Universidad de Granada

Algorítmica

Algoritmos Divide y Vencerás

Laura Calle Caraballo
Cristina María Garrido López
Germán González Almagro
Javier León Palomares
Antonio Manuel Milán Jiménez

5 de abril de 2016

Índice

1. Introducción.

El objetivo de esta práctica es estudiar los algoritmos *Divide y Vencerás*. Estos algoritmos se caracterizan por dividir el problema original en un cierto número de subproblemas que serán resueltos de forma independiente para luego combinar sus soluciones.

En este documento presentaremos un caso concreto donde hemos empleado tanto una aproximación directa como esta técnica, y compararemos su rendimiento para comprobar que, efectivamente, *Divide y Vencerás* es útil.

2. Descripción del problema.

Una serie unimodal consiste en una secuencia de números que es ascendente hasta un índice p, a partir del cual pasa a ser descendente. Nuestra tarea es encontrar ese índice p.

3. Herramientas y metodología empleadas.

3.1. Compilación.

Hemos utilizado el compilador *g++*, de la forma:

```
g++ -OX samplecode.cpp -o sampleprogram -std=c++11
```

Donde X (nivel de optimización) tomará los valores 0, 1, 2 ó 3.

3.2. Medición de tiempos.

Debido a la simplicidad del problema, hemos necesitado la precisión de la biblioteca *chrono* para medir los tiempos de ejecución. Aun así, dichos tiempos dependen en gran medida de la posición en la que se encuentre el pico de la serie unimodal; por ello, para cada tamaño de entrada hemos generado 200 series unimodales diferentes, sobre las que hemos ejecutado el algoritmo, medido su tiempo de ejecución y hecho la media aritmética.

Con respecto a la influencia de la posición del pico de la serie unimodal sobre los tiempos de ejecución:

- Para el algoritmo *Divide y Vencerás*, la posición más ventajosa es la posición central, mientras que cuanto más cerca esté el pico del inicio o el final de la serie, más tiempo tardará en encontrar la solución.
- Para el algoritmo de fuerza bruta, las posiciones más ventajosas son las más próximas al inicio de la serie, al realizar un recorrido lineal.

Como podemos comprobar, una única ejecución para cada tamaño de entrada produce unos tiempos muy dispares, por lo que es necesario utilizar una media aritmética.

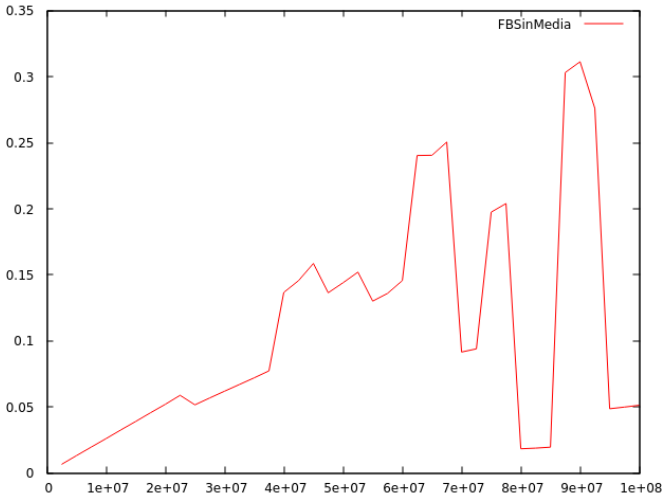

Figura 1: Gráfica del algoritmo de fuerza bruta con datos de una única ejecución.

Sin embargo, la gráfica es bastante más uniforme al hacer la media de 200 ejecuciones.

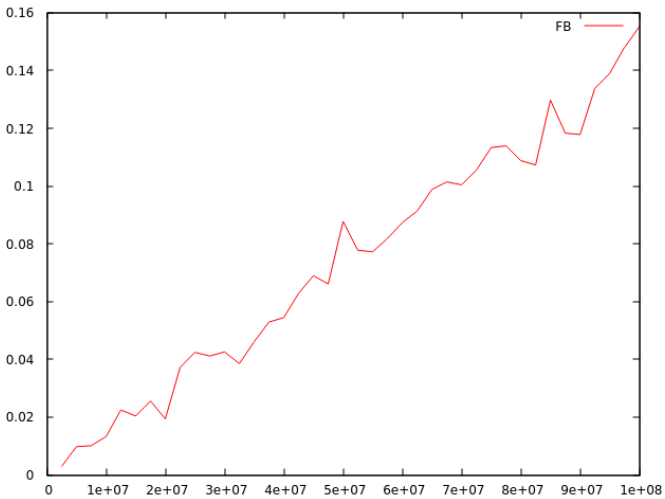

Figura 2: Gráfica del algoritmo de fuerza bruta con la media de los tiempos de ejecución.

4. Aproximación directa.

4.1. Pseudocódigo.

```
function BusquedaFB(elementos, inicio, fin);
begin
    solucion ⟵ inicio;
    if tamaño del vector = 2 and primero < segundo then
        return segundo;
    else if tamaño del vector > 1 then
        for x ∈ elementos do
            if x > siguiente then
                return x;
            end
        end
    end
    return solucion;
end
```

4.2. Código fuente utilizado.

```
int BusquedaFB(int * & v, int ini, int fin){

   int maximo = ini;

   if((fin - ini) == 2 && v[ini] < v[ini+1]){

      return ini+1;
   }

   else if((fin - ini) > 1){

      for (int i=ini+1; i<fin; i++)

         if(v[i-1] > v[i])

            return i-1;

   }

   return maximo;
}
```

4.3. Análisis híbrido.

Como podemos ver en la gráfica, los datos obtenidos se ajustan relativamente bien a la función lineal $f(x) = ax$.

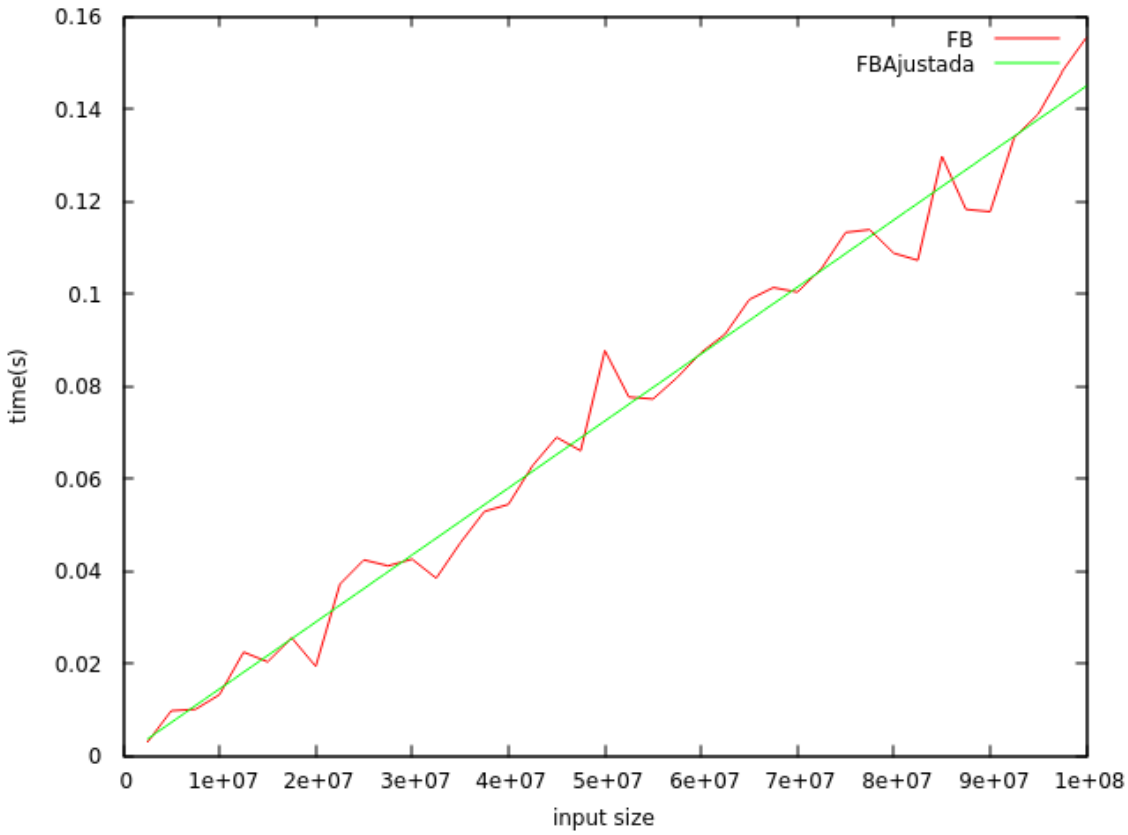

Figura 3: Gráfica del algoritmo de fuerza bruta. Compilación sin optimización. Intel® Core™ i7-5500U CPU @ 2.40GHz.

El análisis híbrido de los datos representados en la gráfica produce el siguiente resultado:

- $a = 1{,}45022 \cdot 10^{-9}$

$f(x) = 1{,}45022 \cdot 10^{-9} \cdot x$

El error en el ajuste es de un 1,051 %.

5. Algoritmo *Divide y Vencerás.*

5.1. Pseudocódigo.

function BusquedaDyV(elementos, inicio, fin);
begin
  if *tamaño del vector* $\leq$ *umbral* **then**
    solucion $\longleftarrow$ BusquedaFB(elementos, inicio, fin);
  else
    solucion $\longleftarrow$ elemento central;
    if *solucion en secuencia ascendente* **then**
      solucion $\longleftarrow$ BusquedaDyV(elementos, solucion+1, fin);
    else if *solucion en secuencia descendente* **then**
      solucion $\longleftarrow$ BusquedaDyV(elementos, inicio, solucion);
    end
  end
  return solucion;
end

5.2. Código fuente utilizado.

```
int busqueda(int * & v, const int ini, const int fin){

   int solucion;

   if((fin - ini) <= umbral)

      solucion = BusquedaFB(v,ini,fin);

   else{

      solucion = (fin + ini)/2;

      if(v[solucion-1] < v[solucion] && v[solucion] < v[solucion+1])

         solucion = busqueda(v,solucion+1,fin);

      else if(v[solucion-1] > v[solucion] && v[solucion] > v[solucion+1])

         solucion = busqueda(v,ini,solucion);
   }

   return solucion;
}
```

5.3. Análisis híbrido.

En la siguiente gráfica vemos que la curva teórica $f(x) = a\log(x)$ no se corresponde con el comportamiento real del algoritmo a pesar de un error en el ajuste de sólo un 2,01 %.

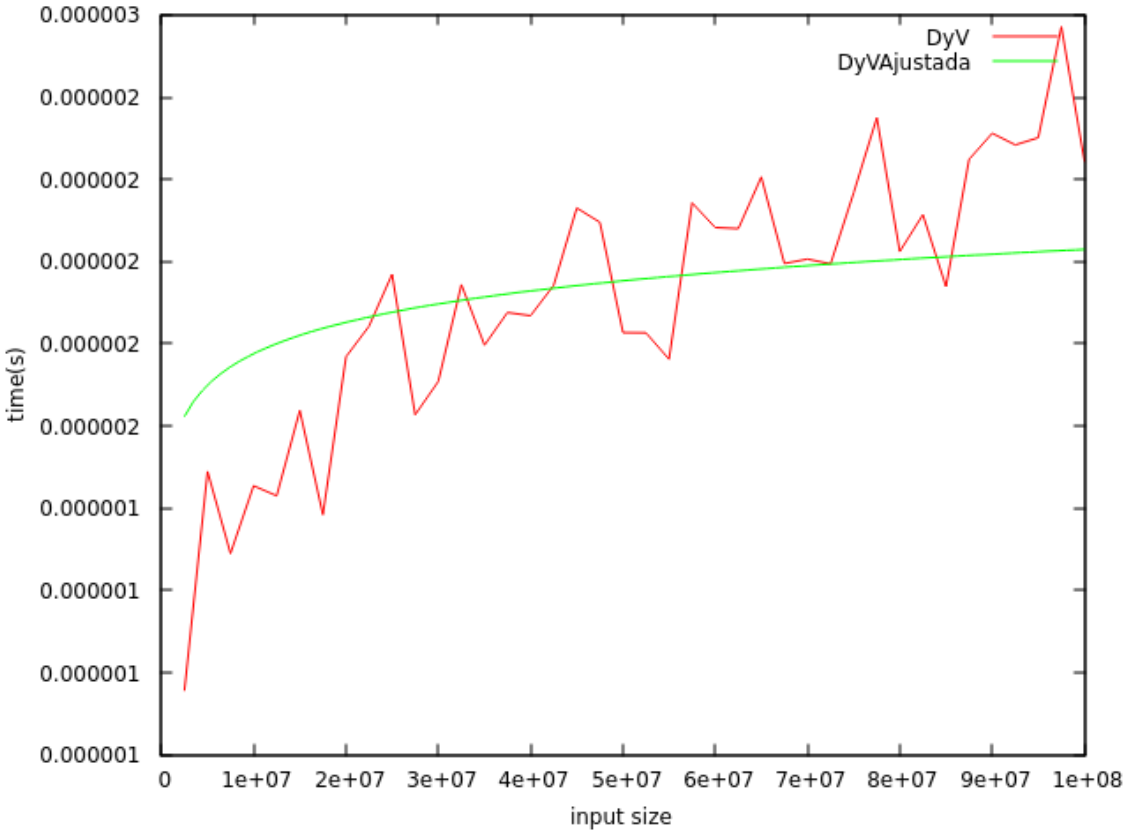

Figura 4: Gráfica del algoritmo *Divide y Vencerás*. Compilación sin optimización. Intel® Core™ i7-5500U CPU @ 2.40GHz.

El análisis híbrido de los datos representados en la gráfica produce el siguiente resultado:

- $a = 1{,}1017 \cdot 10^{-7}$

$f(x) = 1{,}1017 \cdot 10^{-7} \cdot \log(x)$

El error en el ajuste es de un 2,01 %.

6. Comparativa de ambos algoritmos.

6.1. Gráfica de tiempos de ejecución.

A continuación, presentamos una gráfica para observar mejor las diferencias de rendimiento entre los distintos algoritmos:

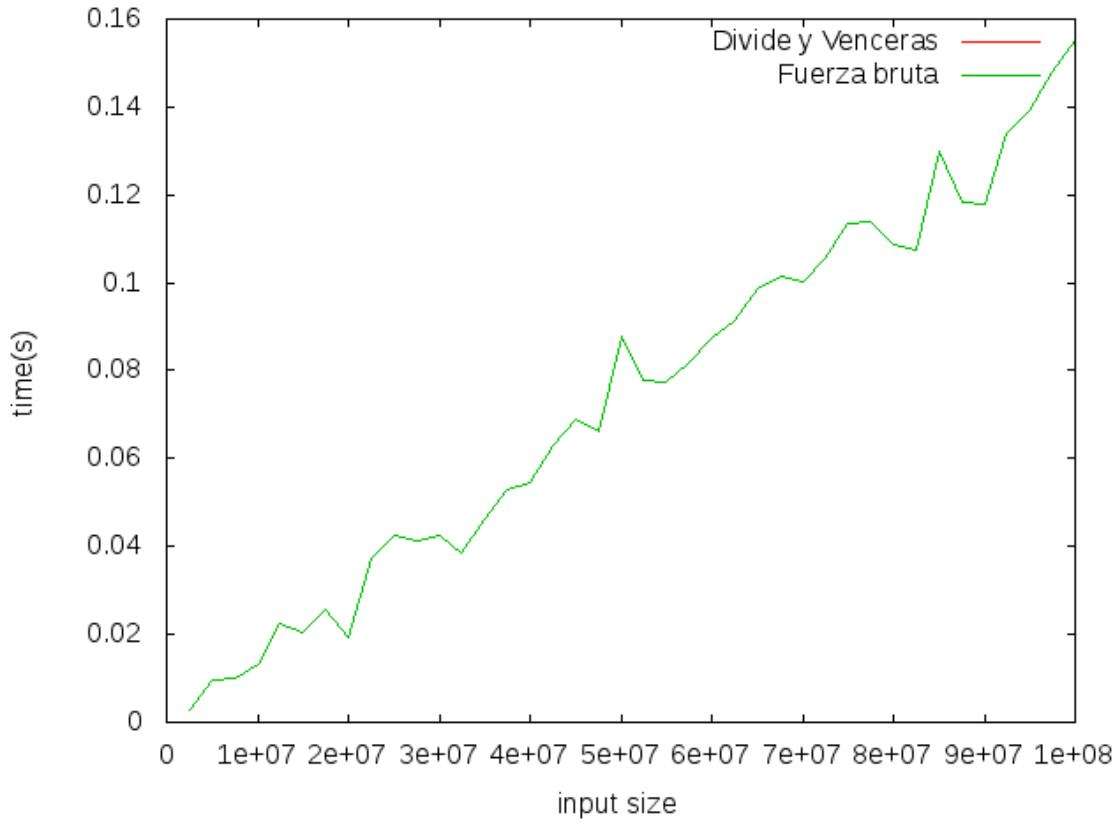

Figura 5: Gráfica comparativa de ambos algoritmos. Compilación sin optimización. Intel® Core™ i7-5500U CPU @ 2.40GHz.

La diferencia de órdenes de eficiencia de ambos algoritmos impide que los tiempos de ejecución del algoritmo *Divide y Vencerás* aparezcan siquiera en la gráfica. Para hacernos una idea, la diferencia de rendimiento es de un 3084,32 % para el tamaño de entrada más pequeño y un 69232,96 % para el tamaño de entrada más grande.

7. Conclusión.

Como hemos podido comprobar, el algoritmo *Divide y Vencerás* tiene tiempos de ejecución mejores que el algoritmo de fuerza bruta. Esto se debe a que, aunque el algoritmo *Divide y Vencerás* consume tiempo dividiendo el problema, sólo analiza la parte en la que sabe que encontrará la solución.