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≤n≤Nmax $\wedge$ $(\forall i:1\leq i< n: v[i]\leq v[i+1])$ **}**

Q={está_enasc=**(**$\exists i: 1\leq i\leq n: (v[i]=e)$**)}**

igual_quee ¿ v[i] = e ? **Y** ¿v[i] > e ? **mayor_quee**

tantas veces como (v[i] = e) =F

Y tantas veces como (v[i] > e) =F

BÚSQUEDA DOBLE

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≤n≤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 $\vee$ v[i]>e) **)}**

igual_quee **mayor_quee**

BÚSQUEDA DOBLE

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≤n≤Nmax ∧ (∀i:1≤ i< n: v[i]≤ v[i+1]) **}**

Q={está_enasc = **(**∃i: 1≤i≤n: **(**v[i]=e

BÚSQUEDA DOBLE

∧ (∀j: 1≤j≤ **i**: v[j] ≤ e)

∧ (∀j: **i** <j≤n: v[j] ≥ e)) **)}**

¬(∃j: 1≤j≤ **i**: v[j] **>** e) **mayor_quee**

Q={está_enasc = **(**∃i: 1≤i≤n: (v[i]=e **=igual_quee**

¬mayor_quee = ¬(∃j:1 ≤j≤**i**:v[j]>e)

∧ (∀j: **i** <j≤n: v[j] ≥ e)) **)}**

BÚSQUEDA DOBLE

si igual o mayor , **está** en pos

I={ ∀i: 1 ≤i≤ **pos-1**: v[i] ≠ e) ∧

igual_quee = (∃i: 1 ≤i≤ **pos** : v[i] =e) ∧

∧ 0 ≤ pos ≤ n **}**

Q={está_enasc = **(**∃i: 1≤i≤n: (v[i]=e **=igual_quee**
¬mayor_quee = ¬(∃j:1 ≤j≤**i**:v[j]>e)
∧ (∀j: **i** <j≤n: v[j] ≥ e)) **)}**

BÚSQUEDA DOBLE

si igual o mayor **está** en pos

I={ ∀i: 1 ≤i≤ **pos-1** : v[i] < e ∧

igual_quee = **(**∃i: 1 ≤i≤ **pos** : v[i] =e **)** ∧

mayor_quee = **(**∃i: 1 ≤i≤ **pos** : v[i] >e **)**

∧ 0 ≤ pos ≤ n **}** **t =** n **-pos**

Esquema búsqueda ascendente 1

```
pos:= 0;
igual_quee:= F;
mayor_quee := F;
(* está_enasc = igual_quee *)
mientras pos <> n ∧ [¬(igual_quee) ∧ ¬(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 ∧ | ¬ (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 igual o mayor , **está** en pos

I={ ∀i: 1 ≤i≤ **pos-1**: v[i] < e) ∧

mayor_o_ igual_quee = (∃i: 1 ≤i≤ **pos** : v[i] ≥ e)

∧ 0 ≤ pos ≤ 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<=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= {$1 \leq n \leq$ **Nmax}**

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

$\vee$ **(**$\forall$i:$1 \leq i \leq n$: **par(**v[i]**)** $\wedge$ 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= {$1 \le n \le Nmax$**}**

Q= { $(\forall i: 1 \le i < tras: \mathbf{par}(v[i]) \wedge \mathbf{impar}(v[tras])) \wedge 1 \le tras \le n \wedge s = \sum i: tras < i \le n: v[i])$

$\vee$ $(\forall i: 1 \le i \le n: \mathbf{par}(v[i])) \wedge tras = n+1 \wedge s = \sum i: tras < i \le n: v[i])$ **}**

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= {$1 \le n \le Nmax$**}**

RECORRIDO desde v[tras+1] **hasta** v[n]

Q= {($s = \sum i: tras < i \le n: v[i]) \wedge$

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

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

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

Combinado: Búsqueda + Recorrido

Q= {(s=Σi:tras<i≤n:v[i]) ∧
(($\forall$i:1≤i<tras:**par(**v[i]) ∧ **impar(**v[tras]) ∧ 1≤tras≤n)
∨ ($\forall$i:1≤i≤n: **par(**v[i]) ∧ tras=n+1))}

Combinado: Búsqueda + Recorrido

IB= "**si** está, **está** en pos"

IB={∀i: 1 ≤i≤ **pos-1** : v[i] **mod** 2 = 0 ∧

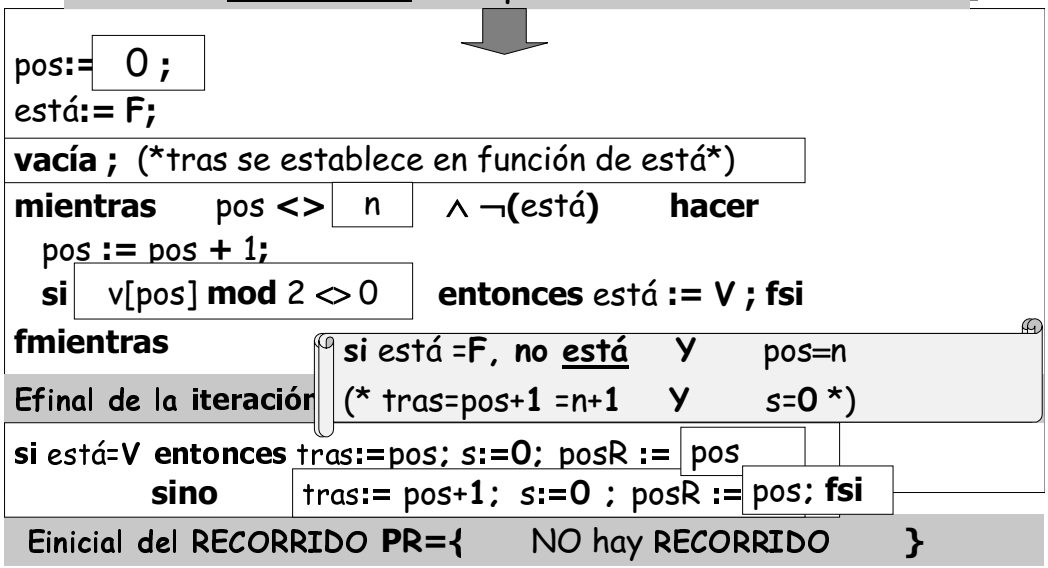
está=(∃i: 1 ≤i≤ **pos** : v[i] **mod** 2 ≠ 0 **)**

∧ 0 ≤ pos ≤ 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**
si está =**V**, **está** en pos, 1≤pos≤n
(* 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



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   ∧ ¬(está)      hacer
  pos := pos + 1;
  si  v[pos] mod 2 <> 0 entonces está := V ; fsi
fmientras
```

Efinal de la iteración de BÚSQUEDA

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

Einicial 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<=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≤n≤m≤**Nmax** ∧ ¬ **duplicados(**v1,1,n**)}**

Q= {es= **(** ∀i:1 ≤i ≤n : **pertenece(**v2,m,v1[i]**))}**

∃j:1≤j≤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<=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≤n≤m≤**Nmax** ∧ ¬ **duplicados(**v1,1,n**)}**

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

Q= {es= ¬(∃i:1≤j≤n: ¬ **pertenece(**v2,m,v1[i]**))}**

∃j:1≤j≤m: v2[j]=v1[i]

BÚSQUEDA en v2 de v1[i]

está

Combinado: Búsqueda en v2 dentro de Búsqueda en v1

Q= es= **(**¬**(**∃i:1≤j≤n: ¬ **pertenece(**v2,m,v1[i]**))))}**

Combinado: Búsqueda en v2 dentro de Búsqueda en v1

IBv1 = "**si** está, **está** en pos"

IBv1**={**∀i: 1 ≤i≤ **pos-1**: **pertenece(**v2,m,v1[i]**)** ∧

está=(∃i: 1 ≤i≤ **pos** : ¬**pertenece(**v2,m,v1[i]**)** **)**

∧ 0 ≤ pos ≤ n **}**

Esquema búsqueda ascendente 1

pos:= 0 ;
está:= F;
vacía ;

mientras pos <> n ∧ ¬(está) **hacer**
  pos := pos + 1;
  si ¬(buscaV(v2, m,v1[pos])) **entonces** está := V ; **fsi**
fmientras

si está =F **entonces** es := V
  sino es := F ; **fsi**
} es := ¬está ;

Efinal del PROBLEMA Q={es= (∀i:1≤i≤n :**pertenece**(v2,m,v1[i])) }

Esquema búsqueda ascendente 1

pos:= 0 ;
es := V;
vacía ;

mientras pos <> n ∧ es **hacer**
  pos := pos + 1;
  si ¬(buscaV(v2, m,v1[pos])) **entonces** es := F; **fsi**
fmientras

vacía ;

Efinal del PROBLEMA Q={es= (∀i:1≤i≤n :**pertenece**(v2,m,v1[i])) }

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

ordenación directa

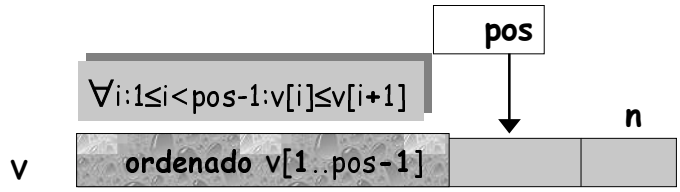
algoritmo ordenar(**DATOS** v:vector[1..**Nmax**]deTipoBase; n:entero;
RESULTADOS v: vector[1..**Nmax**]de TipoBase)

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

RECORRIDO

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

he **ordenado siempre hasta pos** excluido

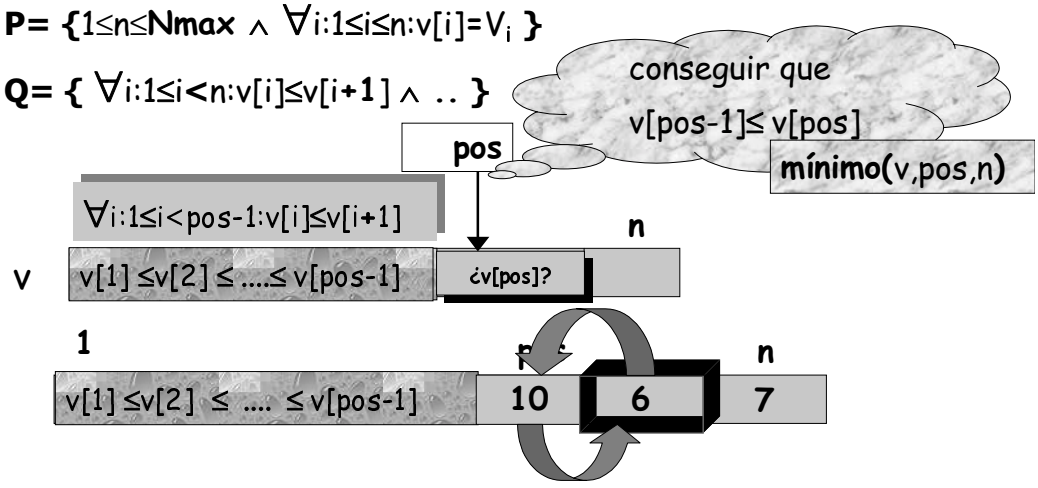


Ejemplo: dado un vector v de n elementos de TipoBase, ordenar crecientemente, in situ , sus elementos

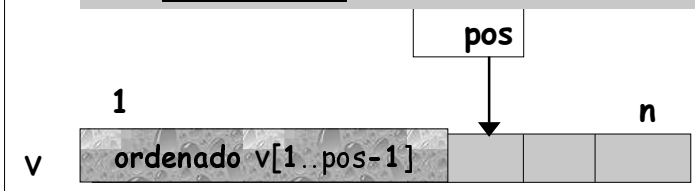
algoritmo ordenar(**DATOS** v:vector[1..**Nmax**]deTipoBase; n:entero;
RESULTADOS v: vector[1..**Nmax**]de TipoBase)

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

Q= { $\forall i: 1 \le i < n: v[i] \le v[i+1] \wedge$ .. **}**



Combinado: Recorrido dentro de Recorrido



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

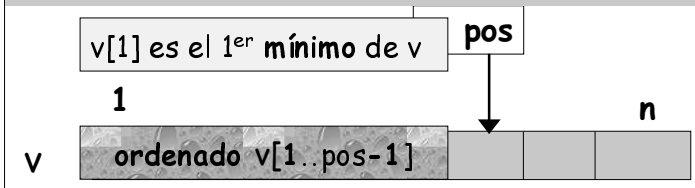
1.-pos_min:=Pmínimo(v, pos,n) | **RECORRIDO desde v[pos]**

2.-intercambio(v[pos],v[pos_min])

I= { ∀ i: $b_1 \leq i < pos$: v[i]= **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**

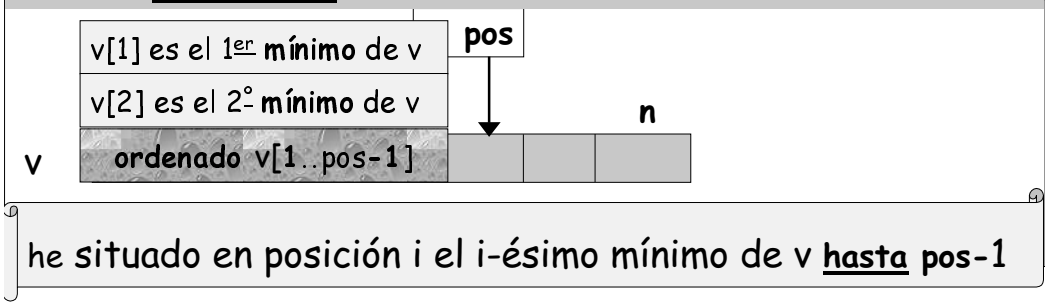
I= { ∀ i: 1 $\leq i < pos$: v[i]= **mínimo(**v,pos,n**)**

v[1] ≤ v[pos-1]

∀ j:pos≤j≤n: v[j] ≥ v[1]

$\wedge$ 1 $\leq pos \leq$ n-1 **}**

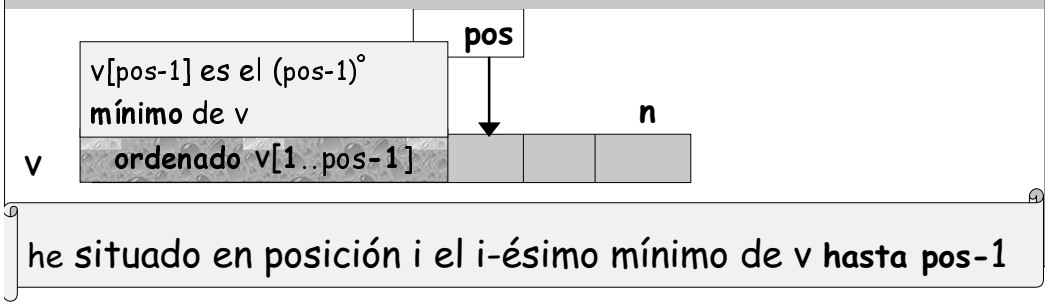
Combinado: Recorrido dentro de Recorrido



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

I= { ∀ i: 1 ≤ i < pos: v[i]= **mínimo**(v,pos,n)
v[1] ≤ v[2] ≤ v[pos-1]
∀j :pos≤j≤n: v[j] ≥ v[2]
∧ 1 ≤ pos ≤ n-1 **}**

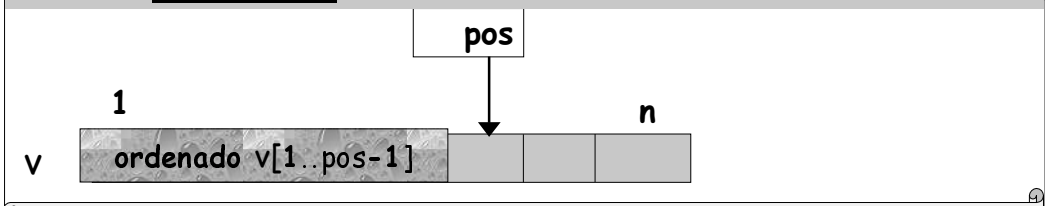
Combinado: Recorrido dentro de Recorrido



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

I= { ∀ i: 1 ≤ i < pos: v[i]= **mínimo**(v,pos,n)
v[1] ≤ v[2] ≤..≤ v[pos-1]
∀j :pos≤j≤n: v[j] ≥ v[pos-1]
∧ 1 ≤ pos ≤ n-1 **}**

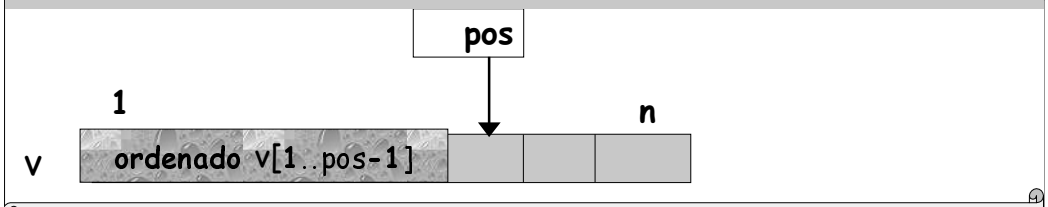
Combinado: Recorrido dentro de Recorrido



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

I= { ∀ i: 1 ≤ i < pos: v[i]= **mínimo**(v,pos,n)
∀i :1≤i<pos-1:v[i]≤v[i+1]
∀j :pos≤j≤n :v[j]≥v[i]
∧ 1 ≤ pos ≤ n-1 **}**

Combinado: Recorrido dentro de Recorrido



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

I= { ∀i :1≤i<pos-1: v[i]≤v[i+1]
∧ ∀i :1≤i ≤pos-1: **(** ∀j: pos≤j≤n : v[j]≥v[i]) **)**
∧ 1 ≤ pos ≤ 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, v1 de n elementos y v2 de m (n≤m), sin elementos repetidos, se pide obtener un tercer vector que sea la intersección de v1 y v2 (sin repetidos)

Combinado: Búsqueda dentro de Recorrido

- Sea v un vector con n componentes distintas entre sí y de tipo t_elemento=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