ELIMINAR ELEMENTOS REPETIDOS

Ejercicio 2. Relación de problemas 1.

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

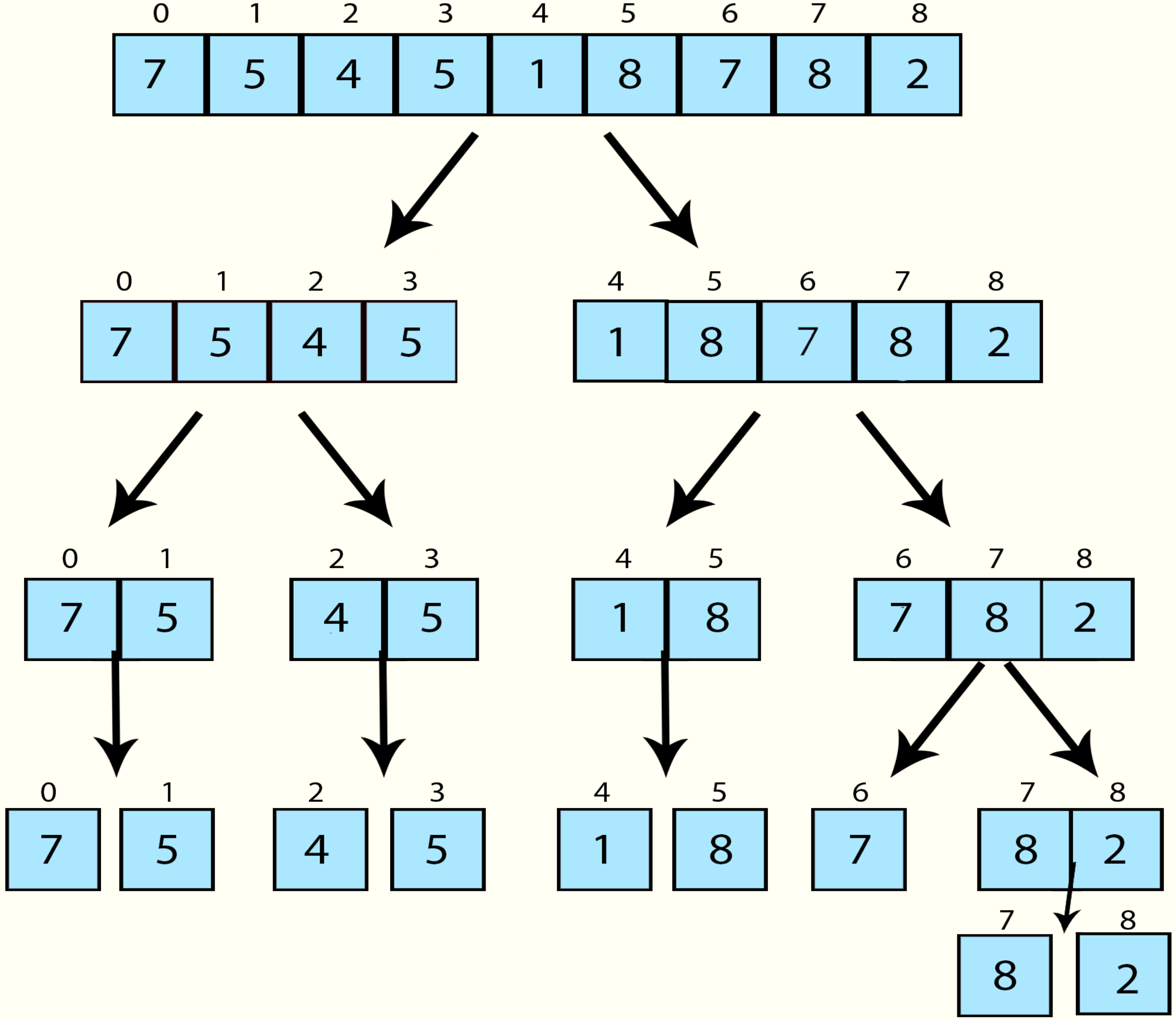
Descripción.

- Este problema consiste en eliminar los elementos repetidos dentro de un vector dejando una sola repetición por elemento.
- El algoritmo básico va comparando cada elemento del vector con los elementos del vector que le siguen y comparando si hay alguno que sea igual que él, y en caso de no existir, lo añade al vector resultado. Por lo tanto tenemos que este algoritmo tendrá un orden $O(n^2)$.
- Nuestra **solución** permite pasar del orden $O(n^2)$ a $O(n \log(n))$ mejorando las prestaciones y eficiencia del calculo que tenemos que realizar, utilizando como algoritmo **Divide y Vencerás**.

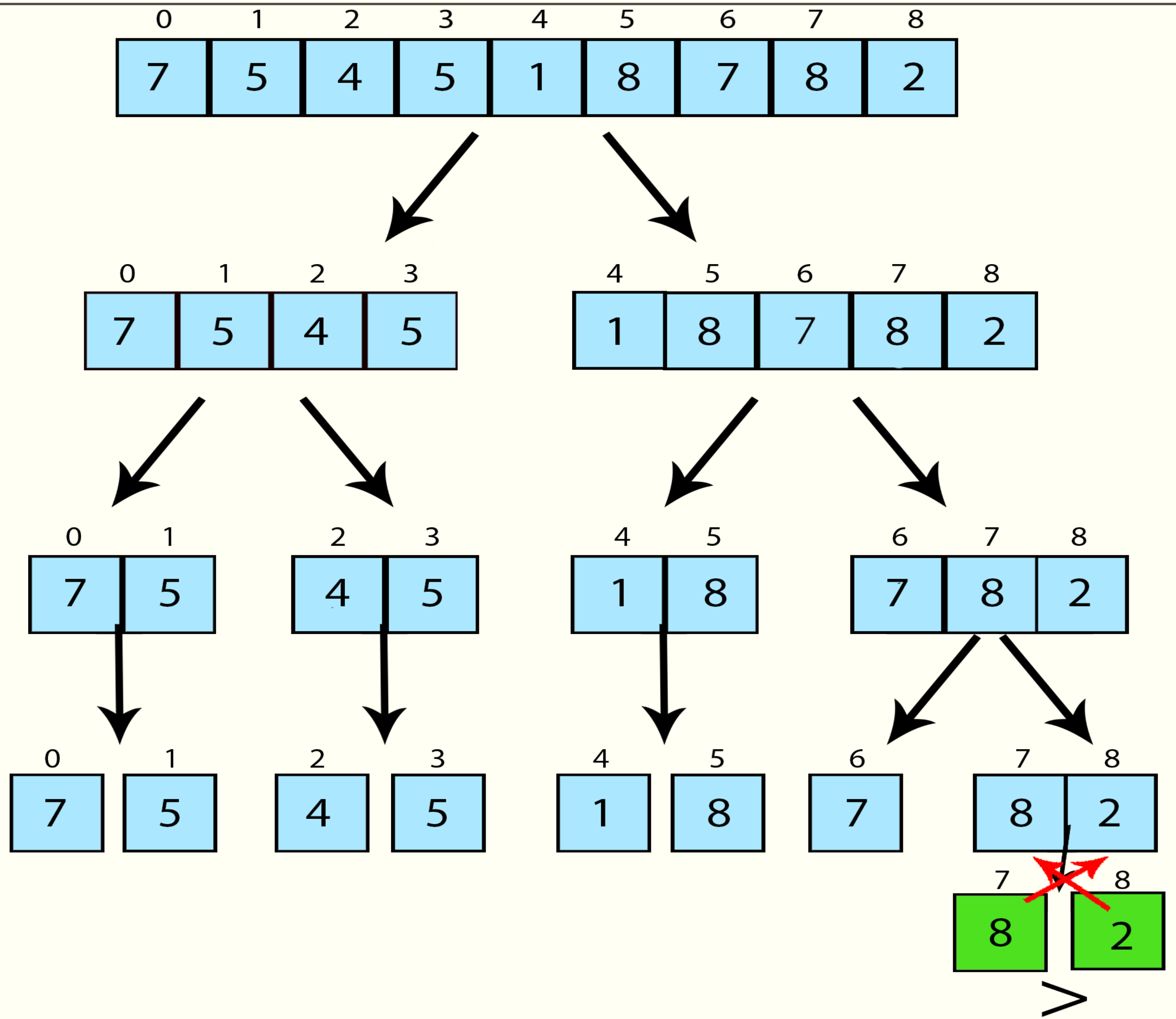
Algoritmo “fuerza bruta”.

```
void eliminar_repetidos( vector<int> &vec ){
  vector<int> sinrep;
  bool encontrado = false;

  for( unsigned int i=0; i<vec.size(); i++ ){
    for( unsigned int j=i+1; j<vec.size(); j++ ){
      if( vec[i] == vec[j] ){
        encontrado = true;
      }
    }
    if( !encontrado ) sinrep.push_back(vec[i]);
    encontrado = false;
  }

  vec.clear();
  for( unsigned int i=0; i<sinrep.size(); i++ ){
    vec.push_back(sinrep[i]);
  }

}
```

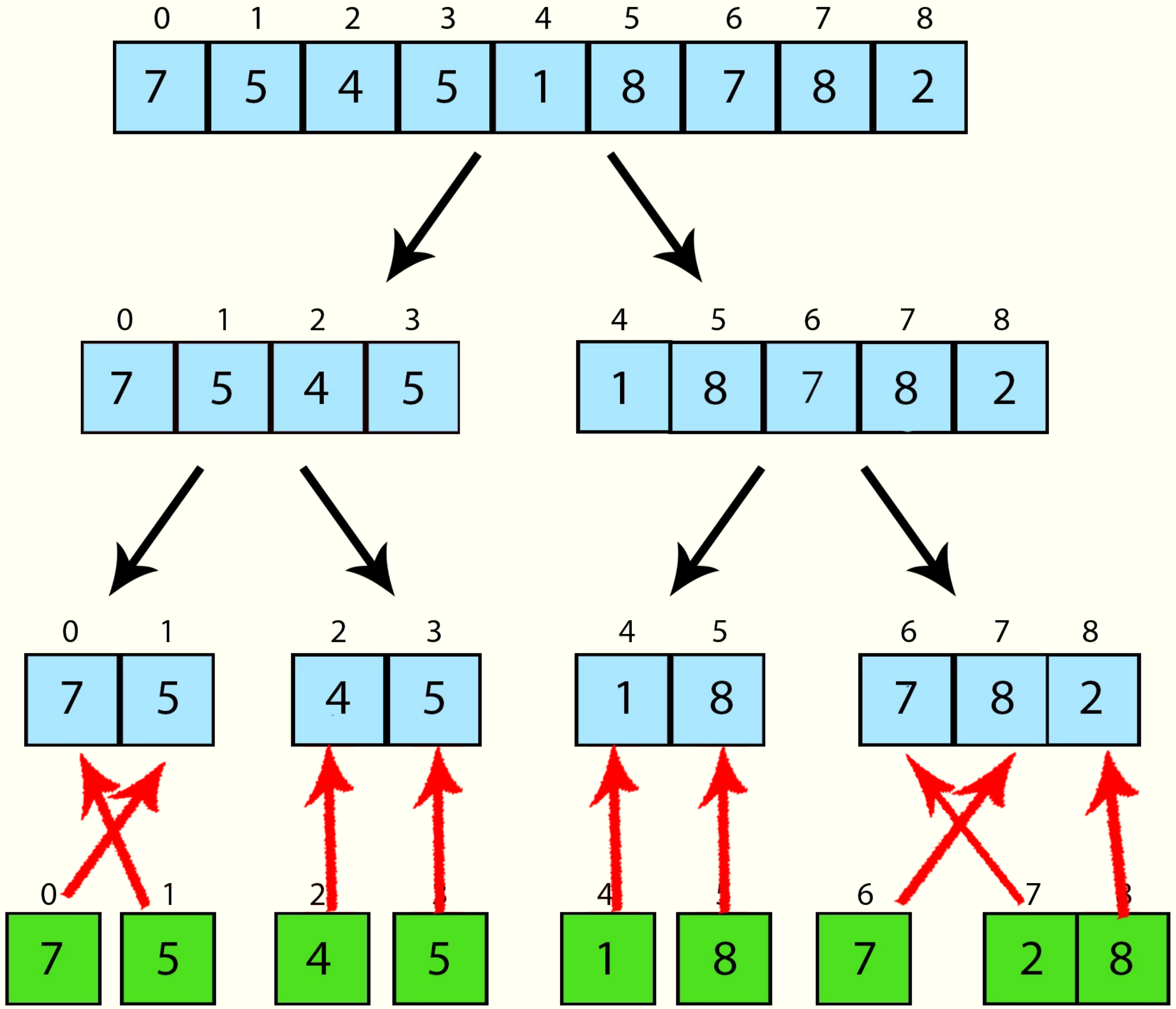
Solución al problema.



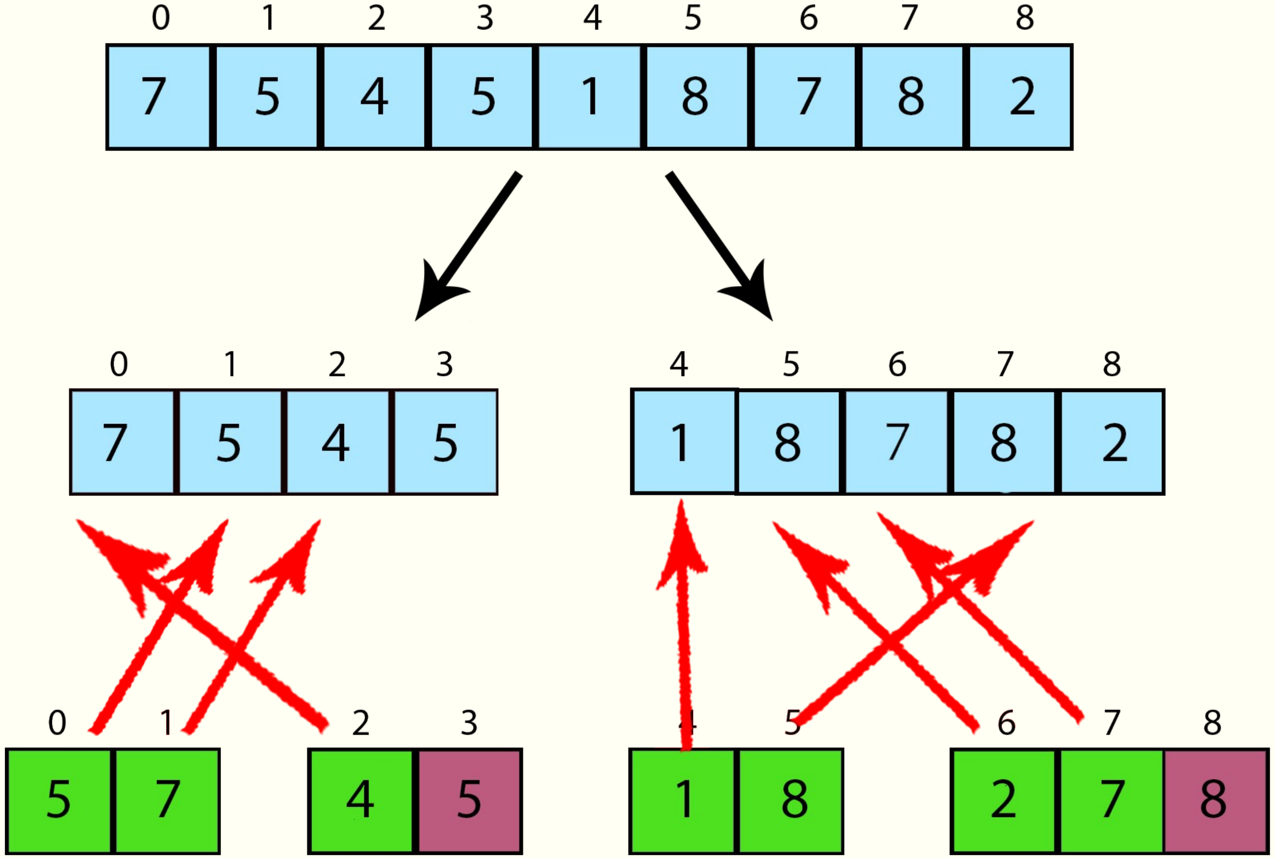
Solución al problema.



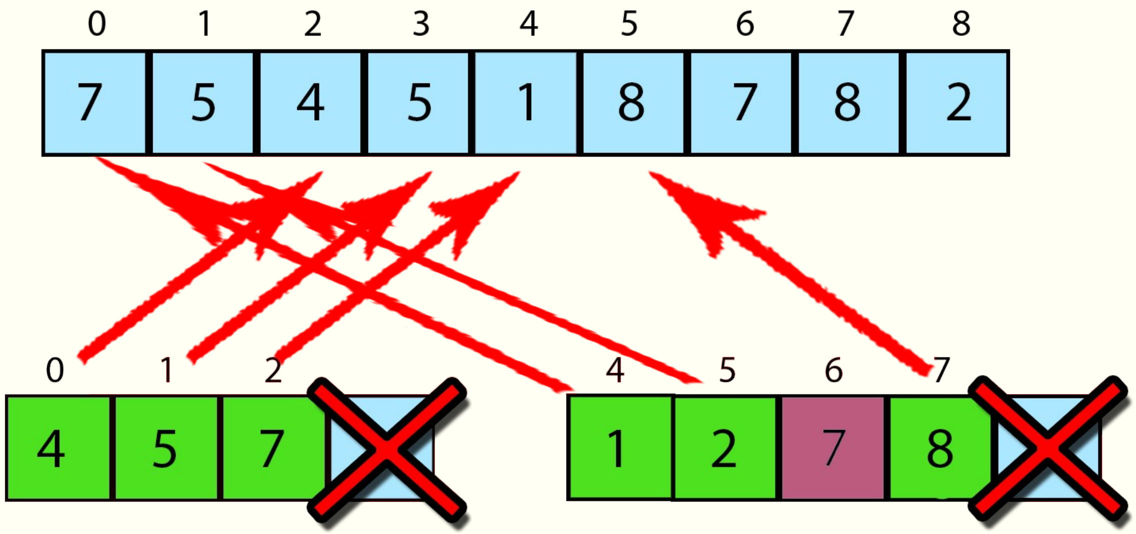
Solución al problema.



Solución al problema.

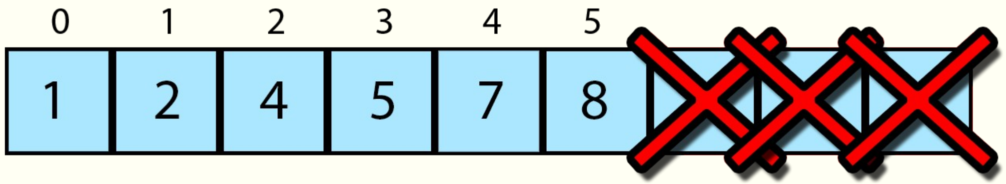


Solución al problema.



Solución al problema con un ejemplo.

Solución al problema.



Solución al problema con un ejemplo.

Algoritmo Divide y Vencerás.

```
void elimRepetidos(vector<int> &v)
{
    vector<int> vector1;
    vector<int> vector2;
    int n1, n2,i,j;

    if (v.size() > 1)
    {
        if (v.size()%2 == 0)
            n1=n2=(int) v.size() / 2;
        else
        {
            n1=(int) v.size() / 2;
            n2=n1+1;
        }
        for(i=0;i<n1;i++)
            vector1.push_back(v[i]);
        for(j=0;j<n2;i++,j++)
            vector2.push_back(v[i]);
        v.clear();
        elimRepetidos(vector1);
        elimRepetidos(vector2);
        combinar(vector1, vector2, v);
    }
}
```

```
void combinar(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.

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

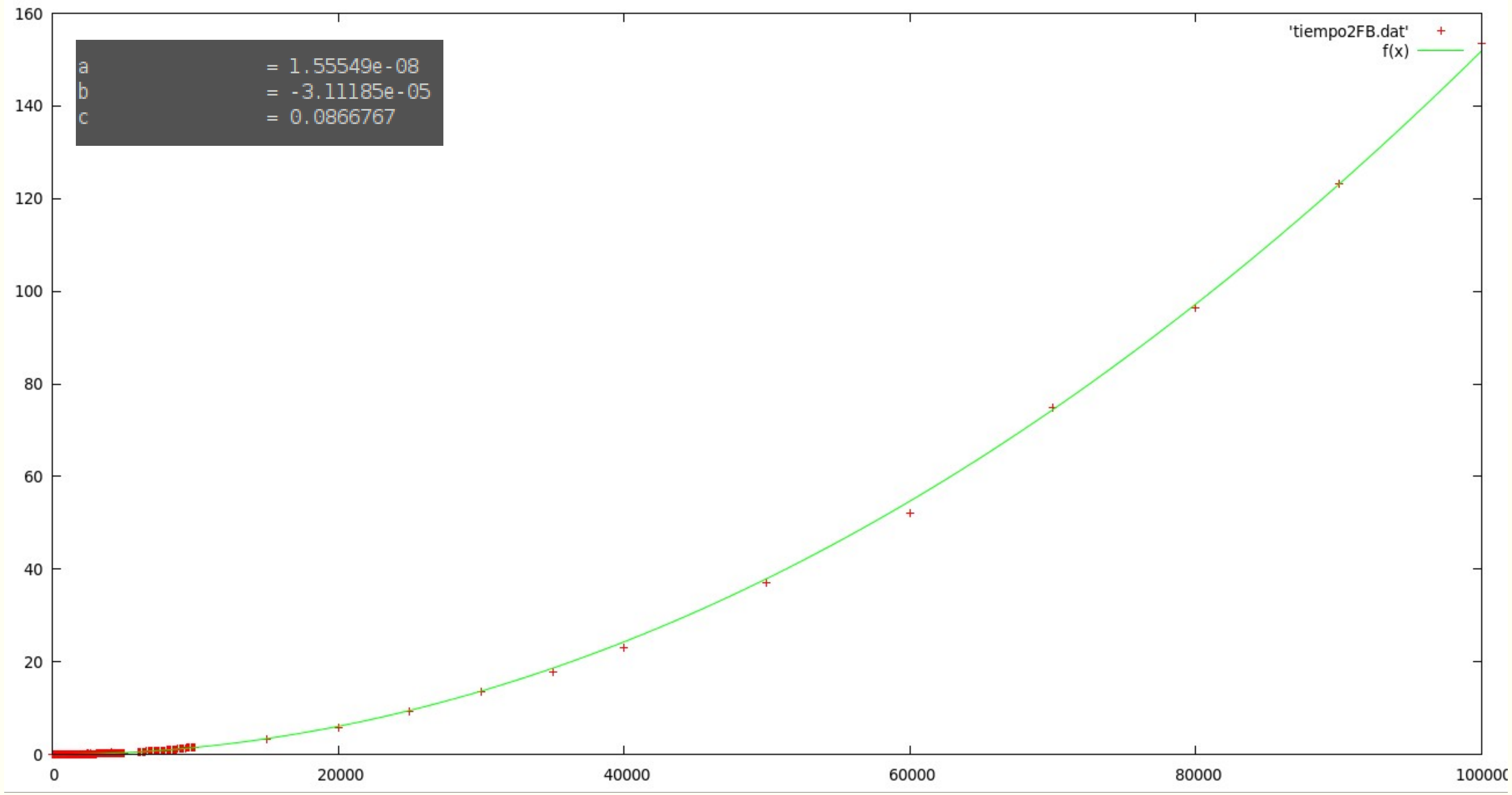
Como $2^k = n$, $k = log_2(n)$

La fórmula finalmente quedaría: $T(n) = nT(1) + c_2nlog_2(n)$

Podemos tomar $T(1) = c_1$ Por lo que la fórmula quedaría $T(n) = c_1n + c_2nlog_2(n)$, por lo que la eficiencia sería $O(nlog(n))$.

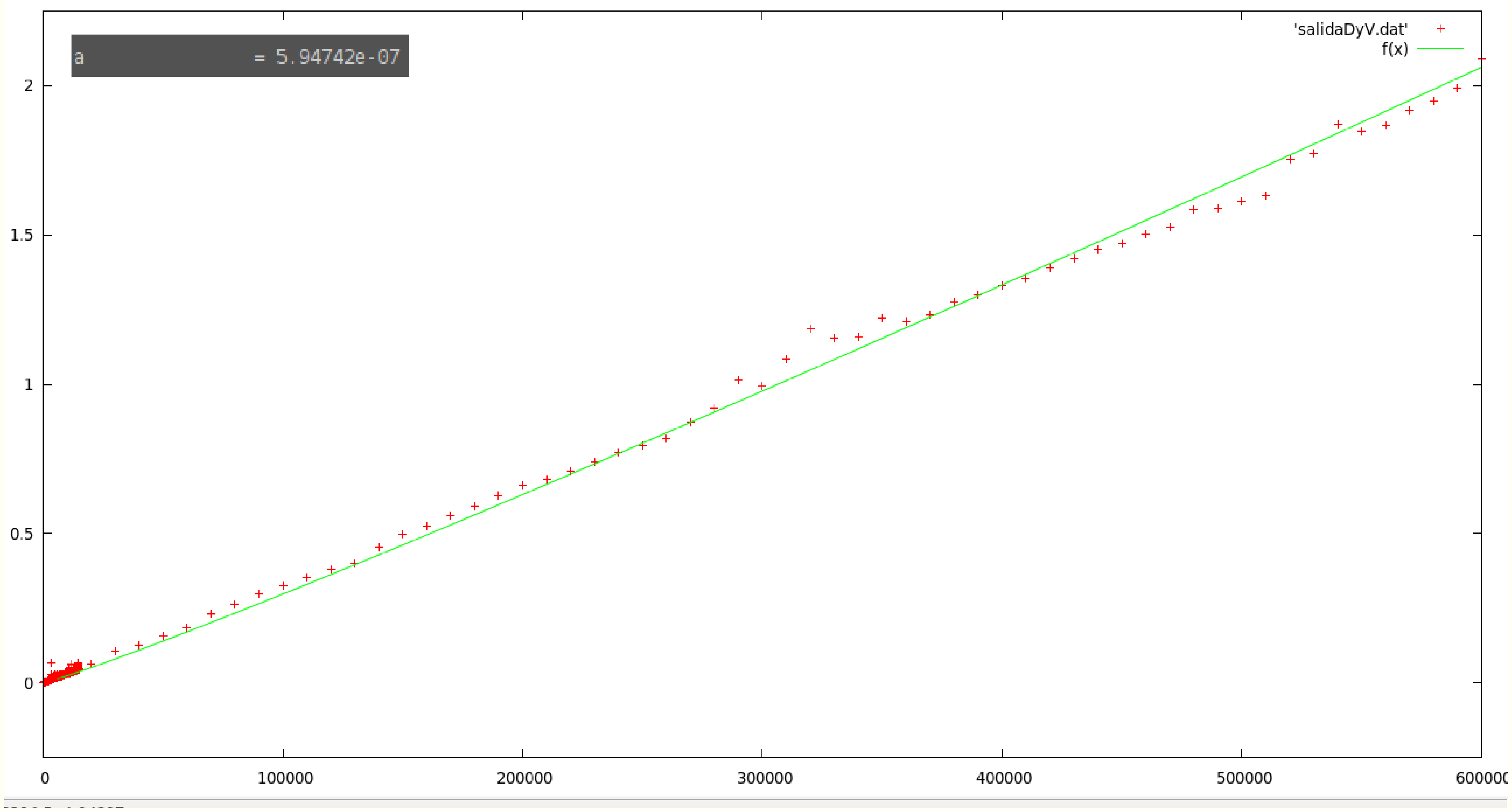
Gráficas comparativas.

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

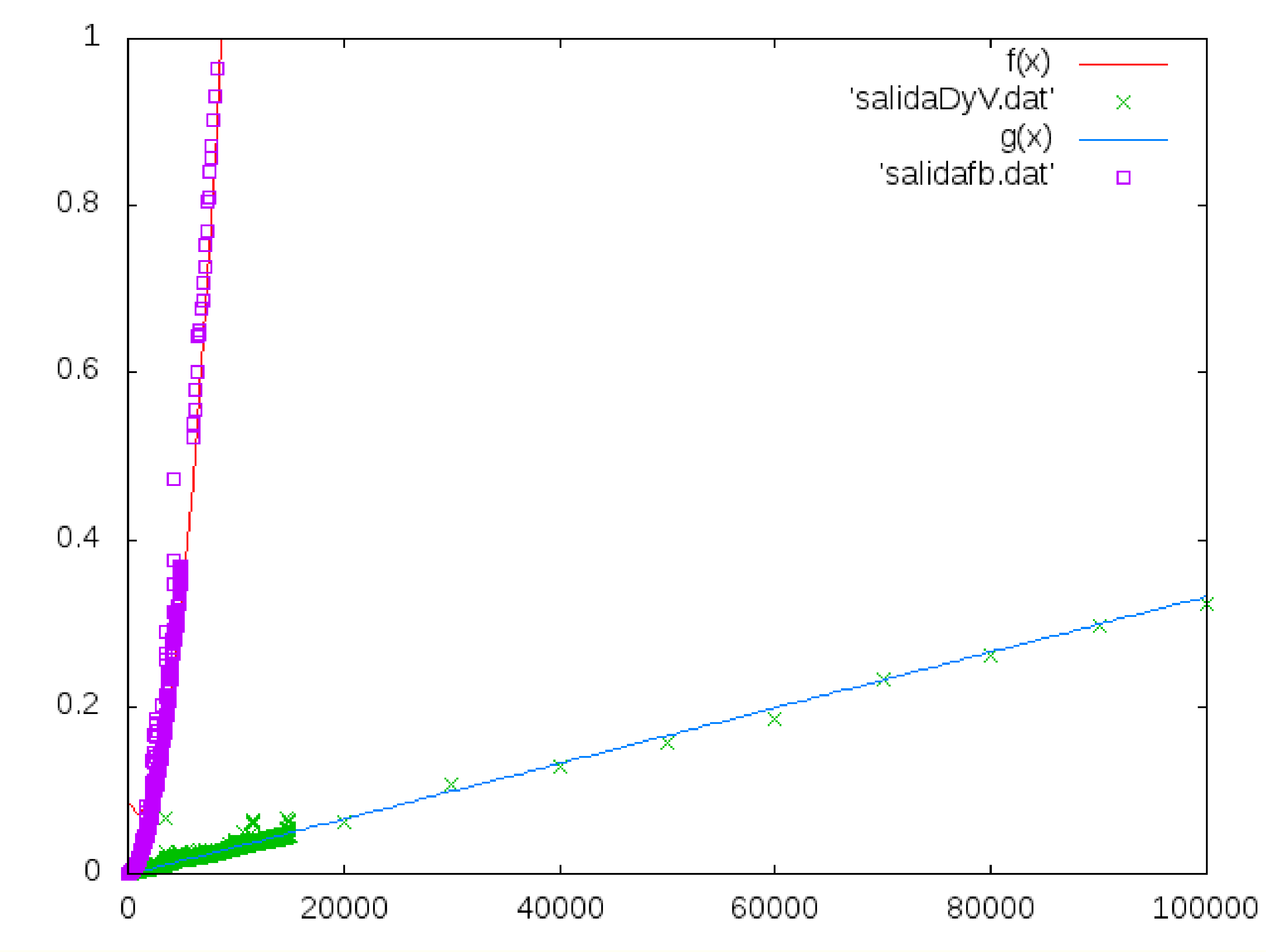


Gráficas comparativas.

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



Gráficas comparativas.



Gráficas comparativas.