ELIMINAR ELEMENTOS REPETIDOS

Algoritmos Divide y Vencerás

¡HOLA!

Somos el Grupo 3

Gregorio Carvajal Expósito

Gema Correa Fernández

Jonathan Fernández Mertanen

Eila Gómez Hidalgo

Elías Méndez García

Alex Enrique Tipán Párraga

EJERCICIO

Dado un vector de *n* elementos, de los cuales algunos pueden estar duplicados, obtener otro vector donde todos los elementos duplicados hayan sido eliminados.

1.

CARACTERÍSTICAS DEL PC

Máquina usada para realizar las medidas de tiempo

PC
CARACTERÍSTICAS

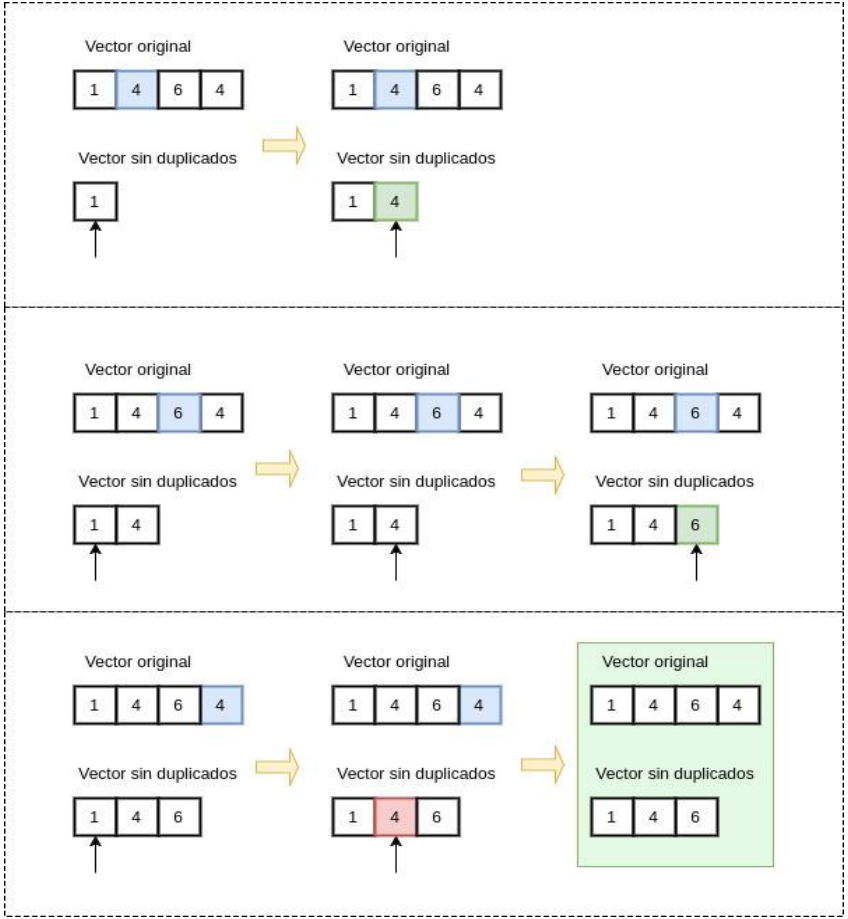
- **Procesador:** Intel(R) Core(TM) i7-4720HQ.
- **Núcleos y velocidad:** 8 Núcleos a 2.60GHz.
- **Cachés:** Caché L1 32K | Caché L2 256K | Caché L3 6144K.
- **Memoria RAM:** 16GB a 1600 MHz.
- **Sistema Operativo:** Arch Linux (rolling release).
- **Versión de kernel:** 4.10.6-1-ARCH
- **Versión de compilador:** g++ (GCC) 6.3.1 201.70306.
- **Opciones de compilación:** -O3 -std=c++11

2.

ALGORITMOS DISEÑADOS

Explicación

ALGORITMO “SENCILLO”



ALGORITMO “SENCILLO”

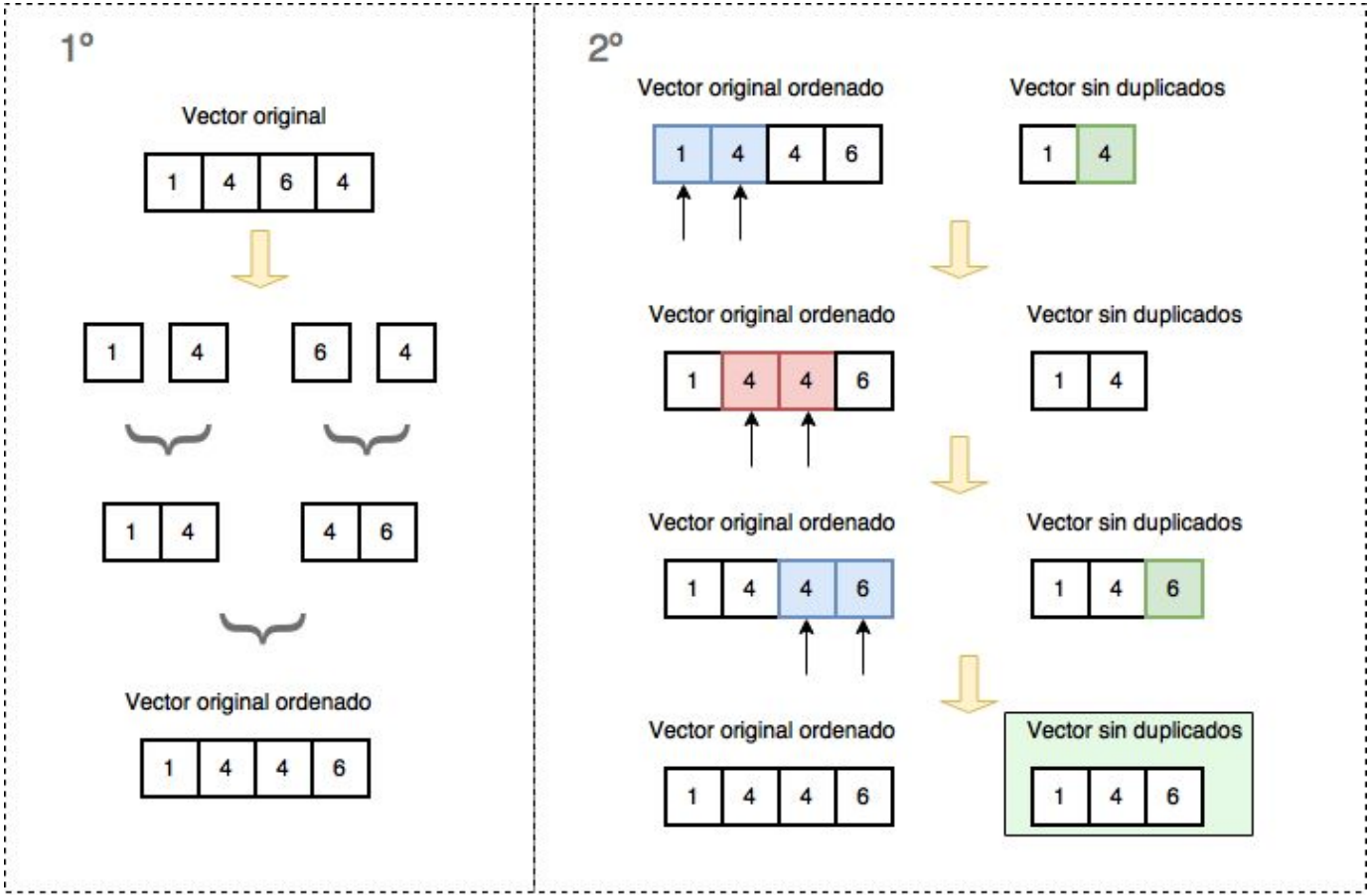
```
inline static Vector* elimina_repetidos(const Vector &vec){

      int elem, n_elem_max = vec.n_elem;
      bool encontrado = false;

      Vector* sin_repetidos = new Vector;
      sin_repetidos->v = new int[n_elem_max];

      for (int i = 0; i < n_elem_max; ++i) {
            elem = vec.v[i];
            for (int j = 0; j < sin_repetidos->n_elem && !encontrado; ++j) {
                  if (elem == sin_repetidos->v[j]) encontrado = true;
            }
            if (!encontrado) {
                  sin_repetidos->v[sin_repetidos->n_elem] = elem;
                  ++sin_repetidos->n_elem;
            } else
                  encontrado = false;
      }
      return sin_repetidos;
}
```

ALGORITMO “DIVIDE Y VENCERÁS”



ALGORITMO “DIVIDE Y VENCERÁS”

```
mergesort(copia, n_elem_max);

Vector* sin_repetidos = new Vector;
sin_repetidos->v = new int[n_elem_max];

sin_repetidos->v[0] = copia[0];
++sin_repetidos->n_elem;

for (int i = 0; i < n_elem_max - 1; ++i)
{
      elem = copia[i];
      sig_elem = copia[i + 1];

      if (elem != sig_elem)
      {
         sin_repetidos->v[sin_repetidos->n_elem] = sig_elem;
         ++sin_repetidos->n_elem;
      }
}
```

3.

EFICIENCIA TEÓRICA

¿Cuál es mejor teóricamente?

ALGORITMO “SENCILLO”

```
[...]

for (int i = 0; i < n_elem_max; ++i) {

  [...]

  for (int j=0; j<sin_repetidos->n_elem && !encontrado; ++j)

     [...]

  if (!encontrado) { [...] } else { [...] }
}
```

$$\sum_{i=0}^{n-1} i = (n-1)\cdot\frac{n}{2} = \frac{n^2-n}{2} \Rightarrow O(n^2)$$

$$\sum_{j=0}^{i-1} 1 = i$$

$$O(1)$$

Por tanto, la eficiencia es $O(n^2)$

ALGORITMO “DIVIDE Y VENCERÁS”

```
mergesort(copia, n_elem_max);

[...]

for (int i = 0; i < n_elem_max - 1; ++i)
{
     [...]

     if (elem != sig_elem)
     {
        [...]
     }
}
```

$O(n \cdot log(n))$

$\sum_{i=0}^{n-2} 1 = n-1 \Rightarrow O(n)$

$O(1)$

Por tanto, la eficiencia es $O(n \cdot log(n))$

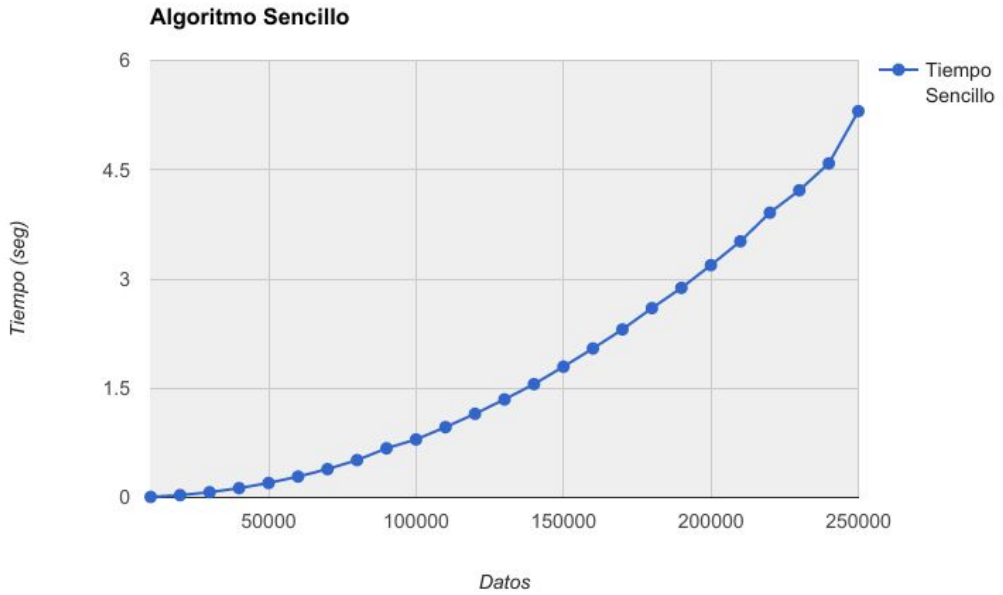
4.

EFICIENCIA EMPÍRICA

Midiendo tiempos de ejecución

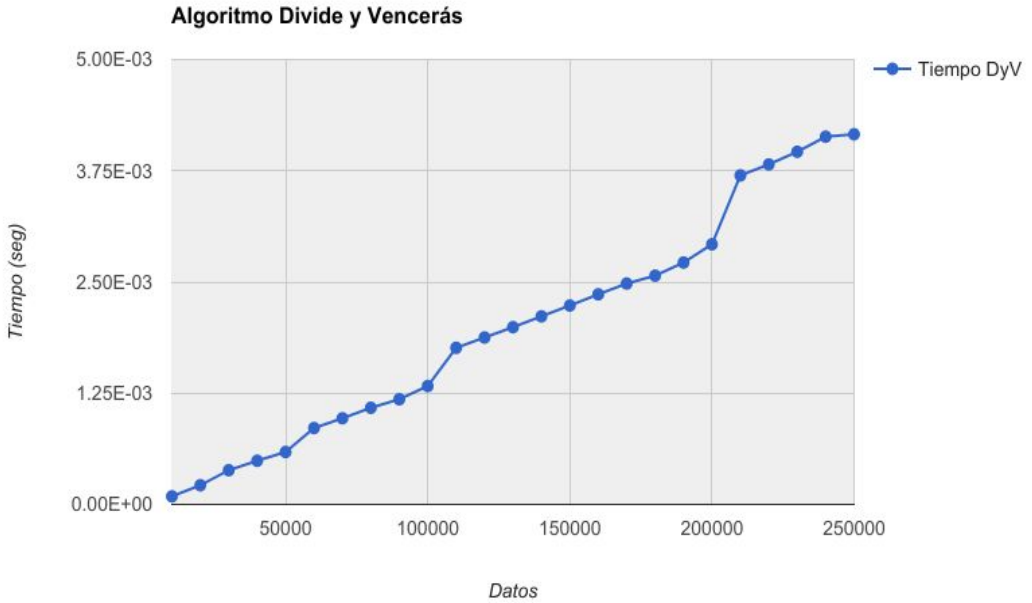
ALGORITMO “SENCILLO”

Nº de datos	Tiempo (seg)
10000	0.00813441
20000	0.032267
30000	0.0722235
40000	0.128482
50000	0.199231
60000	0.287266
. . .	. . .
200000	3.18999
230000	3.51723
220000	3.90999
230000	4.23788
240000	4.58698
250000	5.30422



ALGORITMO “DIVIDE Y VENCERÁS”

Nº de datos	Tiempo (seg)
10000	9.06E-05
20000	0.000235461
30000	0.000384402
40000	0.000490725
50000	0.000589327
60000	0.000858675
. . .	. . .
200000	0.00292369
230000	0.00369807
220000	0.00382324
230000	0.00396376
240000	0.0041334
250000	0.00416028



5.

EFICIENCIA HíBRIDA

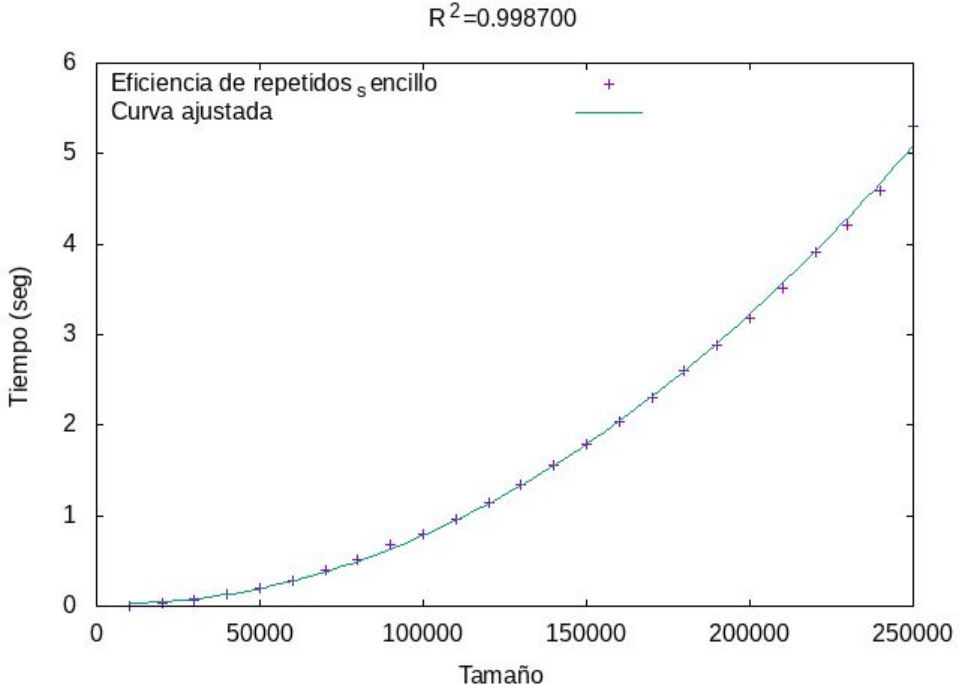
Comprobando los datos obtenidos

ALGORITMO “SENCILLO”

“

Ecuación: $f(x) = a0*x^2+a1*x +a2$

Constantes: a0 = 8.51193e-11
a1 = -1.07647e-06
a2 = 0.0378072

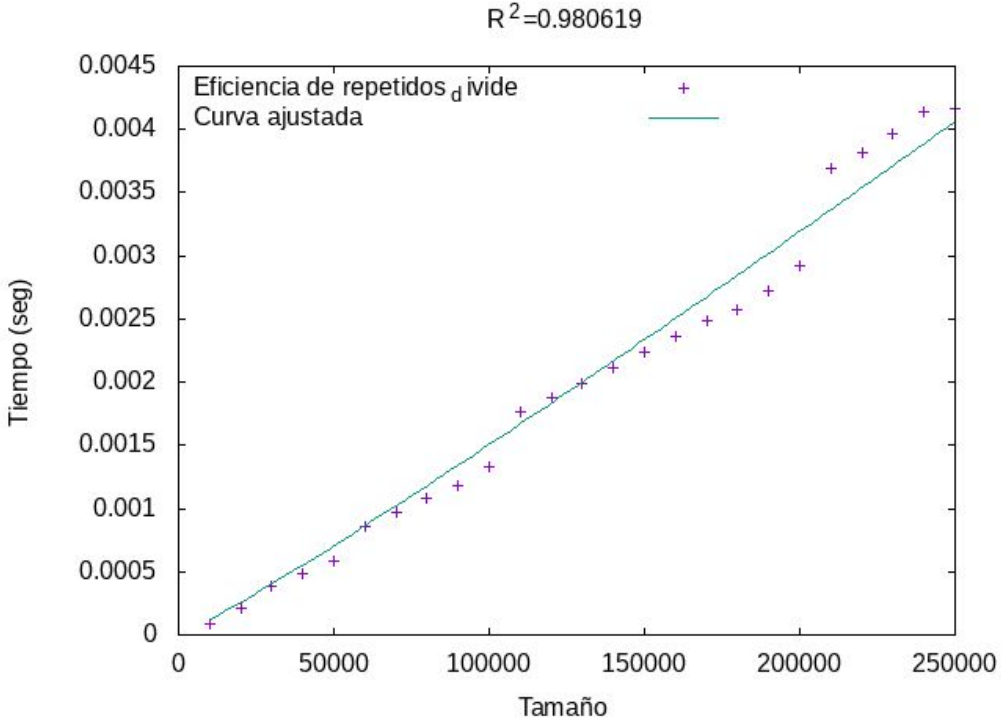


ALGORITMO “DIVIDE Y VENCERÁS”

“

Ecuación: f(x) = a0*x*log10(x)

Constantes: a0 = 3.01119e-09



6.

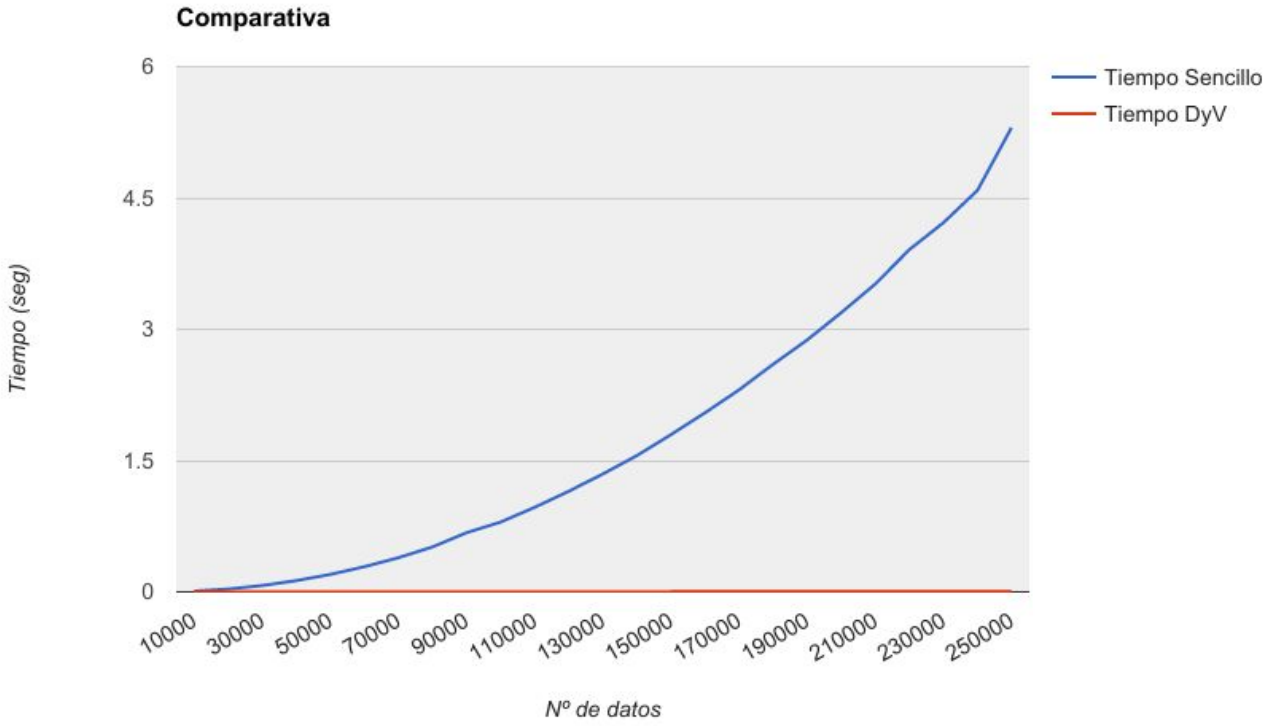
COMPARATIVA

¿Cuál es más rápido?

TABLAS COMPARATIVAS

Nº de datos	Algoritmo Sencillo	Algoritmo DyV
10000	0.00813441	9.06E-05
20000	0.032267	0.000235461
30000	0.0722235	0.000384402
40000	0.128482	0.000490725
50000	0.199231	0.000589327
60000	0.287266	0.000858675
. . .	. . .	. . .
200000	3.18999	0.00292369
230000	3.51723	0.00369807
220000	3.90999	0.00382324
230000	4.23788	0.00396376
240000	4.58698	0.0041334
250000	5.30422	0.00416028

GRÁFICA COMPARATIVA



¡GRACIAS A TODOS!

¿Preguntas?