DESARROLLO WEB EN ENTORNO SERVIDOR

CAPÍTULO 3:

Programación basada en lenguajes de marcas con código embebido

Marcos López Sanz
Juan Manuel Vara Mesa
Jenifer Verde Marín
Diana Marcela Sánchez Fúquene
Jesús Javier Jiménez Hernández
Valeria de Castro Martínez

Ra-Ma®

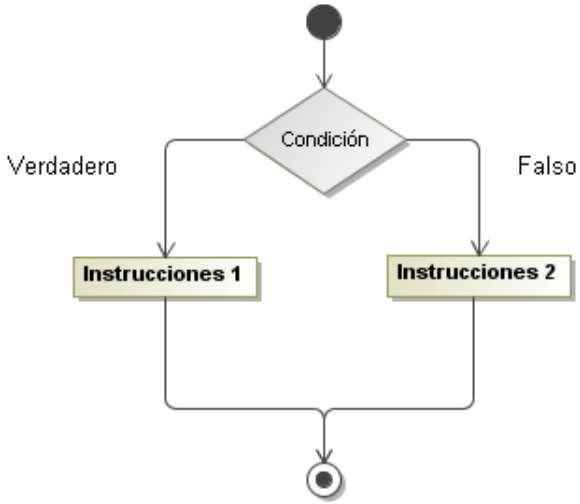
Sentencias condicionales

- Son estructuras de control que permiten decidir el flujo de ejecución de un programa, es decir, el orden en el que las instrucciones de un programa se van ejecutar.

- Tipos:
  - Sentencias if.
  - Sentencias switch.

Sentencias If

- Este tipo de estructuras de control condicionales definen dos flujos de ejecución dependiendo de si se cumple o no la condición establecida por el programador.



Sentencias If

- La sintaxis es la siguiente:
  - Sentencia If.
    - **PHP y JSP:**

```
if (condición){
  instrucciones;
}
```

    - **ASP:**

```
if (condición) then
  instrucciones
end if
```

Sentencias If

- Sentencia If-else.
  - **PHP y JSP:**

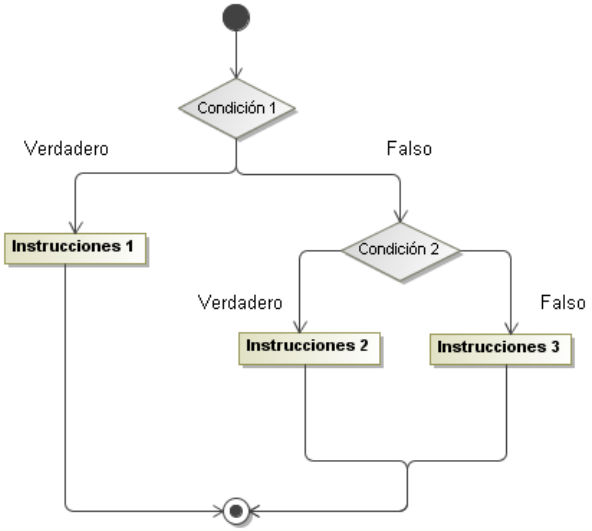
```
if (condición){
   instrucciones1;
}else{
   instrucciones2;
}
```

  - **ASP:**

```
if (condición) then
   instrucciones1
else
   instrucciones2
end if
```

Sentencias If

- Los if anidados permiten evaluar varias condiciones previas antes de ejecutar las instrucciones correspondientes.



Sentencias If

- La sintaxis es la siguiente:
  - **PHP:**

```
if (condición1){
  instrucciones1;
}elseif (condición2){
  instrucciones2;
}else{
  instrucciones3;
}
```

  - **PHP y JSP:**

```
if (condición1){
  instrucciones1;
}else if (condición2){
  instrucciones2;
}else{
  instrucciones3;
}
```

Sentencias If

- **ASP:**
  - ```
    if (condición1) then
      instrucciones1
    elseif (condición2) then
      instrucciones2
    else
      instrucciones3
    end if
    ```
  - ```
    if (condición1) then
      instrucciones1
    else
      if (condición2) then
        instrucciones2
      else
        instrucciones3
      end if
    end if
    ```

Sentencias Switch

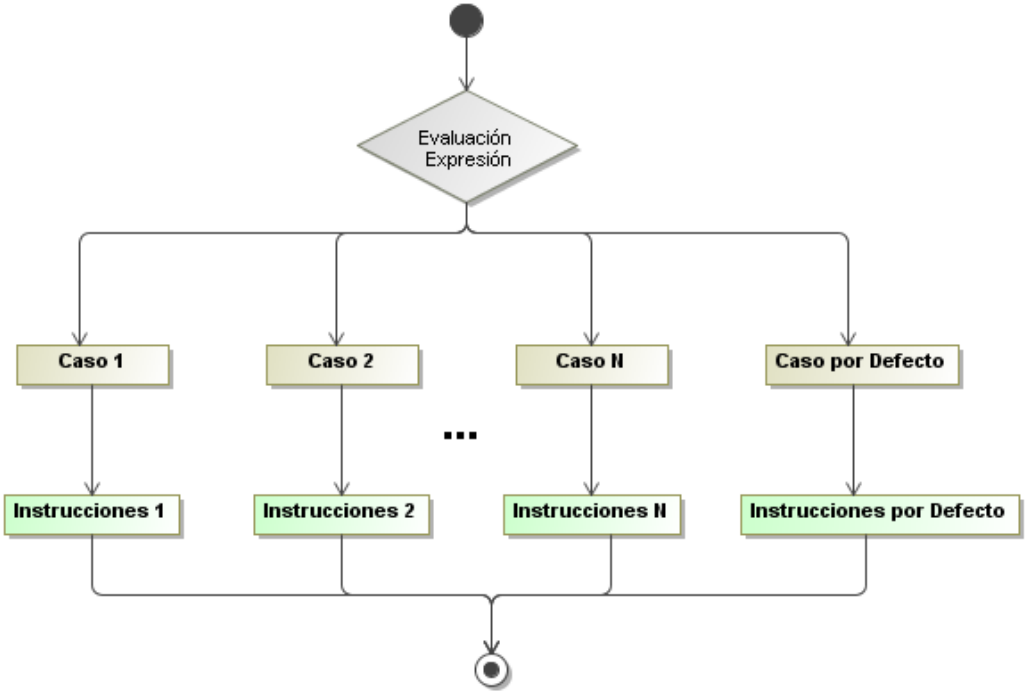
- Estas sentencias se usan cuando dependiendo del valor que toma una variable o expresión, se necesita que se ejecute un conjunto de instrucciones distintas para cada uno de los valores que pueda tomar.

Sentencias Switch

- Comportamiento:
  1. Se calcula el valor de la expresión.
  2. Se compara dicho valor con cada uno de los casos.
     1) Si coincide con un caso se ejecutan las instrucciones contenidas dentro del mismo.
     2) Si no coincide con ningún caso se ejecutan las instrucciones definidas en el caso por defecto si es que está definido (el caso por defecto es opcional).

Sentencias Switch

- Comportamiento (continuación):



Sentencias Switch

- Sintaxis:
  - **PHP y JSP:**

```
switch (expresión){
  case valor1: instrucciones1; break;
  case valor2: instrucciones2; break;
  ...
  case valorN: instruccionesN; break;
  [default: instruccionesN+1;]
}
```

Sentencias Switch

- Sintaxis (continuación):
  - **ASP:**

```
select case (expresión)
  case valor1
    instrucciones1
  case valor2
    instrucciones2
  ...
  case valorN
    instruccionesN
  [case else
   instruccionesN+1]
end select
```

Bucles

- Este tipo de sentencias se utilizan para ejecutar de forma reiterativa una instrucción o grupo de instrucciones.
- Tipos:
  - While o Do While…Loop.
  - Do -While o Do Loop…While.
  - Do Until…Loop.
  - Do Loop…Until.
  - For o For…Next.
  - Foreach.

While o Do While…Loop

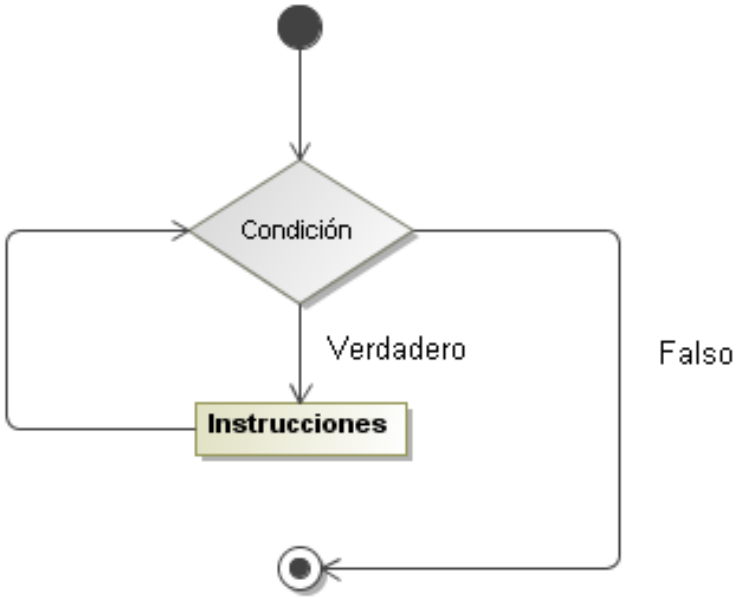
- Este tipo de estructuras permiten ejecutar un número indeterminado de veces una instrucción o grupo de instrucciones, mientras se cumpla la condición.

While o Do While…Loop

- Comportamiento:
  - En cada iteración del bucle se evalúa la condición y si esta es verdadera pasan a ejecutarse las instrucciones contenidas en el cuerpo del bucle. El bucle termina cuando el resultado de evaluar la condición es falso, es decir, cuando la condición ha dejado de cumplirse.

While o Do While…Loop

- Comportamiento (continuación):



While o Do While…Loop

- Sintaxis:
  - **PHP y JSP:**

```
while (condición){
  instrucciones;
}
```

  - **ASP:**
    -
```
do while (condición)
  instrucciones
loop
```
    -
```
while (condición)
  instrucciones
wend
```

While o Do While…Loop

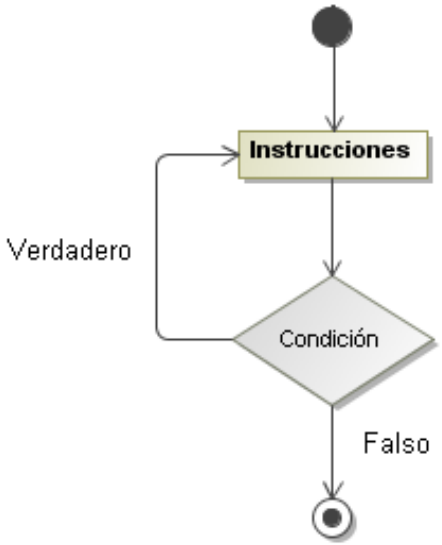
- Sintaxis:
  - ASP ofrece dos formas distintas de definir este bucle:
    1) La primera es la más utilizada y se refiere a la forma nueva de definir este bucle.
    2) La segunda es un vestigio de los inicios de Basic. Actualmente los intérpretes soportan también la segunda forma de definir este bucle, para los programadores reticentes a utilizar la nueva forma, pero es posible que en el futuro los intérpretes dejen de contemplarlo.

Do-While o Do Loop…While

- Este bucle se ejecuta un número indeterminado de veces hasta que el resultado de evaluar la condición es falsa.
- Características:
  - Siempre se ejecuta al menos una vez se cumpla o no la condición.
  - la evaluación de la condición se realiza al final de cada iteración y no al principio.

Do-While o Do Loop…While

- Comportamiento:



Do-While o Do Loop…While

- Sintaxis:
  - **PHP y JSP:**

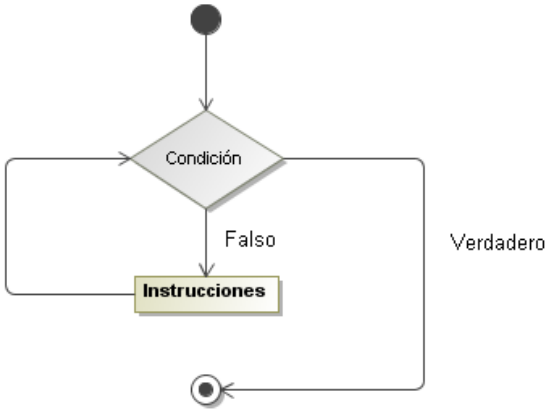
```
do{
  instrucciones;
}while (condición);
```

  - **ASP:**

```
do
  instrucciones
loop while (condición)
```

Bucle Do Until…Loop

- Comportamiento:
  - La condición se evalúa al principio de cada iteración.
    - Si la condición no se cumple se ejecuta las instrucciones que contiene.
    - Si la condición se cumple (es verdadera) el bucle finaliza.



Bucle Do Until…Loop

- Sintaxis:
  - **ASP:**

```
do until (condición)
  instrucciones
loop
```

Bucle Do Loop…Until

- Comportamiento:
  - Se ejecutan las instrucciones hasta que la condición se cumple.
  - La condición se evalúa al final de cada iteración del bucle, lo que significa que este bucle se ejecuta al menos una vez.

Bucle Do Loop…Until

- Sintaxis:
  - **ASP:**

```
do
  instrucciones
loop until (condición)
```

Bucle For o For…Next

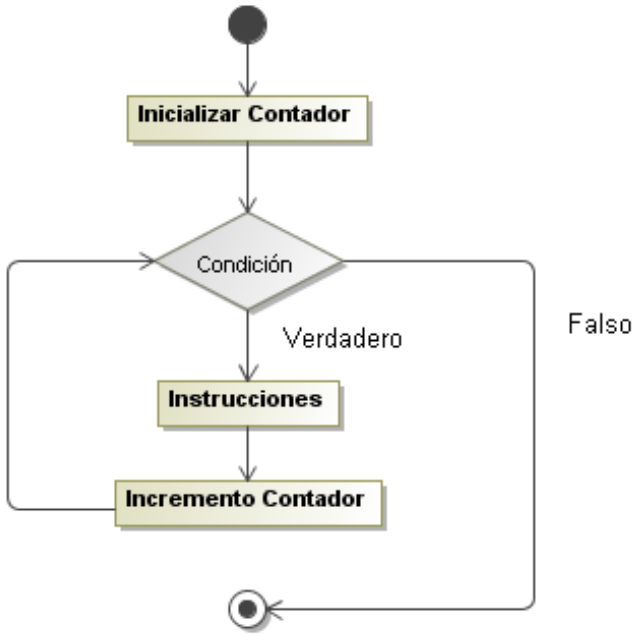
- A diferencia del resto de bucles este tipo de bucles se ejecuta un número determinado de veces. Al igual que el resto de bucles permite ejecutar un conjunto de instrucciones de forma repetitiva.

Bucle For o For…Next

- Comportamiento:
  1. Se inicializa el contador, que controla el número de veces que se ejecuta el bucle.
  2. Se evalúa la condición :
     - Si es verdadera se ejecuta el contenido del bucle y se actualiza el contador.
     - Si es falsa, el bucle finaliza siguiendo con la ejecución de las instrucciones siguientes al bucle.

Bucle For o For…Next

- Comportamiento (continuación):



Bucle For o For…Next

- Sintaxis:
  - **PHP y JSP:**

```
for([contador=valorInicial];[condición];[incremento]){
  instrucciones;
}
```

  - **ASP:**

```
for contador=valorInicial to valorFinal [step incremento]
  instrucciones
next
```

Bucle Foreach

- Proporciona una forma sencilla de iterar sobre los elementos de un array o matrices.
- Sintaxis:
  - **PHP:**
    - ```
      foreach(variableArray as variableValor){
       instrucciones;
      }
      ```
    - ```
      foreach(variableArray as variableIndice => variableValor){
       instrucciones;
      }
      ```

Bucle Foreach

- Sintaxis (continuación):
  - **ASP:**

```
for each variableValor in variableArray
  instrucciones
next
```

Sentencia Break

- Se utiliza para interrumpir la ejecución de un bucle.

- Sintaxis:
  - **ASP:**
    - For…Next: `exit for`
    - Do While…Loop, Do Until…Loop, Do…Loop While o Do…Loop Until: `exit do`
    - While…Wend: `exit while`

Sentencia Break

- Sintaxis (continuación):
  - **PHP:** `break [número];`
  - **JSP:** `break [etiqueta];`

- Ejemplo (JSP):

```
etiqueta1:
while(condición1){
  instrucciones;
  if(condición2){
    break etiqueta1;
  }
}
```

Sentencia Continue

- Esta sentencia se utiliza para saltar el resto de la iteración de un bucle y continuar con la evaluación de la condición y la siguiente iteración.

- Sintaxis:
  - **PHP:** `continue [número];`
  - **JSP:** `continue [etiqueta];`

Arrays

- Los arrays o matrices son estructuras que permiten el almacenamiento de un conjunto de datos.
- Definición: un array o matriz es un conjunto ordenado de elementos identificados por un índice (la posición del elemento dentro de esta colección ordenada), de modo que en cada posición marcada por un índice el array contiene un valor.
- La longitud del array se modifica de forma dinámica siempre que añadimos un nuevo elemento.

Arrays

- Sintaxis (declaración):
  - **PHP:** `nombreVariable=array(clave => valor,…);`
  - **ASP:** `Dim nombreVariable(tamaño)`
  - **JSP:**
    - `tipo[] nombreVariable = new tipo [tamaño];`
    - `tipo[] nombreVariable = {valor1,valor2,…,valorN};`

Arrays

- Sintaxis (inicialización):
  - **PHP:** `array(clave => array(),…);`
  - **ASP:** `Dim nombreVariable(n° filas,n° columnas)`
  - **JSP:**
    - `tipo[][] nombreVariable = new tipo [n°filas][n°columnas];`
    - `tipo [][] nombreVariable = {valor1.1, valor1.2, …, valor1.N},…,{valorN.1,valorN.2,…,valorN.N}};`

Arrays

- Ejemplo (arrays):
  - **PHP:** `$miarray=array(0=>2,1=>4);`
  - **ASP:**

```
Dim miarray(1)
miarray(0)=2
miarray(1)=4
```

  - **JSP:**
    -
```
int[] miarray= new int [2];
miarray[0]=2;
miarray[1]=4;
```
    -
```
int[] miarray= {2,4};
```

Arrays

- Ejemplo (matrices):
  - **PHP:**
```
mimatriz(0 =>array(0=>2,1=>4),1 =>array(0=>1,1=>3));
```
  - **ASP:**
```
Dim mimatriz(1,1)
 mimatriz(0)(0)=2
 mimatriz(0)(1)=4
 mimatriz(1)(0)=1
 mimatriz(1)(1)=3
```
  - **JSP:**
    -
```
int[][] mimatriz = new int [2][2];
 mimatriz[0][0]=2;
 mimatriz[0][1]=4;
 mimatriz[1][0]=1;
 mimatriz[1][1]=3;
```
    -
```
int [][] mimatriz = {{2,4},{1,3}};
```

Arrays

- Sintaxis (acceso a valores de una array):
  - **PHP:** `$miarray[0]`
  - **JSP:** `miarray[0]`
  - **ASP:** `miarray(0)`

Arrays

- Sintaxis (recorrer un array):
  - **PHP:**

```
foreach($miarray as $valor){
  echo $valor;
}
```

  - **ASP:**

```
for each valor in miarray
  response.write(valor)
next
```

  - **JSP:**

```
for(i=0;i<miarray.length;i++){
  out.println(miarray[i]);
}
```

Algoritmos de búsqueda

- Son algoritmos que sirven para buscar un valor dentro de un array.

  - Búsqueda secuencial:
    - Consiste en comparar cada elemento del array con el elemento que pretendemos buscar.
    - Termina cuando encontramos el elemento que buscábamos o cuando llegamos al final del array.
    - Si encuentra el elemento se devuelve la posición del array y en el caso contrario se devuelve un código de error.

Algoritmos de búsqueda

- Búsqueda binaria:
  - Se utiliza en arrays ordenados.
  - Se compara el elemento que divide el array en dos mitades, es decir el del centro, con el elemento que buscamos, si no coinciden se determina en que mitad del array se encuentra para buscar dentro de ella, descartando la otra mitad.

Algoritmos de búsqueda

- Búsqueda binaria (continuación):

A[0]	A[1]	A[2]	A[3]
24	31	36	80

Se desea encontrar el elemento 36

1. Se calcula el elemento central del *array:*

$$centro = \frac{indice_{Bajo} + indice_{Alto}}{2} = \frac{0+3}{2} = 1$$

El elemento central es a[1]=31

2. Se compara con el elemento buscado:

Como a[1]=31 < 36 se descarta la primera mitad del *array* y se continua la búsqueda en la segunda mitad

36	80

3. Se calcula el elemento central del *array* de la sublista derecha:

$$centro = \frac{indice_{Bajo} + indice_{Alto}}{2} = \frac{2+3}{2} = 2$$

El elemento central es a[2]=36

4. a[2]=36 → Hemos encontrado el elemento

Nota: Las imágenes referentes a los algoritmos de ordenación y búsqueda en arrays están basadas en (Joyanes & Zahonero, 2004).

Algoritmos de búsqueda

- Búsqueda binaria (continuación):
  - **PHP:**

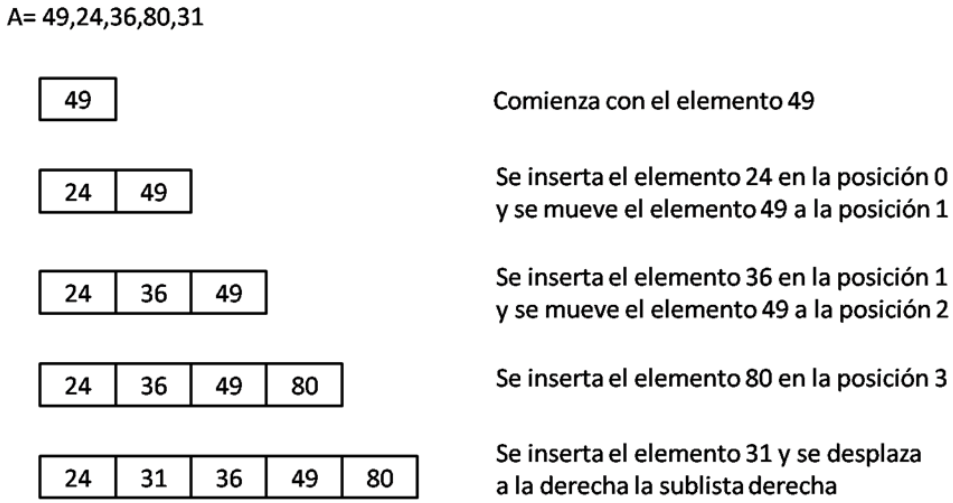
```
array_search(variable,valor);
$a=array(0=> 'blue', 1=> 'green');
$b=array_search($a,'green');
```

  - **JSP:**

```
binarySearch(variable,valor);
String[] a={'blue','green'};
int b=binarySearch(a,'green');
```

Algoritmos de ordenación

- Son algoritmos que sirven para ordenar un array.
  - Inserción directa: se basa en la inserción de los elementos de manera ordenada en la posición que les corresponde.

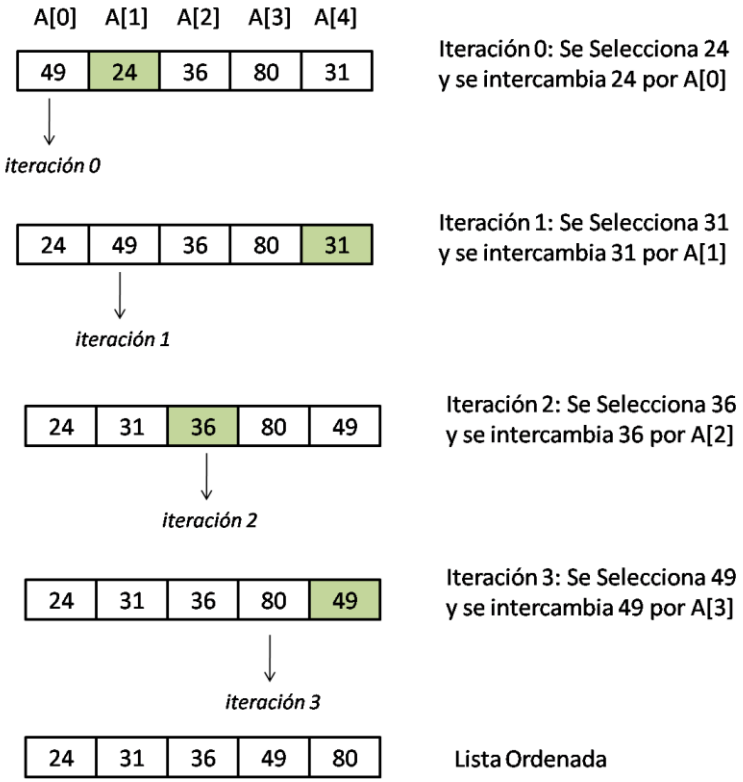


Algoritmos de ordenación

- Selección directa: selecciona el elemento más pequeño de todo el array y se intercambia con el primer elemento. Se busca el siguiente más pequeño y se intercambia por el segundo y así sucesivamente hasta que quede sólo un elemento y por lo tanto la lista esté ordenada.

Algoritmos de ordenación

- Selección directa (continuación):

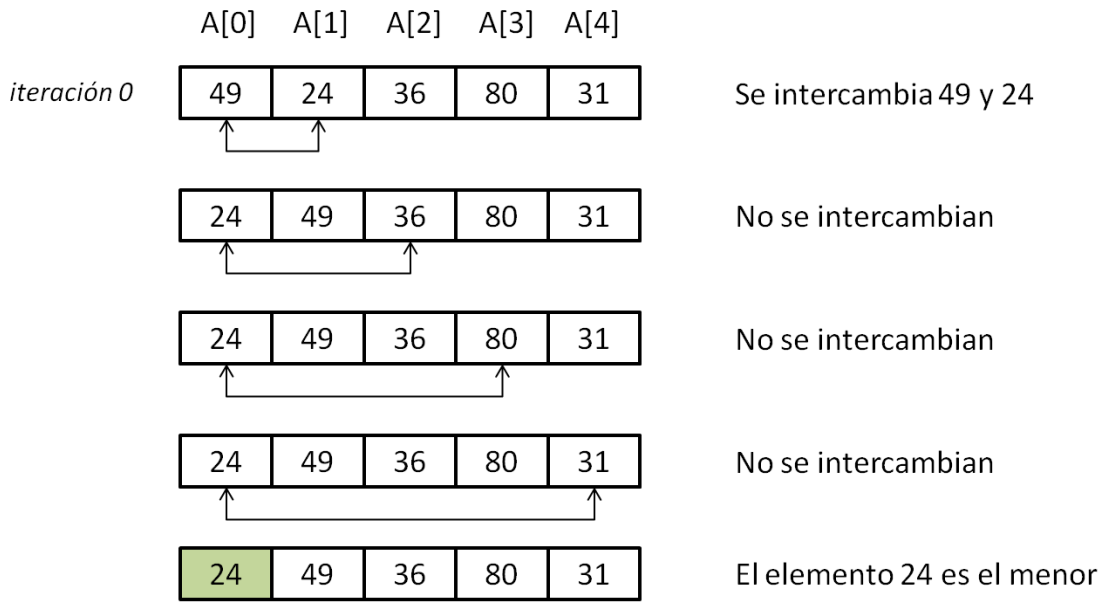


Algoritmos de ordenación

- Intercambio:
  - Ordena un array de manera ascendente.
  - Se basa en la lectura sucesiva del array comparando el elemento de la posición 0 de la lista con el resto, si el elemento de la posición 0 del array es mayor que el otro elemento con el que se le compara se intercambian. Una vez que hemos comparado el primer elemento con todos, se hace lo mismo con el resto de posiciones hasta conseguir ordenar el array.

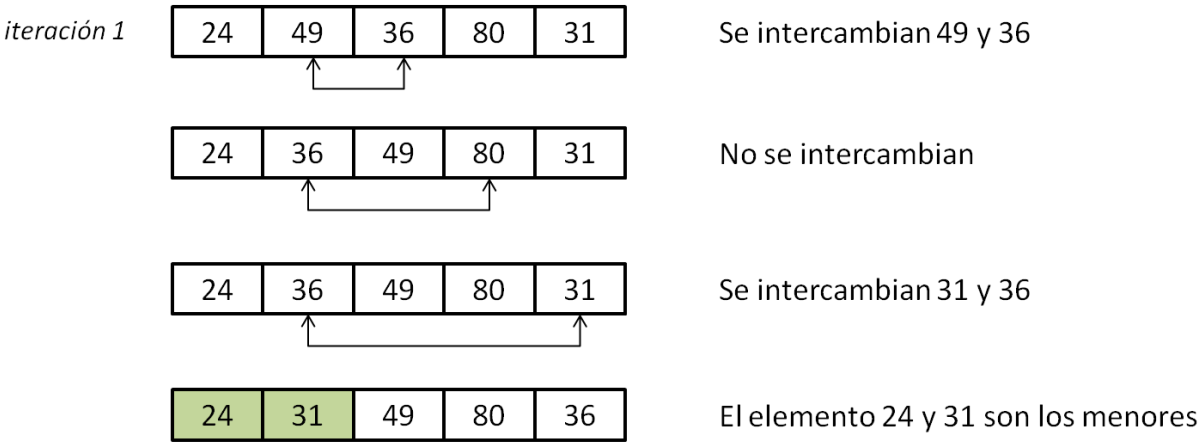
Algoritmos de ordenación

- Intercambio (continuación):



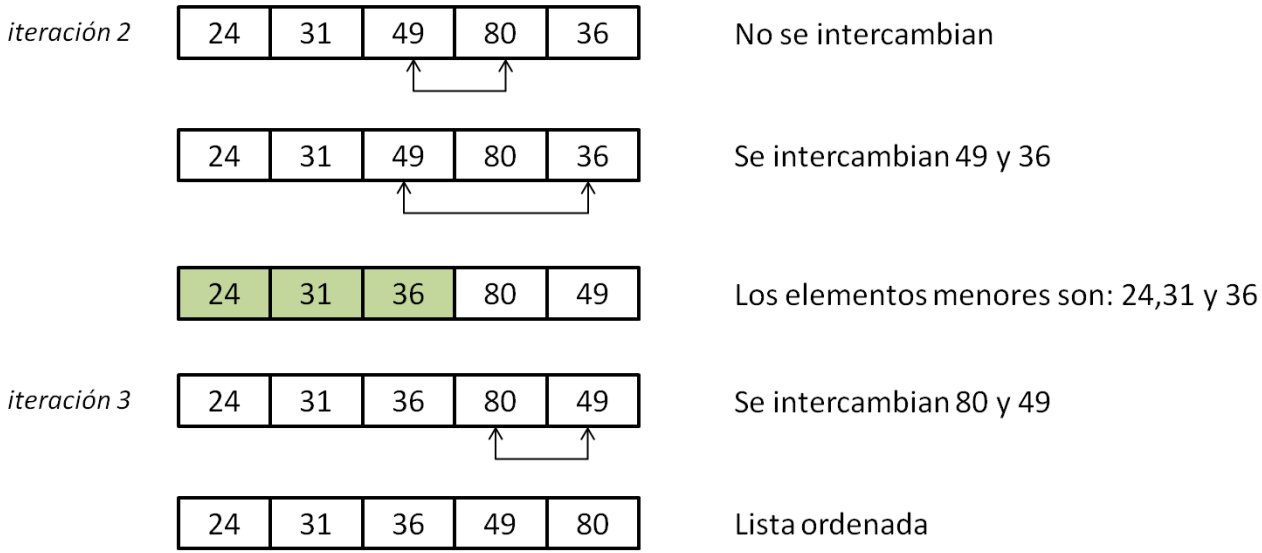
Algoritmos de ordenación

- Intercambio (continuación):



Algoritmos de ordenación

- Intercambio (continuación):

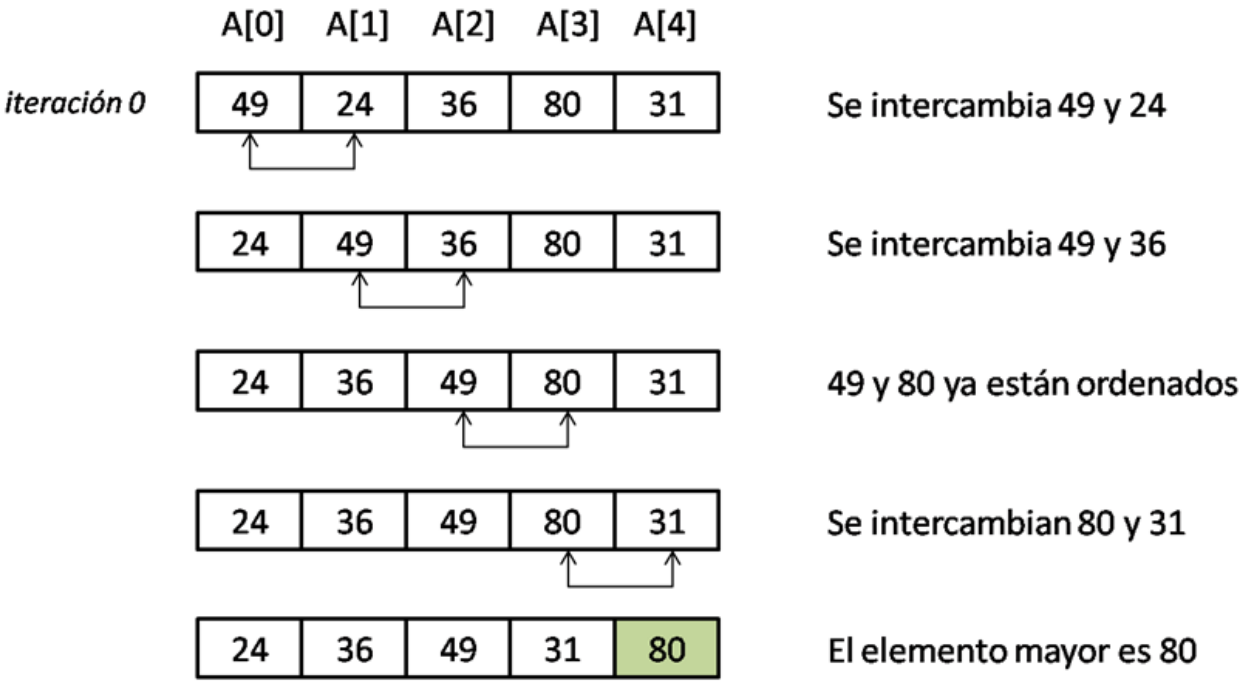


Algoritmos de ordenación

- Burbuja: consiste en el intercambio entre pares de elementos adyacentes.
  - Si un elemento no está ordenado respecto al siguiente se intercambian la posición.
  - Se realizan sucesivas iteraciones hasta que el array queda ordenado.

Algoritmos de ordenación

- Burbuja (continuación):



Algoritmos de ordenación

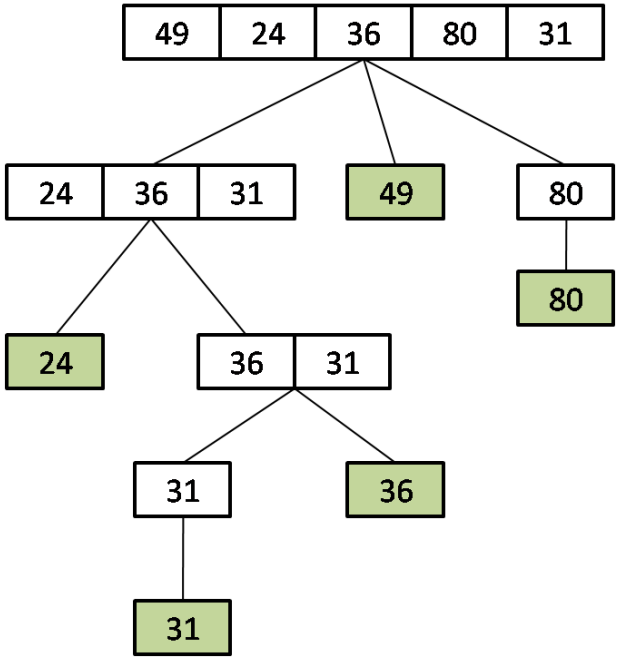
- Burbuja (continuación):

Algoritmos de ordenación

- QuickSort: divide el array en dos partes separadas por elemento central, llamado *pivote*.
  1. Elegir el pivote (ej. el primer elemento).
  2. Dividir el array de tal manera que los elementos menores que el pivote formen parte de la sublista izquierda y los que sean mayores se encuentren en la sublista derecha.
  3. Ordenar cada sublista de forma independiente aplicando este mismo algoritmo.

Algoritmos de ordenación

- QuickSort (continuación):

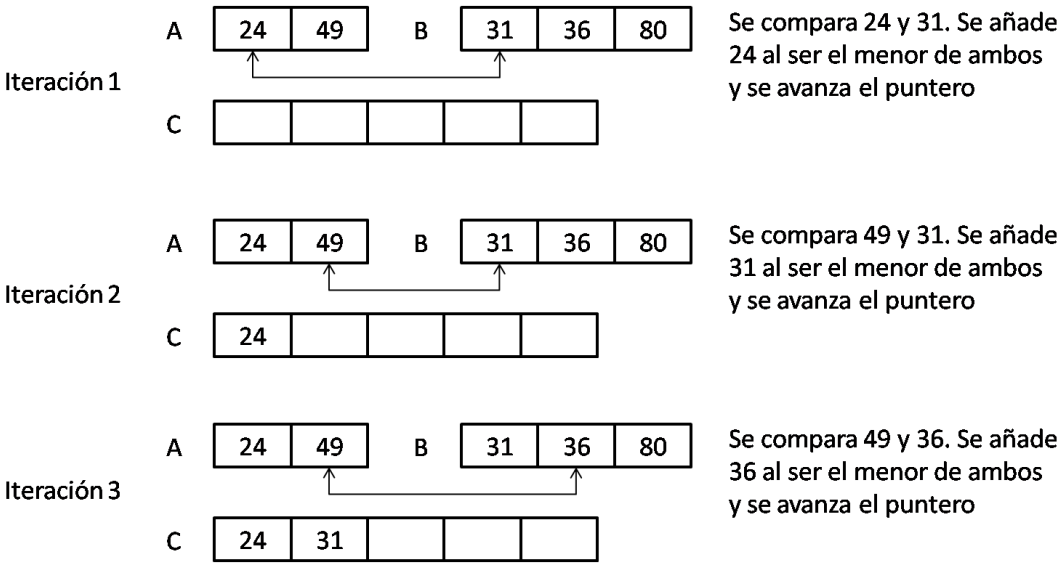


Algoritmos de ordenación

- Mergesort:
  - Parte de dos array ordenados.
  - El objetivo es obtener un array ordenado que contenga la información de ambos.
  - Se utiliza un array nuevo que tenga como tamaño la suma del tamaño de los dos arrays. Después se van comparando los elementos de cada array y se inserta en el nuevo array el menor de los dos.

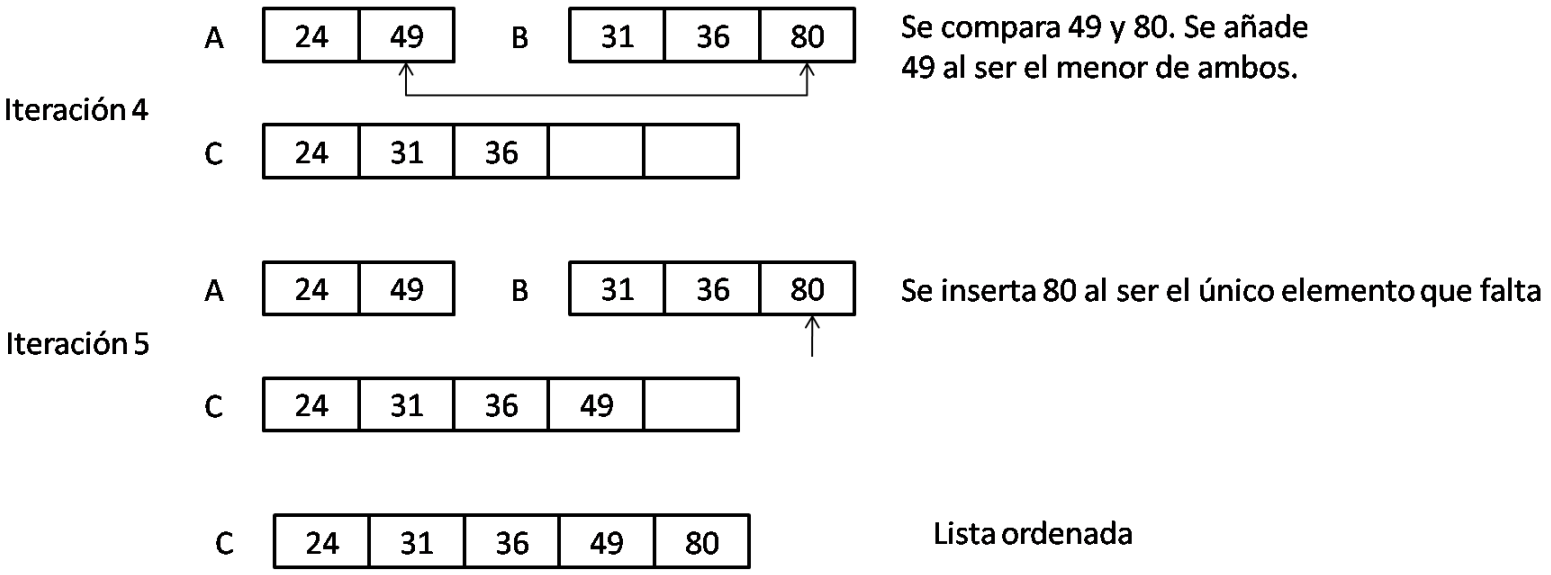
Algoritmos de ordenación

- Mergesort (continuación):



Algoritmos de ordenación

- Mergesort (continuación):



Algoritmos de ordenación

- HeapSort: algoritmo basado en árboles binarios. Para ordenar un array de forma ascendente debemos cumplir la condición de que un nodo padre no puede ser menor a un nodo hijo.
- **PHP:**
  - Ordenar de menor a mayor: `sort(variable);`
  - Ordenar de mayor a menor: `rsort(variable);`
  - Ejemplo:

```
$a=array(0=> 21,1=> 8,2=> 9);
rsort($a);
sort($a);
```

Algoritmos de ordenación

- **JSP:**
  - Ordenar de menor a mayor: `sort(variable);`
  - Ejemplo:

```
String[] a={21,8,9};
sort(a);
```

Algoritmos de inserción

- Se utilizan para añadir nuevos elementos dentro de un array.
  - Si el array no está ordenado basta con añadir el elemento al final del array.
  - Si no esta ordenado hay que recorrer el array para buscar la posición exacta en la que debemos insertar el elemento y desplazar los elementos mayores que estén a la derecha.

Subprogramación

- Un subprograma es un fragmento de código que tiene una funcionalidad específica. Este permite que el código sea modular y lo podamos reutilizar.

- Tipos:
  - Funciones: subprogramas que devuelven un valor como resultado de su ejecución.
  - Procedimientos: son aquellos que se ejecutan sin devolver ningún tipo de valor.

Función

- Sintaxis:
  - **PHP:**

```
function nombre($arg1,$arg2,…){
  instrucciones;
  return $valorDevuelto;
}
```

  - **ASP:**

```
function nombre (arg1,arg2,…)
  instrucciones
  nombre=valordevuelto
end function
```

Función

- Sintaxis (continuación):
  - **JSP:**

```
tipoDevuelto nombre (tipo1 arg1,tipo2 arg2,…){
  instrucciones;
  return valorDevuelto;
}
```

Procedimiento

- Sintaxis:
  - **PHP:**

```
function nombre($arg1,$arg2,…){
  instrucciones;
}
```

  - **ASP:**

```
sub nombre (arg1,arg2,…)
  instrucciones
end sub
```

Procedimiento

- Sintaxis:
  - **JSP:**

```
void nombre (tipo1 arg1,tipo2 arg2,…){
  instrucciones;
}
```

Función/Procedimiento

- Sintaxis (invocar función/procedimiento):
  - **PHP:** `nombre($arg1,$arg2,…);`
  - **ASP:**
    - `nombre (arg1,arg2,…)`
    - Sólo procedimientos: `call nombre (arg1,arg2,…)`
  - **JSP:** `nombre (arg1,arg2,…);`

Formularios

- Son métodos para recolectar información que debe ser rellenada por el usuario.

- Pueden servir para permitir la autentificación de un usuario en una página Web, recoger información que debemos insertar o actualizar en una base de datos, realizar búsquedas de información en el que debamos especificar un criterio de búsqueda, etc.

Método GET/POST

- Son métodos del protocolo HTTP para el intercambio de información entre el cliente y el servidor.
  - El método *GET* envía las variables dentro de la URL de la página.
  - En el método *POST* la información va codificada en el cuerpo de la petición HTTP y por lo tanto viaja oculta.
  - Mientras el método *GET* lo utilizamos para recuperar información, el método *POST* se usa para enviar información a un servidor Web.

Formulario GET

```
<html>
  <head>
    <title>Ejemplo Formularios</title>
  </head>
  <body>
    <h1>Ejemplo de procesado de formularios</h1>
    <FORM ACTION="ejemplo1.php" METHOD="GET">
      Introduzca su nombre:<INPUT TYPE="text" NAME="nombre">
      <BR>
      Introduzca sus apellidos:<INPUT TYPE="text"
      NAME="apellidos"><BR>
      <INPUT TYPE="submit" VALUE="Enviar">
    </FORM>
  </body>
</html>
```

Formulario POST

```
<html>
  <head>
    <title>Ejemplo Formularios</title>
  </head>
  <body>
    <h1>Ejemplo de procesado de formularios</h1>
    <FORM ACTION="ejemplo2.php" METHOD="POST">
      Introduzca su nombre:<INPUT TYPE="text" NAME="nombre">
      <BR>
      Introduzca sus apellidos:<INPUT TYPE="text"
      NAME="apellidos"><BR>
      <INPUT TYPE="submit" VALUE="Enviar">
    </FORM>
  </body>
</html>
```

Recuperación información GET

- GET:
  - **PHP:**
    - ```
      echo $_GET['nombre'];
      echo $_GET['apellidos'];
      ```
    - ```
      print_r($_GET);
      ```
  - **ASP:**

    ```
    response.write(request.QueryString("nombre"))
    response.write(request.QueryString("apellidos"))
    ```
  - **JSP (válido para GET y POST):**

    ```
    out.print(request.getParameter("nombre"));
    out.print(request.getParameter("apellidos"));
    ```

Recuperación información POST

- POST:
  - **PHP:**
    - `echo $_POST['nombre'];`
      `echo $_POST['apellidos'];`
    - `print_r($_POST);`
  - **ASP:**

```
response.write(request.Form("nombre"))
response.write(request.Form("apellidos"))
```

  - **JSP (válido para GET y POST):**

```
out.print(request.getParameter("nombre"));
out.print(request.getParameter("apellidos"));
```