



Duración	2 sesiones.
Fechas de realización	Del 27 de Marzo al 7 de Abril.
Fechas de entrega	En el comienzo de la siguiente práctica, medio indicado por el profesor de prácticas.
Objetivos	<ul style="list-style-type: none">■ Desarrollar herramientas que utilizaremos durante el curso.■ Introducir el concepto de coste espacial y su relación con la pila de recursión.■ Implementar algoritmos recursivos de forma iterativa.■ Analizar el algoritmo de búsqueda binaria y de ordenación directa.
Trabajo a realizar	Implementar y analizar algoritmos de búsqueda y de ordenación directa.

Índice

1. Generación de vectores de prueba	3
1.1. EJERCICIO 1	3
2. Algoritmo de búsqueda binaria	3
2.1. EJERCICIO 2	4
2.2. EJERCICIO 3	4
2.3. EJERCICIO 4	4
2.4. EJERCICIO 5	4
2.5. EJERCICIO 6	4
3. Algoritmos de ordenación directa	4
3.1. EJERCICIO 7	4
3.2. EJERCICIO 8	5
3.3. EJERCICIO 9	5
3.4. EJERCICIO 10	5
4. Anexos	6
4.1. Anexo A. Código suministrado	6
4.2. Anexo B. Costes teóricos de los algoritmos	7



Referencias

- [1] Guia de Doxygen y Estándar de Codificación. [\[doxygen.pdf\]](#)
- [2] Aprenda C++ como si estuviera en primero, J. García de Jaln, J.I Rodríguez, J.M Sarriegui y A. Brazález. Escuela Superior de Ingenieros Industriales de San Sebastián, Universidad de Navarra, 1998. [\[manualecpp.pdf\]](#)
- [3] Tema 5. [DISEÑO DE ALGORITMOS ITERATIVOS](#). Juan José Martínez Durá. 2005.
- [4] Tema 6. [RESOLUCIÓN DE RECURRENCIAS](#). Juan José Martínez Durá. 2005.
- [5] Tema 7. [ESQUEMA ALGORÍTMICO DIVIDE Y VENCERÁS](#). Juan José Martínez Durá. 2005.
- [6] Tema 8. [ALGORITMOS DE ORDENACIÓN](#). Juan José Martínez Durá. 2005.



1. Generación de vectores de prueba

Para la realización de las prácticas asociadas al cálculo de costes vamos a utilizar todos los conceptos que hemos aprendido hasta ahora, así vamos a utilizar la STL para la generación de las estructuras de datos que necesitemos.

En esta práctica y en algunas sucesivas vamos a trabajar sobre vectores. En este caso utilizaremos el contenedor vector definido en el fichero de cabecera `<vector>` y todas las funciones y algoritmos definidas sobre éste. En primer lugar vamos a implementar un par de funciones que agruparemos en la clase de utilidades `DataGenerator`; estas funciones permitirán generar un vector de N elementos ordenados con repetición y un vector de N elementos ordenados sin repetición. El fichero de cabecera asociado a esta librería se denomina `DataGenerator.h` (Listado 1).

En la generación de números aleatorios se utilizará como semilla la función `time()`, y las funciones `srand(unsigned)` y `rand()`, para la generación.

1.1. EJERCICIO 1

Se pide implementar las funciones del fichero de cabecera `DataGenerator.h` en el fichero `DataGenerator.cpp`.

2. Algoritmo de búsqueda binaria

Sea un conjunto donde se ha definido una relación de orden, x un elemento de este conjunto y v un vector de elementos de este conjunto. Considerar las variaciones siguientes del algoritmo de búsqueda dicotómica o binaria (BB).

- BB Normal*: El algoritmo comprueba la posición central del vector y según el valor que encuentre, reduce la búsqueda a la primera o segunda mitad del vector. Este razonamiento vuelve a aplicarse en el subvector resultante hasta que se encuentra x o se comprueba que no está.
- BB desplazada*: El algoritmo es idéntico al anterior, pero en lugar de comprobar la posición central del vector, comprueba la posición $2/3$.

Considerar los casos de:

- Búsqueda con éxito en vectores ordenados SIN elementos repetidos.
- Búsquedas aleatorias en vectores ordenados CON elementos repetidos.

se pide:



2.1. EJERCICIO 2

Implementa un programa que incluya una versión iterativa para cada uno de los dos algoritmos (`Busqueda.h` y `Busqueda.cpp`) y obtén el coste medio en COMPARACIONES sobre el vector para los dos algoritmos en los casos (1) y (2). El coste se calculará para vectores de 20 a 500 elementos con incrementos de 20 unidades. Se deben obtener los siguientes archivos de datos de tipo texto: `bb_a1.dat`, `bb_a2.dat`, `bb_b1.dat` y `bb_b2.dat`.

2.2. EJERCICIO 3

Compara los resultados del coste medio en comparaciones para los casos (1) y (2) para el algoritmo A. Llamalo `bb_a.plt`.

2.3. EJERCICIO 4

Haz lo mismo para el algoritmo B y llama a la gráfica `bb_b.plt`. Comenta los resultados obtenidos. ¿ De qué manera influye la utilización de la configuración distinta de los vectores en A y B a la vista de los resultados ?

2.4. EJERCICIO 5

Compara el coste medio de los dos algoritmos para el caso 1 (`bb_1.plt`). Comentar los resultados obtenidos.

2.5. EJERCICIO 6

Compara el coste medio de los dos algoritmos para el caso 2 (`bb_2.plt`). Comentar los resultados obtenidos.

3. Algoritmos de ordenación directa

Considera los siguientes algoritmos de ordenación directa (figura 1), como se puede observar se utiliza una función *mayor* y un procedimiento *intercambia*.

3.1. EJERCICIO 7

Codifica los tres algoritmos (`Ordenacion.h` y `Ordenacion.cpp`) y construye una función que permita permutar aleatoriamente un vector ordenado, realizando un porcentaje de mezclas (método *interchangeElements* del listado 1)



```
Algoritmo Burbuja
DATOS A: vector[1..n] de N
RES A: vector[1..n] de N
METODO
  PARA i = 2 HASTA n HACER
    PARA j = n HASTA i CON -1 HACER
      SI mayor (A[j-1],A[j]) ENTONCES
        Intercambia (A,j-1,j);
      fSI
    fPARA
  fPARA
fBURBUJA

Algoritmo Selección
DATOS A vector[1..n] de N
RES: A vector[1..n] de N
METODO
  PARA i = 1 HASTA n-1 HACER
    pmin = i
    PARA j = i+1 HASTA n HACER
      SI mayor(A[pmin],A[j]) ENTONCES
        pmin = j
    fSI
    fPARA
    Intercambia(A,i,pmin)
  fPARA
fSELECCION

Algoritmo Inserción
DATOS A vector[1..n] de N
RES A vector[1..n] de N
METODO
  PARA i = 2 HASTA n HACER
    j = i-1
    MIENTRAS (j>1) AND mayor(A[j],A[j+1])
      Intercambia(A,j,j+1)
    j = j-1
  fMIENTRAS
  fPARA
fINSERCCION
```

Figura 1: Algoritmos de ordenación directa

3.2. EJERCICIO 8

¿Cuál/Cuáles de los tres sitúa un elemento en su lugar definitivo en cada iteración ? ¿Cuál/Cuáles mantiene un subvector ordenado no definitivo ?

3.3. EJERCICIO 9

Haz un estudio del coste medio en comparaciones y en intercambios de los tres algoritmos sobre un vector que sea una permutación aleatoria de 40 % de sus elementos. Esto se ejecutará desde una talla de 20 elementos a 500 de 20 en 20. Llama a los ficheros generados `i_com.dat`, `s_com.dat` y `b_com.dat`, para el caso de las comparaciones, y `i_int.dat`, `s_int.dat` y `b_int.dat`, para el caso de los intercambios.

- Visualízalos con sus correspondiente coste teórico y guárdalos en los archivos homólogos *.plt. Comenta el resultado.
- Visualiza en un fichero `intercambios.plt` y `comparaciones.plt` los tres valores obtenidos para estos algoritmos. ¿Podrás afirmar a la vista de las gráficas cuál es el algoritmo más eficiente ?

3.4. EJERCICIO 10

Haz el análisis de comparaciones del algoritmo de selección y de inserción sobre un vector que sea una permutación ordenada de `m` elementos (por ejemplo genera un vector ordenado y desordenalo en un 10 %). Llámalos `i_int_10.dat`, `b_int_10.dat` y `s_int_10.dat`. Visualízalos con sus correspondientes del punto anterior. Llama a los ficheros `i_comparar.plt`, `b_comparar.plt` y `s_comparar.plt`. ¿Qué peculiaridades ofrece el coste de estos algoritmos y por qué ? ¿Existe alguna diferencia con los resultados obtenidos en el caso de vectores más desordenados ?



4. Anexos

4.1. Anexo A. Código suministrado

Listing 1: DataGenerator.h

```
/**
 * Practicas de Metodologia de la Programacion.
 * Curso 2005-2006.
 * Departament d'Informatica.
 * Universitat de Valencia.
 */

#ifndef _DATA_GENERATOR_H
#define _DATA_GENERATOR_H

#include <vector>
using namespace std;

/**
 * Libreria para la generacion aleatoria de vectores.
 *
 * @author <A href="mailto:raul.penya@uv.es">Raúl Peña; l Peñaltilde; a-Ortiz</A>
 *
 * @version 1.0
 */
class DataGenerator {
public:
    /**
     * Crea una instancia de la clase generadora de vectores.
     * <BR>
     * En la creacion de la instancia se inicializa la semilla aleatoria.
     */
    DataGenerator();

    /**
     * Rellena un vector de numeros enteros con posibilidad de repeticion.
     *
     * @param v Vector de enteros a rellenar.
     */
    void generateOrderedVectorWithRepetition(vector<int> &v);

    /**
     * Rellena un vector de numeros enteros sin repeticion de elementos.
     *
     * @param v Vector de enteros a rellenar.
     */
    void generateOrderedVectorWithoutRepetition(vector<int> &v);

    /**
     * Genera un numero natural aleatorio entre 0 y un maximo.
     *
     * @param max Limite maximo del numero a utilizar.
     *
     * @return Numero natural aleatorio.
     */
    int generateRandomNumber(int max);

    /**
     * Mezcla los elementos de un vector, realizando un porcentaje
     * de intercambios con respecto a su tamaño.<BR>
     *
     * Ejemplo de aplicacion:
     * <CODE>
     * interchangeElements({1,2,3,4,5,6,7,8,9,10},40)<BR><BR>
     * Realiza v.size()*40/100 = 4 intercambios de elementos, lo
     * que significa que por ejemplo selecciona el 1 y lo cambia
     * por el 5 (que tambien selecciona aleatoriamente), y hace
     * lo mismo con el 4 y el 7.
     * </CODE>
     *
     * @param v Vector a mezclar.
     * @param percentage Porcentaje que indica el numero de intercambios
     * a realizar en la mezcla del vector.
     */
    void interchangeElements(vector<int> &v, float percentage);
};

#endif
```



4.2. Anexo B. Costes teóricos de los algoritmos

		Intercambios			Comparaciones		
Inserción	3n	$\frac{1}{4}n^2$	$\frac{1}{2}n^2$	n	$\frac{1}{4}n^2$	$\frac{1}{2}n^2$	
Selección	3n	$0,69n \log n$	$\frac{1}{2}n^2$	$\frac{1}{2}n^2$	$\frac{1}{2}n^2$	$\frac{1}{2}n^2$	
Burbuja	0	$\frac{3}{4}n^2$	$\frac{3}{2}n^2$	$\frac{1}{2}n^2$	$\frac{1}{2}n^2$	$\frac{1}{2}n^2$	
	Mejor	Medio	Peor	Mejor	Medio	Peor	