



Vectores

Valores extremos y ordenamiento

Mg. Ing. Facundo S. Larosa

Informática I

Instituto Universitario Aeronáutico
Universidad de la Defensa Nacional (UNDEF)

Ejemplo disparador

- Se desea hacer un programa que registre una serie de valores de temperatura y calcule el valor mínimo y máximo registrados y luego imprimirlos ordenados de menor a mayor...

¿Cómo hacemos?

Algoritmos básicos

- Búsqueda de valores extremos de un vector:
 - Máximo
 - Mínimo
- Ordenamiento de un vector:
 - Método de selección (*selection sort*)
 - Método de burbujeo (*bubble sort*)

Búsqueda de valores extremos de un vector

Los operadores relacionales ($<$, $>$, \leq , \geq) son binarios. Por lo tanto, para encontrar el valor máximo o mínimo de un vector, será necesario recorrerlo comparando de a dos elementos

Búsqueda de valores extremos de un vector

//Declaración de un vector


```
int vec[5]={3,2,5,-4,-2};
```

//Búsqueda del menor valor

1) Suponemos el primer valor como mínimo

2) Comparar el elemento siguiente del vector con el valor mínimo **supuesto**.

3) Si es menor, este es el nuevo valor mínimo supuesto.

vec[0]	3	 Es menor?
vec[1]	2	
vec[2]	5	
vec[3]	-4	
vec[4]	-2	

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	-4
vec[4]	-2

Búsqueda de valores extremos de un vector

//Declaración de un vector

```
int vec[5]={3,2,5,-4,-2};
```

//Búsqueda del menor valor

4) Comparar el elemento siguiente del vector con el valor mínimo **supuesto**.

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	-4
vec[4]	-2

Es menor?



5) Si es menor, este es el nuevo valor mínimo supuesto.

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	-4
vec[4]	-2


Búsqueda de valores extremos de un vector

//Declaración de un vector

```
int vec[5]={3,2,5,-4,-2};
```

//Búsqueda del menor valor

6) Comparar el elemento siguiente del vector con el valor mínimo **supuesto**.

vec[0]	3	Es menor? 
vec[1]	2	
vec[2]	5	
vec[3]	-4	
vec[4]	-2	

7) Si es menor, este es el nuevo valor mínimo supuesto.

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	-4
vec[4]	-2

Búsqueda de valores extremos de un vector

//Declaración de un vector

```
int vec[5]={3,2,5,-4,-2};
```


//Búsqueda del menor valor

8) Comparar el elemento siguiente del vector con el valor mínimo **supuesto**.

9) Si es menor, este es valor mínimo del vector

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	-4
vec[4]	-2

Es menor?



vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	-4
vec[4]	-2

Hemos encontrado el menor valor del vector!!

A programar

- 1) Ingrese 10 valores numéricos que representarán muestras de presión provenientes de un tanque presurizado. Luego, imprimir el mínimo y el máximo de la serie. Determinar si el promedio de los valores es más cercano al mínimo o al máximo y la diferencia. Obtener conclusiones sobre la posible utilización de este cálculo.
- 2) Para el ejercicio anterior, determinar la posición del valor máximo y del valor mínimo.

Métodos de ordenamiento

Existen una variedad de métodos de ordenamiento:

- Método de selección (*selection sort*)
- Método de burbujeo (*bubble sort*)
- Método por mezcla (*merge sort*)
- Método por montículos (*heap sort*)
- Método rápido (*quick sort*)
- Método Shell (*Shell sort*)
- Otros...

Métodos de ordenamiento básicos

Existen una variedad de métodos de ordenamiento:

- Método de selección (*selection sort*)
- Método de burbujeo (*bubble sort*)
- Método por mezcla (*merge sort*)
- Método por montículos (*heap sort*)
- Método rápido (*quick sort*)
- Método Shell (*Shell sort*)
- Otros...

Método de ordenamiento por selección

Este método consiste en buscar un valor extremo del vector (por ejemplo, el mínimo o el máximo) y ubicarlo en la posición adecuada de acuerdo al criterio de ordenamiento. Luego, se vuelve a realizar la misma operación sobre la fracción del vector restante.

Veámoslo con un ejemplo sencillo para luego inferir una metodología general...

Método de ordenamiento por selección

//Declaración de un vector "desordenado"

```
int vec[5]={3,2,5,1,-4};
```

//Vamos a ordenar el vector en orden creciente (de menor a mayor)

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	1
vec[4]	-4

Método de ordenamiento por selección

1) Buscar el valor mínimo* del vector

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	1
vec[4]	-4

2) Intercambiar el valor mínimo por el del lugar que le corresponde

vec[0]	-4
vec[1]	2
vec[2]	5
vec[3]	1
vec[4]	3

* Sin pérdida de generalidad, el valor extremo elegido podría ser el máximo o el mínimo según se trate de ordenar de forma creciente o decreciente

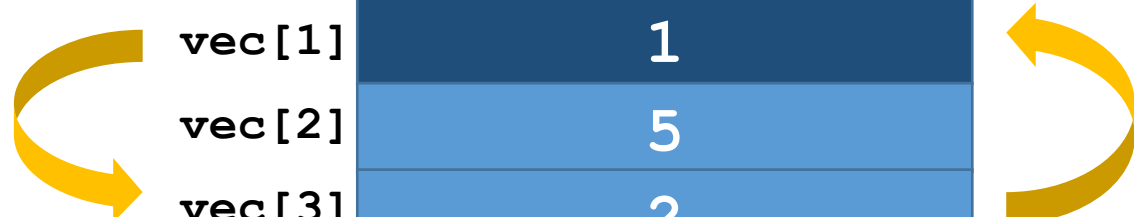
Método de ordenamiento por selección

3) Buscar el valor mínimo de la fracción restante del vector

<code>vec[0]</code>	-4
<code>vec[1]</code>	2
<code>vec[2]</code>	5
<code>vec[3]</code>	1
<code>vec[4]</code>	3

4) Intercambiar el valor mínimo por el del lugar que le corresponde

<code>vec[0]</code>	-4
<code>vec[1]</code>	1
<code>vec[2]</code>	5
<code>vec[3]</code>	2
<code>vec[4]</code>	3



Método de ordenamiento por selección

5) Buscar el valor mínimo de la fracción restante del vector

vec[0]	-4
vec[1]	1
vec[2]	5
vec[3]	2
vec[4]	3

6) Intercambiar el valor mínimo por el del lugar que le corresponde

vec[0]	-4
vec[1]	1
vec[2]	2
vec[3]	5
vec[4]	3

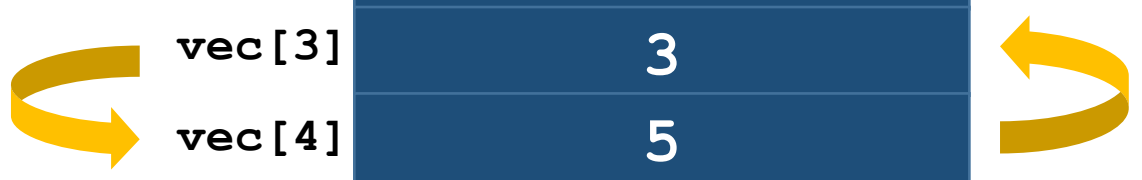
Método de ordenamiento por selección

7) Buscar el valor mínimo de la fracción restante del vector

vec[0]	-4
vec[1]	1
vec[2]	2
vec[3]	5
vec[4]	3

8) Intercambiar el valor mínimo por el del lugar que le corresponde

vec[0]	-4
vec[1]	1
vec[2]	2
vec[3]	3
vec[4]	5



Con este último paso ¡el ordenamiento ya está completado!

Método de ordenamiento por selección: Resumen

Iteración 0

3
2
5
1
-4

-4
2
5
1
3

Iteración 3

-4
1
2
5
3

-4
1
2
3
5

Iteración 1

-4
2
5
1
3

-4
1
5
2
3

Iteración 2

-4
1
5
2
3

-4
1
2
5
3

Complejidad del algoritmo

Para un vector de '**n**' elementos se requieren:

- '**n-1**' comparaciones en el primer paso (para “decantar” el mínimo o máximo entre '**n**' variables)
- '**n-2**' comparaciones en el segundo paso (para “decantar” el mínimo o máximo entre '**n - 1**' variables)
-

En definitiva se requiere una cantidad de comparaciones igual a:

$$(n-1) + (n-2) + \dots + 1 = \sum_{i=1}^{n-1} i = \sum_{i=1}^{n-1} \binom{i}{1}$$

$$\sum_{i=1}^{n-1} \binom{i}{1} = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{1}{2}n(n-1) = \frac{1}{2}(n^2 - n)$$

Método de ordenamiento por selección: Implementación

*/*Vamos a implementar una función a la que se le pasa un vector de enteros Junto con su tamaño y nos devuelve el vector ordenado*/*

```
void selSort (int * vec, int n)
{
```



```
}
```

Método de ordenamiento por selección: Implementación

*/*Vamos a implementar una función a la que se le pasa un vector de enteros junto con su tamaño y nos devuelve el vector ordenado*/*

```
void selSort (int * vec, int n)
{
    int i,j,posMin,aux;

    for(i=0;i<n-1;i++)
    {
        posMin=i;
        for(j=i+1;j<n;j++)
        {
            if(vec[j]<vec[posMin])
                posMin=j;
        }
        aux=vec[i];
        vec[i]=vec[posMin];
        vec[posMin]=aux;
    }
}
```

Búsqueda
del
mínimo

Intercambio (swap) de variables

Método de ordenamiento por burbujeo

//Declaración de un vector "desordenado"

```
int vec[5]={3,2,5,1,-4};
```

//Vamos a ordenar el vector en orden creciente

//(de menor a mayor)

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	1
vec[4]	-4

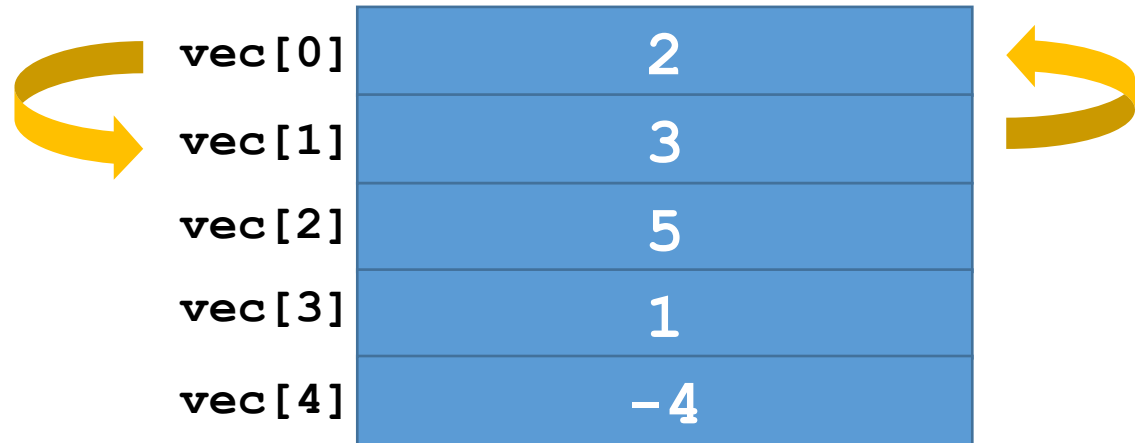


Método de ordenamiento por burbujeo

1) Comparar dos variables adyacentes

vec[0]	3
vec[1]	2
vec[2]	5
vec[3]	1
vec[4]	-4

2) Intercambiar los valores si el orden no es el que corresponde

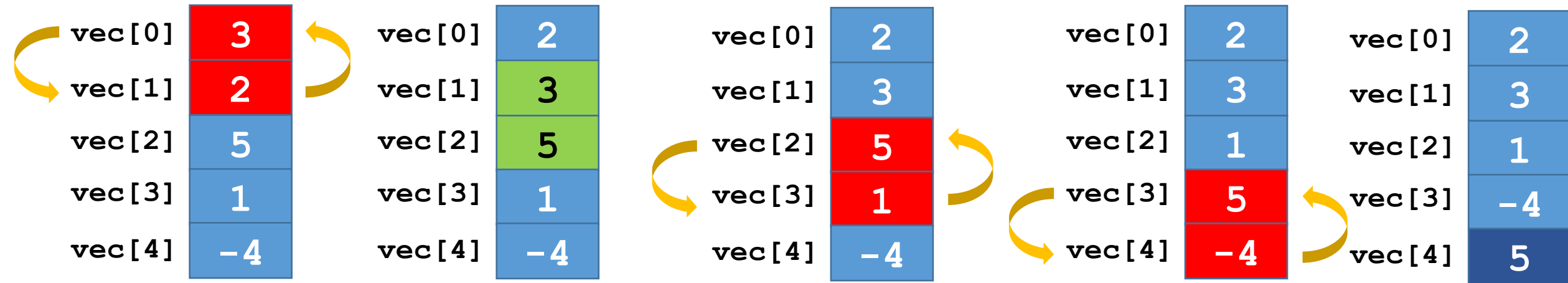


The diagram illustrates the second step of the bubble sort algorithm: swapping adjacent elements if they are in the wrong order. It shows two states of the array. In the initial state (left), the array is [3, 2, 5, 1, -4]. The first two elements, 3 and 2, are highlighted in red. A yellow curved arrow points from the red box of index 0 to the red box of index 1. In the final state (right), the array is [2, 3, 5, 1, -4]. The first two elements, 2 and 3, are now highlighted in blue. A yellow curved arrow points from the blue box of index 1 back to the blue box of index 0, indicating the swap.

vec[0]	2
vec[1]	3
vec[2]	5
vec[3]	1
vec[4]	-4

Método de ordenamiento por burbujeo

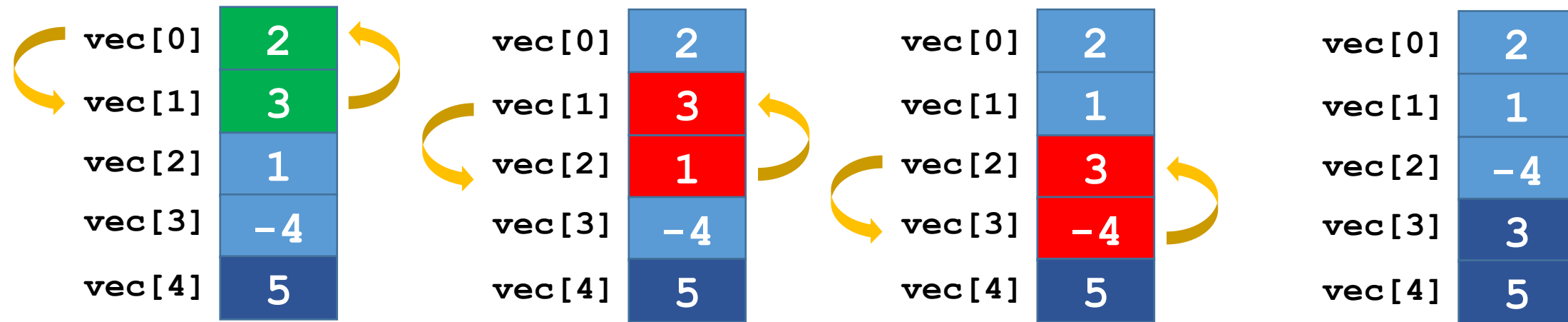
3) El procedimiento se repite hasta llegar al final del vector



Al finalizar la primera iteración, el último elemento del vector está ordenado

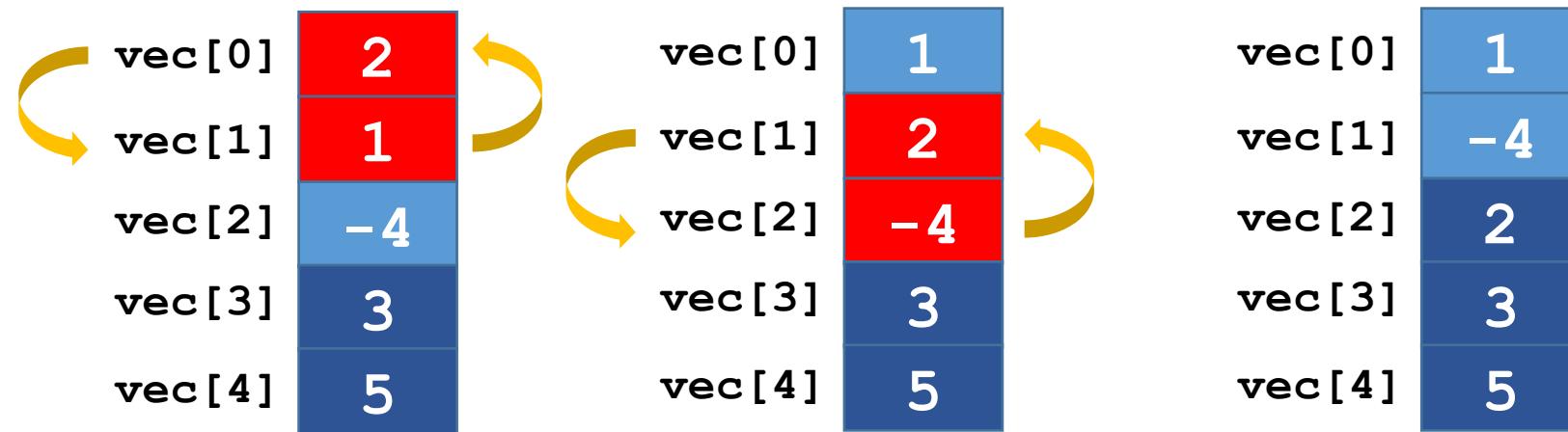
Método de ordenamiento por burbujeo

4) El procedimiento se realiza nuevamente sobre la fracción del vector no ordenada hasta que se completa el ordenamiento...



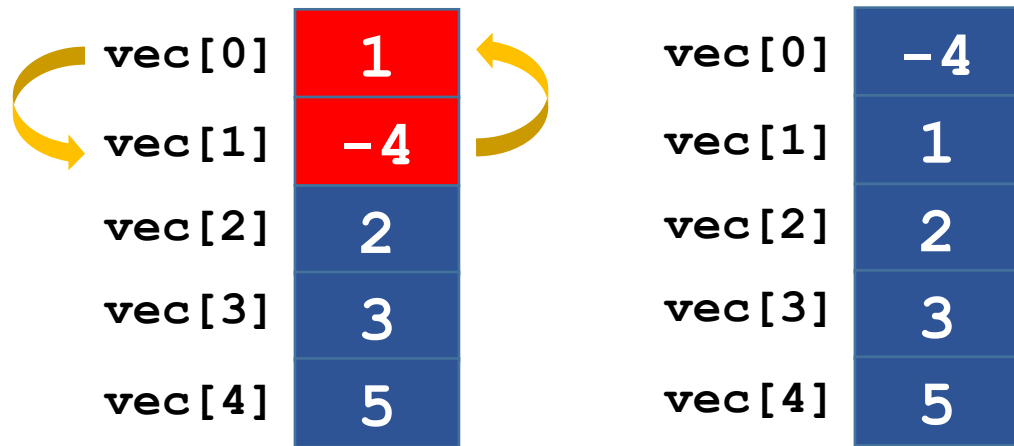
Método de ordenamiento por burbujeo

4) El procedimiento se realiza nuevamente sobre la fracción del vector no ordenada hasta que se completa el ordenamiento...



Método de ordenamiento por burbujeo

4) El procedimiento se realiza nuevamente sobre la fracción del vector no ordenada hasta que se completa el ordenamiento...



Al completar la última iteración el vector está ordenado

Método de ordenamiento por burbujeo : Resumen

Iteración 0

3
2
5
1
-4

2
3
5
1
-4

2
3
5
1
-4

2
3
1
5
-4

2
3
1
-4
5

Iteración 3

1
-4
2
3
5

-4
1
2
3
5

Iteración 1

2
3
1
-4
5

2
3
1
-4
5

2
1
3
-4
5

2
1
-4
3
5

Iteración 2

2
1
-4
3
5

1
2
-4
3
5

1
-4
2
3
5

Método de ordenamiento por burbujeo: Implementación

*/*Vamos a implementar una función a la que se le pasa un vector de enteros junto con su tamaño y nos devuelve el vector ordenado*/*

```
void bubbleSort (int * vec, int n)
{
```



```
}
```

Método de ordenamiento por burbujeo:

Implementación

*/*Vamos a implementar una función a la que se le pasa un vector de enteros junto con su tamaño y nos devuelve el vector ordenado*/*

```
void bubbleSort (int * vec, int n)
{
    int i,j,aux;

    for(i=0;i<n-1;i++)
        for(j=0;j<n-1-i;j++)
            if(vec[j]>vec[j+1])
            {
                aux=vec[j];
                vec[j]=vec[j+1];
                vec[j+1]=aux;
            }
}
```

Intercambio (swap) de variables