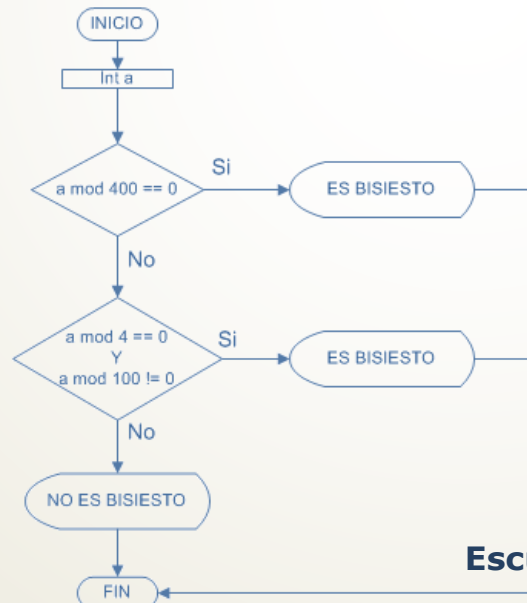


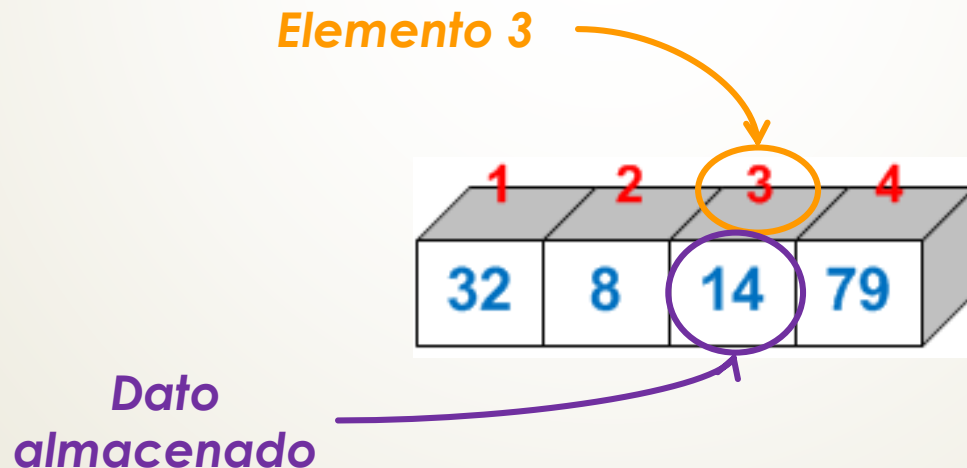
Programación Estructurada

UNIDAD V: ARREGLOS. CONCEPTOS BÁSICOS



Arreglos. Definición

- Un arreglo es un *conjunto finito* de elementos del mismo tipo cuyo acceso se realiza a través de *índices*.
- Los *índices* permiten identificar en forma individual cada elemento del arreglo.



Índices

- Los índices permiten referenciar posiciones específicas de un arreglo.
- Los índices pueden ser constantes, variables o expresiones de tipo ordinal.
- El valor mínimo de un índice de arreglo se denomina límite inferior (**LI**), mientras que el máximo valor se llama límite superior (**LS**).

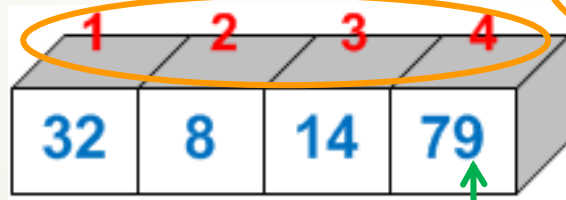
Clasificación

- Según la organización de sus elementos, los arreglos se clasifican en:
 - Unidimensionales (vectores)
 - Bidimensionales (matrices)
 - Multi-dimensionales (n dimensiones)
- Los arreglos emplean tantos índices como dimensiones posean para referenciar sus elementos individuales.

Vectores

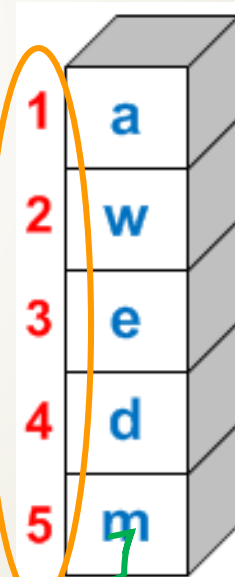
- El vector es el tipo más simple de arreglo ya que sus elementos se disponen en una única fila o columna y pueden accederse utilizando un solo índice.

Valor del índice para cada posición



Vector Fila

Datos del vector



Vector Columna

¿Cómo se declara?

- ▶ En general, se puede aplicar la siguiente sintaxis:

```
PROGRAMA ejemplo_vectores
```

```
CONSTANTES
```

```
    MAX_ELEMENTOS=10
```

```
TIPOS
```

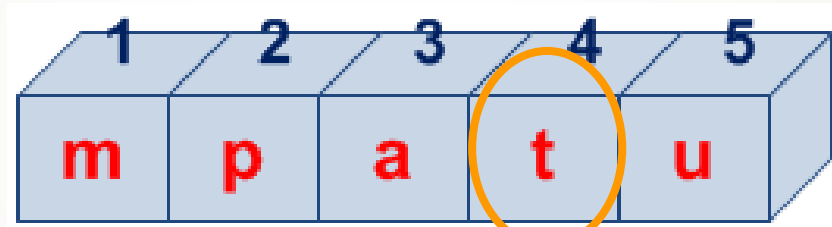
```
    tipo_vector=ARREGLO[1..MAX_ELEMENTOS]de tipo_dato
```

```
VARIABLES
```

```
    nombre_variable: tipo_vector
```

¿Cómo se accede?

- ➔ Para acceder a una posición específica de un vector debe indicarse el nombre del vector y el índice del elemento requerido.



`letras[4]`

- `letras`: nombre del vector
- `4`: posición del vector

Matrices

- Un arreglo bidimensional (matriz o tabla) es un conjunto de elementos, todos del mismo tipo, que se organizan en filas y columnas.
- Requiere de 2 índices para identificar cada posición.
- El primer índice especifica la fila y el segundo la columna.

¿Cómo se declara?

- ➡ En general, se puede aplicar la siguiente sintaxis:

```
PROGRAMA ejemplo_matrices
```

```
CONSTANTES
```

```
    FILAS=12
```

```
    COLS=14
```

```
TIPOS
```

```
    tipo_matriz=ARREGLO[1..FILAS,1..COLS]de tipo_dato
```

```
VARIABLES
```

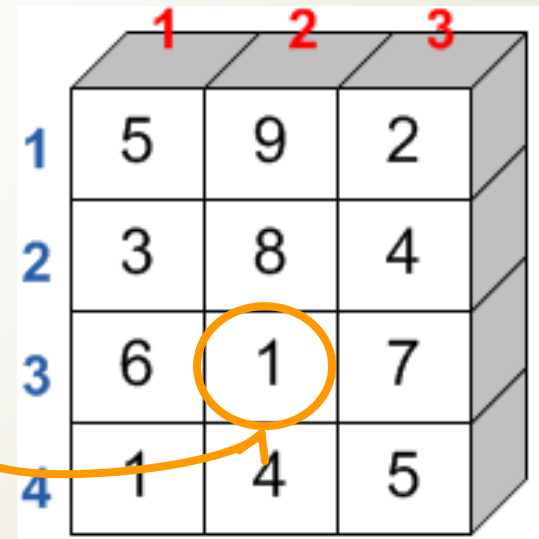
```
    nombre_variable: tipo_matriz
```

¿Cómo se accede?

- Para acceder a una posición específica de una matriz debe indicarse el nombre de la matriz y los índices de fila y columna del elemento requerido.

- *números*: nombre de la matriz
- 3: fila de la matriz
- 2: columna de la matriz

números [3, 2]



	1	2	3
1	5	9	2
2	3	8	4
3	6	1	7
4	1	4	5

Multidimensionales

- Un arreglo se dice multidimensional si posee 3, 4 o n dimensiones.
- Para acceder a los elementos de un arreglo multidimensional es necesario especificar tantos índices como dimensiones tenga el arreglo.

¿Cómo se declara?

- ➡ En general, se puede aplicar la siguiente sintaxis:

```
PROGRAMA ejemplo_multidimensional
```

```
CONSTANTES
```

```
  D1=12
```

```
  D2=4
```

```
  ...
```

```
  DN=7
```

```
TIPOS
```

```
  tipo_mult=ARREGLO[1..D1,1..D2,...,1..DN]de tipo_dato
```

```
VARIABLES
```

```
  nombre_variable: tipo_mult
```

¿Cómo se accede?

- Para acceder a una posición específica de un arreglo multidimensional debe indicarse el nombre del arreglo y los índices correspondientes a cada dimensión.

- *alfabeto*: nombre del arreglo
- 4: primera dimensión
- 3: segunda dimensión
- 1: tercera dimensión

`alfabeto[4,3,1]`

	1	2	3
1	A	Z	U
2	M	V	O
3	E	T	W
4	Q	X	Y

Rango de un arreglo

- El *rango de un arreglo* indica el número de elementos o posiciones que posee un arreglo.
- El rango de un arreglo puede calcularse de la siguiente forma:
 - Rango Vector = $LS - LI + 1$
 - Rango Matriz = $(LS_1 - LI_1 + 1) \times (LS_2 - LI_2 + 1)$
 - Rango n-dimensional
 $= (LS_1 - LI_1 + 1) \times (LS_2 - LI_2 + 1) \times (LS_3 - LI_3 + 1) \times \dots \times (LS_n - LI_n + 1)$

Operaciones

- Asignación
- Lectura/escritura
- Recorrido
- Actualización
 - agregar, insertar y borrar
- Búsqueda
 - secuencial y binaria
- Intercalación
- Ordenación
 - burbuja, selección, inserción, shaker sort, rápido y shell

Asignación

- La operación de asignación permite almacenar valores en un arreglo.
- Esta operación se ejecuta sobre una posición específica, usando la siguiente sintaxis:

`nombre_vector[índice] ← valor`

Por ejemplo: `datos[5] ← 36`

donde *datos* es un arreglo numérico y 5 es el índice que identifica al quinto elemento.

Lectura/escritura

- La operación de *lectura* permite asignar a una posición del arreglo un valor introducido por teclado.

LEER `nombre_vector`[`índice`]

- La operación de *escritura* permite mostrar por pantalla el contenido de una posición del arreglo.

ESCRIBIR `nombre_vector`[`índice`]

Recorrido

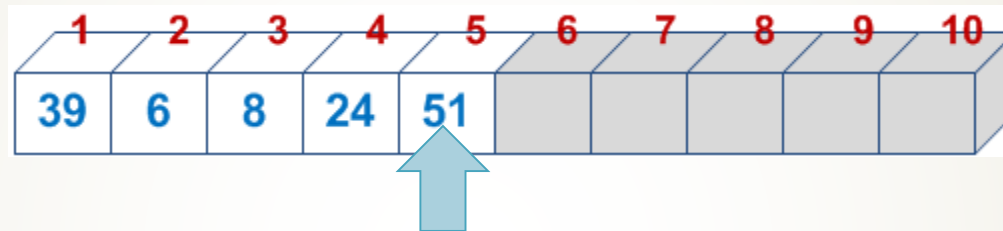
- Las operaciones de asignación, lectura y escritura, se aplican a elementos individuales de un arreglo.
- Para procesar todos los elementos de un arreglo es necesario recorrerlo.
- El acceso sucesivo a las posiciones un arreglo se denomina recorrido del arreglo.
- Para recorrer un arreglo se usan estructuras repetitivas que incluyen sentencias para la asignación, lectura o escritura.

Actualización. Agregar (1)

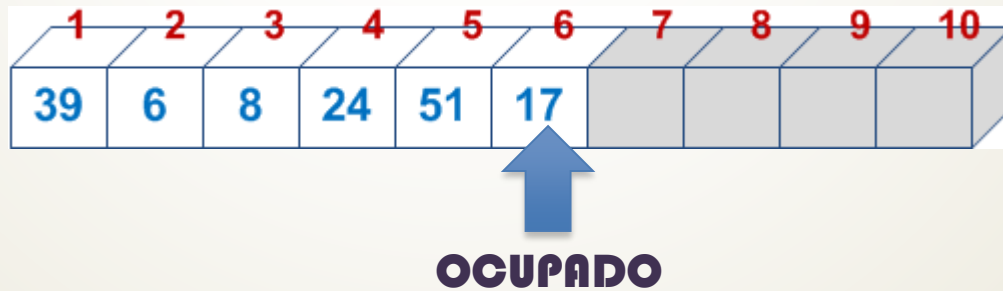
- La operación *agregar* añade un nuevo elemento a un arreglo a continuación del último elemento agregado
- Esta operación verifica que exista espacio de memoria suficiente para el nuevo elemento.
- Se utiliza una variable extra para indicar la cantidad de datos presentes en el arreglo. Esta variable apunta a la última posición ocupada.

Actualización. Agregar (2)

Antes de agregar el valor 17



Después de agregar el valor 17



Actualización. Agregar (3)

PROCEDIMIENTO agregar (E/S num:vector,
E/S ocupado:entero,
E nuevo:entero)

INICIO

SI ocupado=MAX ENTONCES
 ESCRIBIR "VECTOR LLENO"

Verificación
de espacio

SINO
 ocupado ← ocupado+1
 num[ocupado] ← nuevo
FINSI

Agregado del
nuevo
elemento
(actualización
de ocupado)

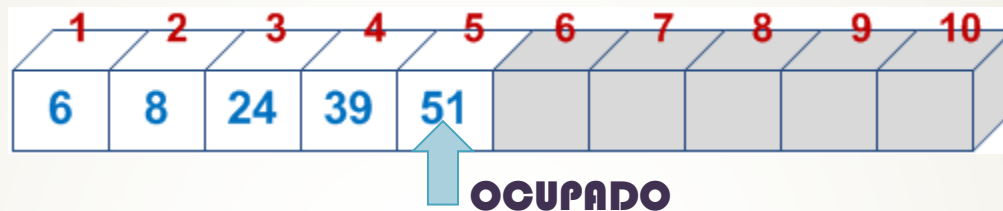
FIN

Actualización. Insertar (1)

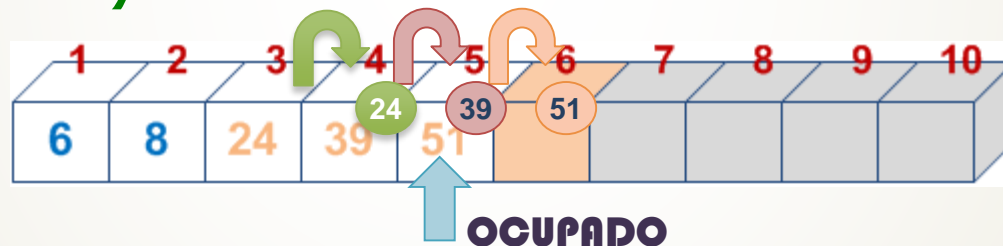
- La operación *insertar* introduce un nuevo elemento en el arreglo, en una posición específica.
- Para dar lugar al nuevo dato, los elementos del arreglo se desplazan un lugar a partir de la posición de inserción.
- Esta operación verifica que exista espacio de memoria suficiente para el nuevo elemento.

Actualización. Insertar (2)

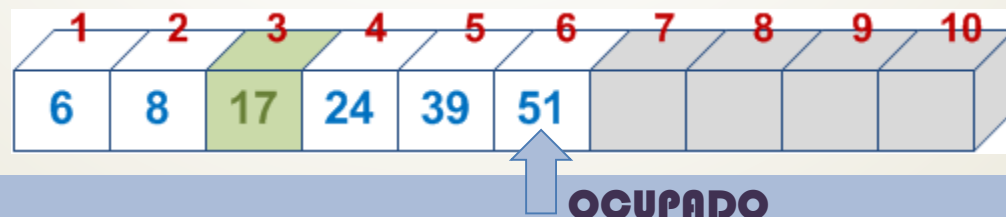
Antes de insertar el valor 17



Desplazamiento de elementos (a partir de la posición de inserción)



Después de insertar el valor 17



Actualización. Insertar (3)

```
PROCEDIMIENTO insertar(E/S num:vector, E/S ocupado:entero,  
                        E nuevo:entero)
```

```
VARIABLES
```

```
    i,j:entero
```

```
INICIO
```

```
    SI ocupado=MAX ENTONCES
```

```
        ESCRIBIR "VECTOR LLENO"
```

```
    SINO
```

```
        i ← 1
```

Identificación
de la posición
de inserción

```
        MIENTRAS (i ≤ ocupado) Y (num[i] < nuevo) HACER
```

```
            i ← i + 1
```

```
        FIN MIENTRAS
```

```
        j ← ocupado
```

Desplazamiento
de datos

```
        MIENTRAS j ≥ i HACER
```

```
            num[j + 1] ← num[j]
```

```
            j ← j - 1
```

```
        FIN MIENTRAS
```

```
        num[i] ← nuevo
```

```
        ocupado ← ocupado + 1
```

Inserción del nuevo dato y
actualización de ocupado

```
    FINSI
```

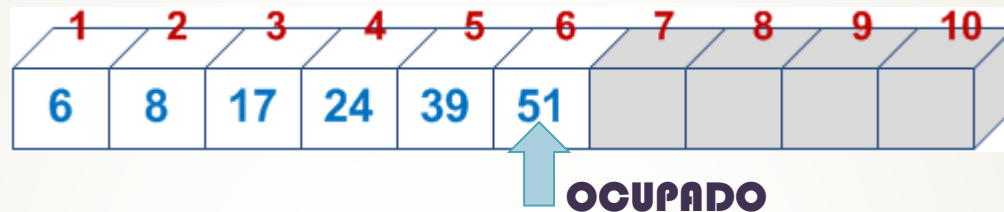
```
FIN
```

Actualización. Borrar (1)

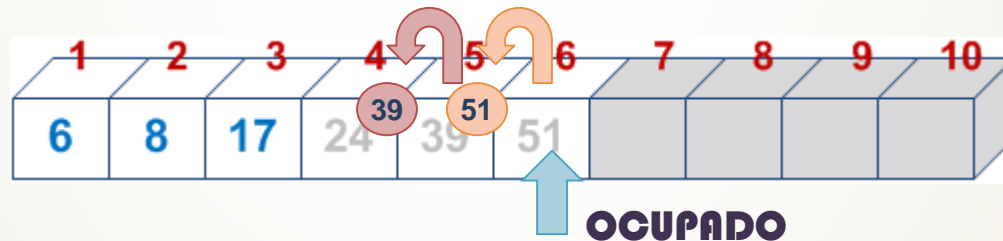
- La operación *borrar* elimina un dato específico del arreglo.
- Al borrar un dato puede ser necesario desplazar los elementos del arreglo (sobreescribiéndose el valor a eliminar).
- Al borrar un dato debe actualizarse la variable ocupado.

Actualización. Borrar (2)

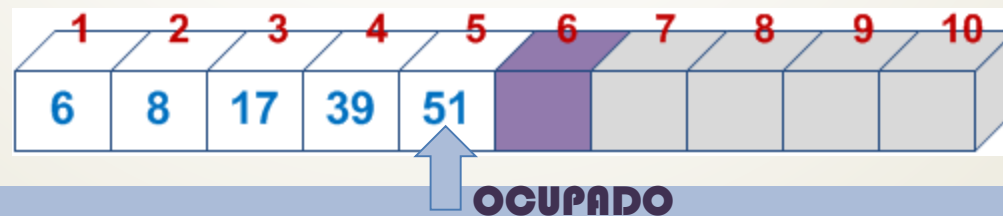
Antes de eliminar el valor 24



Desplazamiento de elementos (hacia la posición de eliminación)



Después de eliminar el valor 24



Actualización. Borrar (3)

Identificación
de la posición
de eliminación

Desplazamiento
de datos
(sobreescripción)

```

...
encontrado ← FALSO
i ← 1
MIENTRAS (i ≤ ocupado) Y NO encontrado HACER
    SI borrado = num[i] ENTONCES
        encontrado ← VERDADERO
    SINO
        i ← i + 1
    FIN SI
FIN MIENTRAS
SI encontrado = VERDADERO ENTONCES
    MIENTRAS i < ocupado HACER
        num[i] ← num[i + 1]
        i ← i + 1
    FIN MIENTRAS
    ocupado ← ocupado - 1
SINO
    ESCRIBIR 'EL ELEMENTO NO EXISTE'
FIN SI

```

Actualización de ocupado

Búsqueda. Secuencial (1)

- La búsqueda secuencial explora secuencialmente los elementos de un arreglo, comparando cada uno con el criterio de búsqueda.
- El arreglo se recorre hasta encontrar el dato buscado o hasta leer completamente el arreglo.

Búsqueda. Secuencial (2)

PROCEDIMIENTO busqueda_sec(E num:vector, E ocup: entero, E buscado:entero)

VARIABLES

i:entero

encontrado:lógico

INICIO

encontrado ← FALSO

i ← 1

MIENTRAS (i ≤ ocup) Y NO encontrado HACER

SI buscado = num[i] ENTONCES

encontrado ← VERDADERO

SINO

i ← i + 1

FIN_SI

FIN_MIENTRAS

SI encontrado = VERDADERO ENTONCES

escribir 'El dato buscado ocupa la posición ', i

SINO

escribir 'Valor no encontrado'

FIN_SI

FIN

Recorrido
del arreglo

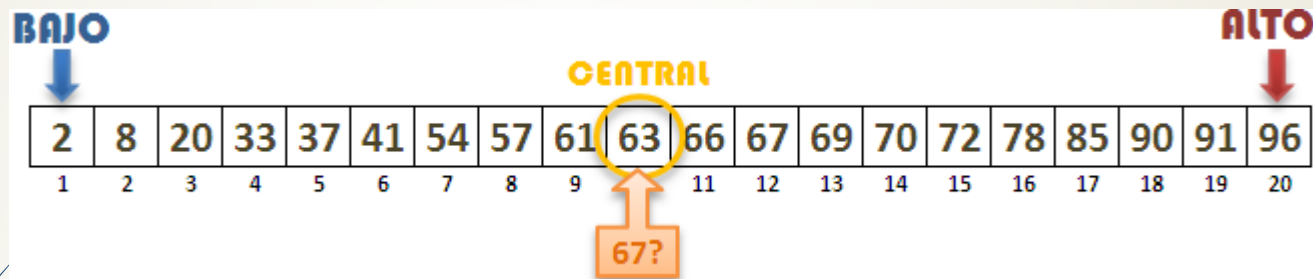
Detección del
elemento buscado

Búsqueda. Binaria (1)

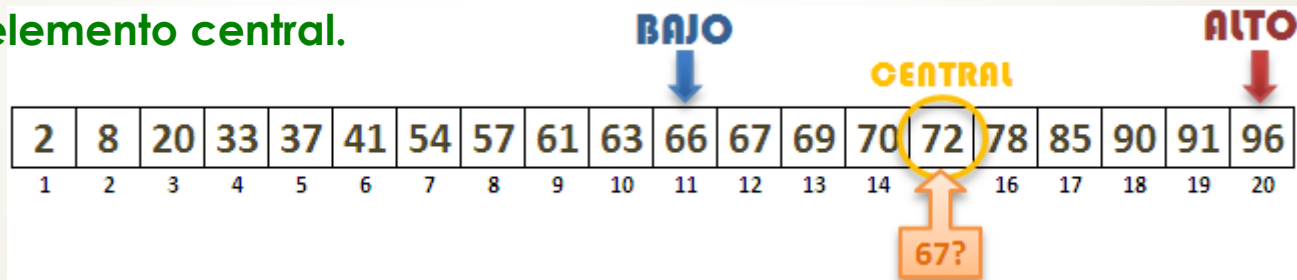
- La búsqueda secuencial resulta ineficiente para grandes volúmenes de datos.
- La búsqueda binaria inicia en el elemento central del arreglo, si éste es el buscado, finaliza; sino se determina si el dato está en la primera o segunda mitad del arreglo. El proceso se reinicia utilizando el elemento central del subarreglo.
- Se aplica sólo sobre vectores ordenados.

Búsqueda. Binaria (2)

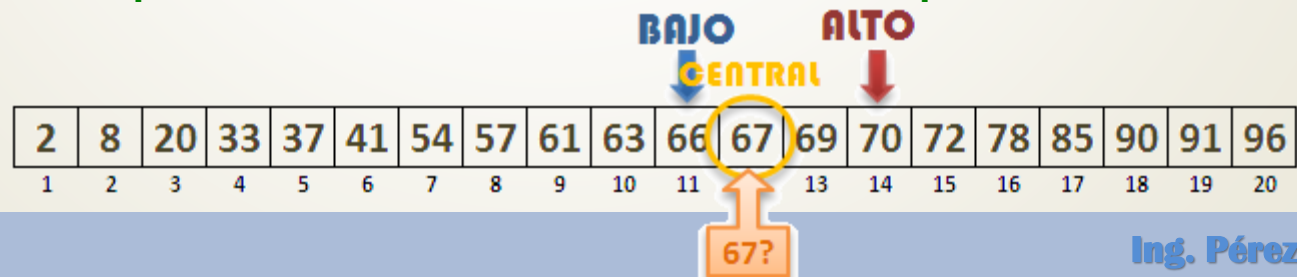
CASO POSITIVO: Se define el intervalo de búsqueda y el elemento central.



Si el dato buscado no coincide con el central, se recalculan el intervalo y el elemento central.

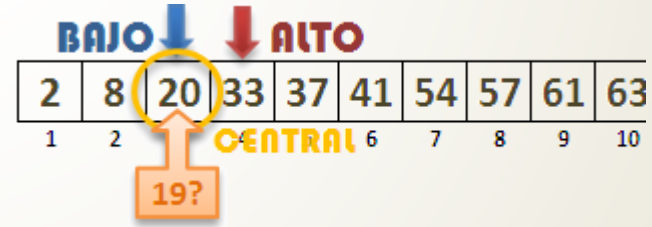
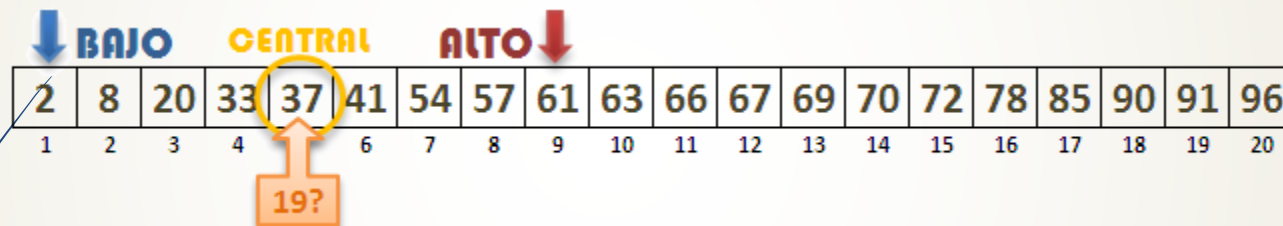
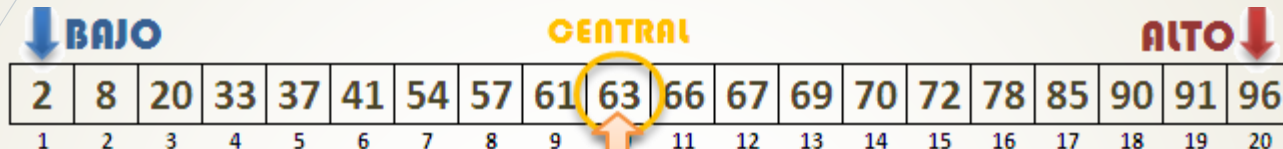


Cuando se produce coincidencia, finaliza la búsqueda.



Búsqueda. Binaria (3)

CASO NEGATIVO



Si el dato no pertenece al arreglo, los índices bajo y alto se cruzan.

Búsqueda. Binaria (4)

FUNCIÓN `busqueda_bin(E num:vector, E ocup:entero, E buscado:entero):lógico`

VARIABLES

`alto,bajo,central:entero`

`encontrado:lógico`

INICIO

`bajo ← 1`

`alto ← ocup`

Intervalo inicial de búsqueda

`encontrado ← FALSO`

MIENTRAS NO encontrado Y (`bajo ≤ alto`) HACER

`central ← (bajo + alto) div 2`

Determinación del elemento central

SI `buscado = num[central]` ENTONCES

`encontrado ← VERDADERO`

SINO

SI `buscado < num[central]` ENTONCES

`alto ← central - 1`

SINO

`bajo ← central + 1`

FIN_SI

FIN_SI

FIN_MIENTRAS

`busqueda_bin ← encontrado`

*Actualización
del intervalo de
búsqueda*

FIN

Bibliografía

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- López Román, Leobardo. Programación estructurada y orientada a objetos. Alfaomega. 2011.
- De Giusti *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta, 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Joyanes Aguilar, Luis. Programación en Turbo Pascal. Mc Graw Hill. 1990.