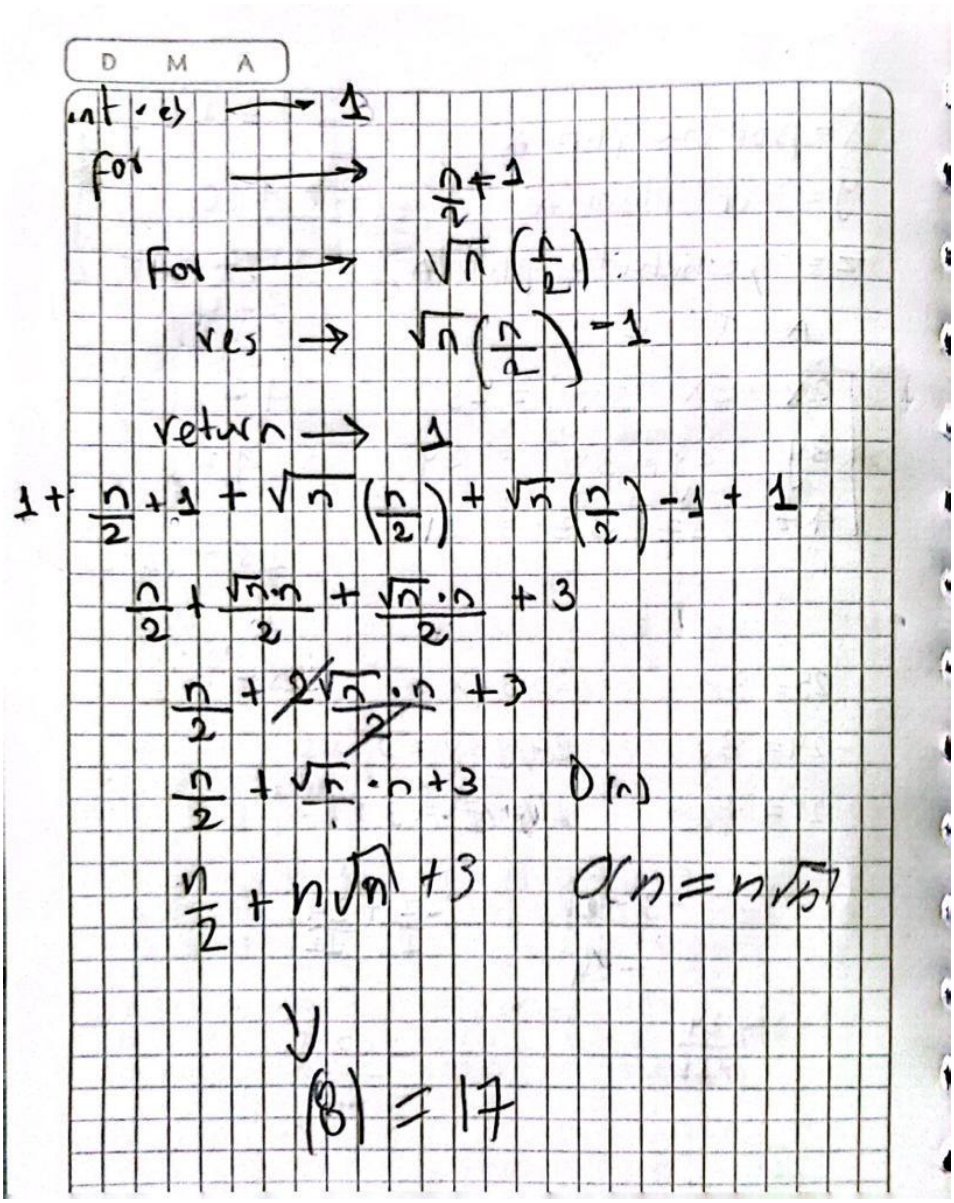
1. $O(n)=Log_2(n^2)$



Algoritmo(8)→8
Debido a que es un logaritmo base 2 de 64 se expresa como $n^2/2^i$, siendo i el numero de veces que el numero de veces que itera el for, esto significa que el numerador se va dividiendo en dos hasta que el entero más pequeño es 1, y la j suma 1 cada vez que 64 se divide entre 2 . De esta manera:

$1.64/2^0+1=65$
$2.64/2^1+2=34$
$3.64/2^2+3=19$
$4.64/2^3+4=12$
$5.64/2^4+5=9$
$6.64/2^5+6=8$
$7.64/2^6+7=8$

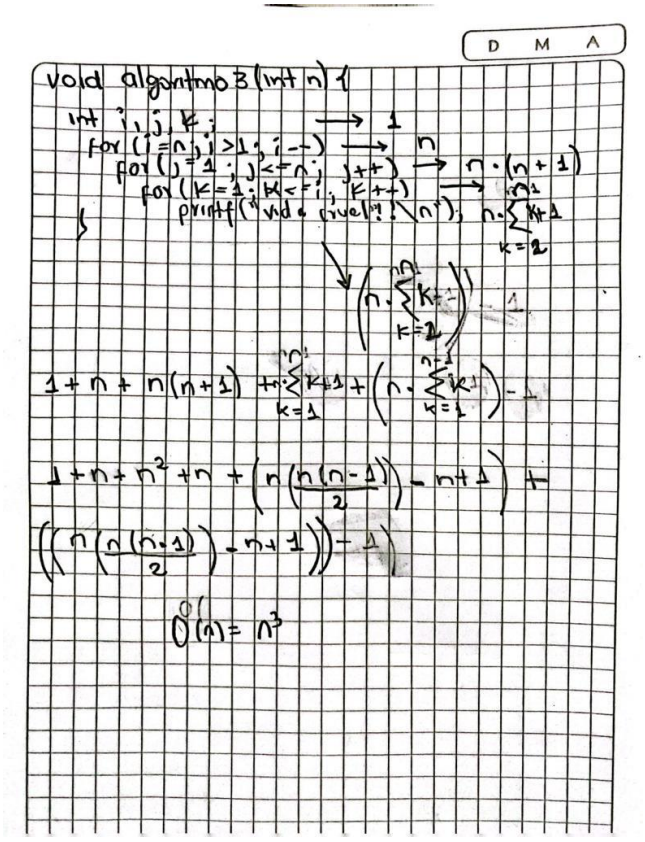
2. $O(n)=n\sqrt{n}$



Algoritmo2(8)→17

Debido a que el segundo for lo realiza $\sqrt{n}$ veces, dando un resultado entero de 2, ósea cada que se ejecuta el segundo for suma 4 al resultado, y si agregamos el primer for que se ejecuta entra n/2 veces al segundo for, nos da que el segundo for se va a ejecutar completamente 4 veces, entonces va a sumar 4, un total de 4 veces al resultado.

3. $O(n)=n^3$



4. No entendimos la complejidad del algoritmo.
5. O(1)

```
void algoritmo5(int n){
  int i = 0;                    → 1
  while(i <= n){                → 7
    printf("%d\n", i);          → 6
    i += n / 5;                 → 6
  }
}
```

6.

Tamaño de entrada	Tiempo	Tamaño de entrada	Tiempo
5	0.022s	35	3.122s
10	0.021s	40	34.39s

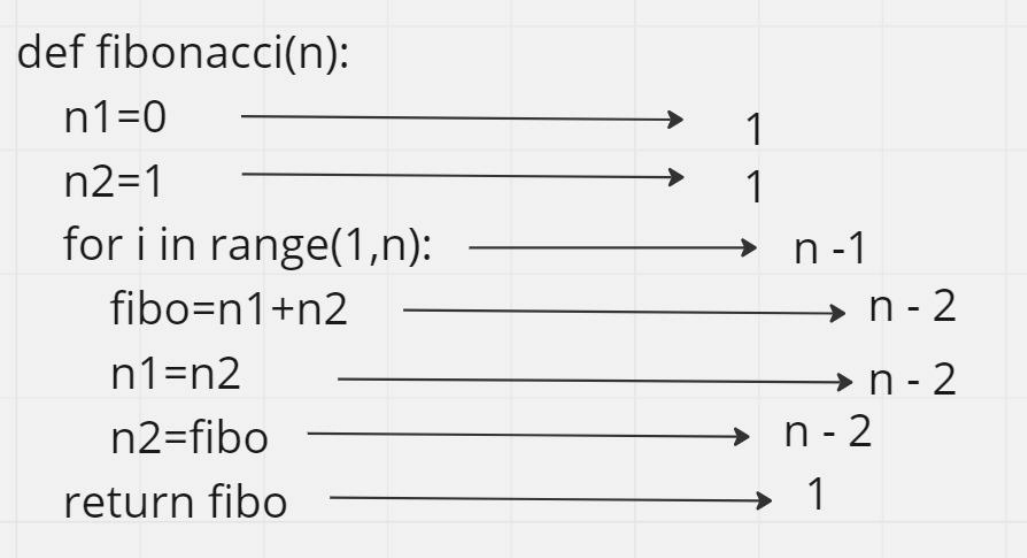
15	0.020s	45	6.20m
20	0.024s	50	71,8 m
25	0.047s	60	
30	0.300s	100	

Hasta 50, al hacer tanto trabajo al llamarse a ella misma el tiempo aumenta mucho cuando el numero va aumentando.

Nosotros creemos que es muy difícil hallar la complejidad de este ejercicio, ya que en realidad es muy incierto lo que la función va a realizar. Si tuviéramos que dar un valor diríamos que el valor es exponencial.

7.

Tamaño de entrada	Tiempo	Tamaño de entrada	Tiempo
5	0.025s	45	0.020s
10	0.021s	50	0.026s
15	0.031s	100	0.021s
20	0.020s	200	0.022s
25	0.021s	500	0.021s
30	0.029s	1000	0.026s
35	0.030s	5000	0.023s
40	0.024s	10000	0.026s



8.

Tamaño de entrada	Tiempo Solución Propia	Tiempo Solución Profesores
100	0.023s	0.023s
1000	0.031s	0.024s
5000	0.252s	0.030s

10000	0.866s	0.038s
50000	18.190s	0.147s
100000	1.9026m	0.334s
200000	4.17950m	0.836s

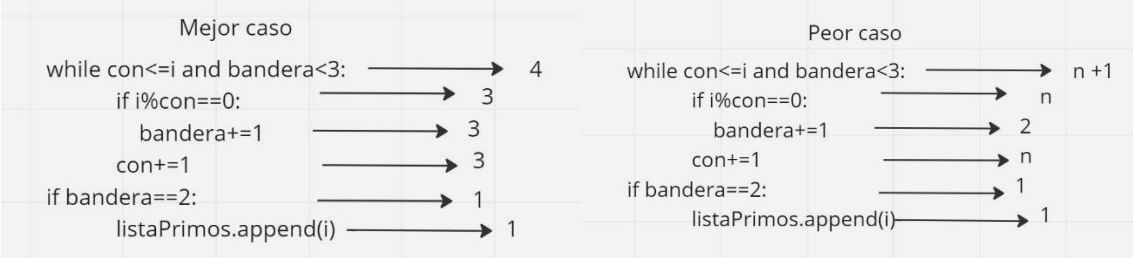
a. Con números bajos la diferencia no es tan notoria, pero cuando se ejecuta con números con mas de tres ceros la diferencia es muy notoria, nosotros diríamos que esta diferencia se debe a que la solución de los profesores es mucho más eficiente, y esta mucho mejor codificado. Además utiliza de mejor manera funciones auxiliares.

   En conclusión la codificación usa conceptos que le permiten al código ahorrarse mucho tiempo, un ejemplo de ello es que en el segundo código no hay necesidad de volver el número primo a string.

b. Solución propia:

   Mejor caso: Que el número no sea primo y es divisible por 2 y 3, O(1)

   Peor caso: Si el número es primo, O(n)=n+1



   Solución profesores:
   Mejor caso: Que n>2, de esta manera O(1)
   Peor caso: Que sea primo, entonces $\sum_{i=i*i}^{n} i + 1$