Búsqueda de la posición del menor elemento en un arreglo a partir de la posición 0 de ese arreglo.

Algoritmo:

1. Se almacena en una variable auxiliar el elemento del arreglo correspondiente a la posición 0 (se lo presupone como el menor), y en otra variable la posición 0.
2. Se compara este valor con el resto de los valores del arreglo, y cada vez que se encuentra un elemento menor al guardado en la variable auxiliar, se actualiza este valor con el encontrado.

```
int posicionMenor(int a[], int cantVal, int pos){
    int menor = a[pos];
    int posMenor = pos;
    int index = pos + 1;
    while (index < cantVal){
        if(menor > a[index]){
            menor = a[index];
            posMenor = index;
        }
        index++;
    }
    return posMenor;
}
```

Ordenación por el método de selección

Algoritmo:

1. Se busca la posición del menor elemento de un arreglo comenzando desde la posición i(0), y se intercambia el menor elemento con el de la posición i.
2. Se repite el primer paso comenzando con i = 0 y terminando con i = N-1; siendo N el tamaño del arreglo.

19	**23**	**45**	**33**	**18**	**1**	**12**	**9**
0	1	2	3	4	5	6	7
i							

1	23	45	33	18	**19**	12	9

i

1	23	45	33	18	**19**	12	9

i

1	**9**	45	33	18	**19**	12	**23**

i

!!

```
void ordenacionSeleccion(int a[ ], int cantVal){
    int posMenor;
    int i = 0;
    int aux;
    while(i< cantVal - 1){ /// llego hasta la anteúltima posición
        posMenor = posicionMenor(a, cantVal, i);
        aux = a[posMenor];
        a[posMenor] = a[i];
        a[i] = aux;
        i++;
    }
}
```

Las instrucciones

```
    aux = a[posMenor];
    a[posMenor] = a[i];
    a[i] = aux;
```

pueden cambiarse por una función que realice la tarea de intercambio.

```
void intercambio(int A[ ], int i, int j){
    int aux = A[i];
    A[i] = A[j];
    A[j] = aux;
}

void intercambio (int *a, int *b){
    int aux;
    aux = *a;
    *a = *b;
```

```
    *b = *aux;
}

void ordenacionSeleccion(int a[ ], int cantVal){
    int posMenor;
    int i = 0;
    while(i<cantVal - 1){ //llego hasta la anteúltima posición
        posMenor = posicionMenor(a, cantVal, i);
        intercambio(&a[porMenor], &a[i]);
        i++;
    }
}
```

Inserción de un elemento en un arreglo ordenado (manteniendo el orden)

Supongamos que debemos insertar un nuevo elemento en un arreglo **ordenado**. Se supone que el arreglo no ha sido usado en forma completa.

10	20	30	40				
			u				

u marca la posición de la última celda ocupada con información útil (validos).
Y sea **dato** una variable con el valor a insertar.

Se recorre el arreglo desde la última posición a la primera buscando el lugar en donde insertar el nuevo dato. Mientras esto se hace, se realiza también un “corrimiento” de los elementos del arreglo, para crear un espacio en donde ubicar el nuevo valor.

Supongamos que se desea insertar el valor **25.**

10	20	30	40				
			u				

10	20	25	30	40			
				u+1			

```
void insertar(int a[ ], int u, int dato){
    int i= u; //ultima pos valida
    while (i >= 0 && dato < a[i]){
        a[i+1] = a[i];
        i--;
    }
    a[i+1] = dato;
}
// Se debe incrementar el valor de u para indicar que ha aumentado el
tamaño del arreglo.
```

Ordenación por inserción

Supongo un arreglo completo y **desordenado**.

```
void ordenacionInsercion(int a[ ], int cantVal){
    int u=0;
    while (u < cantVal - 1){
     //llega hasta la posición del anteúltimo elemento del arreglo.
        insertar(a, u, a[u+1]);
        u++;
    }
}
```

A medida que el algoritmo evoluciona, se marca en el arreglo una “zona ordenada” a la izquierda, separada de otra “zona desordenada” a la derecha. Se va tomando el siguiente elemento (a[u+1]) de la zona desordenada y se lo inserta dentro de la zona ordenada (se utiliza también el lugar u+1, para hacer crecer la zona ordenada).