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

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).

Veremos unas tablas en las que se muestran el tiempo de ejecución para distinto número de elementos en los vectores, todo ello complementado con una gráfica.

Además hemos ajustado estos datos a la función obtenida por la eficiencia teórica por el ajuste de mínimos cuadrados.

Código Divide y Vencerás

El algoritmo consiste en tomar el elemento que se encuentra en mitad del vector y comprobar si es un máximo viendo si es mayor que el elemento de la izquierda y menor que el de la derecha.

- Si es así se ha terminado el algoritmo pues ya hemos encontrado el máximo.
- Si no es así vemos si el elemento esta en la zona creciente o decreciente del vector.
  - En el caso de que este en la zona creciente el máximo se situara en la mitad de la derecha del vector.
  - Si se encuentra en la decreciente en la mitad izquierda.

Se vuelve a aplicar el algoritmo sobre la mitad del vector donde se encuentre el máximo y se repite el proceso hasta que se encuentre el máximo.

Como en cada iteración lo que se hace es dividir el vector por la mitad y buscar el máximo en una mitad el número máximo de iteraciones hasta encontrar el máximo es de $log(n)$, siendo n el tamaño del vector.

Como todo las comprobaciones realizadas en cada iteración son $o(1)$ el algoritmo es $o(log(n))$.

```
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;
}
```

```
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

En este caso lo único que se hace es recorrer el vector hasta ver que empieza a ser decreciente.

En el peor caso puede empezar a ser decreciente en el penúltimo elemento por lo que habría que recorrer todo el vector, de manera que la eficiencia de este algoritmo es O(n).

```
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;
}
```

```
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;
}
```

Eficiencia en el caso secuencial

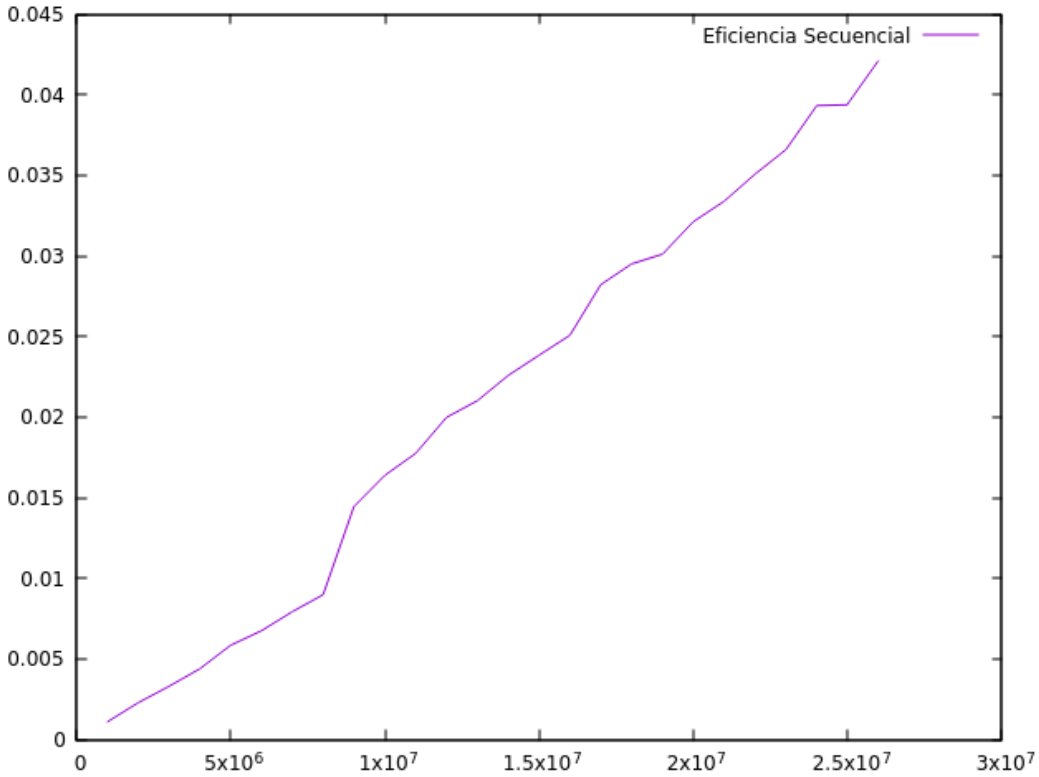
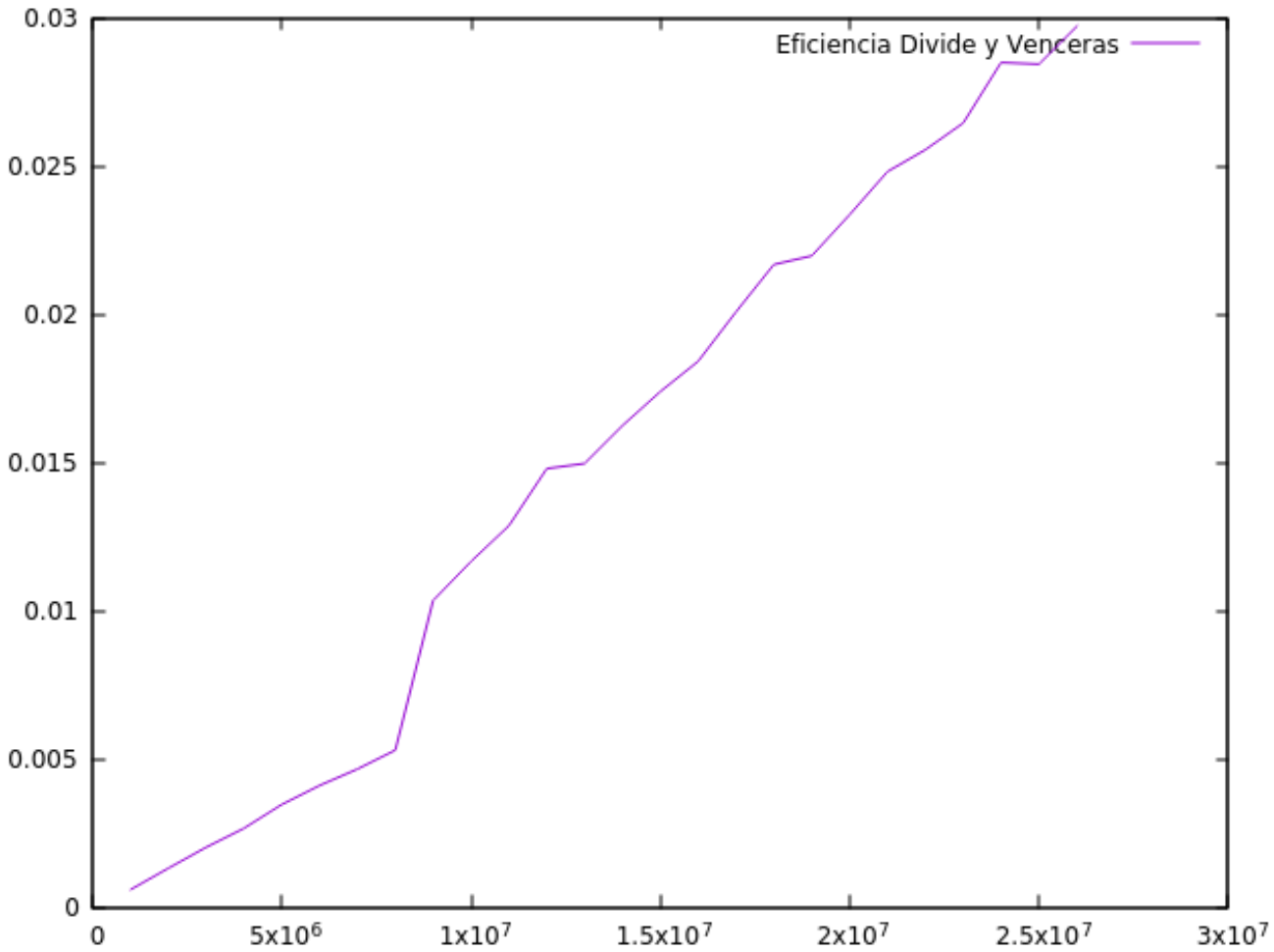
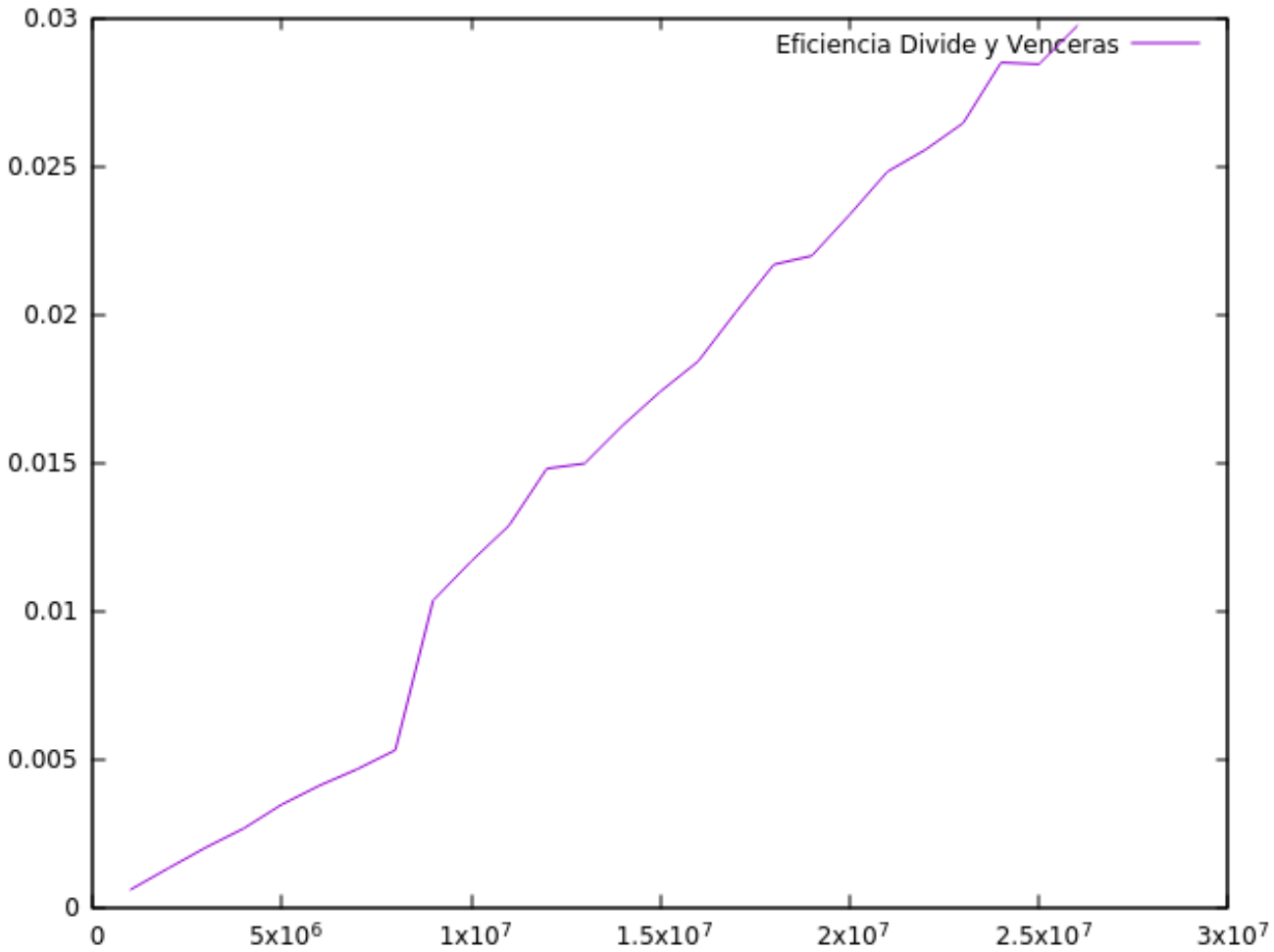


Figura: Pie de imagen

Eficiencia en el caso Divide y Vencerás



Eficiencia en el caso Divide y Vencerás



Ajuste híbrido en el caso secuencial

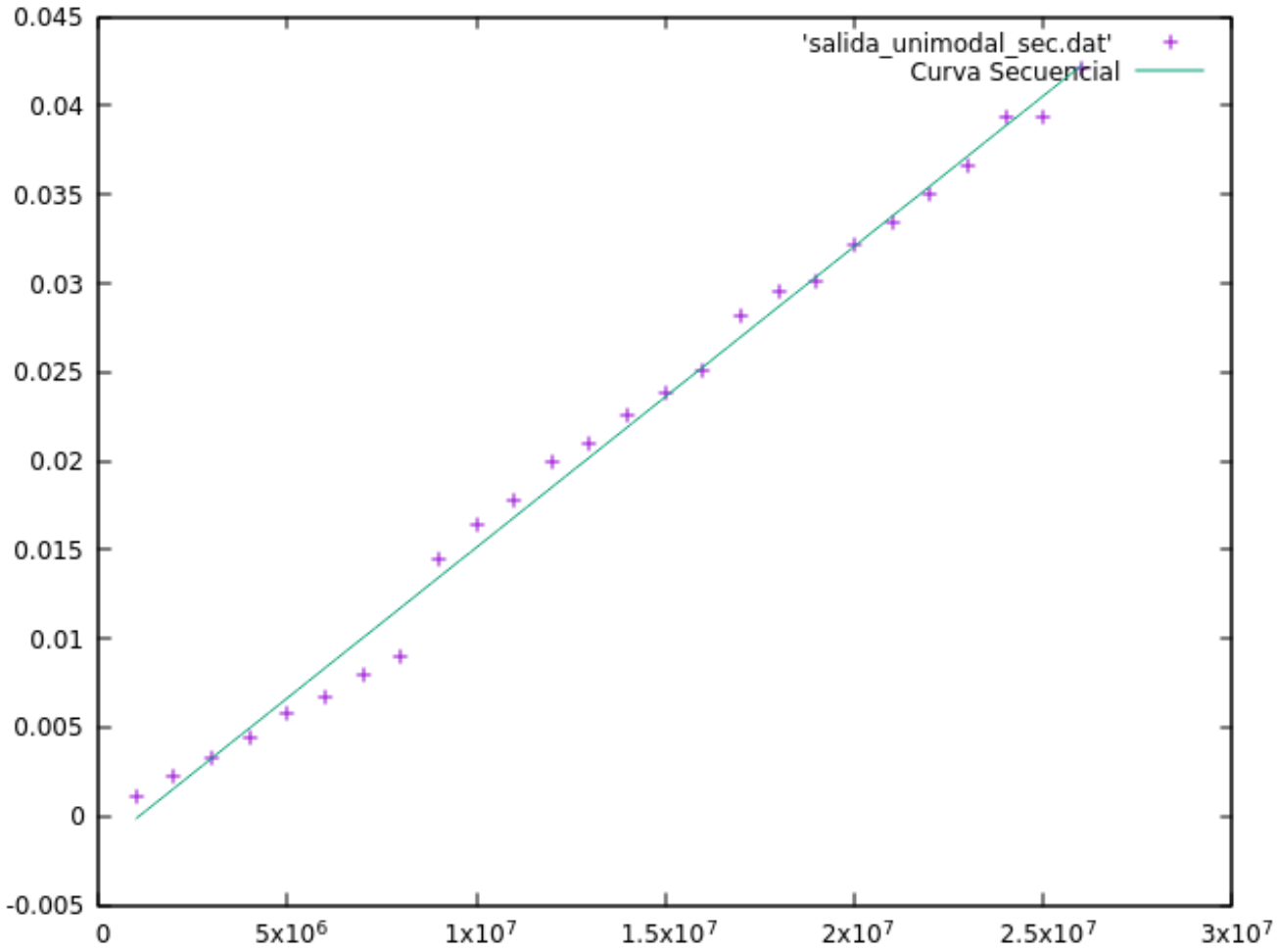
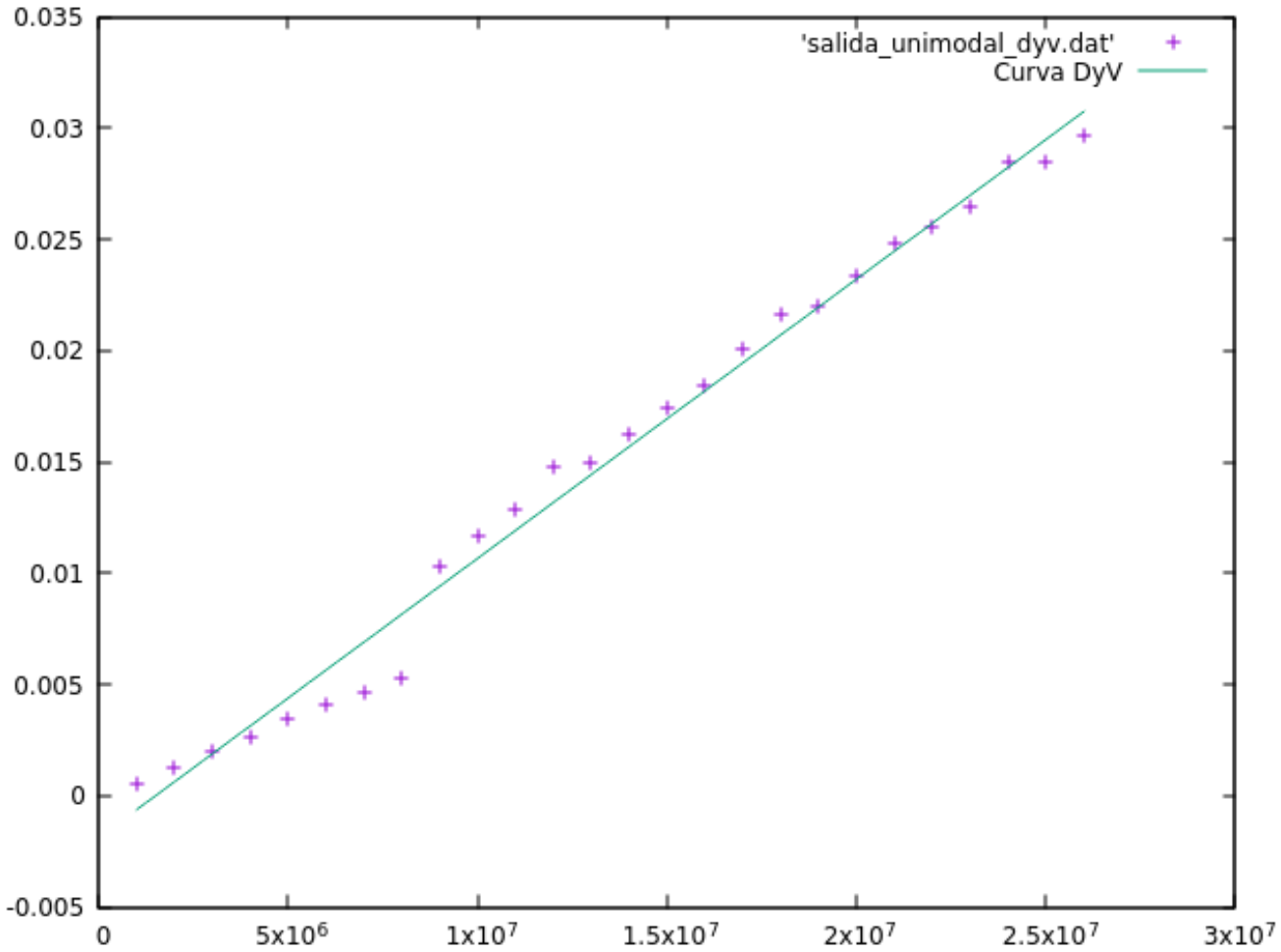


Figura: Pie de imagen

Ajuste híbrido en el caso Divide y Vencerás



Conclusión