|  |
| --- |
| Universidad de Granada |
| Serie unimodal de números |
| Algorítmica |
| **Aarón Bueno Rodríguez**  **Bryan Moreno Picamán**  **Miguel Ángel Rodríguez Serrano** |
|  |
|  |

|  |
| --- |
| 2º Grado Ingeniería Informática |

**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 ∀i, j ≤**

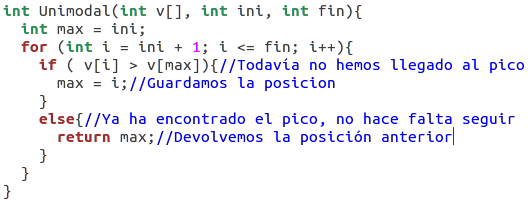
**p, i < j ⇒ v[i] < v[j] y ∀i, j ≥ p, i < j ⇒ v[i] > v[j] (de forma que el máximo se encuentra en la posición p). Diseñe un algoritmo “divide y vencerás” que permita determinar p.**

# Algoritmo sencillo

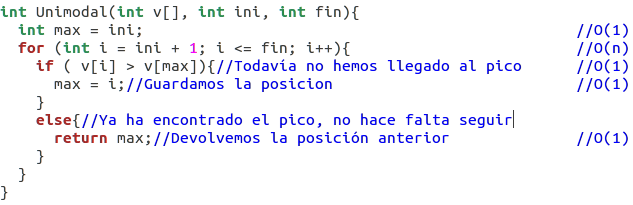
El algoritmo que a cualquier persona se le ocurriría para resolver este problema hubiera funcionado de la siguiente manera:

* Avanzamos desde la primera posición del vector de una en una hasta que encontremos que el valor de la posición actual es menor que el de la anterior.
* Devolvemos la posición anterior, que es el índice.

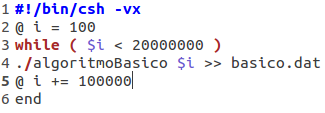
La implementación del algoritmo resultante es la siguiente:



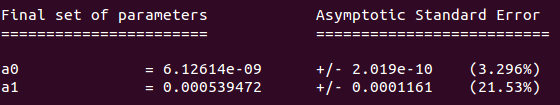
Vamos a analizar su eficiencia:

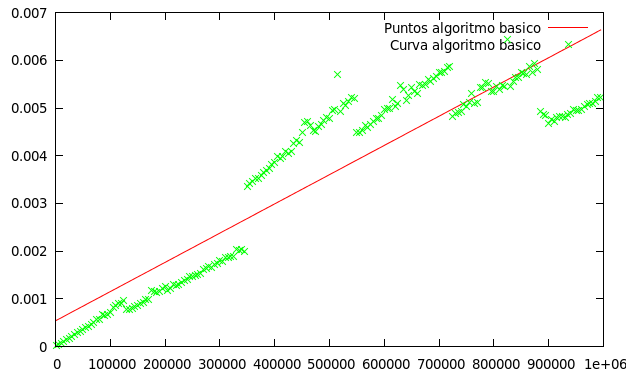


Como se puede ver, el código es del orden de eficiencia teórico O(n). Una vez calculada su eficiencia teórica, vamos a calcular su eficiencia empírica valiéndonos para ello del siguiente script:



Hemos usado la librería *ctime* para calcular el tiempo. Ahora calculamos su eficiencia híbrida con *gnuplot*. Hemos utilizado la función f(x) = a0\*x + a1. Las constantes que han salido son:

****

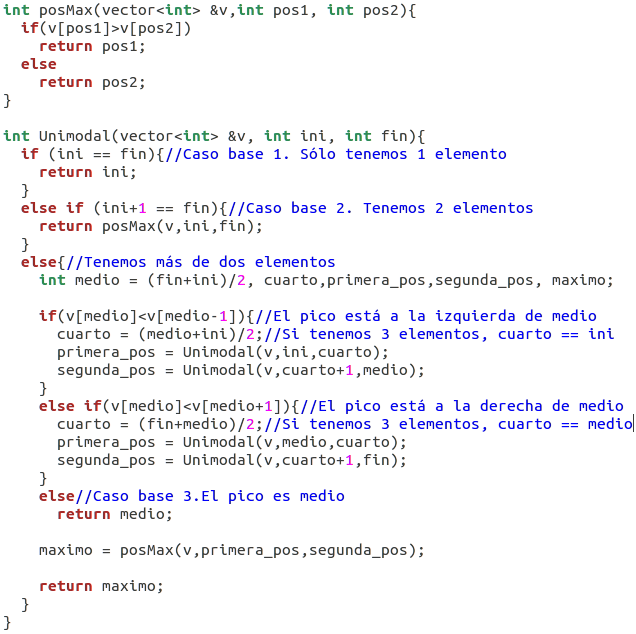
La gráfica resultante tanto de la eficiencia empírica como híbrida ha sido la siguiente:

Los puntos pueden estar dispersos, dependiendo de cuán lejos se encuentre el índice del inicio del vector. Vamos a intentar mejorar este algoritmo creando un algoritmo de tipo Divide y Vencerás.

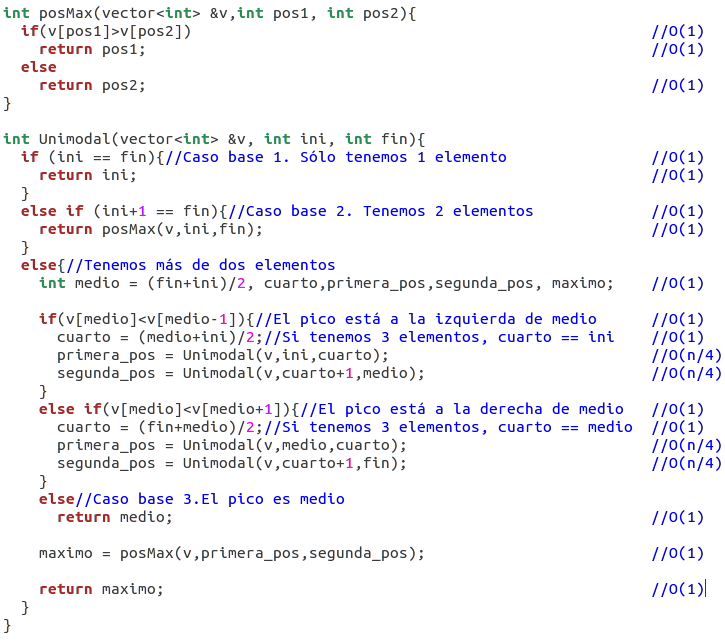
# Algoritmo Divide y Vencerás

Nuestra idea ha sido la siguiente:

* Nos posicionamos en la mitad del vector.
* Comparamos los valores de las posiciones a ambos lados de la mitad:
  + Si el valor en la posición medio es menor que el que hay en medio-1, el índice se encuentra a la izquierda de medio, así que dividimos en dos el vector entre la posición inicial y medio, y devolvemos la posición del valor máximo que haya entre esas dos partes.
  + Si no, si el valor en la posición medio es menor al de medio+1, se hace lo mismo pero esta vez con la mitad entre medio y la posición final dada.
  + Si no es ninguno de los dos casos anteriores, significa que en realidad el punto medio es el que tiene mayor valor, por lo tanto es el índice y lo devuelve.

La implementación del algoritmo explicado queda de la siguiente forma:

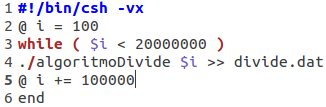
Procedemos a analizar su eficiencia teórica:



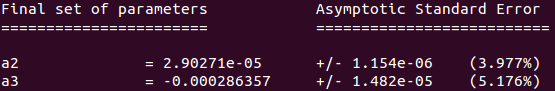
La fórmula maestra resultante se puede apreciar que es:

La resolvemos y queda que es del tipo de eficiencia O(log (n)), por lo que este algoritmo en teoría es bastante más eficiente que el anterior.

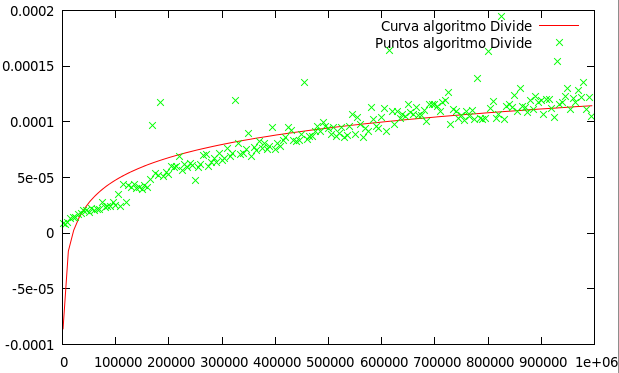
Procedemos a calcular su eficiencia empírica e híbrida. Para la empírica hemos usado el siguiente script:



Metemos los datos calculados en gnuplot. Utilizamos la función g(x)=a2\*log(x) + a3, y calculamos a partir de ellos sus constantes ocultas:

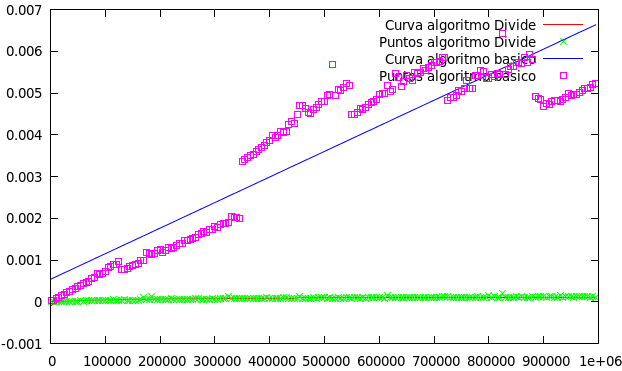


El resultado de la eficiencia híbrida y empírica se encuentra en la siguiente gŕafica:



Recordemos que es normal que los puntos estén dispersos, ya que eso depende de dónde esté situado el pico.

Por último, vamos a comprobar las eficiencias de ambos algoritmos, a ver si es verdad que se cumple que el algoritmo divide y vencerás es más eficiente que el obvio, valiéndonos de una gráfica con los dos algoritmos:



Queda demostrado que para este problema en particular es infinitamente más eficiente un algoritmo Divide y Vencerás que el haber creado un algoritmo normal, como bien puede apreciarse.