



# INFORMATICA I

## *Ordenamiento de un vector por el metodo burbuja*

*Profesor: Ing.Jerónimo Atencio*

*Auxiliares: Ing.Alexander Jiricny*

*Ing.Juan Carlos Cuttitta*

*Universidad Tecnológica Nacional  
Facultad Regional Buenos Aires  
Departamento de Ingeniería Electrónica*

18 de mayo de 2016

# Ordenamiento de un vector por el metodo *burbuja*

MEMORIA en  
ARQUITECTURA x86 32-bits

0x7FFE6E03					0x7FFE6E00
0x7FFE6E07					
0x7FFE6E0B					
0x7FFE6E0F					
0x7FFE6E13					
0x7FFE6E17					
0x7FFE6E1B					
0x7FFE6E1F					
0x7FFE6E23					
0x7FFE6E27					
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF					0xFFFFFFFFC

```
1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }
```

*contienen cualquier valor inicialmente*

0xFFFFFFFF

0xFFFFFC

2/91

# MEMORIA en ARQUITECTURA x86 32-bits

0xXX	0xXX	0xXX	0xXX	aux
0xXX	0xXX	0xXX	0xXX	resto
0xXX	0xXX	0xXX	0xXX	i
0xXX	0xXX	0xXX	0xXX	flag
0x00	0x00	0x00	0x02	vInt[0]
0x00	0x00	0x01	0x1E	vInt[1]
0x00	0x00	0x00	0x09	vInt[2]
0x00	0x00	0x00	0x3B	vInt[3]
0x00	0x00	0x03	0xA0	vInt[4]
0x00	0x00	0x00	0x03	vInt[5]
⋮	⋮	⋮	⋮	
0x00	0x00	0x00	0x00	

contienen  
valores  
iniciales

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

0xFFFFFFFF

0xFFFFF0C

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x00	← resto
0x7FFE6E0B	0xXX	0xXX	0xXX	0xXX	i
0x7FFE6E0F	0xXX	0xXX	0xXX	0xXX	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0xXX	0xXX	0xXX	0xXX	i
0x7FFE6E0F	0xXX	0xXX	0xXX	0xXX	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0xXX	0xXX	0xXX	0xXX	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the state of variables and the array vInt. A red dashed arrow points from the 'i' variable (0x7FFE6E0B) to the 'for' loop condition in the code, indicating the current iteration index. A red bracket above the 'for' loop condition highlights the calculation  $6-1=5$ , representing the number of elements to compare in the current pass.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from `vInt[0]` to line 10.
- Red dashed line from `vInt[1]` to line 12.
- Red dashed line from `vInt[i]` to line 14.
- Red dashed line from `vInt[i+1]` to line 14.
- Red bracket above line 12:  $6-1=5$ .
- Red bracket below line 14:  $0$  and  $1$ .

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	← i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition `i < (CANT-resto)` in the code. A red bracket above the loop condition indicates the calculation `6-1=5`, showing that the loop runs from `i=0` to `i=5`.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from `vInt[1]` to `vInt[2]` (lines 11 and 13).
- Red dashed line from `vInt[2]` to `vInt[3]` (lines 13 and 15).
- Red bracket above line 12:  $6-1=5$ .
- Red bracket above line 14:  $1$ .
- Red bracket above line 15:  $2$ .

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	- vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Handwritten annotations in the code:

- A red bracket above line 12 indicates  $6-1=5$ .
- A blue '1' is written above line 15.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

6-1=5

vInt[1] = vInt[2]

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	← resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	← i
0x7FFE6E0F	0x00	0x00	0x00	0x00	← flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	← vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-1=5

vInt[2] = aux

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating memory access and comparison in the selection sort algorithm. A red dashed arrow points from the `flag` variable to the `vInt[0]` memory location. A red bracket above the `for` loop indicates the calculation  $6-1=5$ , representing the number of comparisons for the first pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	← i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the `for` loop indicates the calculation `6-1=5`, representing the number of elements to compare in the current pass.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	<-vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	<-vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from line 12 to line 14:  $6-1=5$
- Red dashed line from line 14 to line 16: 2
- Red dashed line from line 16 to line 18: 3

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Handwritten annotations in the code:

- A red bracket above the `for` loop condition `(CANT-resto)` with the text `6-1=5`.
- A blue number `2` is written next to the `if` statement.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

6-1=5

vInt[2] = vInt[3]

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-1=5

vInt[3] = aux

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `flag` variable in the memory table to the `while(flag)` loop in the code. A red bracket above the `for` loop indicates the calculation  $6-1=5$ , representing the number of comparisons for the first pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	← i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-1=5`, showing that the loop iterates from `i=0` to `i=5`.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram annotations:

- Red bracket above line 12:  $6-1=5$
- Red dashed arrow from line 15 to line 16:  $3$
- Red dashed arrow from line 17 to line 18:  $4$

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-1=5`, showing that the loop iterates from `i=0` to `i=5`.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	← vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	← vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the execution of the selection sort algorithm on the array `vInt`. The array contains the values {2, 286, 9, 59, 928, 3}. The code uses a `do-while` loop to repeatedly find the minimum element (starting from index `i`) and swap it with the element at index `i+1`. The variable `resto` tracks the current position of the minimum element found in the current pass. The condition `i < (CANT-resto)` ensures that the inner loop only processes the unsorted portion of the array. The diagram highlights the comparison between `vInt[4]` (928) and `vInt[5]` (3), where `vInt[5]` is the minimum and will be swapped with `vInt[4]`. A red bracket indicates the calculation `6-1=5`, representing the number of elements in the current unsorted subarray.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Annotations in the code:

- Line 12:  $6-1=5$  (red bracket over  $(CANT-resto)$ )
- Line 15:  $4$  (blue number over the opening curly brace of the inner loop)

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

6-1=5

vInt[4] = vInt[5]

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	← resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	← i
0x7FFE6E0F	0x00	0x00	0x00	0x01	← flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	← vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-1=5

vInt[5] = aux

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `flag` variable in the memory table to the `while(flag)` loop in the code. A red bracket above the `for` loop indicates the calculation  $6-1=5$ , representing the number of comparisons for the first pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x05	$\leftarrow i$
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

*Note: A red dashed arrow points from the expression  $(CANT - resto)$  in the for loop to the value 5 in the memory table row 0x7FFE6E0B, with the calculation  $6 - 1 = 5$  shown above it.*

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x05	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `resto` variable to the `for` loop condition `(CANT-resto)`. A red bracket above the expression `6-1=5` indicates the calculation of the loop bound.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x05	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7      resto=0;
8      do
9      {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables: aux (0x00), resto (0x00), i (0x00), flag (0x00), and an array vInt of size 6. The array vInt contains the values {2, 286, 9, 59, 928, 3}. The execution of the algorithm is shown in the code block, which implements a selection sort. The code starts by including the standard input/output library and defining the constant CANT as 6. The main function then declares the variables aux, resto, i, flag, and vInt. The array vInt is initialized with the values {2, 286, 9, 59, 928, 3}. The algorithm enters a do-while loop that continues until the flag is 0. Inside the loop, the variable resto is incremented, and the flag is reset to 0. A for loop then iterates over the array vInt, comparing adjacent elements and swapping them if they are in the wrong order. The number of comparisons is reduced by the value of resto, as indicated by the annotation  $6-1=5$ . The flag is set to 1 if a swap occurs, and the loop continues until no swaps are needed.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the state of variables and the array vInt. A red dashed arrow points from the variable 'i' (at address 0x7FFE6E0B) to the 'for' loop condition in the code, indicating the current iteration index. A red bracket above the 'for' loop condition indicates the calculation  $6 - 2 = 4$ , representing the number of elements to compare in the current pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	< vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from `vInt[0]` to line 10.
- Red dashed line from `vInt[1]` to line 12.
- Red dashed line from `vInt[i]` to line 14.
- Red dashed line from `vInt[i+1]` to line 14.
- Red bracket above line 12:  $6-2=4$
- Red bracket below line 14:  $0$  and  $1$

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the `for` loop indicates the calculation `6-2=4`, representing the number of elements remaining to be sorted.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	<- vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	<- vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables and the array vInt. The code snippet shows the selection sort algorithm. Red dashed lines and annotations highlight the execution flow and the calculation of the number of elements to compare (6-2=4).

Annotations in the code:

- Line 11:  $6-2=4$  (red bracket)
- Line 14:  $1$  (red arrow pointing to vInt[i])
- Line 15:  $2$  (red arrow pointing to vInt[i+1])

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	← i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition `i < (CANT-resto)` in the code. A red bracket above the loop condition indicates the calculation `6-2=4`, representing the number of elements remaining to be sorted.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	<-vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	<-vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from line 12 to line 14:  $6-2=4$
- Red dashed line from line 14 to line 16: 2
- Red dashed line from line 16 to line 18: 3

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	← i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the variable `i` (line 6) to the memory address `0x7FFE6E0B` in the memory table. A red bracket above the `for` loop (lines 12-13) indicates the calculation `6-2=4`, representing the number of elements remaining to be sorted.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from `for(i=0; i < (CANT-resto); i++)` to `vInt[4]` in memory.
- Red dashed line from `if (vInt[i] > vInt[i+1])` to `vInt[3]` in memory.
- Red dashed line from `vInt[i]` to `vInt[4]` in memory.
- Red dashed line from `vInt[i+1]` to `vInt[5]` in memory.
- Red bracket above `(CANT-resto)` with text `6-2=4`.
- Red number `3` below `i` in the `if` statement.
- Red number `4` below `i+1` in the `if` statement.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Handwritten annotations in the code:

- A red bracket above the `for` loop condition `(CANT-resto)` with the text `6-2=4`.
- A blue number `3` is written next to the `if` statement.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-2=4

vInt[3] = vInt[4]

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-2=4

vInt[4] = aux

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Annotation:  $6-2=4$  (underlined in red) indicates the calculation of the loop bound in the for loop.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	← i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition `(CANT-resto)` in the code, indicating that the value of `i` is used to calculate the upper bound of the inner loop. A red bracket above the `for` loop condition indicates the calculation `6-2=4`, representing the number of elements in the current unsorted subarray.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt [i] > vInt [i+1])
15         {
16             aux=vInt [i];
17             vInt [i]=vInt [i+1];
18             vInt [i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `resto` variable to the `for` loop condition `(CANT-resto)`. A red bracket above the `(CANT-resto)` expression indicates the calculation  $6-2=4$ .

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory cell in the table. A red bracket above the `for` loop indicates the calculation  $6-2=4$ , representing the number of elements to compare in the current pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables: aux (0x00000001E), resto (0x000000003), i (0x000000000), flag (0x000000000), and an array vInt containing {2, 286, 9, 59, 928, 3}. The code snippet shows the selection sort algorithm. A red dashed arrow points from the 'i' variable in the memory layout to the 'i' variable in the code. A red bracket highlights the calculation  $6-3=3$  in the code, indicating the number of elements to compare in the current pass.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< -vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	< -vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed arrows connect memory addresses to array indices:
  - 0x7FFE6E13 to `vInt[0]`
  - 0x7FFE6E17 to `vInt[1]`
- Red dashed arrows connect array indices to code:
  - `vInt[i]` to line 14
  - `vInt[i+1]` to line 17
- Red bracket above line 12:  $6-3=3$
- Red bracket below line 14: `0`
- Red bracket below line 17: `1`

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the `for` loop indicates the calculation `6-3=3`, representing the number of elements to compare in the current pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	<- vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	<- vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from `vInt[1]` to `vInt[i]` in the `if` statement.
- Red dashed line from `vInt[2]` to `vInt[i+1]` in the `if` statement.
- Red bracket above `(CANT-resto)` with text `6-3=3`.
- Red bracket below `vInt[i]` with text `1`.
- Red bracket below `vInt[i+1]` with text `2`.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition `i < (CANT-resto)` in the code. A red bracket above the loop condition indicates the calculation `6-3=3`, showing that the loop runs for `i` values 0, 1, and 2.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	<-vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	<-vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed line from line 12 to line 14:  $6-3=3$
- Red dashed line from line 14 to line 16: 2
- Red dashed line from line 16 to line 18: 3

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Handwritten annotations in the code:

- A red bracket above the `for` loop condition `(CANT-resto)` with the calculation  $6-3=3$ .
- A blue number `2` is written next to the opening curly brace of the inner `if` loop.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-3=3

vInt[2] = vInt[3]

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-3=3

vInt[3] = aux



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory addresses range from 0x7FFE6E03 to 0xFFFFFFFF. The variables aux, resto, i, and flag are stored in the first four memory locations. The array vInt is stored in the next six locations. The diagram shows the state of memory after the first pass of the selection sort, where the element 3 (at index 5) has been swapped with the element 2 (at index 0). The value 6-3=3 is highlighted in red, indicating the number of elements to be compared in the next pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

A red dashed arrow points from the expression  $6-3=3$  in the code to the value 0x03 in the memory cell at address 0x7FFE6E0B.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory cell (0x7FFE6E0F). A red bracket highlights the calculation `6-3=3` in the `for` loop condition, indicating the current range of elements being compared.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory cell in the table. A red bracket highlights the calculation `6-3=3` in the `for` loop, indicating the number of elements to compare.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables: aux (0x3B), resto (0x04), i (0x00), flag (0x00), and an array vInt containing {2, 286, 9, 59, 928, 3}. The code snippet shows the selection sort algorithm. A red dashed arrow points from the 'i' variable in the memory layout to the 'i' variable in the code. A red bracket highlights the calculation  $6-4=2$  in the code, indicating the number of elements to compare in the current pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	< vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for (i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram annotations:

- Red dashed line from `vInt[0]` to line 10.
- Red dashed line from `vInt[1]` to line 12.
- Red dashed line from `vInt[i]` to line 14.
- Red dashed line from `vInt[i+1]` to line 14.
- Red bracket above line 12:  $6-4=2$ .
- Blue '0' below line 14, pointing to the `>` operator.
- Blue '1' below line 14, pointing to the `>` operator.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	← i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-4=2`, showing that the loop iterates over the last two elements of the array.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	<- vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	<- vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for (i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram annotations:

- Red dashed arrows connect memory addresses to array indices:
  - 0x7FFE6E17 to vInt[1]
  - 0x7FFE6E1B to vInt[2]
- Red bracket above line 12:  $6-4=2$
- Red bracket above line 14:  $1$  (under  $vInt[i]$ ) and  $2$  (under  $vInt[i+1]$ )



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	← aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	- vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Annotation:  $6-4=2$  (under CANT-resto)

Annotation: 1 (under i in the inner loop)

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

6-4=2

vInt[1] = vInt[2]

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	← aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	← vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

6-4=2

← vInt[2] = aux

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `flag` variable in the memory table to the `while(flag)` loop in the code. A red bracket above the `for` loop indicates the calculation  $6-4=2$ , representing the number of elements to compare in the current pass.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition `i < (CANT-resto)` in the code. A red bracket above the loop condition indicates the calculation `6-4=2`, showing that the loop will iterate for `i=0` and `i=1` when `resto` is 4.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables: aux (0x09), resto (0x05), i (0x02), flag (0x01), and an array vInt containing {2, 286, 9, 59, 928, 3}. The code snippet shows the selection sort algorithm. A red dashed arrow points from the 'resto' variable to the 'CANT-resto' expression in the for loop, indicating the current number of elements to compare. A red bracket above the expression 'CANT-resto' shows the calculation: 6-4=2, indicating that the last two elements (928 and 3) are already in their final sorted positions.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables: aux (0x09), resto (0x05), i (0x02), and flag (0x00). The array vInt contains the values {2, 286, 9, 59, 928, 3}. The code snippet shows the first iteration of the selection sort algorithm, where the element at index 0 (2) is compared with the element at index 1 (286). Since 2 < 286, no swap occurs. The next iteration starts at index 1 (286) and compares it with the element at index 2 (9). Since 286 > 9, a swap occurs, resulting in the array {2, 9, 286, 59, 928, 3}. The process continues until the array is sorted.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if (vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the state of variables and the array vInt. The variable i is highlighted in yellow, and the value 6-5=1 is shown in red, indicating the current iteration of the inner loop.



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< -vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	< -vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for (i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating memory access and comparison in the sorting algorithm:

- Line 10: `resto++;` (resto becomes 1)
- Line 12: `for (i=0; i < (CANT-resto); i++)` (CANT=6, resto=1, so i goes from 0 to 4)
- Line 14: `if (vInt[i] > vInt[i+1])` (Comparison between vInt[0] and vInt[1])
- Line 15: `{` (Start of swap block)
- Line 16: `aux=vInt[i];` (Store vInt[0] in aux)
- Line 17: `vInt[i]=vInt[i+1];` (Shift vInt[1] to vInt[0])
- Line 18: `vInt[i+1]=aux;` (Shift aux to vInt[1])
- Line 19: `flag=1;` (Set flag to 1)
- Line 20: `}` (End of swap block)
- Line 21: `}` (End of for loop)
- Line 22: `} while (flag);` (Repeat while flag is 1)

Annotations in the diagram:

- Red dashed arrows point from `vInt[0]` and `vInt[1]` in the code to the corresponding memory locations in the table.
- A red bracket above line 12 indicates the calculation  $6-5=1$ .
- A red bracket below line 14 indicates the comparison between `vInt[0]` and `vInt[1]`.
- A red bracket below line 15 indicates the swap operation.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1  #include <stdio.h>
2  #define CANT 6
3
4  int main (void)
5  {
6  int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7  resto=0;
8  do
9  {
10     resto++;
11     flag=0;
12     for(i=0; i < (CANT-resto); i++)
13     {
14         if(vInt[i] > vInt[i+1])
15         {
16             aux=vInt[i];
17             vInt[i]=vInt[i+1];
18             vInt[i+1]=aux;
19             flag=1;
20         }
21     }
22 } while(flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-5=1`, showing that the loop will execute once.

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03

2

3

9

59

286

928

0x00	0x00	0x00	0x09
0x00	0x00	0x00	0x05
0x00	0x00	0x00	0x01
0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x02
0x00	0x00	0x00	0x03
0x00	0x00	0x00	0x09
0x00	0x00	0x00	0x3B
0x00	0x00	0x01	0x1E
0x00	0x00	0x03	0xA0
⋮	⋮	⋮	⋮
0x00	0x00	0x00	0x00

aux

resto

i

flag

vInt[0]

vInt[1]

vInt[2]

vInt[3]

vInt[4]

vInt[5]

0xFFFFFFFF

0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if(vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while(flag); <-----
23     return (0);
24 }
```

Con while(0) sale  
del loop do-while  
y termina

# Verificar valores con la función *imprimir*

## MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vlnt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vlnt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vlnt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vlnt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vlnt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vlnt[5]
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```
1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p,int n);
5
6 int main (void)
7 {
8     int aux,resto,i,flag, vlnt[CANT]={2,286,9,59,928,3};
9     imprimir(&vlnt[0],CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vlnt[i] > vlnt[i+1]){
16                aux=vlnt[i];
17                vlnt[i]=vlnt[i+1];
18                vlnt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while(flag);
23    imprimir(&vlnt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }
```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003					
0xA0004007	0x00	0x00	0x00	0x06	← n
0xA000400B	0x7F	0xFE	0x6E	0x10	← p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir ( int*p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf (" %d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0xXX	0xXX	0xXX	0xXX	← j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for (i=0; i < (CANT-resto); i++){
15            if (vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir ( int*p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf (" %d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x00	< - j - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir (int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf("%d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 ← vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x00	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux,resto,i,flag,vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0],CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }

```



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x01	<- j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir (int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf("%d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	0x7FFE6E14 vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x01	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir ( int*p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x02	<- j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir (int *p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf("%d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	0x7FFE6E18 vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x02	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for (i=0; i < (CANT-resto); i++){
15            if (vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir ( int*p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf (" %d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x03	<- j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir (int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf("%d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	0x7FFE6E1C vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x03	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux,resto,i,flag,vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0],CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x04	<- j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir ( int*p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf ("%d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	0x7FFE6E20 vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x04	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux,resto,i,flag,vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0],CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }

```



# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x05	<- j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir ( int *p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf (" %d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	0x7FFE6E24 vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x05	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir (&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for (i=0; i < (CANT-resto); i++){
15            if (vInt [i] > vInt [i+1]){
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir (&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir ( int*p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf (" %d \r\n", *(p+j));
31     }
32 }

```

# MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x06	<- j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir (int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while(flag);
23    imprimir(&vInt[0], CANT);
24    return (0);
25 }
26 void imprimir (int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf("%d \r\n", *(p+j));
31     }
32 }

```

```
carlos@carlos-R430-R480-R440: ~  
carlos@carlos-R430-R480-R440:~$ gcc -c burbuja.c -o burbuja.o -Wall  
carlos@carlos-R430-R480-R440:~$ gcc burbuja.o -o burbuja -Wall  
carlos@carlos-R430-R480-R440:~$ ./burbuja  
  
2      286      9      59      928      3  
  
2      3      9      59      286      928  
carlos@carlos-R430-R480-R440:~$
```

- compila con ***gcc -c burbuja.c -o burbuja.o -Wall***
- Linkea con ***gcc burbuja.o -o burbuja -Wall***
- Ejecuta con ***./burbuja***
- Primero imprime en la consola los valores desordenados
- Al finalizar el ordenamiento también los imprime en la consola