MEZCLA DE K VECTORES ORDENADOS

Ejercicio 4. Relación de problemas 1.

Jose Antonio Ruiz Millán
Adrián Peláez Vegas
Julio Antonio Fresneda García
Alejandro Rodríguez Muñoz

Descripción.

- Este problema consiste en dado un numero de vectores ordenados de tamaño n, juntar en un solo vector ordenado el resultado de la unión de todos los vectores.
- El algoritmo básico va comparando cada elemento del vector con otro elemento de otro vector y añadiendo al vector resultado, cuando estos terminan, se realiza la misma operación con el tercer vector, y así sucesivamente. Por lo tanto tenemos que este algoritmo tendrá un orden $O(nk^2)$ siendo n el numero de elementos de los vectores y k el numero de vectores.
- Nuestra **solución** permite pasar del orden $O(nk^2)$ a $O(nk \log(k))$ mejorando las prestaciones y eficiencia del calculo que tenemos que realizar, utilizando como algoritmo **Divide y Vencerás.**

Algoritmo “fuerza bruta”.

```
mezcla = matriz[0];
for( int i = 1; i < matriz.size(); i++){
mezcla = mezclarVectores(mezcla,matriz[i]);
}
```

```
vector<int> mezclarVectores(const vector<int> &arreglo1 , const vector<int>
&arreglo2){
  int x1=0, x2=0;
  vector<int> arreglo3;

  while (x1<arreglo1.size() && x2<arreglo2.size()) {
      if (arreglo1[x1]<arreglo2[x2]) {
          arreglo3.push_back(arreglo1[x1]);
          x1++;
      }
      else if( arreglo1[x1]>arreglo2[x2] ) {
          arreglo3.push_back(arreglo2[x2]);
          x2++;
      }
      else{
        arreglo3.push_back(arreglo1[x1]);
        x1++;
        x2++;
      }
  }
  while (x1<arreglo1.size()) {
      arreglo3.push_back(arreglo1[x1]);
      x1++;
  }
  while (x2<arreglo2.size()) {
      arreglo3.push_back(arreglo2[x2]);
      x2++;
  }
  return arreglo3;
}
```

Algoritmo “fuerza bruta”.

Teniendo varios vectores para ordenar, los dos primeros vectores se tardaría un tiempo de $n + n$.

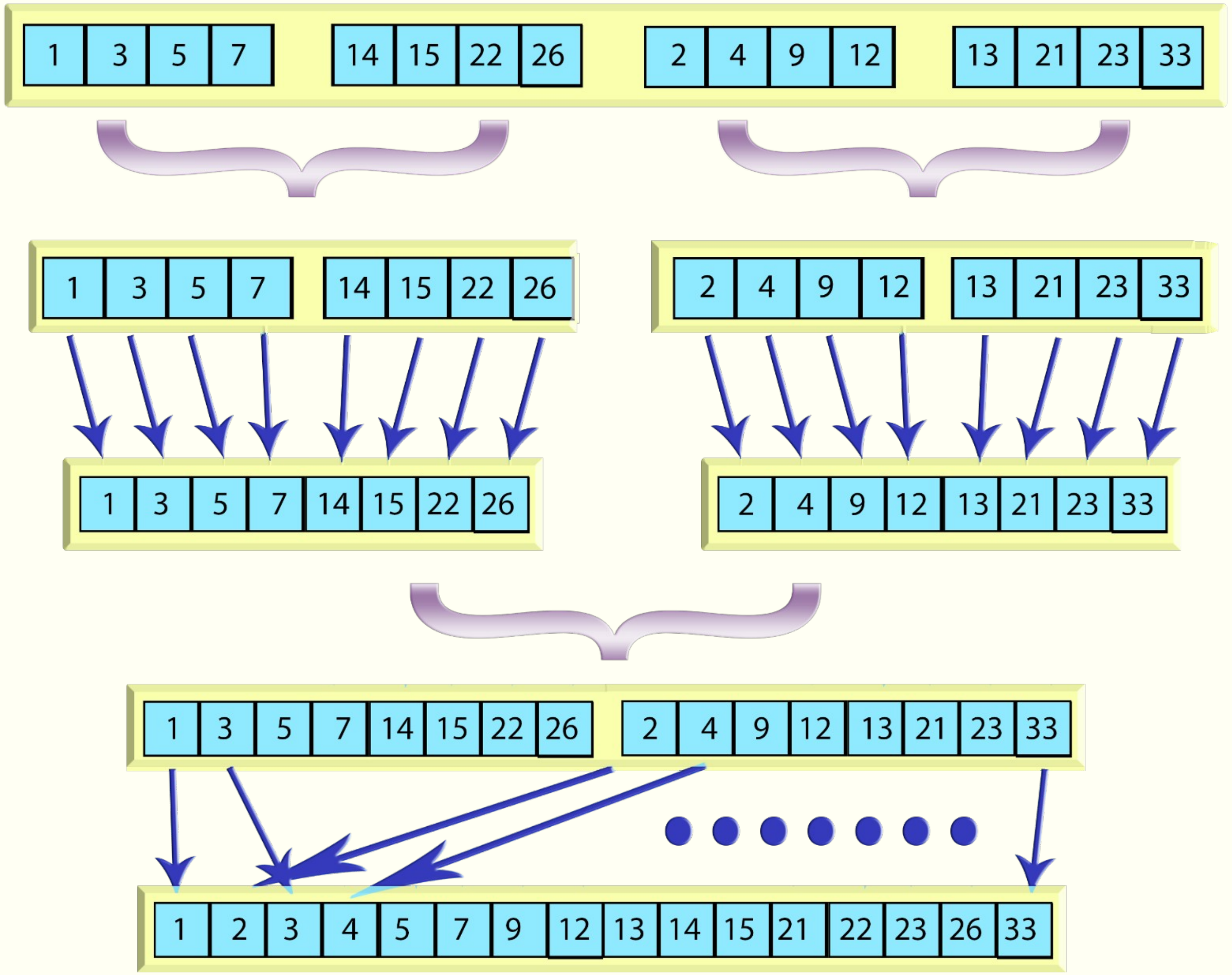
Para mezclar el resultado con el tercero, $2n + n$. Con el cuarto, $3n + n$. Sucesivamente, resultaría un tiempo de $(k-1)n + n$.

Por tanto, el tiempo total de ejecución es:

$$\sum_{i=1}^{k-1}(in + n) = n\sum_{i=1}^{k-1} i + n\sum_{i=1}^{k-1} 1 = n\frac{k(k-1)}{2} + (k-1)n = n\frac{(k-1)(k+2)}{2}$$

Es decir, $O(nk^2)$

Solución al problema.



Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás.

```cpp
vector<int> mezclarKvectores(vector<vector<int>> v){

  vector<vector<int>> v1;
  vector<vector<int>> v2;
  vector<int> r1;
  vector<int> r2;
  vector<int> r;

  if(v.size() > 2){

    for(int i = 0; i < v.size(); i++){
      if(i < v.size()/2){
        v1.push_back(v[i]);
      }
      else{
        v2.push_back(v[i]);
      }
    }

    r1 = mezclarKvectores(v1);
    r2 = mezclarKvectores(v2);
  }
  else{
    r1 = v[0];
    r2 = v[1];
  }

  mezclar2vectores(r1, r2, r);

  return r;
}
```

```cpp
void mezclar2vectores(const vector<int> &arreglo1,const vector<int> &arreglo2,
vector<int> &arreglo3)
{
    int x1=0, x2=0;

    while (x1<arreglo1.size() && x2<arreglo2.size()) {
        if (arreglo1[x1]<arreglo2[x2]) {
            arreglo3.push_back(arreglo1[x1]);
            x1++;
        }
        else if( arreglo1[x1]>arreglo2[x2] ) {
            arreglo3.push_back(arreglo2[x2]);
            x2++;
        }
        else{
          arreglo3.push_back(arreglo1[x1]);
          x1++;
          x2++;
        }
    }
    while (x1<arreglo1.size()) {
        arreglo3.push_back(arreglo1[x1]);
        x1++;
    }
    while (x2<arreglo2.size()) {
        arreglo3.push_back(arreglo2[x2]);
        x2++;
    }
}
```

Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás.

Teóricamente, vamos a suponer que k es una potencia de 2. $k = 2^m$, por lo que mezclaríamos las $k/2$ parejas de vectores (1-2,3-4,5-6...) de longitud n. Una vez mezcladas, se mezclarían las $k/4$ parejas de vectores, de longitud $2n$: (1-2 con 3-4, 5-6 con 7-8...), y así sucesivamente.

Para calcular la eficiencia vamos a usar expansión.

Tenemos que $T(k) = 2T(k/2) + c_2nk$.

$2T(k/2)$ por que llamamos recursivamente a la función dos veces, pasándole $k/2$ como argumento.

c_2nk por que la acción de combinar dos vectores es lineal.

Vamos a calcular $T(k/2)$.

$T(k/2) = 2T(k/4) + c_2n(k/2)$.

Entonces, $T(k) = 4T(k/4) + c_2 2nk$.

También $T(k) = 8T(k/8) + c_2 3nk$.

Se puede concluir que $T(k) = 2^iT(k/2^i) + c_2ink$.

Al llegar al final de la recurrencia, va a haber un momento en el cual en el lado derecho tengamos $T(1)$.

Ésto quiere decir, que cuando $i = m$, tendremos que $T(k) = 2^mT(1)+c_2mnk$.

Sabiendo que $2^m = k$, y $m = log_2k$, y tomando $T(1)$ como c_1, tenemos lo siguiente:

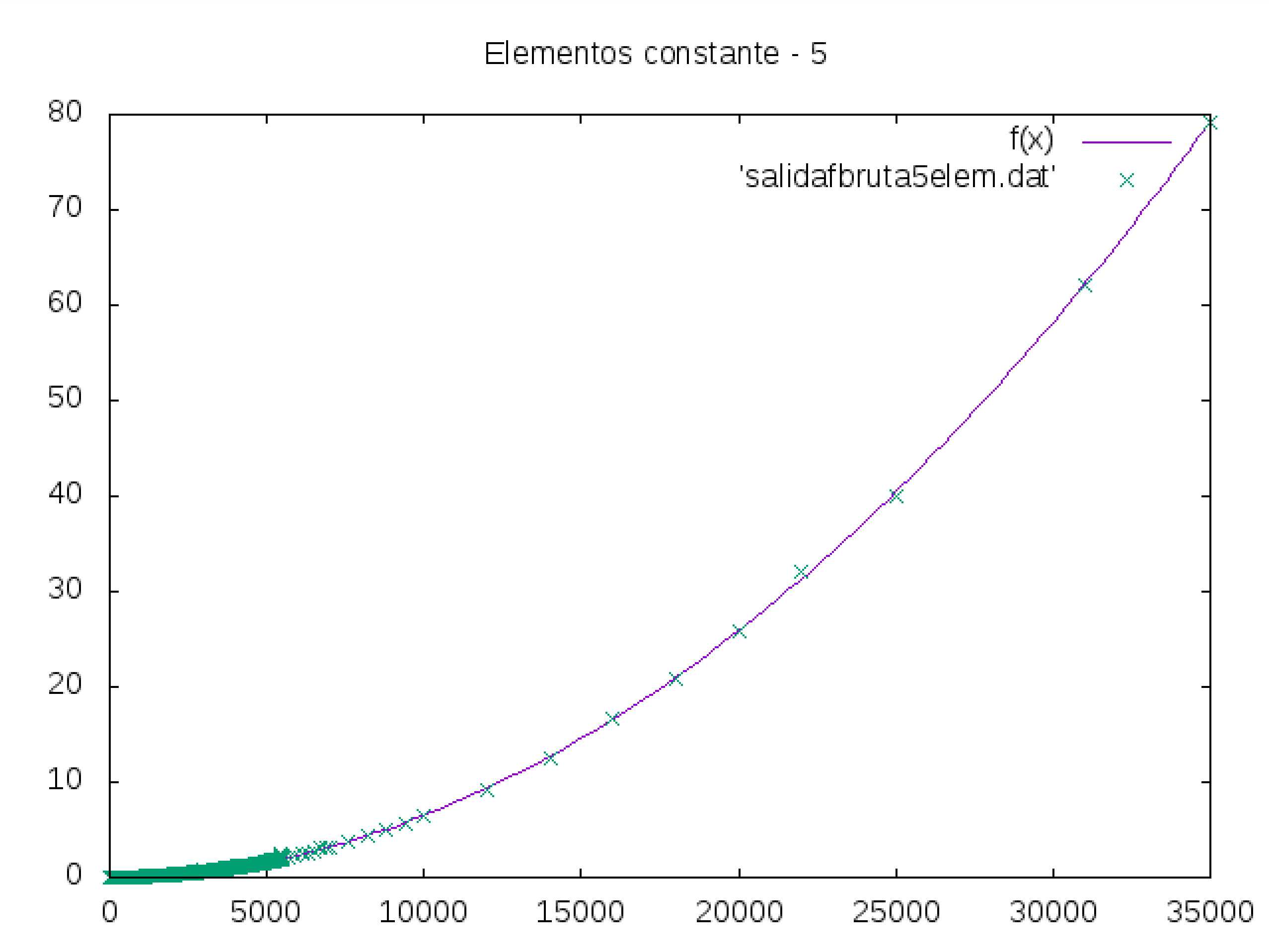
$T(k) = c_1k + c_2nklog_2k$.

$nklog_2k$ hace despreciable k, por lo que tenemos que la eficiencia es $O(log(k)*nk)$.

Solución al problema con un ejemplo.

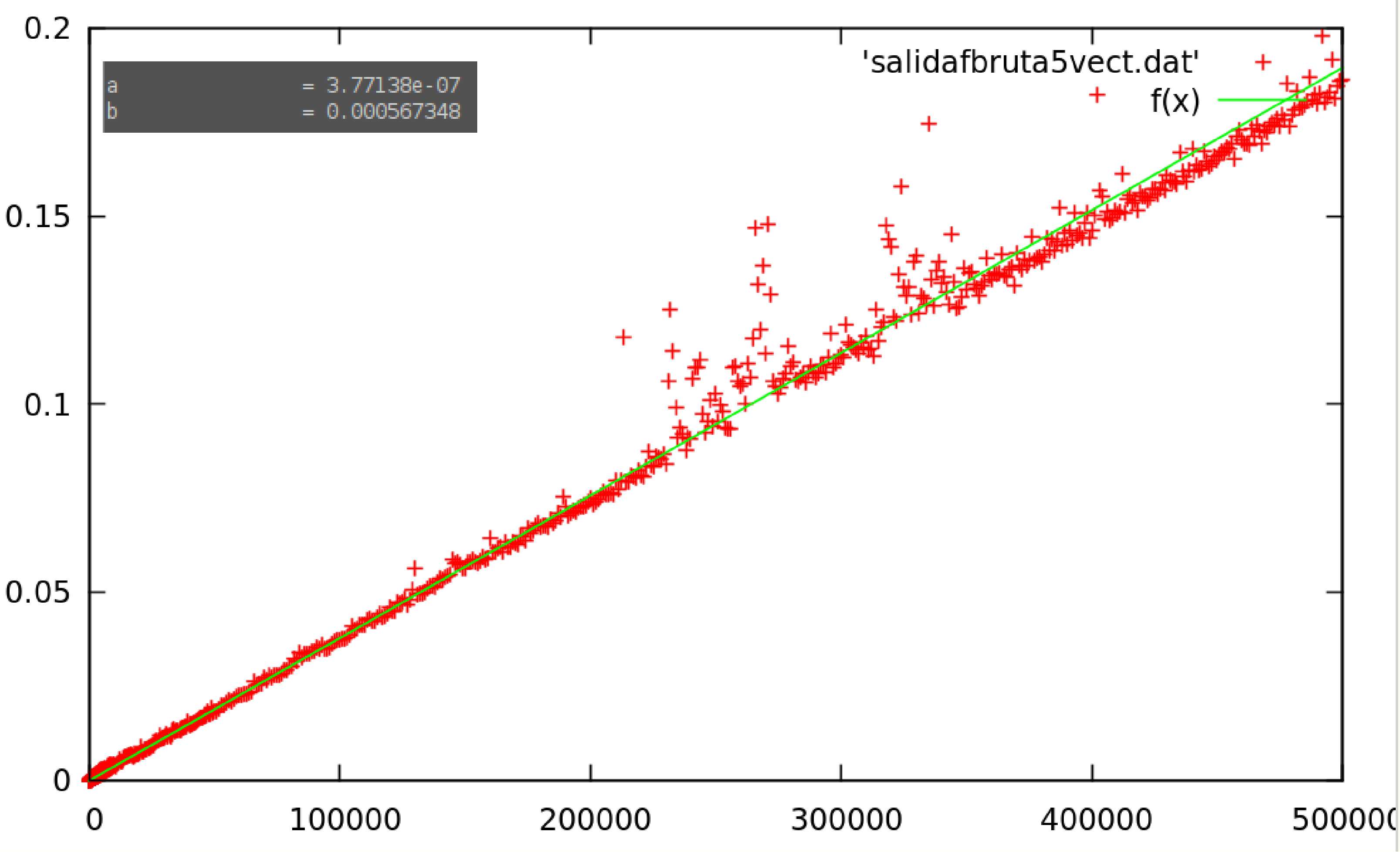
Gráficas comparativas.

Grafica de algoritmo por “fuerza bruta”; con n CONSTANTE ; $O(k^2)$:



Gráficas comparativas.

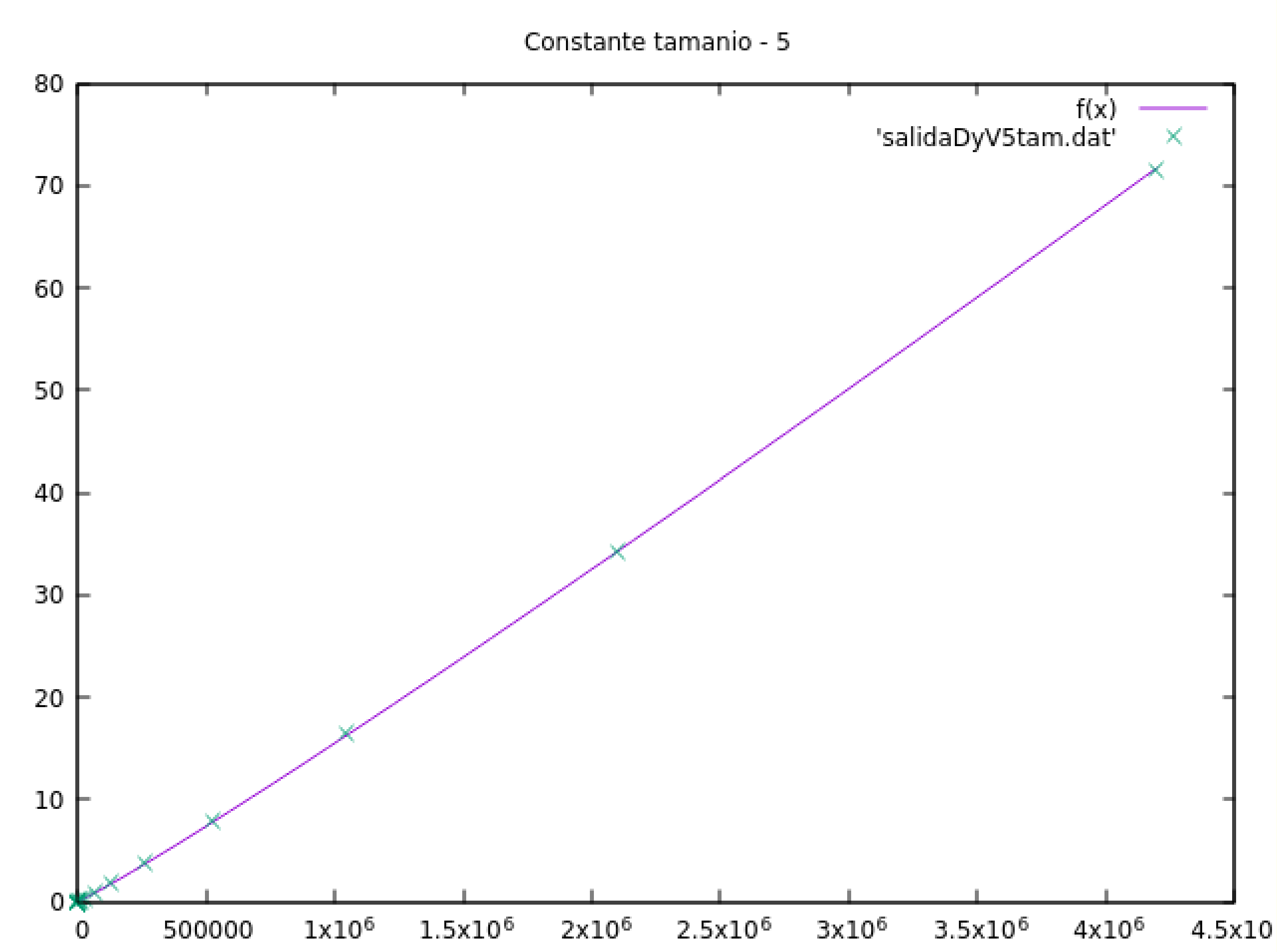
Grafica de algoritmo por “fuerza bruta”; con k CONSTANTE ; O(n):



Solución al problema con un ejemplo.

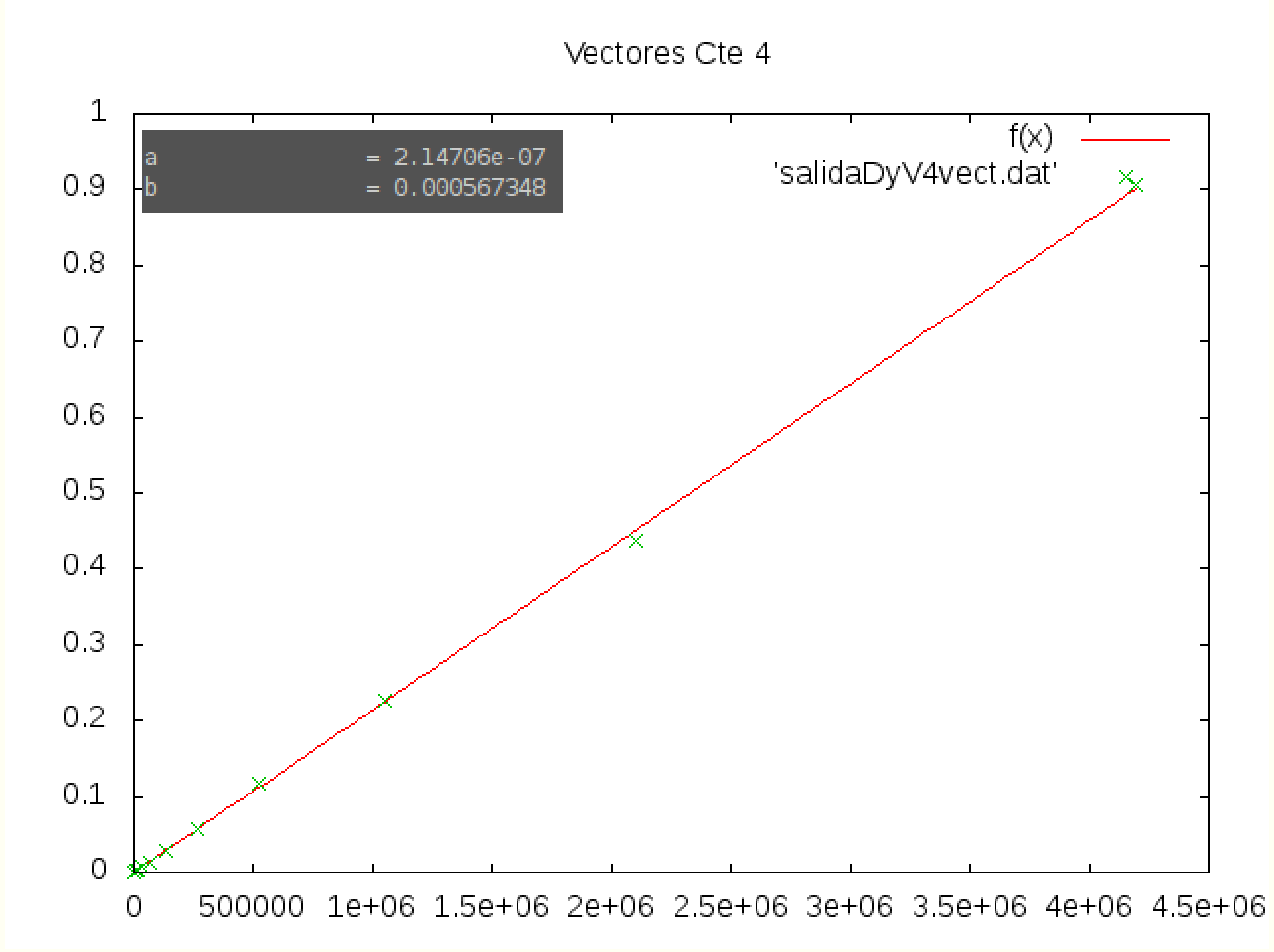
Gráficas comparativas.

Grafica de algoritmo **Divide y Vencerás**; con n CONSTANTE O(k log(k)):



Gráficas comparativas.

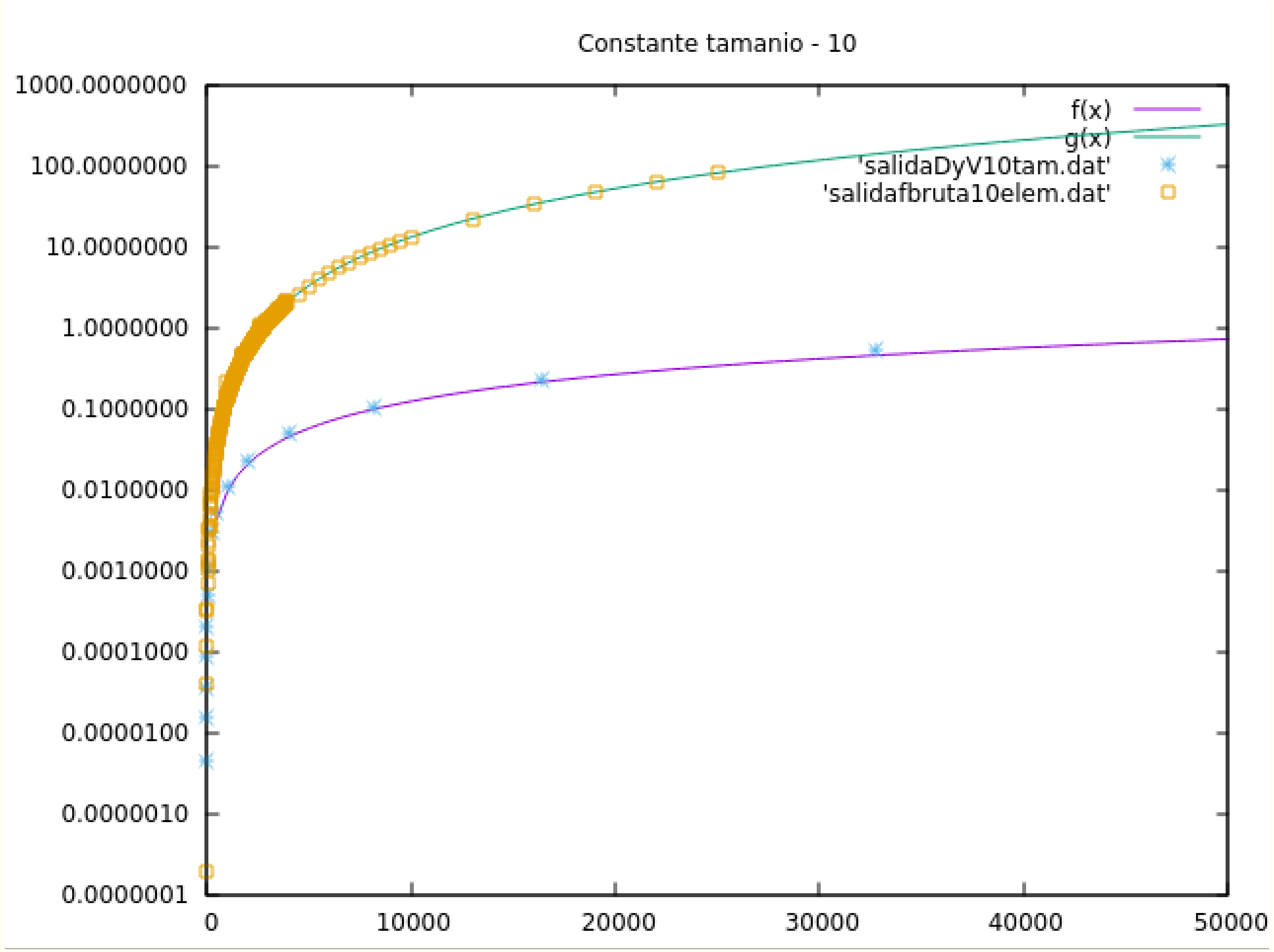
Grafica de algoritmo **Divide y Vencerás**; con k CONSTANTE O(n):



Solución al problema con un ejemplo.

Gráficas comparativas.

Grafica **comparativa** ; con n CONSTANTE:



Gráficas comparativas.

Grafica **comparativa** ; con k CONSTANTE: