Practica Opcional 2
Estructura de Datos y Algoritmos

Búsqueda del umbral óptimo en un algoritmo Divide y Vencerás (DYV)

1. NOMBRE DE LOS INTEGRANTES DEL GRUPO:

- Daniel Serrano Torres
- Juan Luis Pérez Valbuena

2. INTRODUCCIÓN

En esta práctica trata de abordar un problema computacional complejo utilizando técnicas algorítmicas llamadas: divide y vencerás.

El problema trata de un espacio de dos dimensiones con puntos en él, por tanto son representados con coordenadas (x, y). Se busca hallar el par de puntos más cercanos entre sí.

2.1 ALGORITMO DIVIDE Y VENCERÁS (DV)

El algoritmo divide y vencerás para esta práctica se ha planteado de la siguiente manera; dado que la distancia mínima entre dos puntos del espacio no tiene por que ser estrictamente en términos de su coordenada *x* o *y* exclusivamente, no podemos solo medir los puntos por alguna de estas sino que debemos hacerlo teniendo en cuenta a ambas.

Teniendo en cuenta esto lo primero que se realiza es una ordenación de los puntos por la coordenada *x* y se divide el problema por la mitad, tratando cada mitad de la misma forma que el problema original, es decir, se dividirán por la mitad respectivamente hasta obtener una solución trivial (caso base) que nos permita resolver lo espacios mayores de puntos, obteniendo así el grupo de puntos más cercanos entre sí respecto de las coordenadas *x*.

3. DESCRIPCIÓN DE LA DIVISIÓN DEL CÓDIGO EN MÓDULOS

3.1 ARCHIVO *MAIN.CPP*

int main() // Funcion Principal del programa

void inicioAlgoritmoDirecto(Lista<Punto> &lista_de_puntos) // Implementación publica del algoritmo Directo con cuenta de tiempos

Solucion inicioAlgoritmoDYV(Lista<Punto> &lista_de_puntos) // Implementación publica del algoritmo DYV con cuenta de tiempos

3.2 ARCHIVO MEZCLA.CPP

void mezclam(Punto array_puntos[], int a, int m, int b,Comparador menor); // Implementacion privada del algoritmo mergesort que mezcla dos listas

bool menorX(const Punto &p1, const Punto &p2); // Función que compara dos puntos por su abscisa. Su definición es compatible con el tipo Comparador

bool menorigualX(const Punto &p1, const Punto &p2); // Función que compara dos puntos por su abscisa. Su definición es compatible con el tipo Comparador

bool menorY(const Punto &p1, const Punto &p2);

Lista<Punto> mezcla(Lista<Punto> &l1, Lista<Punto> &l2, Comparador menor); // Función de mezcla que recibe la función que compara dos puntos

void mergeSort(Punto array_puntos[], int a, int b); // Implementación privada del algoritmo mergesort

void OrdenacionMergeSort(Lista<Punto> &l,Comparador menor); // Operación publica del algoritmo mergesort

void deListaAarray(Lista<Punto> &l, Punto array_puntos[]);//Transforma una lista a un array

void deArrayALista(Lista<Punto> &l, Punto array_puntos[]); // Transforma un array a lista

void partirListam(Lista<Punto> &original, int medio , Lista<Punto> &izquierda , Lista<Punto> &derecha); // Dada una lista original y un punto medio , parte en dos listas dejando 0-medio elementos en izquierda y medio+1 hasta n elementos en derecha

3.3 ARCHIVO DYV.CPP

Solucion solucionDirecta(Lista<Punto> &puntos, int n); // Dada una lista con dos o tres puntos , devuelve los dos puntos más cercanos , su distancia y una lista con los puntos ordenados crecientemente por su coordenada y.

Lista<Punto> merge(Lista<Punto> &l1, Lista<Punto> &l2); // Dadas dos listas de puntos ordenadas por la coordenada y ,devuelve otra lista con su mezcla ordenada también por la coordenada y.

Lista<Punto> filtraBanda(Lista<Punto> &l, double d, double x); // Dada una lista l de puntos , una distancia d y una abscisa x , devuelve la lista de los puntos de l cuya abscisa diste a lo sumo d de x en valor absoluto

void recorreBanda(Lista<Punto> &l, Punto &p1, Punto &p2, double &d); // Recorre una lista l de puntos , comparando cada uno con los 7 siguientes si los hubiera . Devuelve el par de puntos con menor distancia, y dicha distancia. Si l tiene un punto o ninguno, devuelve una distancia +infinito.

Solucion eligeMinimo(const Solucion &s1, const Solucion &s2,const Punto &p1, const Punto &p2, double d); // Dadas tres soluciones , cada una consistente en un par de puntos y su distancia , devuelve el par más cercano de los tres y su distancia .

Solucion parMasCercano(Lista<Punto> &puntos, int n,int umbral); //Llamada a función del algoritmo DYV

3.4 ARCHIVO FAUX.CPP

bool iguales(Solucion &s,Solucion &s1): Función de testing utilizada durante el desarrollo de la practica.

Lista<Punto> generarListaDePuntos(int n); Rellena una lista con N puntos aleatorios

void imprimeListadePuntos(Lista<Punto> &l); //Imprime una lista de puntos de forma adecuada

void imprimeUnicoPunto(const Punto &p1); //Función que muestra un punto de forma adecuada -- Auxiliar

3.5 ARCHIVO UMBRAL.CPP

bool directaMenor(float tiempoDirecto, float tiempoDV);

Función que indica si el tiempo del algoritmo directo es menor que el del algoritmo de divide y venceras.

int umbral(int numero_de_puntos);

Función que obtiene el número de puntos a partir del cual el algoritmo directo es más eficiente

4. INSTRUCCIONES PARA COMPILAR EL CÓDIGO

Se ha utilizado un entorno de desarrollo integrado como NetBeans o Xcode, etc. De todas maneras puede ser compilado mediante el uso del compilador g++ .

Las únicas dependencias son las propias al uso de las librerías internas del propio lenguaje C++, las cabeceras creadas para la práctica, así como las de los TADs utilizados en el curso.

5. CÁLCULO DEL UMBRAL EXPERIMENTAL

El modo empleado para obtener el umbral donde el algoritmo cuadrático en más eficiente que el algoritmo de divide y vencerás ha sido experimental, se ha comparado el tiempo que tardan ambos algoritmos en resolver el en mismo tamaño de problema.

Partiendo de un tamaño fijo (bastante alto) se van probando los distintos tamaños de problema hasta encontrar el punto mencionado antes.

Esta prueba ha sido realizada en una plataforma con las siguientes características, dando como resultado un umbral de **116** puntos:

S.O.: Mac OS X 10.8.3

CPU: Intel core 2 duo 2,5 GHz

RAM: 2 GB

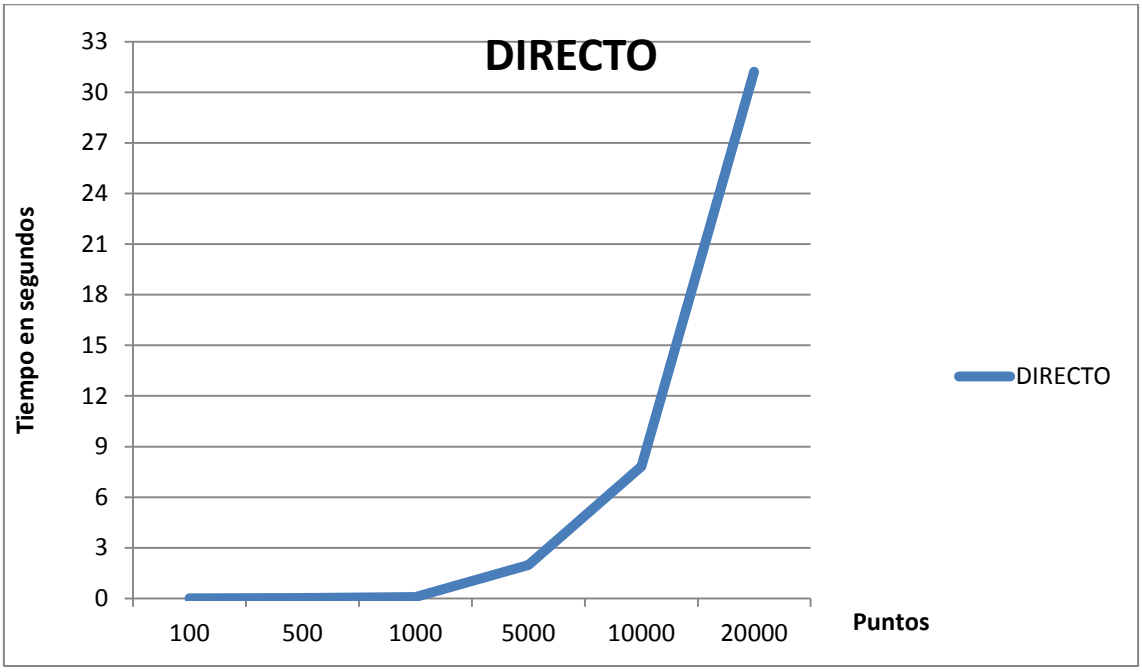
Se incluye la especificación de la plataforma donde se ha obtenido el umbral por que puede obtenerse otro valor en una plataforma diferente.

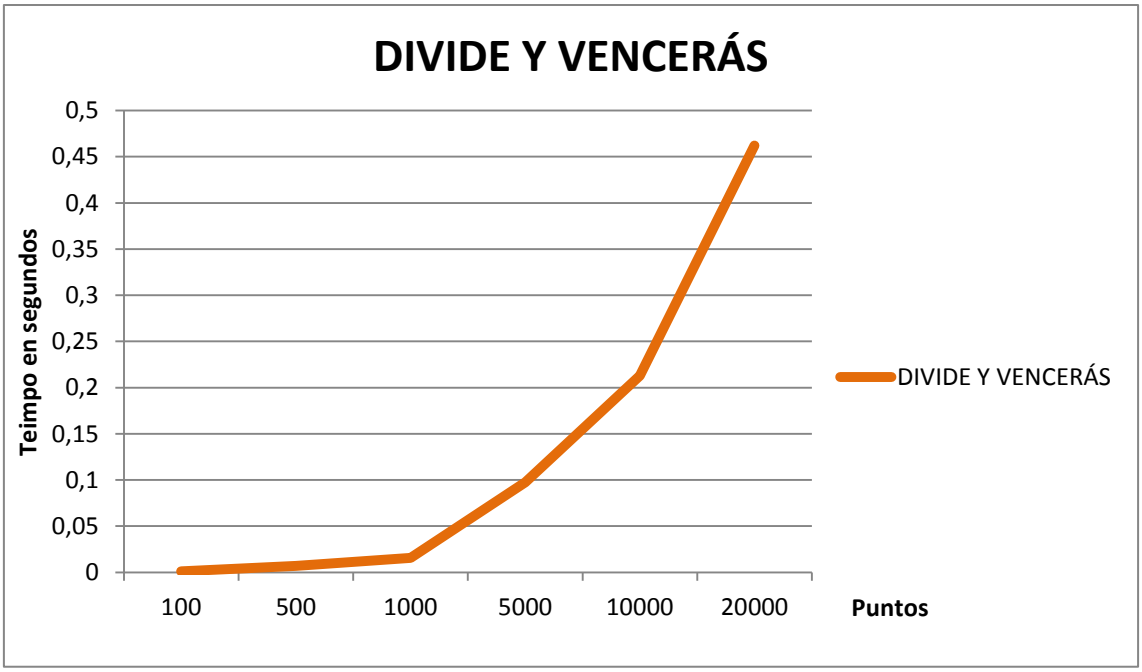
5.1 MÉTODO EMPLEADO PARA OBTENER LOS TIEMPOS Y GENERAR LAS GRAFICAS

Se ha utilizado un cálculo del tiempo tardado por cada algoritmo, y con los datos obtenidos por el programa se han generado las gráficas en una hoja de cálculo.

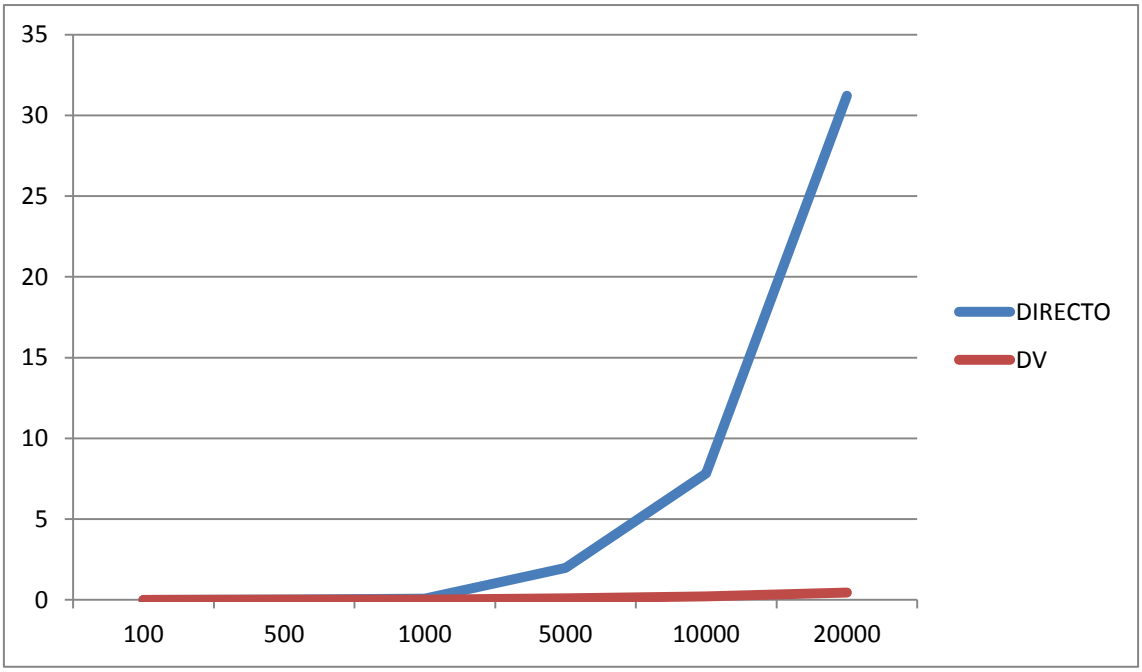
El método concreto de cálculo de tiempo que tarda cada algoritmo ha sido realizado en la implementación del programa, del modo exacto que se indica en el enunciado de esta. Utilizando la función ***clock()*** de C++ y haciendo la diferencia entre el tiempo inicial y el final.

Las gráficas siguientes corresponden a un estudio de 6 casos, con número de puntos: 100, 500, 1000, 5000, 10000, 20000 respectivamente. Todos los casos han sido realizados con ambos algoritmos, cuadrático (directo) y divide y vencerás (DV).





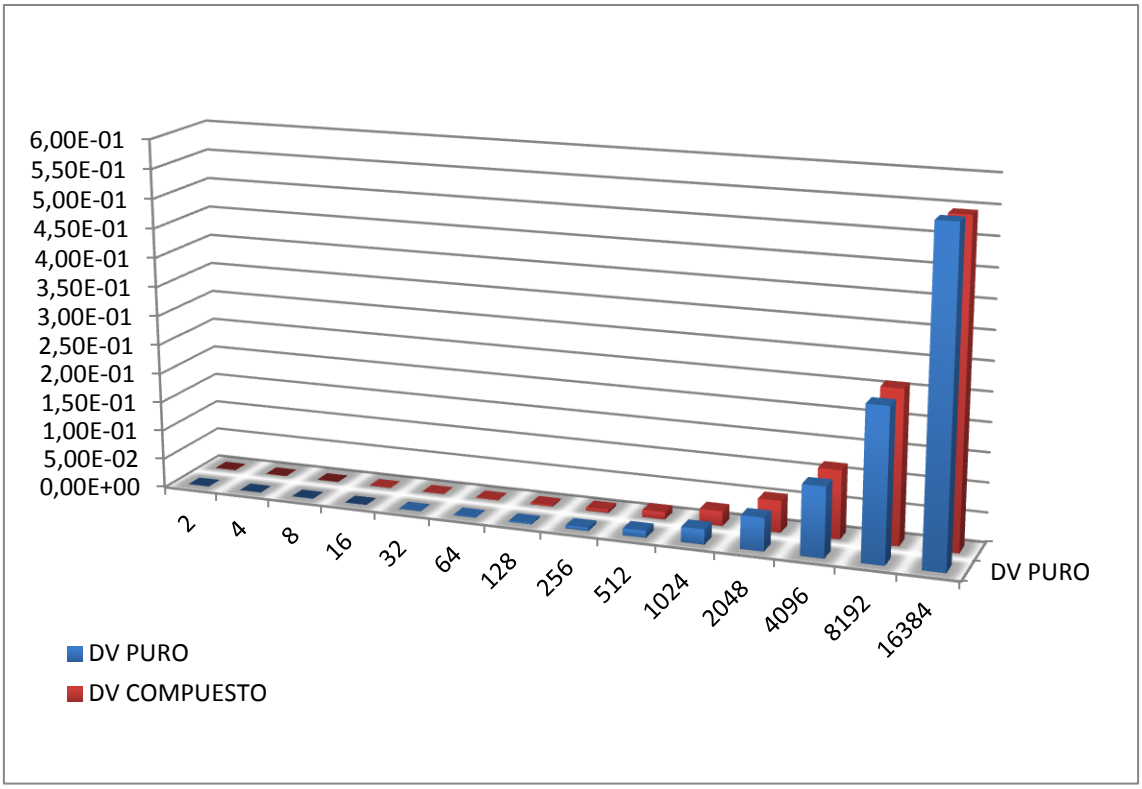
En esta última gráfica se representan ambos algoritmos junto para ver de una forma clara y sencilla las diferencias en cuestión del tiempo tardado por ambos para el mismo tamaño de problema (número de puntos).



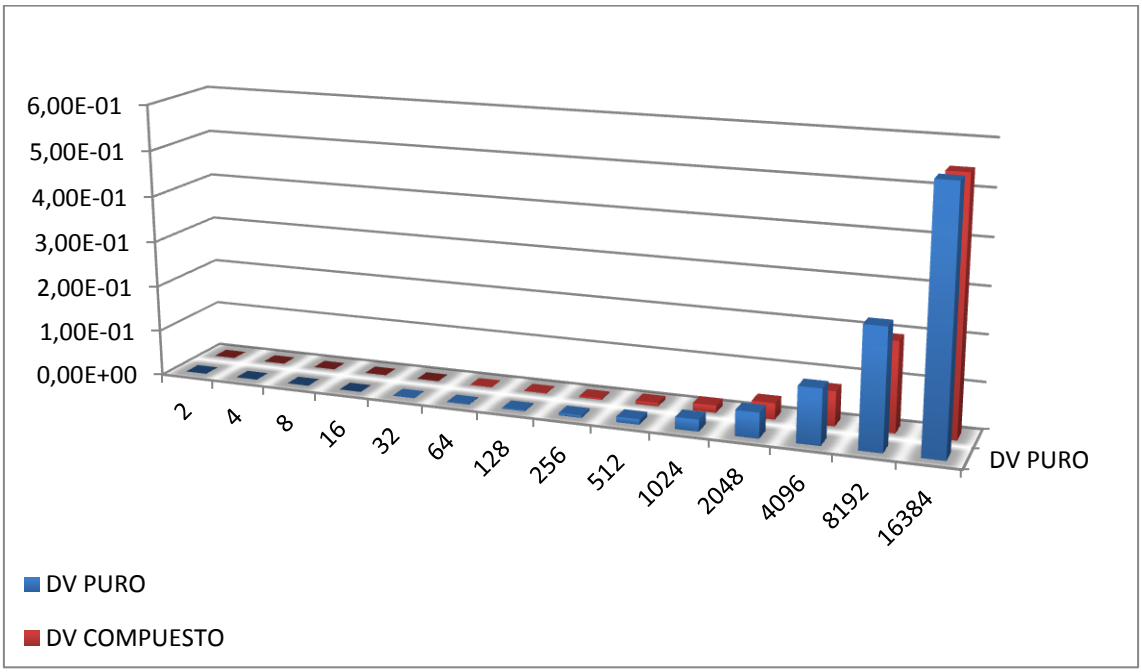
Se puede apreciar rápidamente que cuando el número de puntos del espacio del problema aumenta significativamente el algoritmo cuadrático es mucho más lento.

No se llega a apreciar en la gráfica el umbral donde el algoritmo cuadrático tiene menor coste que el algoritmo DV, por tanto, el umbral obtenido experimentalmente has sido 116 puntos.

La siguiente gráfica expone el resultado de comparar el algoritmo DV puro (sin usar el directo), y el DV compuesto (que usa el directo a partir de cierto número de puntos).



Esta gráfica muestra la relación de los mismos algoritmos y el mismo número de puntos pero con diferentes umbrales.



Umbrales usados en relación a los puntos:

Umbral	**Puntos**
10	2
20	4
30	8
40	16

50	32
60	64
70	128
80	256
90	512
100	1024
110	2048
120	4096
130	8192
140	16384