Práctica 2: Algoritmos Divide y Vencerás

Ignacio Aguilera Martos

Luis Balderas Ruiz

Diego Asterio de Zaballa Rodríguez

Miguel Ángel Torres López

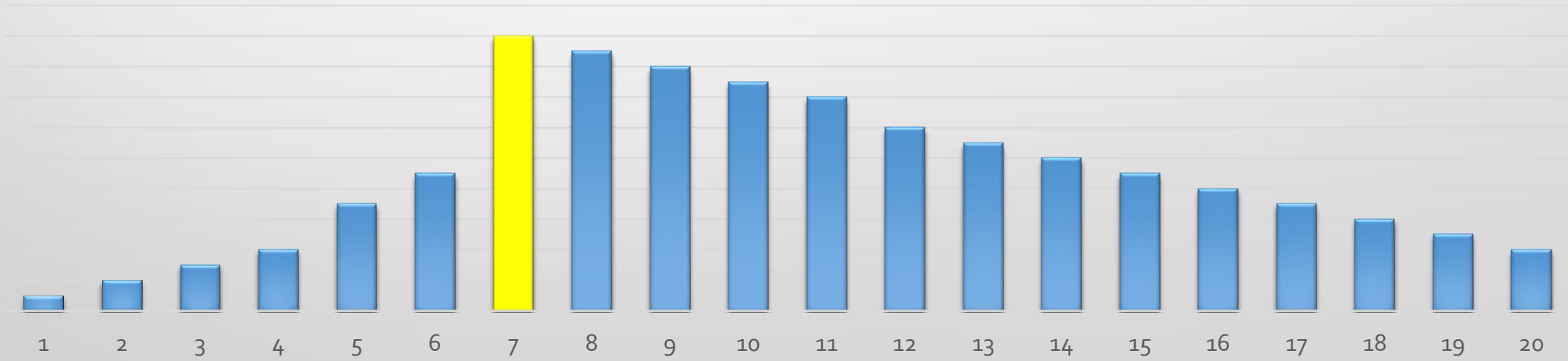
Indice

Problema de la Serie Unimodal de Números

Serie Unimodal de Números

Presentación del problema

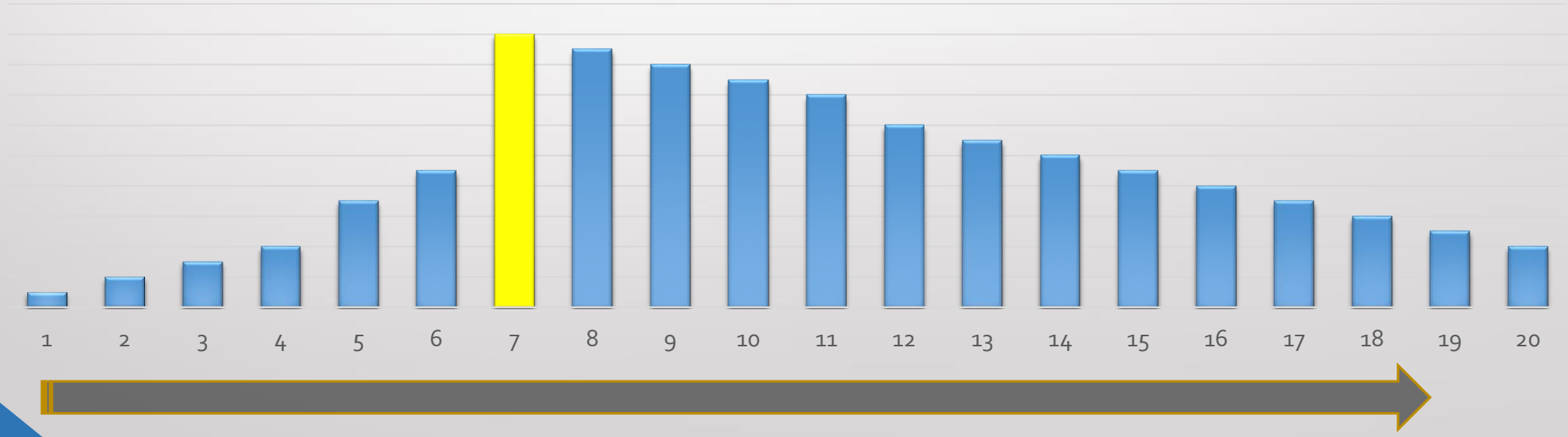
Dado un vector en el que los elementos se suponen ordenados primero de forma ascendente y, llegado un punto de cambio, de forma descendente, se pide encontrar la posición del punto de cambio.



Algoritmo “obvio”. Implementación

```
int serie_unimodal_secuencial(int *v, int n)
{
        int i=0;
        int maximo=0;
        while(v[i]<v[i+1] && ((i+1)<n))
                i++;
        return i;
}
```

- Comparar si el número de la posición actual es menor que el siguiente.
  - Caso afirmativo: Avanzamos una posición.
  - Caso negativo: Devolvemos esa posición.



Algoritmo “obvio”. Eficiencia

Datos Empíricos

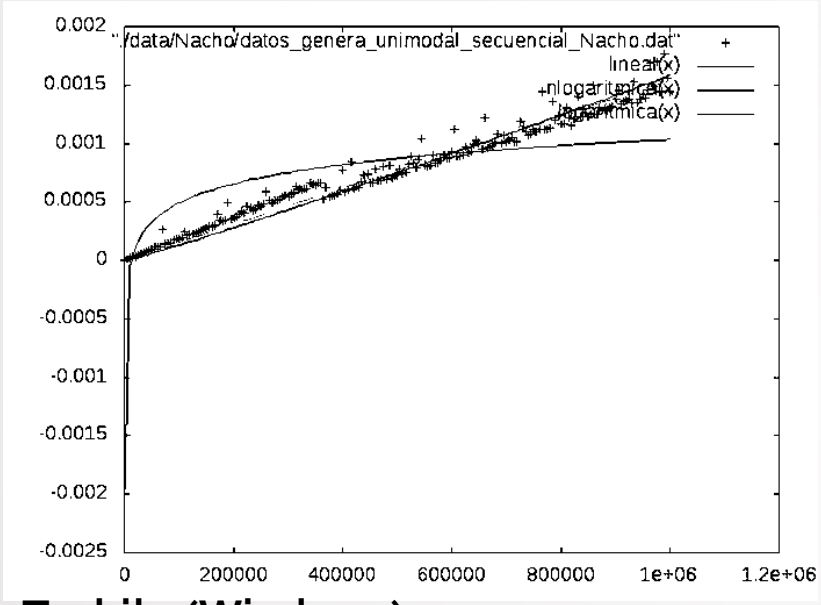
10001	1.1943e-05
15001	1.7457e-05
20001	2.3192e-05
…	…
990001	0.00222124
995001	0.00212957
1000001	0.00221753

Tras un ajuste de mínimos cuadrados con distintas gráficas sobre los datos tomados se obtienen la suma de los residuos al cuadrado:

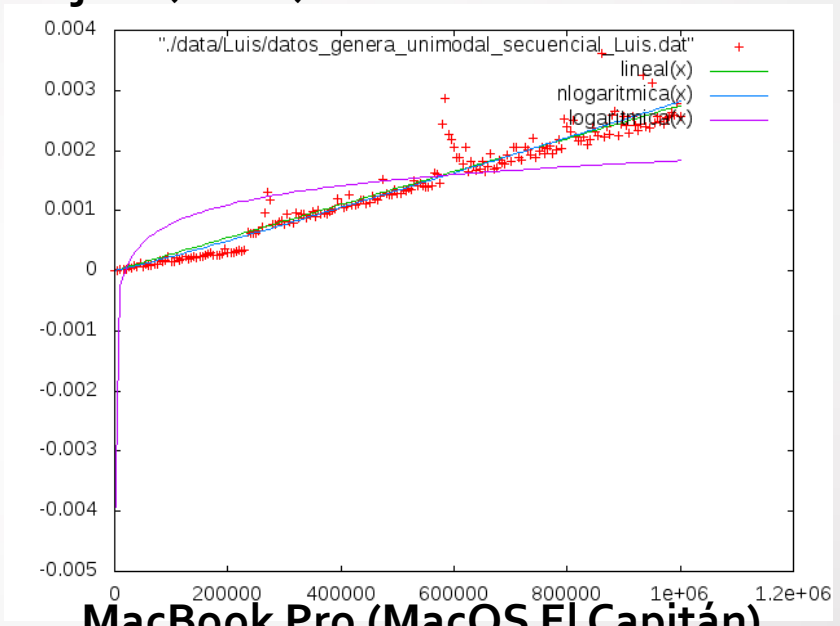
Lineal = 9.07552e-06
nlogarítmica = 9.73699e-06
logarítmica = 7.55603e-05

Algoritmo “obvio”. Eficiencia

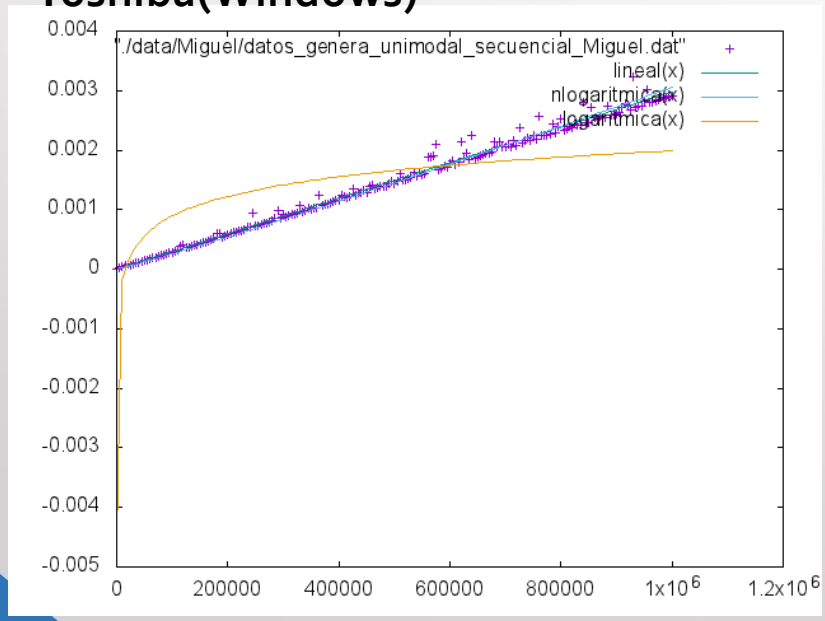
Toshiba(Linux)



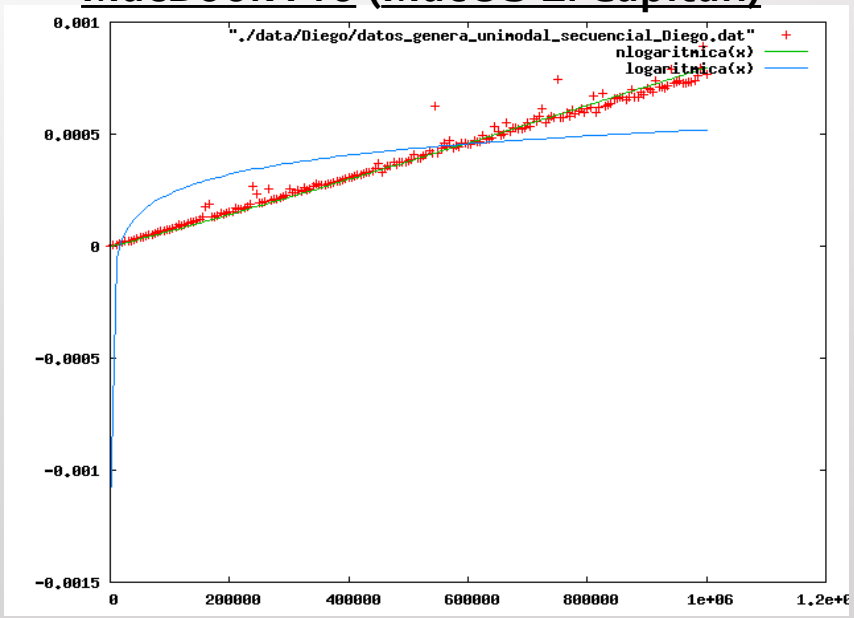
Fujitsu(Linux)



Toshiba(Windows)



MacBook Pro (MacOS El Capitán)



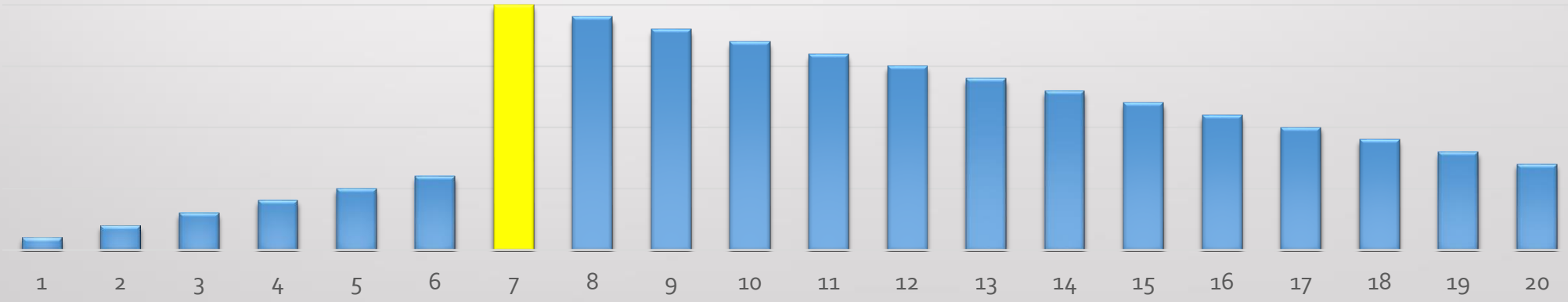
Algoritmos Divide y Vencerás

Primer algoritmo Divide y Vencerás

Primer algoritmo DyV. Implementación

```
int& buscaPuntoDeCambio(int*
v, int indice1, int indice2, int& res)
{
 int indi=(indice1+indice2)/2;
 if(v[indice1]>v[indice1+1])
 {
  res=indice1;
  return res;
 }
 else if(v[indice2]>v[indice2-1])
 {
res=indice2;
  return res;
 }
 else if(v[indi]-v[indi-1]>0 &&  v[indi]-
v[indi+1]>0)
 {
  res=indi;
  return res;
 }else
 {
  buscaPuntoDeCambio(v, indice1,
indi, res);
  buscaPuntoDeCambio(v, indi,
indice2, res);
 }
}
```

1. Dividir el vector por la mitad.
2. Comprobar si alguno de los extremos es el número buscado (creciente o decreciente).
3. Comprobamos si alguno de los dos puntos es medios es el número buscado.
4. Llamada recursiva a la mitad de vector que tenga monotonía correcta.



Primer algoritmo DyV. Eficiencia

Datos Empíricos

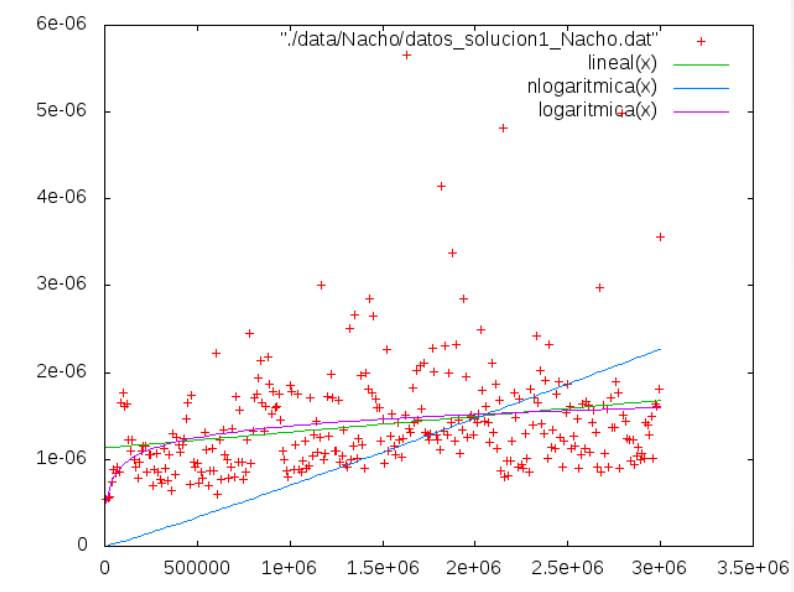
10000	7.36e-07
20000	6.27e-07
30000	6.12e-07
…	…
2970000	1.60e-03
2980000	1,72e-03
2990000	1,70e-03

Tras un ajuste de mínimos cuadrados con distintas gráficas sobre los datos tomados se obtienen la suma de los residuos al cuadrado:

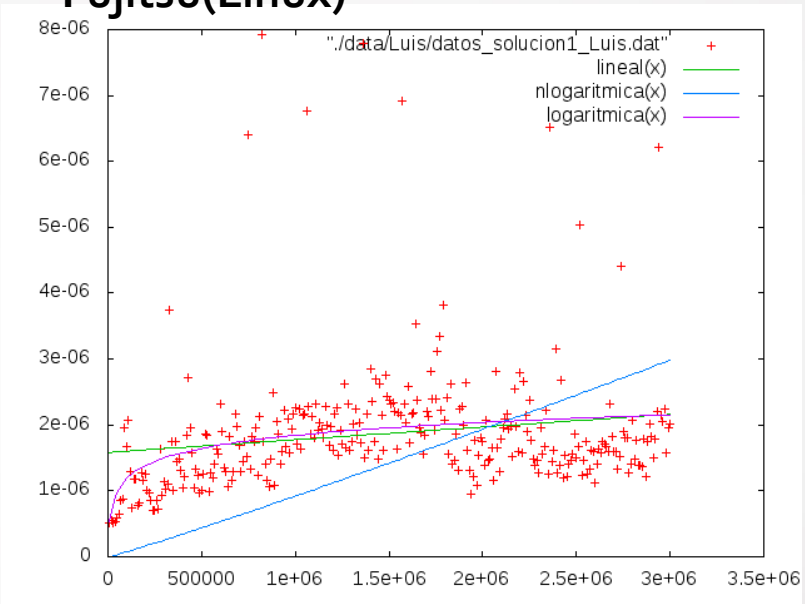
Lineal = 2.85144e-10
nlogarítmica = 3.7745e-10
logarítmica = 2.42327e-10

Primer algoritmo DyV. Eficiencia

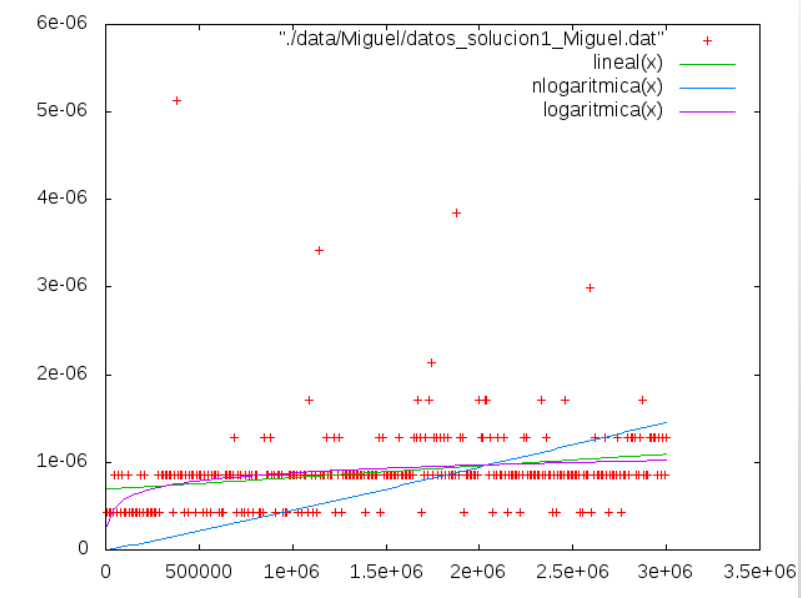
Toshiba(Linux)



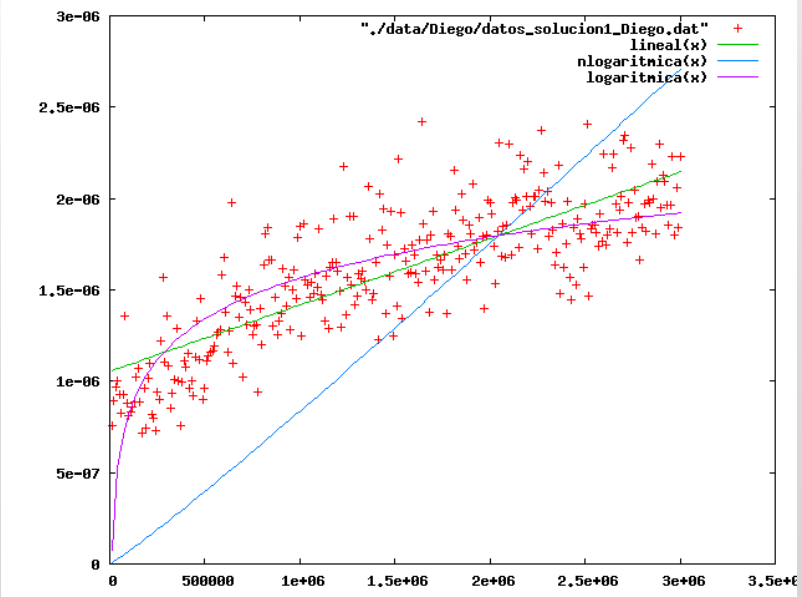
Fujitsu(Linux)



Toshiba(Windows)



MacBook Pro (MacOS El Capitán)



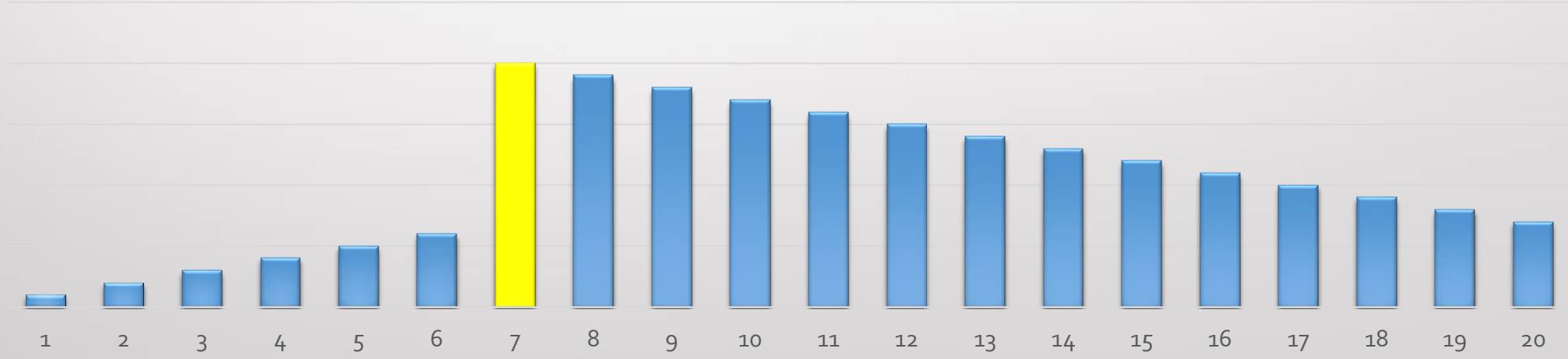
Segundo algoritmo Divide y Vencerás

Segundo algoritmo DyV. Implementación

```
int &  rift_lims(int* arr, int beg, int end, int & res){

  int N = end - beg;

  if (N  == 1) {
    res = beg;
    return arr[beg];
  }
  else if (arr [beg + N/2 - 1] < arr [end - 1])
    return rift_lims(arr, beg + N/2 , end, res);
  else
    return  rift_lims(arr,beg,  beg + N/2, res);
}


int rift(int * arr, int n, int & res){

  int beg = 0, end = n;

  return rift_lims(arr,beg,end,res);
}
```

1. Tomamos el punto medio y el extremo derecho del vector y los comparamos.
   1. El punto medio es mayor. Tomamos el vector a la izquierda.
   2. El punto extremo es mayor. Tomamos el vector a la derecha.

Repetimos el proceso con el vector elegido hasta que el vector tenga tamaño trivial.



Primer algoritmo DyV. Eficiencia

Datos Empíricos

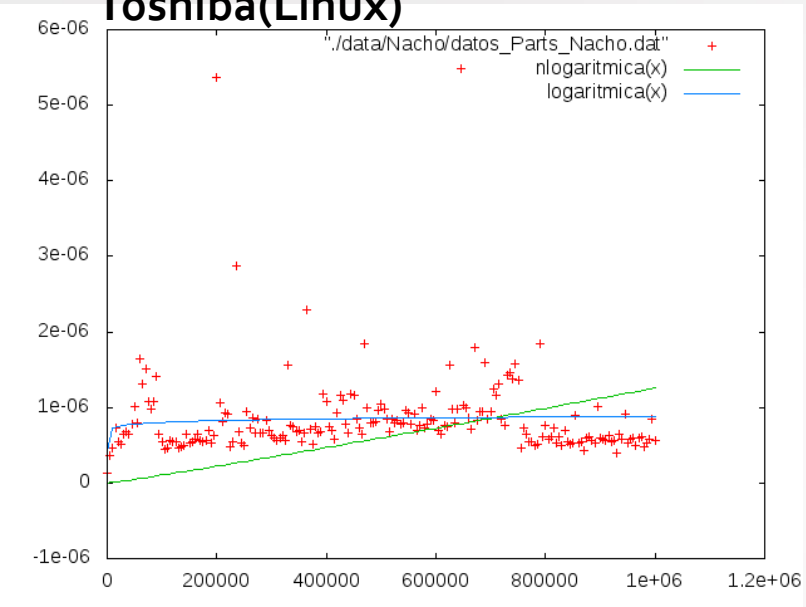
5000	3.71e-07
10000	4.65e-07
15000	7.23e-07
…	…
990000	5.72e-07
995000	8.5e-07
1000000	5.71e-07

Tras un ajuste de mínimos cuadrados con distintas gráficas sobre los datos tomados se obtienen la suma de los residuos al cuadrado:

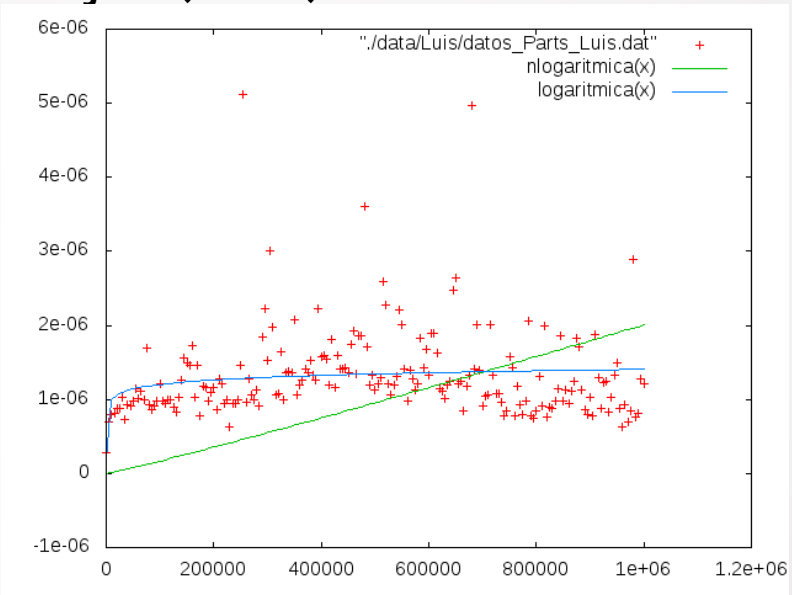
nlogarítmica = 1.0888e-10
logarítmica = 6.58058e-11

Segundo algoritmo DyV. Eficiencia

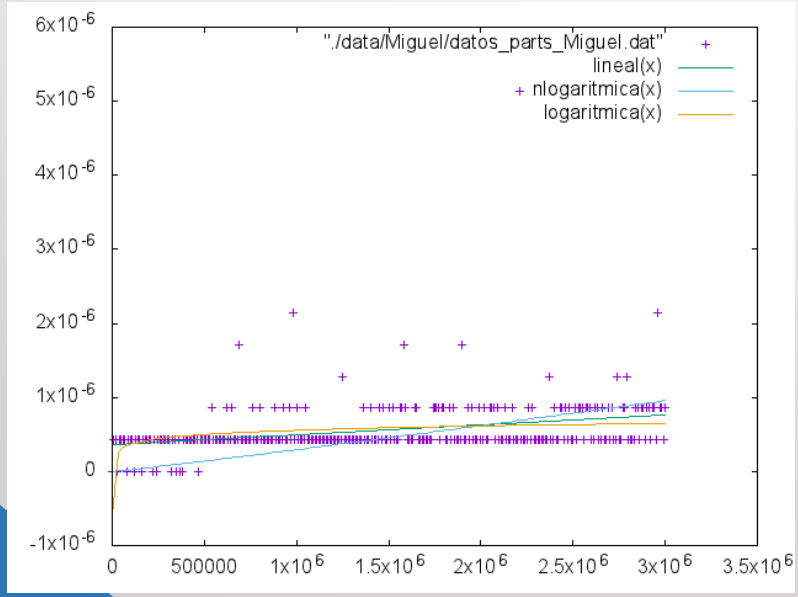
Toshiba(Linux)



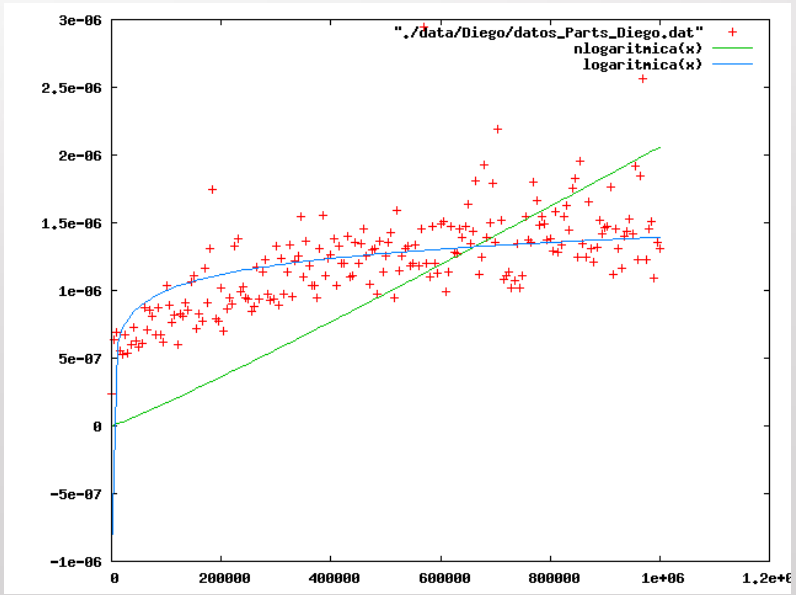
Fujitsu(Linux)



Toshiba(Windows)

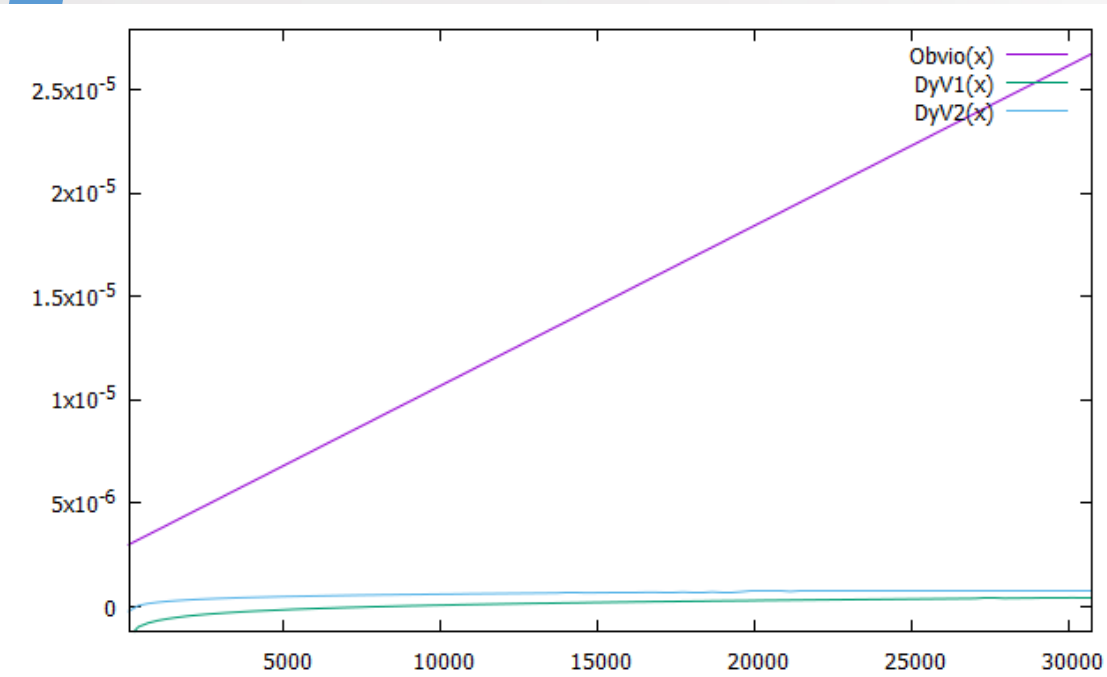


MacBook Pro (MacOS El Capitán)

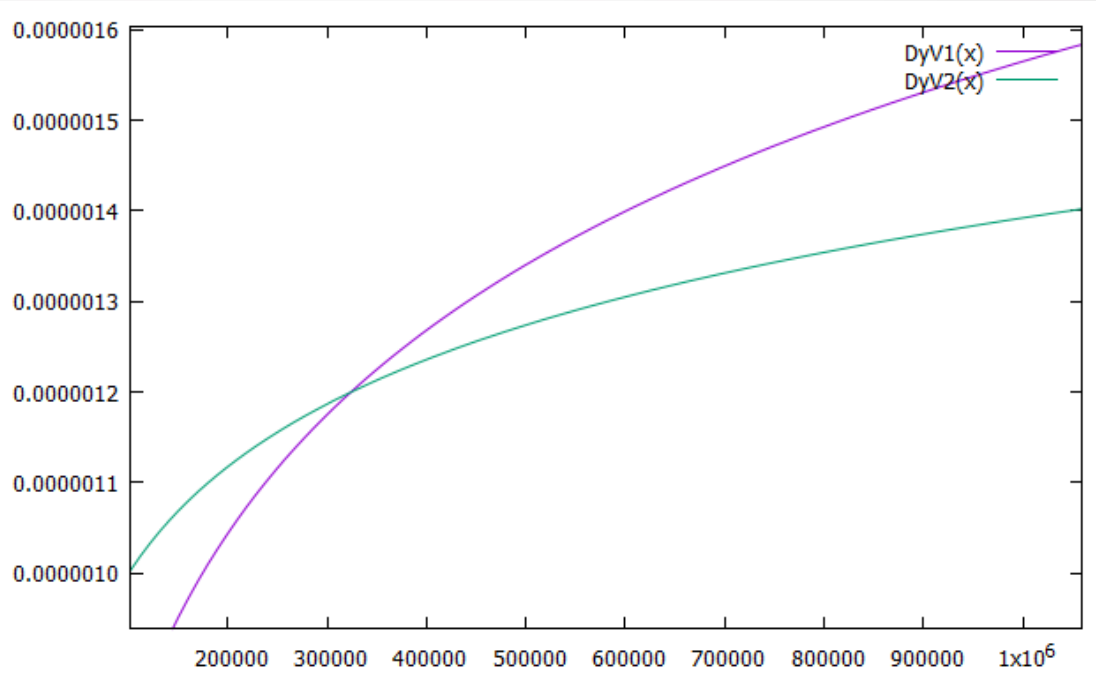


Comparación entre los algoritmos

Comparación de algoritmos



Comparación del algoritmo obvio frente a los Divide y Vencerás



Comparación entre las dos versiones de Divide y Vencerás

Podemos observar que los algoritmos DyV consiguen una mayor eficiencia que el algoritmo obvio, que es casi lineal.
Además, podemos ver que a las eficiencias de los dos algoritmos propuestos para DyV solo los separa una constante oculta.

Problema de la Comparación de Preferencias

Comparación de Preferencias

Presentación del problema

Dadas dos listas de elementos (para simplificar usaremos un vector de enteros) ordenadas según las preferencias de dos personas, encontrar la semejanza entre estas.

Para esto, contaremos el número de intercambios de preferencias que hay entre una lista y otra.

Algoritmo “obvio”. Implementación

```
int CuentaIntercambios(int* v, int tam){
 int inter=0;
 for(int i=0;i<tam;i++){
  for(int j = i; j < tam;j++){
   if(v[j]<v[i]){
    inter++;
   }
  }
 }
 return inter;
}
```

- Desde la posición actual, comprobar si en las posiciones siguientes hay alguna de menor valor.
  - Caso afirmativo. Sumamos uno por cada elemento y avanzamos a la siguiente posición.
  - Caso negativo. Avanzamos la posición actual.

1	2	3	4	5	6	7	8	9
3	1	2	4	5	8	7	9	6

Algoritmo “obvio”. Eficiencia

Datos Empíricos

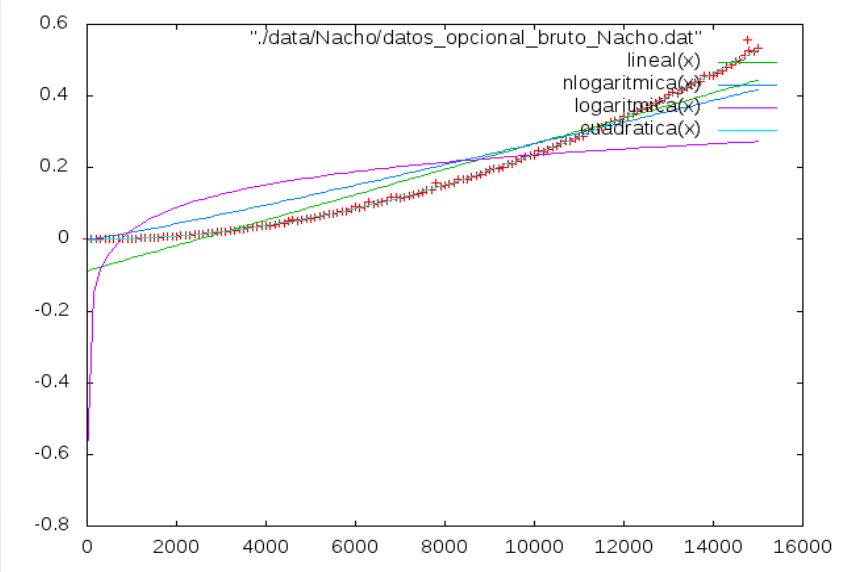
100	2.2967e-05
200	8.3385e-05
300	1.8672e-04
…	…
14800	0.51059
14900	0.514841
15000	0.527147

Tras un ajuste de mínimos cuadrados con distintas gráficas sobre los datos tomados se obtienen la suma de los residuos al cuadrado:

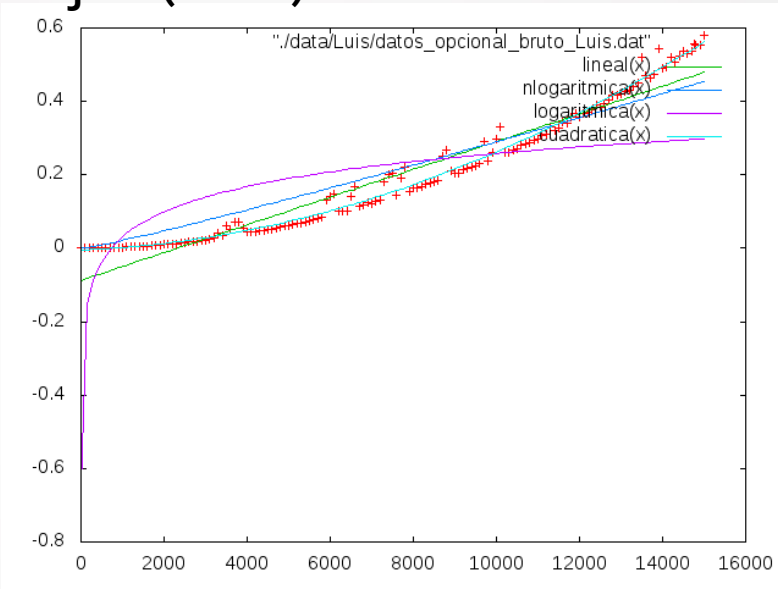
Cuadrática = 0.00112279
Logarítmica = 2.11738
nlogarítmica = 0.39753
Lineal = 0.239461

Algoritmo “obvio”. Eficiencia

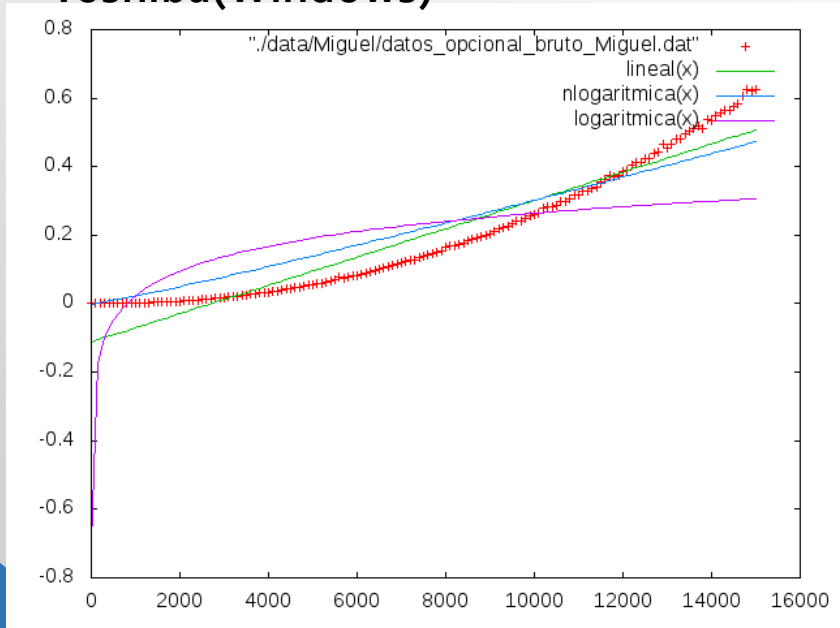
Toshiba(Linux)



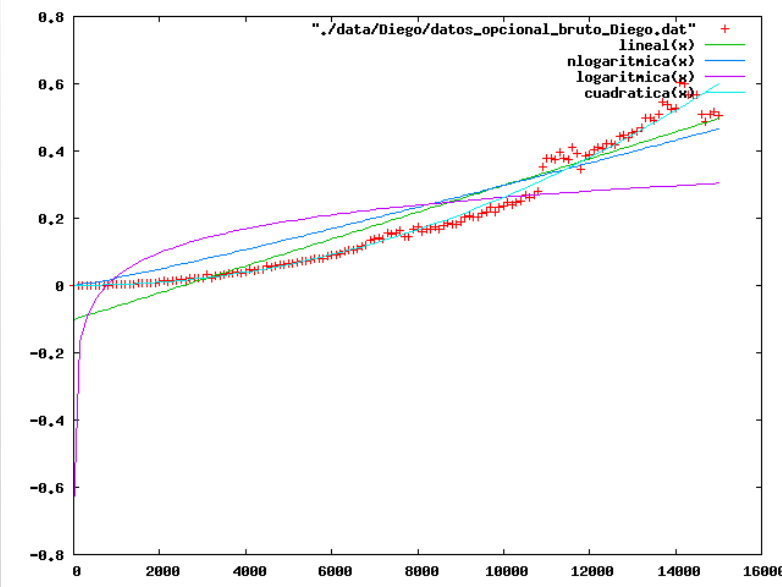
Fujitsu(Linux)



Toshiba(Windows)



MacBook Pro (MacOS El Capitán)



Algoritmo Divide y Vencerás

Algoritmo Divide y Venceras.Implementación

Este algoritmo sigue el patrón de división de vectores utilizado en mergesort. Al fusionar las distintas particiones del vector se acumulan el número de inversiones necesarias para realizar la ordenación.

1	2	3	4	5	6	7	8	9
3	1	2	4	5	8	7	9	6

MergeSort

3	1	2	4	5	8	7	9	6

Algoritmo Divide y Vencerás.Eficiencia

Datos Empíricos

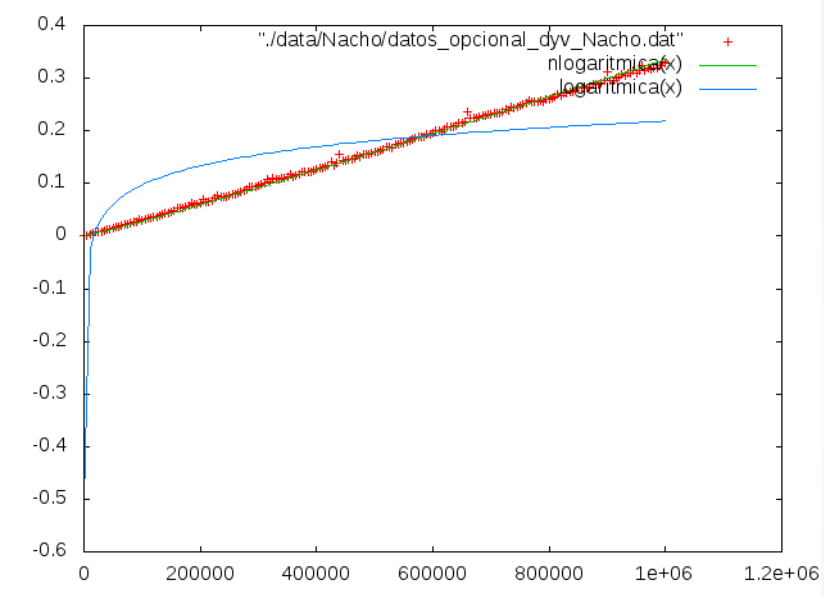
5000	1.7862e-03
10000	3.9614e-03
15000	6.0454e-04
…	…
990000	0.324368
995000	0.332126
1000000	0.337893

Tras un ajuste de mínimos cuadrados con distintas gráficas sobre los datos tomados se obtienen la suma de los residuos al cuadrado:

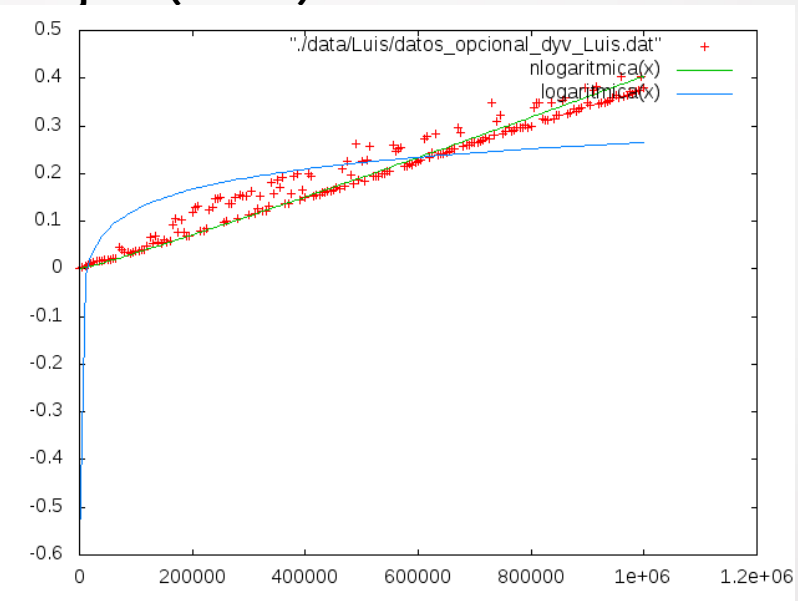
Logarítmica = 0.908574
nlogarítmica = 0.00384006

Algoritmo Divide y Vencerás. Eficiencia

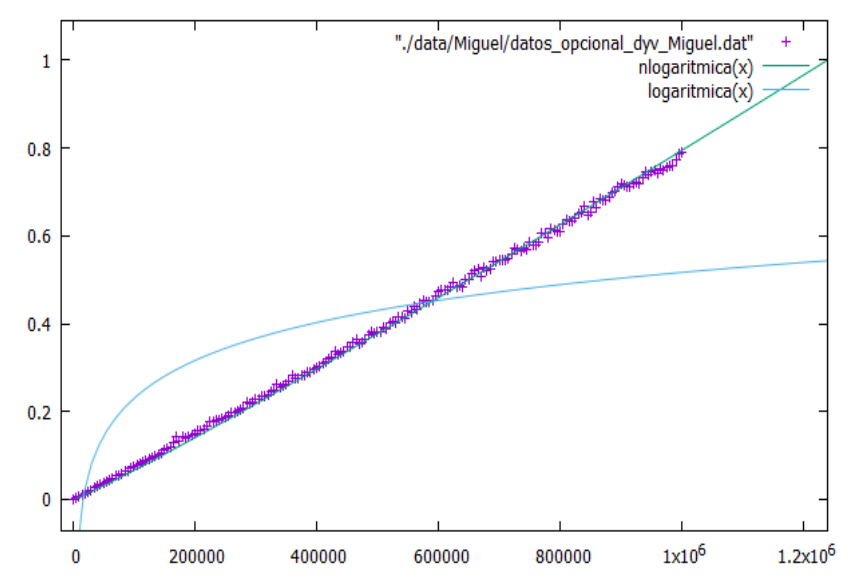
Toshiba(Linux)



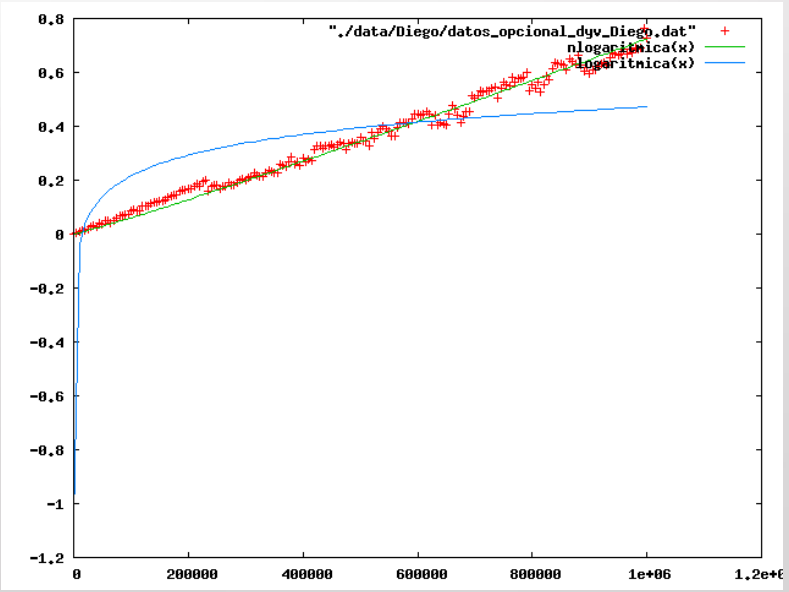
Fujitsu(Linux)



Toshiba(Windows)

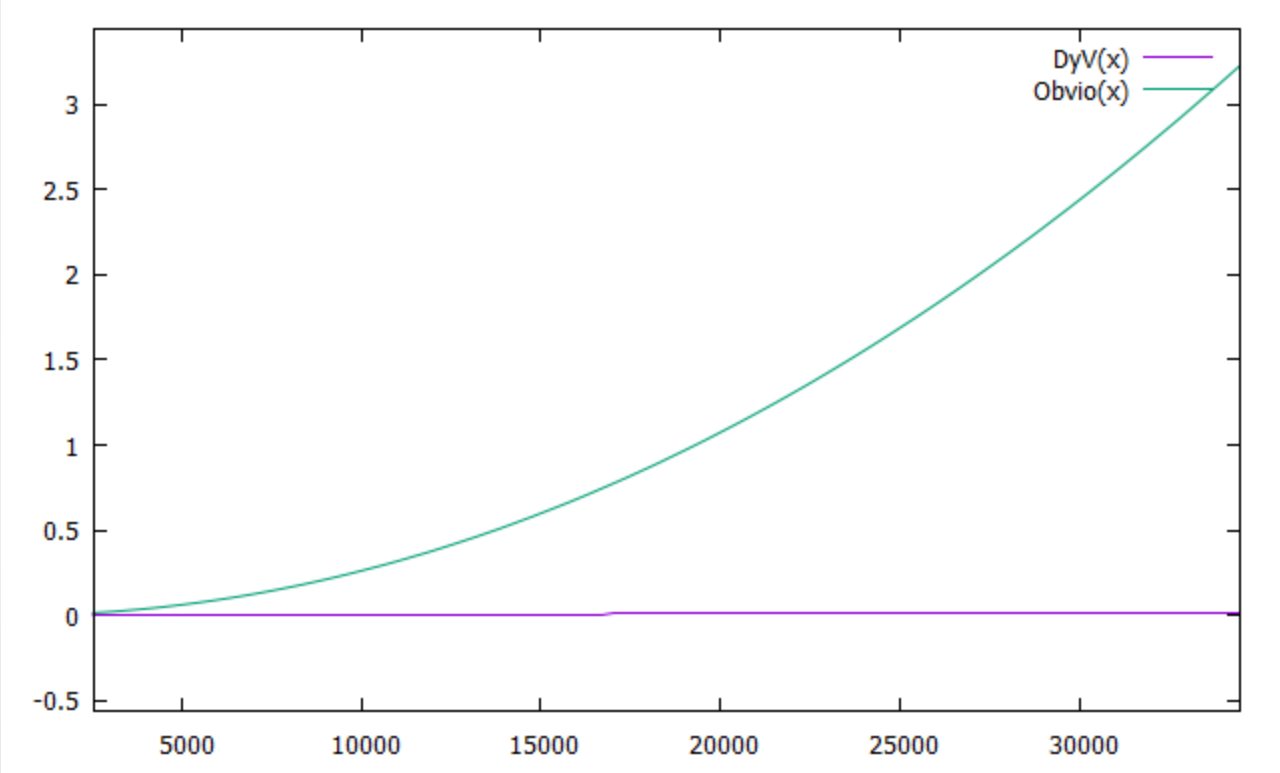


MacBook Pro (MacOS El Capitán)



Comparación entre los algoritmos

Comparación de algoritmos



Comparación del algoritmo obvio frente al Divide y Vencerás

Podemos observar que el algoritmo DyV consigue una mayor eficiencia que el algoritmo obvio, que es cuadrático. Por otro lado el algoritmo DyV se corresponde con una eficiencia nlogarítmica.