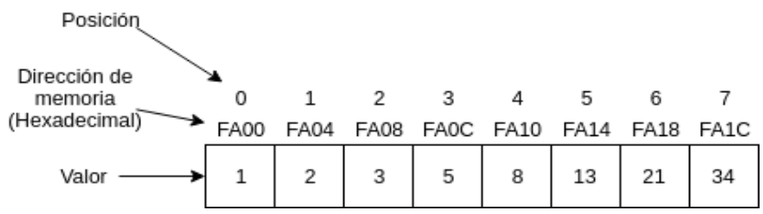
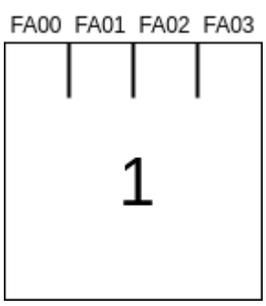
Vectores / Arrays Unidimensionales

Definición

Un vector o array unidimensional es un conjunto de variables, que tienen la particularidad de estar ubicadas de forma contígua en la memoria. Estas variables son del mismo tipo. Decimos que el vector tiene un tipo, que es el de sus variables. Estas variables no tienen un nombre, sólo el vector posee uno, por lo que son referidas por medio de su posición dentro de él.



En la figura se observa un vector de variables de tipo **int**. Prestando atención a las direcciones de memoria, podemos ver que están distanciadas en 4. Estamos asumiendo que el programa donde se generó, fue compilado en modo 32 bits, en el que las variables de tipo int ocupan 4 bytes. La dirección de cada variable es en realidad la dirección de su primer byte (cada byte tiene su propia dirección). Esto se puede evidenciar mejor en la sig. figura:



Declaración de una variable vector

Declaramos un vector de la sig. manera:

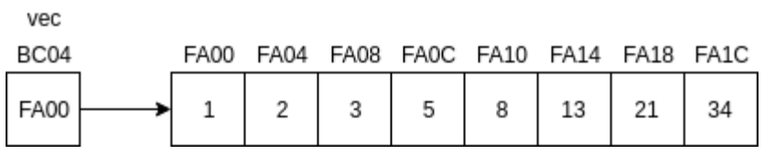
```
int vec[8];
```

También podemos inicializarlo en la declaración:

```
int vec[] = {1, 2, 3, 5, 8, 13, 21, 34};
```

En este caso podemos omitir la dimensión.

Declaramos un vector de tipo int, de 8 posiciones. En realidad, lo que definimos fue un puntero (vec), el cual apunta (su valor es una dirección) a la primera de las 8 variables, como se puede ver en la sig. fig.:



Observar que el puntero vec también tiene su propia dirección.

Acceso, modificación y recorrida del vector

La forma de acceder a una determinada posición del vector es a través de subíndices:

```
int a = vec[5]; //a: 13
vec[5] = 11; //Cambiamos el valor 13 por el 11
```

Si queremos recorrer todo el vector, lo podemos hacer mediante una iteración definida, o sea mediante un ciclo for:

```
int i;
for(i = 0; i < 8; i++)
{
     printf("Valor en la pos %d: %d\n", I, vec[i]);
}
```

Ordenamiento

Método burbujeo

Este método consiste en tomar pares de elementos contíguos, compararlos, y ordenarlos por medio de un intercambio de sus valores. Este proceso se repite, pasando por todos los elementos del vector, hasta que se hayan hecho N-1 pasadas, siendo N la cantidad de elementos del vector, o hasta que en una pasada se verifica que no se realizó ningún intercambio.

Observar que los elementos mayores, se ordenan mucho más rápidamente que los menores, de este hecho toma el nombre el método.

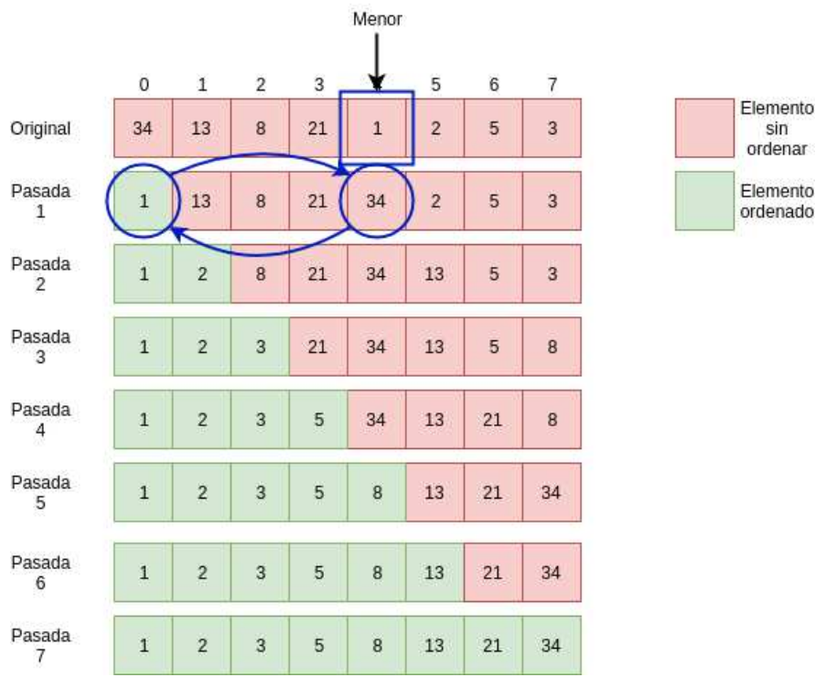
El algoritmo es el siguiente:

```
void ordenarBurbujeo(int* v, int cantElem)
{
	int i = 1, j;
	int huboIntercambios = 1;
	while(huboIntercambios && i < cantElem)
	{
		huboIntercambios = 0;
		for(j = 0; j < cantElem - i; j++)
		{
			if(v[j] > v[j+1])
			{
				intercambiar(&v[j], &v[j+1]);
				huboIntercambios = 1;
			}
		}

		i++;
	}
}
```

El ciclo externo ejecuta una vez por cada pasada. La variable i marca el nro de pasada. El ciclo interno ejecuta la pasada, realizando el intercambio cuando el primer elemento es mayor que el segundo. La variable booleana huboIntercambios se usa para cortar el algoritmo después de una pasada sin intercambios. Cada pasada va siendo más corta que la anterior, debido a que luego de cada una, un elemento es ubicado en su posición final.

Método Selección

El método se basa en encontrar el elemento menor e intercambiarlo con el de la primera posición, esto constituye una pasada. Luego se busca el menor entre los elementos restantes y se intercambia con el segundo. Este proceso sigue hasta que se hayan realizado N-1 pasadas.



Se puede apreciar que en las pasadas 6 y 7 no se realizan intercambios, pero a diferencia de burbujeo, este método es incapaz de detectar si el vector ya está ordenado. Sin embargo, la performance de este algoritmo es superior a la de burbujeo, dado que éste realiza muchos menos intercambios (uno por pasada), cuando burbujeo realiza N-1 intercambios por pasada en el peor caso. Otra cosa a observar es que, al igual que en burbujeo, cada pasada es más corta que la anterior, debido a los elementos que van quedando en su ubicación final.

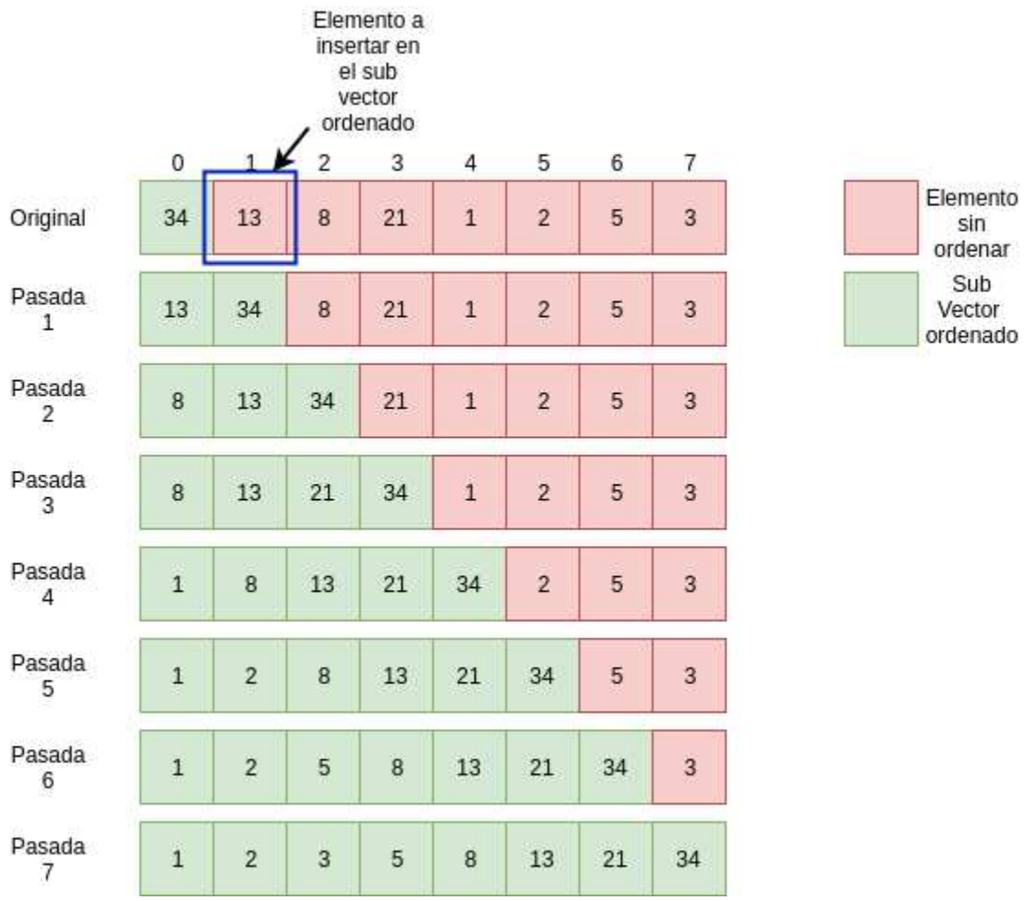
Algoritmo:

```
void ordenarSeleccion(int* v, int cantElem)
{
	int m;

	for(int i = 0; i < cantElem – 1; i++)
	{
		m = buscarMenor(v, i, cantElem – 1);

		if(m != i)
			intercambiar(&v[m], &v[i]);
	}
}

int buscarMenor(int* v, int desde, int hasta)
{
	int m = desde;
	for(int j = desde + 1; j <= hasta; j++)
	{
		if(v[j] < v[m])
			m = j;
	}

	return m;
}
```

La i del for marca la posición donde debe ir el menor elemento que se encuentre en la pasada. La función buscarMenor realiza la búsqueda entre los subíndices indicados, y devuelve su posición. Luego si la posición del menor es distinta a la posición donde debe ir (si no está ya el menor en donde debe ir), se realiza el intercambio.

Método Inserción

Se basa en imaginarnos que existe una parte del vector que ya está ordenada (subvector), y se van insertando en orden los elementos restantes, hasta que todos los elementos hayan quedado en el subvector.



Al inicio, sólo hay un elemento en el subvector, el primero, y como es uno solo, está ordenado. Luego se inserta en forma ordenada el primer elemento del grupo de los desordenados, en este caso el segundo. Al final de la primera pasada hay 2 elementos ordenados en el sub vector. Cada pasada inserta un nuevo elemento en el subvector, hasta que todos los elementos se encuentran en él, y como las inserciones fueron en orden, todo el vector queda ordenado.

El algoritmo:

```
void ordenarInsercion(int* v, int cantElem)
{
	int j;
	int elemAIns;

	for(int i = 1; i < cantElem; i++)
	{
		elemAIns = v[i];
		j = i - 1;
		while(j >= 0 && elemAIns < v[j])
		{
			v[j+1] = v[j];
			j--;
		}

		v[j+1] = elemAIns;
	}
}
```

La variable i indica la posición del elemento a insertar en el subvector, y también el nro de pasada. Se guarda el valor del elemento a insertar en la variable elemAIns, y en el ciclo interno se realiza la inserción ordenada. j apunta inicialmente al último elemento del subvector, y va pasando por todos. Mientras el elemento a insertar sea menor que el apuntado por j, este último se desplaza un lugar a la derecha, dejando un espacio para una posible inserción, u otro desplazamiento. Si el elemento a insertar es mayor o igual, o si j se pasa de 0 (porque el elemento a insertar es menor que todos), se produce la inserción en donde quedó el hueco (j+1).