

Algorítmica

Curso 2023-2024

Grupo Viterbi



PRÁCTICA 2-DIVIDE Y VENCERÁS

Integrantes:

Miguel Ángel De la Vega Rodríguez	miguevrod@correo.ugr.es
Alberto De la Vera Sánchez	joaquin724@correo.ugr.es
Joaquín Avilés De la Fuente	adelaveras01@correo.ugr.es
Manuel Gomez Rubio	e.manuelgmez@go.ugr.es
Pablo Linari Perez	e.pablolinari@go.ugr.es

Facultad de Ciencias UGR
Escuela Técnica Ingeniería Informática UGR
Granada
2023-2024

Índice general

1	Autores	3
2	Objetivos	4
3	Definicion Problema	5
4	Algoritmo Especifico	6
4.1	Problema 1: Subsecuencia de suma máxima.	6
5	Algoritmo Divide y Vencerás	8
5.1	Problema 1: Subsecuencia de suma máxima.	8
6	Conclusiones	12

Apartado 1

Autores

- **Miguel Ángel De la Vega Rodríguez: 20%**
 - Plantilla y estructura del documento \LaTeX
- **Joaquín Avilés De la Fuente: 20%**
 - Tarea
- **Alberto De la Vera Sánchez: 20%**
 - Tarea
- **Manuel Gomez Rubio 20%**
 - Tarea
- **Pablo Linari Pérez: 20%**
 - Tarea

Apartado 2

Objetivos

En esta práctica, se pretende resolver problemas de forma eficiente aplicando la técnica de Divide y Vencerás. Para ello, se han planteado varios problemas cuya solución es conocida (excepto para el problema del viajante), y se han implementado algoritmos que los resuelven mediante el método convencional y mediante la técnica de Divide y Vencerás. Posteriormente, se ha buscado un umbral en el cual ambos tengan el mismo tiempo de ejecución, finalmente, se ha buscado el umbral óptimo para cada problema.

Apartado

3

Definicion Problema

Apartado 4

Algoritmo Especifico

En este apartado, estudiaremos la eficiencia teórica, empírica e híbrida de los algoritmos específicos de cada uno de los problemas.

4.1 Problema 1: Subsecuencia de suma máxima.

Para el primer problema, el algoritmo específico que empleamos es el algoritmo de Kadane.

Estudio teórico

```
1  int kadane(int *a, int size){
2      int max_global = a[0];
3      int max_current = a[0];
4
5      for (int i = 1; i < size; i++) {
6          max_current = max(a[i], max_current + a[i]);
7          if (max_current > max_global) {
8              max_global = max_current;
9          }
10     }
11     return max_global;
12 }
```

Como podemos observar la eficiencia del código en las líneas 6-8, tienen eficiencia $O(1)$. Por tanto, su tiempo de ejecución es constante y notaremos por a . Luego, el bucle `for` se ejecutará $(size - 1) - i + 1$ veces, es decir, $size - i$ veces. Sabiendo que el resto de líneas del código tienen eficiencia $O(1)$, tenemos el siguiente resultado

$$\sum_{i=1}^{size-1} a$$

Tomaremos $size = n$ e $inicial = 1$ para simplificar el cálculo y veamos que obtenemos ahora

$$\sum_{i=1}^{n-1} a = a \cdot \sum_{i=1}^{n-1} 1 = a \cdot n$$

Es claro que $a \cdot n \in O(n)$ y por tanto la eficiencia teórica del algoritmo de kadane es $O(n)$.

Estudio empírico

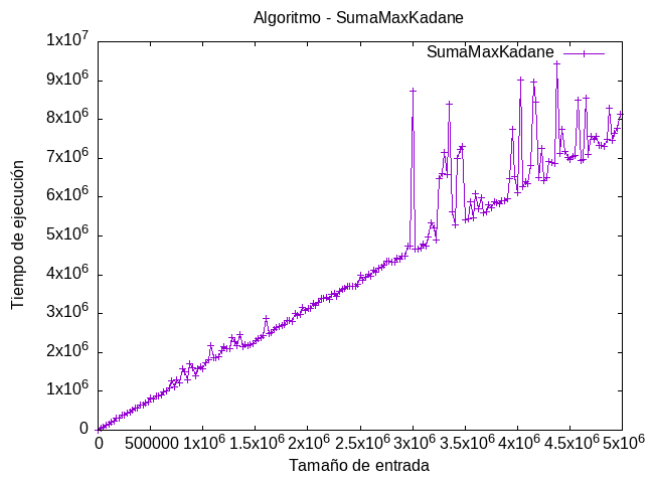


Figura 4.1: Ejecución algoritmo SumaMaxKadane

Estudio híbrido

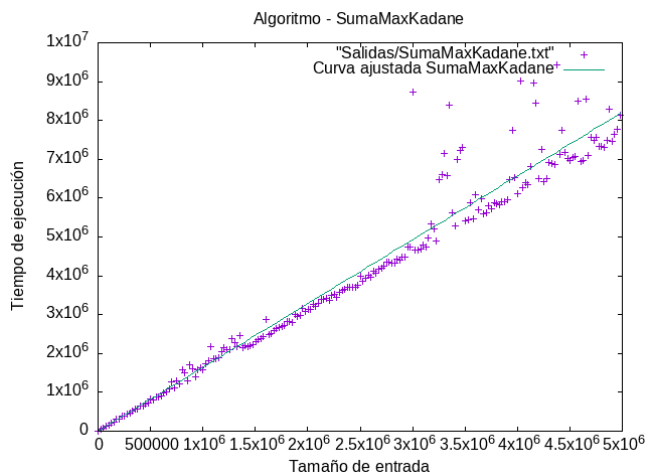


Figura 4.2: Ajuste híbrido algoritmo Kadane

Tras la interpretación de los datos empíricos en gnuplot y de la formula teorica del método, obtenemos que las constantes ocultas son:

$$T_{Kadane} = 1.64263x + 0.996451$$

Apartado 5

Algoritmo Divide y Vencerás

En este apartado, estudiaremos la eficiencia teórica, empírica e híbrida de los algoritmos divide y vencerás de cada uno de los problemas.

5.1 Problema 1: Subsecuencia de suma máxima.

Para el primer problema, el algoritmo específico que empleamos es

Estudio teórico

```
1  SumaData SumaMax (int *v, int inicio, int final){
2      SumaData result, d1, d2;
3      if (inicio==final){
4          result.max_izq = v[inicio];
5          result.max_dch = v[inicio];
6          result.sum = v[inicio];
7          result.max_sub = v[inicio];
8          return (result);
9      }
10
11     int mid = (final+inicio)/2;
12     (d1)=SumaMax(v, inicio, mid);
13     (d2)=SumaMax(v, mid+1, final);
14
15     result.max_izq = max(d1.max_izq, d1.sum+d2.max_izq);
16     result.max_dch = max(d2.max_dch, d2.sum+d1.max_dch);
17     result.sum = d1.sum + d2.sum;
18     int max_cross = d1.max_dch + d2.max_izq;
19     result.max_sub = max(max(max_cross, d1.max_sub), d2.max_sub);
20     return (result);
21 }
```

Este algoritmo devuelve un tipo de dato Suma data, un struct definido por cuatro datos de tipo int. En cuanto a la eficiencia teórica, podemos ver una llamada recursiva a la función SumaMax con vectores de tamaño $\frac{n}{2}$. Teniendo en cuenta que el resto de líneas de código son asignaciones, comparaciones y operaciones aritméticas, que son $O(1)$, obtenemos la siguiente ecuación

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

Al realizar un cambio de variable $n = 2^m$ (luego $m = \log_2(n)$), obtenemos:

$$T(2^m) = 2T(2^{m-1}) + 1$$

$$T(2^m) - 2T(2^{m-1}) = 1$$

Ahora calculamos por un lado la parte homogénea y por otro la no homogénea. En primer lugar, la homogénea:

$$T(2^m) - 2T(2^{m-1}) = 0 \implies p_H(x) = x - 2$$

En cuanto a la parte no homogénea

$$1 = b_1^m q_1(m) \implies b_1 = 1 \wedge q_1(m) = 1 \text{ con grado } d_1 = 0$$

Tenemos entonces el siguiente polinomio característico

$$p(x) = (x - 2)(x - b_1)^{d_1+1} = (x - 2)(x - 1)$$

Por tanto la solución general es

$$t_m = c_{10}2^m m^0 + c_{20}1^m m^0 \xrightarrow{*} t_n = c_{10}n + c_{20} \implies T(n) = c_{10}n + c_{20}$$

donde en (*) hemos deshecho el cambio de variable

Por lo que obtenemos como resultado que $T(n) \in O(n)$

Estudio empírico

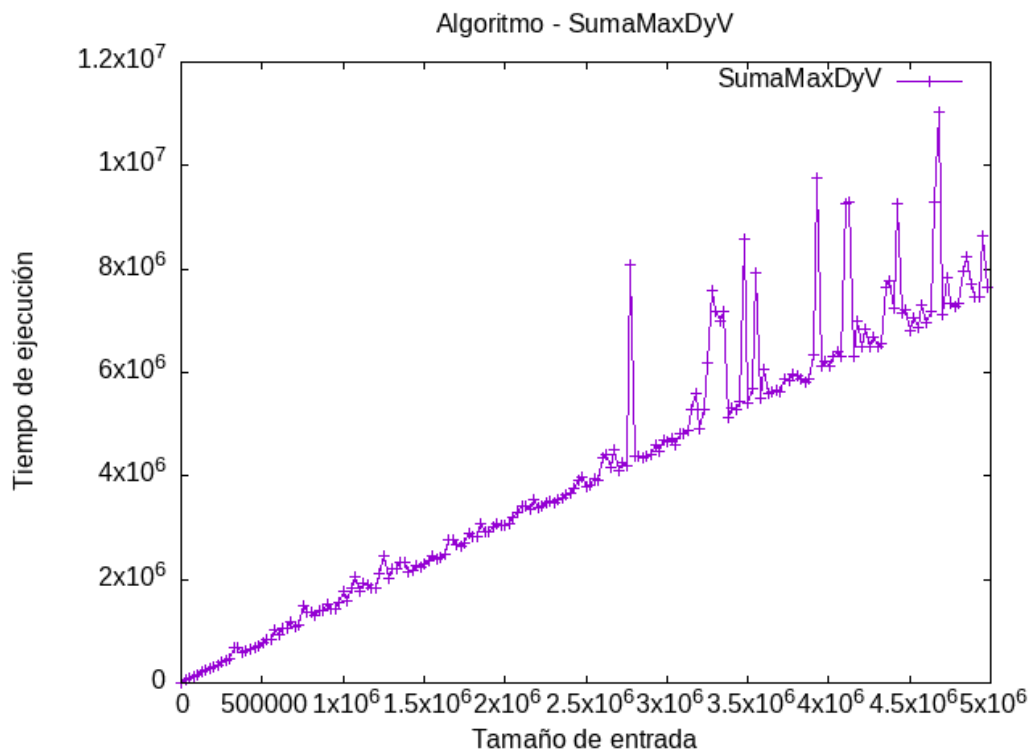


Figura 5.1: Ejecución algoritmo SumaMaxDyV

Estudio híbrido

Tras la interpretación de los datos empíricos en gnuplot y de la formula teorica del método, obtenemos que las constantes ocultas son:

$$T_{SumaMaxDyV} = 1.65508x + 0.991967$$

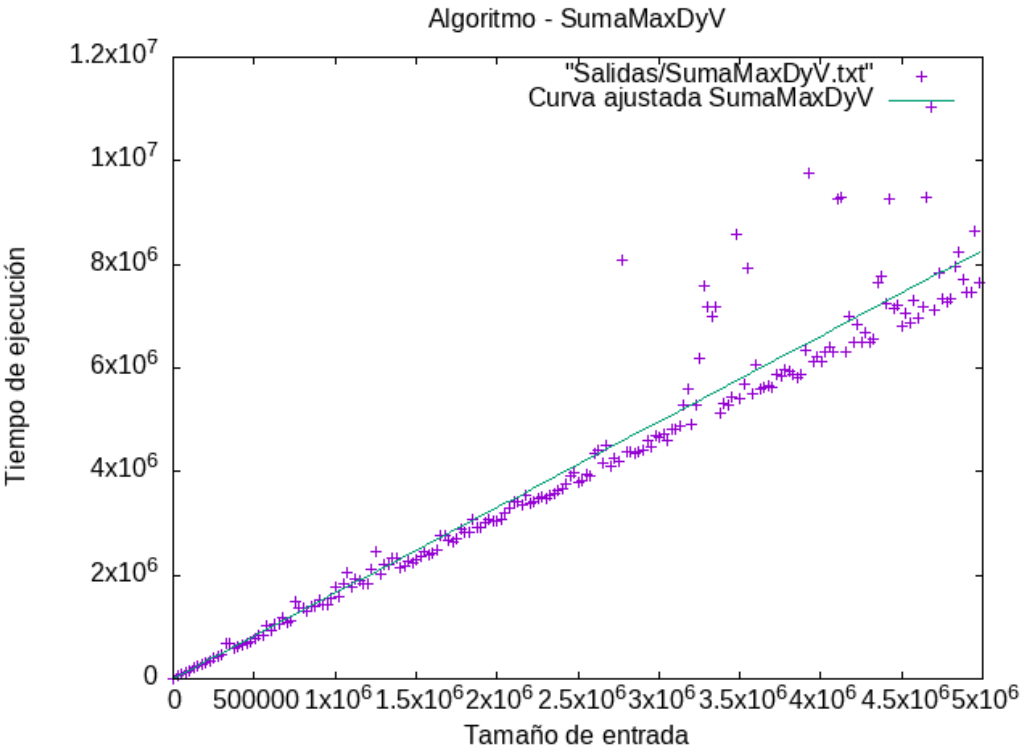


Figura 5.2: Ajuste híbrido algoritmo SumaMaxDyV

Apartado 6

Conclusiones