**Algoritmica 2016**

Práctica 2: Algoritmos Divide y Vencerás

**Autores:**

Alejandro Campoy Nieves

David Criado Ramón

Nour Eddine El Alaoui

Luis Gallego Quero

1. **Introducción :**

En esta práctica, haremos una comparación entre el uso de un algoritmo divide y vencerás y otro de fuerza bruta para resolver un problema que se basa en tener k vectores ordenados (de menor a mayor), cada uno con n elementos, y queremos combinarlos en un único vector ordenado (con k\*n elementos).

El método divide y vencerás (DYV) está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que estos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original. Normalmente esta técnica proporciona una forma natural de diseñar algoritmos eficientes.

**2. Algoritmo divide venceras : basado en Mergesort**

Una forma de optimizar la ordenación de una secuencia con muchos elementos consiste en partir la secuencia en varias secuencias más pequeñas, ordenarlas de forma separada y después hacer una fusión en una secuencia única.

El algoritmo que hemos usado está basado en Mergesort. Básicamente hemos juntado todos los vectores generados en un solo vector provisional y después hacemos la llamada a la función mergesort, que llama mergesort (vector, 0, tamaño,umbral) que es una función recursiva que va a dividir cada vez en dos vectores más pequeños hasta llegar a vectores de tamaño menor que el umbral con lo que se realiza el algoritmo de ordenación por inserción. Al final se hace la fusión de los vectores resultados (aux\_1[l] y aux\_2[l]) haciendo una ordenación final comparando para cada índex el elemento y poniéndolo en un vector final.

**2.1 Funciones usadas en el algoritmo divide y vencerás :**

**void insercion(vector<int>& v, int inicio, int fin) {**

**for (int i = inicio + 1; i < fin; ++i)**

**for (int j = i; v[j] < v[j-1] && j > 0; --j)**

**swap(v[j], v[j-1]);**

**}**

**void juntar(vector<int>& v, int inicio, int fin, const vector<int>& aux\_1, const vector<int>& aux\_2) {**

**for (int i = inicio, j = 0, k = 0; i < fin; ++i)**

**if (aux\_1[j] < aux\_2[k]) {**

**v[i] = aux\_1[j];**

**j++;**

**}**

**else {**

**v[i] = aux\_2[k];**

**k++;**

**}**

**}**

**void mergesort(vector<int>& v, int inicio, int fin, int umbral) {**

**if (fin - inicio < umbral)**

**insercion(v,inicio, fin);**

**else {**

**int mitad = (fin - inicio)/2;**

**vector<int> aux\_1;**

**vector<int> aux\_2;**

**aux\_1.reserve(mitad - inicio + 1);**

**aux\_2.reserve(fin - mitad + 1);**

**for (int l = 0, l2 = inicio; l < mitad; ++l, ++l2)**

**aux\_1.push\_back(v[l2]);**

**aux\_1[mitad] = INT\_MAX;**

**for (int l = 0, l2 = mitad; l < fin - mitad; ++l, ++l2)**

**aux\_2.push\_back(v[l2]);**

**aux\_2[fin-mitad] = INT\_MAX;**

**mergesort(aux\_1, 0, mitad, umbral);**

**mergesort(aux\_2, 0, fin - mitad, umbral);**

**juntar(v, inicio, fin, aux\_1, aux\_2);**

**}**

**}**

**vector<int> dyv(const vector<vector<int>>& datos) {**

**vector<int> salida;**

**int k = datos.size();**

**int n = (k > 0) ? datos[0].size() : 0;**

**// Llenamos el vector**

**salida.reserve(n\*k);**

**for (auto& v : datos)**

**for (auto d: v)**

**salida.push\_back(d);**

**// Ordenamos(mergesort)**

**mergesort(salida,0,salida.size(),100);**

**return salida;**

**}**

**2.2 Cálculo de eficiencia Empírica con el algoritmo divide y Vencerás :**

* Hemos utilizado un tamaño de 30000 entradas tomando datos de 1000 en 1000 para 4 vectores iniciales.
* las pruebas se realizaron en una computadora con las siguientes características :

Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz 4GB RAM

* la macro utilizada para estudiar la eficiencia empírica e híbrida :
* Macro DivideYVencerás

**#!/bin/bash**

**echo "" > dyv.dat**

**i=1000**

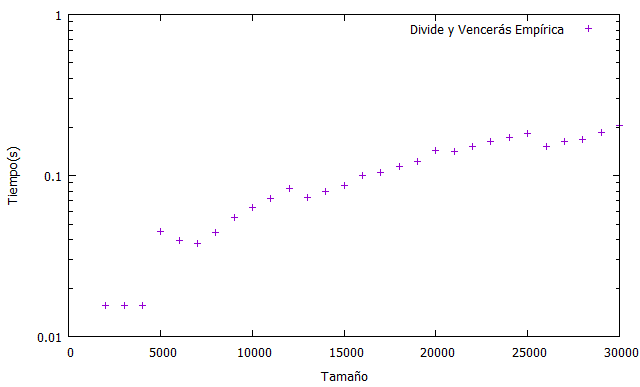
**while [ $i -lt 30000 ]**

**do**

**./dyv 4 $i >> dyv.dat**

**i=$(($i+1000))**

**done**



**Figura 1 : Gráfico de eficiencia Empírica - Algoritmo Divide y Vencerás**

**2.3 Cálculo de eficiencia Híbrida con el algoritmo divide y Vencerás :**

* Para ajustar las funciones en gnuplot y obtener las constantes ocultas definimos una función parametrizada:

**gnuplot> f(x)=a1\*x\*log(x)**

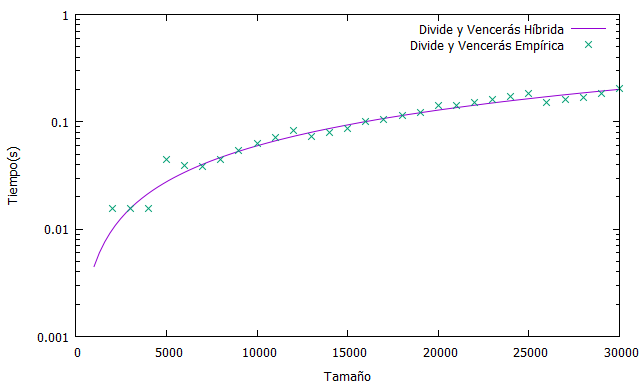
**gnuplot> fit f(x) 'dyv.dat' via a1**

Final set of parameters Asymptotic Standard Error

================== ======================

a1 = 6.49746e-007 +/- 1.017e-008 (1.565%)

* Viendo el gráfico que hemos generado, podemos comprobar que hay un ajuste aproximado, lo que significa que la información teórica y las pruebas empíricas coinciden.



**Figura 2 : Gráfico de eficiencia Híbrida - Algoritmo Divide y Vencerás**

**3. Algoritmo Fuerza Bruta : basado en Burbuja**

Los algoritmos de ordenación por fuerza bruta hacen paradas sucesivas sobre la secuencia, comparando elementos que se encuentran a una distinta posición, los cuales si estuvieran desordenados son intercambiados.

La idea del algoritmo burbuja es recorrer el vector varias veces, y para cada pasada ver cual es el menor elemento de la secuencia. La complejidad del algoritmo es de orden cuadrática, por esta razón no es recomendable para programas que necesiten de velocidad o ejecuten una enorme cantidad de datos.

**3.1 Funciones usadas en el algoritmo por Fuerza Bruta :**

**void burbuja(vector<int>&v) {**

**int n = v.size() - 1;**

**for (int i = 0; i < n ; ++i)**

**for (int j = n; j > i; --j)**

**if (v[j] < v[j-1])**

**swap(v[j] , v[j-1]);**

**}**

**vector<int> clasico(const vector<vector<int>>& datos) {**

**vector<int> salida;**

**int k = datos.size();**

**int n = (k > 0) ? datos[0].size() : 0;**

**salida.reserve(n\*k); // Llenamos el vector**

**for (auto& v : datos)**

**for (auto d: v)**

**salida.push\_back(d);**

**// Ordenamos (Burbuja)**

**burbuja(salida);**

**return salida;**

**}**

**3.2 Cálculo de eficiencia Empírica con el algoritmo Fuerza Bruta :**

* Hemos utilizado un tamaño de 30000 entradas tomando datos de 1000 en 1000 para 4 vectores iniciales.
* Las pruebas se realizaron en una computadora con las siguientes características :

Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz 4GB RAM

* La macro utilizada para estudiar la eficiencia empírica e híbrida :

**#!/bin/bash**

**echo "" > fuerzabruta.dat**

**i=1000**

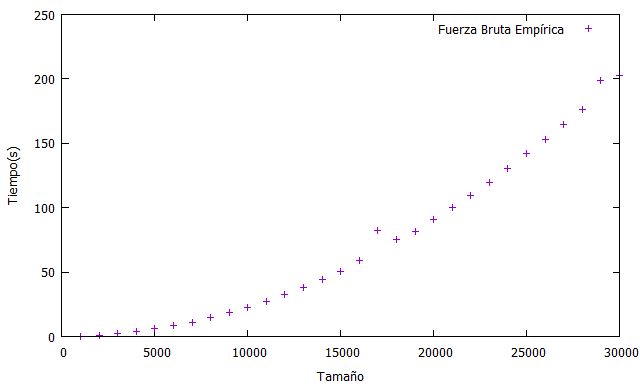
**while [ $i -lt 30000 ]**

**do**

**./fuerzabruta 4 $i >> fuerzabruta.dat**

**i=$(($i+1000))**

**done**



**Figura 3 : Gráfico de eficiencia Empírica - Algoritmo Fuerza bruta**

* Para ajustar las funciones en gnuplot y obtener las constantes ocultas definimos una función parametrizada:

**gnuplot> f(x)=a1\*x\*x+a2\*x+a3**

**gnuplot> fit f(x) 'fuerzabruta.dat' via a1,a2,a3**

Final set of parameters Asymptotic Standard Error

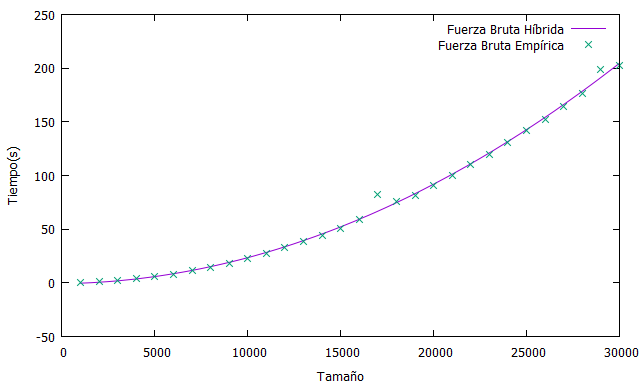
==================== ======================

a1 = 2.21073e-007 +/- 9.578e-009 (4.333%)

a2 = 0.000212506 +/- 0.000306 (144%)

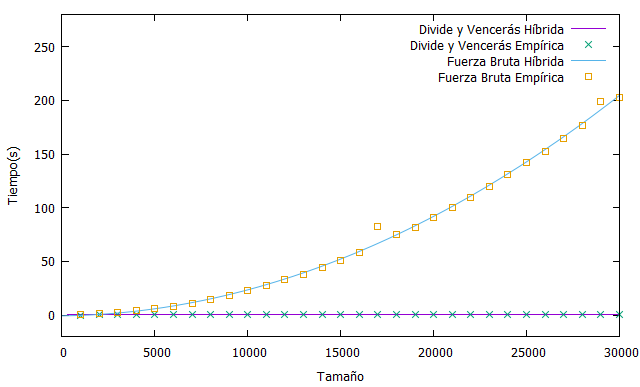
a3 = -0.740793 +/- 2.058 (277.8%)

* Viendo la gráfica que hemos generado, podemos comprobar que hay un ajuste muy aproximado, lo que significa que la información teórica y las pruebas empíricas coinciden.



**Figura 4 : Gráfico de eficiencia Híbrida - Algoritmo Fuerza bruta**

**4. Comparación entre el algoritmo Fuerza Bruta y Divide Vencerás :**



**Figura 5 : gráfico comparativo de los dos Algoritmos**

* Viendo la figura anterior, podemos observar que el algoritmo divide y vencerás es mucho mejor que el algoritmo de fuerza bruta, ya que según los resultados el algoritmo divide y vencerás ha tenido **orden de complejidad O(n\*log(n))**, mientras que el algoritmo fuerza bruta una **orden de complejidad O(n2).**

**Como conclusión podemos decir que el algoritmo divide y vencerás basado en Mergesort ha obtenido una mejor eficiencia que el algoritmo de fuerza bruta.**

* Tabla de datos Recogidos :

|  |  |  |
| --- | --- | --- |
| Tamaño | Tiempo Divide y Vencerás | Tiempo Fuerza Bruta |
| 1000 | 0 | 0.236796 |
| 2000 | 0.0156258 | 1.08539 |
| 3000 | 0.0156264 | 2.05506 |
| 4000 | 0.0156269 | 3.85893 |
| 5000 | 0.0450381 | 5.94503 |
| 6000 | 0.0395267 | 8.2027 |
| 7000 | 0.0380254 | 11.0124 |
| 8000 | 0.0445305 | 14.4518 |
| 9000 | 0.0545361 | 18.3081 |
| 10000 | 0.0630416 | 22.5511 |
| 11000 | 0.0715481 | 27.2799 |
| 12000 | 0.0830555 | 33.0072 |
| 13000 | 0.0730488 | 38.088 |
| 14000 | 0.0800535 | 44.1586 |
| 15000 | 0.0870581 | 50.6827 |
| 16000 | 0.100267 | 58.6421 |
| 17000 | 0.10507 | 82.4228 |
| 18000 | 0.113574 | 75.3837 |
| 19000 | 0.122581 | 81.554 |
| 20000 | 0.143096 | 90.8163 |
| 21000 | 0.141093 | 99.871 |
| 22000 | 0.152102 | 109.802 |
| 23000 | 0.161608 | 119.769 |
| 24000 | 0.172116 | 130.529 |
| 25000 | 0.183122 | 141.945 |
| 26000 | 0.152602 | 152.61 |
| 27000 | 0.161608 | 164.701 |
| 28000 | 0.168112 | 176.369 |
| 29000 | 0.184605 | 198.63 |
| 30000 | 0.205413 | 202.866 |