UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍAS INFORMATICA y DE TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Práctica 2: Algoritmos Divide y Vencerás

Alejandro Casado Quijada
Andrés Ortiz Corrales
Antonio Jiménez Martínez
Jesús Prieto López
Salvador Rueda Molina

Curso 2013-2014
Grado en Informática

1. Objetivo

El objetivo de esta práctica es que el estudiante aprecie la utilidad de la t́
ecnica “divide y vencerás” para resolver problemas de forma más eficiente que otras alternativas más sencillas o directas. Para ello cada equipo de estudiantes deberá resolver uno de los problemas (escogido al azar) que se detallan más adelante, así como exponer y defender su propuesta en clase.

2. Serie unimodal de números

Sea un vector v de números de tamaño n, todos distintos, de forma que existe un índice p (que no es ni el primero ni el último) tal que a la izquierda de p los números están ordenados de forma creciente y a la derecha de p están ordenados de forma decreciente;
es decir $\forall i, j \leq p, i < j \Rightarrow v[i] < v[j]$ y $\forall i, j \geq p, i < j \Rightarrow v[i] > v[j]$ (de forma que el máximo se encuentra en la posición p).

Diseñamos un algoritmo de “divide y vencerás” que determinar el valor máximo en un vector unimodal. Este proceso consiste en buscar el elemento medio y comprobamos si es el máximo entre los elementos anterior y consecutivo. En caso contrario divide el vector en dos y vuelve a aplicar este algoritmo. Así hasta encontrar el máximo. El algoritmo es el siguiente:

```
int dyvunimodal(const std::vector<int> &v,int inicio,int fin) {
       int siz=fin-inicio+1;
    if(siz>2) {
        int p=(siz/2)+inicio;
        int a=v[p-1];
        int b=v[p+1];
        int vp=v[p];
        if(vp>a && vp>b) return vp; //si lo encuentra en la primera
iteracion
        else {
            if(vp<b) {//coger la segunda mitad desde vp
                return dyvunimodal(v,p+1,fin);
            }
            if(vp>b) {//coger la primera mitad desde vp
                return dyvunimodal(v,inicio,p);
            }
        }
    }
    else {
        if(siz==2) { //si quedan 2 elementos, coje el mayor
            if(v[inicio]>v[fin]) return v[inicio];
            else return v[fin];
        }
        else //v.size()==1 si queda uno, lo devuelve
            return v[inicio];
    }
}
```

También creamos un algoritmo de fuerza bruta para encontrar el máximo en un vector unimodal. Este proceso consiste en recorrer el vector y determinar el máximo. El codigo del algoritmo es el siguiente:

```
int fbunimodal(const vector<int> & t) {
    int antes=-1,p=-1;
    if(t.size()>0) {
        int antes=t[0];
        for(int i=1; i<t.size(); i++) {
            if(p==-1) {
                if(antes<t[i]) {
                    antes=t[i];
                }
                else { //antes>t[i]
                    p=antes;
                    antes=t[i];
                }
            }
            else { //p!=-1
                if(antes>t[i]) {
                    antes=t[i];
                }
                else { //antes<t[i]
                    antes=t[i];
                }
            }
        }
    }
    return p;
}
```

Tenemos un generadores de datos para crear un vector unimodal.

3. Cálculo del tiempo teórico

Tenemos dos algoritmo para resolver este problema.
-Divide y vencerás que es de forma recursiva de orden O(log(n)). Conside en dividir el vector e ir comprobando si en ese vector se encuentra el maximo.

-Fuerza bruta, consiste en recorrer el vector y encontrar el máximo. Este algoritmo es de orden O(n).

4. Cálculo del tiempo Empírica

Realizaremos el cálculo de la eficiencia empírica estudiando el comportamiento de los dos algoritmos, de “Divide y vencerás” y de “Fuerza bruta”.

Con el uso del archivo de entradas proporcionado mediremos los tiempos de ejecución para cada tamaño de entrada. Será mediante entradas de vectores unimodales.

Para el cálculo empírico nos hemos ayudado de la biblioteca STL::chrono para una mayor precisión en el tiempo.

Hemos creado un script para ejecutar los distintos algoritmos con diferentes tamaños de entradas.

5. Cálculo del tiempo Híbrida

Para conocer la forma mas exacta de la ecuación del tiempo de ejecución para una situación concreta se utiliza el calculo de la eficiencia híbrida. A la hora de realizar el cálculo teórico se presenta cada término con una constante de valor desconocido.

Entonces necesitamos saber el valor de esas constantes, ajustando la función a un conjunto de puntos. Se utiliza la función del cálculo teórico y el conjunto de puntos lo forman los resultados del análisis empírico y se ajusta con regresión de mínimos cuadrados.

Hemos utilizado las diferentes funciones de regresión para las diferentes grafias:

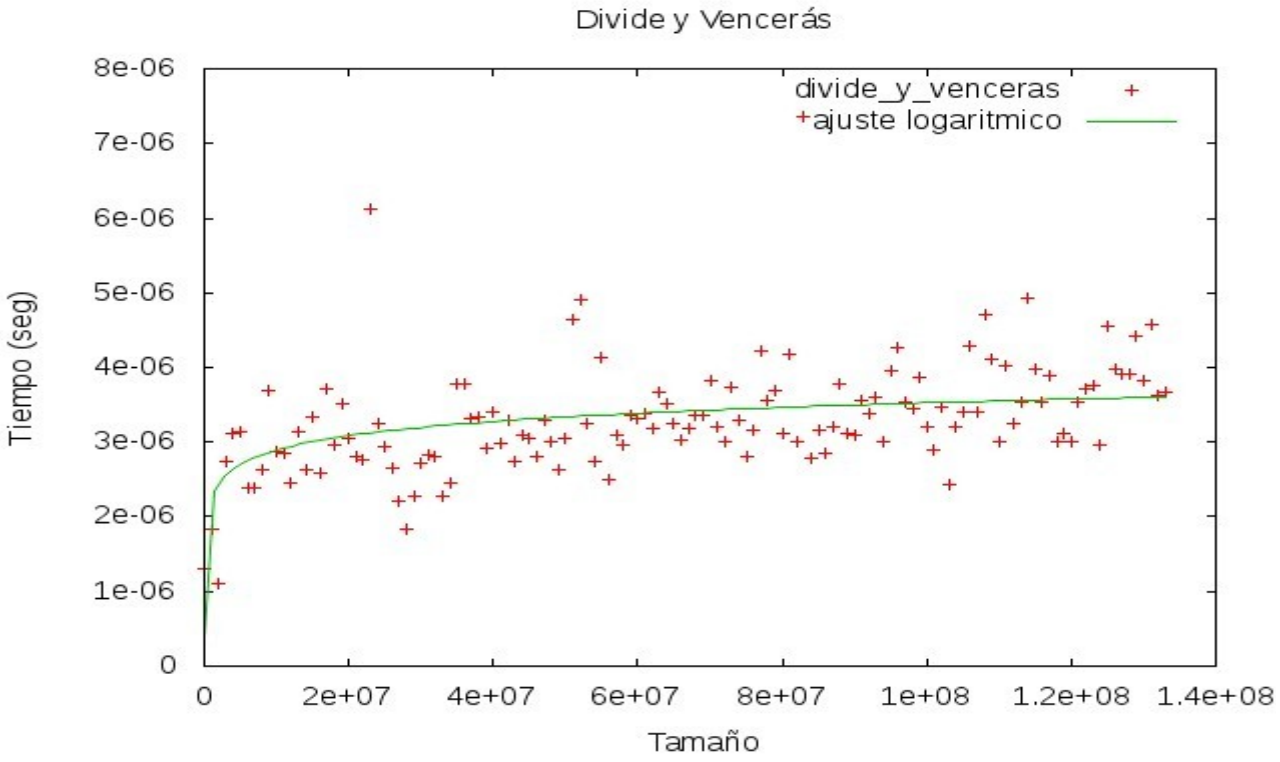
-lineal: a0*x+a1

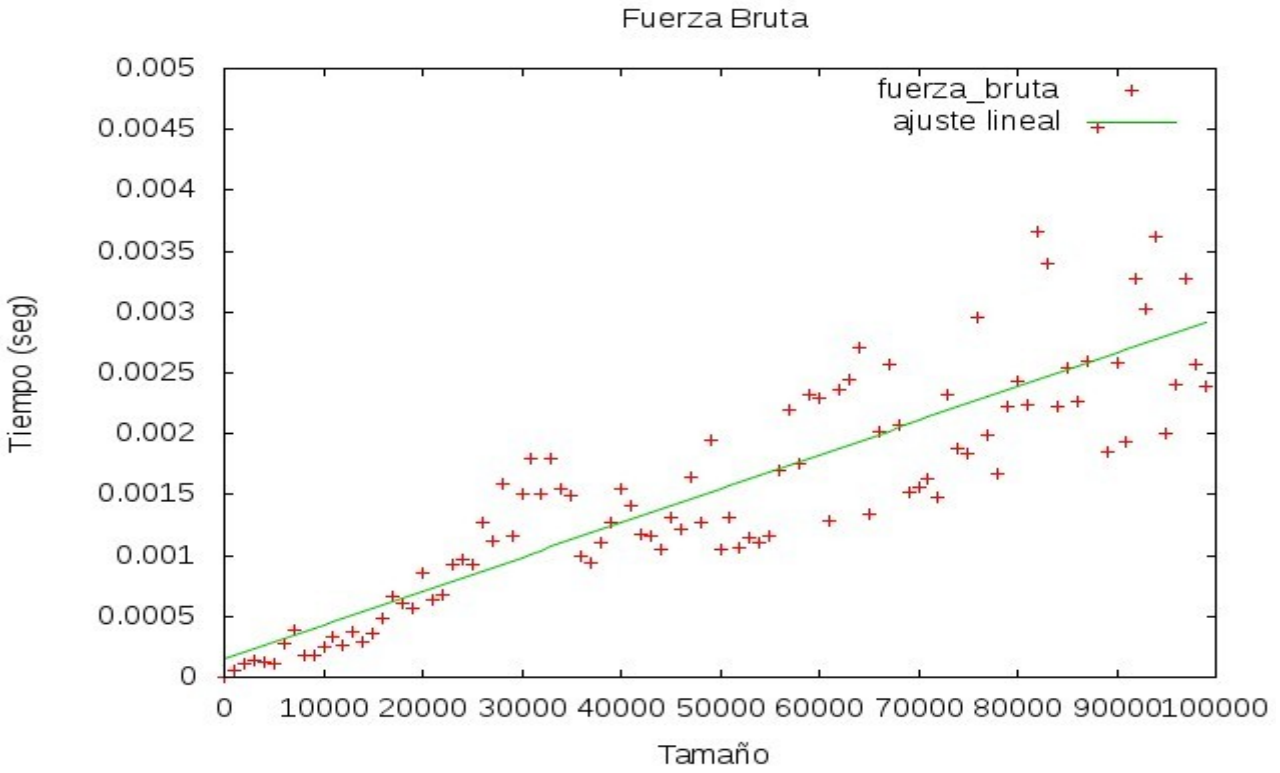
-logaritmicos: a0*log (x)+a1

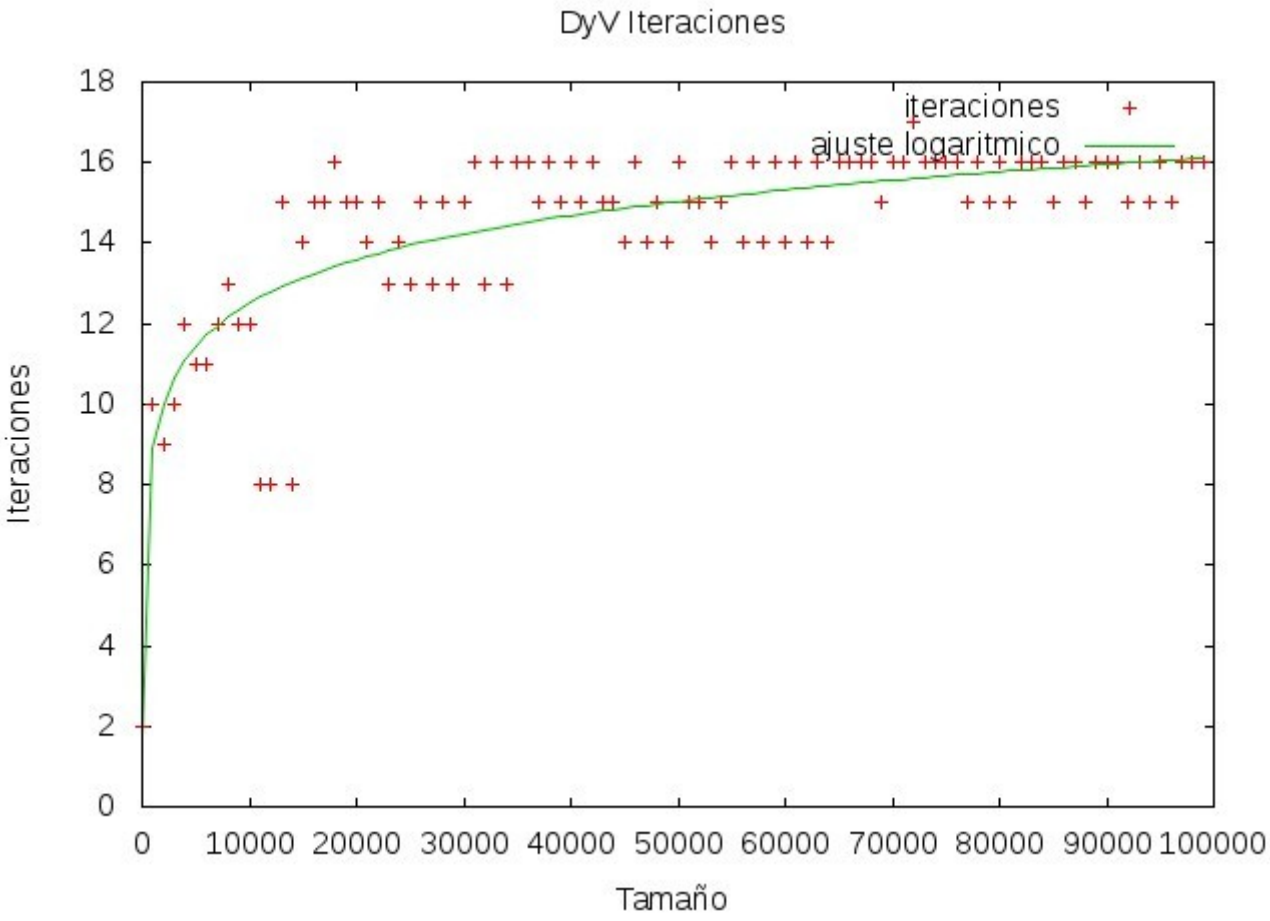
Cuando decimos que aplicamos el algoritmo a <<una situación concreta>> nos referimos a por ejemplo:

- Compilador utilizado: g++
- Ordenador sobre el que se ejecuta, en nuestro caso con la siguientes características:
  - Intel core i3 m330 2.13 Ghz
  - 4 GB RAM
  - S.O. Ubuntu 13.04 64 bits

A continuación le mostraremos las siguientes gráficas y tablas de los diferentes algoritmos. También realizaremos una comparación gráficas del algoritmo de fuerza de orden lineal y el algoritmo de divide y vencerás de orden logarítmico.


Divide y Vencerás
divide_y_venceras
ajuste logaritmico
Tiempo (seg)
Tamaño
8e-06
7e-06
6e-06
5e-06
4e-06
3e-06
2e-06
1e-06
0
0
2e+07
4e+07
6e+07
8e+07
1e+08
1.2e+08
1.4e+08


Fuerza Bruta
fuerza_bruta
ajuste lineal
Tiempo (seg)
Tamaño
0.005
0.0045
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000


DyV Iteraciones
iteraciones
ajuste logaritmico
Iteraciones
Tamaño
0
2
4
6
8
10
12
14
16
18
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000

Fuerza bruta tiempo	
tamaño	tiempo
10	1.593e-06
1010	6.0697e-05
2010	0.000116925
3010	0.000139917
4010	0.000128555
5010	0.000104043
6010	0.000274181
7010	0.000383103
8010	0.00018591
9010	0.000185658
10010	0.000254032
11010	0.000331509
12010	0.000261192
13010	0.000375555
14010	0.00028883
15010	0.000364736
16010	0.000485701
17010	0.000656982
18010	0.000605863
19010	0.000571207
20010	0.00085349
21010	0.000630269
22010	0.000673947
23010	0.000919732
24010	0.000969631
25010	0.000922872
26010	0.00127328
27010	0.00112309
28010	0.0015938
29010	0.00115974
30010	0.001512
31010	0.00179662
32010	0.001499
33010	0.00179178
34010	0.00155062
35010	0.00149367
36010	0.000987614
37010	0.000940374
38010	0.0011064
39010	0.0012755
40010	0.00154133
41010	0.00141199
42010	0.00117109
43010	0.00116569
44010	0.00104972
45010	0.00130903
46010	0.00122078
47010	0.00164952
48010	0.00126912

49010	0.00194341
50010	0.00104295
51010	0.00131866
52010	0.00106807
53010	0.00114979
54010	0.00111058
55010	0.00116478
56010	0.00169787
57010	0.00220047
58010	0.00175802
59010	0.00231674
60010	0.00229303
61010	0.00128087
62010	0.00235924
63010	0.00244597
64010	0.00270247
65010	0.00133663
66010	0.00201032
67010	0.00257012
68010	0.00207419
69010	0.00151808
70010	0.0015653
71010	0.00162584
72010	0.00148154
73010	0.00231453
74010	0.0018783
75010	0.00183334
76010	0.00294969
77010	0.00198281
78010	0.00166808
79010	0.00222109
80010	0.00242789
81010	0.00223381
82010	0.0036574
83010	0.00339197
84010	0.0022169
85010	0.00253513
86010	0.00226047
87010	0.00260214
88010	0.00452017
89010	0.00184524
90010	0.00258477
91010	0.0019328
92010	0.00326928
93010	0.00301881
94010	0.00361966
95010	0.00200688
96010	0.00240751
97010	0.00327811
98010	0.00256818
99010	0.00239251

Divide y vencerás tiempo	
tamaño	tiempo
1000	1.313e-06
1001000	1.836e-06
2001000	1.114e-06
3001000	2.737e-06
4001000	3.116e-06
5001000	3.133e-06
6001000	2.384e-06
7001000	2.397e-06
8001000	2.64e-06
9001000	3.699e-06
10001000	2.876e-06
11001000	2.86e-06
12001000	2.449e-06
13001000	3.13e-06
14001000	2.627e-06
15001000	3.341e-06
16001000	2.584e-06
17001000	3.705e-06
18001000	2.967e-06
19001000	3.523e-06
20001000	3.055e-06
21001000	2.801e-06
22001000	2.755e-06
23001000	6.128e-06
24001000	3.24e-06
25001000	2.94e-06
26001000	2.662e-06
27001000	2.219e-06
28001000	1.83e-06
29001000	2.27e-06
30001000	2.71e-06
31001000	2.82e-06
32001000	2.817e-06
33001000	2.269e-06
34001000	2.456e-06
35001000	3.774e-06
36001000	3.768e-06
37001000	3.306e-06
38001000	3.339e-06
39001000	2.908e-06
40001000	3.413e-06
41001000	2.993e-06
42001000	3.288e-06
43001000	2.747e-06
44001000	3.095e-06
45001000	3.039e-06
46001000	2.815e-06
47001000	3.285e-06
48001000	2.996e-06

49001000	2.633e-06
50001000	3.05e-06
51001000	4.64e-06
52001000	4.897e-06
53001000	3.259e-06
54001000	2.751e-06
55001000	4.135e-06
56001000	2.507e-06
57001000	3.092e-06
58001000	2.962e-06
59001000	3.359e-06
60001000	3.32e-06
61001000	3.383e-06
62001000	3.183e-06
63001000	3.678e-06
64001000	3.514e-06
65001000	3.252e-06
66001000	3.017e-06
67001000	3.189e-06
68001000	3.368e-06
69001000	3.367e-06
70001000	3.816e-06
71001000	3.197e-06
72001000	2.996e-06
73001000	3.741e-06
74001000	3.299e-06
75001000	2.809e-06
76001000	3.164e-06
77001000	4.228e-06
78001000	3.565e-06
79001000	3.683e-06
80001000	3.127e-06
81001000	4.183e-06
82001000	2.998e-06
83001000	7.36e-06
84001000	2.791e-06
85001000	3.164e-06
86001000	2.85e-06
87001000	3.215e-06
88001000	3.782e-06
89001000	3.117e-06
90001000	3.09e-06
91001000	3.568e-06
92001000	3.389e-06
93001000	3.6e-06
94001000	2.996e-06
95001000	3.945e-06
96001000	4.273e-06
97001000	3.539e-06
98001000	3.443e-06
99001000	3.859e-06
100001000	3.204e-06

101001000	2.887e-06
102001000	3.462e-06
103001000	2.432e-06
104001000	3.204e-06
105001000	3.405e-06
106001000	4.293e-06
107001000	3.413e-06
108001000	4.702e-06
109001000	4.121e-06
110001000	3.006e-06
111001000	4.028e-06
112001000	3.252e-06
113001000	3.541e-06
114001000	4.937e-06
115001000	3.972e-06
116001000	3.541e-06
117001000	3.897e-06
118001000	2.999e-06
119001000	3.114e-06
120001000	3.009e-06
121001000	3.546e-06
122001000	3.71e-06
123001000	3.755e-06
124001000	2.954e-06
125001000	4.543e-06
126001000	3.975e-06
127001000	3.905e-06
128001000	3.913e-06
129001000	4.41e-06
130001000	3.816e-06
131001000	4.567e-06
132001000	3.627e-06
133001000	3.659e-06

Divide y vencerás iteraciones	
tamaño	iteraciones
10	2
1010	10
2010	9
3010	10
4010	12
5010	11
6010	11
7010	12
8010	13
9010	12
10010	12
11010	8
12010	8
13010	15
14010	8
15010	14
16010	15
17010	15
18010	16
19010	15
20010	15
21010	14
22010	15
23010	13
24010	14
25010	13
26010	15
27010	13
28010	15
29010	13
30010	15
31010	16
32010	13
33010	16
34010	13
35010	16
36010	16
37010	15
38010	16
39010	15
40010	16
41010	15
42010	16
43010	15
44010	15
45010	14
46010	16
47010	14
48010	15

49010	14
50010	16
51010	15
52010	15
53010	14
54010	15
55010	16
56010	14
57010	16
58010	14
59010	16
60010	14
61010	16
62010	14
63010	16
64010	14
65010	16
66010	16
67010	16
68010	16
69010	15
70010	16
71010	16
72010	17
73010	16
74010	16
75010	16
76010	16
77010	15
78010	16
79010	15
80010	16
81010	15
82010	16
83010	16
84010	16
85010	15
86010	16
87010	16
88010	15
89010	16
90010	16
91010	16
92010	15
93010	16
94010	15
95010	16
96010	15
97010	16
98010	16
99010	16