

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.

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;
}
```

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;

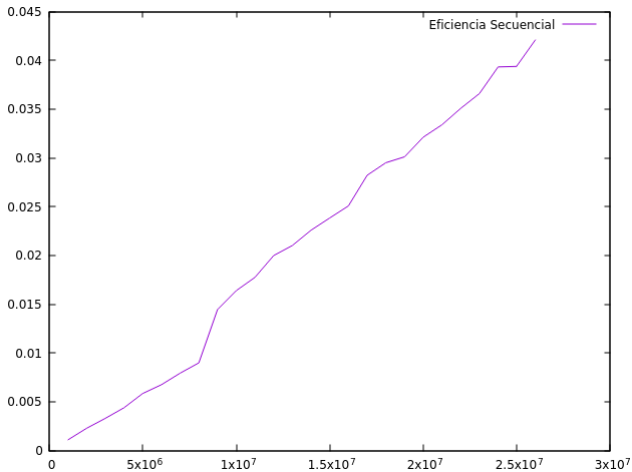
        clock_t tantes;
        clock_t tdespues;
        tantes=clock();
        valor = unimodal_secuencial(array);
        tdespues=clock();
        suma += (double)(tdespues - tantes) / CLOCKS_PER_SEC;
    }
    cout << v_size << " " << suma/100 << endl;
}
```

Tabla Datos

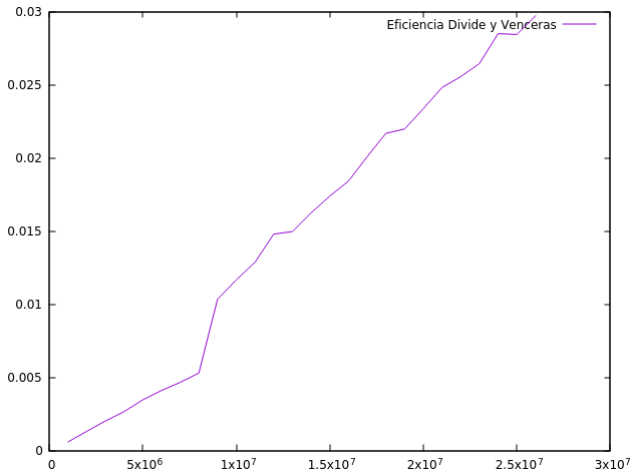
Tamaño Vectores	Tiempo DyV	Tiempo Secuencial
1000000	0.00057548	0.00108062
2000000	0.00130396	0.00227099
3000000	0.00200936	0.00329093
4000000	0.00265309	0.00437678
5000000	0.00345555	0.00583811
6000000	0.00409567	0.00673738
7000000	0.00464896	0.00792554
8000000	0.0052876	0.00897848
9000000	0.0103381	0.0144504
10000000	0.0116623	0.016394
11000000	0.0128614	0.0177504
12000000	0.0147915	0.0199869
13000000	0.014961	0.0210103
14000000	0.0162433	0.0225911

Tamaño Vectores	Tiempo DyV	Tiempo Secuencial
15000000	0.0173869	0.0238454
16000000	0.0184078	0.0250778
17000000	0.0200822	0.0282202
18000000	0.0216763	0.0295068
19000000	0.021972	0.0301068
2000000	0.0233563	0.0321234
2100000	0.0248113	0.0333933
2200000	0.0255466	0.0350716
2300000	0.0264495	0.0365949
2400000	0.0284914	0.039327
2500000	0.0284393	0.0393759
2600000	0.0297109	0.0420735

Eficiencia en el caso secuencial



Eficiencia en el caso Divide y Vencerás



Ajuste híbrido en el caso secuencial

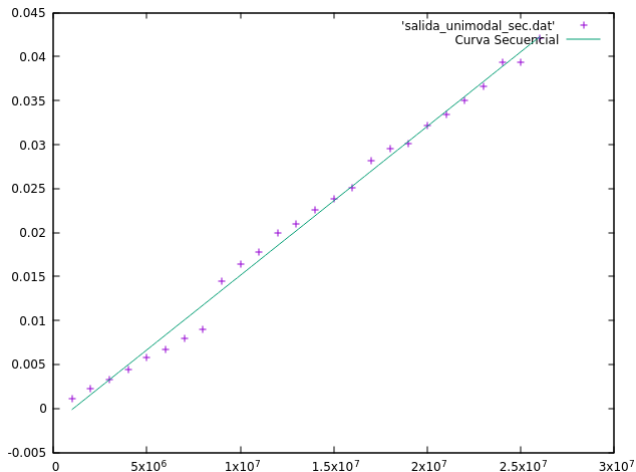


Figura: Ajustada a la función $f(x) = a_0 * x + a_1$

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

$$a_0 = 1,69539e - 09$$

$$a_1 = -0,00183396$$

Ajuste híbrido en el caso Divide y Vencerás

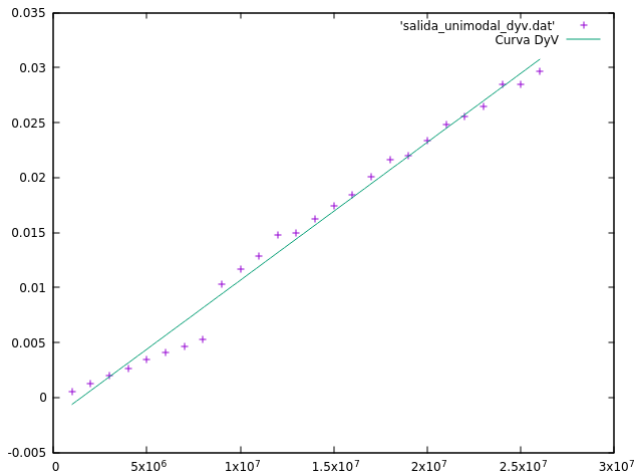


Figura: Ajustada a la función $f(x) = a_0 * \log(x) + a_1 * x + a_2$

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

$$a_0 = 1,69539e - 09$$

$$a_1 = 1,25523e - 09$$

$$a_2 = -0,00189877$$

Conclusión

Como hemos podido observar, los algoritmos de Divide y Vencerás son mucho más eficientes y rápidos que los que podamos pensar de primeras para dar solución a un problema, esto básicamente funciona así gracias a la genial idea del método.

Dividiendo el problema en otros más pequeños y descartando las soluciones que no sean adecuadas, consiguiendo mayor eficiencia y rapidez que realizando un algoritmo de forma Greedy o de la primera forma que se nos ocurriese.