MEMORIA PRÁCTICA 3:

FUNDAMENTOS DE ANÁLISIS DE ALGORITMOS

1º INGENIERÍA INFORMÁTICA

ALBERTO GALÉ GONZÁLEZ

DNI: 49084021-C

Índice

1.- Algoritmos de Búsqueda:

1.1.- Introducción:

El objetivo principal de la práctica es estudiar empíricamente los distintos algoritmos de búsqueda (estudiados en la asignatura)

Entonces se estudiara experimentalmente el comportamiento temporal de un algoritmo de búsqueda comparándolo, además, con el de otros procedimientos de búsqueda. Estudiaremos y compararemos los algoritmos de búsqueda Secuencial, Binaria y Ternaria. Al finalizarlo, se dispondrá de los tiempos utilizados para ordenar un vector para poder elegir la alternativa más adecuada.

1.2. - Pseudocódigos y análisis de costes

Busqueda Secuencial

```
funcion BusquedaSecuencial(V[1..n];size,key:int)

   i←1;

   mientras (v[i]!= key && i<=size)

   {

       i←i+1;

   }

   fmientras
   si(v[i]==key)

       devolver i;   // posición donde se encuentra el elemento en el array

   sino

       devolver -1; // no se encuentra el elemento en el array

   fsi
ffuncion
```

Busqueda Binaria

```
 funcion BusquedaBinaria(V[1..n];primero,ultimo,clave:int)

    si primero ≥ ultimo  entonces
       devolver V[ultimo]=clave;

    fsi

    mitad ← ((ultimo - primero + 1) / 2);

    si clave = V[mitad] entonces
       devolver cierto;

    sino
```

```
        si clave < V[mitad] entonces

                devolver BusquedaBinaria (V, primero, mitad-1, clave);

        sino

                si clave > V[mitad] entonces

                   devolver BusquedaBinaria (V, mitad+1, ultimo, clave);

                fsi

        fsi

    fsi
ffuncion
```

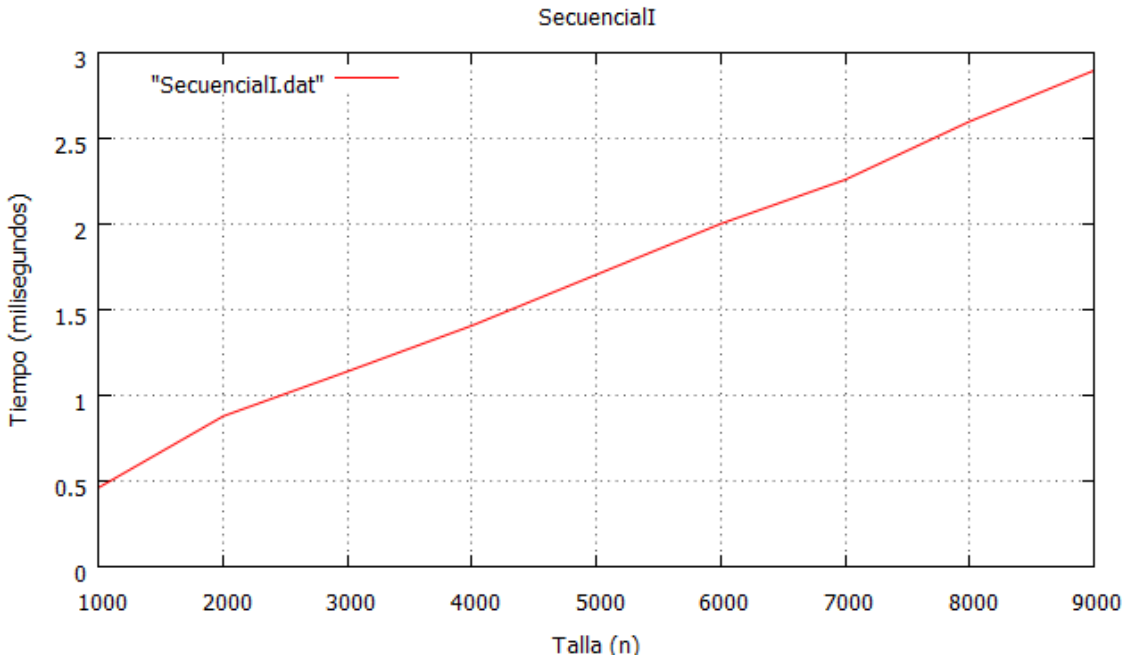
Busqueda Ternaria

```
funcion BusquedaTernaria(V[1..n];primero,ultimo,clave:int)

    si primero ≥ ultimo

       entonces devolver
       V[ultimo]=clave;

    fsi

    tercio ← ((ultimo - primero + 1) / 3);

    si clave = V[primero+tercio]
       entonces devolver cierto;

    sino

       si clave < V[primero+tercio] entonces

                devolver BusquedaTernaria (V, primero, primero+tercio-1, clave);

       sino

                si clave = V[ultimo-tercio]
                       entonces devolver
                       cierto;

                sino

                       si clave < V[ultimo-tercio] entonces

                          devolver BusquedaTernaria (V, primero+tercio+1, ultimo-
                       tercio-1, clave);

                       sino

                              devolver BusquedaTernaria (V, ultimo-tercio+1, ultimo,
                              clave);

                       fsi

                fsi

       fsi

fsifuncion
```

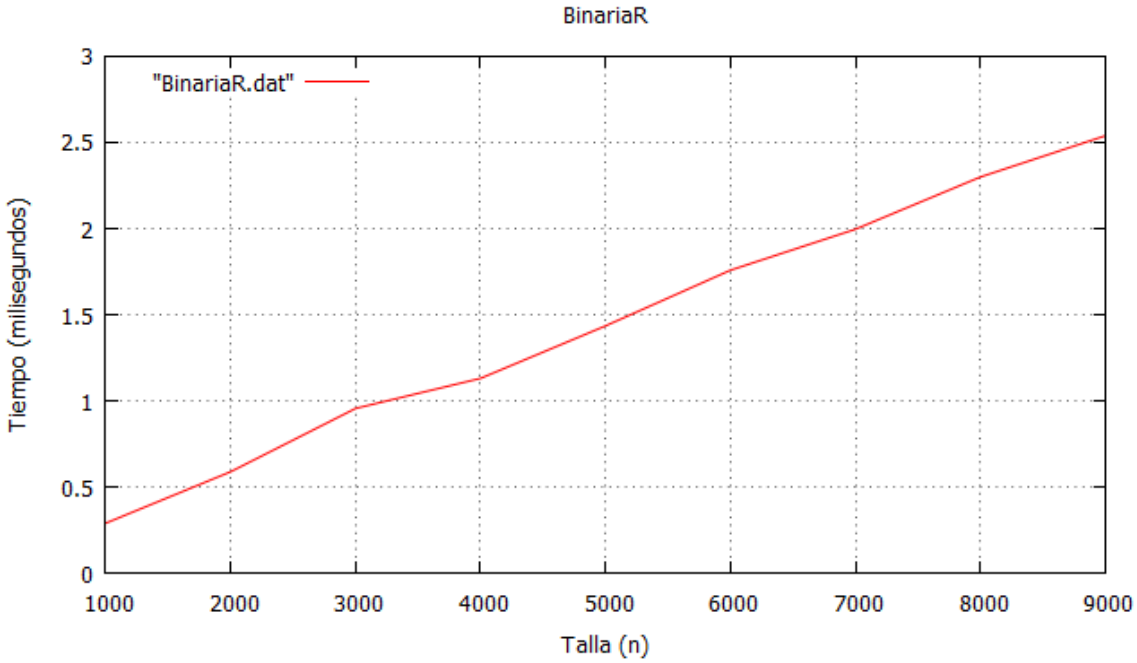
1.3.- Gráficas de costes

Primero se mostrara las gráficas de costes del caso medio:

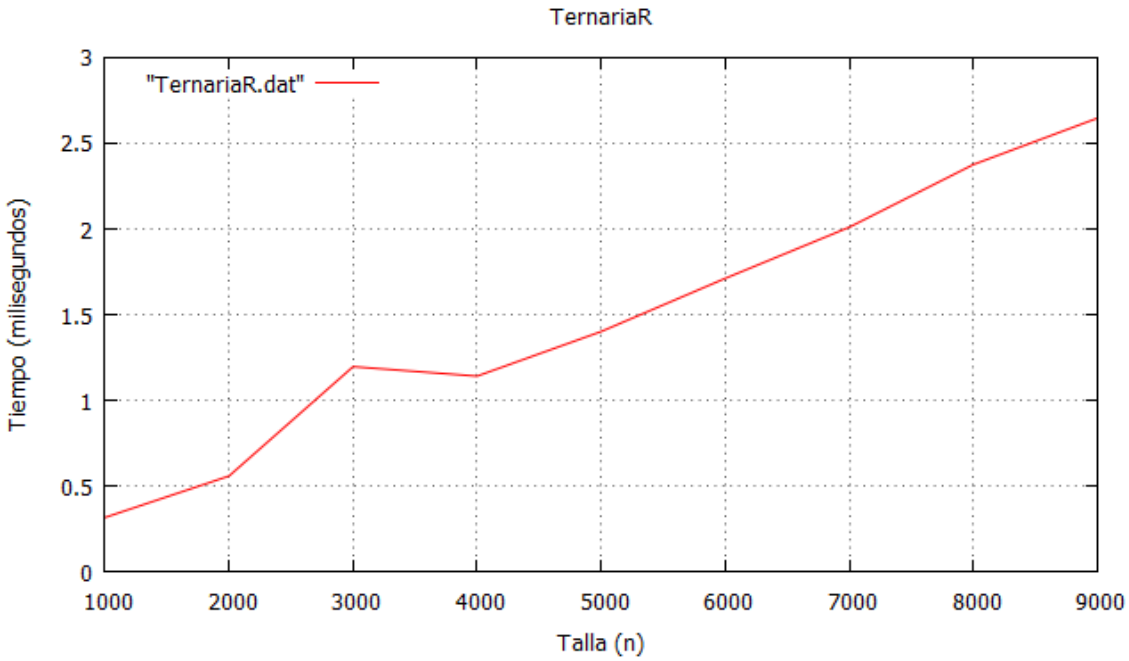
Secuencial



Binaria

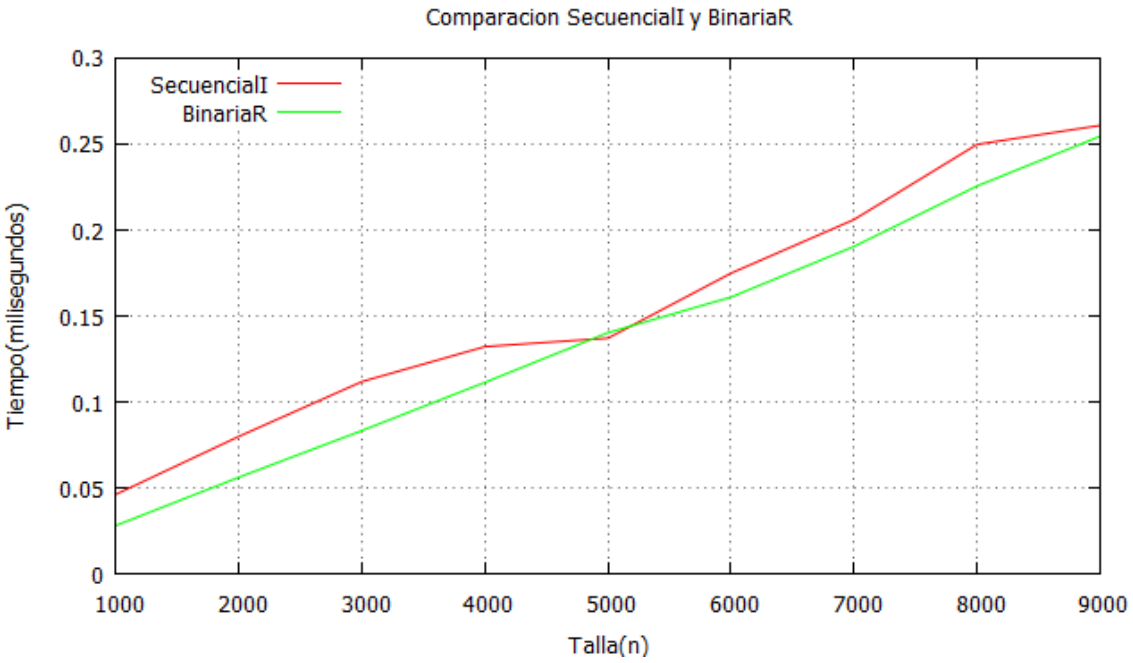


Ternaria

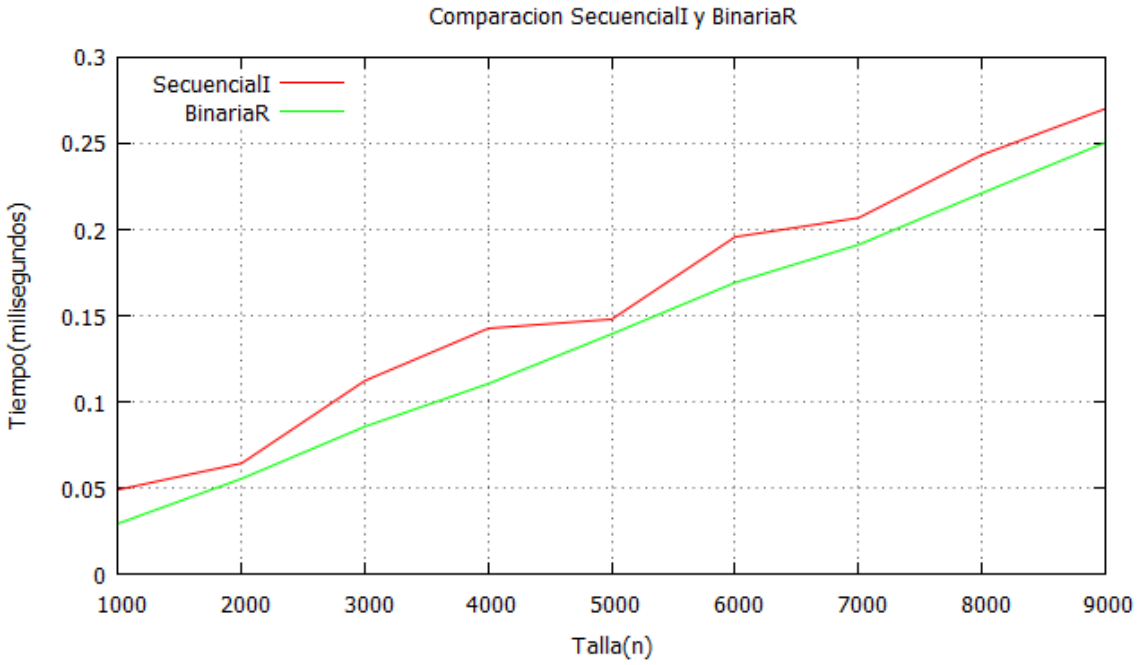


Ahora vamos a comparar los tres casos:

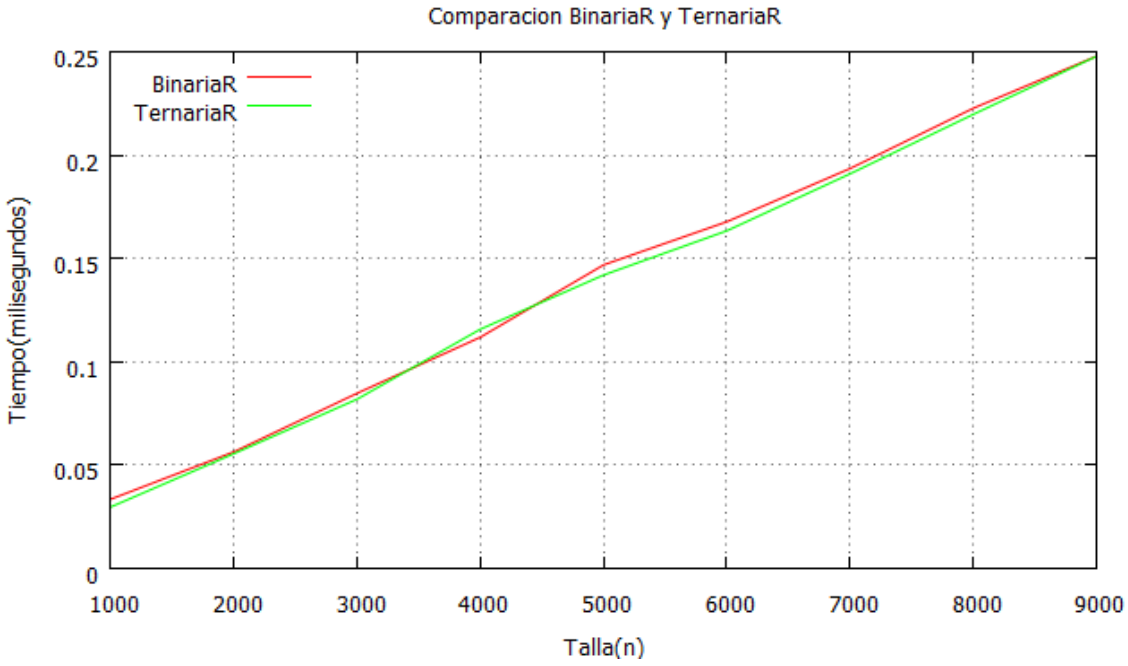
Secuencial y Binaria



Secuencial y Ternaria



Binaria y Ternaria



1.4.- Conclusiones

Como conclusión, los tres sistemas de búsqueda son bastante parecidos en el tiempo de ejecución aunque existen pequeñas variaciones a medida que aumentas el tamaño de n por lo demás para esta práctica no tengo ninguna conclusión.

2.- Diseño de la aplicación

Elemental.h

* La clase Elemental contiene los #include de todos los demás .cpp y .h :

```
#include <string>
#include <vector>
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
#include <stdio.h>
#include <windows.h>
#include <ctime>  //para time
#include <cstdlib> // for srand, rand

using namespace std;
```

TestBusqueda.h

* La clase TestBusqueda contiene los metodos para:

* 1. Comprobar que los métodos de búsqueda de la clase AlgoritmosBusqueda funcionan adecuadamente.

* 2. Calcular la eficiencia para el caso medio de un método de búsqueda,

* permitiendo guardar los datos e imprimir la gráfica correspondiente con ajuste al Orden de complejidad.

* 3. Comparar el coste temporal de dos métodos de búsqueda

* permitiendo guardar los datos e imprimir la gráfica correspondiente.

```
#ifndef _TEST_BUSQUEDA
#define _TEST_BUSQUEDA
#include "Elemental.h"



class TestBusqueda
{
        vector<string> nombreAlgoritmo;
public:

        TestBusqueda();
        ~TestBusqueda();
```

```
        /*
         * Busca un elemento en un array de enteros según el método indicado
         * param key: clave o elemento a buscar
  * param v: el array de enteros donde buscar
         * param size: tamaño del array de enteros
  * param metodo: Metodo de búsqueda a utilizar
  * return Tiempo empleado en la búsqueda (en milisegundos)
  */
  static double buscaEnArrayDeInt(int key,int v[],int size,int metodo,int &posicion);

        /*
         * Comprueba que los metodos de búsqueda funcionan correctamente
  */
  void comprobarMetodosBusqueda();


 /*
        * Calcula el caso medio de un metodo de búsqueda,
        * Permite las opciones de crear el fichero de datos y la gráfica correspondiente.
  * param metodo: metodo de búsqueda a estudiar.
  */
        void casoMedio(int metodo);
 /*
        * Compara dos metodos de busqueda.
        * Permite las opciones de crear el fichero de datos y la gráfica correspondiente.
  * param metodo1: Primer metodo de búsqueda a comparar
  * param metodo2: Segundo metodo de búsqueda a comparar
  */
  void comparar(int metodo1, int metodo2);

};

#endif
```

TestBusqueda.cpp

* La clase TestBusqueda contiene los metodos para:

* 1. Comprobar que los métodos de búsqueda de la clase AlgoritmosBusqueda funcionan adecuadamente.

* 2. Calcular la eficiencia para el caso medio de un método de búsqueda,

* permitiendo guardar los datos e imprimir la gráfica correspondiente con ajuste al Orden de complejidad.

* 3. Comparar el coste temporal de dos métodos de búsqueda

* permitiendo guardar los datos e imprimir la gráfica correspondiente.

```
#include "TestBusqueda.h"
```

```
#include "Constantes.h"
#include "AlgoritmosBusqueda.h"
#include "AlgoritmosOrdenacion.h"
#include "ConjuntoInt.h"
#include "Graficas.h"
#include "Mtime.h"


TestBusqueda::TestBusqueda()
{
        nombreAlgoritmo.push_back("SecuencialI");
        nombreAlgoritmo.push_back("BinariaR");
        nombreAlgoritmo.push_back("TernariaR");
}
TestBusqueda::~TestBusqueda()
{
}

/*
 * Busca en un array de enteros según el método indicado
 * param v: el array de enteros a ordenar
 * param size: tamaño del array de enteros a ordenar
 * param metodo: Metodo de búsqueda a utilizar
 * return Tiempo empleado en la busqueda (en milisegundos)
 */
double TestBusqueda::buscaEnArrayDeInt(int key,int v[],int size,int metodo,int &posicion)
{
   AlgoritmosBusqueda strategia;
                Mtime t;
                LARGE_INTEGER t_inicial, t_final;
                QueryPerformanceCounter(&t_inicial);
// Invoca al método de búsqueda elegido
   switch (metodo)
                {
      case SECUENCIALIt: posicion=strategia.busquedaSecuencialIt(v, size, key);
         break;
                                case BINARIARc: posicion=strategia.busquedaBinariaRc(v, size, key);
         break;
                                case TERNARIARc: posicion=strategia.busquedaTernariaRc(v, size, key);
         break;
                }
                QueryPerformanceCounter(&t_final);
   return t.performancecounter_diff(&t_final, &t_inicial) * 1000000;
}

/*
```

```
 * Comprueba los metodos de búsqueda
 */
void TestBusqueda::comprobarMetodosBusqueda(){
	int talla;
	cout<<endl<<endl<<"Introduce la talla: ";
	cin>>talla;
	system("cls");
	ConjuntoInt *v= new ConjuntoInt(talla);
	AlgoritmosOrdenacion AlOrdena;
	for (int metodo = 0; metodo < nombreAlgoritmo.size(); metodo++){
		v->GeneraVector(talla);
		/* Ordenar array*/
		AlOrdena.ordenaInsercion(v->getDatos(),talla);
		cout <<endl<<endl<< "vector para el metodo "<<nombreAlgoritmo[metodo]<< ":"<<endl<<endl;
		v->escribe();
		int key;
		cout<<endl<<endl<<"Introduce la clave a buscar: ";
		cin>>key;
		int posicion;
		buscaEnArrayDeInt(key,v->getDatos(),talla,metodo,posicion);
		cout<<endl<<endl<<"posicion de "<<key<<" buscado con el metodo "<<nombreAlgoritmo[metodo]<< " : "<<posicion<<endl<<endl;
		v->vaciar();
		system("pause");
		system("cls");
	}
	system("cls");
}

/*
 * Compara dos metodos de búsqueda.
 * Permite las opciones de crear el fichero de datos y la gráfica correspondiente.
 * param metodo1: Primer metodo de búsqueda a comparar
 * param metodo2: Segundo metodo de búsqueda a comparar
 */
void TestBusqueda::comparar(int metodo1, int metodo2)
{
	system("cls");
	LARGE_INTEGER t_inicial, t_final;
	AlgoritmosOrdenacion AlOrdena;
	AlgoritmosBusqueda strategia;
	Graficas g;
	Mtime t;
	int talla[10];
	double tiemposM1[10], tiemposM2[10];
	int posArray = 0, key;
```

```
string nombreMetodo1 = nombreAlgoritmo[metodo1];
string nombreMetodo2 = nombreAlgoritmo[metodo2];

cout << "\n\n\t*** Comparacion " << nombreMetodo1 << " y " << nombreMetodo2 << " ***" << endl << endl;
cout << "\t\tTiempo de ejecucion promedio" << endl << endl << endl;
cout << "\t\t\t" << nombreMetodo1 << "\t" << nombreMetodo2 << endl << endl;
cout << "\t" << "Talla" << "\t\t" << "Tiempo" << "\t\t" << "Tiempo" << endl << endl;

std::cout.precision(4);
string tipoFinal1, tipoFinal2;
tipoFinal1 = nombreMetodo1 + ".dat";
tipoFinal2 = nombreMetodo2 + ".dat";
ofstream fout1(tipoFinal1.c_str());
ofstream fout2(tipoFinal2.c_str());

for (int i = tallaIni; i< tallaFin; i = i + incTalla)
{
        double acumulador_tiempo1 = 0; //Acumulador para el 1º metodo
        double acumulador_tiempo2 = 0; // Acumulador para el 2º metodo

        for (int j = 0; j < NUMREPETICIONES; j++) //For para repeticion metodo1
        {
                ConjuntoInt *v = new ConjuntoInt(i);
                v->GeneraVector(i); //Creo el vector segun la talla que sea.
                AlOrdena.ordenaSeleccion(v->getDatos(), tallaIni);//Ordeno el array
                key = v->getMitad(); //Obtengo el elemento mitad.
                QueryPerformanceCounter(&t_inicial); //Inicio el tiempo
                switch (metodo1)
                {
                case SECUENCIALIt:
                        strategia.busquedaSecuencialIt(v->getDatos(), tallaIni, key);
                        break;
                case BINARIARc:
                        strategia.busquedaBinariaRc(v->getDatos(), tallaIni, key);
                        break;
                case TERNARIARc:
                        strategia.busquedaTernariaRc(v->getDatos(), tallaIni, key);
                        break;
                }
                QueryPerformanceCounter(&t_final);// Para tiempo
                acumulador_tiempo1 = acumulador_tiempo1 + (t.performancecounter_diff(&t_final,
&t_inicial) * 10000);
        }

        acumulador_tiempo1 = acumulador_tiempo1 / NUMREPETICIONES;//Para hacer la media
```

```
                tiemposM1[posArray] = acumulador_tiempo1; //Meto el tiempo del 1º metodo en el vector

                for (int j = 0; j < NUMREPETICIONES; j++) //For para repeticiones metodo2
                {
                        ConjuntoInt *v = new ConjuntoInt(i);
                        v->GeneraVector(i); //Creo el vector segun la talla que sea.
                        AlOrdena.ordenaSeleccion(v->getDatos(), tallaIni);//Ordeno el array
                        key = v->getMitad(); //Obtengo el elemento mitad.
                        QueryPerformanceCounter(&t_inicial); //Inicio el tiempo
                        switch (metodo2)
                        {
                        case SECUENCIALIt:
                                strategia.busquedaSecuencialIt(v->getDatos(), tallaIni, key);
                                break;
                        case BINARIARc:
                                strategia.busquedaBinariaRc(v->getDatos(), tallaIni, key);
                                break;
                        case TERNARIARc:
                                strategia.busquedaTernariaRc(v->getDatos(), tallaIni, key);
                                break;
                        }
                        QueryPerformanceCounter(&t_final);// Para tiempo
                        acumulador_tiempo2 = acumulador_tiempo2 + (t.performancecounter_diff(&t_final,
&t_inicial) * 10000);
                }
                acumulador_tiempo2 = acumulador_tiempo2 / NUMREPETICIONES;//Para hacer la media
                tiemposM2[posArray] = acumulador_tiempo2;

                fout1 << "\t" << i << "\t\t" << acumulador_tiempo1 << "\n";
                fout2 << "\t" << i << "\t\t" << acumulador_tiempo2 << "\n";

                talla[posArray] = i;
                posArray++;
                cout << "\t" << i << "\t\t" << setprecision(3) << acumulador_tiempo1 << "\t\t" <<
acumulador_tiempo2 << "\n";
        }
        cout << "\n";
        fout1.close();
        fout2.close();
        cout << "Datos guardados en los ficheros " << tipoFinal1<<" y "<<tipoFinal2 << endl;

        //Pregunta para generar la gráfica
        char op;
        cout << "\n\nQuiere crear la grafica(s/n): ";
        cin >> op;
        if ((op == 's'))
        {
```

```
                g.generarGraficaCMP(nombreMetodo1, nombreMetodo2);
                cout << "\nGrafica guardada en el fichero " << nombreMetodo1 + nombreMetodo2 + ".plt" << endl;
        }
        system("pause");
}
/*
* Calcula el caso medio de un metodo de búsqueda,
* Permite las opciones de crear el fichero de datos y la gráfica correspondiente.
* param metodo: metodo de búsqueda a estudiar.
*/
void TestBusqueda::casoMedio(int metodo)
{
        system("cls");
        LARGE_INTEGER t_inicial, t_final;
        AlgoritmosOrdenacion AlOrdena;
        AlgoritmosBusqueda strategia;
        Graficas g;
        Mtime t;

        // 0 = Busqueda Secuencial .... 1 = Busqueda Binaria
        system("cls");
        double tiempo[10]; //Array que almacena los tiempos
        int talla[10]; //Array que alamacena la talla
        int contadorDos = 0, key;

        cout << "\n\n\t*** Ordenacion por " << nombreAlgoritmo[metodo] << " ***" << endl << endl;
        cout << "\tTiempos de ejecucion promedio" << endl << endl << endl;
        cout << "\t\t" << "Talla" << "\t" << "Tiempo" << endl << endl;
        //double acumulador_tiempo = 0;

        string tipoFinal;
        tipoFinal = nombreAlgoritmo[metodo] + ".dat";
        ofstream fout(tipoFinal.c_str());

        for (int i = tallaIni; i < tallaFin; i = i + incTalla)
        {
                double tiempo_medio = 0;
                for (int j = 0; j < NUMREPETICIONES; j++)
                {
                        ConjuntoInt *v = new ConjuntoInt(i);
                        v->GeneraVector(i);
                        AlOrdena.ordenaSeleccion(v->getDatos(), i); //Ordeno el array
                        key = v->getMitad(); //Obtengo el elemento mitad.
                        QueryPerformanceCounter(&t_inicial); //Inicio el tiempo
                        switch (metodo)
```

```
                        {
                        case SECUENCIALIt:
                                strategia.busquedaSecuencialIt(v->getDatos(), tallaIni, key);
                                break;
                        case BINARIARc:
                                strategia.busquedaBinariaRc(v->getDatos(), tallaIni, key);
                                break;
                        case TERNARIARc:
                                strategia.busquedaTernariaRc(v->getDatos(), tallaIni, key);
                                break;

                        }
                        QueryPerformanceCounter(&t_final);// Para tiempo
                        tiempo_medio = tiempo_medio + (t.performancecounter_diff(&t_final, &t_inicial) *
100000);
                        v->vaciar();
                        v->~ConjuntoInt();
                }
                tiempo_medio = tiempo_medio / NUMREPETICIONES; //Hacemos la media.
                cout << "\t\t" << i << "\t" << tiempo_medio << " \n";
                talla[contadorDos] = i;
                tiempo[contadorDos] = tiempo_medio;
                contadorDos++;

                fout << "\t\t" << i << "\t" << tiempo_medio << "\n";

        }
        fout.close();
        cout << "\n";
        cout << "\nDatos guardados en el fichero " << tipoFinal << endl;

        //Pregunta para generar la gráfica
        char op;
        cout << "\n\nQuiere crear la grafica(s/n): ";
        cin >> op;

        if (op == 's')
        {
                int num_orden;
                if (metodo == SECUENCIALIt)
                        num_orden = 1;
                else num_orden = 2;

                g.generarGraficaMEDIO(nombreAlgoritmo[metodo], num_orden);
                cout << "\nGrafica guardada en el fichero " << nombreAlgoritmo[metodo] + ".plt" << endl;
        }
```

```
        system("pause");
}
```

AlgoritmosBusqueda.h

* Clase AlgoritmosBusqueda que implementa los Algoritmos de Busqueda para buscar un elemento en un vector de enteros.

* Define las implementaciones de los siguientes metodos de busqueda

* de busqueda en vectores:

* - Secuencial

* - Binaria o dicotomica

* - Ternaria

```
#ifndef _BUSQUEDA
#define _BUSQUEDA
#include "Elemental.h"
class AlgoritmosBusqueda
{
public:
   AlgoritmosBusqueda();
   ~AlgoritmosBusqueda();

 /*
         * FunciÃ³n busquedaSecuencialIt, implementa el mÃ©todo de bÃºsqueda secuencial Iterativo
         * param v: el array de enteros donde buscar
         * param size: tamaÃ±o del array
         * param key: clave o elemento a buscar
         * return posiciÃ³n de la clave en el array
         */
  int busquedaSecuencialIt(int v[], int size,int key);

        /*
         * FunciÃ³n busquedaBinariaRc, implementa el mÃ©todo de bÃºsqueda binaria Recursivo
         * param v: el array de enteros donde buscar
         * param size: tamaÃ±o del array
         * param key: clave o elemento a buscar
         * return posiciÃ³n de la clave en el array
         */
  int busquedaBinariaRc(int v[], int size,int key);
         int BinariaRc(int A[], int left, int right, int key);

        /*
         * FunciÃ³n busquedaTernariaRc, implementa el mÃ©todo de bÃºsqueda ternaria recursiva
```

```
        * param v: el array de enteros donde buscar
        * param size: tamaÃ±o del array
        * param key: clave o elemento a buscar
        * return posiciÃ³n de la clave en el array
        */
        int busquedaTernariaRc(int v[], int size,int key);
        int TernariaRc(int A[], int left, int right, int key);
};
#endif
```

AlgoritmosBusqueda.cpp

* Clase AlgoritmosBusqueda que implementa los Algoritmos de Busqueda para buscar un elemento en un vector de enteros.

* Define las implementaciones de los siguientes metodos de busqueda

* de busqueda en vectores:

* - Secuencial

* - Binaria o dicotomica

* - Ternaria

```
#include "AlgoritmosBusqueda.h"


/*
 * ImplementaciÃ³n de los mÃ©todos de la clase AlgoritmosBusqueda
 */
AlgoritmosBusqueda::AlgoritmosBusqueda() { }
AlgoritmosBusqueda:: ~AlgoritmosBusqueda() { }

/*
        * FunciÃ³n busquedaSecuencialIt, implementa el mÃ©todo de bÃºsqueda secuencial Iterativo
        * param v: el array de enteros donde buscar
        * param size: tamaÃ±o del array
        * param key: clave o elemento a buscar
        * return posiciÃ³n de la clave en el array
        */
int AlgoritmosBusqueda::busquedaSecuencialIt(int v[], int size,int key)
{
        int i=0;
        while(v[i]!=key && i<size){
                i=i+1;
        }
/*      if(i==size){
                return -1;
        }*/
```

```
		if(v[i]==key){//&& key!=0
			return i;
		}
		else{
			return -1;
		}
}

/*
		 * FunciÃ³n busquedaBinariaRc, implementa el mÃ©todo de bÃºsqueda binaria Recursivo
		 * param v: el array de enteros donde buscar
		 * param size: tamaÃ±o del array
		 * param key: clave o elemento a buscar
		 * return posiciÃ³n de la clave en el array
		 */

int AlgoritmosBusqueda::busquedaBinariaRc(int v[], int size,int key)
{
		return BinariaRc(v,0,size-1,key);
}

int AlgoritmosBusqueda::BinariaRc(int v[], int left, int right, int key)
{
		if(left>=right){
			if(v[right]==key){
				return right;
			}
			else return -1;
		}
		int mitad=((left+right+1)/2);
		if(key==v[mitad]){
			return mitad;
		}
		else{
			if(key<v[mitad]){
				return BinariaRc(v,left,mitad-1,key);
			}
			else{
				if(key>v[mitad]){
					return BinariaRc(v,mitad+1,right,key);
				}
			}
		}
}
```

```
/*
	* FunciÃ³n busquedaTernariaRc, implementa el mÃ©todo de bÃºsqueda ternaria recursiva
	* param v: el array de enteros donde buscar
	* param size: tamaÃ±o del array
	* param key: clave o elemento a buscar
	* return posiciÃ³n de la clave en el array
	*/

int AlgoritmosBusqueda::busquedaTernariaRc(int v[], int size,int key)
{
	return TernariaRc(v,0,size-1,key);
}

int AlgoritmosBusqueda::TernariaRc(int v[], int left, int right, int key)
{

	if (left >= right)
		{
			if (v[right] == key){
				return right;
			}
			else return -1;
		}
	int tercio = ((right - left + 1) / 3);

	if (key == v[left + tercio]){
		 return left + tercio;
	}
	else{
		if (key<v[right - tercio]){
			return TernariaRc(v, left, left + tercio - 1, key);
		}
		else
		{
			if (key<v[right - tercio]){
				return TernariaRc(v, left + tercio + 1, right - tercio - 1, key);
			}
			else{
		return TernariaRc(v, right - tercio + 1, right, key);
			}
		}
	}
}
```

AnalisisAlgoritmos.cpp

* Clase principal del programa donde se incluye el main

```
#include "TestBusqueda.h"
#include "Constantes.h"
#include "AlgoritmosBusqueda.h"
#include "AlgoritmosOrdenacion.h"
#include "ConjuntoInt.h"
#include "Graficas.h"
#include "Mtime.h"
#include "TestOrdenacion.h"
#include <iostream>
using namespace std;

                        /** Programa principal **/
int main(){
        int opciones;
        do{system("cls");
        cout<<"\t\t\t\t\t*** FAA. Practica 3. Curso 15/16 ***\n";
        cout<<"\t\t\t\t\t\t\t\tAlberto Gale Gonzalez\n\n";
        cout<<"\t\t\t*** MENU PRINCIPAL ***\n\n";
        cout<<"\t\t1.- Menu de ordenacion\n\n";
        cout<<"\t\t2.- Menu de busqueda\n\n";
        cout<<"\t\t0.- Salir\n\n";
        cout<<"\t\t---------\n\n";
        cout<<"\t\tElige una opcion: ";
        cin>>opciones;
        switch (opciones){

        case 1: {system("cls");

                int opciones1;
                do{system("cls");
                TestOrdenacion objeto;
                cout<<"\t\t\t*** MENU DE ORDENACION ***\n\n";
                cout<<"\t*** ANALISIS EXPERIMENTAL DE ALGORITMOS DE ORDENACION ***\n\n";
                cout<<"\t\t1.- Probar los metodos de ordenacion\n\n";
                cout<<"\t\t2.- Obtener el caso medio de un metodo de ordenacion\n\n";
                cout<<"\t\t3.- Comparar dos metodos\n\n";
                cout<<"\t\t0.- Volver al menu principal\n\n";
                cout<<"\t\t---------\n\n";
                cout<<"\t\tElige una opcion: ";
                cin>>opciones1;
                switch (opciones1){
                case 1: {system("cls");
                        objeto.comprobarMetodosOrdenacion();
```

```
                    system("pause");
                    break;
                            }

        case 2: {system("cls");
                    int opcion1;
                    cout<<"\n\n\t\t*** Metodo a estudiar para el caso medio***\n\n";
                    cout<<"\t\t0.- Burbuja\n\n";
                    cout<<"\t\t1.- Insercion\n\n";
                    cout<<"\t\t2.- Seleccion\n\n";
                    cout<<"\t\t---------\n\n";
               cout<<"\t\tElige una opcion: ";
               cin>>opcion1;
   objeto.casoMedio(opcion1);
   break;
}

        case 3: {system("cls");
        int opcion2,opcion3;
                    cout<<"\n\n\t\t*** Metodos a comparar***\n\n";
                    cout<<"\t\t0.- Burbuja\n\n";
                    cout<<"\t\t1.- Insercion\n\n";
                    cout<<"\t\t2.- Seleccion\n\n";
                    cout<<"\t\t---------\n\n";
               cout<<"\t\tElige el primer metodo: ";
               cin>>opcion2;
               cout<<"\n\n\t\tElige el segundo metodo: ";
               cin>>opcion3;
      objeto.comparar(opcion2,opcion3);
                    system("pause");
                    break;
                            }
        case 0:{system("pause");
        cout<<"\n\n\n";
                    break;
                      }

        default: {system("cls");
                    cout<<"Opcion invalida";
                    system("pause");
                    break;
                             }
        }
}while(opciones1!=0);
        break;
}
```

```
case 2: {system("cls");
        int opciones2;
        do{system("cls");
        TestBusqueda objeto;
        cout<<"\t\t\t*** MENU DE BUSQUEDA ***\n\n";
        cout<<"\t*** ANALISIS EXPERIMENTAL DE ALGORITMOS DE ORDENACION ***\n\n";
        cout<<"\t\t1.- Probar los metodos de busqueda\n\n";
        cout<<"\t\t2.- Obtener el caso medio de un metodo de busqueda\n\n";
        cout<<"\t\t3.- Comparar dos metodos\n\n";
        cout<<"\t\t0.- Volver al menu principal\n\n";
        cout<<"\t\t---------\n\n";
        cout<<"\t\tElige una opcion: ";
        cin>>opciones2;
        switch (opciones2){
        case 1: {system("cls");
                objeto.comprobarMetodosBusqueda();
                system("pause");
                break;
                        }

        case 2: {system("cls");
                int opcion1;
                cout<<"\n\n\t\t*** Metodo a estudiar para el caso medio***\n\n";
                cout<<"\t\t0.- Secuencial Iterativa\n\n";
                cout<<"\t\t1.- Binaria Recursiva\n\n";
                cout<<"\t\t2.- Ternaria Recursiva\n\n";
                cout<<"\t\t---------\n\n";
           cout<<"\t\tElige una opcion: ";
           cin>>opcion1;
   objeto.casoMedio(opcion1);
   break;
}

        case 3: {system("cls");
        int opcion2,opcion3;
                cout<<"\n\n\t\t*** Metodos a comparar***\n\n";
                cout<<"\t\t0.- Secuencial Iterativa\n\n";
                cout<<"\t\t1.- Binaria Recursiva\n\n";
                cout<<"\t\t2.- Ternaria Recursiva\n\n";
                cout<<"\t\t---------\n\n";
           cout<<"\t\tElige el primer metodo: ";
           cin>>opcion2;
           cout<<"\n\n\t\tElige el segundo metodo: ";
           cin>>opcion3;
```

```
                objeto.comparar(opcion2,opcion3);
                                system("pause");
                                break;
                                        }
                        case 0:{system("pause");
                        cout<<"\n\n\n";
                                break;
                                  }

                        default: {system("cls");
                                cout<<"Opcion invalida";
                                system("pause");
                                break;
                                        }
                        }
        }while(opciones2!=0);
                        break;
        }


        case 0:{system("pause");
                        cout<<"\n\n\n";
                                break;
                                  }

                        default: {system("cls");
                                cout<<"Opcion invalida";
                                system("pause");
                                break;
                                        }
        }



        }while(opciones!=0);


        return 0;
}
```

NOTA:

Además de todos estos, hay que añadirle los archivos de la práctica 2º que son los mismos sin ninguna modificación importante.

3.- Conclusión y valoración personal

En mi opinión, este programa es bastante útil a la hora de averiguar que algoritmo es más eficiente teniendo en cuenta cuánto tiempo tarda en ejecutarse cada algoritmo. Aún asi, y tal como ocurria en la práctica anterior, su principal problema es que no es constante y no se ajusta al tiempo estimado teóricamente, ya que todo depende de la velocidad de ejecución de cada procesador. Aun asi, el desarrollo de esta aplicación resulta muy útil para comprender bien como funciona los algoritmos y observar cómo se comportar en el ordenador para diferenciar los casos prácticos de los teóricos que aunque resulten muy parecidos, siempre existen pequeñas variaciones.