

## Programación 1

# Tema 12

---

## Algoritmos básicos de trabajo con estructuras de datos indexadas



Escuela de  
Ingeniería y Arquitectura  
**Universidad** Zaragoza

# Índice

---

- Algoritmos de **recorrido**
- Algoritmos de **búsqueda**
  - Secuencial
  - Binaria
- Algoritmos de **distribución**
- Algoritmos de **ordenación**
  - Por **selección**

# Tipos de datos. Esquemas

```
/*  
 * Definición de un tipo de dato genérico «Dato» sobre el cual  
 * se van a plantear los esquema algorítmicos que se presentan  
 * en este tema.  
 */  
  
struct Dato {  
    tipoCampo1 c1;           // campo 1º del registro  
    tipoCampo2 c2;           // campo 2º del registro  
    ...  
    tipoCampok ck;           // campo k-ésimo del registro  
};
```

# Tipos de datos. Ejemplos

```
const int MAX_LONG_NOMBRE = 24;
const int MAX_LONG_APELLIDOS = 24;

/*
 * Representa la información relevante de un ciudadano:
 * nombre y apellidos, número de identificación fiscal,
 * fecha de nacimiento, estado civil y sexo
 */
struct Ciudadano {
    char nombre[MAX_LONG_NOMBRE];           // su nombre
    char apellidos[MAX_LONG_APELLIDOS];     // su(s) apellido(s)
    Nif nif;                                // su número NIF
    Fecha nacimiento;                       // su fecha de nacimiento
    bool estaCasado;                         // true: casado, false: soltero
    bool esMujer;                           // true: mujer, false: hombre
};
```



# Forma de presentación de los algoritmos

---

- Presentación del esquema con el tipo de datos genérico Dato
- Presentación de ejemplos con el tipo de datos Ciudadano
  - Código abundantemente documentado en los apuntes, en la página web de la asignatura y, en ocasiones, en estas transparencias
  - En ocasiones, en estas transparencias, código sin documentación adicional (complementado con la explicación del profesor en el aula)

# Índice

---

- **Algoritmos de recorrido**
- Algoritmos de búsqueda
  - Secuencial
  - Binaria
- Algoritmos de distribución
- Algoritmos de ordenación
  - Por selección

# Algoritmos de recorrido

## Esquema

```
/*  
 * Pre:   $n \geq 0$  y «T» tiene al menos «n» componentes.  
 * Post: Se han tratado los datos de las primeras «n»  
 *       componentes del vector «T».  
 */  
void recorrer(const Dato T[], const int n) {  
    [ Acciones previas al tratamiento de los datos de «T» ]  
    for (int i = 0; i < n; i++) {  
        /* Se han tratado las primeras i-1 componentes de «T» */  
        [ Trata ahora el elemento T[i] ]  
        /* Se han tratado las primeras i componentes de «T» */  
    }  
    [ Acciones posteriores al tratamiento de las primeras «n» componentes de «T» ]  
}
```

## Ejemplo: Mostrar

```
/*  
 * Pre:  $n \geq 0$  y «T» tiene al menos «n»  
 * componentes.  
 * Post: Presenta por pantalla un listado de  
 * la información de los ciudadanos de  
 * las primeras «n» componentes del  
 * vector «T», a razón de un ciudadano  
 * por línea y añade una línea  
 * adicional en blanco.  
 */  
void mostrar(const Ciudadano T[],  
             const int n);
```



## Ejemplo: Mostrar

```
void mostrar(const Ciudadano T[],
             const int n) {
    for (int i = 0; i < n; i++) {
        // Se han mostrado Los ciudadanos de
        // Las primeras i-1 componentes de «T»
        mostrar(T[i]);
        // Se han mostrado Los ciudadanos de
        // Las primeras «i» componentes de «T»
    }
    /* Escribe por pantalla una línea en
       * blanco adicional. */
    cout << endl;
}
```

## Ejemplo: Mostrar

```
void mostrar(const Ciudadano T[],  
             const int n) {  
    for (int i = 0; i < n; i++) {  
        mostrar(T[i]);  
    }  
    cout << endl;  
}
```

## Ejemplo. Contar

```
/*  
 * Pre:  $n \geq 0$  y «T» tiene al  
 * menos «n» componentes.  
 * Post: Ha devuelto el número  
 * de solteros de las  
 * primeras «n»  
 * componentes del vector  
 * «T».  
 */  
int numSolteros(const Ciudadano T[],  
                const int n);
```

# Ejemplo. Contar

```
int numSolteros(const Ciudadano T[], const int n) {  
    /* Aún no se ha identificado ningún soltero. */  
    int cuenta = 0;  
    for (int i = 0; i < n; ++i) {  
        /* cuenta == nº de solteros de las primeras i-1  
        * componentes de «T» */  
        if (!T[i].estaCasado) {  
            cuenta = cuenta + 1;  
        }  
        /* cuenta == nº de solteros de las primeras «i»  
        * componentes de «T» */  
    }  
    /* cuenta == nº de solteros de las primeras «n»  
    * componentes de «T» */  
    return cuenta;  
}
```

## Ejemplo. Contar

```
int numSolteros(const Ciudadano T[],
                const int n) {
    int cuenta = 0;
    for (int i = 0; i < n; ++i) {
        if (!T[i].estaCasado) {
            cuenta = cuenta + 1;
        }
    }
    return cuenta;
}
```

# Ejemplo.

## Determinación de mínimos o máximos

```
/* Pre:   $n > 0$  y «T» tiene al menos  
*       «n» componentes.  
* Post: Ha devuelto el ciudadano de  
*       más edad de entre las  
*       primeras «n» componentes del  
*       vector «T».  
*/  
Ciudadano masEdad(const Ciudadano T[],  
                  const int n);
```

# Ejemplo.

## Determinación de mínimos o máximos

```
Ciudadano masEdad(const Ciudadano T[], const int n) {  
    // indMayor == índice del ciudadano de más edad;  
    // inicialmente: primera componente del vector «T»  
    int indMayor = 0;  
    for (int i = 1; i < n; i++) {  
        // indMayor == índice del ciudadano de más edad de entre  
        // las primeras «i» - 1 componentes de «T»  
        if (esMayor(T[i], T[indMayor])) {  
            indMayor = i;  
        }  
        // indMayor == índice del ciudadano de más edad de entre  
        // las primeras «i» componentes de «T»  
    }  
    // indMayor == índice del más viejo en las primeras «n»  
    // componentes del vector «T»  
    return T[indMayor];  
}
```

# Ejemplo.

## Determinación de mínimos o máximos

```
Ciudadano masEdad(const Ciudadano T[],  
                  const int n) {  
    int indMayor = 0;  
    for (int i = 1; i < n; i++) {  
        if (T[i].nacimiento.dia + 100 * T[i].nacimiento.mes  
            + 10000 * T[i].nacimiento.ano  
            < T[indMayor].nacimiento.dia  
            + 100 * T[indMayor].nacimiento.mes  
            + 10000 * T[indMayor].nacimiento.ano) {  
            indMayor = i;  
        }  
    }  
    return T[indMayor];  
}
```



# Ejemplo.

## Determinación de mínimos o máximos

```
Ciudadano masEdad(const Ciudadano T[],  
                  const int n) {  
    int indMayor = 0;  
    for (int i = 1; i < n; i++) {  
        if (esMayor(T[i], T[indMayor])) {  
            indMayor = i;  
        }  
    }  
    return T[indMayor];  
}
```

**Código documentado:** en las transparencias anteriores y en

<http://webdiis.unizar.es/asignaturas/PROG1/doc/programacionCPP/Capitulo12/algoritmos-recorrido.cpp>

# Ejemplo.

## Determinación de mínimos o máximos

```
/*  
    * Pre: ---  
    * Post: Ha devuelto «true» si y solo si  
    * La fecha de nacimiento del  
    * «ciudadano1» es estrictamente  
    * anterior a la fecha de nacimiento del  
    * «ciudadano2».  
*/  
bool esMayor(const Ciudadano ciudadano1,  
              const Ciudadano ciudadano2);
```

ciudadano.h

# Ejemplo.

## Determinación de mínimos o máximos

ciudadano.cpp

```
bool esMayor(const Ciudadano ciudadano1,  
             const Ciudadano ciudadano2) {  
    return esAnterior(  
        fechas.hy  
        fechas.cpp  
        fechaNacimientoCompacta(ciudadano1),  
        fechaNacimientoCompacta(ciudadano2));  
}
```

# Ejemplo.

## Determinación de mínimos o máximos

```
/*  
    * Pre: ---  
    * Post: Ha devuelto la fecha de nacimiento del  
    * ciudadano «c» en formato «aaaammdd».  
    */  
int fechaNacimientoCompacta(const Ciudadano c) {  
    int fecha;  
    componer(c.nacimiento.dia, c.nacimiento.mes,  
              c.nacimiento.agno, fecha);  
    return fecha;  
}
```

ciudadano.cpp

fechas.hy  
fechas.cpp

# Índice

---

- Algoritmos de recorrido
- **Algoritmos de búsqueda**
  - Secuencial
  - Binaria
- Algoritmos de distribución
- Algoritmos de ordenación
  - Por selección

# Algoritmos de búsqueda. Esquema

```
/*  
 * Pre:  $n \geq 0$  y «T» tiene al menos «n» componentes.  
 * Post: Si entre los datos de las primeras «n»  
 *       componentes del vector «T» hay uno que  
 *       satisface el criterio de búsqueda, entonces  
 *       ha devuelto el índice de dicho elemento en  
 *       el vector; si no lo hay, ha devuelto un  
 *       valor negativo.  
 */  
int busqueda(const Dato T[], const int n);
```

# Algoritmos de búsqueda

## Esquema (1/2)

```
int busqueda(const Dato T[], const int n) {  
    int i = 0;  
    // Espacio inicial de búsqueda: Las componentes del vector «T» indexadas  
    // entre «i» (== 0) y «n» - 1, ambas inclusive  
    bool encontrado = false;  
    while (!encontrado && i < n) {  
        // Sin éxito tras buscar en las primeras i-1 componentes de «T»  
        [Analiza el elemento T[i]]  
        if (T[i] satisface el criterio de búsqueda) {  
            encontrado = true;  
        }  
        else {  
            i = i + 1;  
        }  
    }  
    // encontrado || i ≥ n  
    ...  
}
```

# Algoritmos de búsqueda

## Esquema (2/2)

```
int busqueda(const Dato T[], const int n) {  
    ...  
    // encontrado || i ≥ n  
  
    // Discriminación del éxito de la búsqueda  
    if (encontrado) {  
        return i;  
    }  
    else {  
        return -1;  
    }  
}
```



## Ejemplo. Buscar por DNI

```
/*  
 * Pre:  $n > 0$  y «T» tiene al menos «n»  
 * componentes.  
 * Post: Si entre los ciudadanos almacenados en  
 * las primeras «n» componentes del vector  
 * «T» hay uno cuyo DNI es igual a  
 * «dniBuscado», entonces ha devuelto el  
 * índice de dicho elemento en el vector;  
 * si no lo hay, ha devuelto un negativo.  
 */  
int buscar(const Ciudadano T[], const int n,  
           const int dniBuscado);
```

# Ejemplo. Buscar por DNI

```
int buscar(const Ciudadano T[], const int n,  
           const int dniBuscado) {  
    int i = 0;  
    bool encontrado = false;  
    /* Búsqueda */  
    while (!encontrado && i < n) {  
        if (T[i].nif.dni == dniBuscado) {  
            encontrado = true;  
        }  
        else {  
            i = i + 1;  
        }  
    }  
    // encontrado || i ≥ n  
    ...  
}
```

## Ejemplo. Buscar por DNI

```
int buscar(const Ciudadano T[], const int n,  
          const int dniBuscado) {  
    ...  
    // encontrado ||  $i \geq n$   
    /* Discriminación del éxito */  
    if (encontrado) {  
        return i;  
    }  
    else {  
        return -1;  
    }  
}
```

# Ejemplo. Buscar por DNI

```
int buscar(const Ciudadano T[], const int n,  
          const int dniBuscado) {  
    int i = 0; bool encontrado = false;  
    while (!encontrado && i < n) {  
        if (T[i].nif.dni == dniBuscado) {  
            encontrado = true;  
        }  
        else {  
            i = i + 1;  
        }  
    }  
    if (encontrado)  
        return i;  
    else  
        return -1;  
}
```

} **Código documentado:** en las transparencias anteriores y en  
<http://webdiis.unizar.es/asignaturas/PROG1/doc/programacionCPP/Capitulo12/algoritmos-busqueda.cpp>

# Algoritmos de búsqueda

## Esquema con garantía de éxito

```
/*
 * Pre:   $n > 0$ , «T» tiene al menos «n» componentes y entre los datos de las
 *        primeras «n» componentes del vector «T» hay uno que satisface el
 *        criterio de búsqueda.
 * Post: Ha devuelto el índice de un elemento de las primeras «n» componentes
 *        del vector «T» que satisface el criterio de búsqueda.
 */
int busqueda(const Dato T[], const int n) {
    /* Búsqueda secuencial con garantía de éxito */
    int i = 0;
    while (no satisface T[i] el criterio de búsqueda) {
        /* Se han descartado las primeras «i» componentes de «T» */
        i = i + 1;
        /* Espacio búsqueda: componentes de «T» indexadas en [i, n-1] */
    }
    return i;
}
```



# Búsquedas en vectores ordenados

---

- Adivinar un número del 1 al 10
- Preguntas disponibles:
  - ¿Es el número  $i$ ?, con  $i \in \mathbb{N}$
- El número es el 6



# Búsquedas en vectores ordenados

---

- Adivinar un número del 1 al 10000
- Preguntas disponibles:
  - ¿Es el número  $i$ ?, con  $i \in \mathbb{N}$

# Búsquedas en vectores ordenados

---

- Adivinar un número del 1 al 10000
  - Preguntas disponibles:
    - ~~¿Es el número  $i$ ?~~
    - ¿Es mayor que  $i$ ?
    - ~~¿Es menor que  $i$ ?~~
- con  $i \in \mathbb{N}$



# Búsquedas en vectores ordenados

---

		[1, 10000]
1.	¿Es mayor que 5000? No	→ [1, 5000]
2.	¿Es mayor que 2500? Sí	→ [2501, 5000]
3.	¿Es mayor que 3750? Sí	→ [3751, 5000]
4.	¿Es mayor que 4375? Sí	→ [4376, 5000]
5.	¿Es mayor que 4688? Sí	→ [4689, 5000]
6.	¿Es mayor que 4844? Sí	→ [4845, 5000]
7.	¿Es mayor que 4922? No	→ [4845, 4922]
8.	¿Es mayor que 4883? No	→ [4845, 4883]
9.	¿Es mayor que 4864? Sí	→ [4865, 4883]
10.	¿Es mayor que 4874? No	→ [4865, 4874]
11.	¿Es mayor que 4869? Sí	→ [4870, 4874]
12.	¿Es mayor que 4872? No	→ [4870, 4872]
13.	¿Es mayor que 4871? No	→ [4870, 4871]
14.	¿Es mayor que 4870? No	→ [4870, 4870]

# Búsqueda binaria

dniBuscado = 30

t

0	1	2	3	4	5	6	7	8	9
dni 8	dni 15	dni 30	dni 35	dni 60	dni 68	dni 70	dni 71	dni 77	dni 79
nom "C..."	nom "B..."	nom "C..."	nom "F..."	nom "G..."	nom "G..."	nom "G..."	nom "L..."	nom "I..."	nom "J..."
nac ...	nac ...	nac ...	nac ...	nac ...	nac ...	nac ...	nac ...	nac ...	nac ...
casd true	casd fals	casd fals	casd true	casd true	casd true	casd fals	casd fals	casd true	casd fals



# Algoritmo de búsqueda binaria

## Esquema

```
/*  
 * Pre:   $n > 0$ , «T» tiene al menos «n» componentes y los  
 *       elementos de las primeras «n» componentes del vector  
 *       «T» están ordenados de menor a mayor valor.  
 * Post: Si entre los datos almacenados en las primeras «n»  
 *       componentes del vector «T» hay uno que satisface el  
 *       criterio de búsqueda, entonces ha devuelto el índice de  
 *       dicho elemento; si no lo hay, ha devuelto un valor  
 *       negativo.  
 */  
int buquedaBinaria(const Dato T[], const int n);
```



# Algoritmo de búsqueda binaria

## Esquema

```
int buquedaBinaria(const Dato T[], const int n,
                   datoBuscado) {
    int inf = 0;
    int sup = n - 1;
    /* Búsqueda */
    while (inf < sup) {
        int medio = (inf + sup) / 2;
        if (el valor de T[medio] es inferior al de datoBuscado) {
            inf = medio + 1;
        }
        else {
            sup = medio;
        }
    }
    ...
}
```



# Algoritmo de búsqueda binaria

## Esquema

```
int buquedaBinaria(const Dato T[],  
                   const int n, datoBuscado) {  
    ...  
    /* Discriminación del éxito */  
    if (T[inf] satisface el criterio de búsqueda) {  
        return inf;  
    }  
    else {  
        return -1;  
    }  
}
```

# Algoritmo de búsqueda binaria

## Ejemplo

```
/*  
 * Pre:   $n > 0$ , «T» tiene al menos «n» componentes y los  
 *       datos de las primeras «n» componentes del vector  
 *       «T» están ordenados por valores del DNI  
 *       crecientes.  
 * Post: Si entre los ciudadanos almacenados en las  
 *       primeras «n» componentes del vector «T»  
 *       hay uno cuyo DNI es igual a «dniBuscado»,  
 *       entonces ha devuelto el índice de dicho elemento  
 *       en el vector; si no lo hay, ha devuelto un valor  
 *       negativo.  
 */  
int buscar(const Ciudadano T[], const int n,  
           const int dniBuscado);
```



# Algoritmo de búsqueda binaria

## Ejemplo

```
int buscar(const Ciudadano T[], const int n,  
           const int dniBuscado) {  
    // Espacio de búsqueda: establecimiento en T[0..n-1]  
    int inf = 0;  
    int sup = n - 1;  
  
    /* Búsqueda */  
    ...  
  
    /* Discriminación del éxito */  
    ...  
}
```



# Algoritmo de búsqueda binaria

## Ejemplo

```
...  
/* Búsqueda */  
// Espacio de búsqueda: T[0..n-1]  
while (inf < sup) {  
    // Espacio de búsqueda: T[inf..sup]  
    int medio = (inf + sup) / 2;  
    if (dniBuscado > T[medio].nif.dni) {  
        // Espacio de búsqueda: T[medio+1..sup]  
        inf = medio + 1;  
    }  
    else {  
        // Espacio de búsqueda: T[inf..medio]  
        sup = medio;  
    }  
    // Espacio de búsqueda: T[inf..sup]  
}  
// inf >= sup  
// Espacio de búsqueda: T[inf]  
...
```





# Algoritmo de búsqueda binaria

## Ejemplo

```
int buscar(const Ciudadano T[], const int n,  
          const int dniBuscado) {  
  
    ...  
  
    /* Discriminación del éxito */  
    if (T[inf].nif.dni == dniBuscado) {  
        return inf;  
    }  
    else {  
        return -1;  
    }  
}
```



# Algoritmo de búsqueda binaria

## Ejemplo

```
int buscar(const Ciudadano T[], const int n,
           const int dniBuscado) {
    int inf = 0;
    int sup = n - 1;
    while (inf < sup) {
        int medio = (inf + sup) / 2;
        if (dniBuscado > T[medio].nif.dni)
            inf = medio + 1;
        else
            sup = medio;
    }
    if (T[inf].nif.dni == dniBuscado)
        return inf;
    else
        return -1;
}
```

**Código documentado:** en las transparencias anteriores y en

<http://webdiis.unizar.es/asignaturas/PROG1/doc/programacionCPP/Capitulo12/algoritmos-busqueda.cpp>

# Índice

---

- Algoritmos de recorrido
- Algoritmos de búsqueda
  - Secuencial
  - Binaria
- **Algoritmos de distribución**
- Algoritmos de ordenación
  - Por selección

# Algoritmo de distribución. Esquema

```
/*  
 * Pre:  $n > 0$ , «T» tiene al menos «n» componentes y sea k el  
 * número de elementos de las primeras «n» componentes del  
 * vector «T» que satisfacen una determinada propiedad P.  
 * Post: Las primeras «n» componentes del vector «T» son una  
 * permutación de los datos iniciales de «T» en la que  
 * todos los elementos de las primeras «k-1» componentes  
 * del vector «T» satisfacen la propiedad P y ninguno de  
 * los elementos de las componentes del vector «T» con  
 * índices entre «k» y  $n-1$ , ambos inclusive, la satisface.  
 */  
void distribuir(Dato T[], const int n);
```

# Algoritmo de distribución. Esquema

```
void distribuir (Dato T[], const int n) {  
    int inf = 0;  
    int sup = n - 1;  
    while (inf < sup) {  
        if (T[inf] satisface P) {  
            inf = inf + 1;  
        }  
        else if (T[sup] no satisface P) {  
            sup = sup - 1;  
        }  
        else {  
            Dato aux = T[inf]; T[inf] = T[sup]; T[sup] = aux;  
            inf = inf + 1; sup = sup - 1;  
        }  
    }  
}
```

**Código documentado:** en los apuntes y en  
<http://webdiis.unizar.es/asignaturas/PROG1/doc/programacionCPP/Capitulo12/algoritmos-modificacion.cpp>

# Algoritmo de distribución. Ejemplo

```
/*  
 * Pre:   $n > 0$  y «T» tiene al menos «n» componentes.  
 * Post: Las primeras «n» componentes del vector «T»  
 *       es una permutación de los datos iniciales  
 *       de «T» en la que todos los ciudadanos  
 *       solteros tienen un índice en el vector  
 *       menor que cualquier ciudadano casado.  
 */  
void distribuir(Ciudadano T[], const int n);
```

# Algoritmo de distribución. Ejemplo

```
void distribuir(Ciudadano T[], const int n) {  
    int inf = 0;    int sup = n - 1;  
    while (inf < sup) {  
        if (!T[inf].estaCasado) {  
            inf++;  
        }  
        else if (T[sup].estaCasado) {  
            sup--;  
        }  
        else {  
            permutar(T[inf], T[sup]);  
            inf++; sup--;  
        }  
    }  
}
```

**Código documentado:** en las transparencias anteriores y en

<http://webdiis.unizar.es/asignaturas/PROG1/doc/programacionCPP/Capitulo12/algoritmos-modificacion.cpp>

# Algoritmo de distribución. Ejemplo

```
/*  
 * Pre:  uno = A y otro = B  
 * Post: uno = B y otro = A  
 */  
void permutar(Ciudadano& uno, Ciudadano& otro) {  
    Ciudadano aux = uno;  
    uno = otro;  
    otro = aux;  
}
```



# Índice

---

- Algoritmos de recorrido
- Algoritmos de búsqueda
  - Secuencial
  - Binaria
- Algoritmos de distribución
- **Algoritmos de ordenación**
  - Por selección

# Algoritmos de ordenación

## Especificación

```
/*  
 * Pre:   $n > 0$  y «T» tiene al menos «n» componentes.  
 * Post: El contenido de las primeras «n»  
 *       componentes del vector «T» es una  
 *       permutación del contenido iniciales de «T»  
 *       en la que todos ellos están ordenados según  
 *       un determinado criterio C.  
 */  
void ordenar(Dato T[], const int n);
```

# Ordenación por selección

## Esquema

```
void ordenacion (Dato T[], const int n) {  
    for (int i = 0; i < n - 1; i++) {  
        // Los elementos de las primeras «i» - 1 componentes del vector  
        // «T» ya están ordenados según el criterio C.  
        int iMenor = i;  
        for (int j = i + 1; j < n; j++) {  
            if (T[j] es menor que T[iMenor] según C) {  
                iMenor = j;  
            }  
        }  
        // T[iMenor] es el menor de T[i..n-1]. Permuta T[i] y T[iMenor]  
        Dato aux = T[i];  
        T[i] = T[iMenor];  
        T[iMenor] = aux;  
        // Los elementos de las primeras «i» componentes del vector  
        // «T» ya están ordenados según el criterio C.  
    }  
    // Los elementos de las primeras «i» - 1 componentes del vector  
    // «T» ya están ordenados según el criterio C.  
}
```

# Ordenación por selección

## Ejemplo

```
/*  
 * Pre:   $n > 0$  y «T» tiene al menos «n»  
 * componentes.  
 * Post: El contenido de las primeras «n»  
 * componentes del vector «T» es una  
 * permutación del contenido inicial de  
 * «T» en la que todos ellos están  
 * ordenados de forma que tienen valores  
 * del DNI crecientes.  
 */  
void ordenar(Ciudadano T[], const int n);
```

# Ordenación por selección

## Esquema

```
void ordenar(Ciudadano T[], const int n) {  
    // Ordenación de un vector por el método de selección  
    for (int i = 0; i < n - 1; i++) {  
        /* Los ciudadanos de las primeras i-1 componentes de «T» son los de menor  
         * valor de DNI y ya están ordenados */  
  
        // Selección del ciudadano con menor valor de DNI de T[i..n-1]  
        int iMenor = i;  
        for (int j = i + 1; j < n; j++) {  
            // T[iMenor] es el de menor DNI de T[i..j-1]  
            if (T[j].nif.dni < T[iMenor].nif.dni) {  
                iMenor = j;  
            }  
            // T[iMenor] es el de menor DNI de T[i..j]  
        }  
        // T[iMenor] es el de menor DNI de T[i..n-1]. Permuta T[i] y T[iMenor]  
        permutar(T[i], T[iMenor]);  
  
        /* Los ciudadanos de las primeras i-1 componentes del vector «T» son los  
         * de menor valor de DNI y ya están ordenados */  
    }  
}
```

Esquema de búsqueda  
de mínimo (en solo una  
parte del vector)

# Algoritmo de ordenación por selección

## □ Select-sort with Gypsy folk dance

### ■ Extraído de *I Programmer*

<http://www.i-programmer.info/news/150-training-a-education/2255-sorting-algorithms-as-dances.html>



# Índice

---

- ❑ Algoritmos de recorrido
- ❑ Algoritmos de búsqueda
  - Secuencial
  - Binaria
- ❑ Algoritmos de distribución
- ❑ Algoritmos de ordenación
  - Por selección