Práctica 2: Algoritmos Divide y Vencerás

Daniel Bolaños Martínez, José María Borrás Serrano,
Santiago De Diego De Diego, Fernando De la Hoz Moreno

ETSIIT

Introducción

Nos ha tocado resolver el ejercicio serie unimodal de números.

Para ello, hemos diseñado un algoritmo basado en "divide y vencerás" el cual tiene como objetivo encontrar el valor máximo de una serie unimodal. El orden de eficiencia de este algoritmo es $O(log(n))$ y lo hemos comparado con el algoritmo trivial para este problema que es de orden $O(n)$.

Desarrollo de la Práctica

Para la comparación hemos obtenido unas tablas en las que se muestran el tiempo de ejecución según distintos número de elementos en los vectores, hemos representado los datos en una gráfica y hemos ajustado estos datos a la función obtenida por la eficiencia teórica por el ajuste de mínimos cuadrados.

Código Divide y Vencerás

Listing 1: Función unimodal DyV

```
int unimodal(vector<int> v)
{
        bool fin=false;
        int maximo=v.size()-1;
        int indice=maximo/2;
        int minimo;

        while(!fin)
        {
        if(v.at(indice-1)<v.at(indice))
           if(v.at(indice+1)<v.at(indice))
                           fin=true;
                   else
                   {
                                   minimo=indice;
                                   indice=indice+((maximo-indice)/2);
                           }
                else
                {
                           maximo=indice;
                           indice=minimo+((indice-minimo)/2);
                }
        }
        return indice;
}
```

Listing 2: Función main DyV

```
int main(int argc, char* argv[])
{
        vector<int> array;
        int valor = -1;
        double suma=0;

        int v_size = atoi(argv[1]);
        array.resize(v_size);

        for(int i=0; i<100; ++i)
        {
                int p = 1 + rand() %(v_size-2);
                array.at(p) = v_size-1;
                for (int i=0; i<p; i++)
                        array.at(i)=i;
                for (int i=p+1; i<v_size; i++)
                        array.at(i)=v_size-1-i+p;

                clock_t tantes;
                clock_t tdespues;
                tantes=clock();
                valor = unimodal(array);
                tdespues=clock();
                suma += (double)(tdespues - tantes) / CLOCKS_PER_SEC;
        }
        cout << v_size <<" "<< suma/100 << endl;
}
```

Código secuencial

Listing 3: Función unimodal Secuencial

```
int unimodal_secuencial(vector<int> v)
{
        bool fin=false;
        int indice=1;

        while(!fin)
        {
        if(v.at(indice+1)<v.at(indice))
                fin=true;
        else
                indice++;
        }

        return indice;
}
```

Listing 4: Función main Secuencial

```
int main(int argc, char* argv[])
{
        vector<int> array;
        int valor = -1;
        double suma=0;

        int v_size = atoi(argv[1]);
        array.resize(v_size);

        for(int i=0; i<100; ++i)
        {
                int p = 1 + rand() %(v_size-2);
                array.at(p) = v_size-1;
                for (int i=0; i<p; i++)
                        array.at(i)=i;
                for (int i=p+1; i<v_size; i++)
                        array.at(i)=v_size-1-i+p;

                clock_t tantes;
                clock_t tdespues;
                tantes=clock();
                valor = unimodal_secuencial(array);
                tdespues=clock();
                suma += (double)(tdespues - tantes) / CLOCKS_PER_SEC;
        }
        cout << v_size <<" "<< suma/100 << endl;
}
```

Tabla Datos DyV

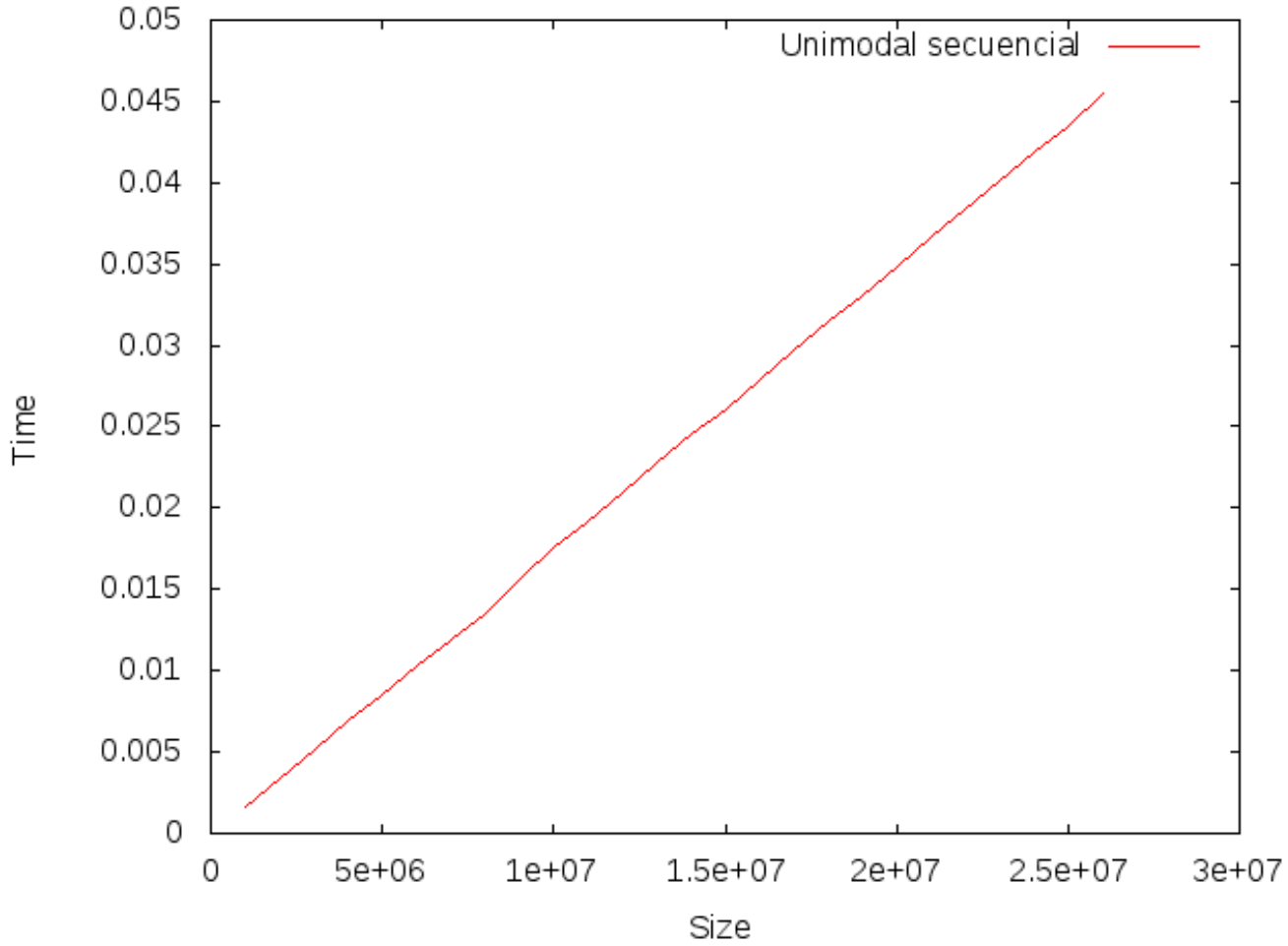
Tamaño Vectores	Tiempo Divide y Vencerás
1048576	7.796e-05
2097152	0.00016308
4194304	0.00038871
8388608	0.00117717
16777216	0.00227126
33554432	0.00456919
67108864	0.00894183
134217728	0.0170173
268435456	0.0335588
536870912	0.0668834

Tabla Datos Secuencial

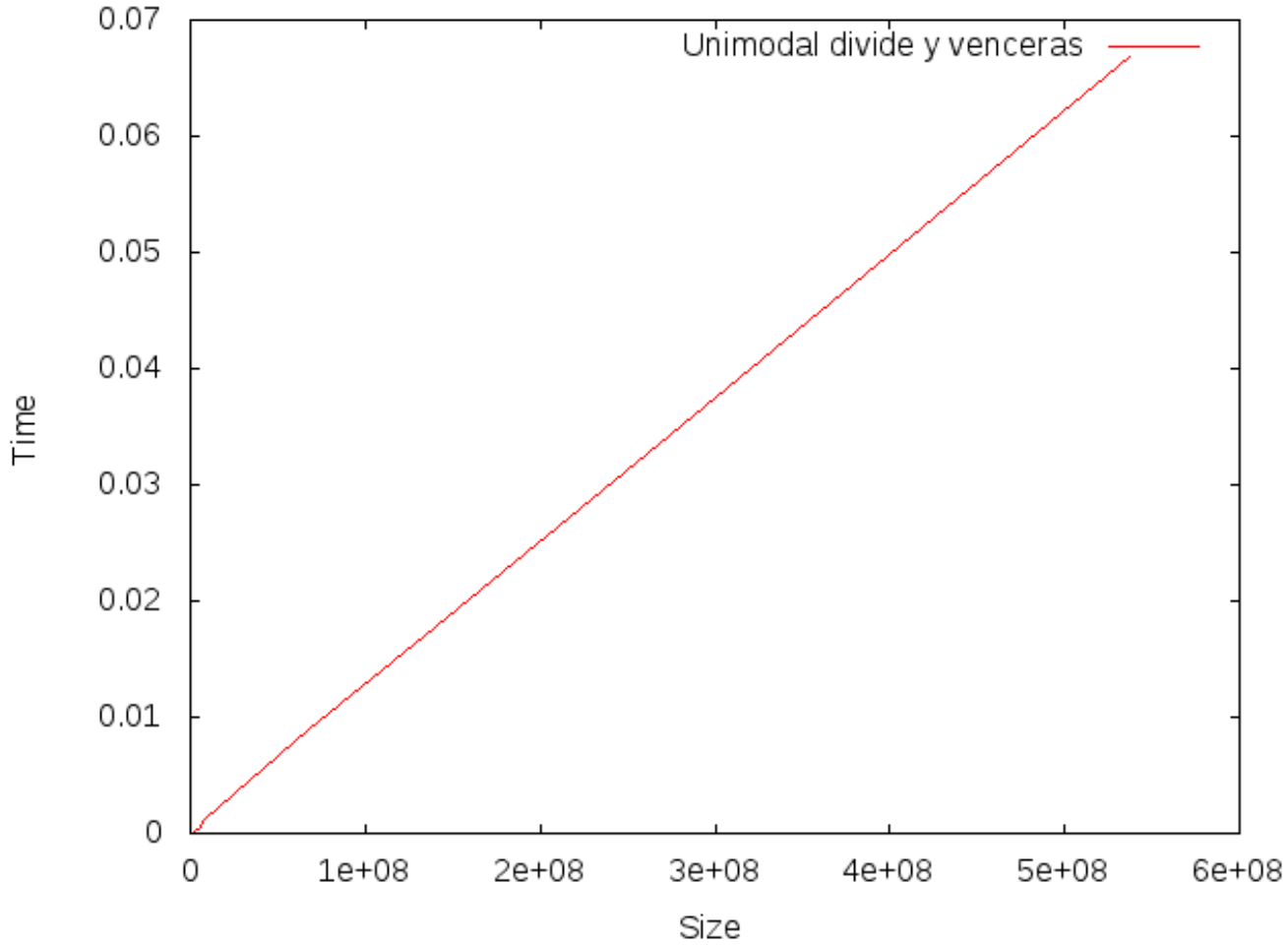
Tamaño Vectores	Tiempo Secuencial
1000000	0.00169148
2000000	0.00341387
3000000	0.00515229
4000000	0.00688878
5000000	0.00583811
6000000	0.0102687
7000000	0.0119547
8000000	0.013579
9000000	0.0157071
10000000	0.017487
11000000	0.0192033
12000000	0.0209426

Tamaño Vectores	Tiempo Secuencial
13000000	0.022794
14000000	0.0245116
15000000	0.0260875
16000000	0.0278383
17000000	0.0296462
18000000	0.0314487
19000000	0.033057
20000000	0.0348266
21000000	0.0367226
22000000	0.0383142
23000000	0.0401301
24000000	0.0418608
25000000	0.0434716
26000000	0.0455227

Eficiencia en el caso secuencial



Eficiencia en el caso Divide y Vencerás



Ajuste híbrido en el caso secuencial

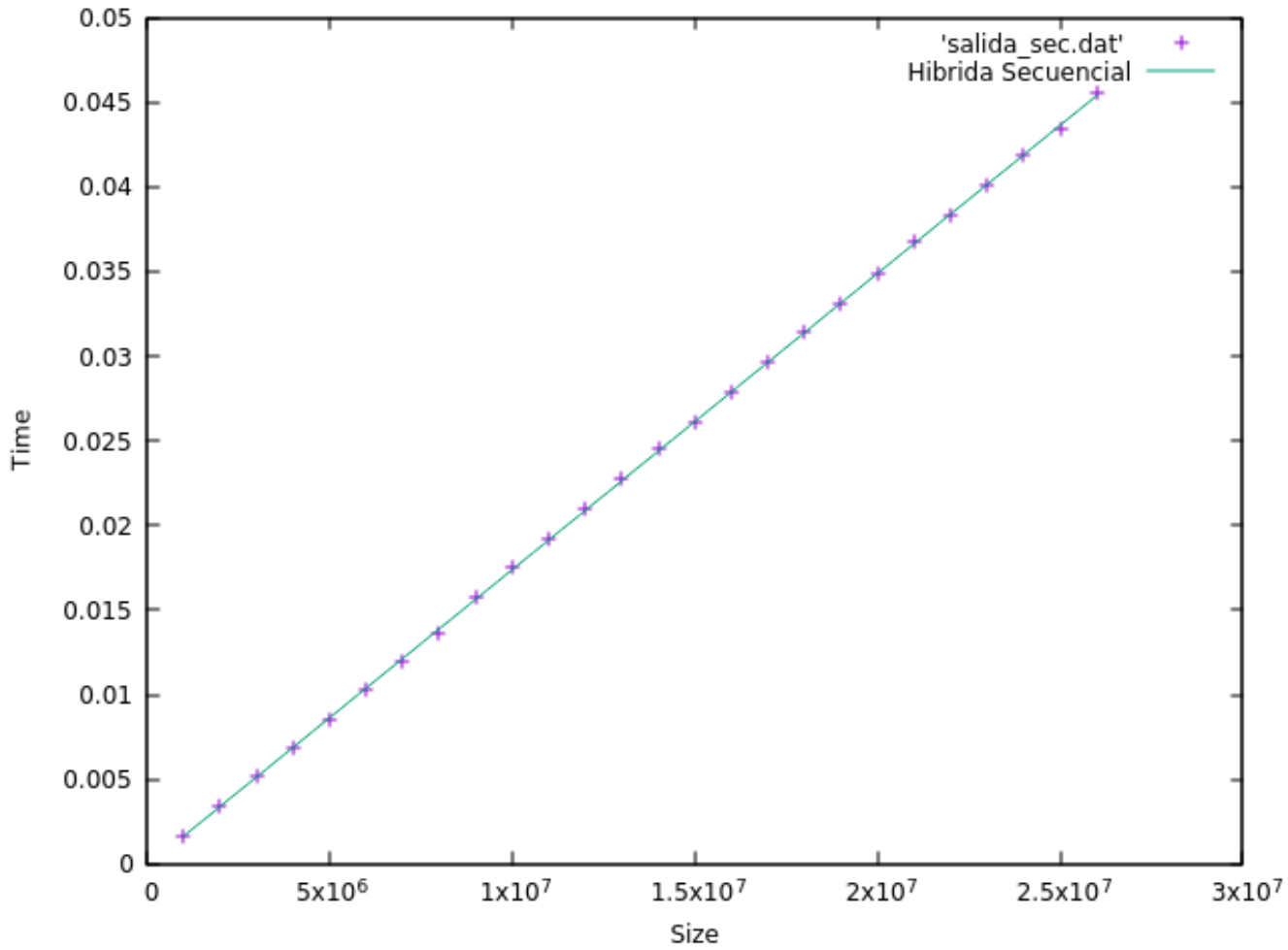


Figura: Ajustada a la función $f(x) = a_0 * x + a_1$

$$f(x) = a_0 * x + a_1$$

$$a_0 = 1{,}75072e - 09$$

$$a_1 = -0{,}000131396$$

Ajuste híbrido en el caso Divide y Vencerás

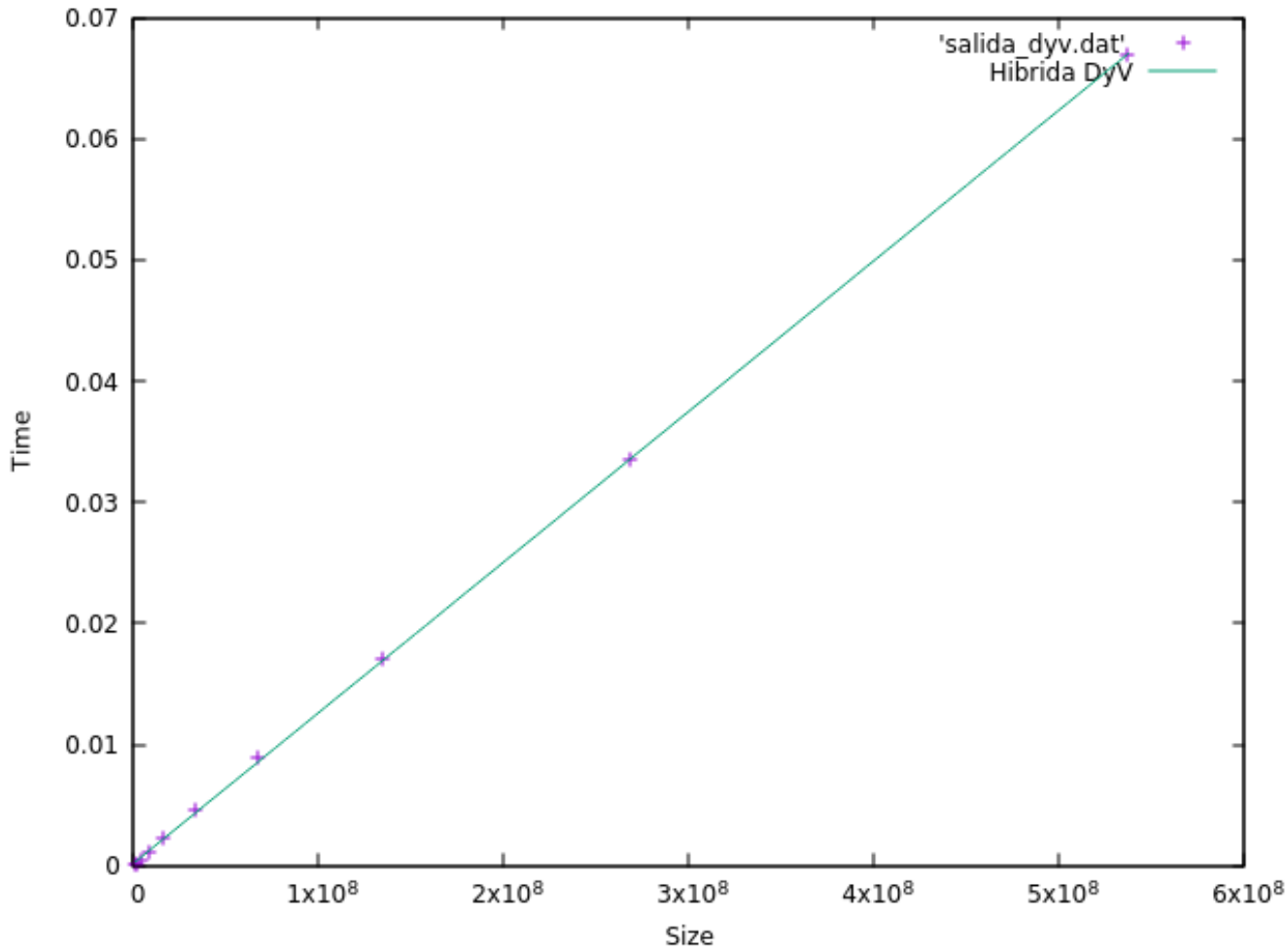


Figura: Ajustada a la función $f(x) = a_0 * log(x) + a_1 * x + a_2$

$$f(x) = a_0 * log(x) + a_1 * x + a_2$$

$$a_0 = 1{,}75072e - 09$$

$$a_1 = 1{,}24488e - 10$$

$$a_2 = 0{,}000151147$$

Correlación

- Unimodal secuencial:
  Coeficiente de correlación en el caso lineal:
  0,999967757
  Coeficiente de correlación en el caso logarítmico:
  0,999967634

- Unimodal Divide y Vencerás:
  Coeficiente de correlación en el caso lineal:
  0,993561274
  Coeficiente de correlación en el caso logarítmico:
  0,99566217

  En el caso secuencial el ajuste lineal es mejor, mientras que en Divide y Vencerás el mejor ajuste es el logarítmico.

Conclusión

Como podemos observar, el mismo problema se puede resolver de forma más rápida y eficiente si empleamos un algoritmo de tipo Divide y Vencerás que uno secuencial. En este caso con

Divide y Vencerás podemos conseguir que la eficiencia del algoritmo pase de ser $O(n)$ a $O(logn)$, por lo que somos capaces de procesar muchos más datos en un tiempo menor. De esta forma se puede concluir que siempre que vayamos a

usar datos lo bastante grandes es mejor realizar el algoritmo mediante Divide y Vencerás que mediante uno secuencial.