Algorítmica Carlos Quesada Pérez

2º GII Miguel Ángel López Sánchez

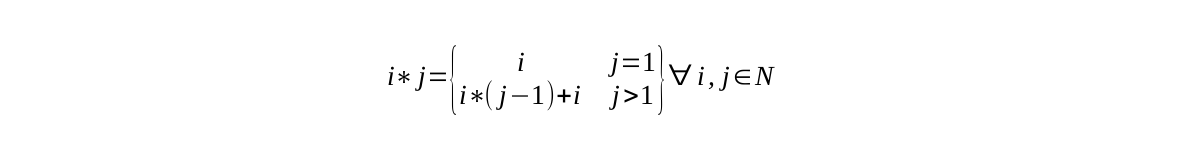
Grupo B3 David Serrano Domínguez

**Práctica 2**

**Ejercicio 1**

**(1.1) Diseño del algoritmo básico.**

Se pide resolver el problema de la multiplicación de números naturales sin la operación de multiplicación. Se define la operación de la siguiente forma:



Por tanto, el algoritmo básico a diseñar es bastante sencillo e intuitivo: sumar ‘i’ a sí mismo ‘j’ veces.

*Algoritmo M=MultiplicaBasico(i, j: numeros naturales(>0))*

*Si j=1, hacer:*

*M=copia de i*

*Devolver M*

*En otro caso:*

*M=0*

*Mientras j>1, hacer:*

*M=M+i*

*j=j-1*

*Fin-Mientras*

*Devolver M*

**(1.2) Análisis y diseño de componentes del algoritmo Divide y Vencerás.**

Este problema cumple con los requisitos para poder aplicar la técnica de Divide y Vencerás: permite la división del problema en subproblemas, existe un caso base, permite la combinación de soluciones.

Al ser un proceso de cálculo iterativo en el cual los datos no dependen unos de otros se puede dividir equitativamente el proceso en tantas partes como se desee, en este caso decidimos dividirlo en K=2 subproblemas. En este problema el caso base se da cuando j=1, en cuyo caso se devuelve el valor de i. La combinación de las soluciones parciales también es simple, consiste en sumar todas las soluciones parciales obtenidas.

**(1.3) Diseño del algoritmo Divide y Vencerás y función de combinación.**

Adaptando el diseño del algoritmo básico y teniendo en cuenta lo explicado en el apartado anterior, aplicando la plantilla DyV el algoritmo queda de la siguiente forma:

*Algoritmo M=multiplicaDyV(i, j: numeros naturales(>0))*

*Si j=1, hacer: #Caso base de la recurrencia*

*M=copia de i*

*Devolver M*

*En otro caso: #Caso general de la recurrencia*

*#Dividir el producto en dos subproductos más pequeños*

*M1=multiplicaDyV(i, floor(j/2)) #floor redondea hacia abajo*

*M2=multiplicaDyV(i, ceil(j/2)) #ceil redondea hacia arriba*

*M=M1+M2*

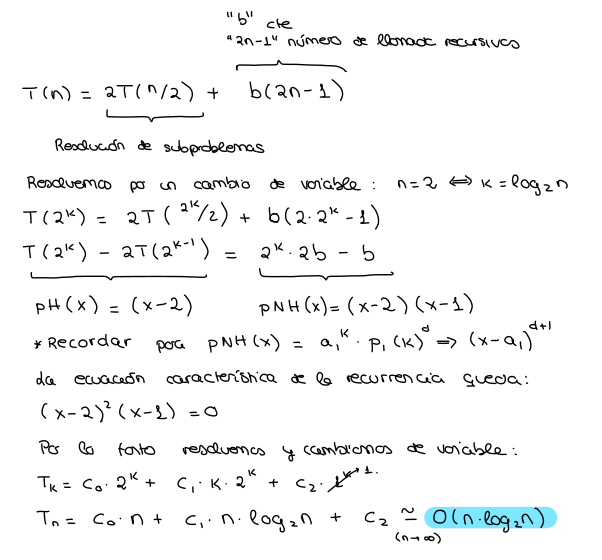
*Devolver M*

**(1.4) Análisis de eficiencia de los métodos básico y Divide y Vencerás. Justificación de si el diseño Divide y Vencerás realizado mejora al algoritmo básico.**

Según como está pensado y diseñado el algoritmo, se considera el segundo natural del producto i\*j como el tamaño del caso del problema (por tanto, n=j).

La eficiencia del algoritmo básico es muy simple, puesto que sólo contiene un bucle de j iteraciones y algunas asignaciones constantes. Su eficiencia por tanto es lineal O(n).

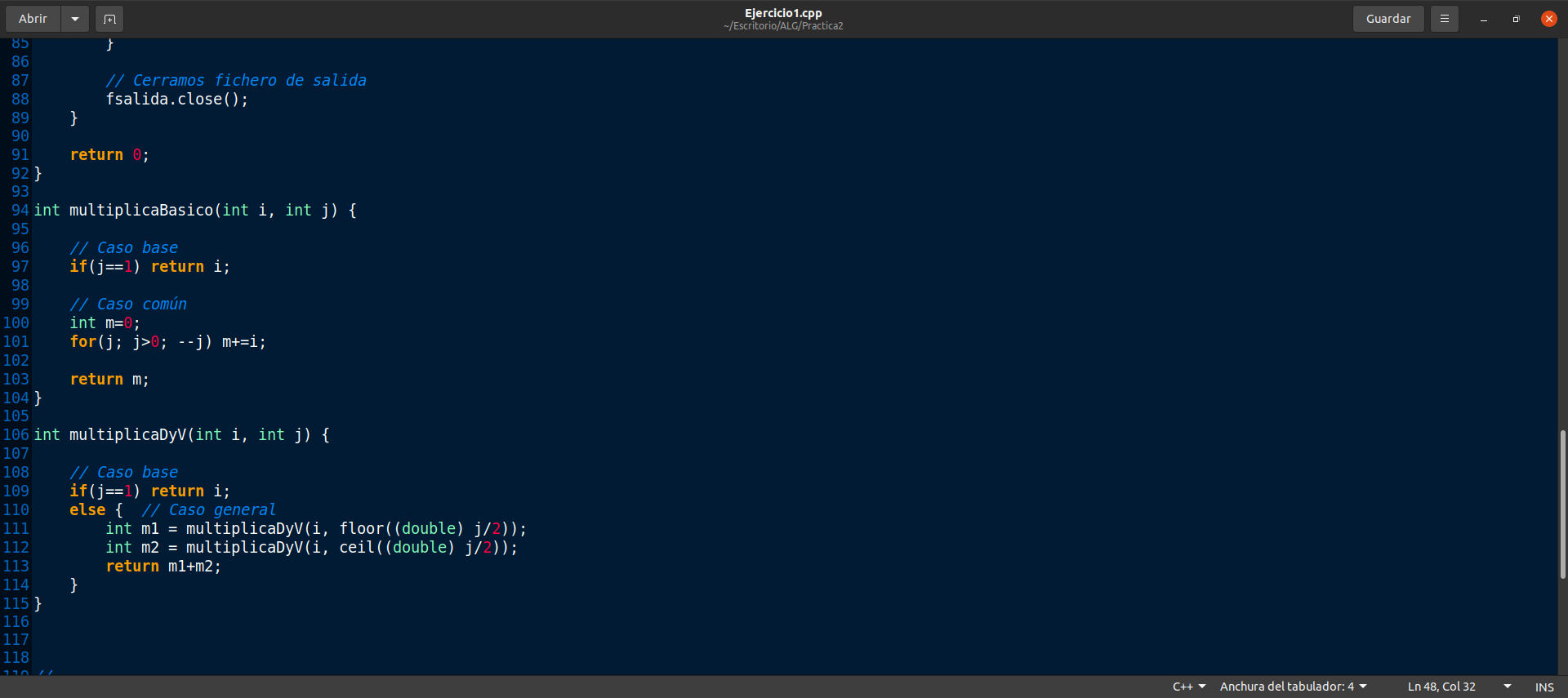
La eficiencia del algoritmo Divide y Vencerás es algo más complejo:



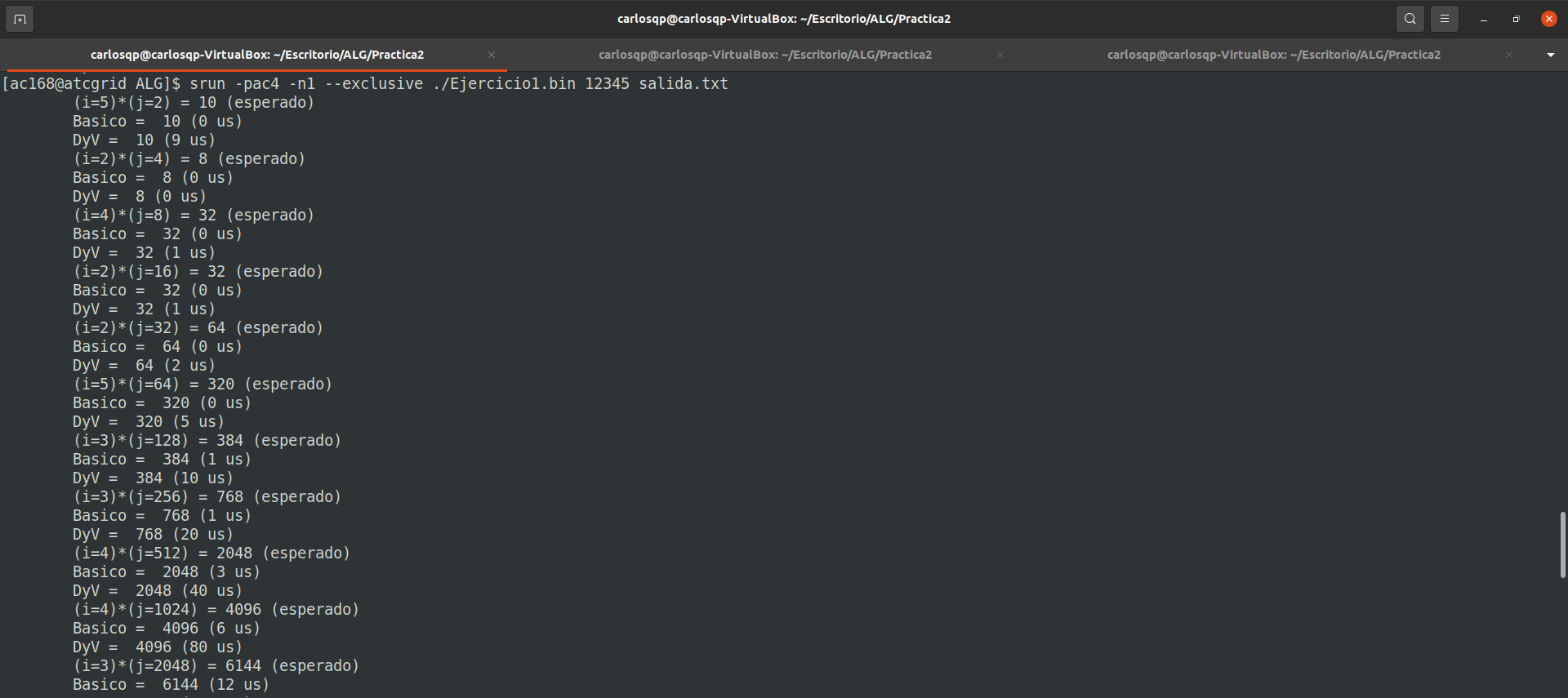
Si comparamos ambas eficiencias, observamos que el algoritmo básico (O(n)) es mejor que el algoritmo Divide y Vencerás (O(n\*log2(n)). Esta diferencia de eficiencias se debe a las sumas de combinación de los subproblemas. Tiene sentido que no sea posible mejorar a dicho algoritmo puesto que la ejecución del básico es lineal, y con la técnica de Divide y Vencerás se debe de ejecutar el básico K veces y combinar las K soluciones parciales.

**(1.5) Implementación de los métodos y pruebas de ejecución.**

Implementación del los métodos:

****

Prueba de ejecución: Ejercicio1.cpp



Resultados obtenidos:

| **Tamaño (j)** | **T. Basico (us)** | **T. DyV(us)** |
| --- | --- | --- |
| 2 | 0 | 9 |
| 4 | 0 | 0 |
| 8 | 0 | 1 |
| 16 | 0 | 1 |
| 32 | 0 | 2 |
| 64 | 0 | 5 |
| 128 | 1 | 10 |
| 256 | 1 | 20 |
| 512 | 3 | 40 |
| 1024 | 6 | 80 |
| 2048 | 12 | 161 |
| 4096 | 25 | 349 |
| 8192 | 51 | 660 |
| 16384 | 102 | 1302 |
| 32768 | 205 | 2609 |
| 65536 | 410 | 5208 |
| 131072 | 771 | 9242 |
| 262144 | 1449 | 16116 |
| 524288 | 2399 | 26833 |
| 1048576 | 3765 | 41843 |
| 2097152 | 5849 | 65044 |
| 4194304 | 10020 | 126296 |
| 8388608 | 20034 | 252594 |
| 16777216 | 40079 | 390837 |
| 33554432 | 52612 | 662412 |
| 67108864 | 105047 | 1325046 |
| 134217728 | 210106 | 2649876 |
| 268435456 | 420208 | 5299396 |

Tabla I: Ejecución hasta tamaño de caso 2^28. Se observa como la eficiencia es mucho mejor para el algoritmo básico.

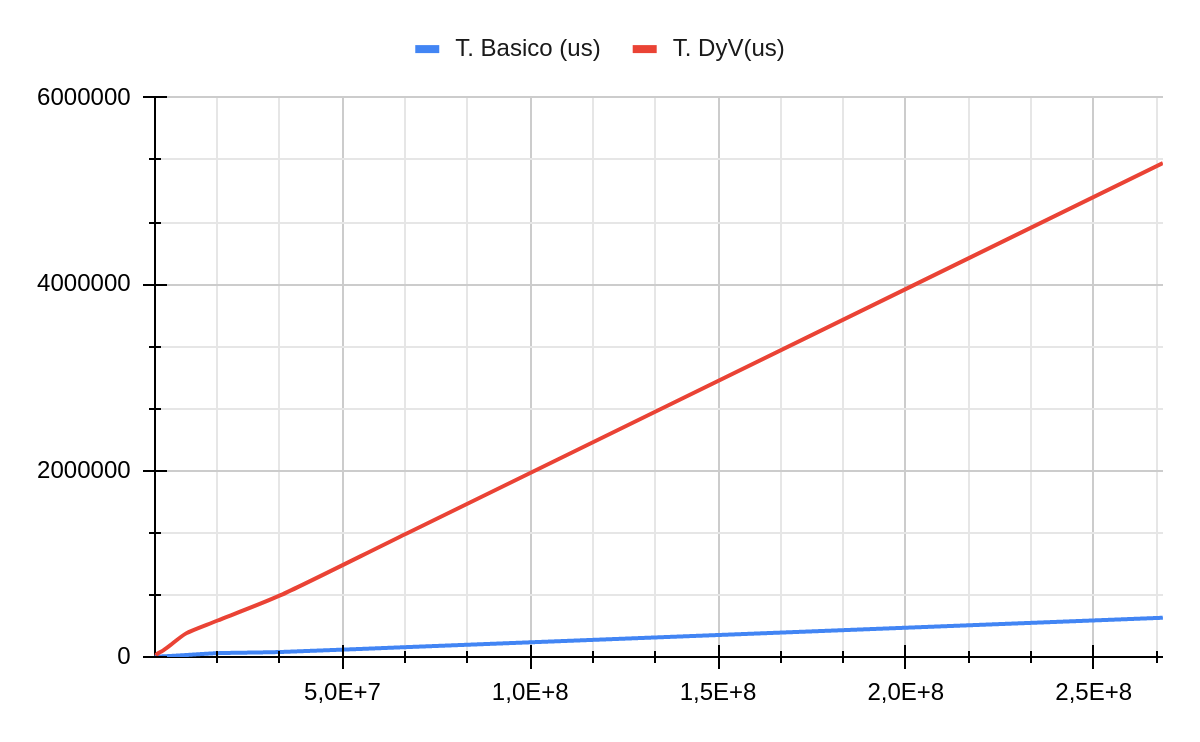
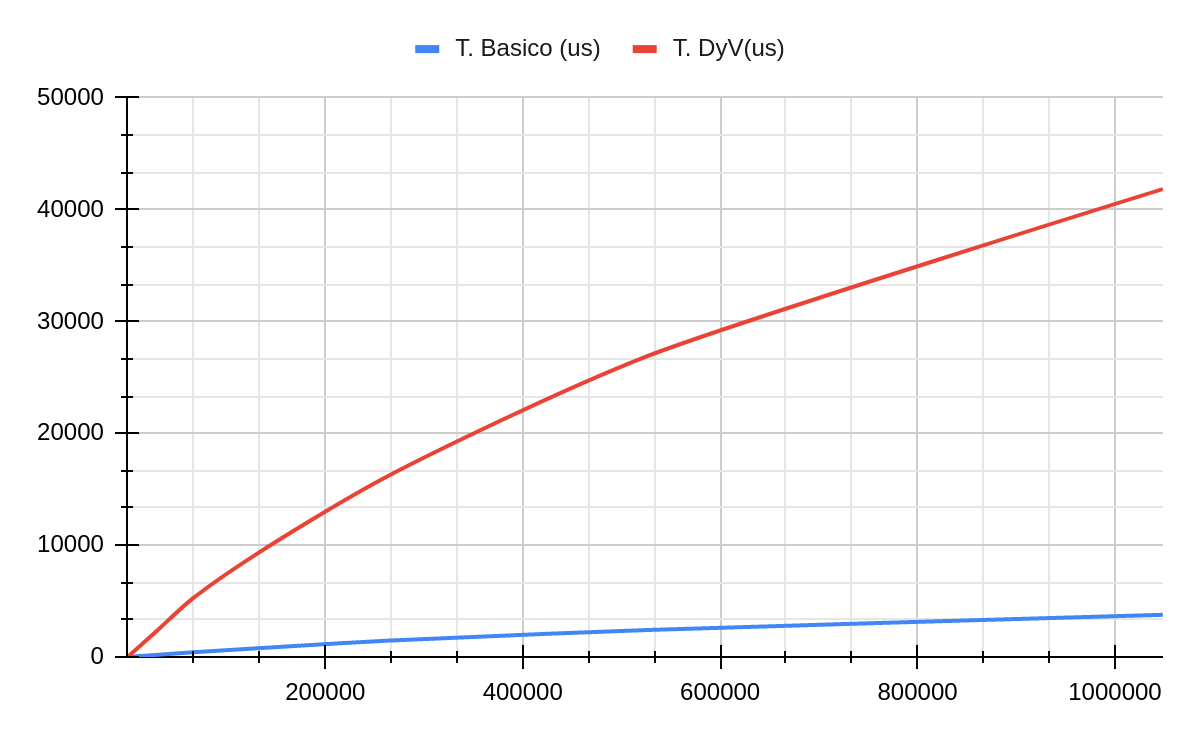


Tabla II: Ejecución hasta tamaño de caso 2^20 (ampliación de la anterior). Se puede observar el comportamiento logarítmico del algoritmo DyV y el comportamiento lineal del básico.



**Ejercicio 2**

**(2.1) Diseño del algoritmo básico.**

Se nos pide diseñar un algoritmo básico/a la fuerza el cual nos permite calcular si un número es un cuadrado perfecto, siendo estos números lo cuales son resultado de elevar un número al cuadrado,es decir,2²=4,4²=16,etc..

El algoritmo diseñado es el siguiente:

CuadradoPerfectoBasico(número)

raiz =0

encontrado = false

si numero==1

resultado = true;

sí no

for i=0 hasta i < numero

si i \* i == numero

resultado = true

return resultado

**(2.2) Análisis y diseño de componentes del algoritmo Divide y Vencerás.**

Adaptando el diseño del algoritmo básico y teniendo en cuenta lo explicado en el apartado anterior, aplicando la plantilla DyV el algoritmo queda de la siguiente forma:

Nuestro algoritmo Divide y Vencerás, se basa en mejorar el cálculo de la raíz cuadrada del número que nos pasan y para ello lo que hacemos es primero comprobar si se trata del número uno y en dicho caso devolver directamente que si es cuadrado perfecto, hacemos esto, ya que después dividimos el número que comprobar entre dos y el 1 es un número que podría darnos problemas.Ahora lo que hacemos es guardar en auxiliar la mitad de nuestro número para que así después gracias a esto saber si la raíz cuadrada de este se encuentra en la mitad superior o inferior de dicha mitad, consiguiendo de esta manera solo tener que recorrer como máximo la mitad de nuestro número y consiguiendo una eficiencia logarítmica.

**(2.3) Diseño del algoritmo Divide y Vencerás y función de combinación.**

CuadradoPerfecto(número)

resultado = false

aux = numero/2

si numero==1

resultado = true

sí no

si aux\*aux>numero

for i=0 hasta i<=numero/2

si i \* i == numero

resultado = true;

si no

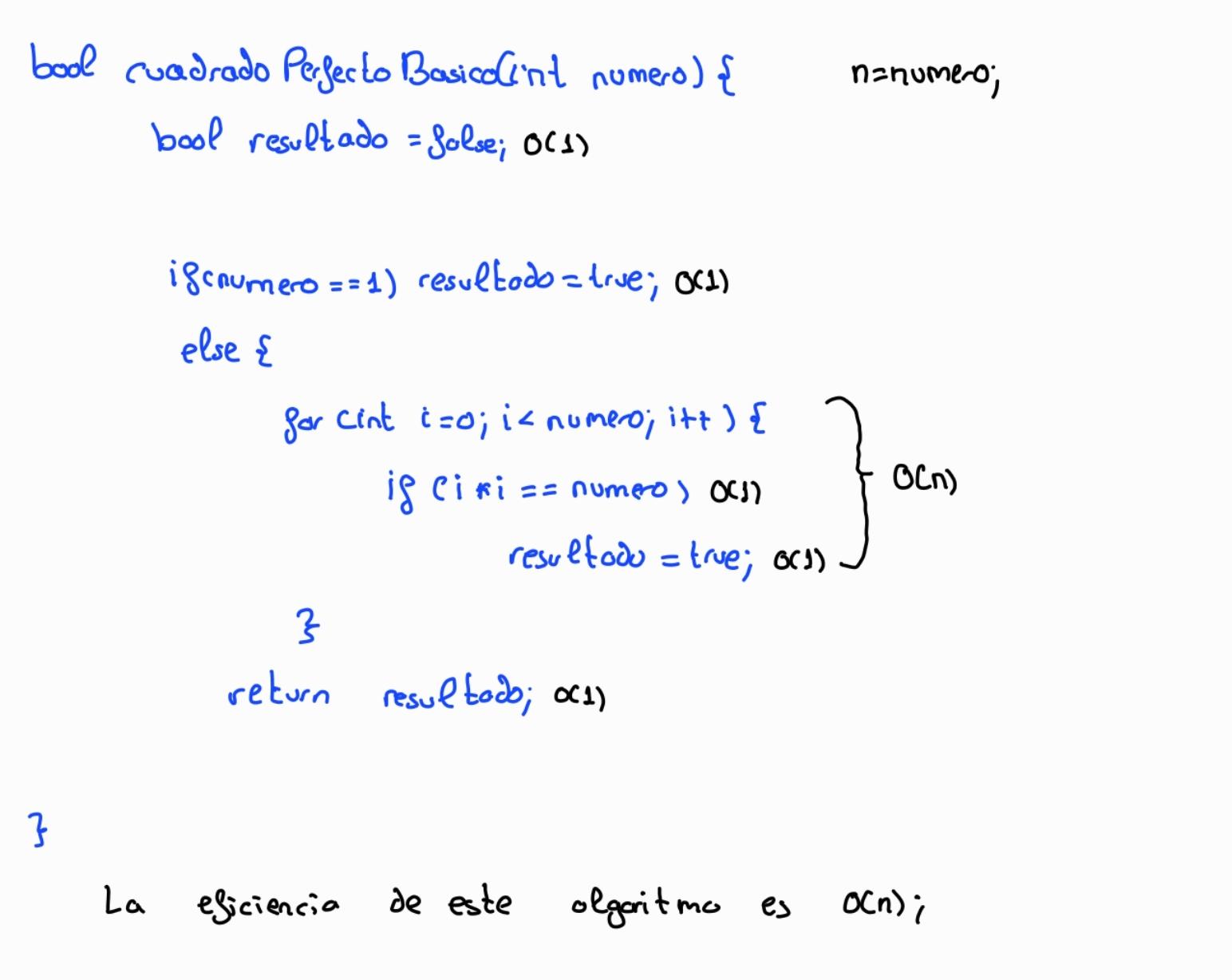
for i=numero/2 hasta i<=numero

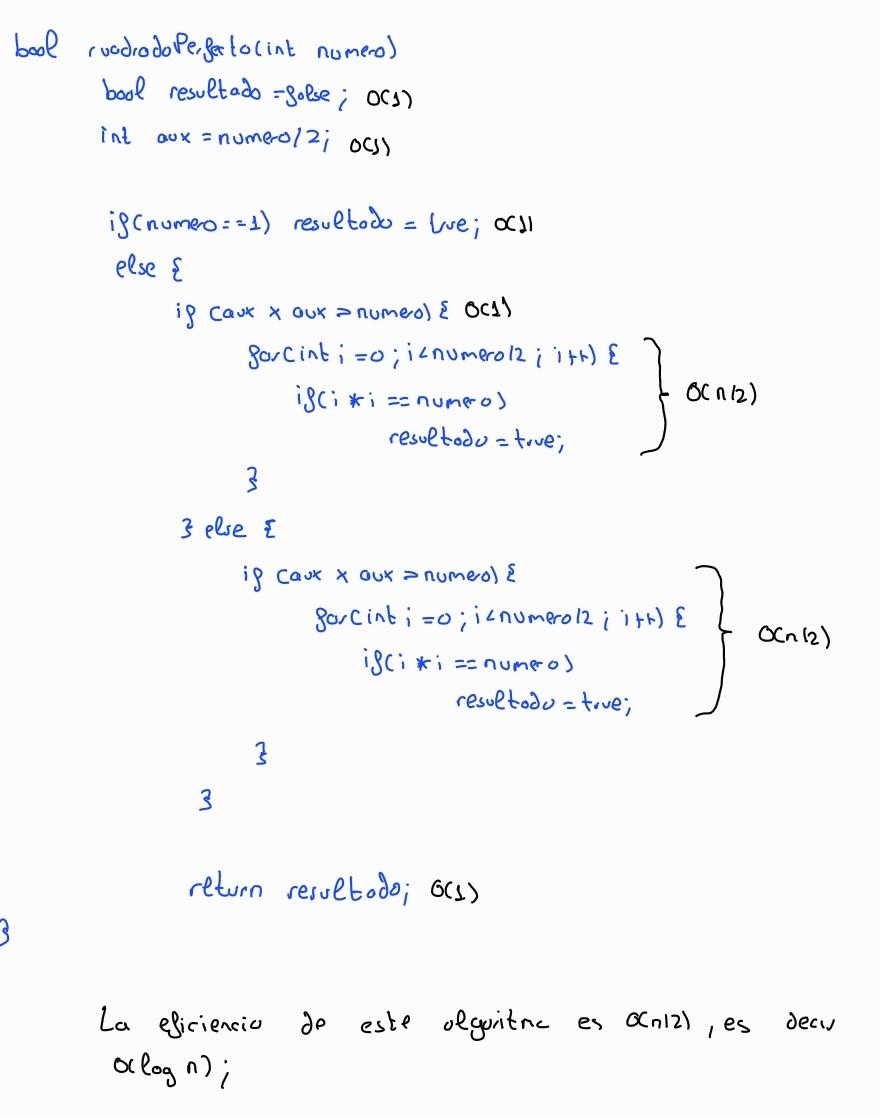
si i \* i == numero

resultado = true;

return resultado

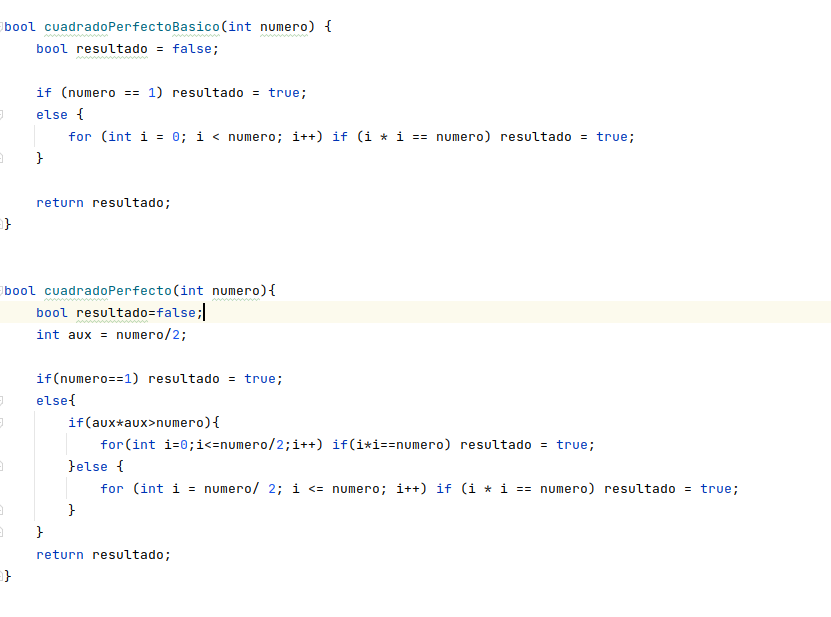
**(2.4) Análisis de eficiencia de los métodos básico y Divide y Vencerás. Justificación de si el diseño Divide y Vencerás realizado mejora al algoritmo básico.**

****

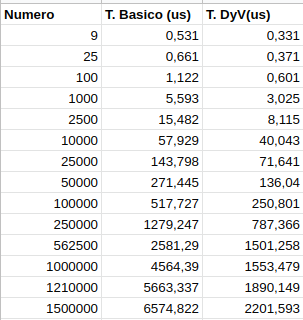
****

**(2.5) Implementación de los métodos y pruebas de ejecución.**

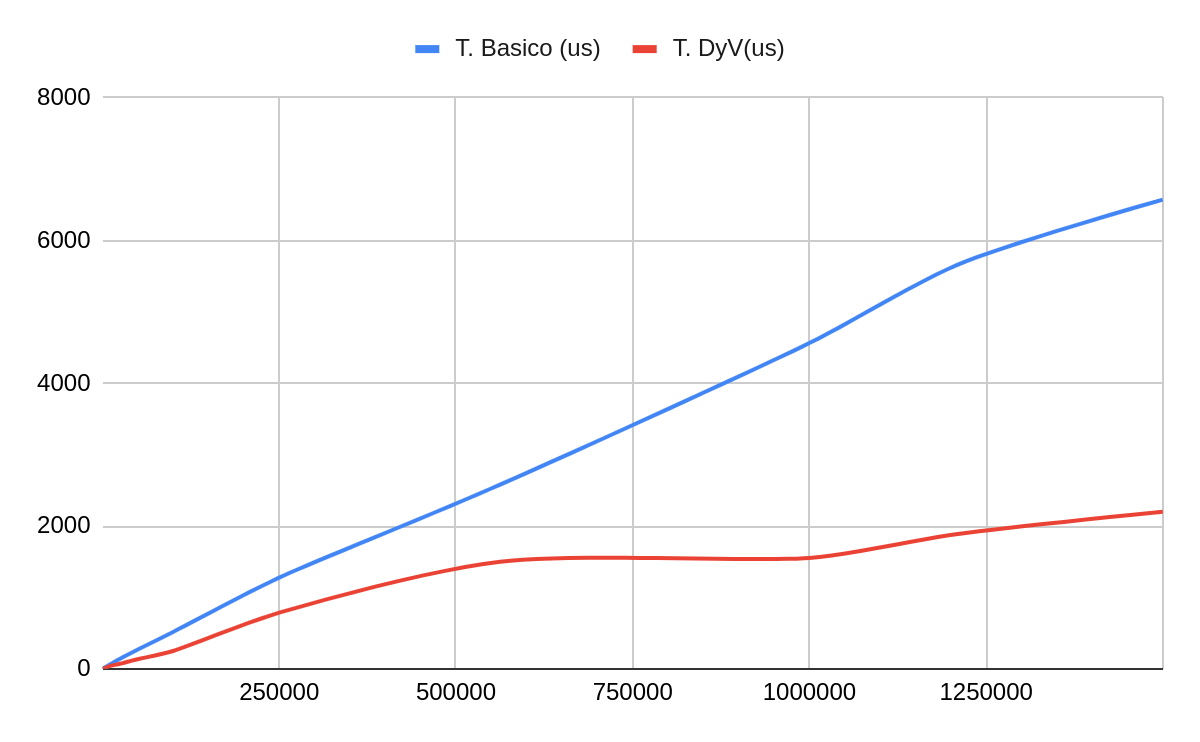
A continuación se muestra como se han implementado ambos algoritmos tanto el básico como el divide y vencerás.

****

Los números usados para poner a prueba los distintos algoritmos, tanto el básico como el DyV, demostrando que este último es más eficiente que el básico tanto de forma teórica como práctica



Se puede observar que la gráfica de el algoritmo básica es de orden lineal, sin embargo la de el algoritmo divide y vencerás es orden log n por tanto hemos conseguido obtener un algoritmo el cual es mucho más eficiente que el algoritmo básico diseñado al principio

****

**Ejercicio 3**

**(3.1) Diseño del algoritmo básico.**

Se nos pide diseñar un algoritmo el cual sea capaz de calcular el valor de ‘y’ dado un valor ‘n’ mediante la siguiente fórmula, n = y\*(y+1)\*(y+2). Mediante este cálculo tenemos que saber si un número N, se puede calcular mediante valores ‘y’ por lo que el algoritmo es el siguiente:

CalculoN(N>0)

encontrado = false

for(int y=0;y<N; !encontrado;y++)

n = y\*(y+1)\*(y+2)

si (N == n)

encontrado = true

return y

return -1 (en el caso de que no haya ningún resultado que coincida)

**(3.2) Análisis y diseño de componentes del algoritmo Divide y Vencerás.**

Aplicando lo que indica el método de resolución Divide y Vencerás aplicado al algoritmo anterior, hacemos una búsqueda binaria aplicando la recursividad entre funciones. En este caso, el algoritmo divide y vencerás será mucho más eficiente con respecto al algoritmo base, ya que su eficiencia teórica es más eficiente por lo que tarda menos en su ejecución.

**(3.3) Diseño del algoritmo Divide y Vencerás y función de combinación.**

CalculoN(N>0)

CalculoParcial(0,N,N)

CalculoParcial(min,max,busca)

centro = (min+max)/2

si (min<max-1)

n = centro\*(centro+1)\*(centro+2)

si (n menor que busca)

min = centro

return CalculoParcial(min,max,busca)

sino

max = centro

return CalculoParcial(min,max,busca)

sino

si (min == max) return min

n=min\*(min+1)\*(min+2)

si (n == busca) return min

n=max\*(max+1)\*(max+2)

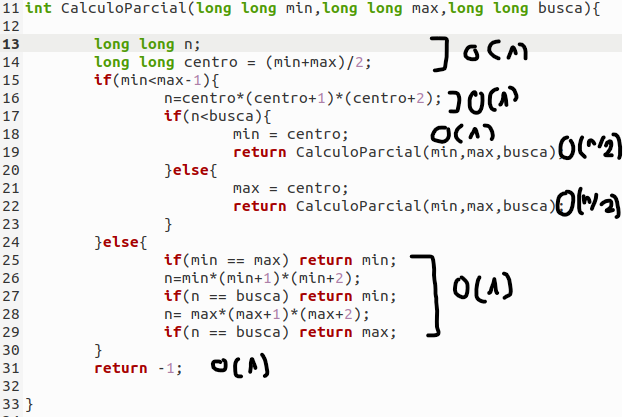
si (n == busca) return max

return -1

(en el caso de que no sea correcto, devolverá -1 el cual es un valor nulo por lo que no cuenta)

**(3.4) Análisis de eficiencia de los métodos básico y Divide y Vencerás. Justificación**

**de si el diseño Divide y Vencerás realizado mejora al algoritmo básico.**

****

*Caso Base ->*  n= 1 ; T(1) = 1

Caso General -> T(n) = T(n/2) ya que solo se podrá ejecutar una parte de la recursividad a la vez, o un caso u otro.

Hacemos un cambio de variable para resolver la recurrencia:

n= 2^k , por lo que k = log2n

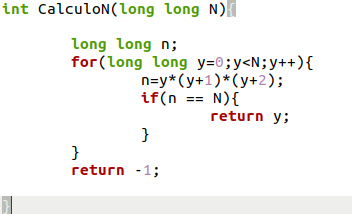
T(2^k) = T(2^k/2) —> Igualamos a 0 —> T(2^k) - T(2^k/2) = 0, una vez tenemos la recurrencia homogénea, la resolvemos.

tk-tk-1 = 0, sacamos la ecuación característica, (x-1) = 0 por lo que la ecuación es: tk =c1\*k

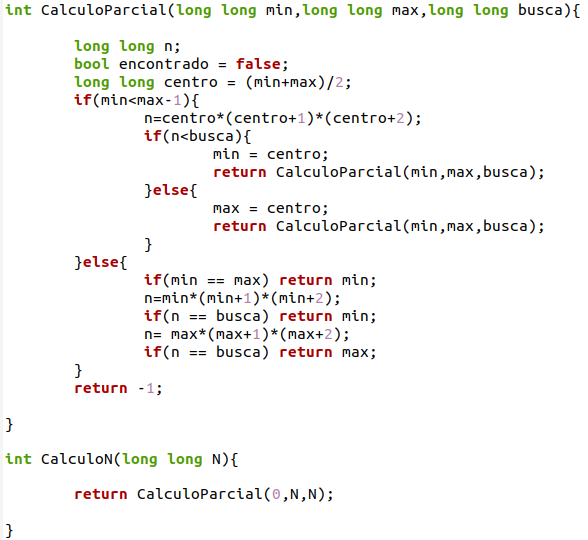
volvemos los valores antes del cambio : **tn = c1\*logn**

**(3.5) Implementación de los métodos y pruebas de ejecución.**

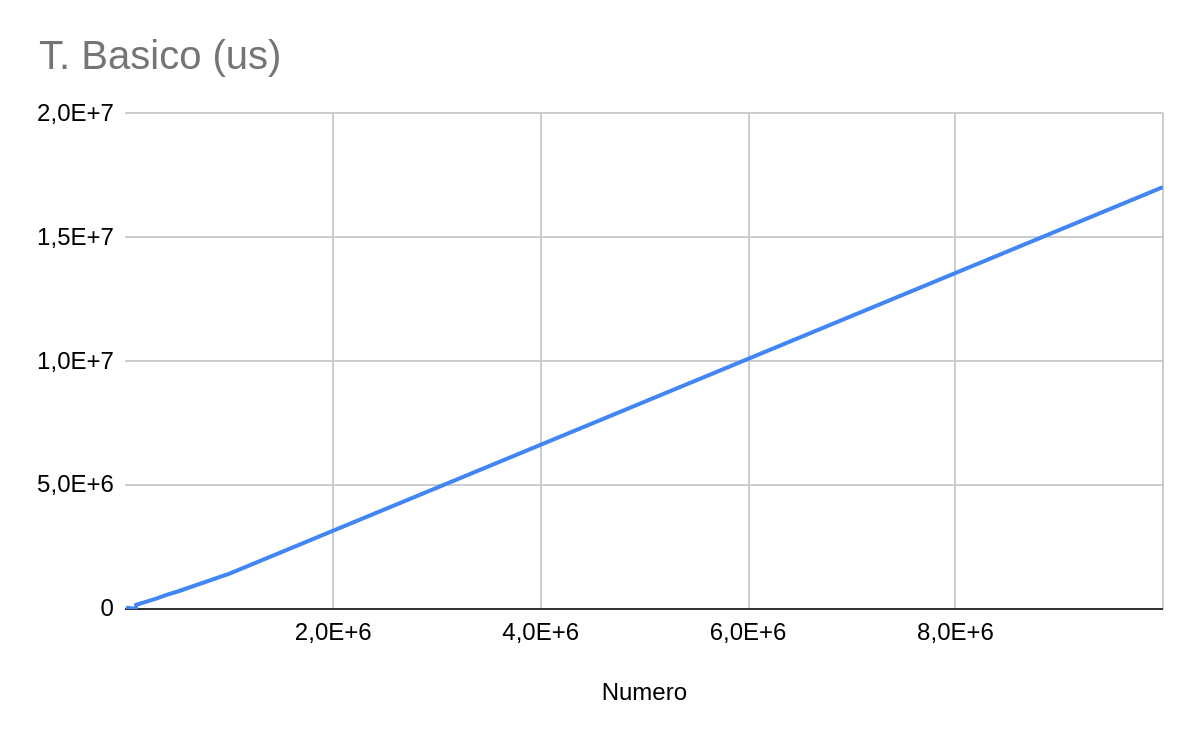
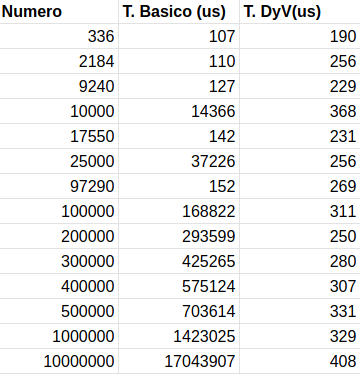
Estos son los siguientes algoritmos que hemos realizado, el algoritmo básico:

****

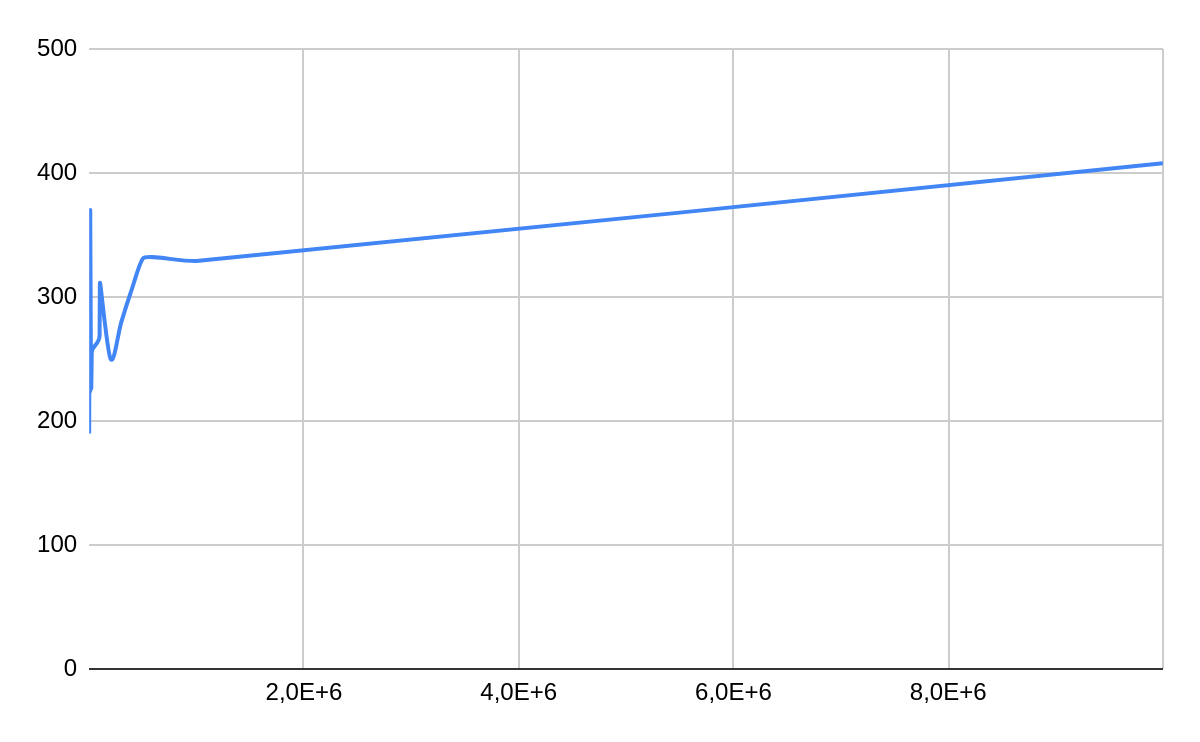
y el algoritmo DyV:

****

**Tabla de ejecuciones con sus gráficos correspondientes.**

****

**Tiempo de ejecución del algoritmo base.**

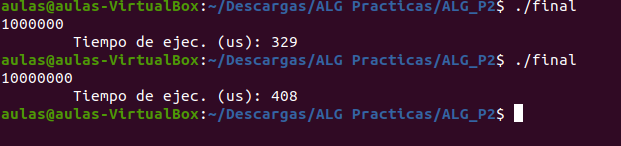
****

**Tiempo de ejecución del algoritmo DyV**

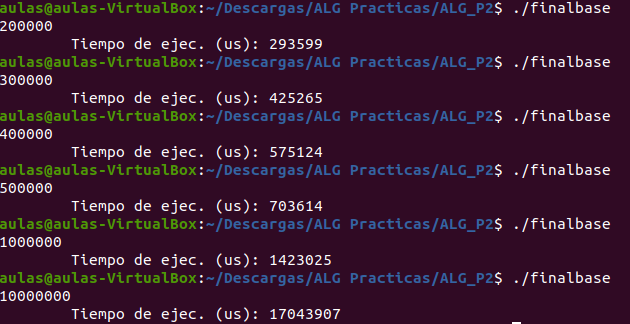
Se puede observar que el tiempo de ejecución del algoritmo base es claramente superior ya que su eficiencia es menor y en casos de números muy grandes produce tiempos de ejecución muy malos. En cambio, en el caso del algoritmo DyV el tiempo de ejecución es mucho menor y no hay mucha diferencia entre los tiempos, esto se debe a que es una función logarítmica, aunque no se aprecia correctamente en el gráfico.

**Pruebas de ejecución del algoritmo:**

***Caso DyV:***

****

**Caso base:**

****