CADA ELEMENTO EN SU POSICIÓN

Ejercicio 3. 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 vector con números distintos, hallar si existe un índice tal que V[i] == i.
- El algoritmo básico va comparando cada elemento del vector, recorriéndolo hasta encontrar el elemento. Por lo tanto tenemos que este algoritmo tendrá un orden O(n).
- Nuestra **solución** permite pasar del orden O(n) a O(log(n)) mejorando las prestaciones y eficiencia del calculo que tenemos que realizar, utilizando como algoritmo **Divide y Vencerás (Búsqueda binaria)**.

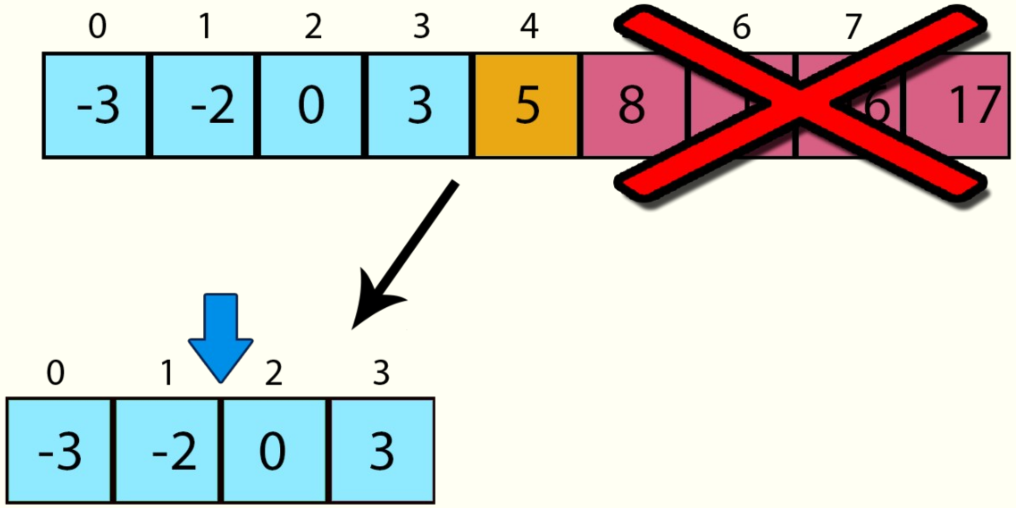
Algoritmo “fuerza bruta”.

```
bool bBinaria(const vector <int> &arr){

    for(int i=0; i<arr.size(); i++){
            if(arr[i]==i){
                    return true;
            }
    }

    return false;

}
```

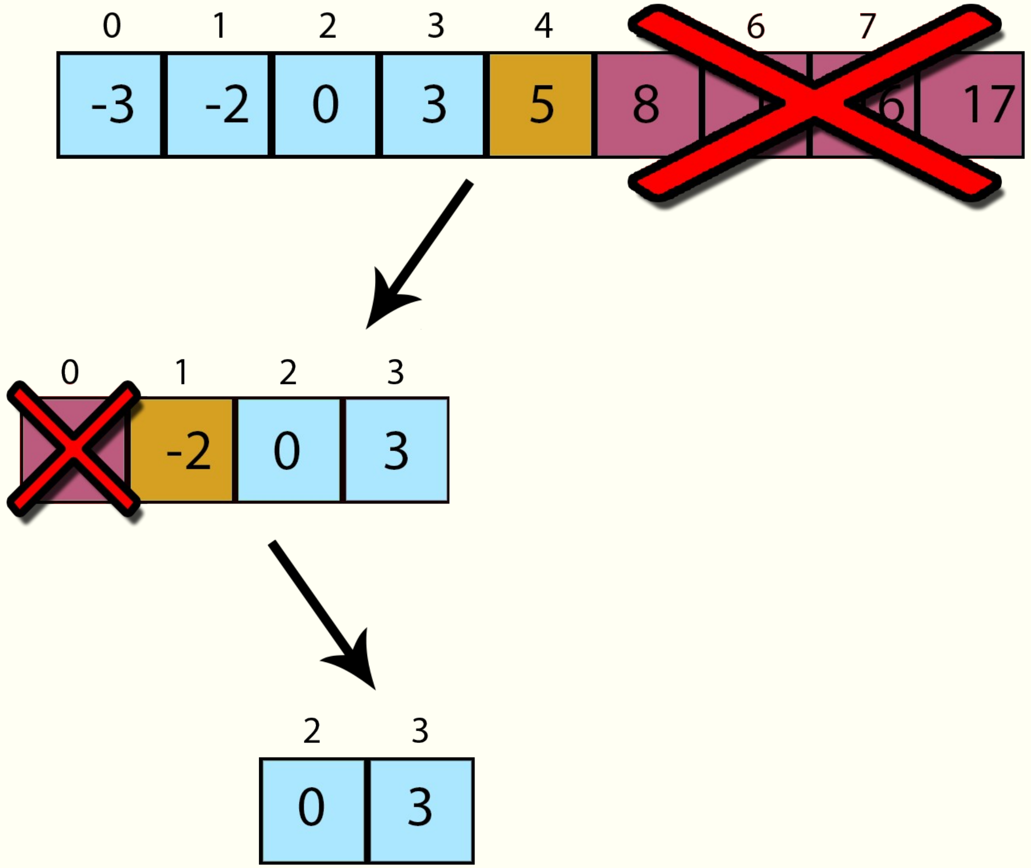
Solución al problema (sin repetidos).

0	1	2	3	4	5	6	7	8
-3	-2	0	3	5	8	11	16	17

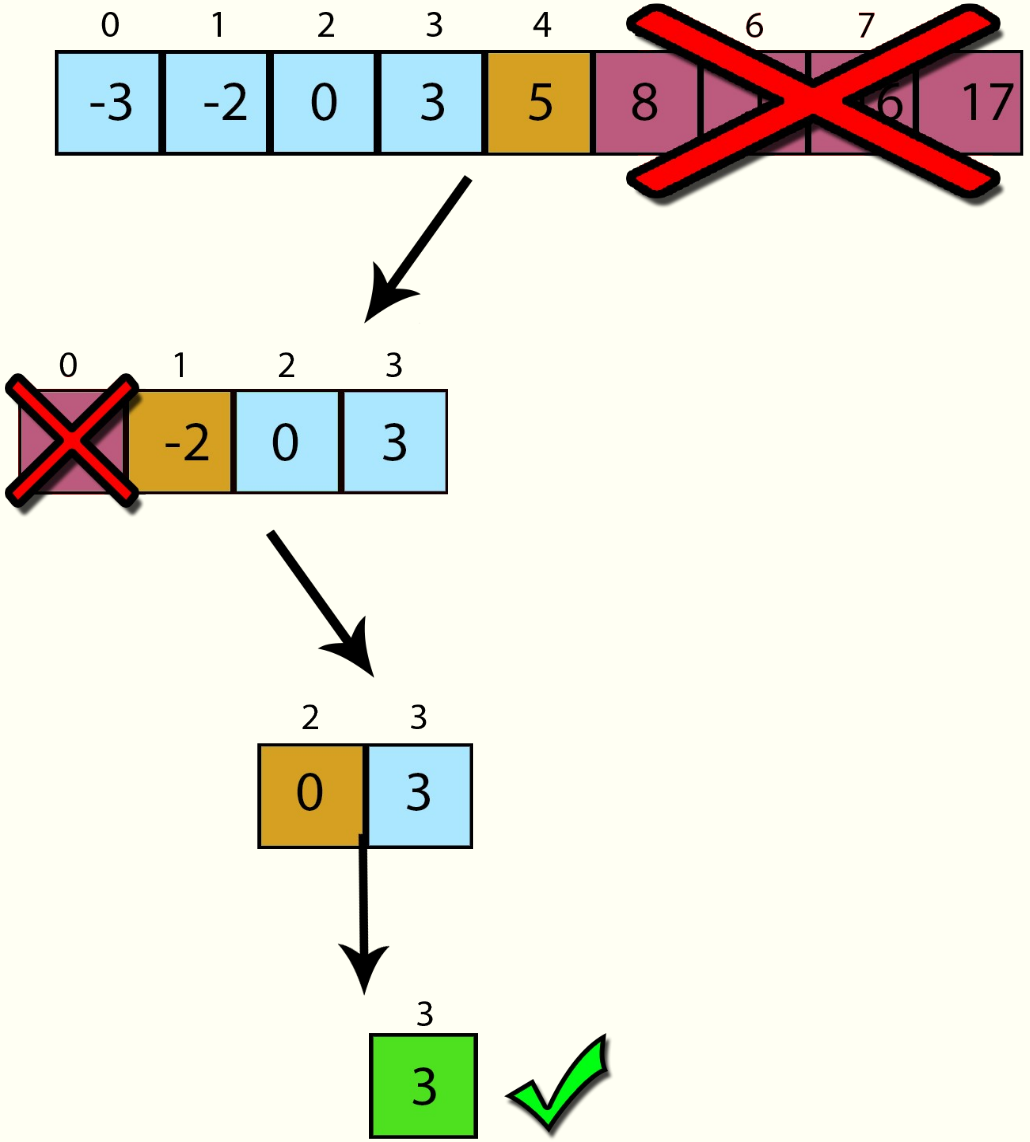
Solución al problema (sin repetidos).



Solución al problema (sin repetidos).



Solución al problema (sin repetidos).



Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás (sin repetidos).

```
bool bBinaria(const vector<int> &arr, int primero, int ultimo){
    int aux;
    bool res = false;

    aux = (primero+ultimo)/2;
    if(arr[aux]!=aux && (primero>=ultimo)){
            return false;
    }

    if(arr[aux]==aux){
            return true;
    }

    if(arr[aux]<aux){
            res = bBinaria(arr, aux+1, ultimo);
    }

    else{
            res = bBinaria(arr, primero, aux-1);
    }
    return res;
}
```

Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás (sin repetidos).

Vamos a considerar que evaluamos un vector de n elementos, siendo n potencia de 2 (2^k). En el peor caso:

$$T(n) \begin{cases} c_1 & si\ n=1 \\ T(n/2)+c_2 & si\ n>1,\ n=2^k \end{cases}$$

Para saber la eficiencia, vamos a usar expansión

$T(n) = T(n/2) + c_2$

$T(n/2) = T(n/4) + c_2$

Es decir:

$T(n) = T(n/4) + 2c_2$

o

$T(n) = T(n/8) + 3c_2$

En general:

$T(n) = T(n/2^i) + ic_2$, siendo i el número de llamadas recursivas.

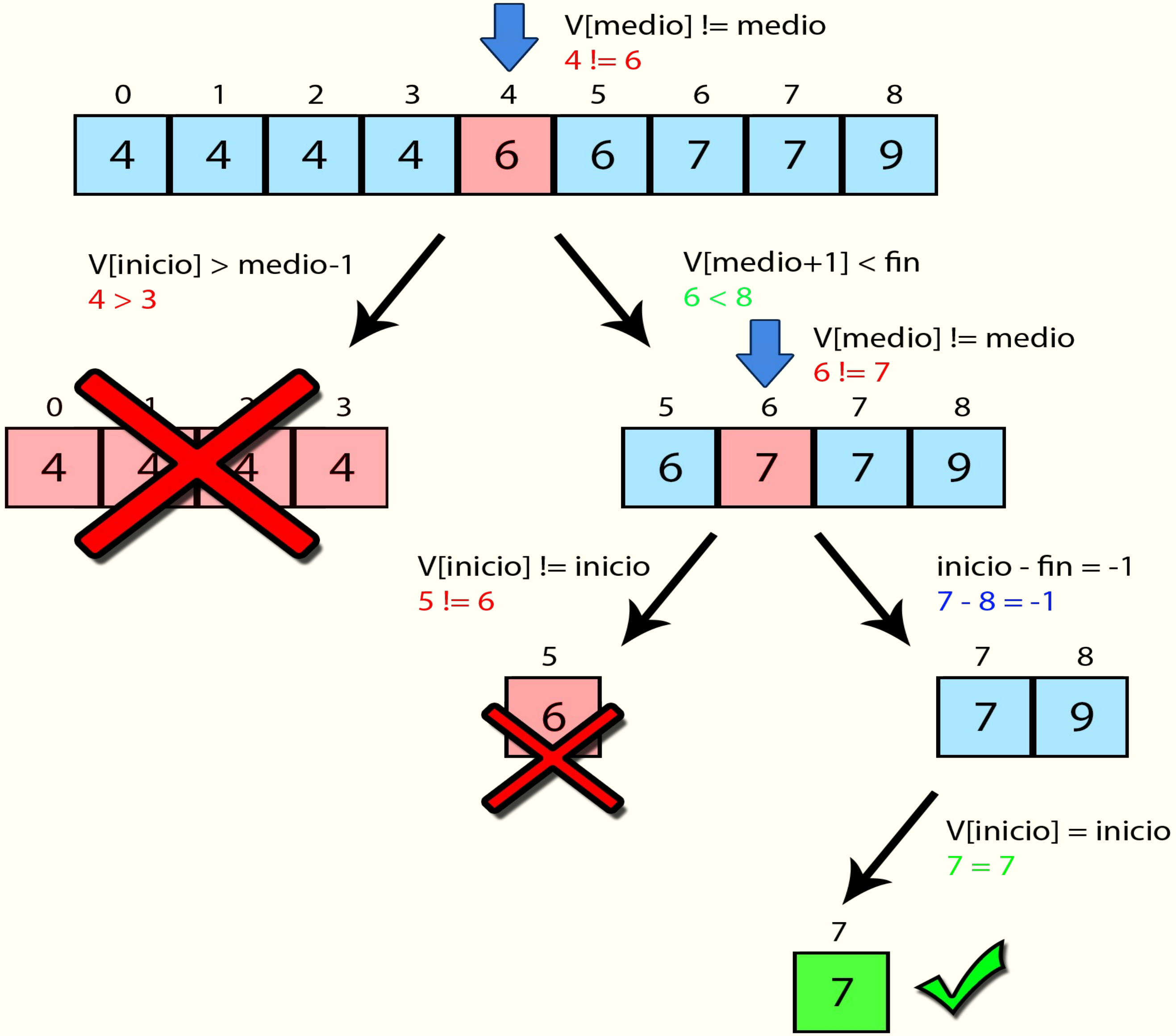
Cuando $i = k$, quiere decir que no habrá más llamadas recursivas, es decir, en la parte derecha hay $T(1)$.

La fórmula quedaría:

$T(n) = T(1) + kc_2$

Como $2^k = n$, $k = log_2(n)$. Por lo que la eficiencia sería $O(log(n))$.

Solución al problema (con repetidos).



Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás (con repetidos).

```
bool en_su_posicion( const vector<int> &v, int ini, int fin, int &pos )
{
  int medio = (ini+fin)/2;
  if( v[medio] == medio ){
    pos = medio;
    return true;
  }

  if( ini - fin > 1 || ini - fin < -1 ){

    if( ( v[ini] > medio-1 || v[medio-1] < ini ) && ( v[medio+1] > fin || v[fin] < medio+1 ) ) return
false;
    if( v[ini] > medio-1 || v[medio-1] < ini ) return en_su_posicion(v,medio+1,fin,pos);
    if( v[medio+1] > fin || v[fin] < medio+1 ) return en_su_posicion(v,ini,medio-1,pos);
    return ( en_su_posicion(v,medio+1,fin,pos) || en_su_posicion(v,ini,medio-1,pos) );

  }
  if( ini - fin == 1 || ini - fin == -1 ){
    return( v[ini] == ini || v[fin] == fin );
  }
  return false;
}
```

Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás (con repetidos).

Vamos a considerar que evaluamos un vector de n elementos, siendo n potencia de 2 (2^k).

En el peor caso:

$$T(n)\begin{cases} c1 & si\, n{=}1 \\ 2T(n/2){+}c_2 & si\, n{>}1{,}n{=}0 \end{cases}$$

Para saber la eficiencia, vamos a usar expansión:

$T(n) = 2T(n/2) + c_2$

$T(n/2) = 2T(n/4) + c_2$

Es decir: $T(n) = 4T(n/4) + 2c_2$ o $T(n) = 8T(n/8) + 3c_2$

En general: $T(n) = 2^iT(n/2^i) + ic_2$, siendo i el número de llamadas recursivas.

Cuando $i = k$, quiere decir que no habrá más llamadas recursivas, es decir, en la parte derecha hay $T(1)$.

La fórmula quedaría: $T(n) = 2^kT(1) + kc_2$

Como $2^k = n$, $k = log_2(n)$

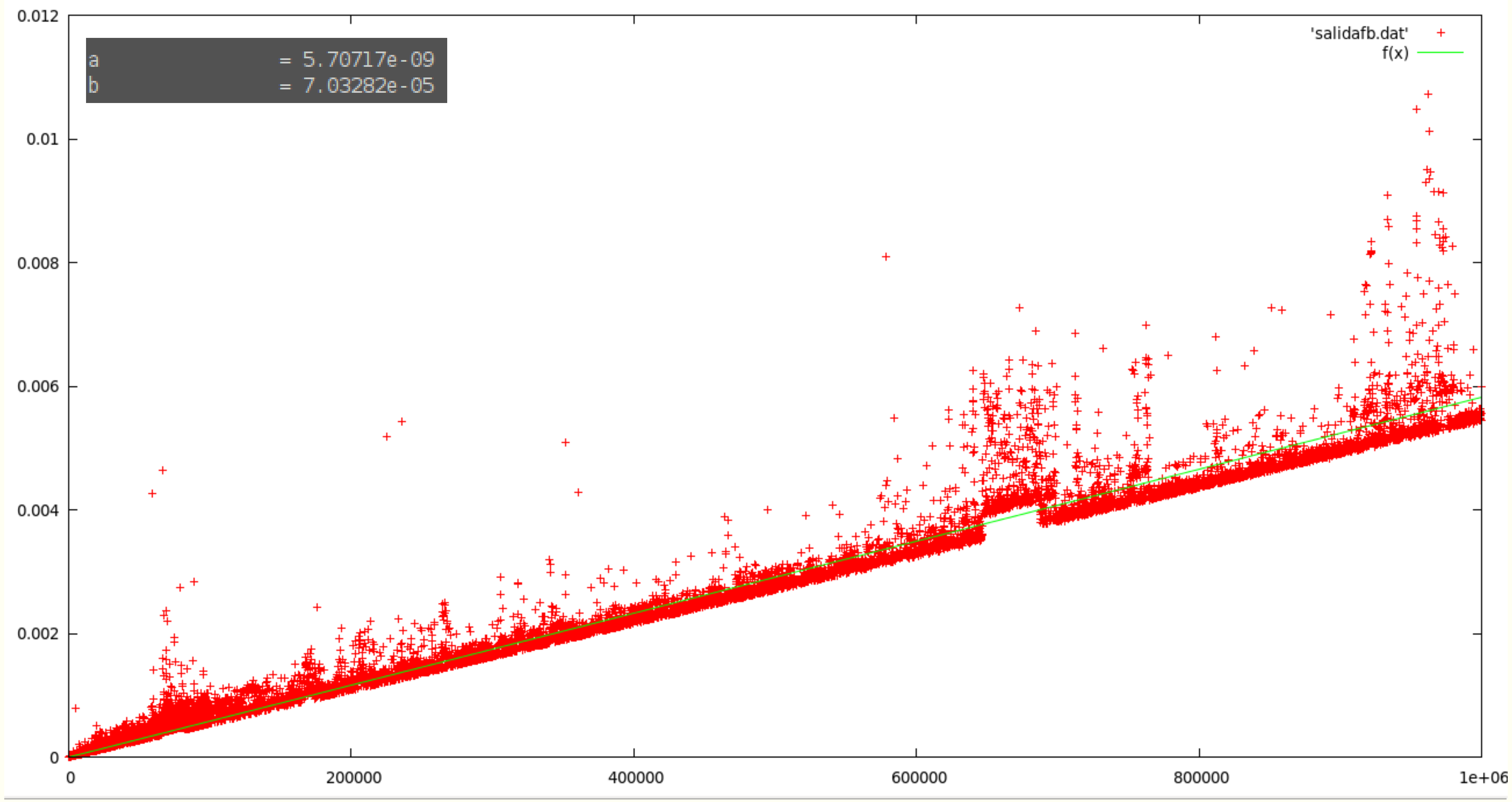
La fórmula finalmente quedaría: $T(n) = nT(1) + c_2log_2(n)$

Podemos tomar $T(1) = c_1$ Por lo que la fórmula quedaría $T(n) = c_1n + c_2log_2(n)$, por lo que la eficiencia sería $O(n)$.

Solución al problema con un ejemplo.

Gráficas comparativas.

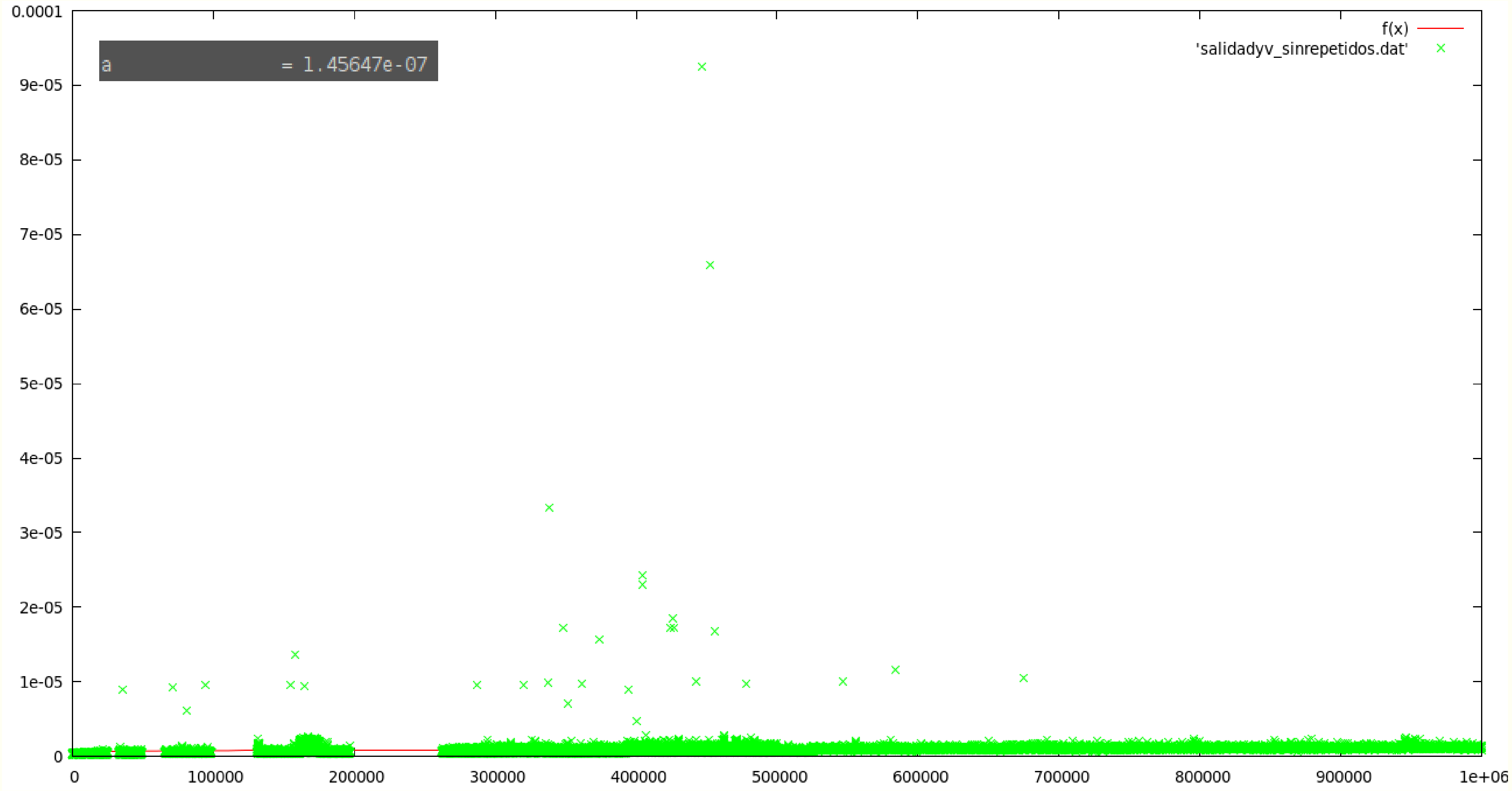
Grafica de algoritmo por “fuerza bruta”; O(n):



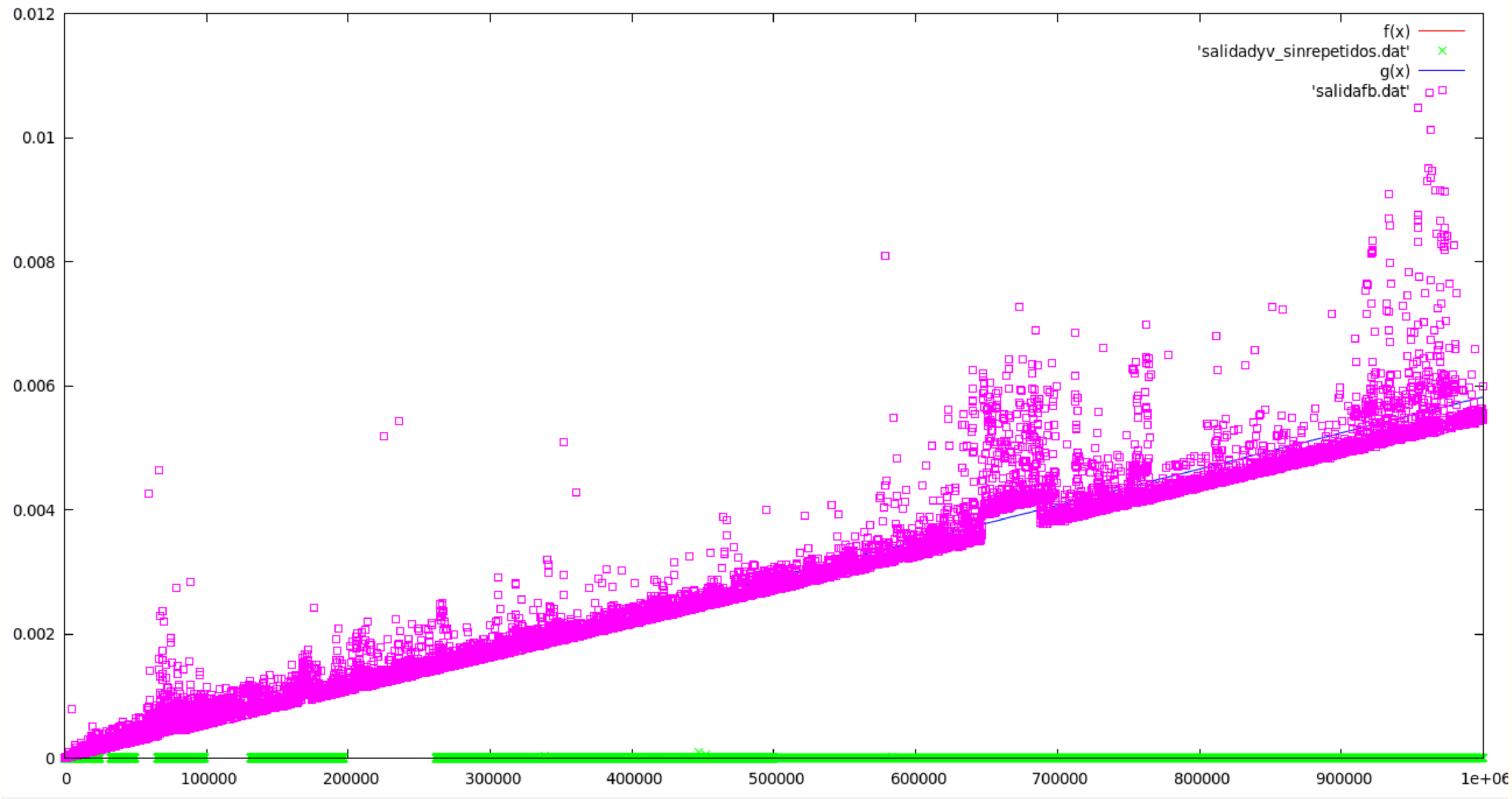
Solución al problema con un ejemplo.

Gráficas comparativas.

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

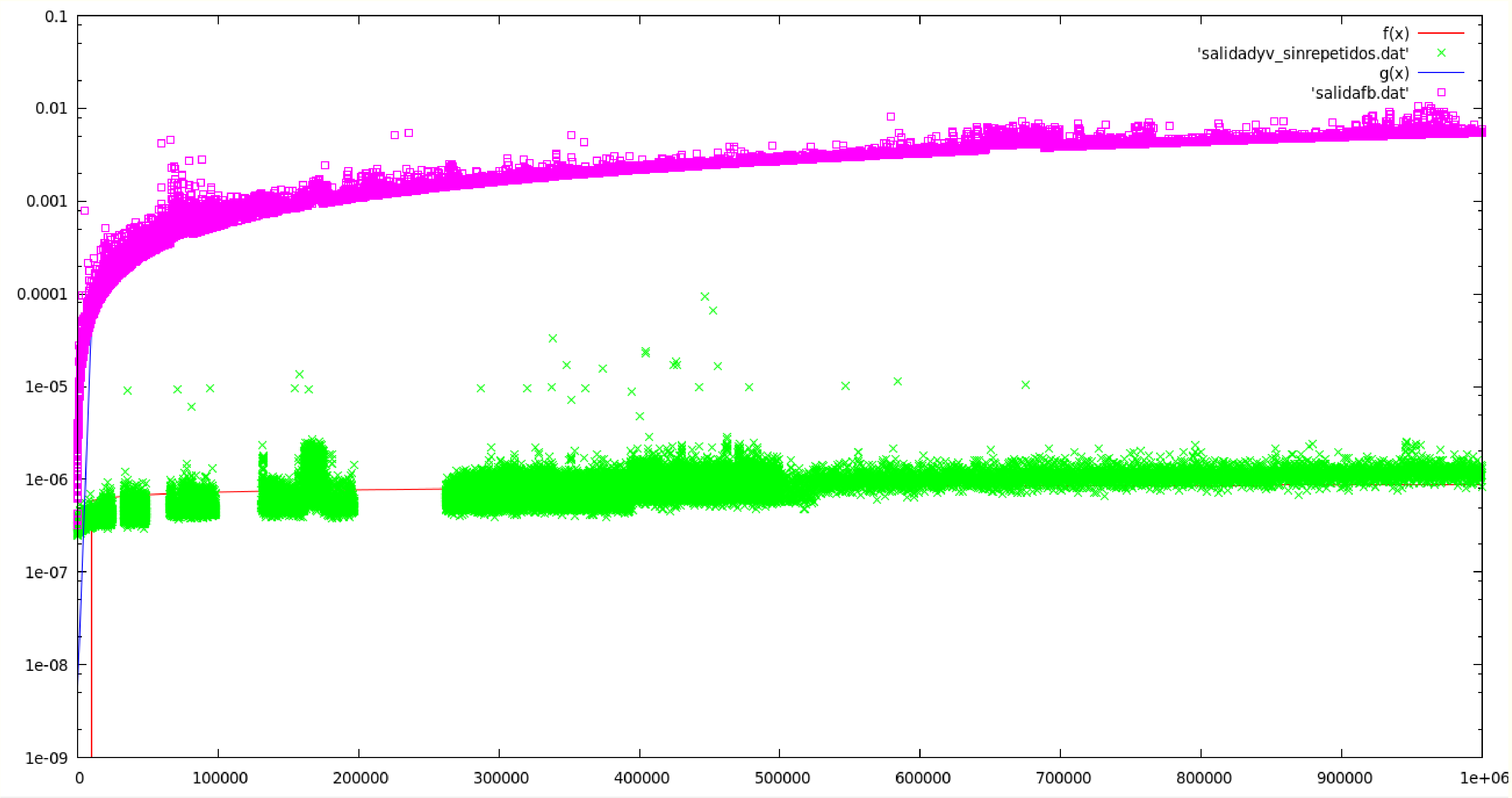


Gráficas comparativas.



Solución al problema con un ejemplo.

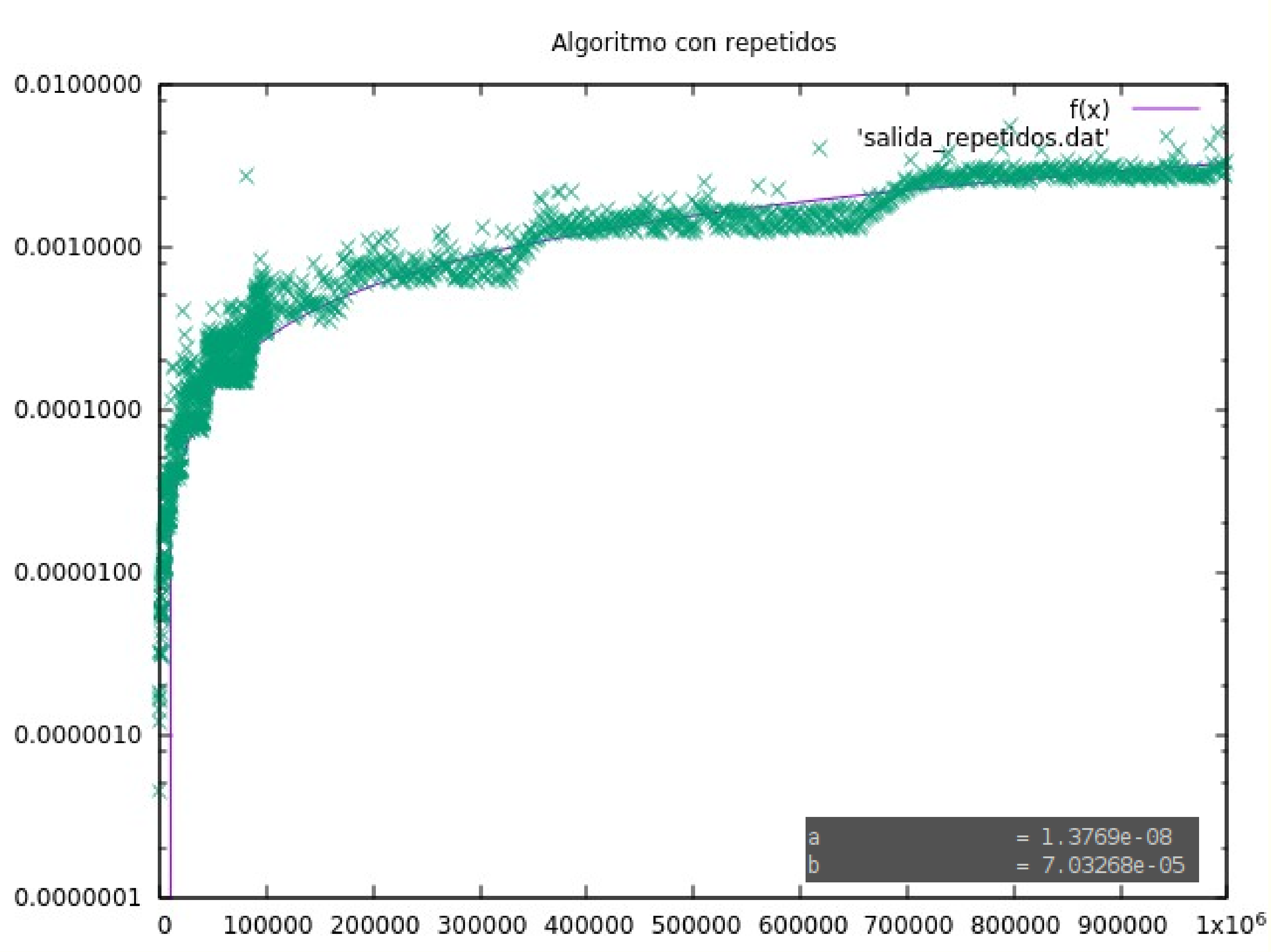
Gráficas comparativas.



Solución al problema con un ejemplo.

Gráficas comparativas (con repetidos).

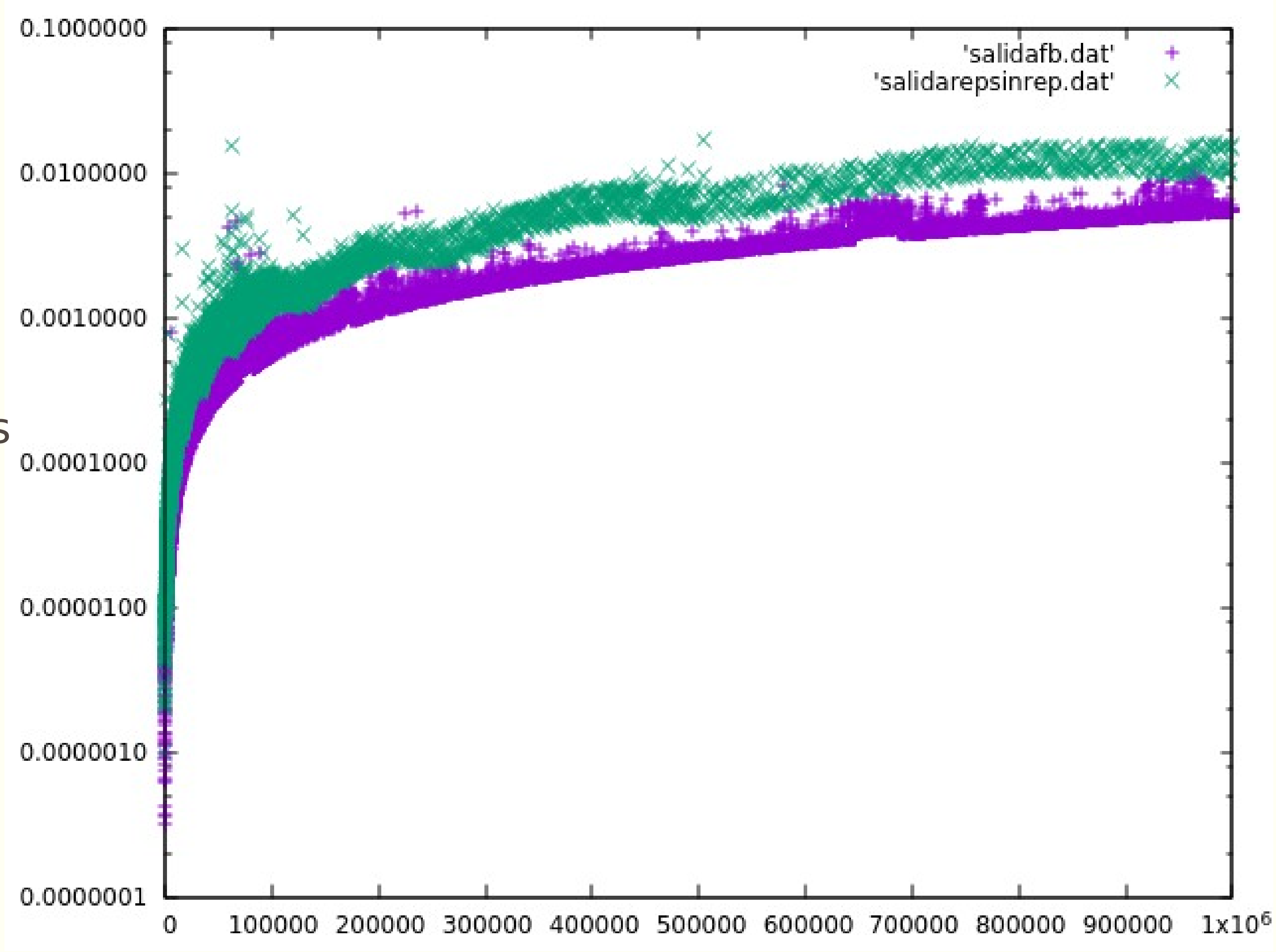
Grafica de algoritmo **Divide y Vencerás**; O(n):



Solución al problema con un ejemplo.

Gráficas comparativas (alg. Rep) (sin repetidos).

Usando el algoritmo
De elementos
repetidos, probamos
con un vector sin
Repetidos.



Solución al problema con un ejemplo.

Gráficas comparativas (con repetidos).