Memoria Práctica 2: Algoritmos Divide y Vencerás.

Realizado por:

Jose Manuel Rodriguez Calvo

Jesús Baeza Álvarez

Jorge García Moreno

Alejandro Sánchez Molina

David López Maldonado

Índice.

Tareas realizadas:

1.-Introducción.

2.-Algoritmo sin Divide y Vencerás.

3.-Algoritmo con Divide y Vencerás.

4.-Comparación de ambos algoritmos.

5.-Prestaciones PC utilizado

1.- Introducción.

En esta práctica hemos implementado dos algoritmos para la resolución del problema que nos ha tocado, uno de ellos lo hemos resuelto implementando un algoritmo sin hacer uso de la técnica Divide y Vencerás , el otro sin embargo hace uso de esta técnica. Tras implementar estos dos algoritmos en los que hemo hecho uso del generador aleatorio subido a decsai para poder rellenar la matriz con elementos aleatorios, hemos creado un macro con el que nos hemos ayudado para obtener los tiempos de cada uno de los algoritmos, y gracias al software gnuplot hemos realizado unas gráficas en base a los datos obtenidos, también comparamos ambas gráficas y sacamos conclusiones.

El enunciado propuesto a resolver era:

***“Mezclando k vectores ordenados”***

Se tienen k vectores ordenados (de menor a mayor), cada uno con n elementos, y queremos

combinarlos en un único vector ordenado (con kn elementos). Una posible alternativa consiste

en, utilizando un algoritmo clásico, mezclar los dos primeros vectores, posteriormente mezclar

el resultado con el tercero, y ası́ sucesivamente.

2.Algoritmo de ordenación sin utilizar Divide y Vencerás.

Éste algoritmo se basa en que disponiendo de un número k de vectores ordenados (de menor a mayor), los cuales, cada uno de ellos tiene un número n de elementos. Entonces mezcla los dos primeros vectores, posteriormente mezcal el resultado con el tercero, y así sucesivamente hasta obtener el resultado de todos los elementos de los vectores ordenado en un solo vector.

Código:

int \* resultado = new int[N];

int contador=0;

clock\_t tantes, tdespues;

tantes = clock();

for(int i=0;i<k;i++){

for(int j=0;j<n;j++){

//if(!esta(resultado,N,T[i][j]))

resultado[contador] = T[i][j];

contador++;

}

mergesort(resultado,N);

}

tdespues = clock();

for (int i=0; i < N; i++){

//cout << resultado[i] << " " ;

}

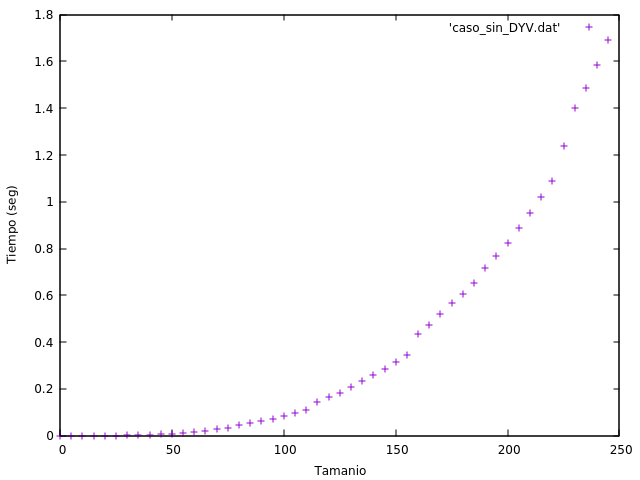
cout << (double)(tdespues-tantes) /CLOCKS\_PER\_SEC<<endl;

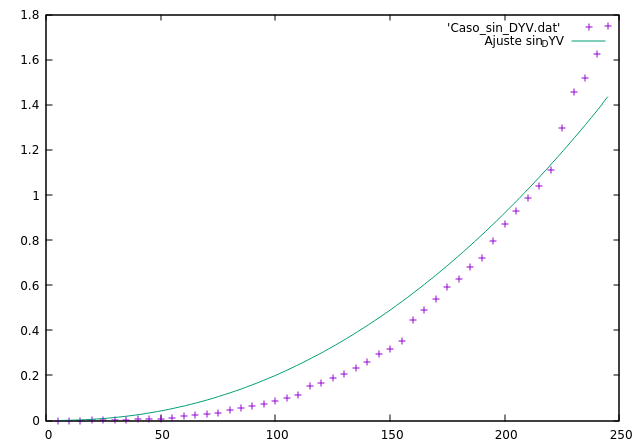
En base a el código que se nos ha proporcionado para la creación de un número determinado de vectores de un tamaño también determinado, hemos empleado este trozo de código para poder ordenar cada vector con el resultante de la ordenación anterior.

Asimismo, hemos creado dos variables del tipo clock\_t llamadas tantes y tdespues para poder contar el tiempo que tarda en realizar la ordenación en conjunto de todos los vectores.

Para el análisis híbrido hemos probado funciones y una f(x)=ax² \* log(x) se ajusta bastante bien, dando un valor de a =4.34302e-06 con error de 2.635%

Gráfica:  
Aquí tenemos una gráfica para la obtención de los vectores ordenados para el tiempo que tarda en realizarlo.





3.- Algoritmo de ordenación utilizando Divide y Vencerás.

Para la realización del algoritmo mediante la utilización del método Divide y vencerás, hemos creado un vector para guardar las filas de T, que se encuentra realizado en el algoritmo inicial entregado para la práctica como un puntero. Y finalmente lo pasamos a matriz que es un vector de vectores de int.

Después llamamos al método mezclaVectores y se lo metemos a la matriz de vectores4, que hemos creado llamada resultado. Si el tamaño es menor o igual que 1, devolvemos directamente T, ya que ya estaría ordenado, sino, dividimos la matriz resultado en dos mitades, denominadas mitadsuperior y mitadinferior.

En el caso de que el número de vectores fuese par, se recoge desde el principio hasta la mitad, y se la metemos en la mitad superior, luego se rellena desde la mitad hasta el final de la matriz. En el caso de que no fuese par, a la mitad superior le corresponde desde el principio hasta la mitad + 1, y posteriormente a la mitadinferior le asignamos desde la mitad hasta el final de resultado.

En ambos casos a las dos mitades le hacemos el mezclavectores, el método que hemos implementado para emplear el mergesort con dos vectores.

Cuando ya el tamaño de mitadsuperior y mitad inferior es menor o igual a 1, se le asigna a resultado la llamada del método: Mezclar(mitadSup, mitadInf).

El método que hemos denominado mezclar, reserva inicialmente tanta memoria como necesita para mezclar las dos matrices.

Siendo tamav1 es el número de elementos de la primera mitad de la matriz dividida, tama2 el número de elementos de la segunda mitad de la matriz dividida, vamos insertando en resultado los elementos de manera que primero mete el menor, y va avanzando el puntero del número metido. Cuando acaba, inserta los números que faltan por recorrer en la segunda ordenadamente.

Para el análisis híbrido hemos probado funciones y una f(x)=ax² se ajusta bastante bien, dando un valor de a = 6.82137-e07 con error de 2,313%

Código:

vector<vector<int> > Mezclar(vector<vector<int> > v1 , vector<vector<int> > v2){

int contadorv1=0,contadorv2=0;

int fil1=0,fil2=0,col1=0,col2=0;

int fila=0,col=0;

int tamav1=v1[0].size()\*v1.size();

int tamav2=v2[0].size()\*v2.size();

vector<vector<int> > resultado;

resultado.resize(v1.size()+v2.size());

for(int i=0 ; i<v1.size()+v2.size() ; i++){

resultado[i].resize(v1[0].size());

}

while(contadorv1 < tamav1 && contadorv2 < tamav2 ){

if(v1[fil1][col1] < v2[fil2][col2]){

resultado[fila][col]=v1[fil1][col1];

contadorv1++;

col++;

col1++;

if(col == v1[fil1].size()){

col=0;

fila++;

}

if(col1 == v1[fil1].size()){

fil1++;

col1=0;

}

}

else{

resultado[fila][col]=v2[fil2][col2];

contadorv2++;

col++;

col2++;

if(col == v2[fil2].size()){

col=0;

fila++;

}

if(col2 == v2[fil2].size()){

fil2++;

col2=0;

}

}

}

if(contadorv1!=tamav1){

for(int i=contadorv1 ; i<tamav1 ; i++){

resultado[fila][col]=v1[fil1][col1];

col++;

col1++;

if(col == v1[fil1].size()){

col=0;

fila++;

}

if(col1 == v1[fil1].size()){

fil1++;

col1=0;

}

}

}

else if (contadorv2!=tamav2){

for(int i=contadorv2 ; i<tamav2 ; i++){

resultado[fila][col]=v2[fil2][col2];

col++;

col2++;

if(col == v2[fil2].size()){

col=0;

fila++;

}

if(col2 == v2[fil2].size()){

fil2++;

col2=0;

}

}

}

return resultado;

}

vector<vector<int> > mezclarVectores(vector<vector<int> > &T){

if(T.size()>1){

int medio=0;

vector<vector<int> > mitadSup;

vector<vector<int> > mitadInf;

if(T.size()%2 == 0){

medio=(T.size()/2);

for(int i =0 ; i<medio ; i++){

mitadSup.push\_back(T[i]);

}

for(int i =medio ; i<T.size() ; i++){

mitadInf.push\_back(T[i]);

}

mitadSup=mezclarVectores(mitadSup);

mitadInf=mezclarVectores(mitadInf);

}

else{

medio=T.size()/2;

for(int i =0 ; i<medio+1 ; i++){

mitadSup.push\_back(T[i]);

}

for(int i =medio+1 ; i<T.size() ; i++){

mitadInf.push\_back(T[i]);

}

mitadSup=mezclarVectores(mitadSup);

mitadInf=mezclarVectores(mitadInf);

}

vector<vector<int> > resultado;

resultado=Mezclar(mitadSup,mitadInf);

return resultado;

}

else{

return T;

}

}

int main(int argc, char \* argv[]){

//genera la matriz random

vector<vector<int> > matriz;

vector<int> Aux;

delete [] aux;

for (int i=0; i<k; i++) {

for (int j=0; j<n; j++){

Aux.push\_back(T[i][j]);

}

matriz.push\_back(Aux);

Aux.clear();

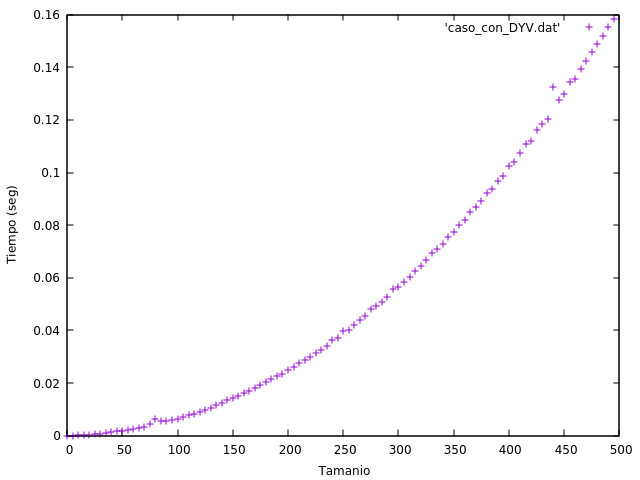
}

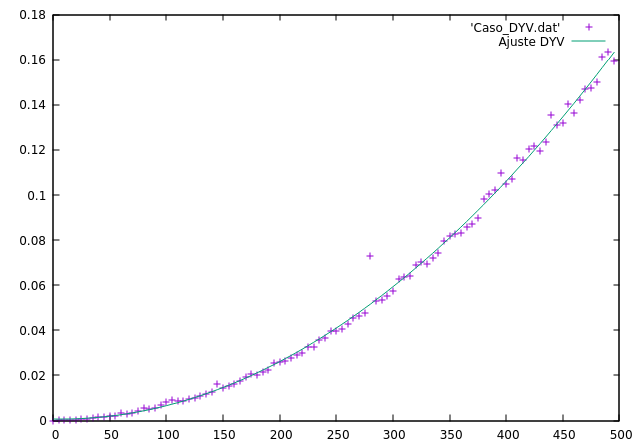
vector<vector<int> > resultado;

resultado=mezclarVectores(matriz);

}

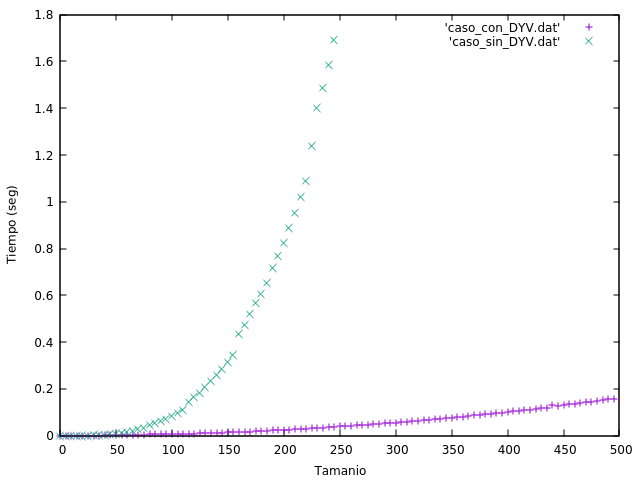
Gráfica:





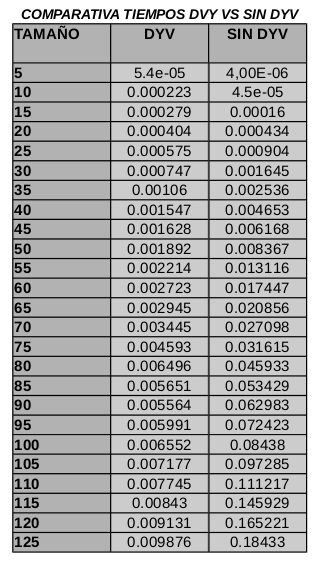
4.- Comparación de ambos algoritmos.

Aquí tenemos una representación gráfica para los valores de ambos algoritmos, de tal manera que podemos ver la diferencia de tiempo entre ámbos en función de los datos del tamaño de los vectores:



También presentamos una tabla comparando con el tiempo de ejecución de ambos:

Hay que tener en cuenta que en la toma de datos hemos modificado número de vectores y cantidad de datos en la misma medida con el mismo número(‘./MezclaVectores i i’ con i de 5 en 5).



5.- Prestaciones PC utilizado

Para la medida de tiempos hemos utilizado un PC equipado con un Intel i3-5005U CPU @ 2.00GHz, utilizando Ubuntu 16.04. Para la toma de tiempo el PC únicamente estaba trabajando en dicha tarea.