Programación 2

Lección 16. Diseño de algoritmos de distribución y de ordenación de los datos de un vector

1. Especificación de algoritmos de distribución y de ordenación de los datos de un vector
2. Algoritmo de distribución de los datos de un vector
3. Algoritmos de ordenación de ordenación interna de los datos de un vector
   - Ordenación por selección directa
   - Ordenación por inserción directa
   - Ordenación por intercambio directo
   - Ordenación rápida o quicksort
4. Análisis comparativo de sus costes

1. Especificación de algoritmos de distribución y de ordenación de los datos de un vector

Consideraremos estos predicados parametrizados para facilitar la escritura de predicados más complejos:

```
esPermutación(v1,v2,desde,hasta) =
    (PT alfa EN [desde,hasta].
        (Núm beta EN [desde,hasta]. v1[beta] = v1[alfa])
        = (Núm beta EN [desde,hasta]. v2[beta] = v1[alfa]) )

ordenado(v,desde,hasta) =
    (PT alfa EN [desde,hasta-1]. v[alfa] <= v[alfa+1])
```

Función de distribución de los datos de un vector:

```
/*
 * Pre: v = Va  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Va,0,n-1) AND
 *       (PT alfa EN [0,n-1].
 *           (v[alfa] <= frontera ->
 *                (PT beta EN [0,alfa]. v[beta] <= frontera)) AND
 *           (v[alfa] > frontera ->
 *                (PT beta EN [alfa,n-1]. v[beta] > frontera)) )
 */
template <typename T>
void distribucion (T v[], const int n, const T frontera);
```

Función de ordenación de los datos de un vector:

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void ordenacion (T v[], const int n) ;
```

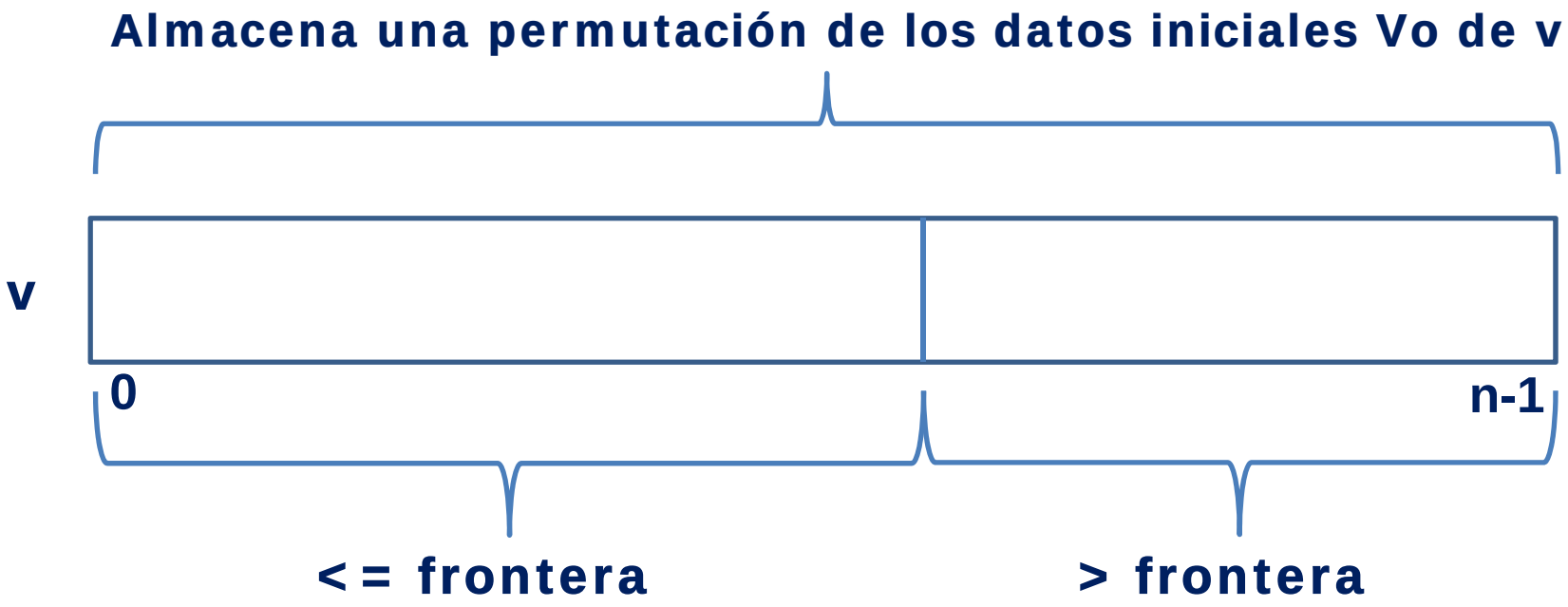
2. Algoritmo de distribución de los datos de un vector

Se va a diseñar un algoritmo de distribución de los datos de un vector:

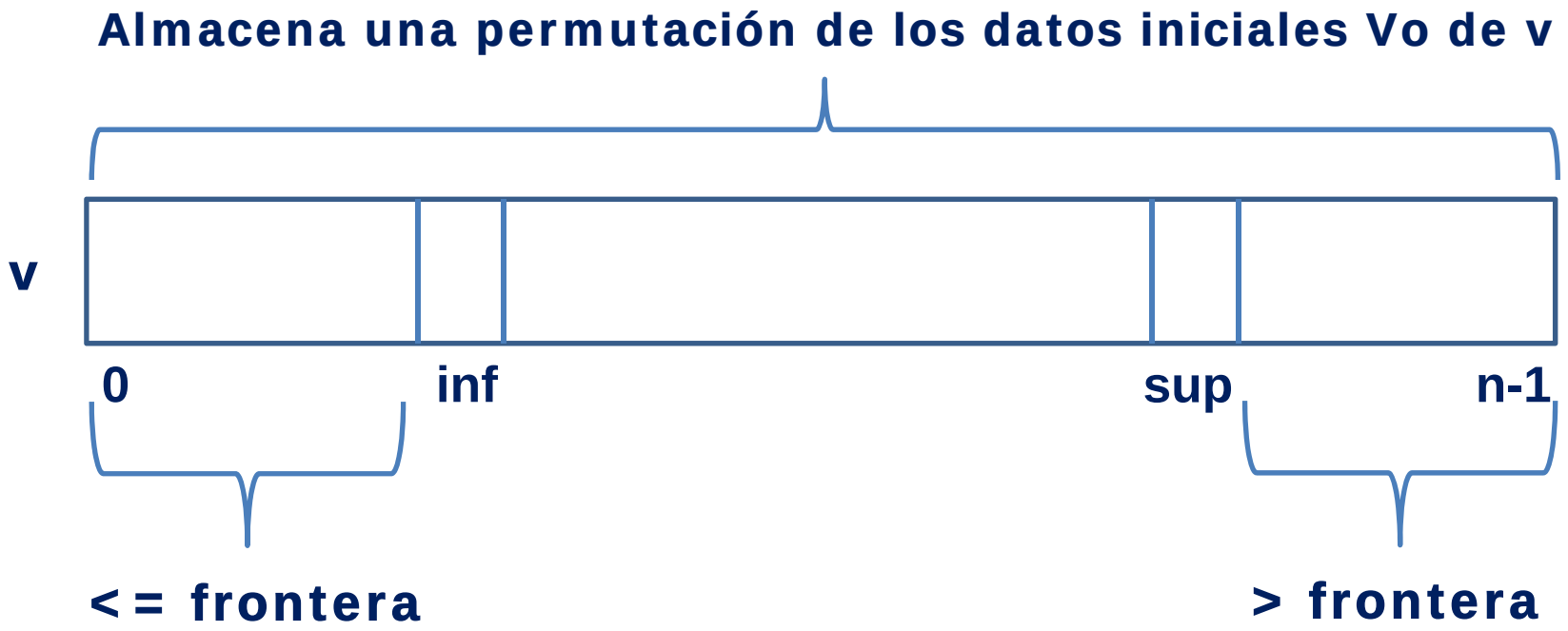
```
/*
 * Pre: v = Va  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Va,0,n-1) AND
 *       (PT alfa EN [0,n-1].
 *           (v[alfa] <= frontera ->
 *                (PT beta EN [0,alfa]. v[beta] <= frontera)) AND
 *           (v[alfa] > frontera ->
 *                (PT beta EN [alfa,n-1]. v[beta] > frontera)) )
 */
template <typename T>
void distribucion (T v[], const int n, const T frontera);
```

2. Algoritmo de distribución de los datos de un vector

Se va a diseñar un algoritmo de distribución de los datos de un vector:



Algoritmo de distribución de los datos de un vector:



```
/*
 * Pre: v = va  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,va,0,n-1) AND …
 */
template <typename T>
void distribucion (T v[], const int n, const T frontera) {
   int inf = 0, sup = n - 1;
   while (inf != sup + 1) {
       // Inv: n > 0 AND n <= #v AND esPermutación(v,va,0,n-1)
       //      AND inf > 0 AND inf <= sup + 1 AND sup <= n - 1
       //      AND (PT alfa [0,inf-1].v[alfa] <= frontera)
       //      AND (PT alfa [sup+1,n-1].v[alfa] > frontera)
       if (v[inf] <= frontera) { inf = inf + 1; }
       else if (v[sup] > frontera) { sup = sup - 1; }
       else {  // Permuta los elementos v[inf] y v[sup]
           T dato = v[inf]; v[inf] = v[sup]; v[sup] = dato;
           inf = inf + 1; sup = sup - 1;
       }
   }
}
```

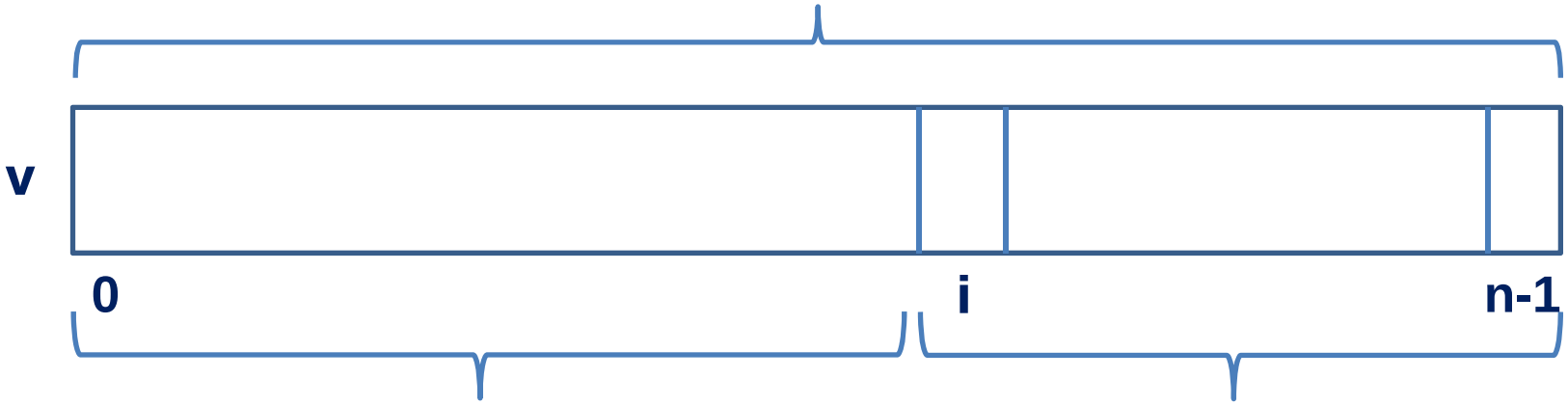
3. Algoritmos de ordenación interna de los datos de un vector

Se van a diseñar estos algoritmos de ordenación interna de vectores:

- Ordenación por selección directa
- Ordenación por inserción directa
- Ordenación por intercambio directo o método de la burbuja
- Ordenación rápida o ‘quicksort’

Ordenación por selección directa

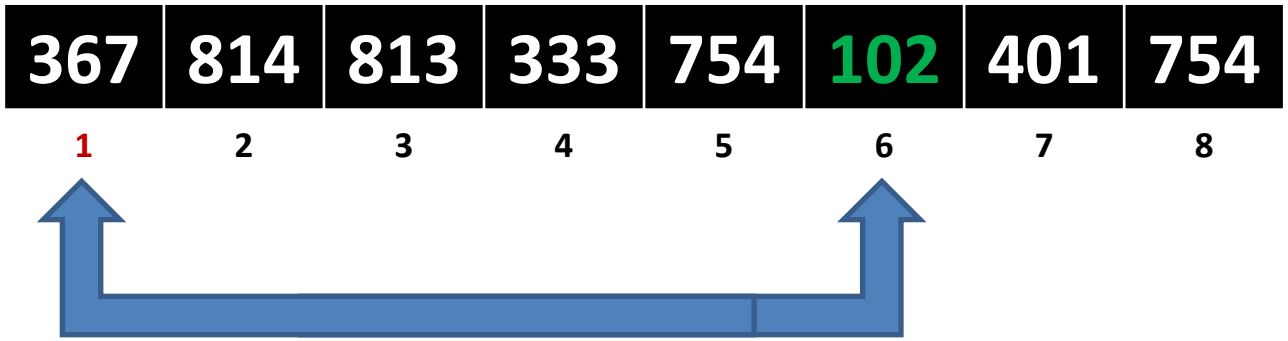
Almacena una permutación de los datos iniciales Vo de v

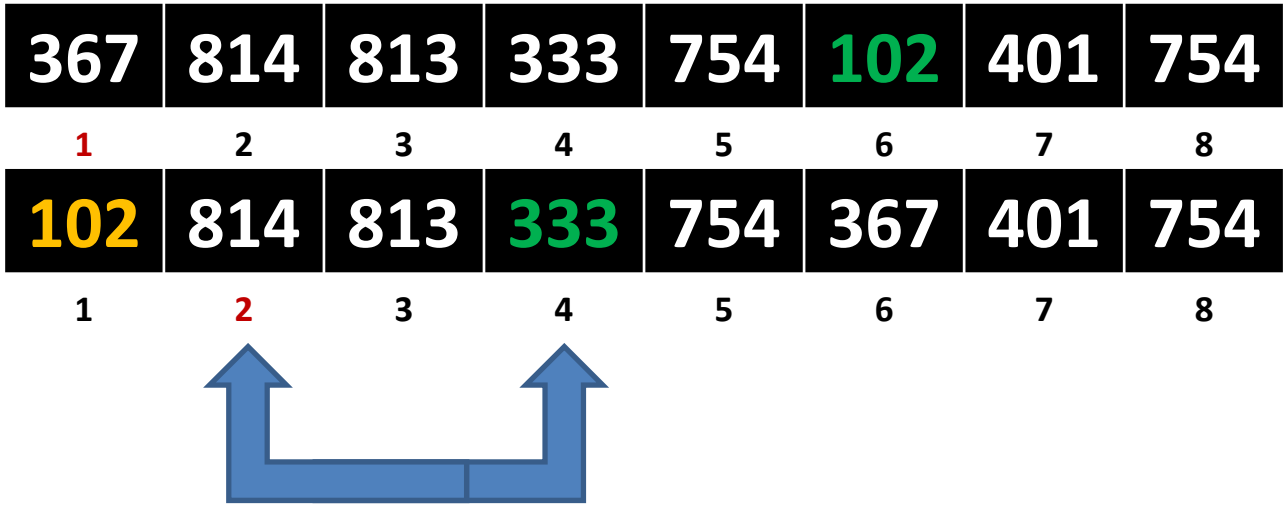


Aquí están los menores y están ordenados

Se busca la posición iMenor del menor, v[iMenor], y se permuta con v[i]

Algoritmo de ordenación interna de vectores por selección directa:




367 814 813 333 754 102 401 754
1 2 3 4 5 6 7 8
102 814 813 333 754 367 401 754
1 2 3 4 5 6 7 8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

102	814	813	333	754	367	401	754
1	2	3	4	5	6	7	8

102	333	813	814	754	367	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

102	814	813	333	754	367	401	754
1	2	3	4	5	6	7	8

102	333	813	814	754	367	401	754
1	2	3	4	5	6	7	8

102	333	367	814	754	813	401	754
1	2	3	4	5	6	7	8

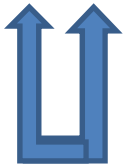
367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

102	814	813	333	754	367	401	754
1	2	3	4	5	6	7	8

102	333	813	814	754	367	401	754
1	2	3	4	5	6	7	8

102	333	367	814	754	813	401	754
1	2	3	4	5	6	7	8

102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8



367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
102	814	813	333	754	367	401	754
1	2	3	4	5	6	7	8
102	333	813	814	754	367	401	754
1	2	3	4	5	6	7	8
102	333	367	814	754	813	401	754
1	2	3	4	5	6	7	8
102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8
102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

102	814	813	333	754	367	401	754
1	2	3	4	5	6	7	8

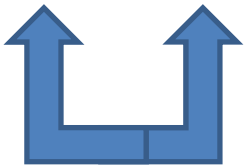
102	333	813	814	754	367	401	754
1	2	3	4	5	6	7	8

102	333	367	814	754	813	401	754
1	2	3	4	5	6	7	8

102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8

102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8

102	333	367	401	754	754	814	813
1	2	3	4	5	6	7	8



367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
102	814	813	333	754	367	401	754
1	2	3	4	5	6	7	8
102	333	813	814	754	367	401	754
1	2	3	4	5	6	7	8
102	333	367	814	754	813	401	754
1	2	3	4	5	6	7	8
102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8
102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8
102	333	367	401	754	754	814	813
1	2	3	4	5	6	7	8
102	333	367	401	754	754	813	814
1	2	3	4	5	6	7	8

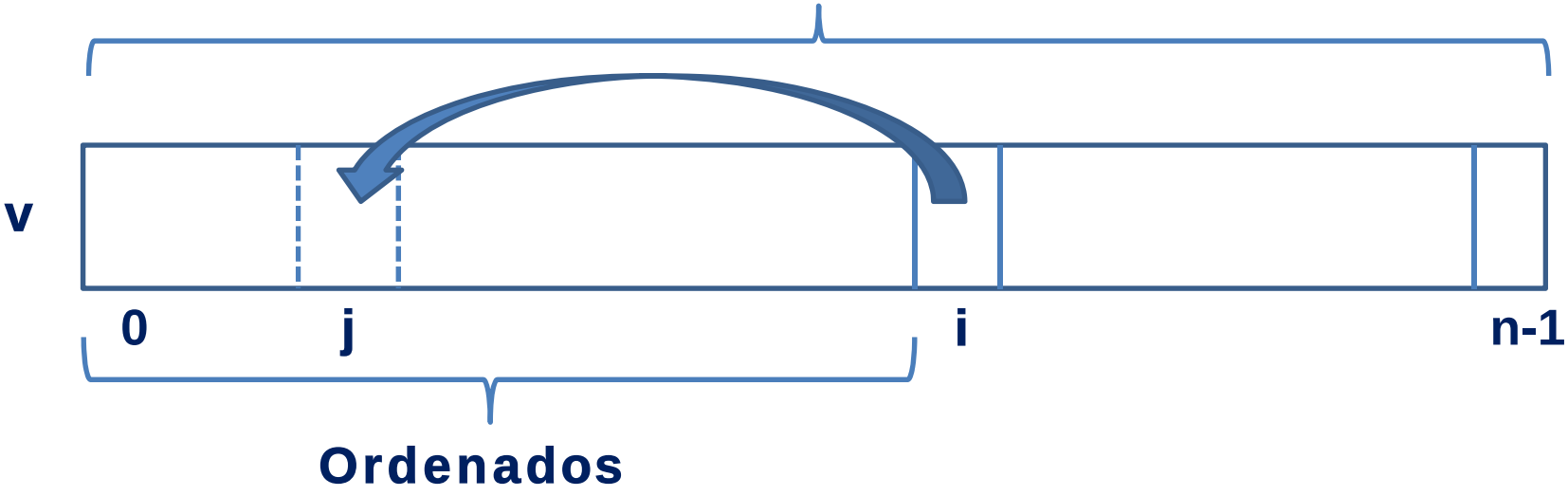
```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,v0,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void seleccion (T v[], const int n) {
   for (int i = 0; i != n - 1; ++i) {
      // Inv1: Seleccionados y reubicados los i datos menores de Vo[0,n-1]
      int iMenor = i, j = i;
      while (j != n - 1) {
         // Inv2: Recorrido de v[i,n-1] buscando el menor de sus elementos
         j = j + 1;
         if (v[j] < v[iMenor]) {
            iMenor = j;
         }
      }
      //  Permuta v[i] y v[iMenor]
      T dato = v[i];
      v[i] = v[iMenor];   v[iMenor] = dato;
   }
}
```

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,v0,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void seleccion (T v[], const int n) {
   for (int i = 0; i != n - 1; ++i) {
       // Inv1: n > 0 AND n <= #v AND esPermutación(v,Vo,0,n-1) AND ordenado(v,0,i-1)
       //       AND (PT alfa EN [0,i-1].(PT beta EN [i,n-1]. v[alfa] <= v[beta]) )
      int iMenor = i, j = i;
      while (j != n - 1) {
         // Inv2: Recorrido de v[i,n-1] buscando el menor de sus elementos
         j = j + 1;
         if (v[j] < v[iMenor]) {
            iMenor = j;
         }
      }
      // Permuta v[i] y v[iMenor]
      T dato = v[i];
      v[i] = v[iMenor];   v[iMenor] = dato;
   }
}
```

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,v0,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void seleccion (T v[], const int n) {
   for (int i = 0; i != n - 1; ++i) {
      // Inv1: n > 0 AND  n <= #v AND esPermutación(v,Vo,0,n-1) AND ordenado(v,0,i-1)
      //       AND (PT alfa EN [0,i-1].(PT beta EN [i,n-1].v[alfa] <= v[beta]) )
      int iMenor = i, j = i;
      while (j != n - 1) {
         // Inv2: n > 0 AND esPermutación(v,Vo,0,n-1) AND ordenado(v,0,i-1) AND
         //       (PT alfa EN [0,i-1].(PT beta EN [i,n-1].v[alfa] <= v[beta]) ) AND
         //       (PT alfa EN [i,j].v[iMenor] <= v[alfa])
         j = j + 1;
         if (v[j] < v[iMenor]) {
            iMenor = j;
         }
      }
      // Permuta v[i] y v[iMenor]
      T dato = v[i];
      v[i] = v[iMenor];   v[iMenor] = dato;
   }
}
```

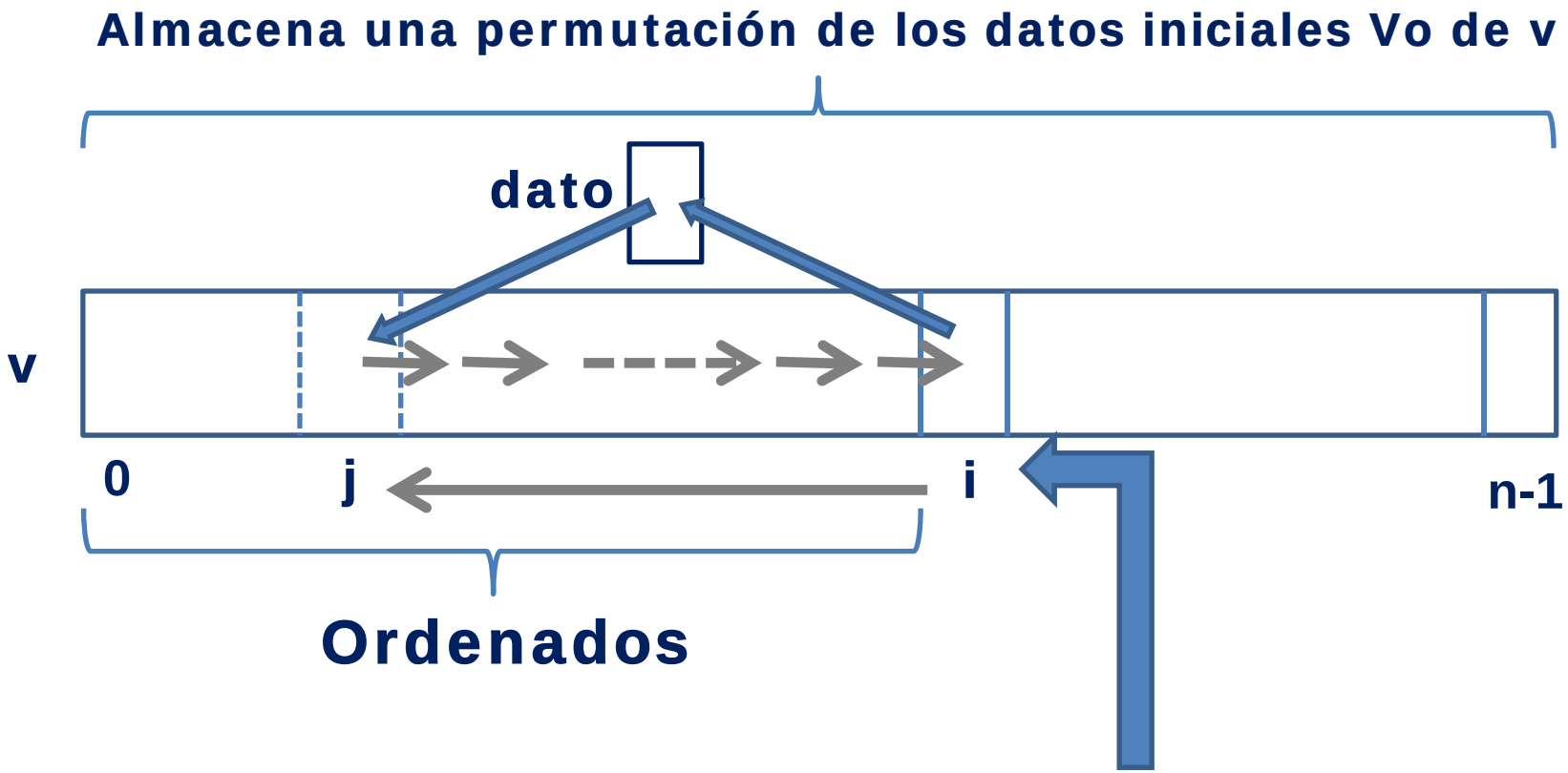
Ordenación por inserción directa

Almacena una permutación de los datos iniciales Vo de v



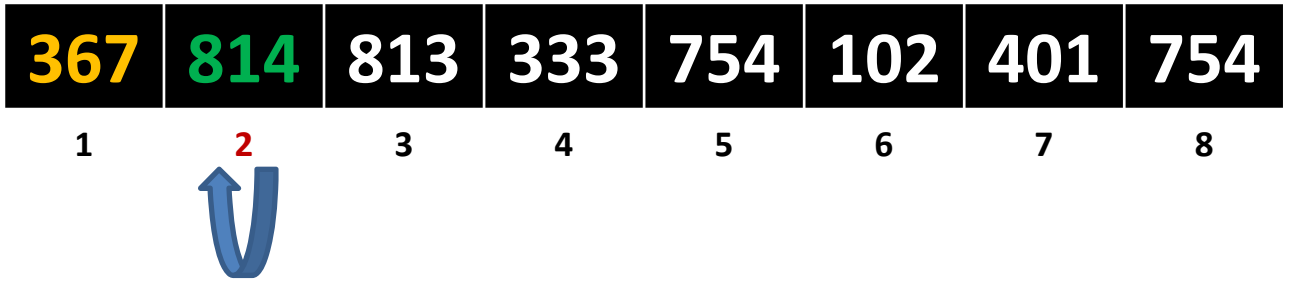
Debe insertar v[i] en la posición v[j] que le corresponda en v[0..i]

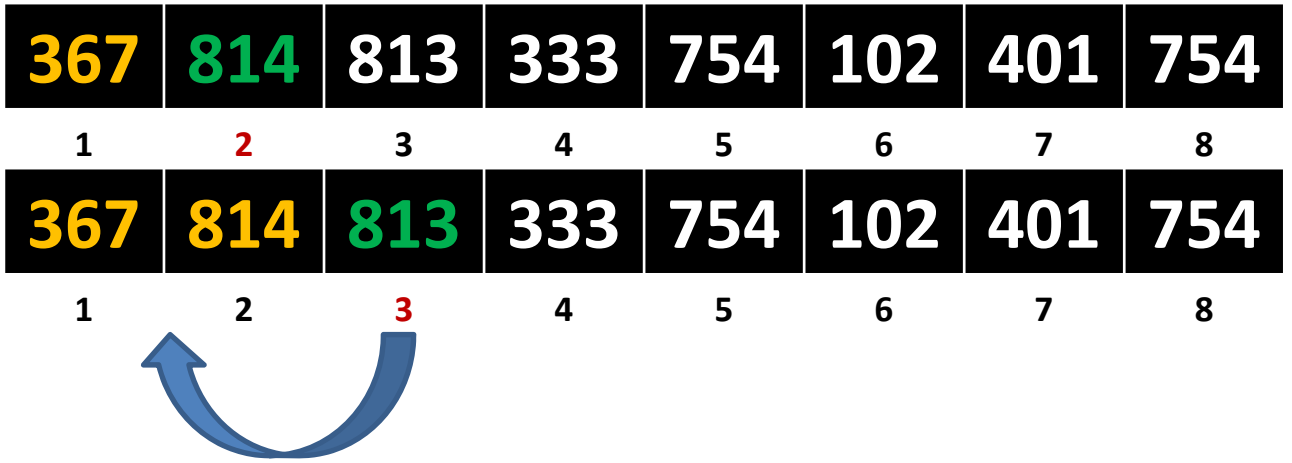
Ordenación por inserción directa



Inserta v[i] en la posición v[j] que le corresponda, previo desplazamiento de los elementos v[j..i-1]

Algoritmo de ordenación interna de vectores por inserción directa:




367 814 813 333 754 102 401 754
1 2 3 4 5 6 7 8
367 814 813 333 754 102 401 754
1 2 3 4 5 6 7 8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

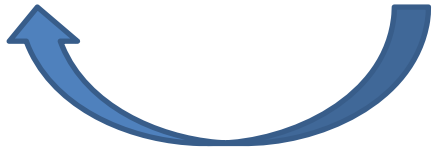
367	813	814	333	754	102	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	813	814	333	754	102	401	754
1	2	3	4	5	6	7	8

333	367	813	814	754	102	401	754
1	2	3	4	5	6	7	8



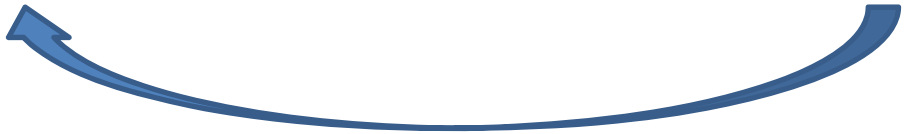
367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	813	814	333	754	102	401	754
1	2	3	4	5	6	7	8

333	367	813	814	754	102	401	754
1	2	3	4	5	6	7	8

333	367	754	813	814	102	401	754
1	2	3	4	5	6	7	8



367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
367	813	814	333	754	102	401	754
1	2	3	4	5	6	7	8
333	367	813	814	754	102	401	754
1	2	3	4	5	6	7	8
333	367	754	813	814	102	401	754
1	2	3	4	5	6	7	8
102	333	367	754	813	814	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8

367	813	814	333	754	102	401	754
1	2	3	4	5	6	7	8

333	367	813	814	754	102	401	754
1	2	3	4	5	6	7	8

333	367	754	813	814	102	401	754
1	2	3	4	5	6	7	8

102	333	367	754	813	814	401	754
1	2	3	4	5	6	7	8

102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8

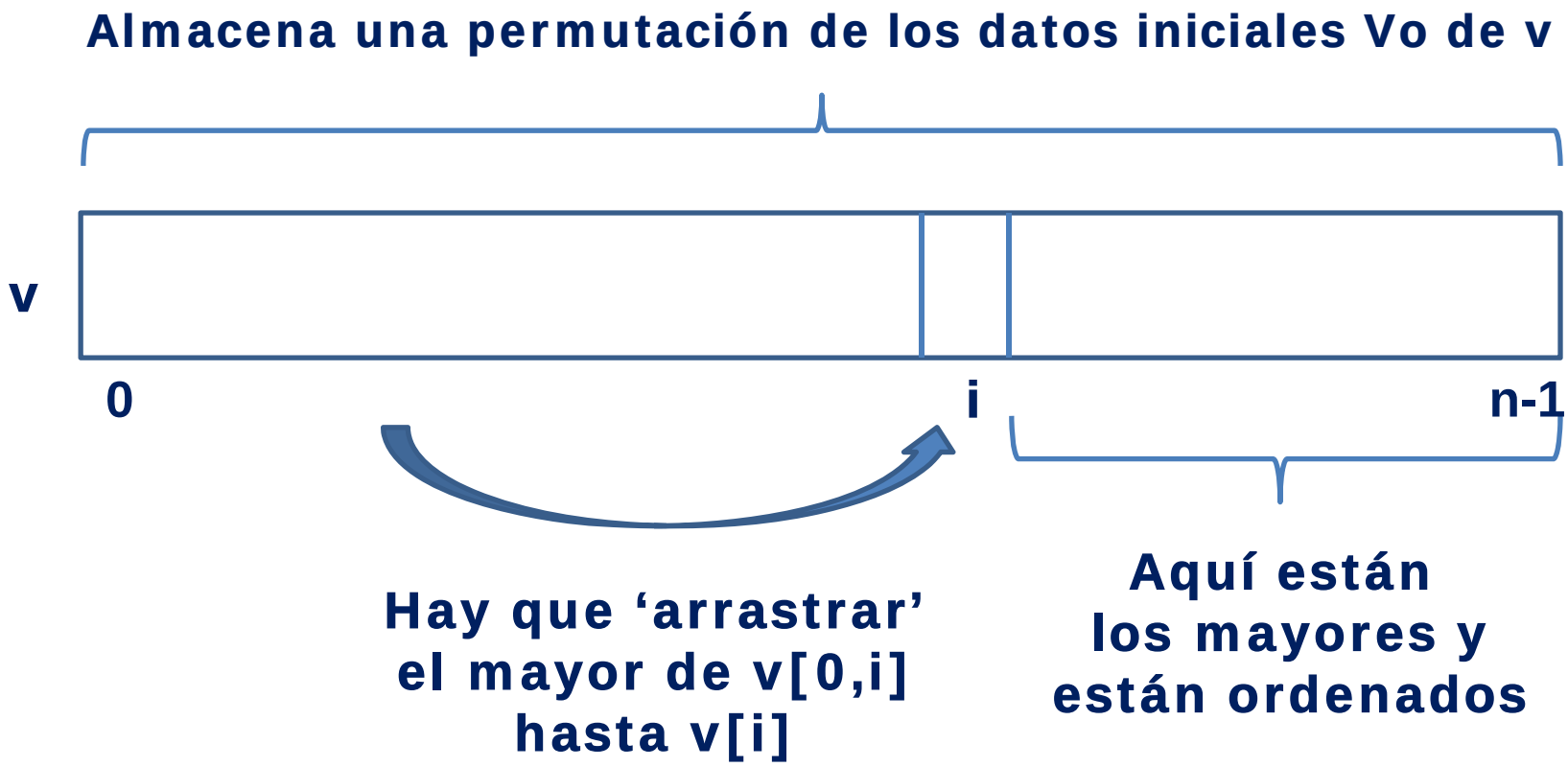
367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
367	813	814	333	754	102	401	754
1	2	3	4	5	6	7	8
333	367	813	814	754	102	401	754
1	2	3	4	5	6	7	8
333	367	754	813	814	102	401	754
1	2	3	4	5	6	7	8
102	333	367	754	813	814	401	754
1	2	3	4	5	6	7	8
102	333	367	401	754	813	814	754
1	2	3	4	5	6	7	8
102	333	367	401	754	754	813	814
1	2	3	4	5	6	7	8

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,v0,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void insercion (T v[], const int n) {
    for (int i = 1; i <= n - 1; ++i) {
        // Inv1: Insertados de forma ordenada los elementos de v[0,i-1]
        T dato = v[i];
        int j = i, iLimite = 0;
        while (j != iLimite) {
            // Inv2: Búsqueda del punto donde insertar el elemento dato
            if (v[j-1] <= dato) {
               iLimite = j;
            }
            else {
               v[j] = v[j-1];
               j = j - 1;
            }
        }
        v[j] = dato;
    }
}
```

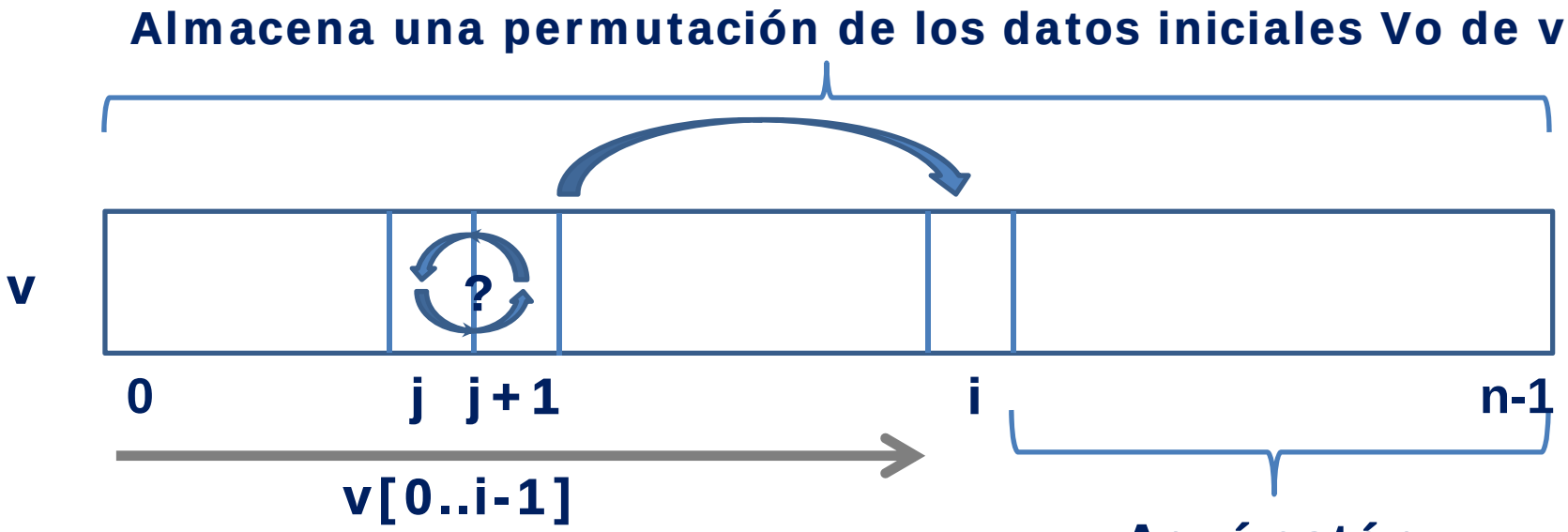
```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void insercion (T v[], const int n) {
    for (int i = 1; i <= n - 1; ++i) {
        // Inv1: n > 0 AND n <= #v AND esPermutación(v,Vo,0,n-1)
        //       AND ordenado(v,0,i-1)
        T dato = v[i];
        int j = i, iLimite = 0;
        while (j != iLimite) {
            // Inv2: Búsqueda del punto donde insertar el elemento dato
            if (v[j-1] <= dato) {
                iLimite = j;
            }
            else {
                v[j] = v[j-1];
                j = j - 1;
            }
        }
        v[j] = dato;
    }
}
```

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void insercion (T v[], const int n) {
    for (int i = 1; i <= n - 1; ++i) {
        // Inv1: n > 0 AND n <= #v AND esPermutación(v,Vo,0,n-1)
        //       AND ordenado(v,0,i-1)
        T dato = v[i];
        int j = i, iLimite = 0;
        while (j != iLimite) {
            // Inv2: n > 0 AND n <= #v AND esPermutación*(v,Vo,0,n-1) AND
            //       ordenado(v,0,j-1) AND ordenado(v,j,i) AND
            //       (PT alfa EN [0,j-1].
            //                   (PT beta EN [j+1,i].v[alfa] <= v[beta]) ) AND
            //       j <= i AND dato = Vo[i] AND
            //       (PT alfa EN [j+1,i].dato < v[alfa])
            if (v[j-1] <= dato) { iLimite = j; }
            else { v[j] = v[j-1]; j = j - 1; }
        }
        v[j] = dato;
    }
}
```

Ordenación por intercambio directo



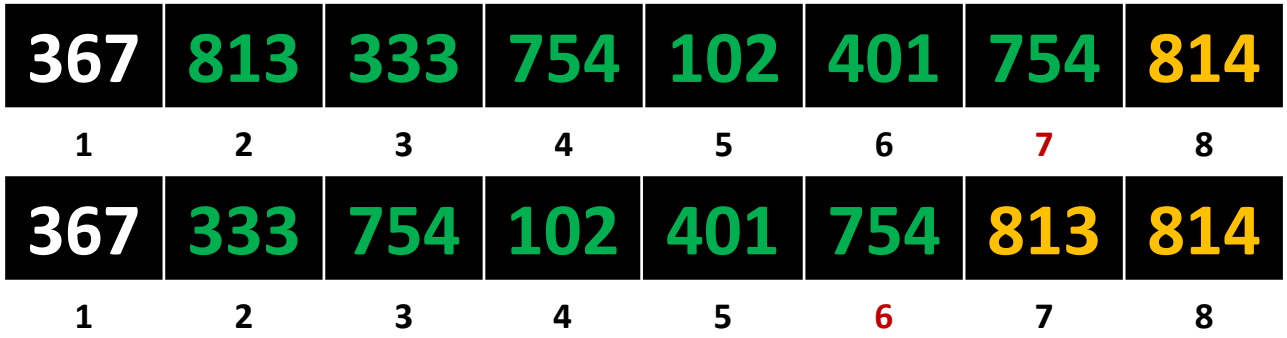
Ordenación por intercambio directo



Hay que arrastrar el mayor de v[0,i] hasta v[i], comparando pares consecutivos de v[j,j+1] y, si no están ordenados entre sí, se permutan (se intercambian)

Algoritmo de ordenación interna de vectores por intercambio directo (método de la burbuja):

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
367	813	333	754	102	401	754	814
1	2	3	4	5	6	7	8

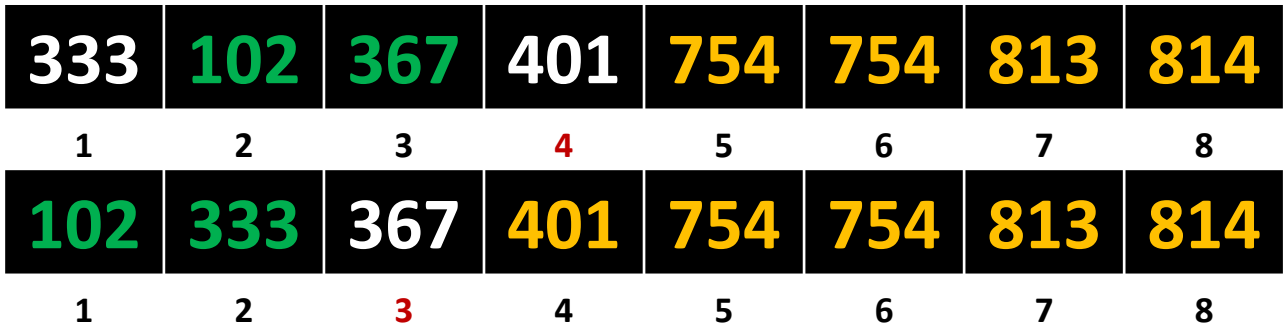

367 813 333 754 102 401 754 814
1 2 3 4 5 6 7 8
367 333 754 102 401 754 813 814
1 2 3 4 5 6 7 8

367	333	754	102	401	754	813	814
1	2	3	4	5	6	7	8

333	367	102	401	754	754	813	814
1	2	3	4	5	6	7	8

333	367	102	401	754	754	813	814
1	2	3	4	5	6	7	8

333	102	367	401	754	754	813	814
1	2	3	4	5	6	7	8


333 102 367 401 754 754 813 814
1 2 3 4 5 6 7 8
102 333 367 401 754 754 813 814
1 2 3 4 5 6 7 8

4 5

102	333	367	401	754	754	813	814
1	2	3	4	5	6	7	8

102	333	367	401	754	754	813	814
1	2	3	4	5	6	7 7	8


102 333 367 401 754 754 813 814
1 2 3 4 5 6 7 8
102 333 367 401 754 754 813 814
1 2 3 4 5 6 7 8

367	814	813	333	754	102	401	754
1	2	3	4	5	6	7	8
367	813	333	754	102	401	754	814
1	2	3	4	5	6	7	8
367	333	754	102	401	754	813	814
1	2	3	4	5	6	7	8
333	367	102	401	754	754	813	814
1	2	3	4	5	6	7	8
333	102	367	401	754	754	813	814
1	2	3	4	5	6	7	8
102	333	367	401	754	754	813	814
1	2	3	4	5	6	7	8
102	333	367	401	754	754	813	814
1	2	3	4	5	6	7	8
102	333	367	401	754	754	813	814
1	2	3	4	5	6	7	8

```
/*
 * Pre: v = v0  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,v0,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void intercambio (T v[], const int n) {
    for (int i = 0; i < n - 1; i++) {
        // Inv1: Ubicados en su posición definitiva los i mayores elementos
        for (int j = 0; j != n – 1 - i; j++) {
            // Inv2: Recorrido de v[0,n-1-j] permutando pares de elementos
            //       consecutivos en desorden
            if (v[j+1] < v[j]) {
                // Permuta los valores de v[j] y v[j+1]
                T dato = v[j];
                v[j] = v[j+1];
                v[j+1] = dato;
            }
        }
    }
}
```

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void intercambio (T v[], const int n) {
    for (int i = 0; i < n - 1; i++) {
        // Inv1: n > 0 AND n <= #v AND esPermutación(v,Vo,0,n-1) AND
        //       ordenado(v,n-1-i,n-1) AND
        //       (PT alfa EN [0,n-1-i].(PT beta EN [n-i,n-1]. v[alfa]<=v[beta]) )
        for (int j = 0; j != n – 1 - i; j++) {
            // Inv2: Recorrido de v[0,n-1-j] permutando pares de elementos
            //       consecutivos en desorden
            if (v[j+1] < v[j]) {
                // Permuta los valores de v[j] y v[j+1]
                T dato = v[j];
                v[j] = v[j+1];
                v[j+1] = dato;
            }
        }
    }
}
```

```
/*
 * Pre: v = v0  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,v0,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void intercambio (T v[], const int n) {
    for (int i = 0; i < n - 1; i++) {
        // Inv1: n > 0 AND n <= #v AND esPermutación(v,Vo,0,n-1) AND
        //       ordenado(v,n-1-i,n-1) AND
        //       (PT alfa EN [0,n-1-i].(PT beta EN [n-i,n-1]. v[alfa]<=v[beta]) )
        for (int j = 0; j != n – 1 - i; j++) {
            // Inv2: n > 0 AND n <= #v AND esPermutación(v,v0,0,n-1) AND
            //       ordenado(v,0,j-1) AND j < n – 1 - i AND
            //       (PT alfa EN [0,n-i].
            //                       (PT beta EN [n-i,n-1].v[alfa] <= v[beta]) ) AND
            //       (PT alfa EN [0,j]. v[alfa] <= v[j])
            if (v[j+1] < v[j]) {
                // Permuta los valores de v[j] y v[j+1]
                T dato = v[j];
                v[j] = v[j+1];
                v[j+1] = dato;
            }
        }
    }
}
```

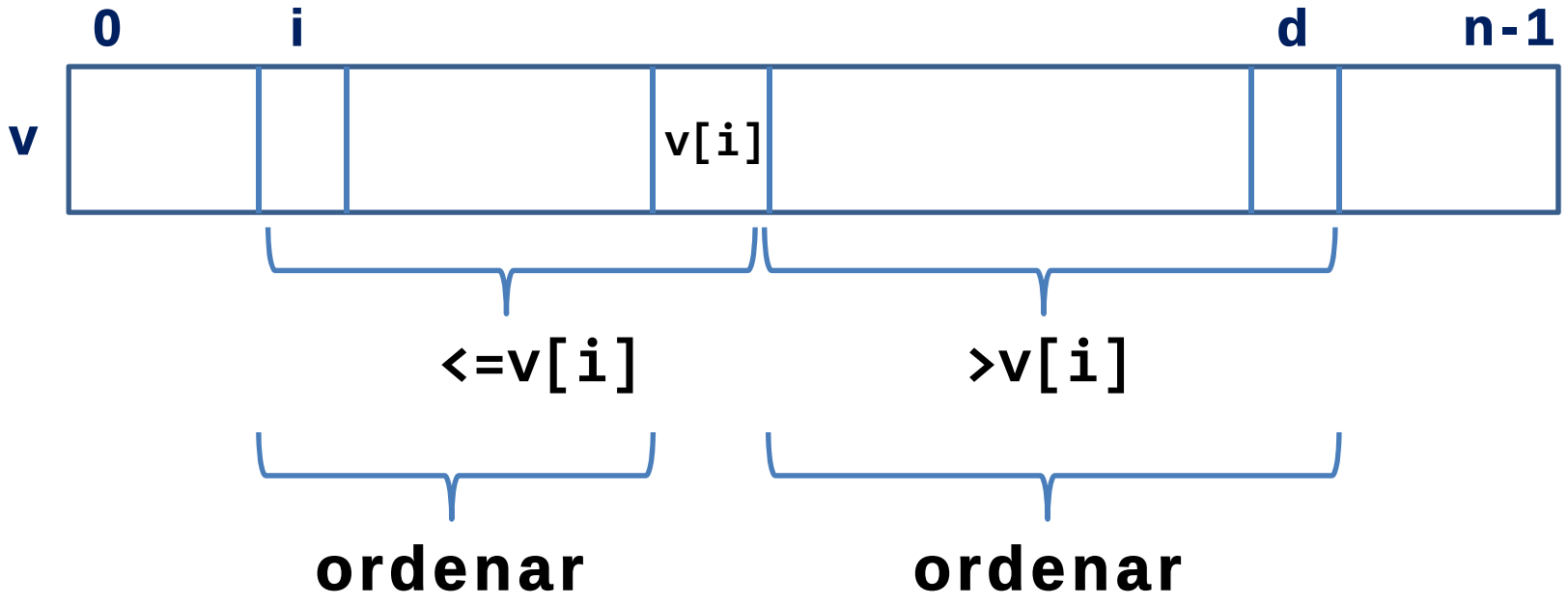
Algoritmo rápido de ordenación interna de vectores o ‘quicksort’:

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void quicksort (T v[], const int n) {
    // Ordenación del vector v[0,n-1] aplicando el método de ordenación
    // rápida de C.A.R.Hoare, también conocido como quicksort
    quicksort(v, 0, n-1);
}
```

```
/*
 * Pre: v = Vo  AND  n > 0  AND  n <= #v
 * Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
 */
template <typename T>
void quicksort (T v[], const int n) {
   quicksort(v, 0, n-1);
}

/*
 * Pre: v = Va  AND  desde >= 0  AND  hasta < #v
 * Post: esPermutación(v,Va,izda,dcha) AND ordenado(v,izda,dcha)
 */
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {

    … código de la función …

}
```

Algoritmo rápido de ordenación o quicksort

Distribuir según `v[i]`:



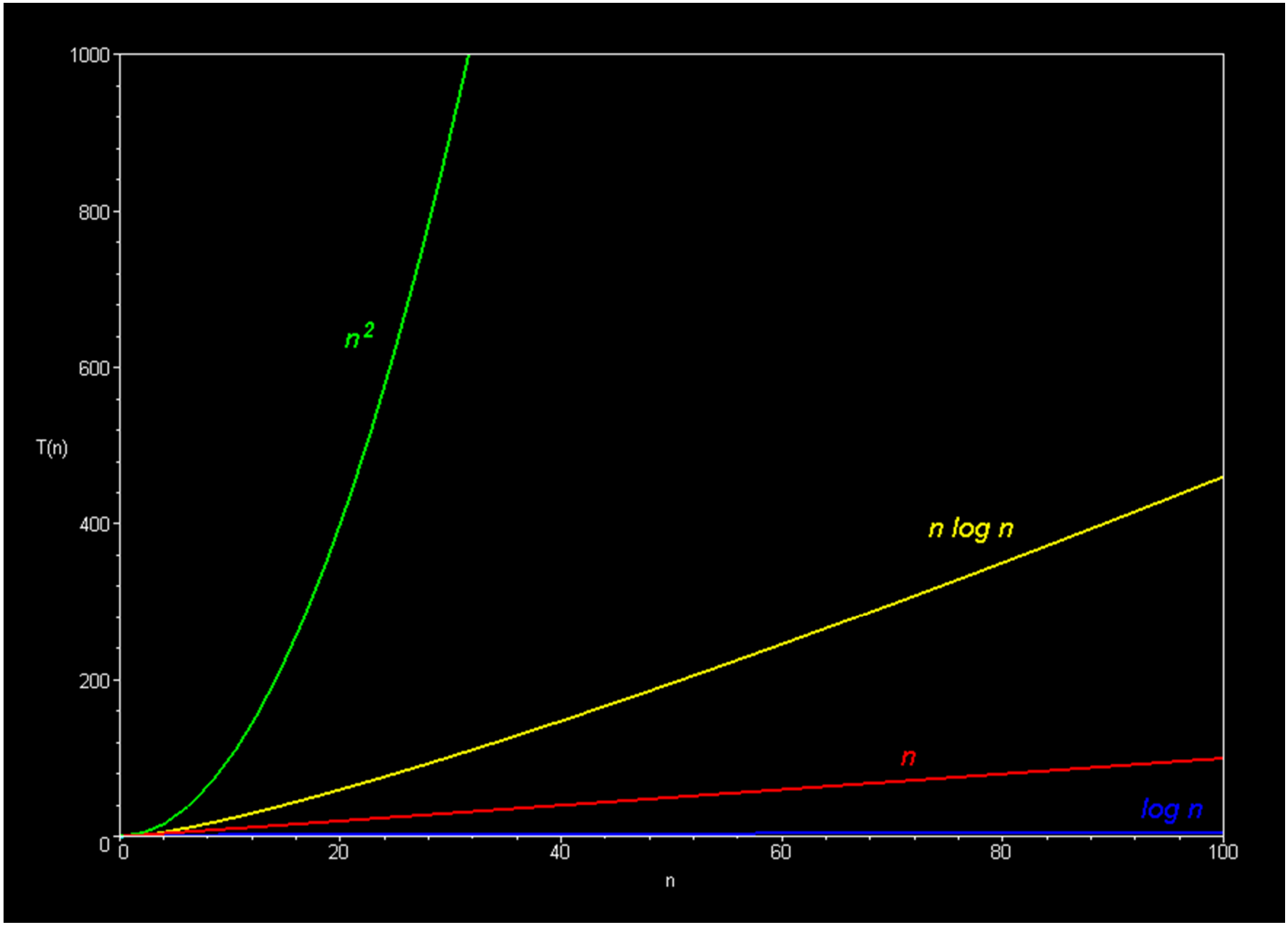
```
/*
 * Pre: v = Va AND  izda >= 0  AND  dcha < #v
 * Post: esPermutación(v,Va,izda,dcha) AND ordenado(v,izda,dcha)
 */
template <typename T>
void quicksort (T v[],  const int izda, const int dcha) {
    if (izda < dcha) {  // Son dos, como mínimo, los datos a ordenar
        T pivote = v[izda];     //  Selecciona el "pivote"
        int i = izda + 1, d = dcha;
        while (i != d + 1) {
            if (v[i] <= pivote) { i = i + 1; }
            else if (v[d] >= pivote) { d = d - 1; }
            else {
                T dato = v[i];
                v[i] = v[d]; v[d] = dato;
                i = i + 1; d = d - 1;
            }
        }
        // Sitúa el pivote en la frontera entre los datos cuyo valor
        // es menor o igual que él y los que son mayor o igual que él
        v[izda] = v[d]; v[d] = pivote;
        quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);
    }
}
```

```
// Pre: v = Va AND  izda >= 0  AND  dcha < #v
// Post: esPermutación(v,Va,izda,dcha) AND ordenado(v,izda,dcha)
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {
    if (izda < dcha) {
        T pivote = v[izda];     //  Selecciona el "pivote“
        // Distribuye los datos de v[izda+1,dcha] en función de su valor
        // respecto al “pivote”
        int i = izda + 1, d = dcha;
        while (i != d + 1) {
            if (v[i] <= pivote) { i = i + 1; }
            else if (v[d] >= pivote) { d = d - 1; }
            else {
                T dato = v[i];
                v[i] = v[d]; v[d] = dato;
                i = i + 1; d = d - 1;
            }
        }
        // Sitúa el “pivote” en la frontera entre los datos cuyo valor
        // es menor o igual que él y los de valor mayor o igual que él
        v[izda] = v[d]; v[d] = pivote;
        quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);
    }
}
```

4. Análisis comparativo de sus costes

Algoritmo de ordenación	Coste en el caso mejor	Coste en el caso peor
selección directa	$O(n^2)$	$O(n^2)$
inserción directa	$O(n)$	$O(n^2)$
intercambio directo	$O(n^2)$	$O(n^2)$
rápido o *quicksort*	$O(n \times \log n)$	$O(n^2)$

¿Qué algoritmo de los anteriores debo programar para ordenar un vector? ¿Por qué?


1000
800
600
400
200
0
T(n)
0
20
40
60
80
100
n
n 2
n log n
n
log n