UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y CC. Computación
E.T.S.I. Telecomunicación

BÚSQUEDA Y ORDENACIÓN

Tema 5

Programación I

Tema 5: BÚSQUEDA Y ORDENACIÓN

- INTRODUCCIÓN
  - Búsqueda:
    - Localización de un elemento en una colección de datos.
  - Ordenación:
    - Organización de una colección de datos de acuerdo a algún criterio de ordenación.
  - Algunas Funciones Útiles de la Biblioteca:
    - Funciones matemáticas
    - Control del tiempo
    - Generación de números aleatorios

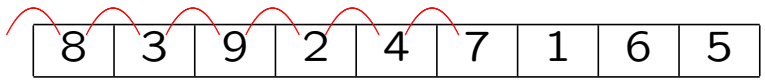
Tema 5: BÚSQUEDA Y ORDENACIÓN

BÚSQUEDA LINEAL (SECUENCIAL)

- Adecuada como mecanismo de búsqueda general en colecciones de datos **sin organización** conocida.
- Si encuentra el elemento buscado, entonces devuelve el índice donde se encuentra el elemento en el array, en otro caso devuelve un índice con valor *fuera de rango*.

```
int busq_lineal (const Vector& v, int x)
{
   int i = 0;
   // Nótese la evaluación en CORTOCIRCUITO
   while ((i < int(v.size()))   // &&(x != v[i])
           && ! encontrado(x, v[i])) {
      ++i;
   }
   return i;
}
```

```
int busq_lineal_alt (const Vector& v, int x)
{
   int idx = int(v.size());
   bool ok = false;
   for (int i = 0; (i < int(v.size())) && !ok; ++i) {
       if (encontrado(x, v[i])) { // (x == v[i])
          ok = true;
          idx = i;
       }
   }
   return idx;
}
```

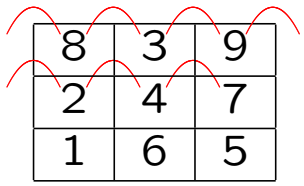


BÚSQUEDA LINEAL EN ARRAY 2 DIMENSIONES

- Adecuada como mecanismo de búsqueda general en colecciones de datos **sin organización** conocida.

- Si encuentra el elemento buscado, entonces devuelve los índices donde se encuentra el elemento en el array 2D, en otro caso devuelve un valor *fuera de rango* en el parámetro del índice de la fila y el valor del índice de la columna queda inespecificado.

```
void busq_lineal_2d (const Matriz& m, int x,
                     int& f, int& c)
{
   f = 0;
   c = 0;
   // Nótese la evaluación en CORTOCIRCUITO
   while ((f < int(m.size()))   // &&(x != m[f][c])
           && ! encontrado(x, m[f][c])) {
      ++c;
      if (c >= int(m[f].size())) {
         c = 0;
         ++f;
      }
   }
}
```




```
void busq_lineal_2d_alt (const Matriz& m, int x,
                         int& ff, int& cc)
{
   ff = int(m.size());
   bool ok = false;
   for (int f = 0; (f < int(m.size()))&&!ok; ++f) {
      for (int c = 0; (c < int(m[f].size()))&&!ok; ++c) {
         if (encontrado(x, m[f][c])) { // (x == m[f][c])
            ok = true;
            ff = f;
            cc = c;
         }
      }
   }
}
```

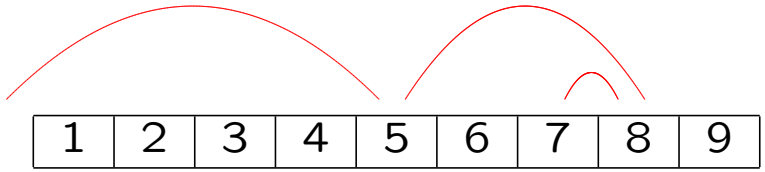
Tema 5: BÚSQUEDA Y ORDENACIÓN

1. Introducción
2. Búsqueda Lineal
3. Búsqueda Binaria
4. Ordenación por Intercambio
5. Ordenación por Selección
6. Ordenación por Inserción
7. Algunas Funciones Útiles de la Biblioteca
8. Aplicación a estructuras de datos
9. Bibliografía: [DALE89a], [JOYA03]

BÚSQUEDA BINARIA

- Adecuada cuando la colección de datos se encuentra **ordenada** por algún criterio.

```
int busqueda_binaria (const Vector& v, int x)
{
    int i = 0;
    int f = int(v.size());
    int m = (i + f) / 2;
    while ((i < f) && ! encontrado(x, v[m])) { // &&(x != v[m])
        if (es_menor(x, v[m])) {  // (x < v[m])
            f = m;
        } else {
            i = m + 1;
        }
        m = (i + f) / 2;
    }
    if (i >= f) {
        m = int(v.size());
    }
    return m;
}
```

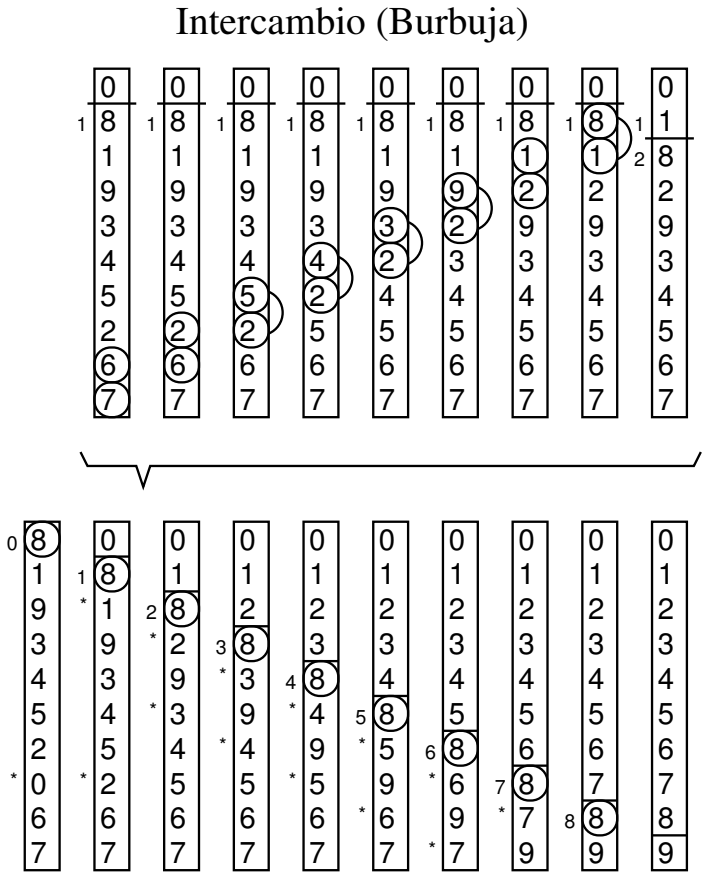


Tema 5: BÚSQUEDA Y ORDENACIÓN

1. Introducción
2. Búsqueda Lineal
3. Búsqueda Binaria
4. Ordenación por Intercambio
5. Ordenación por Selección
6. Ordenación por Inserción
7. Algunas Funciones Útiles de la Biblioteca
8. Aplicación a estructuras de datos
9. Bibliografía: [DALE89a], [JOYA03]

ORDENACIÓN POR INTERCAMBIO (BURBUJA)

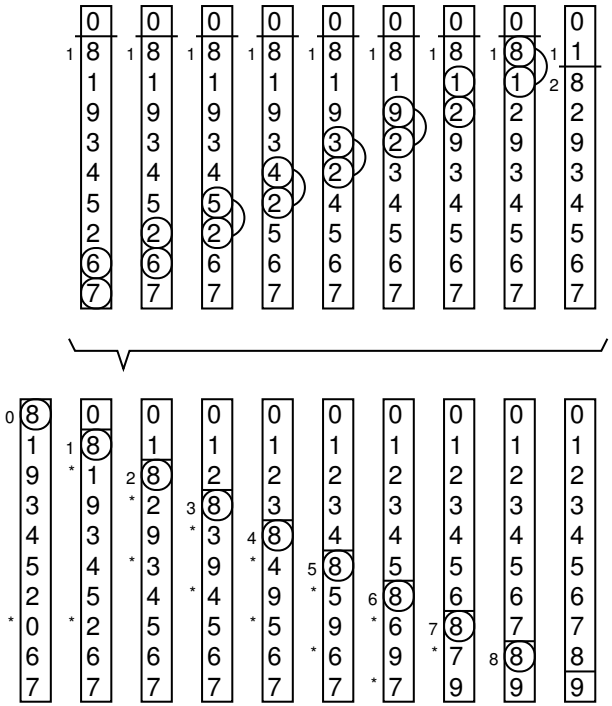
- Se hacen múltiples recorridos sobre la **zona no ordenada** del array, ordenando los elementos **consecutivos**, trasladando en cada uno de ellos al elemento más pequeño hasta el inicio de dicha zona.



ORDENACIÓN POR INTERCAMBIO (BURBUJA)

```
inline void intercambiar (int& x, int& y)
{
   int aux = x;
   x = y;
   y = aux;
}
void subir_menor_intercambio (Vector& v, int i_elm)
{
   for (int i = int(v.size()) - 1; i > i_elm; --i) {
      if (es_menor(v[i], v[i - 1])) { // (v[i] < v[i-1])
         intercambiar(v[i], v[i - 1]);
      }
   }
}
void ord_intercambio (Vector& v)
{
   for (int i_elm = 0; i_elm < int(v.size()) - 1; ++i_elm) {
      subir_menor_intercambio(v, i_elm);
   }
}
```

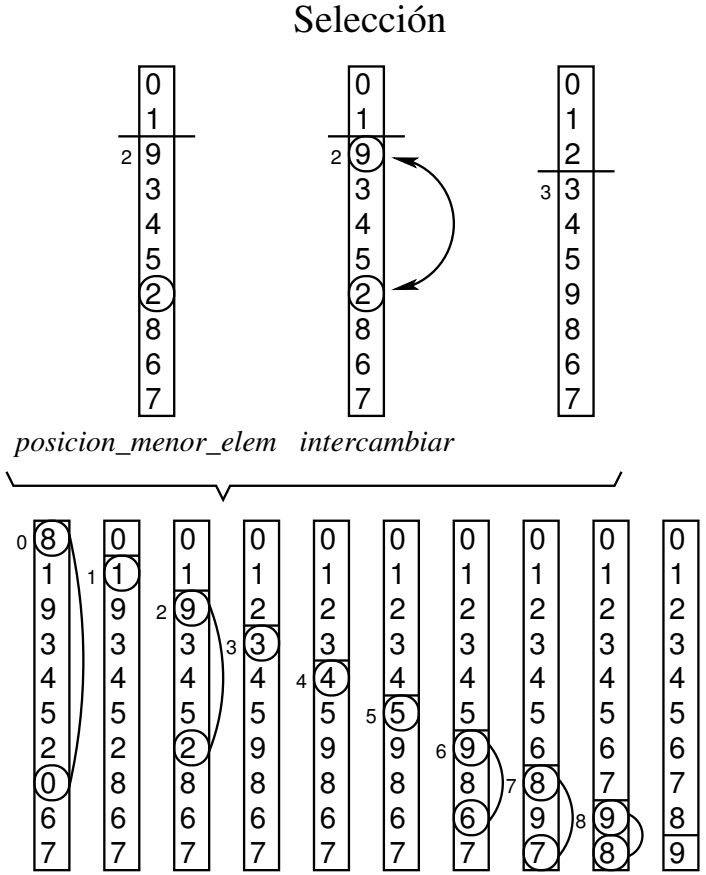
Intercambio (Burbuja)



Tema 5: BÚSQUEDA Y ORDENACIÓN

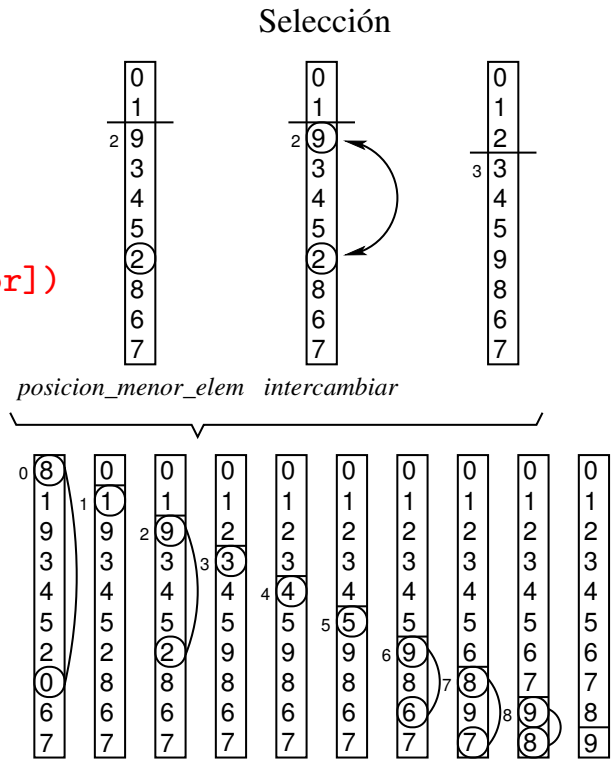
ORDENACIÓN POR SELECCIÓN

- Se busca el elemento más pequeño de la **zona no ordenada** del array, y se traslada al inicio dicha zona, repitiendo el proceso hasta ordenar completamente el array.



ORDENACIÓN POR SELECCIÓN

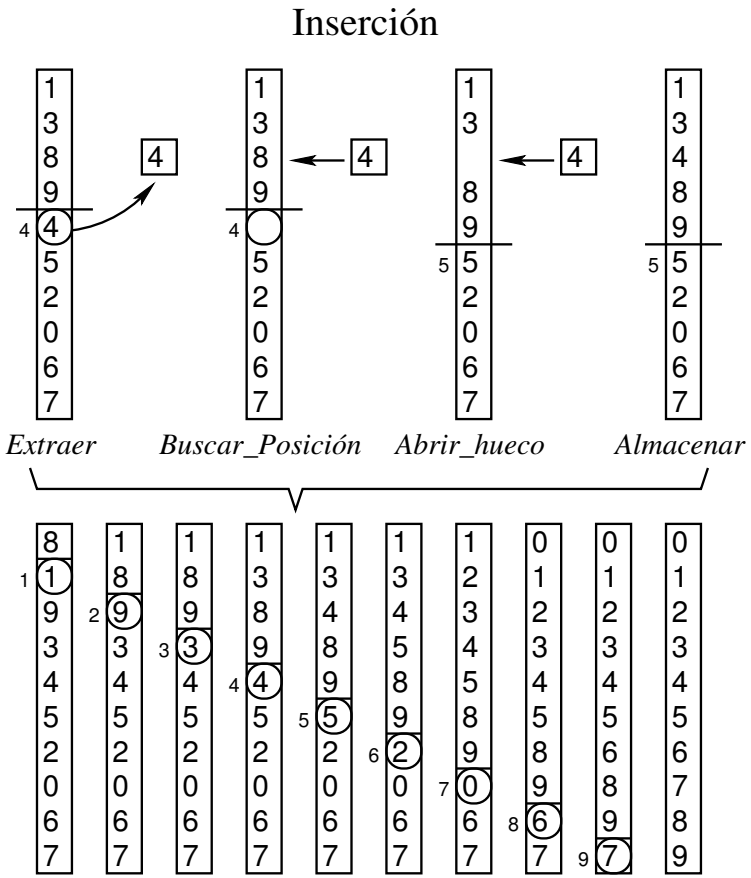
```
inline void intercambiar (int& x, int& y)
{
    int aux = x;
    x = y;
    y = aux;
}
int posicion_menor (const Vector& v, int i_elm)
{
    int pos_menor = i_elm;
    for (int i = pos_menor + 1; i < int(v.size()); ++i) {
        if (es_menor(v[i], v[pos_menor])) {  // (v[i] < v[pos_menor])
            pos_menor = i;
        }
    }
    return pos_menor;
}
void subir_menor_seleccion (Vector& v, int i_elm)
{
    int pos_menor = posicion_menor(v, i_elm);
    if (pos_menor != i_elm) {
        intercambiar(v[pos_menor], v[i_elm]);
    }
}
void ord_seleccion (Vector& v)
{
    for (int i_elm = 0; i_elm < int(v.size()) - 1; ++i_elm) {
        subir_menor_seleccion(v, i_elm);
    }
}
```



Tema 5: BÚSQUEDA Y ORDENACIÓN

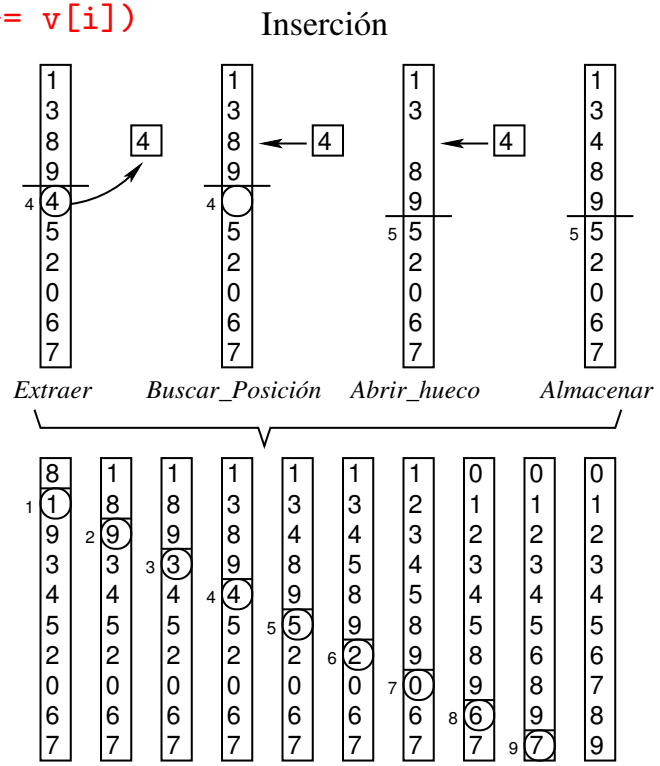
ORDENACIÓN POR INSERCIÓN

- Se toma el primer elemento de la **zona no ordenada** del array, y se inserta en la posición adecuada de la **zona ordenada** del array, repitiendo el proceso hasta ordenar completamente el array.



ORDENACIÓN POR INSERCIÓN

```
int buscar_posicion (const Vector& v, int n_elms, int elm)
{
   int i = 0;
   while ((i < n_elms) && ! es_menor(elm, v[i])) {  // &&(elm >= v[i])
      ++i;
   }
   return i;
}
void abrir_hueco (Vector& v, int n_elms, int pos)
{
   for (int i = n_elms; i > pos; --i) {
      v[i] = v[i - 1];
   }
}
void insertar (Vector& v, int i_elm)
{
   int elm = v[i_elm]; // Extraer
   int pos = buscar_posicion(v, i_elm, elm);
   abrir_hueco(v, i_elm, pos);
   v[pos] = elm; // Almacenar
}
void ord_insercion (Vector& v)
{
   for (int i_elm = 1; i_elm < int(v.size()); ++i_elm) {
      insertar(v, i_elm);
   }
}
```



Tema 5: BÚSQUEDA Y ORDENACIÓN

1. Introducción
2. Búsqueda Lineal
3. Búsqueda Binaria
4. Ordenación por Intercambio
5. Ordenación por Selección
6. Ordenación por Inserción
7. Algunas Funciones Útiles de la Biblioteca
8. Aplicación a estructuras de datos
9. Bibliografía: [DALE89a], [JOYA03]

ALGUNAS FUNCIONES ÚTILES DE LA BIBLIOTECA <cmath>

`double sin(double r) ;`	seno, $\sin r$ (en radianes)
`double cos(double r) ;`	coseno, $\cos r$ (en radianes)
`double tan(double r) ;`	tangente, $\tan r$ (en radianes)
`double asin(double x) ;`	arco seno, $\arcsin x, x \in [-1, 1]$
`double acos(double x) ;`	arco coseno, $\arccos x, x \in [-1, 1]$
`double atan(double x) ;`	arco tangente, $\arctan x$
`double atan2(double y, double x) ;`	arco tangente, $\arctan y/x$
`double sinh(double r) ;`	seno hiperbólico, $\sinh r$
`double cosh(double r) ;`	coseno hiperbólico, $\cosh r$
`double tanh(double r) ;`	tangente hiperbólica, $\tanh r$
`double sqrt(double x) ;`	$\sqrt{x}, x \geq 0$
`double pow(double x, double y) ;`	x^y
`double exp(double x) ;`	e^x
`double log(double x) ;`	logaritmo neperiano, $\ln x, x > 0$
`double log10(double x) ;`	logaritmo decimal, $\log x, x > 0$
`double ceil(double x) ;`	menor entero $\geq x$, $\lceil x \rceil$
`double floor(double x) ;`	mayor entero $\leq x$, $\lfloor x \rfloor$
`double fabs(double x) ;`	valor absoluto de x, $\lvert x \rvert$
`double fmod(double x, double y) ;`	resto de x/y

```
#include <iostream>
#include <cmath>
using namespace std ;
const double PI = 3.14159265 ;
int main()
{
    double c = cos(PI) ;
    cout << "Coseno de PI: " << c << endl ;
}
```

ALGUNAS FUNCIONES ÚTILES DE LA BIBLIOTECA <ctime>

`clock_t clock() ;`	retorna el tiempo de CPU utilizado (`CLOCKS_PER_SEC`)
`time_t time(0) ;`	retorna el tiempo de calendario (en segundos)

```
#include <iostream>
#include <ctime>
using namespace std ;
int main()
{
    time_t t1 = time(0) ;
    clock_t c1 = clock() ;
    // ... procesamiento ...
    clock_t c2 = clock() ;
    time_t t2 = time(0) ;
    cout << "Tiempo de CPU: " << double(c2 - c1)/double(CLOCKS_PER_SEC)
         << " seg" << endl ;
    cout << "Tiempo total: " << (t2 - t1) << " seg" << endl ;
}
```

ALGUNAS FUNCIONES ÚTILES DE LA BIBLIOTECA `<cstdlib>`

`int abs(int n) ;`	retorna el valor absoluto del número `int n`
`long labs(long n) ;`	retorna el valor absoluto del número `long n`
`void srand(unsigned semilla) ;`	inicializa el generador de números aleatorios
`int rand() ;`	retorna un aleatorio entre `0` y `RAND_MAX` (ambos inclusive)

```
#include <cstdlib>
#include <ctime>
using namespace std ;

// inicializa el generador de numeros aleatorios
inline void ini_aleatorio()
{
    srand(time(0)) ;
}

// Devuelve un numero aleatorio entre 0 y max (exclusive)
inline int aleatorio(int max)
{
    return int(max*double(rand())/(RAND_MAX+1.0)) ; // return rand() % max ;
}

// Devuelve un numero aleatorio entre min y max (ambos inclusive)
inline int aleatorio(int min, int max)
{
    return min + aleatorio(max-min+1) ;
}
```

Tema 5: BÚSQUEDA Y ORDENACIÓN

- APLICACIÓN: Agenda Personal Ordenada

  - La información personal que será almacenada es la siguiente: Nombre, Teléfono, Dirección, Calle, Número, Piso, Código Postal y Ciudad

  - Las operaciones a realizar con dicha agenda serán:

    1. Añadir los datos de una persona
    2. Acceder a los datos de una persona a partir de su nombre.
    3. Borrar una persona a partir de su nombre.
    4. Modificar los datos de una persona a partir de su nombre.
    5. Listar el contenido completo de la agenda.

```
#include <iostream>
#include <string>
#include <cassert>
#include <array>
using namespace std;
// - Constantes ----
const int MAX_PERSONAS = 50;
const int OK = 0;
const int AG_LLENA = 1;
const int NO_ENCONTRADO = 2;
const int YA_EXISTE = 3;
// - Tipos ------
struct Direccion {
    int num;
    string calle;
    string piso;
    string cp;
    string ciudad;
};
struct Persona {
    string nombre;
    string tel;
    Direccion direccion;
};
```

```
// - Tipos ------
typedef array<Persona, MAX_PERSONAS> Personas;
struct Agenda {
    int n_pers;
    Personas pers;
};
// - Subalgoritmos --
void Inicializar (Agenda& ag)
{
    ag.n_pers = 0;
}
```

```
// - Subalgoritmos --
void Leer_Direccion (Direccion& dir)
{
    cin >> ws;
    getline(cin, dir.calle);
    cin >> dir.num;
    cin >> dir.piso;
    cin >> dir.cp;
    cin >> ws;
    getline(cin, dir.ciudad);
}
//---------------------------
void Leer_Persona (Persona& per)
{
    cin >> ws;
    getline(cin, per.nombre);
    cin >> per.tel;
    Leer_Direccion(per.direccion);
}
```

```
// - Subalgoritmos --
void Escribir_Direccion (const Direccion& dir)
{
    cout << dir.calle << ", ";
    cout << dir.num << " ";
    cout << dir.piso << endl;
    cout << dir.cp << " ";
    cout << dir.ciudad << endl;
}
//---------------------------
void Escribir_Persona (const Persona& per)
{
    cout << per.nombre << endl;
    cout << per.tel << endl;
    Escribir_Direccion(per.direccion);
}
```

```
// - Subalgoritmos --
// Busca una Persona en la Agenda Ordenada
// Devuelve su posicion si se encuentra, o bien  >= ag.n_pers en otro caso
int Buscar_Persona_Binaria (const string& nombre, const Agenda& ag)
{
    int i = 0;
    int f = ag.n_pers;
    int m = (i + f) / 2;
    while ((i < f) && (nombre != ag.pers[m].nombre)) {
        if (nombre < ag.pers[m].nombre) {
            f = m;
        } else {
            i = m + 1;
        }
        m = (i + f) / 2;
    }
    if (i >= f) {
        m = ag.n_pers;
    }
    return m;
}
```

```
// - Subalgoritmos --
int Buscar_Posicion (const string& nombre, const Agenda& ag)
{
   int i = 0;
   while ((i < ag.n_pers) && (nombre > ag.pers[i].nombre)) {
      ++i;
   }
   return i;
}
//---------------------------
void Anyadir_Ord (Agenda& ag, int pos, const Persona& per)
{
   for (int i = ag.n_pers; i > pos; --i) {
      ag.pers[i] = ag.pers[i - 1];
   }
   ag.pers[pos] = per;
   ++ag.n_pers;
}
//---------------------------
void Eliminar_Ord (Agenda& ag, int pos)
{
   --ag.n_pers;
   for (int i = pos; i < ag.n_pers; ++i) {
      ag.pers[i] = ag.pers[i + 1];
   }
}
```

```
// - Subalgoritmos --
void Anyadir_Persona_Ord (const Persona& per, Agenda& ag, int& ok)
{
    int i = Buscar_Posicion(per.nombre, ag);
    if ((i < ag.n_pers) && (per.nombre == ag.pers[i].nombre)) {
        ok = YA_EXISTE;
    } else if (ag.n_pers == int(ag.pers.size())) {
        ok = AG_LLENA;
    } else {
        ok = OK;
        Anyadir_Ord(ag, i, per);
    }
}
//---------------------------
void Borrar_Persona_Ord (const string& nombre, Agenda& ag, int& ok)
{
    int i = Buscar_Persona_Binaria(nombre, ag);
    if (i >= ag.n_pers) {
        ok = NO_ENCONTRADO;
    } else {
        ok = OK;
        Eliminar_Ord(ag, i);
    }
}
```

```
// - Subalgoritmos --
void Modificar_Persona_Ord (const string& nombre, const Persona& nuevo, Agenda& ag, int& ok)
{
    int i = Buscar_Persona_Binaria(nombre, ag);
    if (i >= ag.n_pers) {
        ok = NO_ENCONTRADO;
    } else {
        Eliminar_Ord(ag, i);
        Anyadir_Persona_Ord(nuevo, ag, ok);
    }
}
//---------------------------
void Imprimir_Persona_Ord (const string& nombre, const Agenda& ag, int& ok)
{
    int i = Buscar_Persona_Binaria(nombre, ag);
    if (i >= ag.n_pers) {
        ok = NO_ENCONTRADO;
    } else {
        ok = OK;
        Escribir_Persona(ag.pers[i]);
    }
}
```

```
// - Subalgoritmos --
void Imprimir_Agenda (const Agenda& ag, int& ok)
{
   for (int i = 0; i < ag.n_pers; ++i) {
      Escribir_Persona(ag.pers[i]);
   }
   ok = OK;
}
//---------------------------
char Menu ()
{
   char opcion;
   cout << endl;
   cout << "a. -  Anadir Persona" << endl;
   cout << "b. -  Buscar Persona" << endl;
   cout << "c. -  Borrar Persona" << endl;
   cout << "d. -  Modificar Persona" << endl;
   cout << "e. -  Imprimir Agenda" << endl;
   cout << "x. -  Salir" << endl;
   do {
      cout << "Introduzca Opcion: ";
      cin >> opcion;
   } while ( ! (((opcion  >=  'a')  && (opcion  <=  'e'))  ||  (opcion  ==  'x')));
   return opcion;
}
```

```
// - Subalgoritmos --
void Escribir_Cod_Error (int cod)
{
    switch (cod) {
    case OK:
        cout << "Operacion correcta" << endl;
        break;
    case AG_LLENA:
        cout << "Agenda llena" << endl;
        break;
    case NO_ENCONTRADO:
        cout << "La persona no se encuentra en la agenda" << endl;
        break;
    case YA_EXISTE:
        cout << "La persona ya se encuentra en la agenda" << endl;
        break;
    }
}
```

```
// - Principal ----
int main ()
{
    Agenda ag;
    char opcion;
    Persona per;
    string nombre;
    int ok;
    Inicializar(ag);
    do {
        opcion = Menu();
        switch (opcion) {
        case 'a':
            cout << "Introduzca los datos de la Persona" << endl;
            cout << "(nombre, tel, calle, num, piso, cod_postal, ciudad)" << endl;
            Leer_Persona(per);
            Anyadir_Persona_Ord(per, ag, ok);
            Escribir_Cod_Error(ok);
            break;
        case 'b':
            cout << "Introduzca Nombre" << endl;
            cin >> nombre;
            Imprimir_Persona_Ord(nombre, ag, ok);
            Escribir_Cod_Error(ok);
            break;
```

```
        case 'c':
            cout << "Introduzca Nombre" << endl;
            cin >> nombre;
            Borrar_Persona_Ord(nombre, ag, ok);
            Escribir_Cod_Error(ok);
            break;
        case 'd':
            cout << "Introduzca Nombre" << endl;
            cin >> nombre;
            cout << "Nuevos datos de la Persona" << endl;
            cout << "(nombre, tel, calle, num, piso, cod_postal, ciudad)" << endl;
            Leer_Persona(per);
            Modificar_Persona_Ord(nombre, per, ag, ok);
            Escribir_Cod_Error(ok);
            break;
        case 'e':
            Imprimir_Agenda(ag, ok);
            Escribir_Cod_Error(ok);
            break;
        }
    } while (opcion  !=  'x' );
}
```

Tema 5: BÚSQUEDA Y ORDENACIÓN

1. Introducción
2. Búsqueda Lineal
3. Búsqueda Binaria
4. Ordenación por Intercambio
5. Ordenación por Selección
6. Ordenación por Inserción
7. Algunas Funciones Útiles de la Biblioteca
8. Aplicación a estructuras de datos
9. Bibliografía: [DALE89a], [JOYA03]