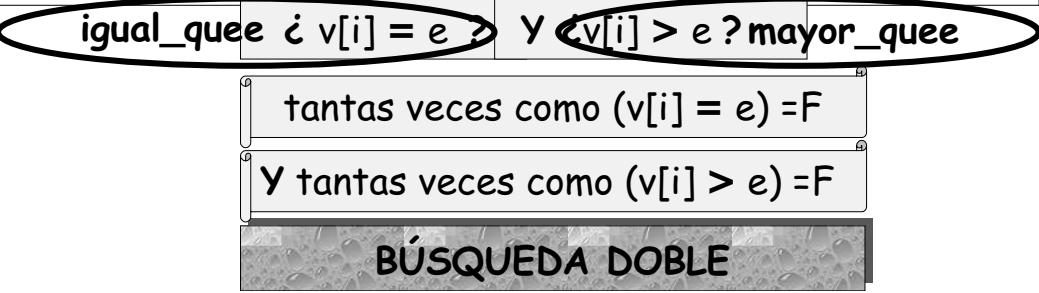


**Ejemplo:** sea  $v$  un vector con  $n$  componentes de tipo carácter ordenadas ascendentemente. Indicar si el carácter  $e$  está en  $v$

**algoritmo doble** (**DATOS**  $v$ : vector[1..Nmax] de carácter;  
 $n$ : entero;  $e$ : carácter;  
**RESULTADOS** está\_enasc: lógico)  
 $P = \{1 \leq n \leq Nmax \wedge (\forall i: 1 \leq i < n: v[i] \leq v[i+1])\}$   
 $Q = \{\text{está\_enasc} = (\exists i: 1 \leq i \leq n: (v[i] = e))\}$



**Ejemplo:** sea  $v$  un vector con  $n$  componentes de tipo carácter ordenadas ascendentemente. Indicar si el carácter  $e$  está en  $v$

**algoritmo doble** (**DATOS**  $v$ : vector[1..Nmax] de carácter;  
 $n$ : entero;  $e$ : carácter;  
**RESULTADOS** está\_enasc: lógico)  
 $P = \{1 \leq n \leq Nmax \wedge (\forall i: 1 \leq i < n: v[i] \leq v[i+1])\}$   
 $Q = \{\text{está\_enasc} = (\exists i: 1 \leq i \leq n: (v[i] = e \vee v[i] > e))\}$



**Ejemplo:** sea  $v$  un vector con  $n$  componentes de tipo carácter ordenadas ascendentemente. Indicar si el carácter  $e$  está en  $v$

**algoritmo** doble (**DATOS**  $v$ : vector[1..Nmax] de carácter;  
 $n$ : entero;  $e$ : carácter;  
**RESULTADOS**  $está\_enasc$ : lógico)

$$P = \{ 1 \leq n \leq Nmax \wedge (\forall i: 1 \leq i < n: v[i] \leq v[i+1]) \}$$

$$Q = \{ está\_enasc = (\exists i: 1 \leq i \leq n: (v[i] = e$$

**BÚSQUEDA DOBLE**

$$\wedge (\forall j: 1 \leq j \leq i: v[j] \leq e)$$

$$\wedge (\forall j: i < j \leq n: v[j] \geq e) \} \}$$

$$\neg (\exists j: 1 \leq j \leq i: v[j] > e) \text{ mayor\_quee}$$

$$Q = \{ está\_enasc = (\exists i: 1 \leq i \leq n: (v[i] = e \text{ =igual\_quee} \wedge \neg (\exists j: 1 \leq j \leq i: v[j] > e) \text{ mayor\_quee} \wedge (\forall j: i < j \leq n: v[j] \geq e) \} \} ) \}$$

**BÚSQUEDA DOBLE**

si igual mayor , está en pos

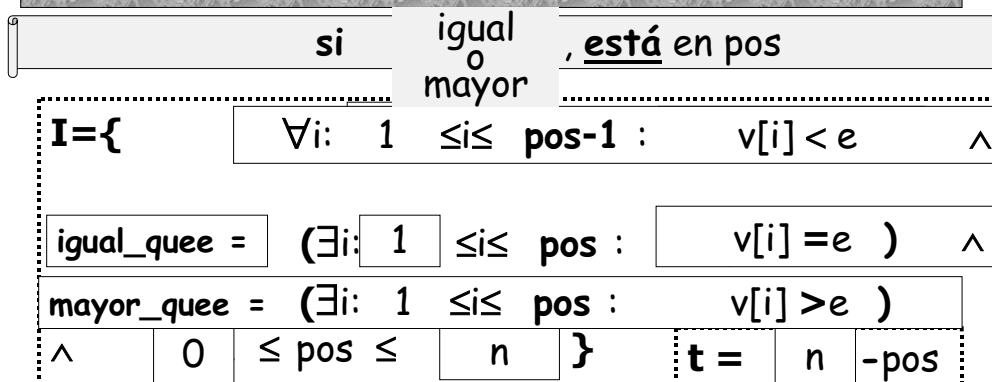
$I = \{ \forall i: 1 \leq i \leq pos-1: v[i] \neq e \wedge$

$igual\_quee = (\exists i: 1 \leq i \leq pos: v[i] = e) \wedge$

$\wedge 0 \leq pos \leq n \}$

$$Q = \{ \text{está\_enasc} = (\exists i: 1 \leq i \leq n: (v[i]=e \text{ =igual\_quee} \\ \neg \text{mayor\_quee} \Rightarrow \neg (\exists j: 1 \leq j \leq i: v[j] > e)) \\ \wedge (\forall j: i < j \leq n: v[j] \geq e)) ) \}$$

### BÚSQUEDA DOBLE



## Esquema búsqueda ascendente 1

```

pos := 0;
igual_quee := F;
mayor_quee := F;
(* está_enasc = igual_quee *)
mientras pos <> n  $\wedge$  [  $\neg$ (igual_quee)  $\wedge$   $\neg$ (mayor_quee) ] hacer
    pos := pos + 1;
    si v[pos] >= e entonces si v[pos] = e entonces igual_quee := V
                                sino mayor_quee := V fsi
fsi
fmientras
si igual_quee = F
    entonces está_enasc := F
    sino está_enasc := V; fsi
    } está_enasc := igual_quee;
  
```

## Esquema búsqueda ascendente 1

pos := 0;

mayor\_o\_igual\_quee := F;

(\* está\_enasc = igual\_quee \*)

**mientras** pos <> n  $\wedge$   $\neg$  (mayor\_o\_igual\_quee) **hacer**

pos := pos + 1;

**si** v[pos] >= e **entonces** mayor\_o\_igual\_quee := V

**fsi**

**fmientras**

**si** v[pos] < e **entonces** está\_enasc := F  
**sino** está\_enasc := V; **fsi**

## BÚSQUEDA DOBLE

si  $\frac{\text{igual}}{0}$  , está en pos  
 $\frac{\text{mayor}}{0}$

$I = \{ \forall i: 1 \leq i \leq \text{pos}-1: v[i] < e \} \wedge$   
 $\frac{\text{mayor\_o\_igual\_quee}}{\text{igual\_quee}} = \{ \exists i: 1 \leq i \leq \text{pos} : v[i] \geq e \} \wedge$   
 $0 \leq \text{pos} \leq n \}$

## 4. Combinación de técnicas de recorrido y búsqueda

**Ejemplo 1:** dado un vector de enteros  $v$ , diseñar un algoritmo que calcule la **suma** de los elementos del vector que aparecen **tras el primer número impar**

**Ejemplo 2:** dado un vector  $v$  de  $n$  elementos de **TipoBase**, ordenar **crecientemente** sus elementos

**Ejemplo 3:** dados  $v1$ :vector[1..Nmax]de carácter, con  $n$  elementos no repetidos, y  $v2$ :vector[1..Nmax]de carácter con  $m$  elementos,  $n \leq m$ , se quiere comprobar si  $v1$  es **subconjunto** de  $v2$

**Ejemplo:** dado un vector de enteros  $v$ , diseñar un algoritmo que calcule la **suma** de los elementos del vector que aparecen **tras el primer número impar**

**algoritmo** suma\_tras(**DATOS**  $v$ :t\_vector;  $n$ :entero;  
**RESULTADOS**  $s$ , tras:entero)

$P = \{1 \leq n \leq N_{max}\}$

$Q = \{(\exists i: 1 \leq i \leq n: \text{impar}(v[i]) \wedge 1 \leq \text{tras} \leq n \wedge s = \sum_{i: \text{tras} < i \leq n: v[i]})$

$\vee (\forall i: 1 \leq i \leq n: \text{par}(v[i]) \wedge \text{tras} = n + 1 \wedge s = 0 )\}$

**Ejemplo:** dado un vector de enteros  $v$ , diseñar un algoritmo que calcule la suma de los elementos del vector que aparecen tras el primer número impar

**algoritmo** suma\_tras(DATOS  $v:t\_vector$ ;  $n:entero$ ;  
**RESULTADOS**  $s, tras:entero$ )

$P = \{1 \leq n \leq N_{max}\}$

$Q = \{ (\forall i: 1 \leq i < tras: par(v[i]) \wedge 1 \leq tras \leq n \wedge s = \sum_{i: tras < i \leq n: v[i]} \wedge impar(v[tras]) \vee (\forall i: 1 \leq i \leq n: par(v[i]) \wedge tras = n + 1 \wedge s = \sum_{i: tras < i \leq n: v[i]}) \}$

**Ejemplo:** dado un vector de enteros  $v$ , diseñar un algoritmo que calcule la suma de los elementos del vector que aparecen tras el primer número impar

**algoritmo** suma\_tras(DATOS  $v:t\_vector$ ;  $n:entero$ ;  
**RESULTADOS**  $s, tras:entero$ )

$P = \{1 \leq n \leq N_{max}\}$

$Q = \{ (s = \sum_{i: tras < i \leq n: v[i]} \wedge \text{RECORRIDO desde } v[tras+1] \text{ hasta } v[n] \wedge (\forall i: 1 \leq i < tras: par(v[i]) \wedge impar(v[tras]) \wedge 1 \leq tras \leq n) \vee (\forall i: 1 \leq i \leq n: par(v[i]) \wedge tras = n + 1) ) \}$

**BÚSQUEDA**  $impar(v[i])$  tantas veces como  $(impar(v[i]))=F$

Combinado: Búsqueda + Recorrido

$$Q = \{ (s = \sum_{i: \text{tras} < i \leq n: v[i]) \wedge$$

$$( (\forall i: 1 \leq i < \text{tras}: \text{par}(v[i]) \wedge \text{impar}(v[\text{tras}]) \wedge 1 \leq \text{tras} \leq n)$$

$$\vee (\forall i: 1 \leq i \leq n: \text{par}(v[i]) \wedge \text{tras} = n+1) ) \}$$

**Combinado: Búsqueda + Recorrido**

IB= "si está, está en pos"

$$IB = \{ \forall i: 1 \leq i \leq \text{pos}-1 : v[i] \bmod 2 = 0 \wedge$$

$$\text{está} = ( \exists i: 1 \leq i \leq \text{pos} : v[i] \bmod 2 \neq 0 )$$

$$\wedge 0 \leq \text{pos} \leq n \}$$

**Combinado: Búsqueda + Recorrido**

```

pos := 0;
está := F;
vacía ; (* tras se establece en función de está *)
mientras pos <> n ^ ¬(está) hacer
    pos := pos + 1;
    si v[pos] mod 2 <> 0 entonces está := V ; fsi
fmientras
Efinal de la iteración (* tras=pos y se suma a partir de tras+1 *)
si está=V entonces tras:=pos; s:=0; posR := pos
sino
Einicial del RECORRIDO PR={ calcular s a partir de tras+1 }

```

### Combinado: Búsqueda + Recorrido

```

pos:= 0 ;
está:= F;
vacía ; (*tras se establece en función de está*)
mientras pos <> n ^ ¬(está) hacer
    pos := pos + 1;
    si v[pos] mod 2 <> 0 entonces está := V ; fsi
fmientras
    si está =F, no está y pos=n
    Final de la iteración (* tras=pos+1 =n+1 y s=0 *)
si está=V entonces tras:=pos; s:=0; posR := pos
    sino tras:= pos+1; s:=0 ; posR := pos; fsi
Inicial del RECORRIDO PR={ NO hay RECORRIDO }
    
```

### Combinado: Búsqueda + Recorrido

IB= "si está, está en pos"

```

s:=0;
si está=F entonces tras:=pos+1 sino tras:=pos; fsi
posR := pos ;
    
```

IR= "he sumado siempre hasta posR incluido"

```

IR={ s= Σi : pos+1 ≤ i ≤ posR: v[i]
    ^ pos ≤ posR ≤ n }
    
```

### Combinado: Búsqueda + Recorrido

```
pos:= 0 ;
está:= F;
vacía ; (*tras se establece en función de está*)
mientras pos <> n  $\wedge$   $\neg$ (está) hacer
    pos := pos + 1;
    si v[pos] mod 2 <> 0 entonces está := V ; fsi
fmientras
```

Final de la iteración de BÚSQUEDA

```
si está=F entonces tras:=pos+1 sino tras:=pos; fsi
```

Inicial del RECORRIDO PR={ calcular s a partir de tras+1 }

```
posR := pos ;
s:=0;
mientras posR <> n hacer
    posR := posR + 1 ;
    s := s + v[posR] ;
fmientras
```

**Ejemplo:** dados v1:vector[1..Nmax]de carácter, con n elementos no repetidos, y v2:vector[1..Nmax]de carácter con m elementos,  $n \leq m$ , se quiere comprobar si v1 es subconjunto de v2

**algoritmo** subconjunto(**DATOS** v1,v2:vector[1..Nmax]de carácter;  
n,m :entero;  
**RESULTADOS** es:lógico)

**P=** { $1 \leq n \leq m \leq N_{max} \wedge \neg$  duplicados(v1,1,n)}

**Q=** {es= (  $\forall i:1 \leq i \leq n$  : pertenece(v2,m,v1[i]))}

$\exists j:1 \leq j \leq m: v2[j]=v1[i]$

BÚSQUEDA en  
v2 de v1[i]

**Ejemplo:** dados  $v1$ :vector[1..Nmax]de carácter, con  $n$  elementos no repetidos, y  $v2$ :vector[1..Nmax]de carácter con  $m$  elementos,  $n \leq m$ , se quiere comprobar si  $v1$  es **subconjunto** de  $v2$

**algoritmo** subconjunto(**DATOS**  $v1, v2$ :vector[1..Nmax]de carácter;  
 $n, m$  :entero;  
**RESULTADOS** es:lógico)

**P**=  $\{1 \leq n \leq m \leq Nmax \wedge \neg \text{duplicados}(v1, 1, n)\}$

**BÚSQUEDA** en  $v1$  del elemento que **NO** pertenece a  $v2$

**Q**=  $\{es = \neg(\exists i: 1 \leq i \leq n: \neg \text{pertenece}(v2, m, v1[i]))\}$

$\exists j: 1 \leq j \leq m: v2[j] = v1[i]$

**BÚSQUEDA** en  $v2$  de  $v1[i]$

está

Combinado: Búsqueda en  $v2$  dentro de Búsqueda en  $v1$

**Q**=  $es = \neg(\exists i: 1 \leq i \leq n: \neg \text{pertenece}(v2, m, v1[i]))\}$

Combinado: Búsqueda en  $v2$  dentro de Búsqueda en  $v1$

IBv1 = "si está, está en pos"

IBv1 =  $\{\forall i: 1 \leq i \leq pos-1: \text{pertenece}(v2, m, v1[i]) \wedge$

$\text{está} = (\exists i: 1 \leq i \leq pos : \neg \text{pertenece}(v2, m, v1[i]) )$

$\wedge 0 \leq pos \leq n \}$

Esquema búsqueda ascendente 1	
pos:= 0 ;	
está:= F;	
vacía ;	
<b>mientras</b> pos <> n $\wedge$ $\neg$ (está) <b>hacer</b>	
pos := pos + 1;	
<b>si</b> $\neg$ (buscaV(v2, m, v1[pos])) <b>entonces</b> está := V ; <b>fsi</b>	
<b>fmientras</b>	
<b>si</b> está =F <b>entonces</b> es := V	} es := $\neg$ está ;
<b>sino</b> es := F ; <b>fsi</b>	
Efinal del PROBLEMA Q={es= ( $\forall i:1 \leq i \leq n$ : pertenece(v2,m,v1[i])) }	

Esquema búsqueda ascendente 1	
pos:= 0 ;	
es := V;	
vacía ;	
<b>mientras</b> pos <> n $\wedge$ es <b>hacer</b>	
pos := pos + 1;	
<b>si</b> $\neg$ (buscaV(v2, m, v1[pos])) <b>entonces</b> es := F; <b>fsi</b>	
<b>fmientras</b>	
vacía ;	
Efinal del PROBLEMA Q={es= ( $\forall i:1 \leq i \leq n$ : pertenece(v2,m,v1[i])) }	

**Ejemplo:** dado un vector  $v$  de  $n$  elementos de TipoBase, ordenar crecientemente, in situ, sus elementos

**ordenación directa**

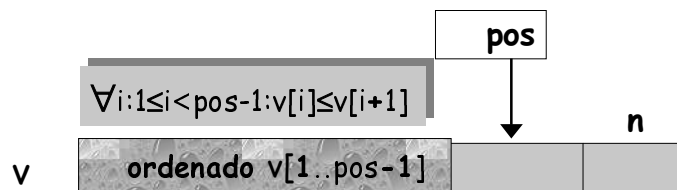
**algoritmo** ordenar(**DATOS**  $v$ :vector[1..Nmax]deTipoBase;  $n$ :entero;  
**RESULTADOS**  $v$ : vector[1..Nmax]de TipoBase)

$P = \{ 1 \leq n \leq Nmax \wedge \forall i: 1 \leq i \leq n: v[i] = V_i \}$

**RECORRIDO**

$Q = \{ \forall i: 1 \leq i < n: v[i] \leq v[i+1] \wedge \text{permutación}(v, V, 1, n) \}$

he ordenado siempre hasta pos excluido



**Ejemplo:** dado un vector  $v$  de  $n$  elementos de TipoBase, ordenar crecientemente, in situ, sus elementos

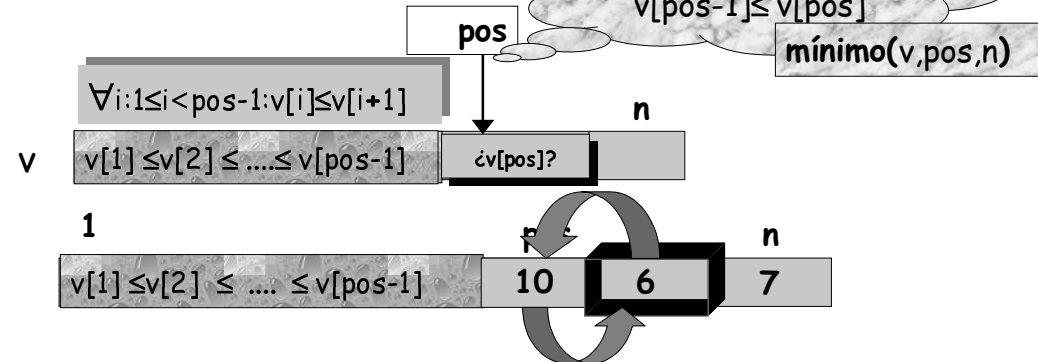
**algoritmo** ordenar(**DATOS**  $v$ :vector[1..Nmax]deTipoBase;  $n$ :entero;  
**RESULTADOS**  $v$ : vector[1..Nmax]de TipoBase)

$P = \{ 1 \leq n \leq Nmax \wedge \forall i: 1 \leq i \leq n: v[i] = V_i \}$

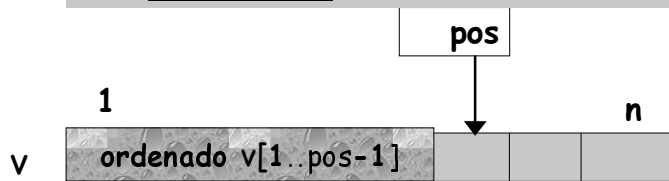
$Q = \{ \forall i: 1 \leq i < n: v[i] \leq v[i+1] \wedge \dots \}$

conseguir que  $v[pos-1] \leq v[pos]$

mínimo( $v, pos, n$ )



### Combinado: Recorrido dentro de Recorrido



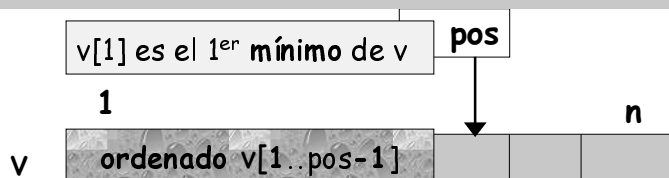
he ordenado siempre hasta  $pos-1$  **RECORRIDO desde  $v[1]$**

1.- $pos\_min := P_{\text{mínimo}}(v, pos, n)$  **RECORRIDO desde  $v[pos]$**

2.-intercambio( $v[pos], v[pos\_min]$ )

$$I = \{ \forall i: b_1 \leq i < pos: v[i] = \text{mínimo}(v, pos, n) \\ \wedge b_1 \leq pos \leq b_n + 1 \}$$

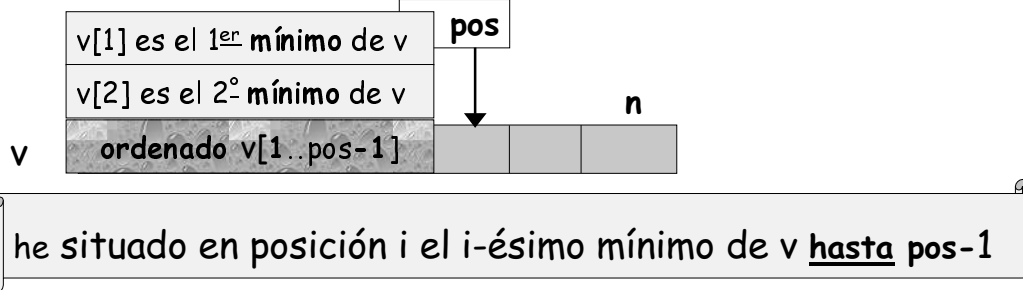
### Combinado: Recorrido dentro de Recorrido



he situado en posición  $i$  el  $i$ -ésimo mínimo de  $v$  hasta  $pos-1$

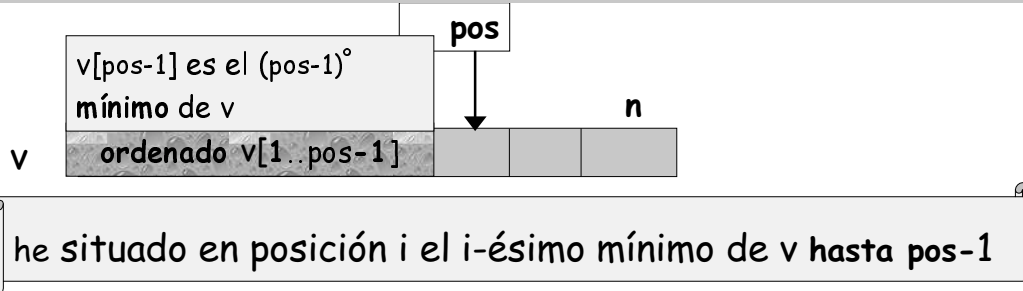
$$I = \{ \forall i: 1 \leq i < pos: v[i] = \text{mínimo}(v, pos, n) \\ v[1] \leq v[pos-1] \\ \forall j: pos \leq j \leq n: v[j] \geq v[1] \\ \wedge 1 \leq pos \leq n-1 \}$$

### Combinado: Recorrido dentro de Recorrido



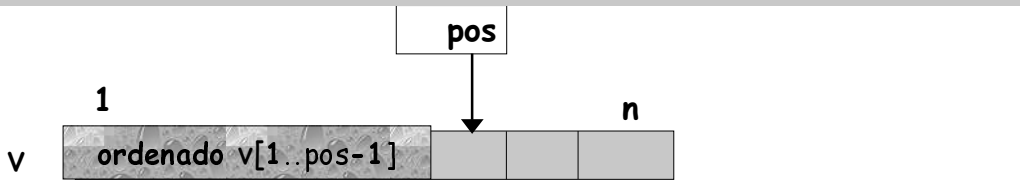
$$\mathbf{I} = \left\{ \forall i: \boxed{1} \leq i < \boxed{pos}: \begin{array}{l} v[i] = \text{mínimo}(v, pos, n) \\ v[1] \leq v[2] \leq v[pos-1] \\ \forall j: pos \leq j \leq n: v[j] \geq v[2] \end{array} \right. \\
 \wedge \boxed{1} \leq pos \leq \boxed{n-1} \}$$

### Combinado: Recorrido dentro de Recorrido



$$\mathbf{I} = \left\{ \forall i: \boxed{1} \leq i < \boxed{pos}: \begin{array}{l} v[i] = \text{mínimo}(v, pos, n) \\ v[1] \leq v[2] \leq \dots \leq v[pos-1] \\ \forall j: pos \leq j \leq n: v[j] \geq v[pos-1] \end{array} \right. \\
 \wedge \boxed{1} \leq pos \leq \boxed{n-1} \}$$

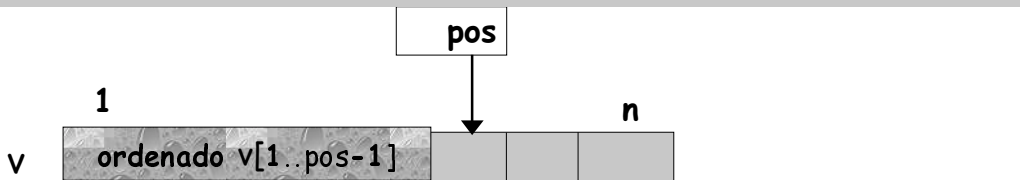
### Combinado: Recorrido dentro de Recorrido



he situado en posición  $i$  el  $i$ -ésimo mínimo de  $v$  hasta  $pos-1$

$$\mathbf{I} = \left\{ \begin{array}{l} \forall i: 1 \leq i < pos: v[i] = \text{mínimo}(v, pos, n) \\ \forall i: 1 \leq i < pos-1: v[i] \leq v[i+1] \\ \forall j: pos \leq j \leq n: v[j] \geq v[i] \end{array} \right. \\
 \wedge 1 \leq pos \leq n-1 \}$$

### Combinado: Recorrido dentro de Recorrido



he situado en posición  $i$  el  $i$ -ésimo mínimo de  $v$  hasta  $pos-1$

$$\mathbf{I} = \left\{ \begin{array}{l} \forall i: 1 \leq i < pos-1: v[i] \leq v[i+1] \\ \wedge \forall i: 1 \leq i \leq pos-1: ( \forall j: pos \leq j \leq n: v[j] \geq v[i] ) \end{array} \right. \\
 \wedge 1 \leq pos \leq n-1 \}$$

## Esquema recorrido ascendente 2

```

pos:= 1 ;
vacía;
t = n-1 -pos+1
mientras pos<> n hacer RECORRIDO desde v[1] hasta v[n-1]
    pos_min:=Pmínimo(v,pos,n); RECORRIDO desde v[pos] hasta v[n]
    intercambio(v[pos], v[pos_min]); t = n -Ppos+1
    pos := pos + 1;
fmientras
vacía ;

```

Combinado: Recorrido dentro de Recorrido

## Ejemplos propuestos

▪ Dado un vector de enteros  $v$ , diseñar un algoritmo que asigne a una variable  $z$  el número de ceros que hay en  $v$

Recorrido

▪ Dados 2 vectores,  $v_1$  de  $n$  elementos y  $v_2$  de  $m$  ( $n \leq m$ ), sin elementos repetidos, se pide obtener un tercer vector que sea la intersección de  $v_1$  y  $v_2$  (sin repetidos)

Combinado: Búsqueda dentro de Recorrido

▪ Sea  $v$  un vector con  $n$  componentes distintas entre sí y de tipo  $t_{\text{elemento}} = \text{tupla paridad:carácter; valor: carácter}$  ftupla;

Sabiendo que hasta una cierta posición de  $v$  todos sus componentes son pares y que tras ella todas son impares, indicar en una única iteración si el carácter  $e$  aparece en la zona par del vector

Búsqueda doble