

Electrocardiógrafo

José Antonio Córdoba Gómez
Paula de la Hoz Garrido
Juan Pablo Sáez
Cristina Garrido López
Celia Pedregosa Moreno
Ana Isabel Guerrero Tejera

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

12 de mayo de 2017

Tabla de Contenidos

1 Introducción

2 Algoritmo Bruto

3 Algoritmo Divide y Vencerás

4 Conclusiones

Descripción

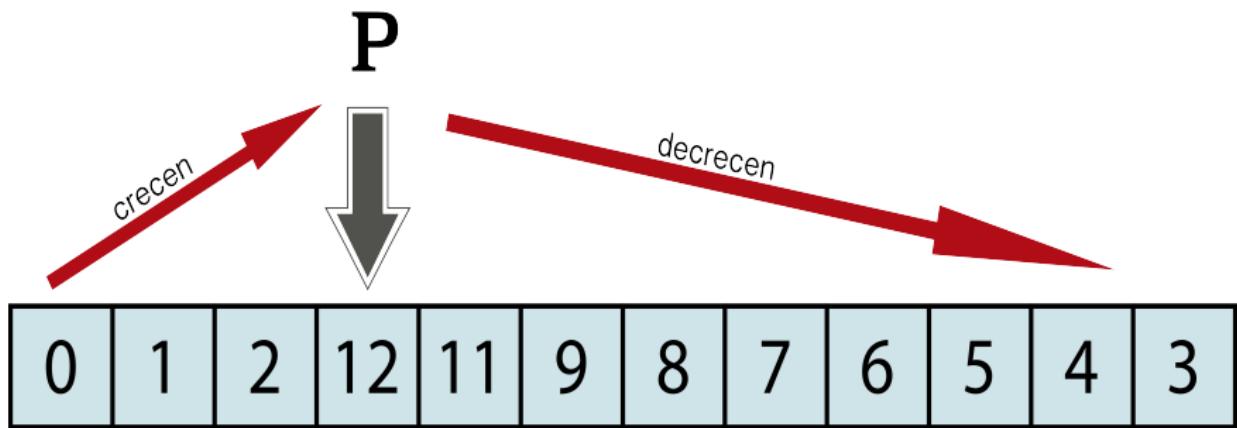
Sea un vector v de números de tamaño n , todos distintos, de forma que existe un índice p (que no es ni el primero ni el último) tal que a la **izquierda de p** los números están ordenados de forma **creciente** y a la **derecha de p** están ordenados de forma **decreciente**; es decir

$$\forall i, j \leq p, i < j \Rightarrow v[i] < v[j] \quad \text{y} \quad \forall i, j \geq p, i < j \Rightarrow v[i] > v[j] \quad (1)$$

Estrategia de resolución

Recorremos la estructura de datos desde el inicio y buscamos el **primer y único** elemento máximo que encontremos.

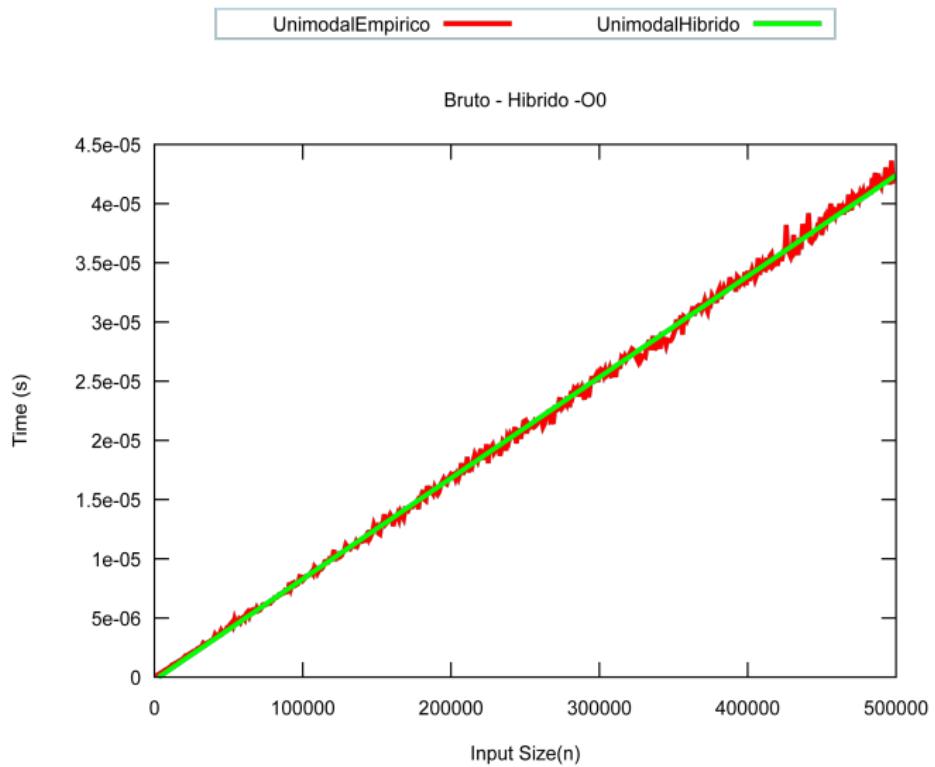
Breve explicación



Algoritmo Bruto - Implementación

```
int unimodalBruto(vector<int> v, int size){  
    bool found = false;  
    int pos = 0;  
  
    for(int i=0; i< size-1 && !found; i++)  
        if(v[i]>v[i+1]) {  
            pos = i;  
            found = true;  
        }  
  
    if(found == false)  
        pos = size-1;  
    return pos;  
}
```

Breve explicación



Descripción

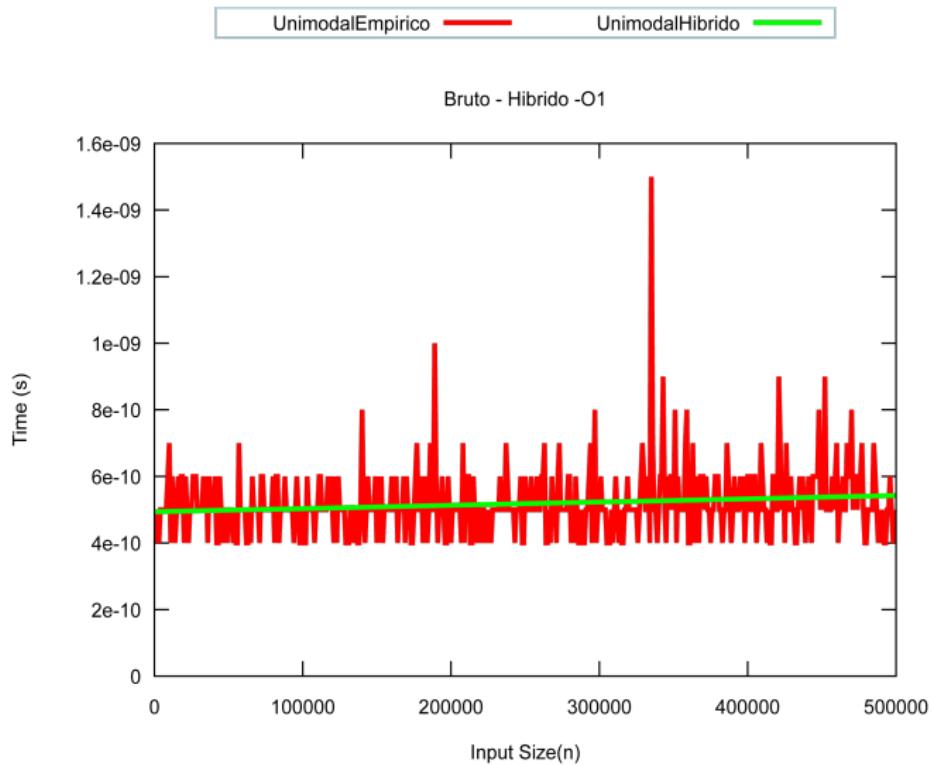
Sabemos que la función que describe la eficiencia de este algoritmo tiene la siguiente forma:

$$T(n) = a \cdot n + b \quad (2)$$

Al realizar el ajuste de los datos con la herramienta *gnuplot* obtenemos el valor de las constantes ocultas, quedando por tanto:

$$T(n) = 8,5281 \cdot 10^{-11} \cdot n - 2,43262 \cdot 10^{-07} \quad (3)$$

Optimización 1

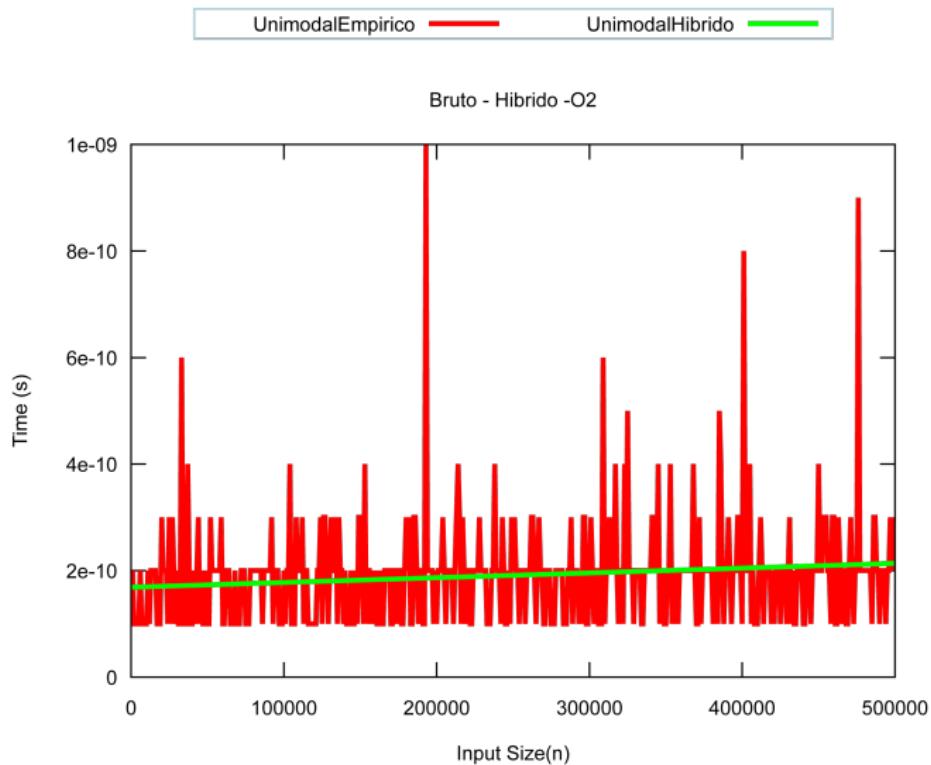


Descripción

En este caso la función queda de la siguiente forma:

$$T(n) = 9,75348 \cdot 10^{-17} \cdot n + 4,93652 \cdot 10^{-07} \quad (4)$$

Optimización 2



Descripción

En este caso la función queda de la siguiente forma:

$$T(n) = 8,96154 \cdot 10^{-17} \cdot n + 1,68779 \cdot 10^{-07} \quad (5)$$

Mejora

Como podemos ver, la mejora se encuentra en las constantes ocultas, concretamente, conseguimos una mejora muy sustancial en la pendiente, que pasamos de $a = 8,5281 \cdot 10^{-11}$ a $a = 9,75348e \cdot 10^{-17}$

Estrategia de resolución

Nos posicionamos sobre el elemento que ocupa la posición media de la estructura:

- ① Si el elemento a su izquierda es mayor y si elemento a la derecha es menor, entonces, nos quedamos con la primera mitad de la estructura.
- ② Si el elemento a su izquierda es menor y su elemento a la derecha es mayor, entonces, nos quedamos con la segunda mitad de la estructura.
- ③ Si el elemento a su izquierda es menor y su elemento a la derecha es menor: ¡Hemos encontrado el elemento que buscábamos!

Breve explicación - Paso 0

0



¡Voy a ser tu guía en este ejemplo!
Este es el caso inicial. ¿Continuamos?

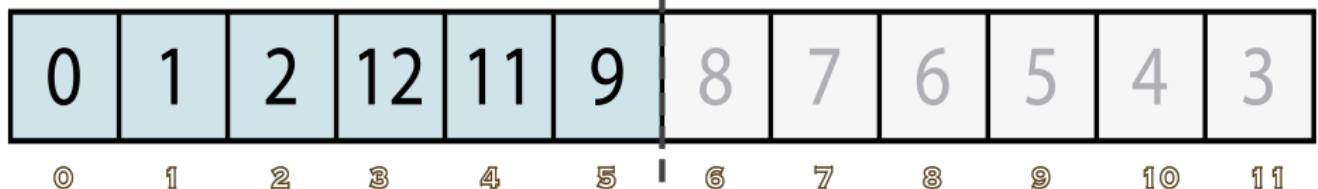
0	1	2	12	11	9	8	7	6	5	4	3
0	1	2	3	4	5	6	7	8	9	10	11

Breve explicación - Paso 1

1

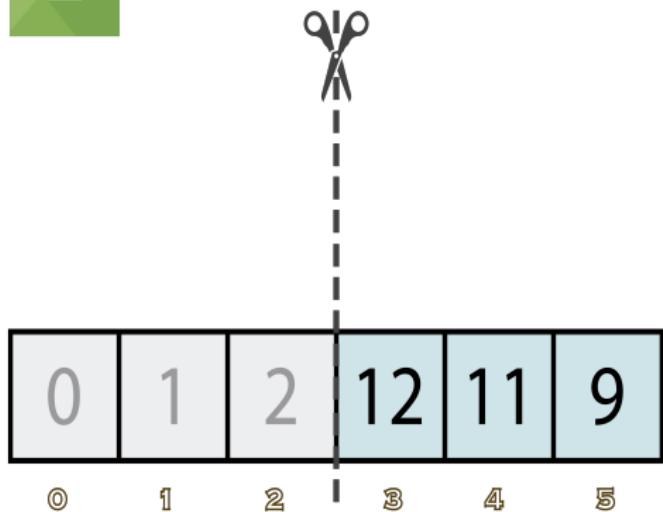


No nos encontramos con un caso extremo así que nos situamos en la posición media de la estructura. Como se cumple la condición 1 \rightarrow Recorto la estructura por la derecha



Breve explicación - Paso 2

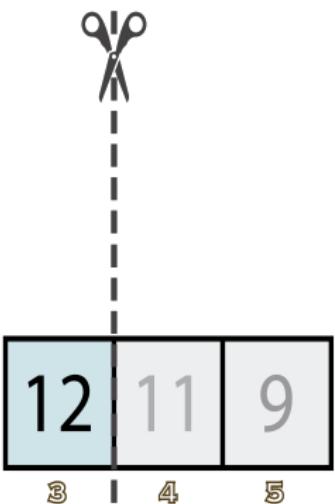
2



Volvemos a situarnos en la parte media de la estructura, y ahora nos encontramos que se cumple el segundo caso, así que recortamos la parte izquierda de la estructura

Breve explicación - Paso 3

3



Nos volvemos a situar otra vez en la posición media y comprobamos que se cumple la primera condición, así que cortamos la parte derecha de la estructura.

Breve explicación - Paso 4

4

P



12

3

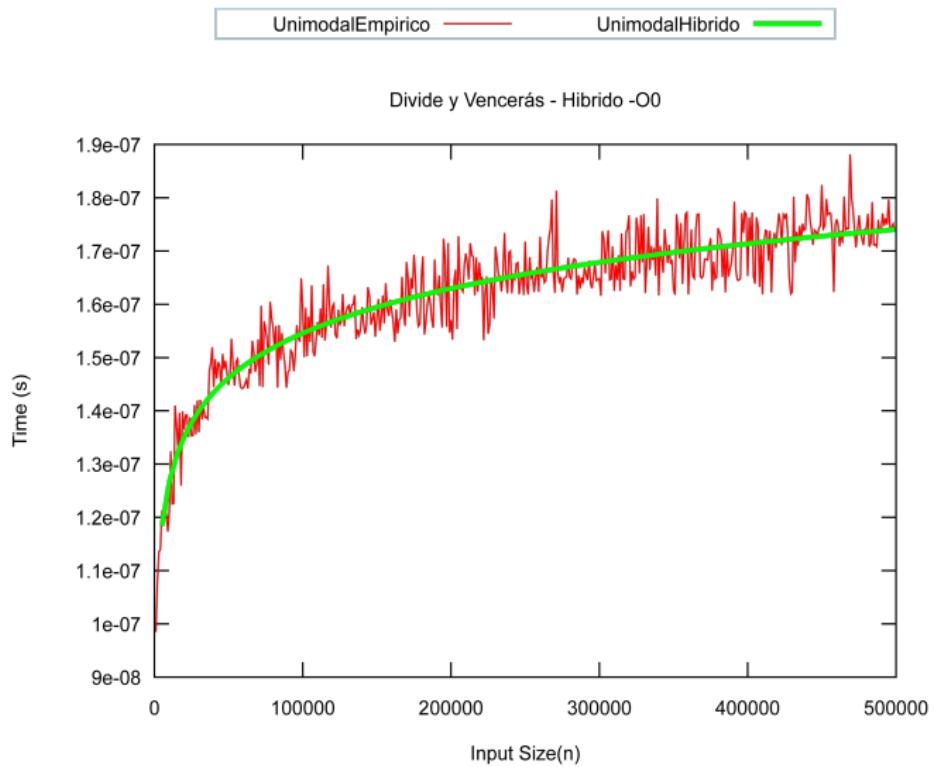


¡Hurra! Hemos encontrado el elemento en
un pis pas. Avísame cuando tenga que volver
a pegar unos cuantos recortes ;)

Algoritmo Bruto - Implementación

```
int unimodalDyV(vector<int> &v, int inicio , int final){
    int posicionMedia = (inicio + final) / 2;
    bool found = false;
    if(posicionMedia > inicio){
        if(v[posicionMedia -1] > v[posicionMedia]
        && v[posicionMedia] > v[posicionMedia+1])
            return unimodalDyV(v, inicio , posicionMedia );
        else if(v[posicionMedia -1] < v[posicionMedia]
        && v[posicionMedia] < v[posicionMedia+1])
            return unimodalDyV(v, posicionMedia , final );
        else if(v[posicionMedia] > v[posicionMedia -1]
        && v[posicionMedia+1]<v[posicionMedia ])
            return posicionMedia ;
    }else{
        if(v[inicio] <= v[final])
            return final;
        else
            return inicio ;
    }
}
```

Análisis Híbrido



Descripción

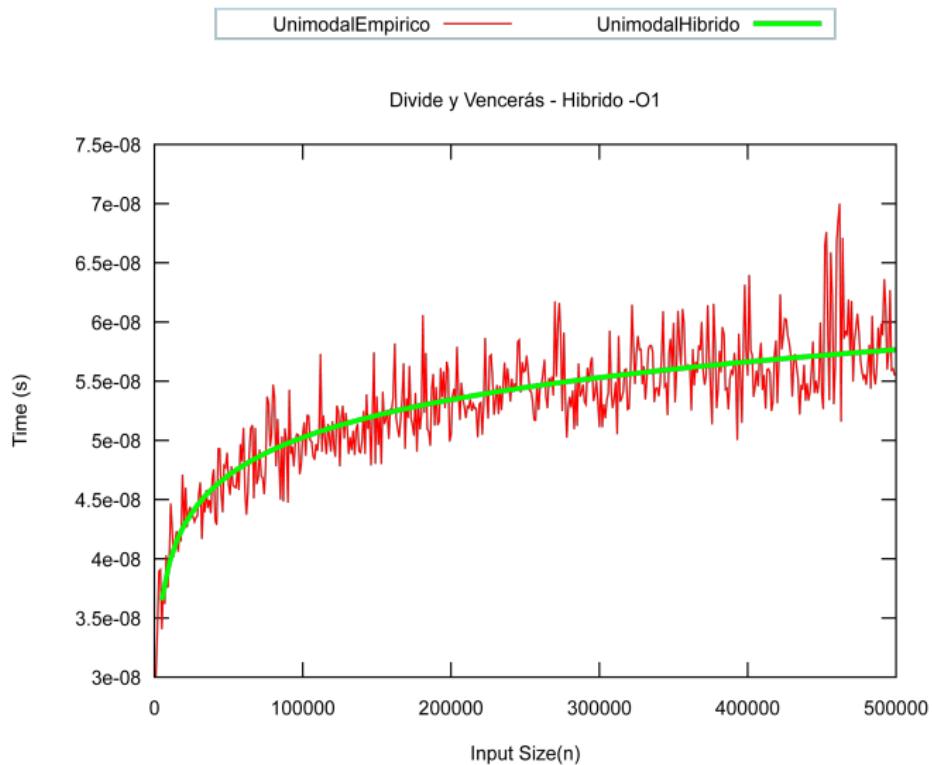
Sabemos que la función que describe la eficiencia de este algoritmo tiene la siguiente forma:

$$T(n) = a \cdot \log(n) + b \quad (6)$$

Al realizar el ajuste de los datos con la herramienta *gnuplot* obtenemos el valor de las constantes ocultas, quedando por tanto:

$$T(n) = 1,21165 \cdot 10^{-08} \cdot \log(n) + 1,50656 \cdot 10^{-08} \quad (7)$$

Optimización 1

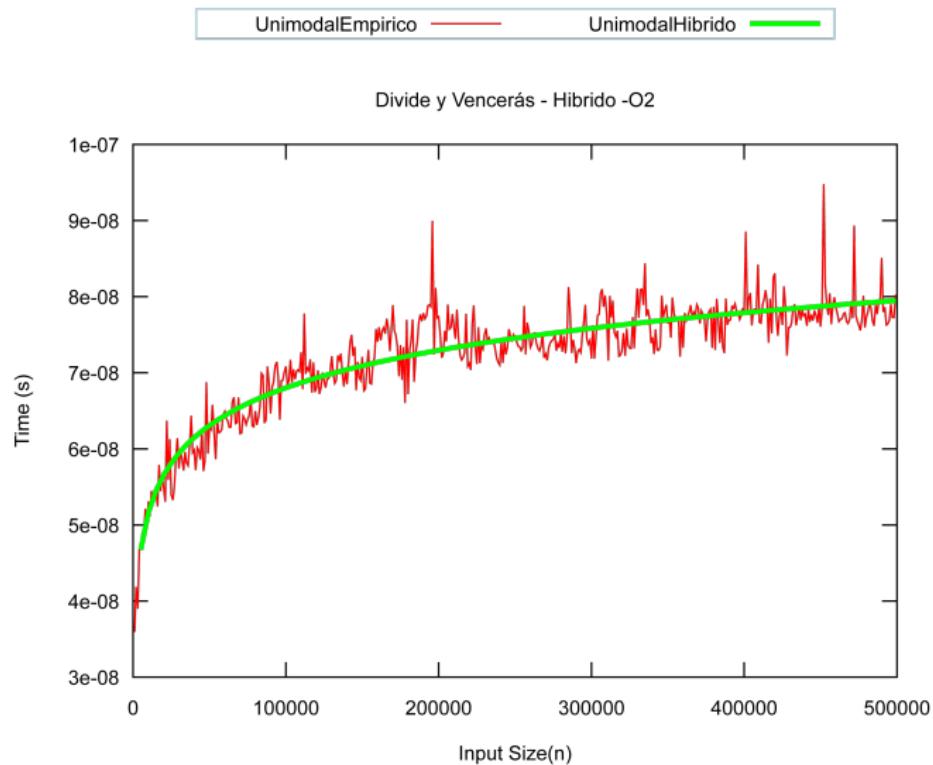


Descripción

En este caso la función queda de la siguiente forma:

$$T(n) = 4,61006 \cdot 10^{-09} \cdot \log(n) - 2,83008 \cdot 10^{-09} \quad (8)$$

Optimización 2



Descripción

En este caso la función queda de la siguiente forma:

$$T(n) = 7,12567 \cdot 10^{-09} \cdot \log(n) - 1,4022 \cdot 10^{-08} \quad (9)$$

Mejora

De nuevo, volvemos a ver que la mejoría se encuentra, como no podía ser de otra forma, en el valor de las constantes ocultas. También podemos observar que la mejoría es menos notoria si la comparamos con la mejora obtenida después de optimizar el algoritmo bruto hacia la optimización 1.

Mejora

Hemos podido comprobar de primera mano que un cambio en la optimización o en las prestaciones de la computadora donde se ejecute nuestro algoritmo hace variar las constantes multiplicativas.

Estos cambios pueden ser sustanciales, pero un cambio de algoritmo puede variar la el orden de eficiencia, lo que nos lleva a un cambio mucho más drástico que el que nos proporcionan las constantes ocultas.

Conclusión: mejorar el algoritmo es mejor que mejorar las prestaciones de nuestro ordenador.

Tabla Optimizacion 0

Comparativa por intervalos	00 Bruto	00 DyV
[2 - 50.000)	2.10151e-06	1.36033e-07
[50.000 - 100.000)	6.18549e-06	1.50331e-07
[100.000 - 150.000)	1.03512e-05	1.57129e-07
[150.000 - 200.000)	1.45749e-05	1.60535e-07
[200.000 - 250.000)	1.89019e-05	1.64477e-07
[250.000 - 300.000)	2.29669e-05	1.66396e-07
[300.000 - 350.000)	2.71551e-05	1.68927e-07
[350.000 - 400.000)	3.17235e-05	1.69562e-07
[400.000 - 450.000)	3.58912e-05	1.72166e-07
[450.000 - 500.000)	4.05783e-05	1.74867e-07



Observa que el crecimiento del algoritmo bruto es mucho mayor en cada salto de intervalo que el del algoritmo divide y vencerás

Tabla Optimización 1

Comparativa por intervalos	O1 ↓ Bruto	O1 ↓ DyV
[2 - 50.000)	5.12500e-10	4.31580e-08
[50.000 - 100.000)	4.96000e-10	4.85582e-08
[100.000 - 150.000)	4.98000e-10	5.06538e-08
[150.000 - 200.000)	5.18000e-10	5.28082e-08
[200.000 - 250.000)	4.96000e-10	5.40650e-08
[250.000 - 300.000)	5.16000e-10	5.47370e-08
[300.000 - 350.000)	5.36000e-10	5.53255e-08
[350.000 - 400.000)	5.32000e-10	5.65562e-08
[400.000 - 450.000)	5.40000e-10	5.61571e-08
[450.000 - 500.000)	5.38000e-10	5.88707e-08



Con la optimización podemos observar que el algoritmo bruto se comporta 100 veces mejor que el algoritmo divide y vencerás.

Tabla Optimización 2

Comparativa por intervalos	O2 Bruto	O2 DyV
[2 - 50.000)	1.68750e-10	5.58776e-08
[50.000 - 100.000)	1.70000e-10	6.51426e-08
[100.000 - 150.000)	1.80000e-10	7.02357e-08
[150.000 - 200.000)	1.92000e-10	7.43789e-08
[200.000 - 250.000)	1.94000e-10	7.41088e-08
[250.000 - 300.000)	1.82000e-10	7.42217e-08
[300.000 - 350.000)	2.12000e-10	7.63556e-08
[350.000 - 400.000)	2.04000e-10	7.70189e-08
[400.000 - 450.000)	1.88000e-10	7.81729e-08
[450.000 - 500.000)	2.20000e-10	7.85669e-08



Con esta optimización, se han estabilizado más los tiempos. Vemos que el procedimiento recursivo no se puede mejorar más.

Tabla resumen

Comparativa por intervalos	00	00	01	01	02	02
	Bruto	DyV	Bruto	DyV	Bruto	DyV
[2 - 50.000)	2.10151e-06	1.36033e-07	5.12500e-10	4.31580e-08	1.68750e-10	5.58776e-08
[50.000 - 100.000)	6.18549e-06	1.50331e-07	4.96000e-10	4.85582e-08	1.70000e-10	6.51426e-08
[100.000 - 150.000)	1.03512e-05	1.57129e-07	4.98000e-10	5.06538e-08	1.80000e-10	7.02357e-08
[150.000 - 200.000)	1.45749e-05	1.60535e-07	5.18000e-10	5.28082e-08	1.92000e-10	7.43789e-08
[200.000 - 250.000)	1.89019e-05	1.64477e-07	4.96000e-10	5.40650e-08	1.94000e-10	7.41088e-08
[250.000 - 300.000)	2.29669e-05	1.66396e-07	5.16000e-10	5.47370e-08	1.82000e-10	7.42217e-08
[300.000 - 350.000)	2.71551e-05	1.68927e-07	5.36000e-10	5.53255e-08	2.12000e-10	7.63556e-08
[350.000 - 400.000)	3.17235e-05	1.69562e-07	5.32000e-10	5.65562e-08	2.04000e-10	7.70189e-08
[400.000 - 450.000)	3.58912e-05	1.72166e-07	5.40000e-10	5.61571e-08	1.88000e-10	7.81729e-08
[450.000 - 500.000)	4.05783e-05	1.74867e-07	5.38000e-10	5.88707e-08	2.20000e-10	7.85669e-08

Agradecimientos



Muchas gracias por atenderme. Espero que hayas comprendido todo lo que te he dicho. Recuerda que puedes pedirme una tutoría, avisando a: alansito@ugr.es