Programación 1

Tema 12

Algoritmos con vectores

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

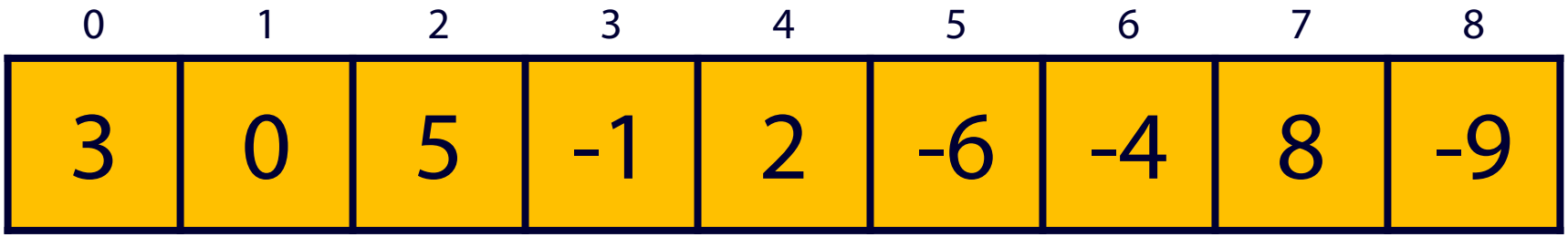
Índice

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

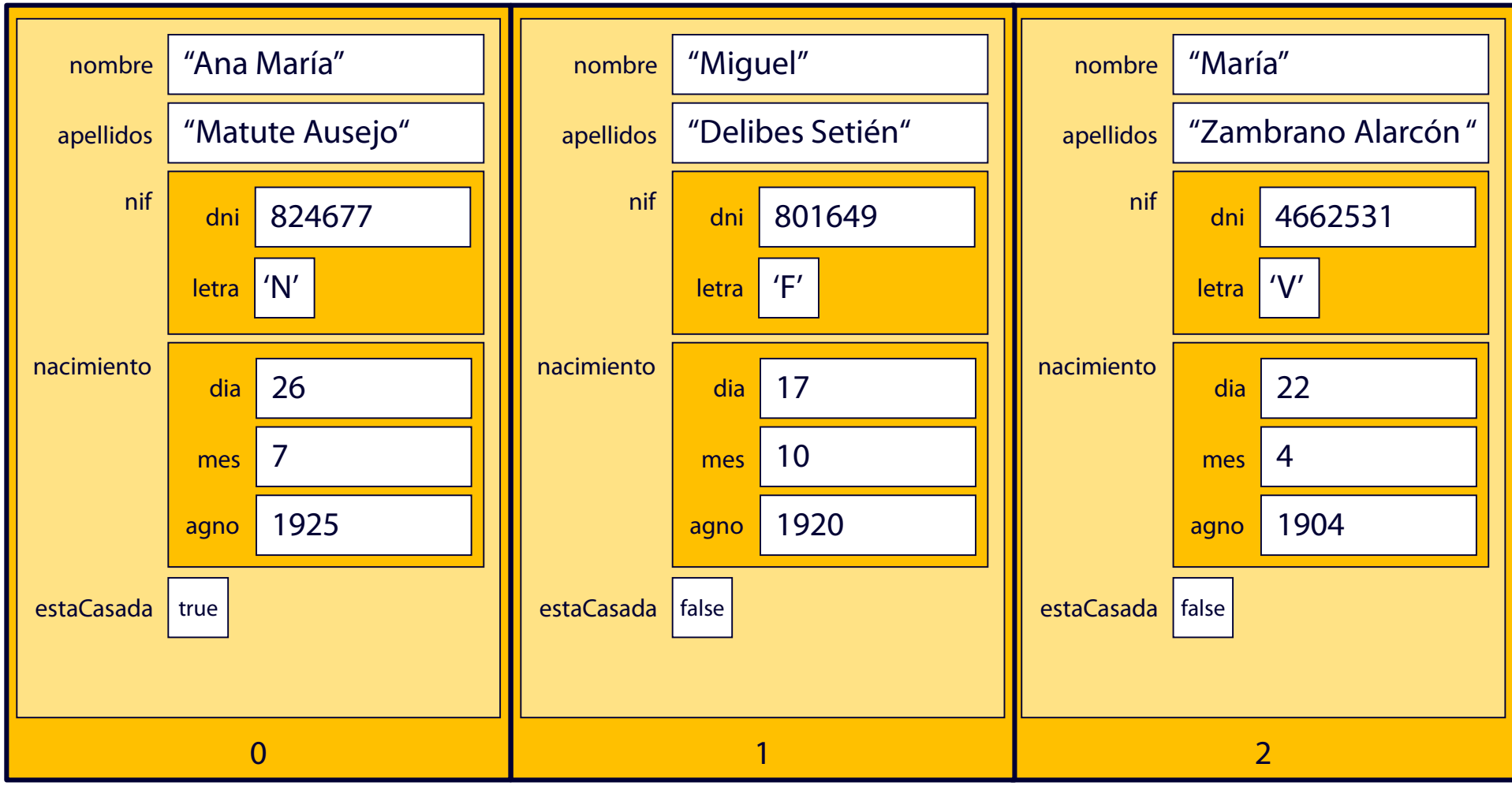
Estructura de estas transparencias

- En clase
  - Algoritmos con vectores de enteros
  - Algunos con vectores de registros
- Apuntes del profesor Javier Martínez (Moodle)
  - Explicaciones textuales
  - Esquema genérico de los algoritmos
  - Algoritmos con vectores de registros de tipo `Persona`
- En el repositorio de GitHub
  - Algoritmos con vectores de enteros
  - Algoritmos con vectores de registros de tipo `Persona`

Ejemplo de vector de enteros



Ejemplo de vector de Personas



Índice

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

Algoritmo de recorrido.
Escribir en pantalla

```
/*
 * Pre:  «T» tiene al menos «n» componentes.
 * Post: Escribe en la pantalla un listado
 *       con los elementos de las primeras «n»
 *       componentes del vector «T», a razón
 *       de un elemento por línea.
 */
void mostrar(const int T[], const unsigned n);
```

Algoritmo de recorrido.

Escribir en pantalla

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

Algoritmo de recorrido.
Contar negativos

```
/*
 * Pre:  «T» tiene al menos «n»
 *       componentes.
 * Post: Devuelve el número de
 *       datos negativos de las
 *       primeras «n» componentes del
 *       vector «T».
 */
unsigned numNegativos(const int T[],
                      const unsigned n);
```

Algoritmo de recorrido.
Contar negativos

```
unsigned numNegativos(const int T[],
                      const unsigned n) {
    unsigned cuenta = 0;
    for (unsigned i = 0; i < n; i++) {
        if (T[i] < 0) {
            cuenta++;
        }
    }
    return cuenta;
}
```

Algoritmo de recorrido.
Cálculo del máximo

```
/*
 * Pre:  «T» tiene al menos «n»
 *       componentes y «n» > 0.
 * Post: Devuelve el máximo valor
 *       de entre los de las primeras
 *       «n» componentes del vector «T».
 */
int maximo(const int T[],
           const unsigned n);
```

Algoritmo de recorrido.
Cálculo del máximo

```
int maximo(const int T[],
           const unsigned n) {
    unsigned indMayor = 0;
    for (unsigned i = 1; i < n; i++) {
        if (T[i] > T[indMayor]) {
            indMayor = i;
        }
    }
    return T[indMayor];
}
```

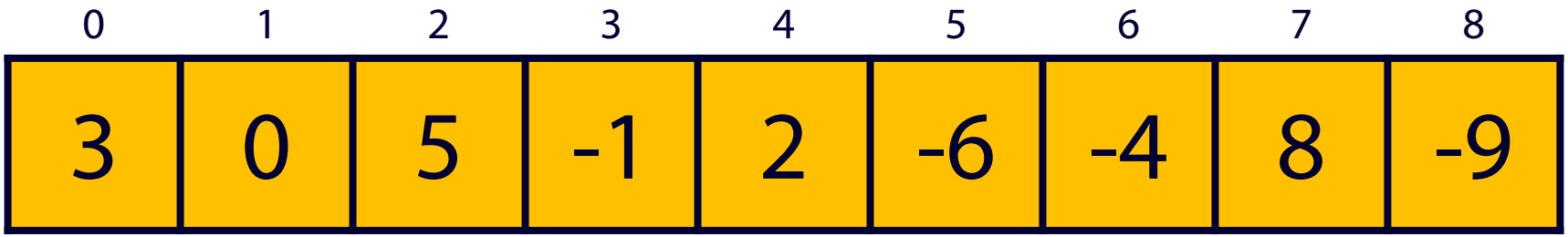
Índice

- Algoritmos de recorrido
- **Algoritmos de búsqueda**
  - Lineal o secuencial
  - Binaria o dicotómica
- Algoritmos de distribución
- Algoritmos de ordenación
  - Por selección

Algoritmos de búsqueda.
Búsqueda secuencial

```
/*
 * Pre:  «T» tiene al menos «n» componentes.
 * Post: Si entre los datos almacenados en las
 *       primeras «n» componentes del vector
 *       «T» hay uno cuyo valor es igual a
 *       «datoBuscado», entonces devuelve el
 *       índice de dicho elemento en el vector; si
 *       no lo hay, devuelve un dato negativo.
 */
int buscar(const int T[], const unsigned n,
           const int datoBuscado);
```

Ejemplo de vector




```
buscar(v, 9, -4) → 6
buscar(v, 9,  7) → -1
```

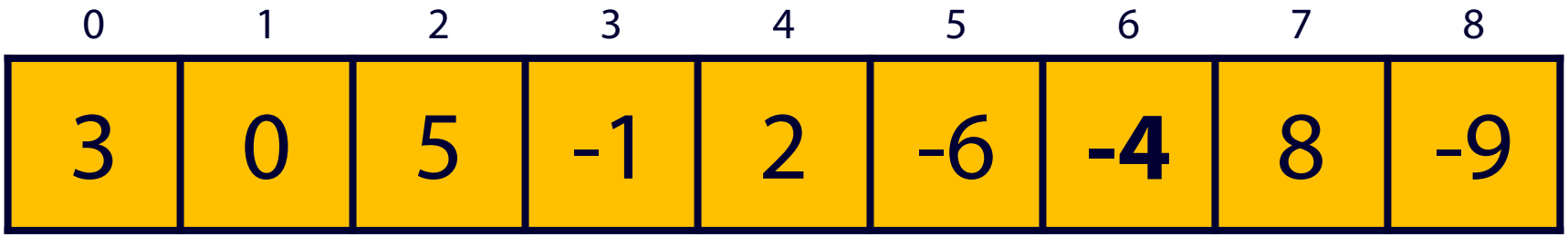
Algoritmos de búsqueda.
Búsqueda secuencial

```
int buscar(const int T[], const unsigned n, const int datoBuscado) {
    unsigned i = 0;
    bool encontrado = false;

    while (!encontrado && i < n) {
        if (T[i] == datoBuscado) {
            encontrado = true;
        }
        else {
            i = i + 1;
        }
    } // encontrado || i ≥ n

    if (encontrado) {
        return i;
    }
    else {
        return -1;
    }
}
```

Algoritmo de búsqueda.
Búsqueda con garantía de éxito

```
/*
 * Pre:  «T» tiene al menos «n» componentes y en al
 *       menos una de ellas se encuentra «datoBuscado».
 * Post: Si entre los datos almacenados en las
 *       primeras «n» componentes del vector
 *       «T» hay uno cuyo valor es igual a
 *       «datoBuscado», entonces devuelve el
 *       índice de dicho elemento en el vector; si no
 *       lo hay, devuelve un dato negativo.
 */
unsigned buscarGarantizado(const int T[],
                           const int datoBuscado);
```

Ejemplo de búsqueda con garantía de éxito




```
buscarGarantizado(v, -4) → 6
```

Algoritmo de búsqueda.

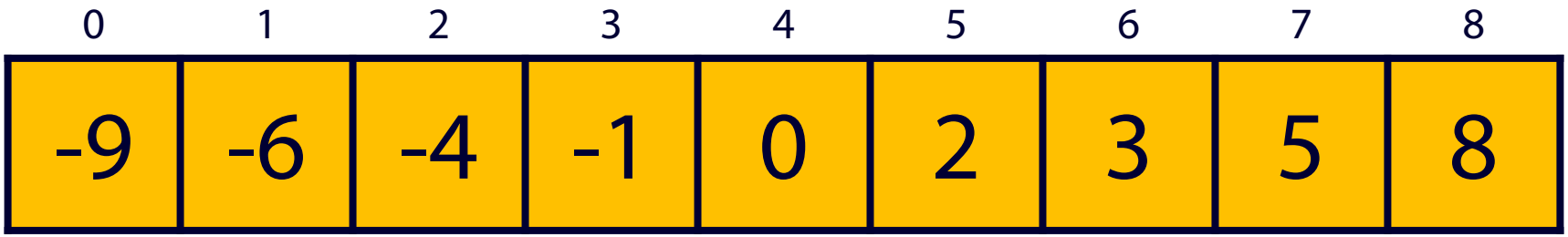
Búsqueda con garantía de éxito

```
unsigned buscarGarantizado(const int T[],
                           const int datoBuscado) {
    unsigned i = 0;

    /* Búsqueda */
    while (T[i] != datoBuscado) {
        i++;
    } // T[i] == datoBuscado

    return i;
}
```

Algoritmo de búsqueda.
Búsqueda dicotómica

```
/*
 * Pre:  «T» tiene al menos «n» componentes, y los
 *       elementos de las primeras «n» componentes del vector
 *       «T» están ordenados por valores crecientes.
 * Post: Si entre las personas almacenadas en las primeras «n»
 *       componentes del vector «T» hay una cuyo valor es igual
 *       a «datoBuscado», entonces devuelve el índice de
 *       dicho elemento en el vector; si no lo hay, devuelve
 *       un valor negativo.
 */
int buscarDicotomico(const int T[], const unsigned n,
                     const int datoBuscado);
```

Ejemplo de vector ordenado




```
buscar(v, 9, -4) → 2
buscar(v, 9,  7) → -1
```

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 $i \in \mathbb{N}$

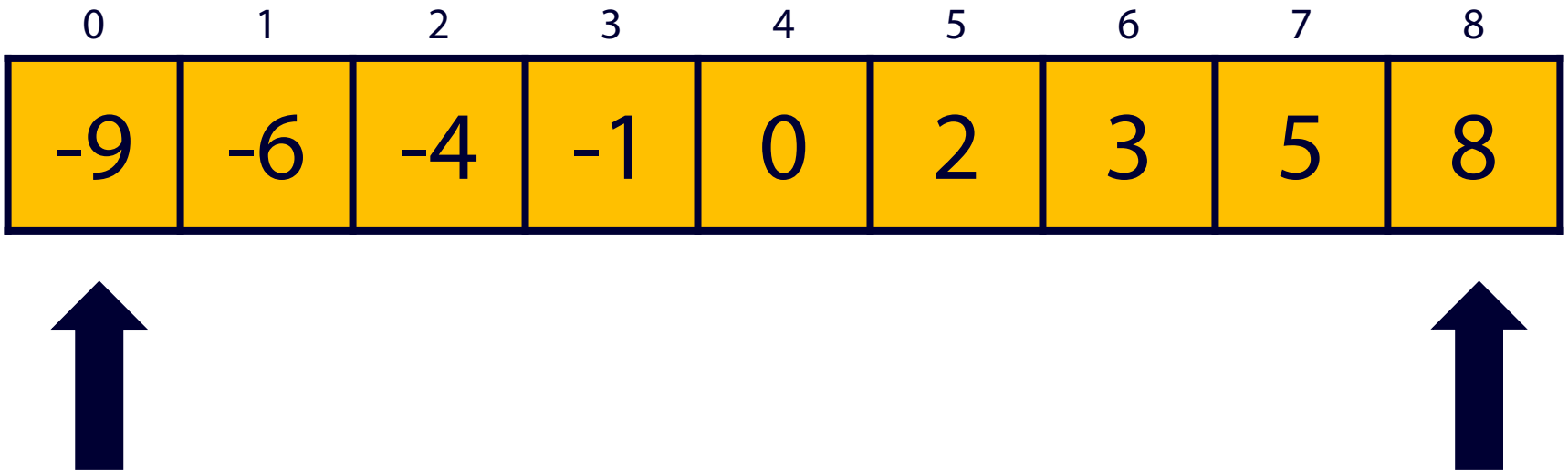
Búsquedas en vectores ordenados

[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]

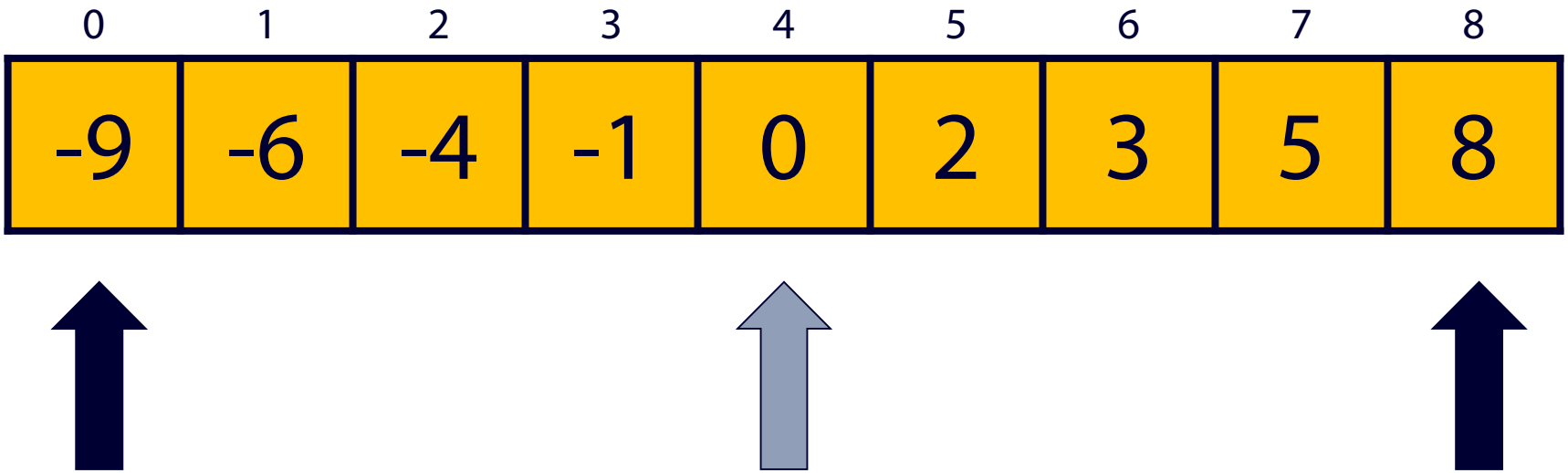
Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



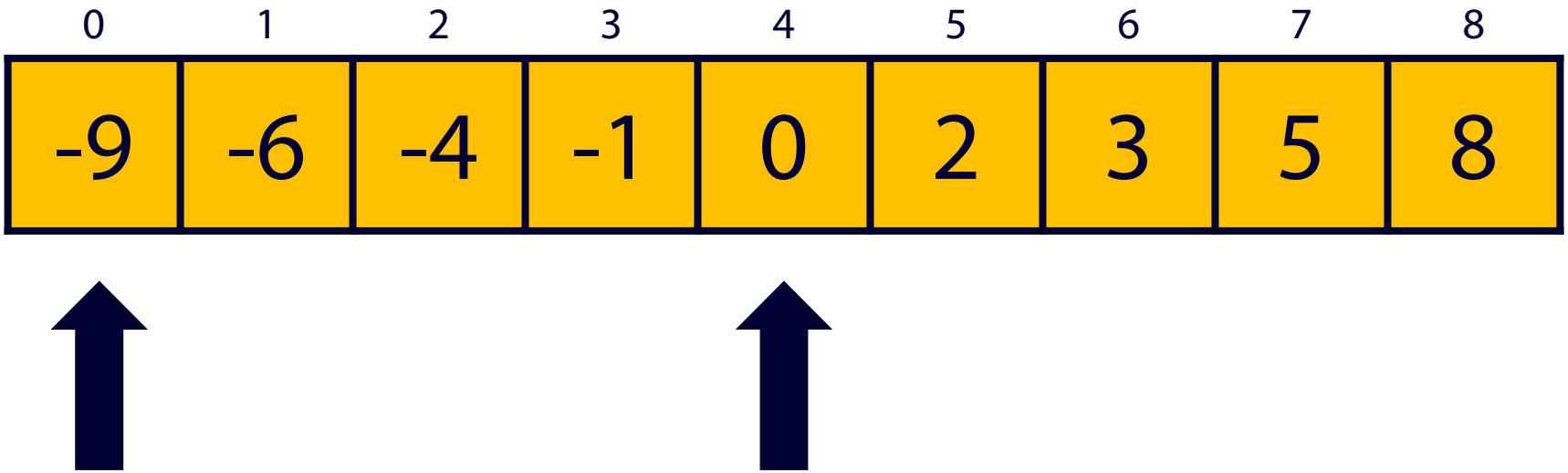
Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



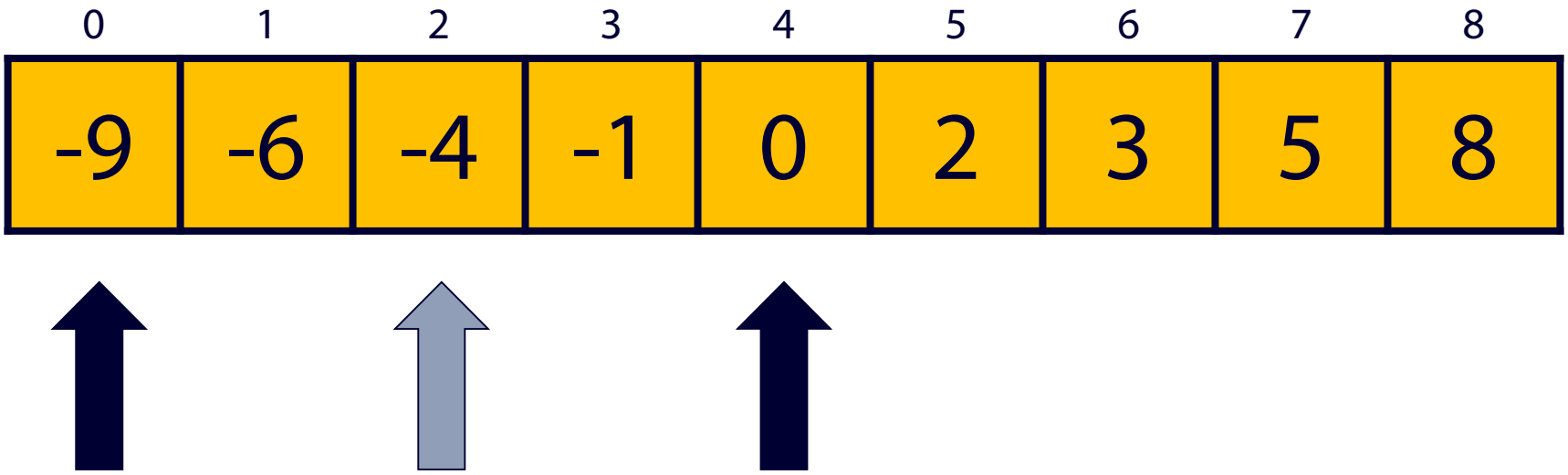
Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



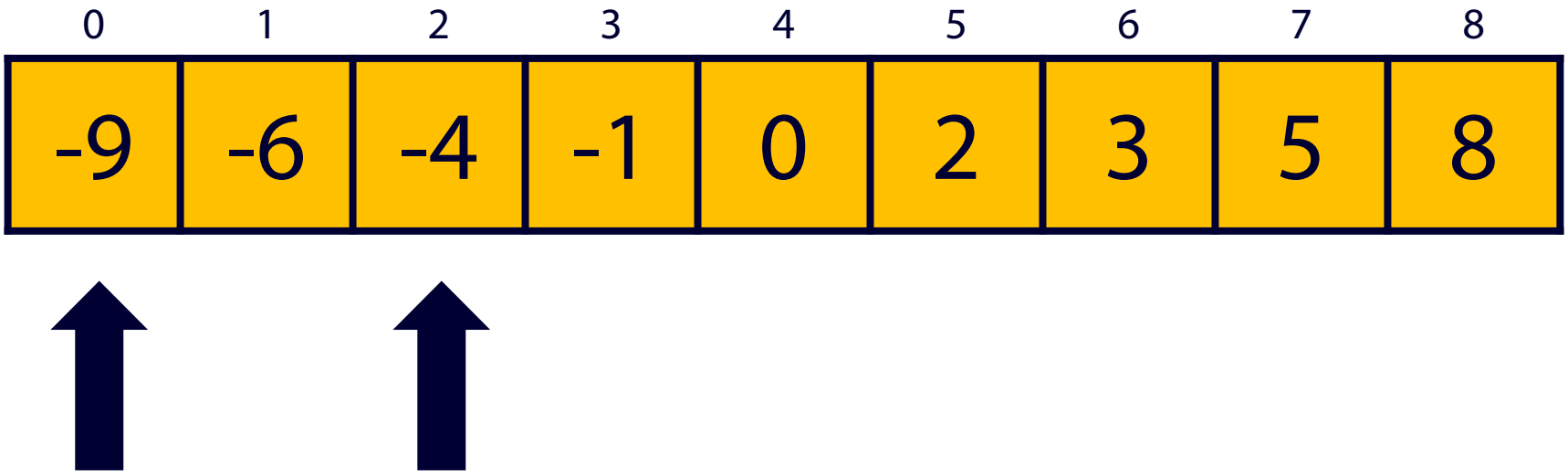
Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



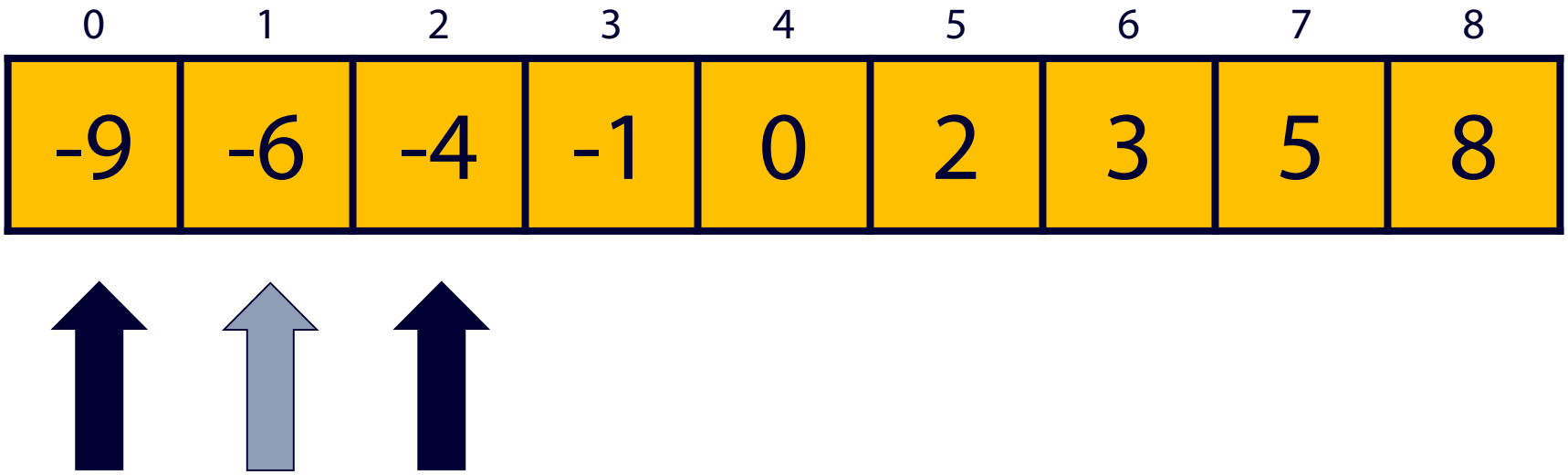
Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



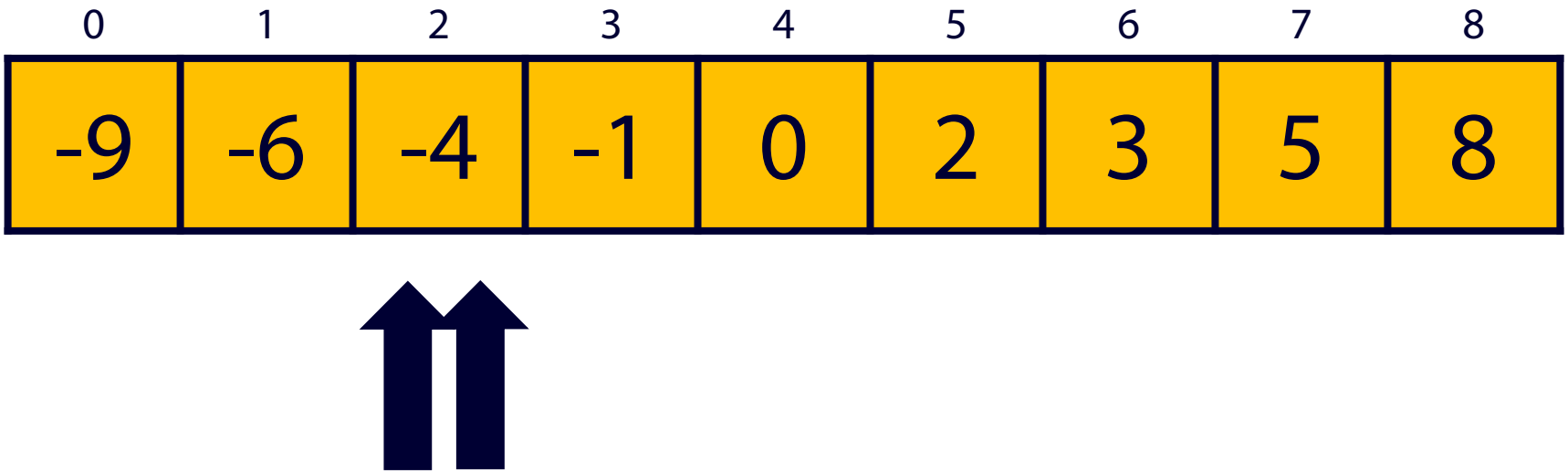
Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



Ejemplo de vector ordenado

```
buscar(v, 9, -4)
```



Algoritmo de búsqueda.
Búsqueda dicotómica

```
int buscarDicotomico(const int T[], const unsigned n,
                     const int datoBuscado) {
    if (n > 0) {
        unsigned inf = 0;
        unsigned sup = n - 1;

        while (inf < sup) {
            unsigned medio = (inf + sup) / 2;
            if (datoBuscado > T[medio]) {
                inf = medio + 1;
            }
            else {
                sup = medio;
            }
        }
        ...
```

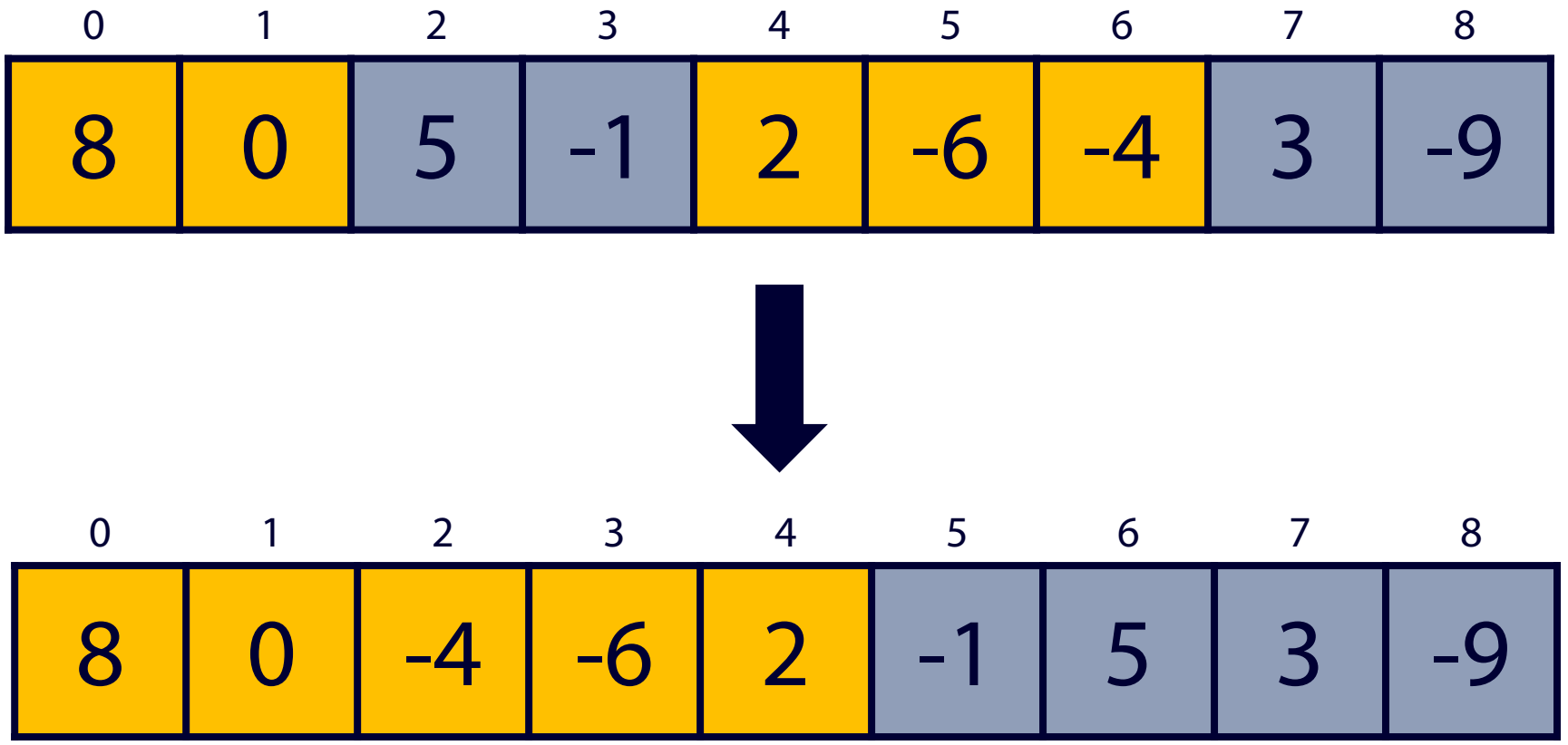
Algoritmo de búsqueda.
Búsqueda dicotómica

```
int buscarDicotomico(const int T[], const unsigned n,
                     const int datoBuscado) {
        ...

        if (T[inf] == datoBuscado) {
            return inf;
        }
        else {
            return -1;
        }
    }
    else { // n == 0
        return -1;
    }
}
```

Índice

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

Distribución
Pares vs. impares



Distribución

```
/*
 * Pre:  «T» tiene al menos «n» componentes.
 * Post: Las primeras «n» componentes del
 *       vector «T» son una permutación de los
 *       datos iniciales de «T» en la que
 *       todos los datos pares tienen un
 *       índice en el vector menor que
 *       cualquiera de los impares.
 */
void distribuir(int T[],
                const unsigned n);
```

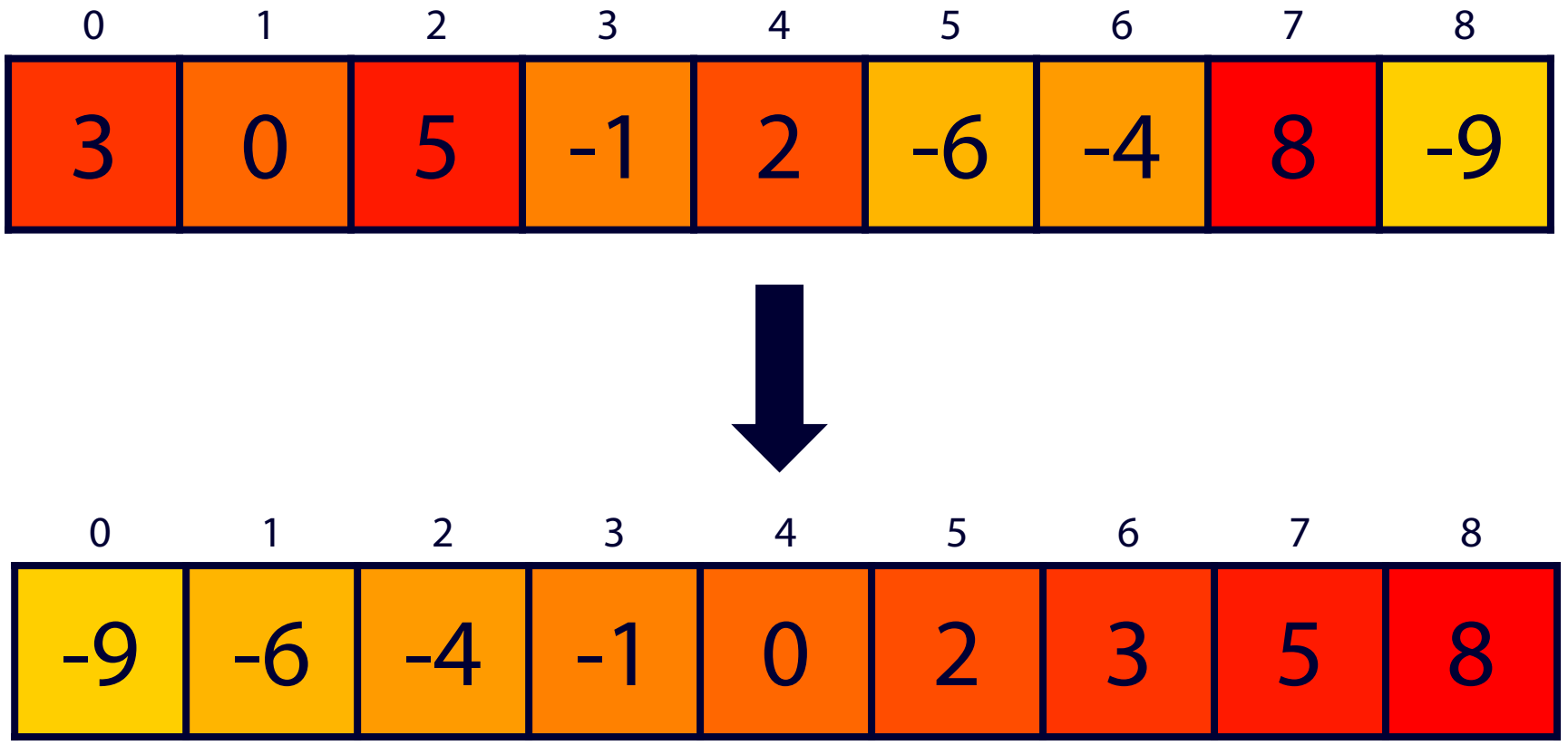
Distribución

```
void distribuir(int T[], const unsigned n) {
    int inf = 0;
    int sup = n - 1;
    while (inf < sup) {
        if (T[inf] % 2 == 0) {
            inf = inf + 1;
        }
        else if (T[sup] % 2 != 0) {
            sup = sup - 1;
        }
        else {
            permutar(T[inf], T[sup]);
            inf = inf + 1;
            sup = sup - 1;
        }
    }
}
```

Índice

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

Ordenación



Ordenación

```
/*
 * Pre:  «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(int T[], const unsigned n);
```

Ordenación

```
void ordenar(int T[], const unsigned n) {
    if (n != 0) {
        for (unsigned i = 0; i < n - 1; i++) {
            unsigned iMenor = i;
            for (unsigned j = i + 1; j < n; j++) {
                if (T[j] < T[iMenor]) {
                    iMenor = j;
                }
            }
            permutar(T[i], T[iMenor]);
        }
    }
}
```

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