Práctica 2: Algoritmos Divide y Vencerás
Algoritmo de Comparación de Preferencias en un Vector Ordenado.

Adrián Portillo Sánchez
Alejandro Durán Castro
Javier Labrat Rodríguez
Jose Antonio Martínez López
Jose Juán Pérez González

5 de abril de 2015

Índice

1. Introducción

Muchos sitios web intentan comparar las preferencias de dos usuarios para realizar sugerencias a partir de las preferencias de usuarios con gustos similares a los nuestros. Dado un ranking de n productos (p.ej. películas) mediante el cual los usuarios indicamos nuestras preferencias, un algoritmo puede medir la similitud de nuestras preferencias contando el número de inversiones: Dos productos i y j están invertidos en las preferencias de A y B si el usuario A prefiere el producto i antes que el j, mientras que el usuario B prefiere el producto j antes que el i. Esto es, cuantas menos inversiones existan entre dos rankings, más similares serán las preferencias de los usuarios representados por esos rankings.
Por simplicidad podemos suponer que los productos se pueden identificar mediante enteros 1, . . . , n, y que uno de los rankings siempre es 1, . . . , n (si no fuese así bastaría reenumerarlos) y el otro es a1, a2, . . . , an, de forma que dos productos i y j están invertidos si $i < j$ pero $ai > aj$ De esta forma nuestra representación del problema será un vector de enteros v de tamaño n,de forma que $v[i] = ai, i = 1, ..., n$.
El objetivo es diseñar, analizar la eficiencia e implementar un algoritmo "divide y vencerás" para medir la similitud entre dos rankings. Compararlo con el algoritmo de "fuerza bruta" obvio. Realizar también un estudio empírico e híbrido de la eficiencia de ambos algoritmos.

2. Marco de ejecución de los Algoritmos.

Ambos algoritmos, fuerza bruta y divide y vencerás, serán ejecutados como funciones de un mismo código, que será el marco donde estos algoritmos se ejecuten.

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <climits>
#include <cassert>

using namespace std;

double uniforme() {
  int t = rand();
  double f = ((double)RAND_MAX+1.0);
  return (double)t/f;
}

int main(int argc, char * argv[]) {
  if (argc != 2) {
    cerr << "Formato " << argv[0] << " <num_elem>" << endl;
    return -1;
  }
  int n = atoi(argv[1]);
  int * T = new int[n];
  assert(T);
  srand(time(0));

  for (int j=0; j<n; j++) T[j]=j;
  for (int j=n-1; j>0; j--) {
    double u=uniforme();
    int k=(int)(j*u);
    int tmp=T[j];
    T[j]=T[k];
    T[k]=tmp;
  }

  clock_t tantes;
  clock_t tdespues;
  unsigned int contador = 0;

  tantes = clock();
  contador = CompararPreferencias(T, n);
  tdespues = clock();

  cout << n << "datos" << (double)(tdespues - tantes)/CLOCKS_PER_SEC << "seg"
  << contador << "inversiones" <<endl;
  delete [] T;
  return 0;
}
```

3. Algoritmo Obvio de Fuerza Bruta.

El algoritmo de fuerza bruta no es más que el recorrido completo del algoritmo, realizando todas las comparaciones posibles, como tiene que realizar n comparaciones por cada elemento del vector este será de orden O(n^2), y servirá para demostrar la mejora que supone el algoritmo Divide y Vencerás, y para tener una referencia en la que basar las comparaciones de los análisis del algoritmo Divide y Vencerás. El código de la función CompararPreferencias() del algortimo de fuerza bruta es el siguiente:

```
int CompararPreferencias(int * preferencias, const int TAMA) {

    int total = 0;

    for (int i = 0; i < n-1; ++i) {
      for (int j = i+1; j < n; ++j) {
        if(T[i]>T[j])
          total++;
      }
    }

    return total;
}
```

Este algoritmo no hará mas que recorrer los elementos del vector uno a uno, y para cada elemento del vector compararlo con todos los siguientes, y si se ha provocado una inversión aumentar el contador, es un algoritmo obvio, que servirá de referencia para ver cuánto mejor es el algoritmo Divide y Vencerás.

4. Análisis del Algoritmo de Fuerza Bruta.

4.1. Características de la máquina utilizada

Hardware

Procesador: Intel Pentium Dual CPU T2390 @ 1.86GHz x 2

Memoria: 1,8 GiB

Sistema Operativo: Ubuntu 14.04 LTS 64-bit

4.2. Compilador usado y opciones de compilación

g++ -O2 -o preferenciasfuerzabruta preferenciasfuerzabruta.cpp

4.3. Script de ejecución

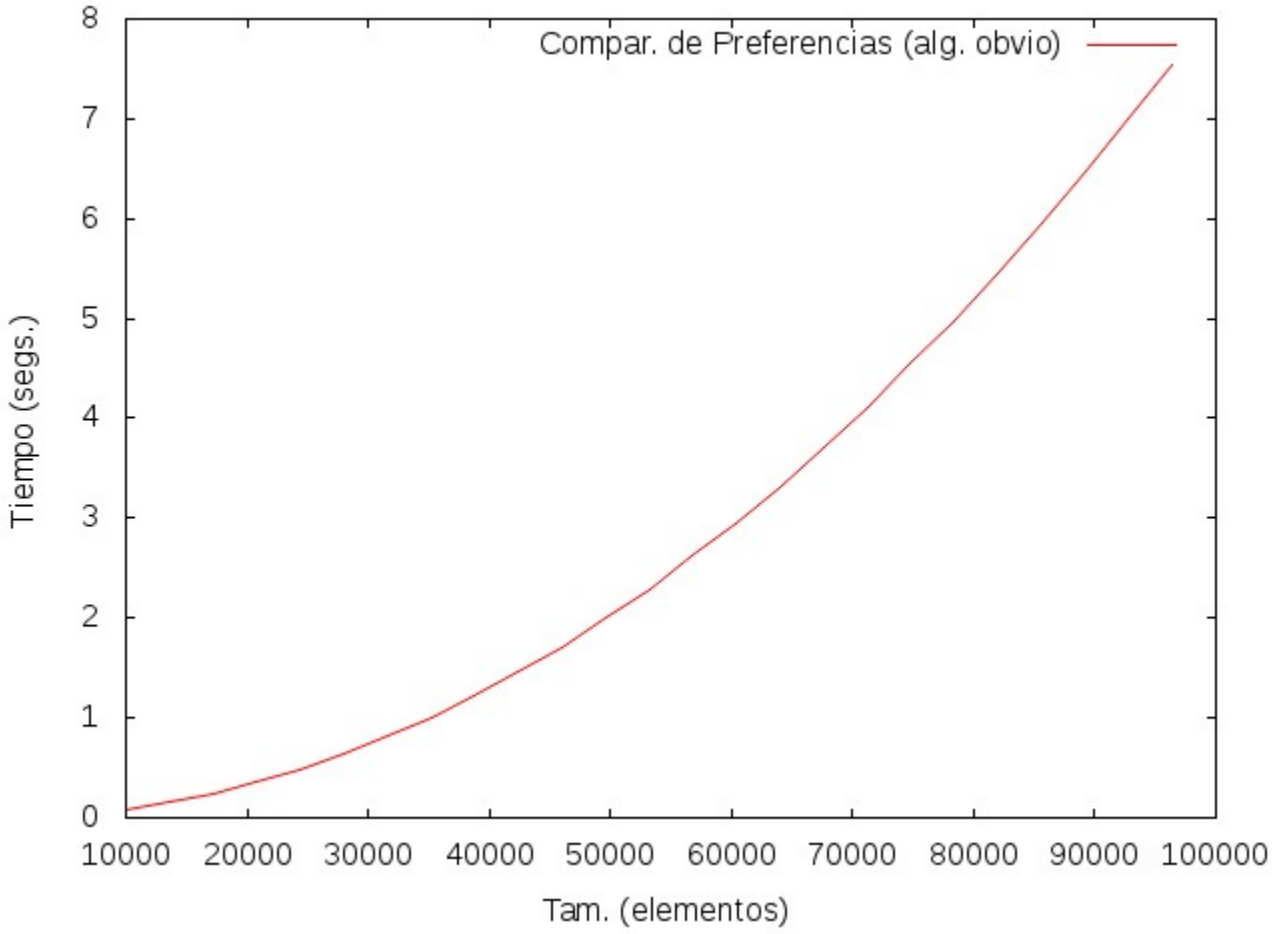
```
#!/bin/csh -vx

@ ini = 10000
@ fin = 100000
@ inc = 3600

set ejecutable = preferenciasfuerzabruta
@ i = $ini
while ( $i < $fin )
        ./{$ejecutable} $i >> salida_{$ejecutable}.dat
        @ i += $inc
end
```

4.4. Eficiencia Empírica.

4.4.1. Tabla de resultados de eficiencia empírica

Entrada (nº de elementos del vector.)	**Fuerza Bruta.**
10000	0.085013
13600	0.150597
17200	0.240659
20800	0.351986
24400	0.48289
28000	0.634687
31600	0.813784
35200	1.00338
38800	1.22489
42400	1.45426
46000	1.71152
49600	2.00041
53200	2.29517
56800	2.62445
60400	2.95342
64000	3.31483
67600	3.71655
71200	4.10363
74800	4.55106
78400	4.97663
82000	5.46354
85600	5.94939
89200	6.47332
92800	6.99986
96400	7.54656

4.4.2. Gráfica de Eficiencia Empírica.



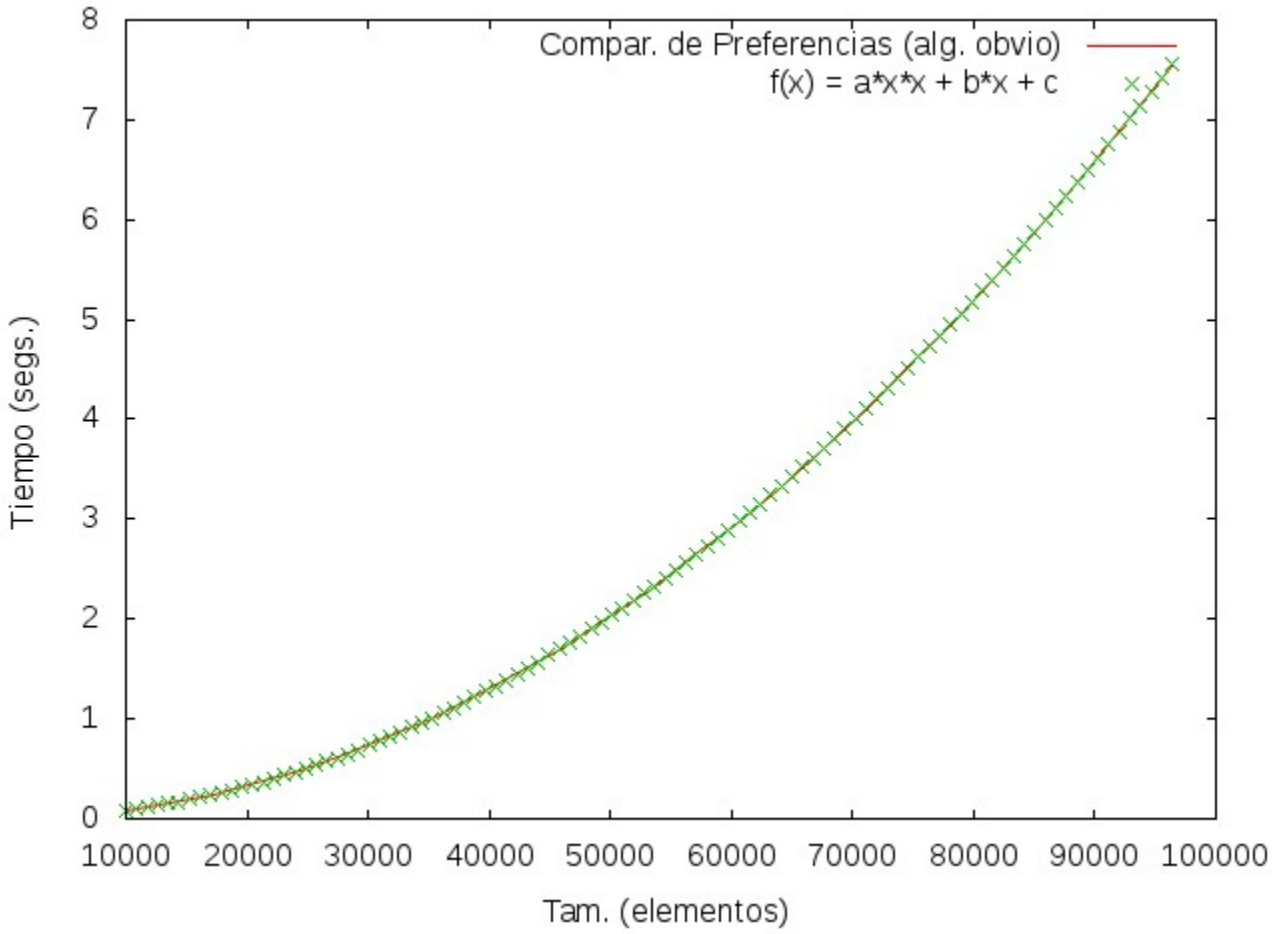
4.5. Eficiencia híbrida.

Ajuste de la función

FuerzaBruta, $f(x) = a * x^2 + b * x + c$

Final set of parameters	Asymptotic Standard Error
$a = 8{,}16045e^{-10}$	$+/- 2{,}113e^{-12}(0{,}259\,\%)$
$b = -4{,}1613e^{-07}$	$+/- 2{,}301e^{-07}(55{,}3\,\%)$
$c = 0{,}00748637$	$+/- 0{,}005401(72{,}14\,\%)$

Se puede observar que la función se ajusta perfectamente a una función polinómica de segundo grado.



5. Algoritmo Divide y Vencerás.

El algoritmo Divide y Vencerás divide en vector en dos partes, cada cual se va llama a la función CompararPreferencias() de forma recursiva y llamando a ContarInversiones(), contaría las inversiones en cada parte del vector, y las sumaría al final.
Las funciones CompararPreferencias() y ContarInversiones() del algoritmo Divide y Vencerás serían las siguientes:

```
int ContarInversiones(int * preferencias1, int * preferencias2, const int TAMA) {
  int total = 0;
  for(int i=0; i < TAMA; i++) {
    int j = 0;
    bool menor = true;
    while(j < TAMA && menor) {
      if(preferencias2[i %TAMA] < preferencias1[j]) {
        j++;
        total++;
      }
      else
        menor = false;
    }
  }
  return total;
}

int CompararPreferencias(int * preferencias, const int TAMA) {
  if(TAMA == 0)
    return 0;

  const int MITAD = TAMA/2;
  int * preferencias1 = new int[MITAD];
  int * preferencias2 = &preferencias[MITAD];
  preferencias1 = preferencias;

  int pref1 = CompararPreferencias(preferencias1, MITAD);
  int pref2 = CompararPreferencias(preferencias2, MITAD);
  int total = pref1 + pref2;

  total += ContarInversiones(preferencias1, preferencias2, MITAD);

  return total;
}
```

6. Análisis del Algoritmo Divide y Vencerás.

6.1. Características de la máquina utilizada

Hardware

Procesador: Intel Core i5-2430M CPU @ 2.40GHz x4

Memoria: 6 GiB

Sistema Operativo: Ubuntu 14.04 LTS 64-bit

6.2. Compilador usado y opciones de compilación

g++ -O2 -o preferenciasdyv preferenciasdyv.cpp

6.3. Script de ejecución

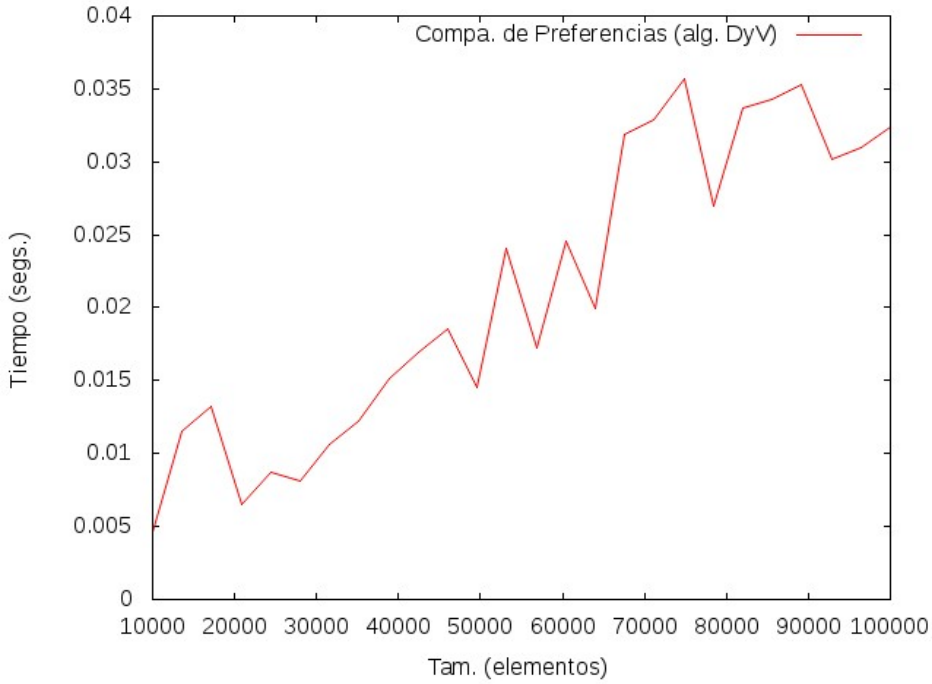
```
#!/bin/csh -vx

@ ini = 10000
@ fin = 100000
@ inc = 3600

set ejecutable = preferenciasDyV
@ i = $ini
while ( $i <= $fin )
        ./{$ejecutable} $i >> salida_{$ejecutable}.dat
        @ i += $inc
end
```

6.4. Eficiencia Empírica

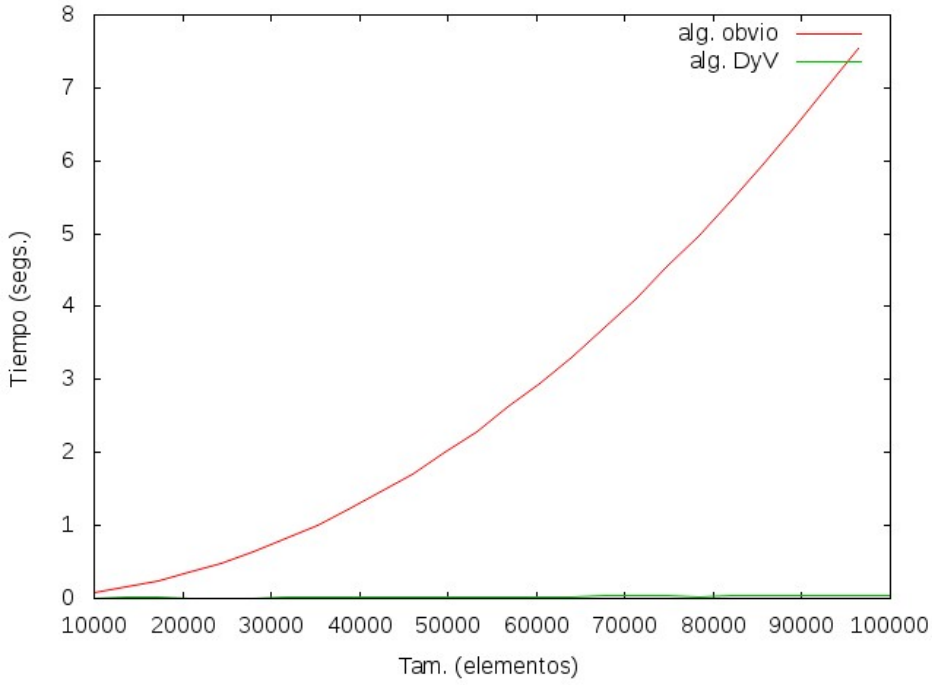
6.4.1. Tabla de resultados de eficiencia empírica.

Entrada (nº de elementos del vector.)	Divide y Vencerás.
10000	0.004589
13600	0.011569
17200	0.013278
20800	0.006509
24400	0.008689
28000	0.008154
31600	0.010622
35200	0.012215
38800	0.015158
42400	0.016969
46000	0.018529
49600	0.014562
53200	0.024093
56800	0.017212
60400	0.024519
64000	0.019905
67600	0.031842
71200	0.032872
74800	0.035738
78400	0.033637
82000	0.033637
85600	0.034262
89200	0.035241
92800	0.030205
96400	0.031015
100000	0.032339

6.4.2. Gráfica de Eficiencia Empírica.



6.4.3. Gráfica Comparativa entre Divide y Vencerás y Fuerza Bruta.



6.5. Eficiencia híbrida.

Ajuste de la función

$f(x) = a * x^2 + b * x + c$

DyV, $g(x) = log_2(x) * c' + d$

Final set of parameters	Asymptotic Standard Error
$a = -7{,}19307e^{-13}$	$+/- 1{,}22e^{-12}(169{,}6\,\%)$
$b = 4{,}2327e^{-07}$	$+/- 1{,}373e^{-07}(32{,}45\,\%)$
$c = 0{,}00060036$	$+/- 0{,}003329(554{,}6\,\%)$
$c' = 0{,}00979785$	$+/- 0{,}001071(10{,}93\,\%)$
$d = -0{,}130804$	$+/- 0{,}01664(12{,}72\,\%)$

Se puede observar que la función se ajusta bastante mejor a una función logarítmica que a una polinómica de segundo grado.