

PROGRAMACION 3

TUDAI

Facultad de Ciencias Exactas - UNICEN

Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];

Estructura de Datos

Array

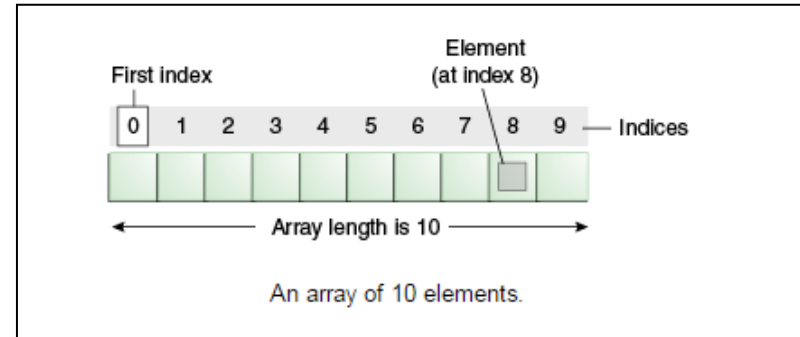
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



Es una secuencia en un **bloque de memoria contiguo**.

Estructura de Datos

Array

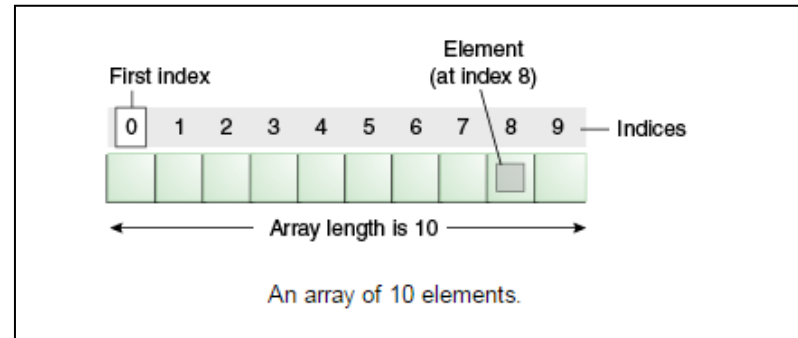
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

int[] head = {2, 6, 8, 7, 1};

Mismo efecto que:

int[] head = new int[5];

head[0] = 2;

head[1] = 6;

head[2] = 8;

....

Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

int[] head = {2, 6, 8, 7, 1}

Mismo efecto que:

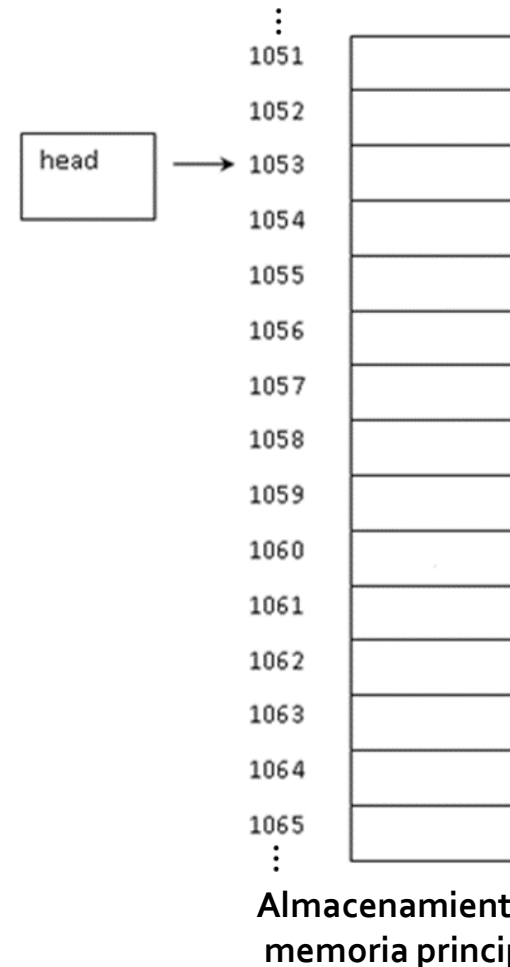
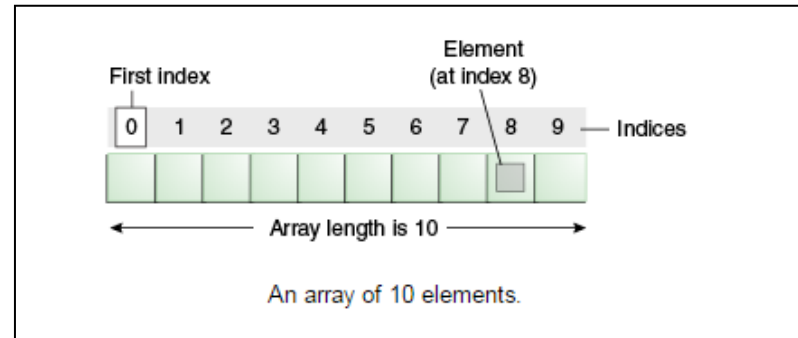
int[] head = new int[5];

head[0] = 2;

head[1] = 6;

head[2] = 8;

....



Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

int[] head = {2, 6, 8, 7, 1};

Mismo efecto que:

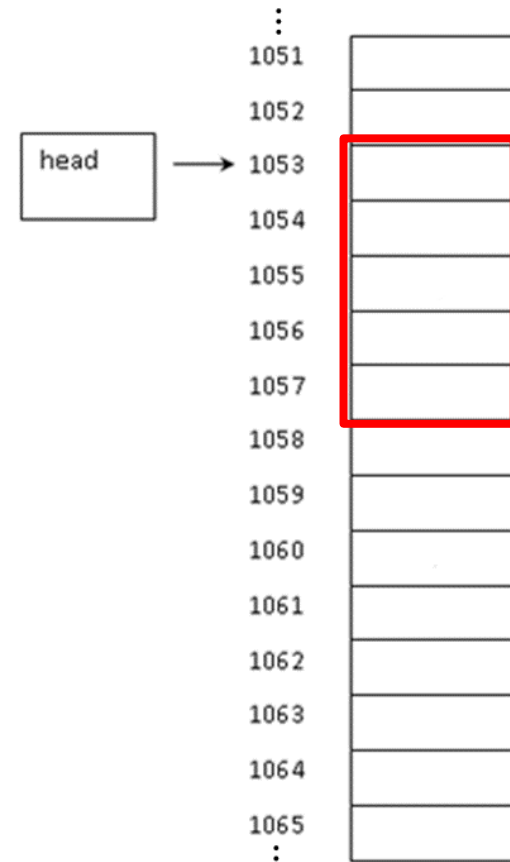
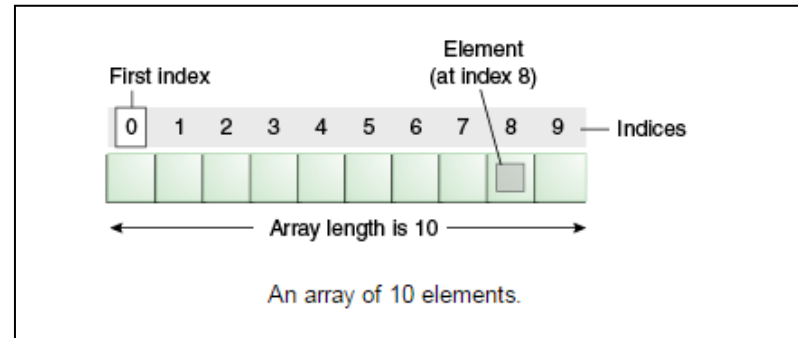
int[] head = new int[5];

head[0] = 2;

head[1] = 6;

head[2] = 8;

....



Almacenamiento en
memoria principal

Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

int[] head = {2, 6, 8, 7, 1};

Mismo efecto que:

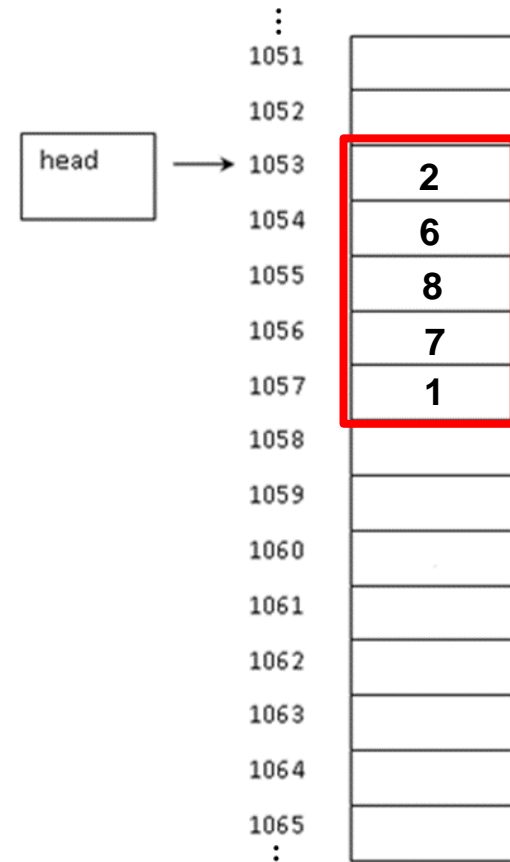
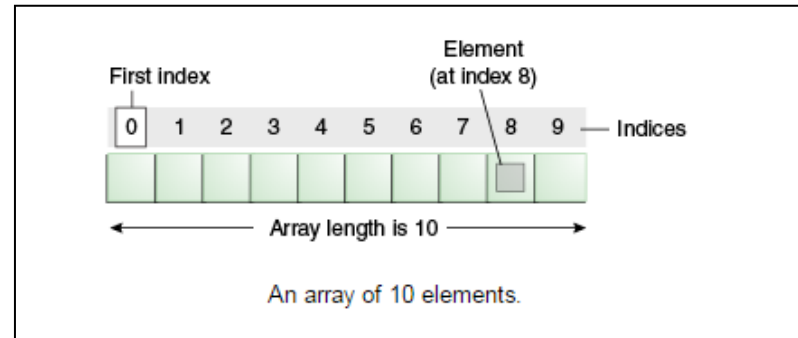
int[] head = new int[5];

head[0] = 2;

head[1] = 6;

head[2] = 8;

....



Almacenamiento en
memoria principal

Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

int[] head = {2, 6, 8, 7, 1};

Mismo efecto que:

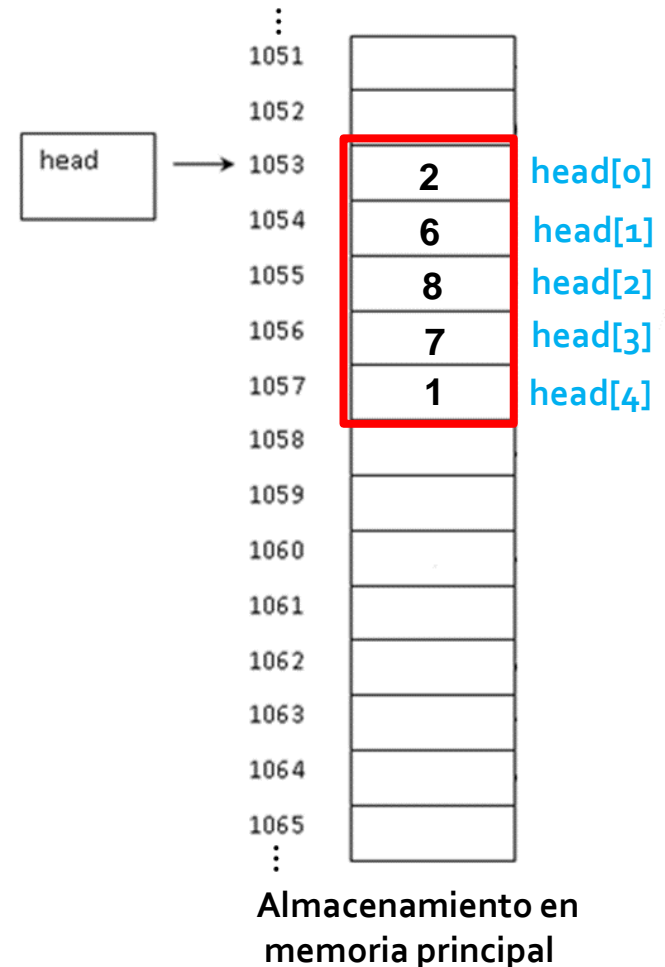
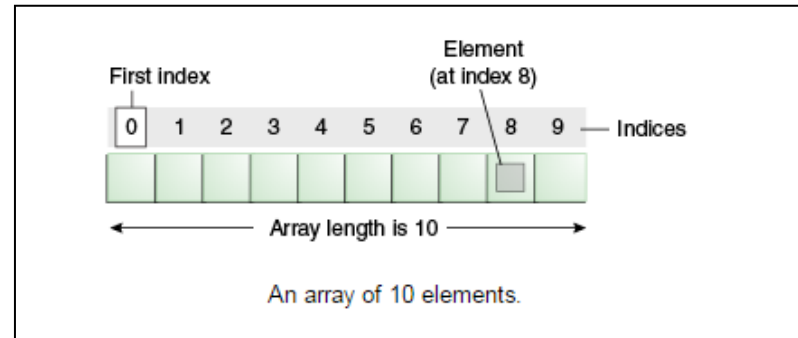
int[] head = new int[5];

head[0] = 2;

head[1] = 6;

head[2] = 8;

....



Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

```
int[ ] unArray;
```

// reserva espacio para 10 nros. enteros

// valor por defecto 0

```
unArray = new int[10];
```

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

```
int[] head = {2, 6, 8, 7, 1}
```

Mismo efecto que:

```
int[] head = new int[5];
```

```
head[0] = 2;
```

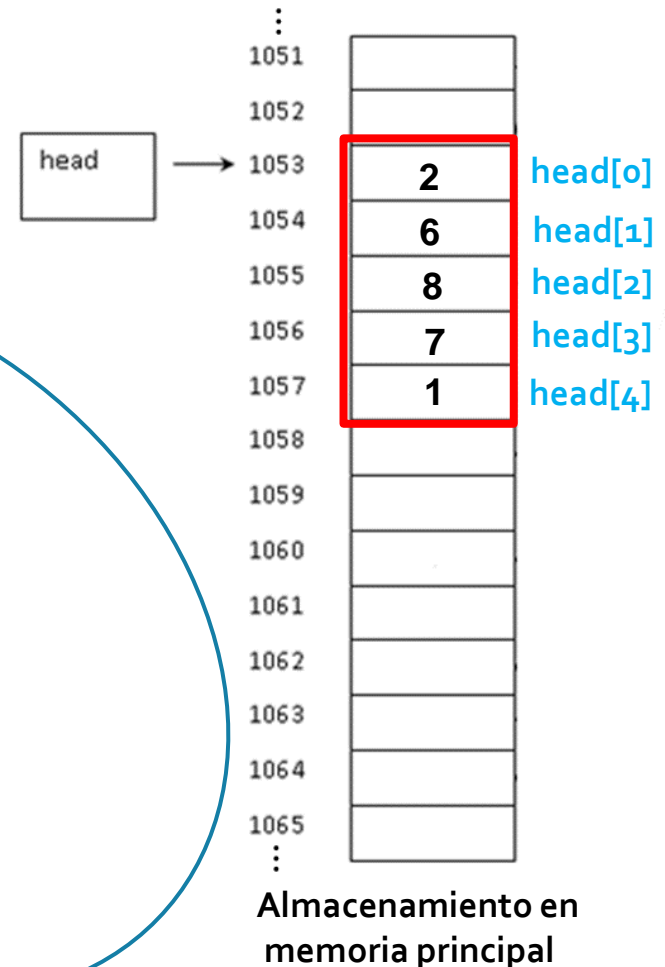
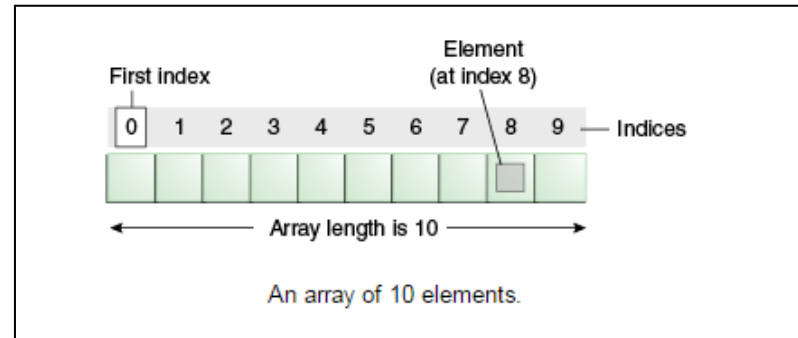
```
head[1] = 6;
```

```
head[2] = 8;
```

....



Cómo hace para buscar el elemento **head[4]** ???



Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

```
int[ ] unArray;
```

// reserva espacio para **10** nros. enteros

// valor por defecto 0

```
unArray = new int[10];
```

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

```
int[] head = {2, 6, 8, 7, 1}
```

Mismo efecto que:

```
int[] head = new int[5];
```

```
head[0] = 2;
```

```
head[1] = 6;
```

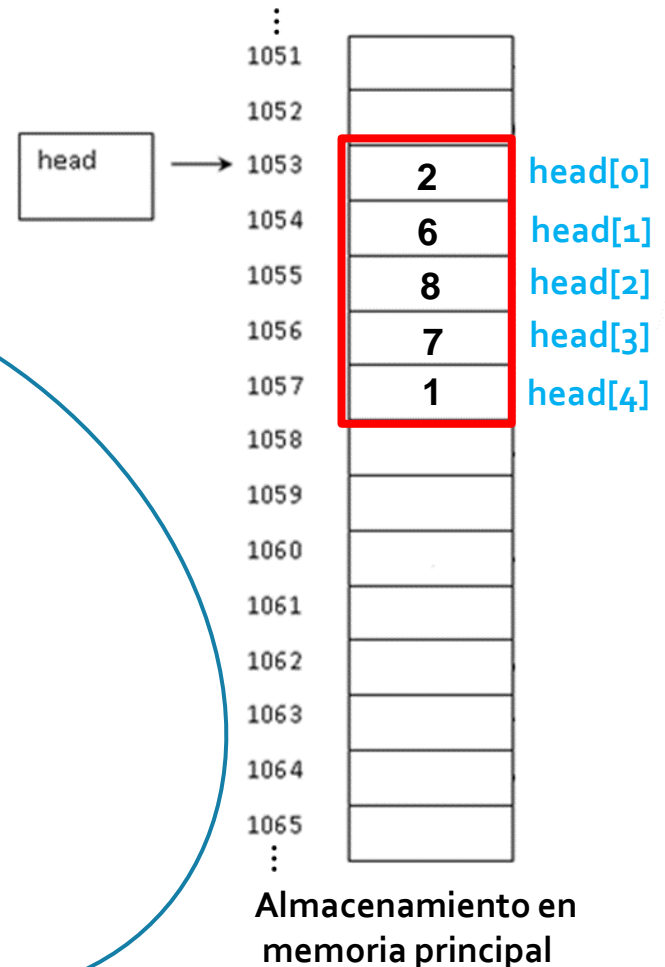
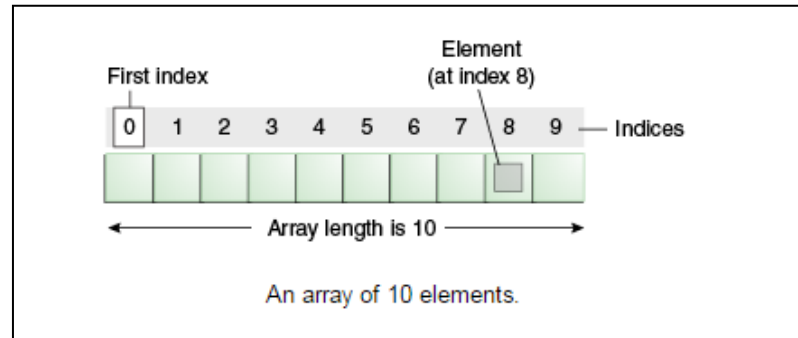
```
head[2] = 8;
```

....



Cómo hace para buscar el elemento **head[4]** ???

Sabe que **head** arranca en la dirección 1053 y esa es la dirección del elemento cero.



Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

```
int[ ] unArray;
```

// reserva espacio para 10 nros. enteros

// valor por defecto 0

```
unArray = new int[10];
```

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

```
int[] head = {2, 6, 8, 7, 1}
```

Mismo efecto que:

```
int[] head = new int[5];
```

```
head[0] = 2;
```

```
head[1] = 6;
```

```
head[2] = 8;
```

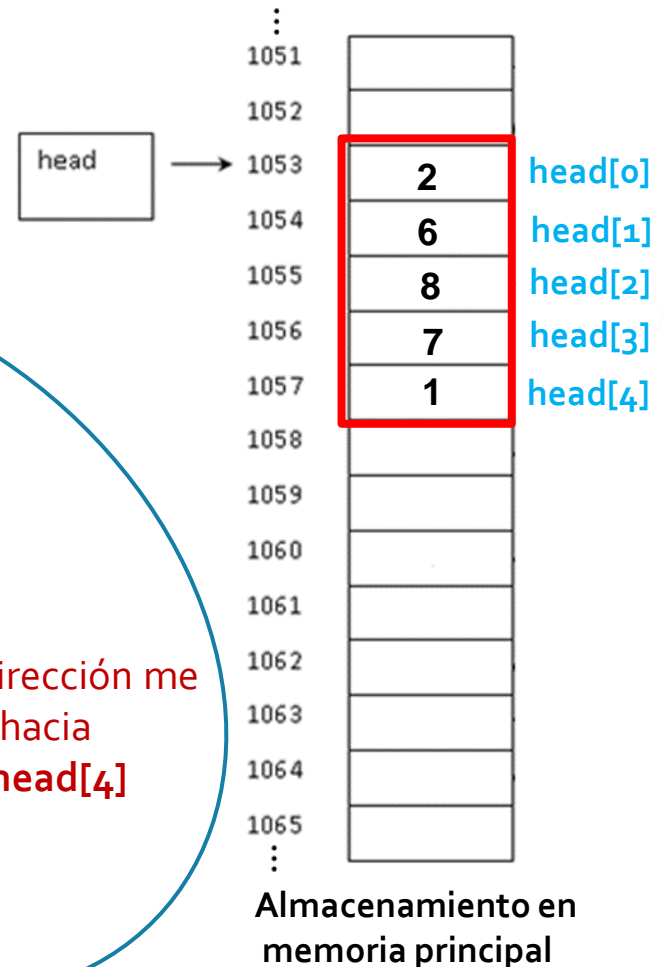
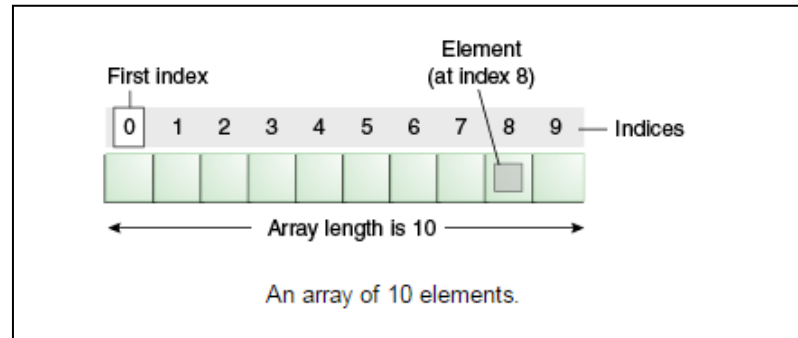
....



Cómo hace para buscar el elemento **head[4]** ???

Sabe que **head** arranca en la dirección 1053 y esa es la dirección del elemento cero.

Entonces si de esa dirección me muevo 4 elementos hacia adelante ahí estará **head[4]**



Estructura de Datos

Array

// en JAVA declara variable de tipo array de enteros

```
int[ ] unArray;
```

// reserva espacio para 10 nros. enteros

// valor por defecto 0

```
unArray = new int[10];
```

Es una secuencia en un **bloque de memoria contiguo**.

Ejemplo JAVA:

```
int[] head = {2, 6, 8, 7, 1}
```

Mismo efecto que:

```
int[] head = new int[5];
```

```
head[0] = 2;
```

```
head[1] = 6;
```

```
head[2] = 8;
```

....

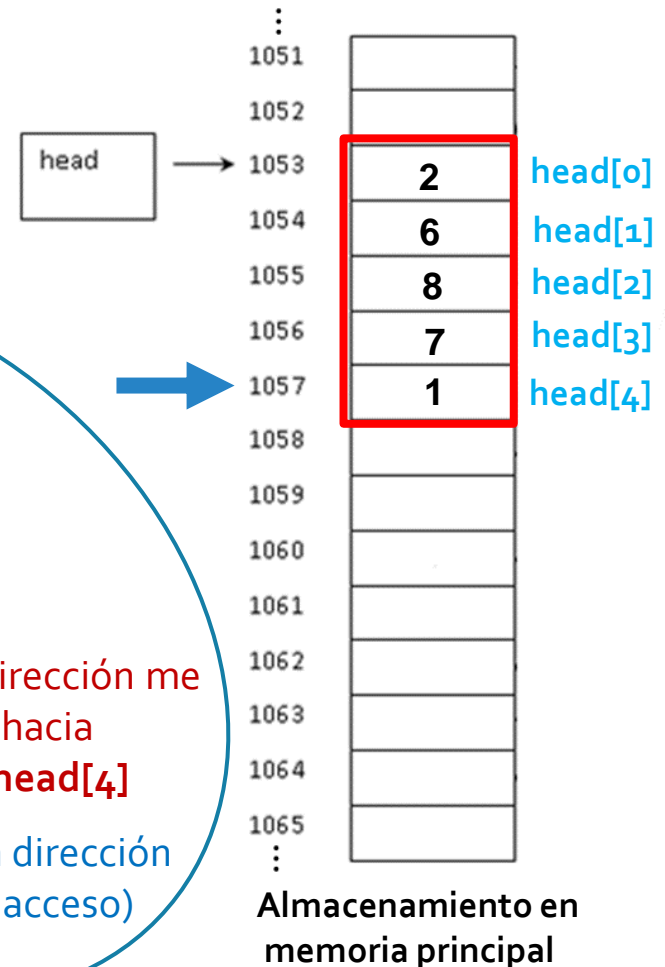
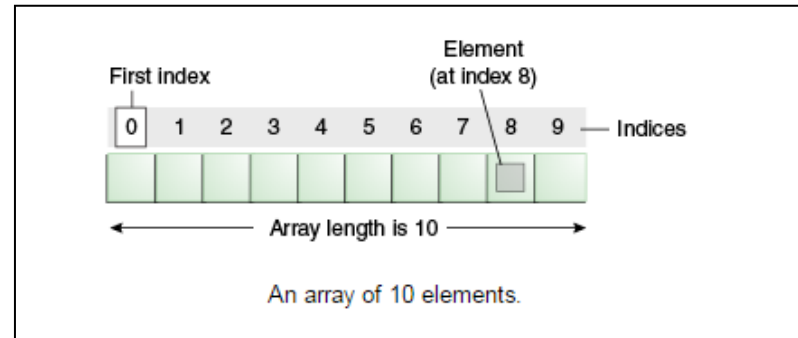


Cómo hace para buscar el elemento **head[4]** ???

Sabe que **head** arranca en la dirección 1053 y esa es la dirección del elemento cero.

Entonces si de esa dirección me muevo 4 elementos hacia adelante ahí estará **head[4]**

O sea estará en la dirección $1053 + 4 = 1057$ (un acceso)



(notación Big O)

(notación Big O)

- La vamos a usar para caracterizar operaciones o algoritmos y poder compararlos entre sí respecto a su comportamiento (nos enfocaremos en el tiempo que insume).

(notación Big O)

- La vamos a usar para caracterizar operaciones o algoritmos y poder compararlos entre sí respecto a su comportamiento (nos enfocaremos en el tiempo que insume).
- Llamaremos n al tamaño de la entrada en general. Si hablamos de una estructura de datos, por ejemplo un array, n será la cantidad de datos que contiene. Si hablamos de un algoritmo n será el tamaño del problema a resolver, ya veremos ejemplos de esto.

(notación Big O)

- La vamos a usar para caracterizar operaciones o algoritmos y poder compararlos entre sí respecto a su comportamiento (nos enfocaremos en el tiempo que insume).
- Llamaremos n al tamaño de la entrada en general. Si hablamos de una estructura de datos, por ejemplo un array, n será la cantidad de datos que contiene. Si hablamos de un algoritmo n será el tamaño del problema a resolver, ya veremos ejemplos de esto.
- La notación Big O siempre mira el peor caso (pesimista).

(notación Big O)

- La vamos a usar para caracterizar operaciones o algoritmos y poder compararlos entre sí respecto a su comportamiento (nos enfocaremos en el tiempo que insume).
- Llamaremos n al tamaño de la entrada en general. Si hablamos de una estructura de datos, por ejemplo un array, n será la cantidad de datos que contiene. Si hablamos de un algoritmo n será el tamaño del problema a resolver, ya veremos ejemplos de esto.
- La notación Big O siempre mira el peor caso (pesimista).
- Nos indica cómo se comporta la operación o el algoritmo respecto al tamaño de la entrada n . Cómo se escribe? Diremos que tal operación es **$O(\text{algo en función de } n)$** .

(notación Big O)

Por ejemplo

- $O(n)$ nos indica que el tiempo de ejecución que insume (en el peor caso) es proporcional n

(notación Big O)

Por ejemplo

- $O(n)$ nos indica que el tiempo de ejecución que insume (en el peor caso) es proporcional n
- $O(n^2)$ nos indica que el tiempo de ejecución que insume (eepe) es proporcional a n^2

(notación Big O)

Por ejemplo

- $O(n)$ nos indica que el tiempo de ejecución que insume (en el peor caso) es proporcional n
- $O(n^2)$ nos indica que el tiempo de ejecución que insume (eepe) es proporcional a n^2
- $O(\log n)$ nos indica que el tiempo de ejecución que insume (eepe) es proporcional a $\log n$.

(notación Big O)

Por ejemplo

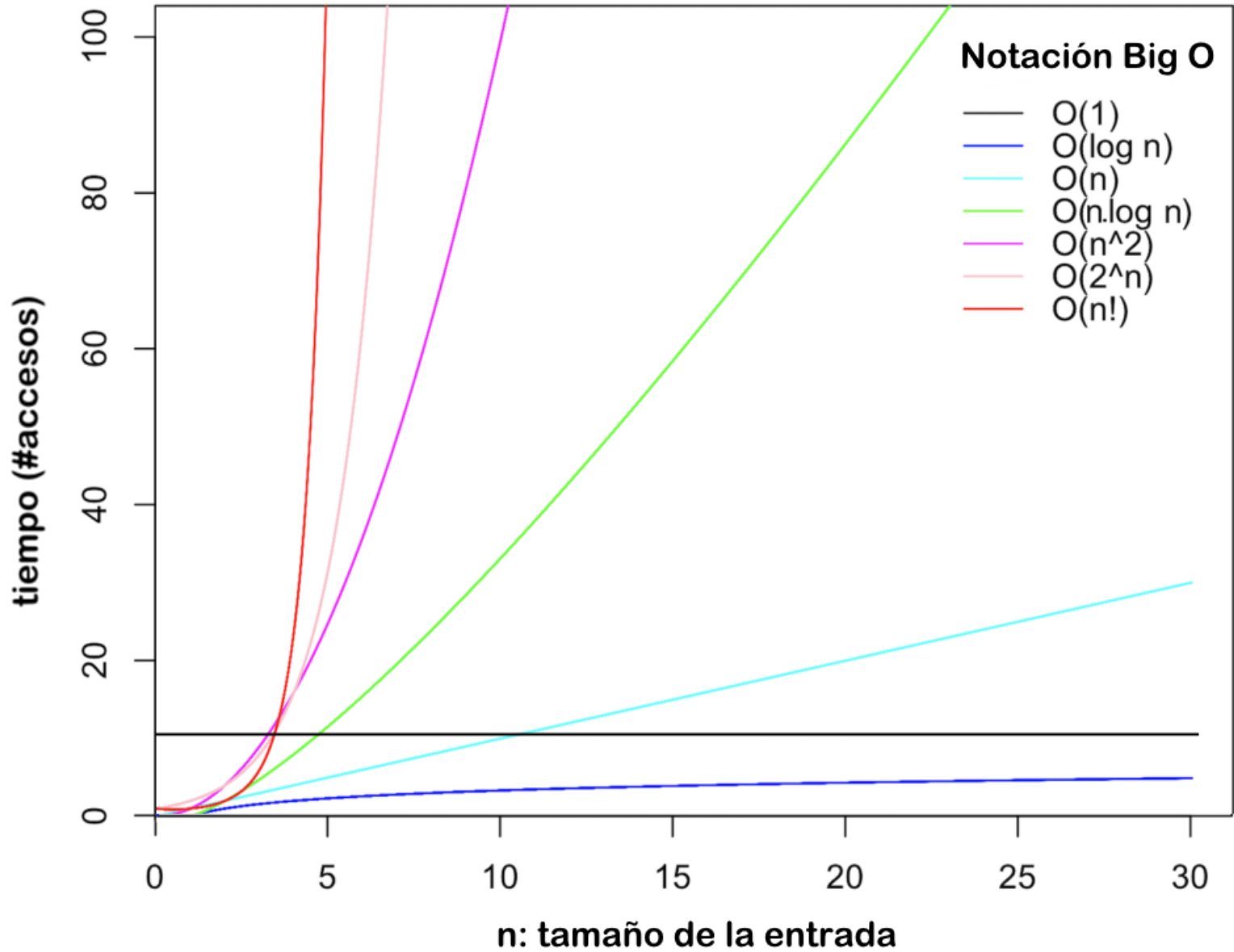
- $O(n)$ nos indica que el tiempo de ejecución que insume (en el peor caso) es proporcional a n
- $O(n^2)$ nos indica que el tiempo de ejecución que insume (eepec) es proporcional a n^2
- $O(\log n)$ nos indica que el tiempo de ejecución que insume (eepec) es proporcional a $\log n$.
- $O(1)$ es algo especial ya que indica que el tiempo no depende del tamaño de la entrada, o sea la operación o algoritmo siempre tarda lo mismo, un tiempo constante.

(notación Big O)

Por ejemplo

- $O(n)$ nos indica que el tiempo de ejecución que insume (en el peor caso) es proporcional a n
- $O(n^2)$ nos indica que el tiempo de ejecución que insume (eepec) es proporcional a n^2
- $O(\log n)$ nos indica que el tiempo de ejecución que insume (eepec) es proporcional a $\log n$.
- $O(1)$ es algo especial ya que indica que el tiempo no depende del tamaño de la entrada, o sea la operación o algoritmo siempre tarda lo mismo, un tiempo constante.
- En esta anotación no se tienen en cuenta los términos o factores constantes dentro de $O()$ así es que $O(2.n)$, $O(10.n)$, $O(n+2)$, $O(n+25)$, $O(n-1)$ son todas $O(n)$
- Para tener una idea de qué Big O tiene una operación dada, usaremos la estrategia de estimar la cantidad de accesos a memoria que hace esa operación.

(notación Big O)



Array

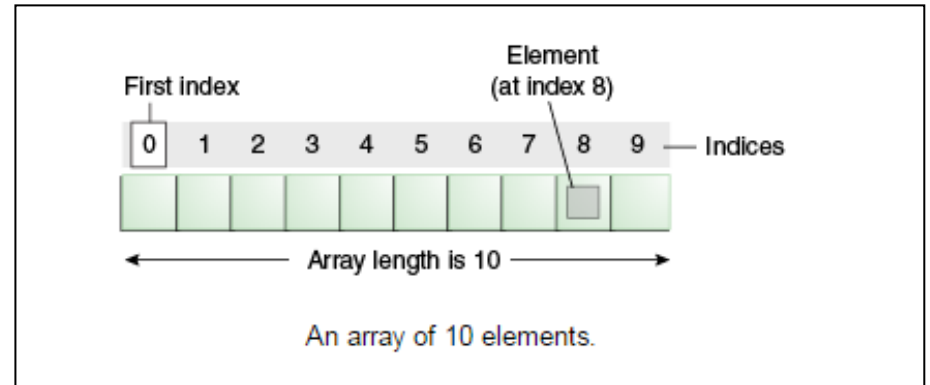
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



- Es una secuencia en un **bloque de memoria contiguo**.

Array

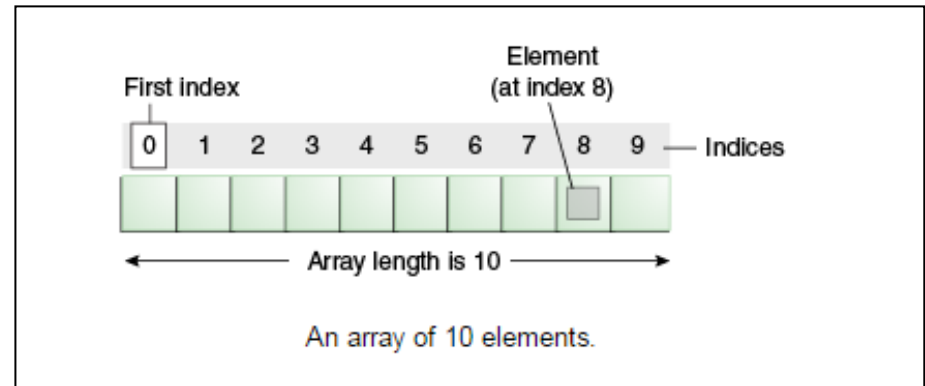
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador [], por ej: ***int k = unArray[10];***
(en un sólo acceso a memoria obtengo elemento en el índice 10)

Array

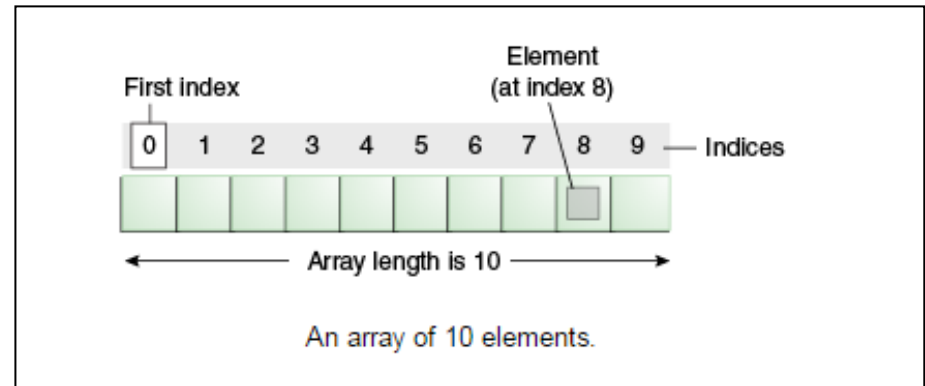
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador [], por ej: ***int k = unArray[10];***
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).

Array

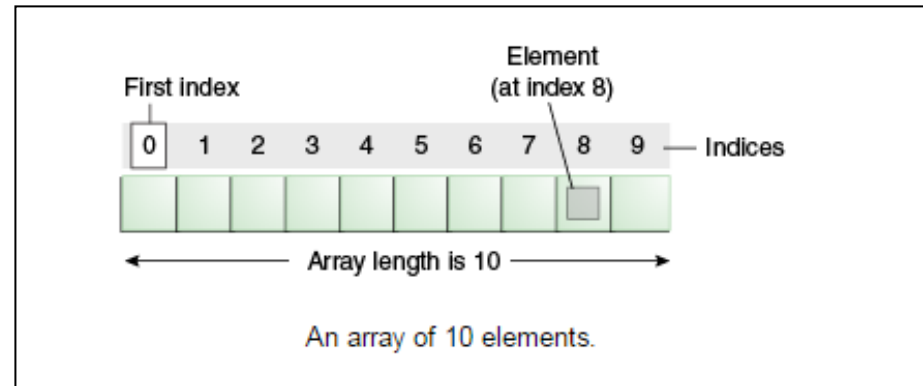
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador [], por ej: ***int k = unArray[10];***
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)

Array

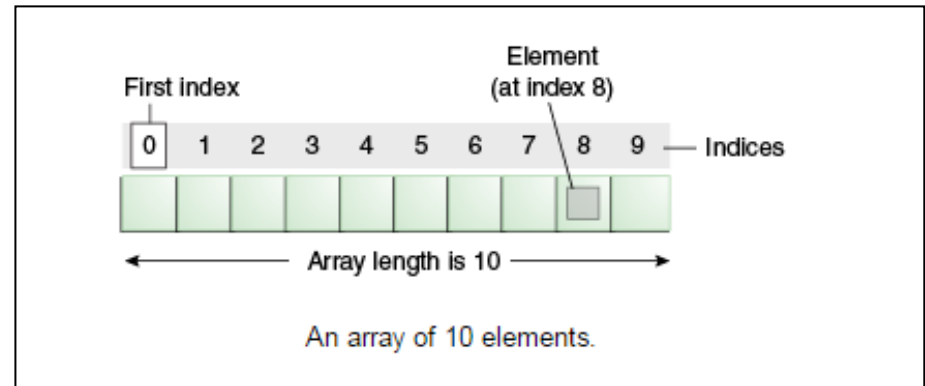
// en JAVA declara variable de tipo array de enteros

`int[] unArray;`

// reserva espacio para **10** nros. enteros

// valor por defecto 0

`unArray = new int[10];`



- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador `[]`, por ej: **`int k = unArray[10];`**
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)
- ¿Si quiero escribir un valor en la posición *i* ?

Array

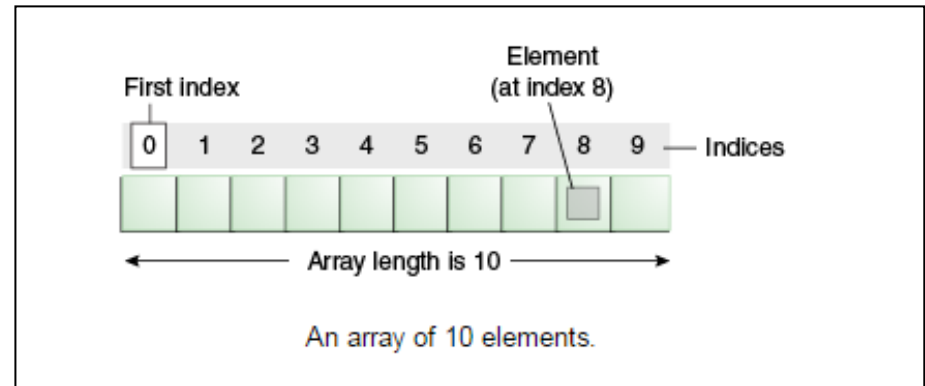
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador [], por ej: ***int k = unArray[10];***
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)
- ¿Si quiero escribir un valor en la posición *i* ?
- ¿y si lo quiero **insertar** en la posición *i* ?

Array

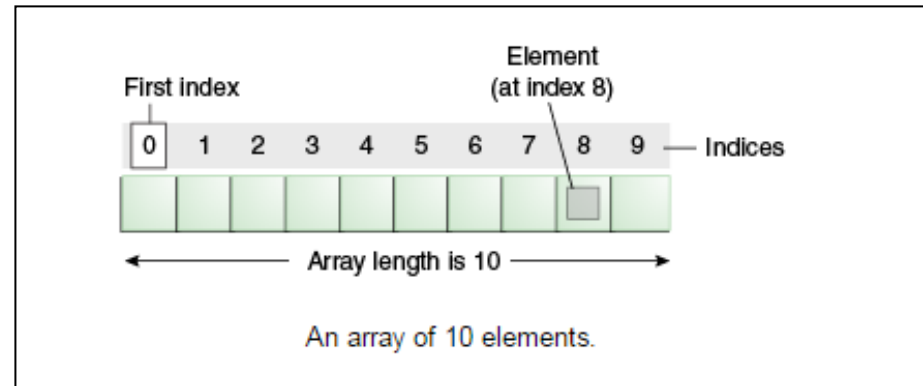
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador `[]`, por ej: ***int k = unArray[10];***
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)
- ¿Si quiero escribir un valor en la posición *i* ?
- ¿y si lo quiero **insertar** en la posición *i* ?
- ¿y si queda chico el espacio ?

Array

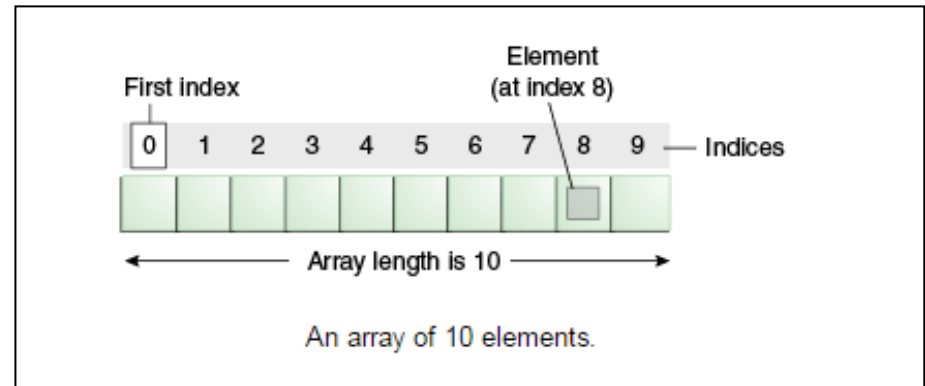
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



Notación Big O
(peor caso)

- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador `[]`, por ej: ***int k = unArray[10];***
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)
- ¿Si quiero escribir un valor en la posición *i* ?
- ¿y si lo quiero **insertar** en la posición *i* ?
- ¿y si queda chico el espacio ?

Array

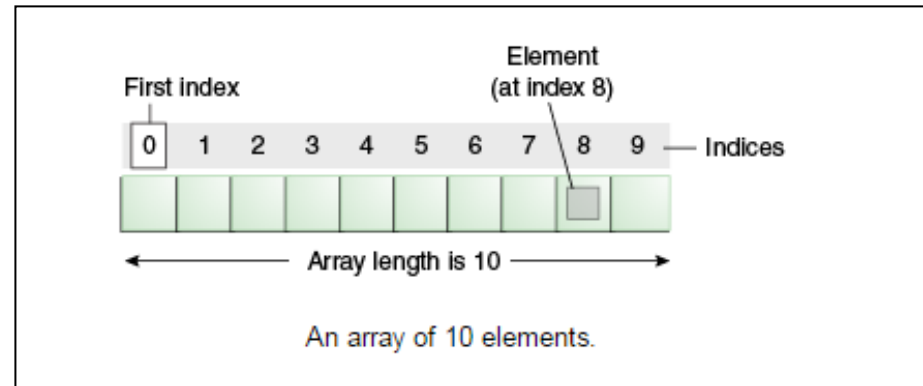
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



Notación Big O (peor caso)

- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador `[]`, por ej: ***int k = unArray[10];*** ***O(1)*** insume tiempo constante (siempre lo mismo, 1 acceso).
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)
- ¿Si quiero escribir un valor en la posición *i* ?
- ¿y si lo quiero **insertar** en la posición *i* ?
- ¿y si queda chico el espacio ?

Array

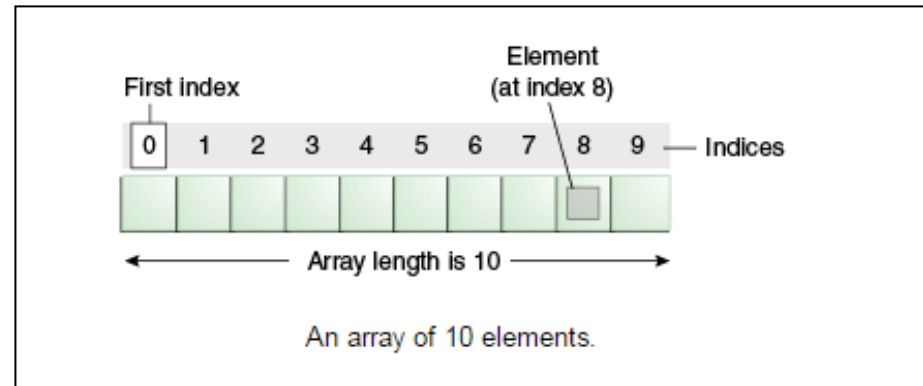
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



**Notación Big O
(peor caso)**

- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador [], por ej: ***int k = unArray[10];*** ***O(1)*** insume tiempo constante (siempre lo mismo, 1 acceso).
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento ***i*** para acceder en forma directa ? (inicio +***i***)
- ¿Si quiero escribir un valor en la posición ***i*** ? ***O(1)***
- ¿y si lo quiero **insertar** en la posición ***i*** ?
- ¿y si queda chico el espacio ?

Array

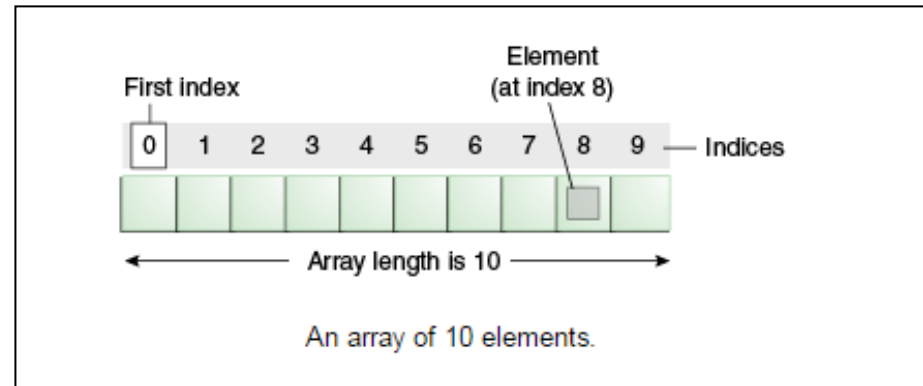
// en JAVA declara variable de tipo array de enteros

int[] unArray;

// reserva espacio para 10 nros. enteros

// valor por defecto 0

unArray = new int[10];



Notación Big O (peor caso)

- Es una secuencia en un **bloque de memoria contiguo**.
- **acceso directo** mediante el operador `[]`, por ej: ***int k = unArray[10];*** **$O(1)$** **insume tiempo constante (siempre lo mismo, 1 acceso).**
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento de asignar memoria (usamos ***new***).
- ¿Cómo sabe donde se encuentra el elemento *i* para acceder en forma directa ? (inicio +i)
- ¿Si quiero escribir un valor en la posición *i* ? **$O(1)$**
- ¿y si lo quiero **insertar** en la posición *i* ? **$O(n)$** **En el PEOR CASO debo hacer n corrimientos.**
- ¿y si queda chico el espacio ?

Array

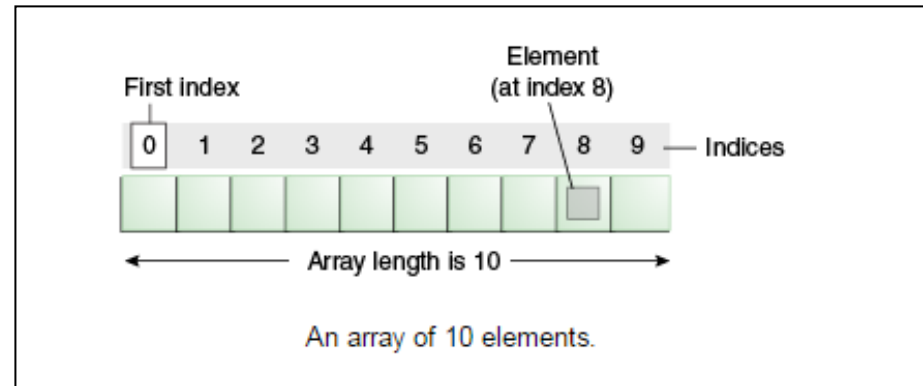
// en JAVA declara variable de tipo array de enteros

```
int[ ] unArray;
```

// reserva espacio para **10** nros. enteros

// valor por defecto 0

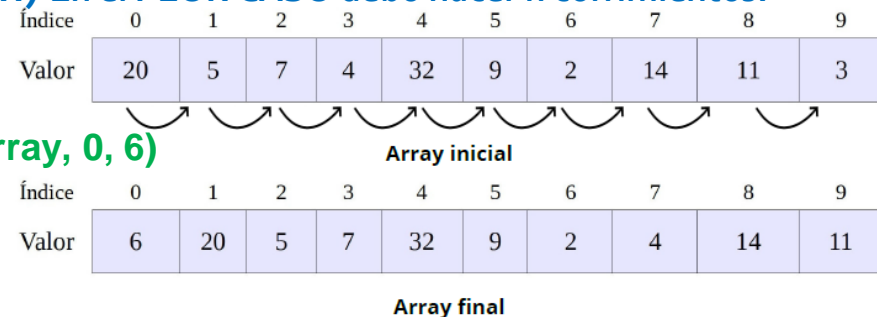
```
unArray = new int[10];
```



Notación Big O (peor caso)

- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador `[]`, por ej: **`int k = unArray[10];`** **$O(1)$** **insume tiempo constante (siempre lo mismo, 1 acceso).**
(en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño? Debe declararse al momento de asignar memoria (usamos **`new`**).
- ¿Cómo sabe donde se encuentra el elemento `i` para acceder en forma directa? (inicio + `i`)
- ¿Si quiero escribir un valor en la posición `i`? **$O(1)$**
- ¿y si lo quiero **insertar** en la posición `i`? **$O(n)$** **En el PEOR CASO debo hacer `n` corrimientos.**
- ¿y si queda chico el espacio?

insertar(unArray, 0, 6)



Array

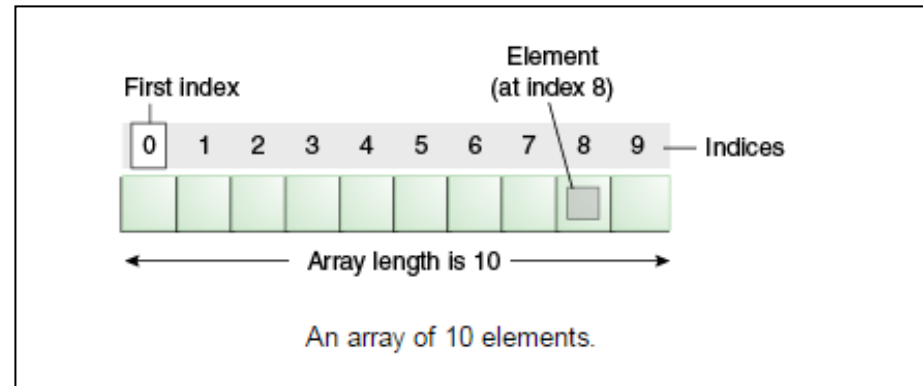
// en JAVA declara variable de tipo array de enteros

```
int[ ] unArray;
```

// reserva espacio para **10** nros. enteros

// valor por defecto 0

```
unArray = new int[10];
```



Notación Big O (peor caso)

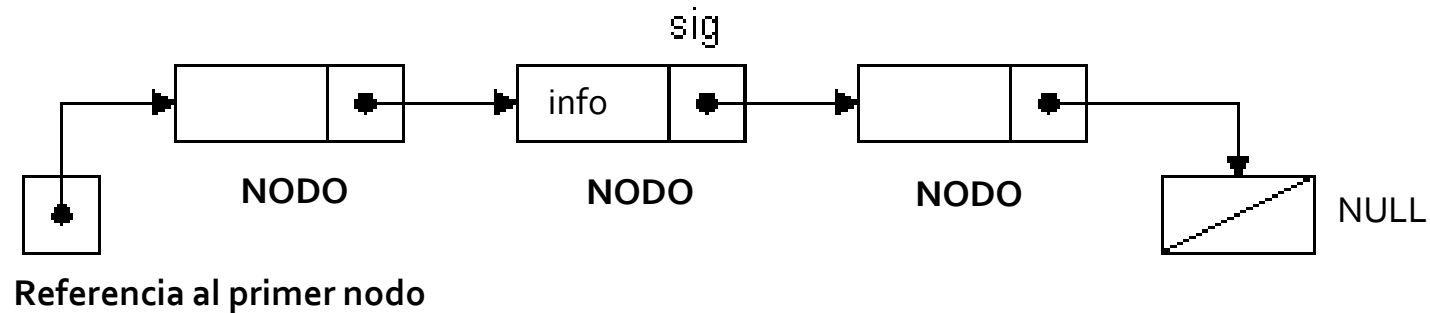
- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador `[]`, por ej: **`int k = unArray[10];`** **$O(1)$** insume tiempo constante (siempre lo mismo, 1 acceso).
(en un sólo acceso a memoria obtengo elemento en el índice 9)
- ¿tamaño? Debe declararse al momento de asignar memoria (usamos **`new`**).
- ¿Cómo sabe donde se encuentra el elemento `i` para acceder en forma directa? (inicio + `i`)
- ¿Si quiero escribir un valor en la posición `i`? **$O(1)$**
- ¿y si lo quiero **insertar** en la posición `i`? **$O(n)$** En el PEOR CASO debo hacer `n` corrimientos.
- ¿y si queda chico el espacio? **$O(n)$**

insertar(unArray, 0, 6)



Lista vinculada

Las listas vinculadas son aquellas en las que **en cada elemento de la lista (al que llamaremos nodo) se incluye, además de información, un campo llamado *puntero* o *referencia* que sirve para encadenar dicho nodo con el siguiente de la lista.**



Lista vinculada

Cada nodo tiene al menos dos campos:

- un campo de información o valor,
- la referencia al siguiente elemento

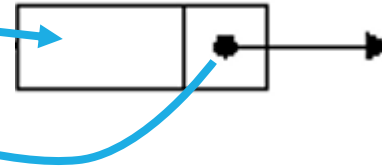
No es necesario que los nodos de la lista sean almacenados en posiciones físicas adyacentes, la referencia (puntero) al siguiente nodo indica dónde se encuentra dicho nodo en la memoria.

Una lista vinculada sin ningún nodo se llama **lista vacía**, en este caso la variable que apunta al primer nodo tiene el valor **nulo (null)**.

Lista vinculada

Cada nodo tiene al menos dos campos:

- un campo de información o valor,
- la referencia al siguiente elemento



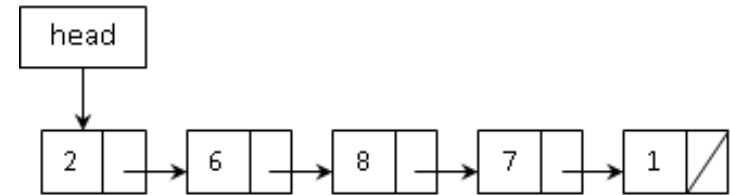
No es necesario que los nodos de la lista sean almacenados en posiciones físicas adyacentes, la referencia (puntero) al siguiente nodo indica dónde se encuentra dicho nodo en la memoria.

Una lista vinculada sin ningún nodo se llama **lista vacía**, en este caso la variable que apunta al primer nodo tiene el valor **nulo (null)**.

Lista vinculada

Representación de la lista vinculada de nros. enteros

Variable **head**, es una referencia al primer nodo.



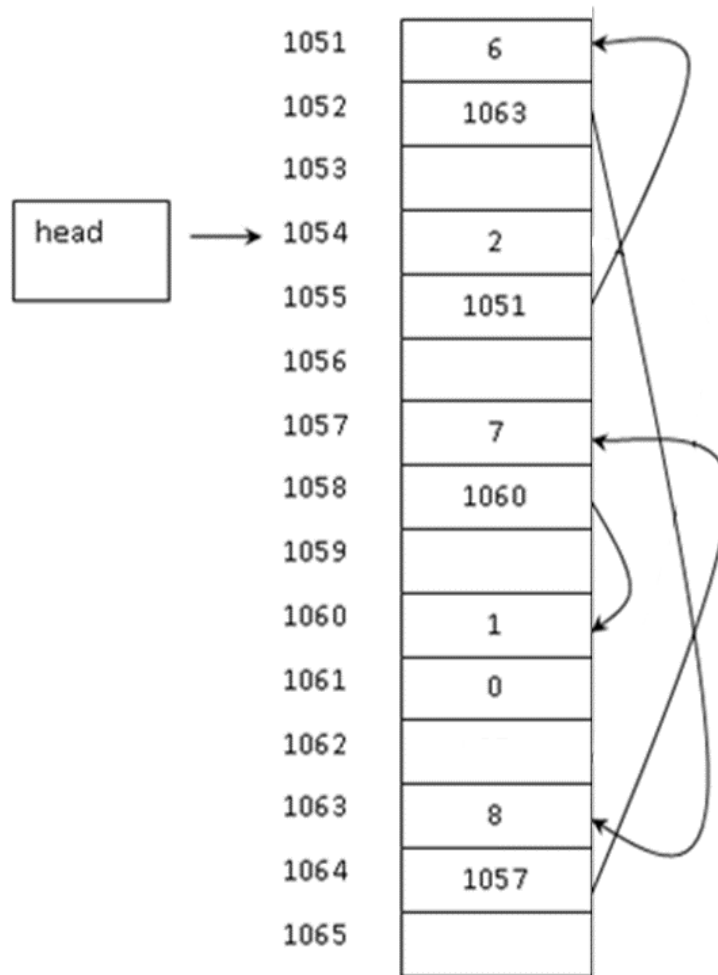
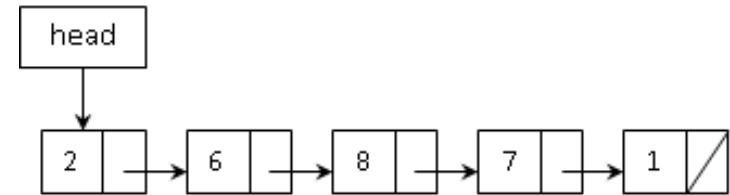
head	→	1051	6
		1052	1063
		1053	
		1054	2
		1055	1051
		1056	
		1057	7
		1058	1060
		1059	
		1060	1
		1061	0
		1062	
		1063	8
		1064	1057
	1065		

Almacenamiento en memoria principal

Lista vinculada

Representación de la lista vinculada de nros. enteros

Variable **head**, es una referencia al primer nodo.

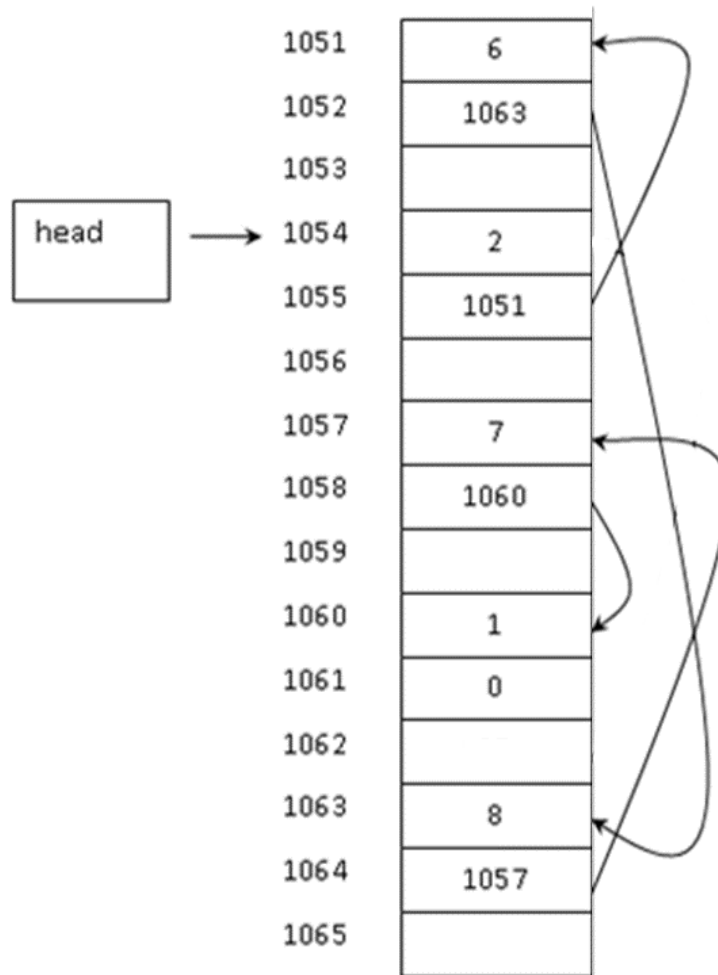
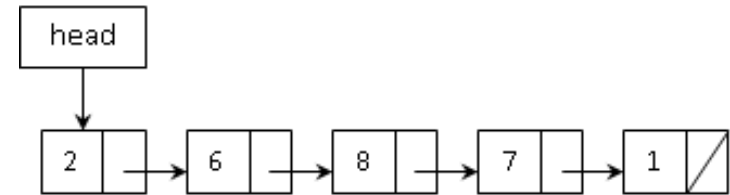


Almacenamiento en memoria principal

Lista vinculada

Representación de la lista vinculada de nros. enteros

Variable **head**, es una referencia al primer nodo.



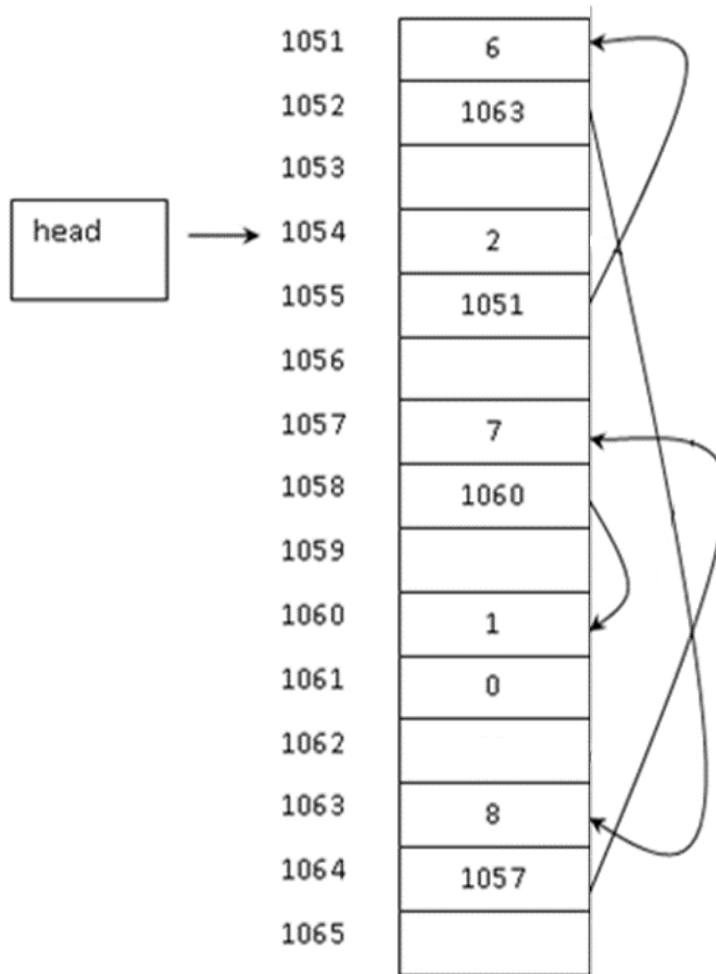
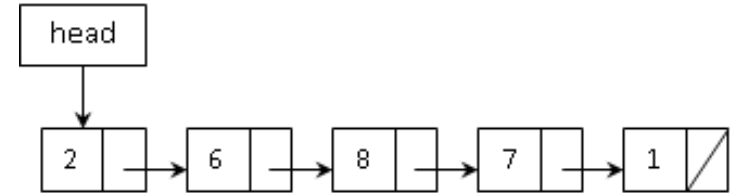
**Cantidad de accesos
para obtener el cuarto
elemento?**

Almacenamiento en memoria principal

Lista vinculada

Representación de la lista vinculada de nros. enteros

Variable **head**, es una referencia al primer nodo.



Almacenamiento en memoria principal

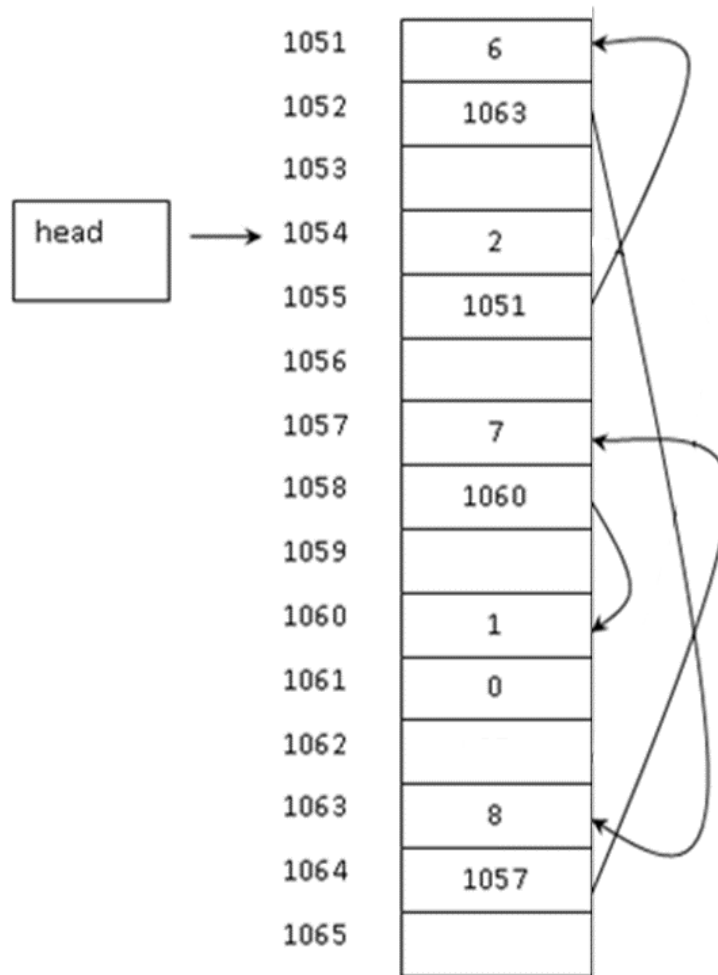
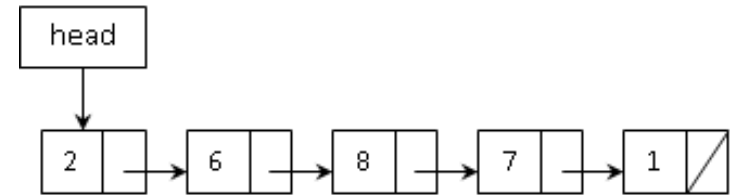
Cantidad de accesos
para obtener el cuarto
elemento?

Y para obtener el último?
Y si tengo n elementos?

Lista vinculada

Representación de la lista vinculada de nros. enteros

Variable **head**, es una referencia al primer nodo.



Almacenamiento en memoria principal

Cantidad de accesos
para obtener el cuarto
elemento?

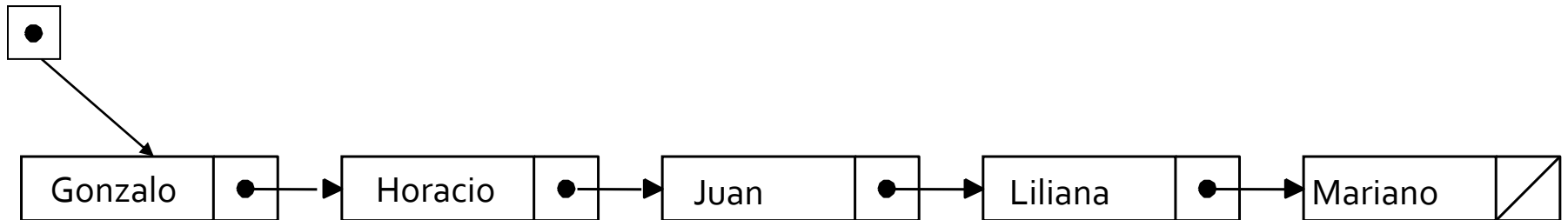
Y para obtener el último?
Y si tengo n elementos?

Obtener el valor
en una posición dada es $O(n)$

Lista vinculada - Operaciones

Inserción de elementos

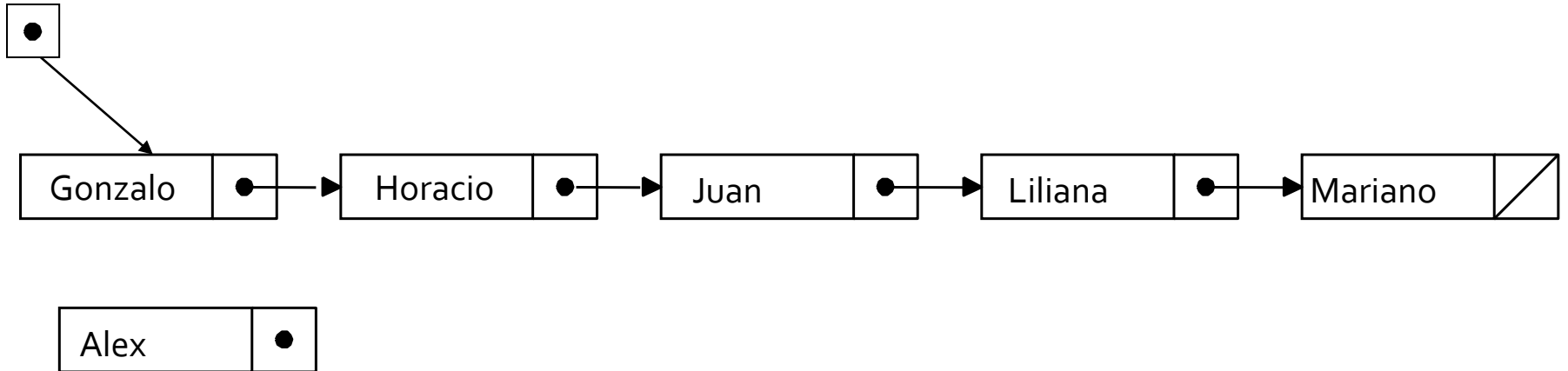
Supongamos que queremos mantener la lista de nombres ordenada.



Lista vinculada - Operaciones

Inserción de elementos

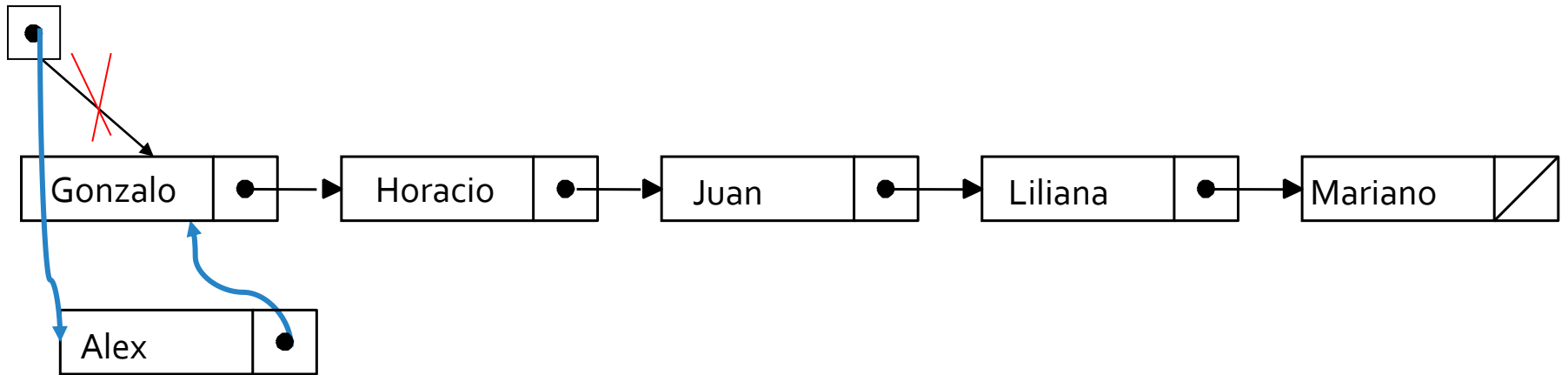
Supongamos que queremos mantener la lista de nombres ordenada.



Lista vinculada - Operaciones

Inserción de elementos

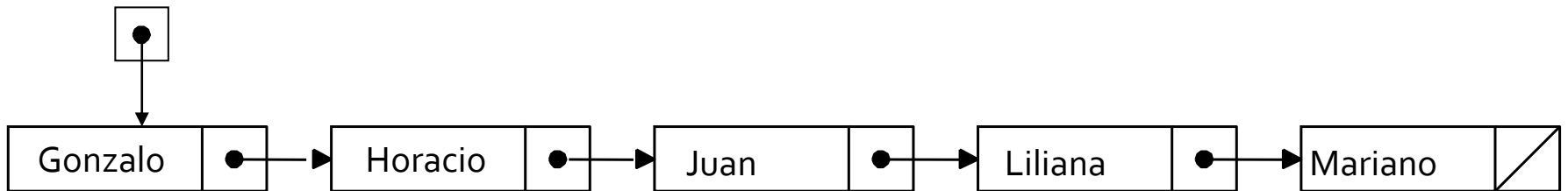
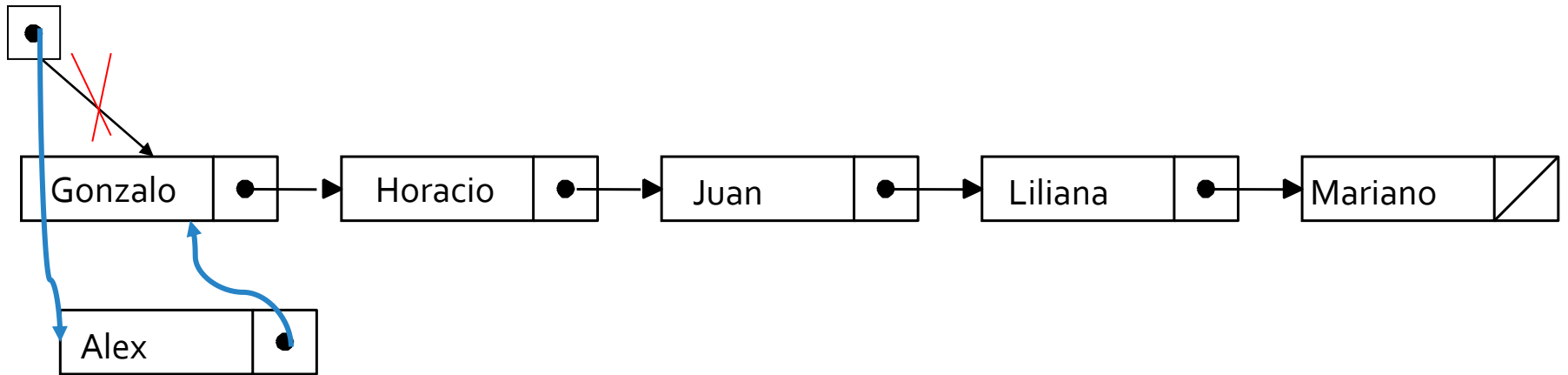
Supongamos que queremos mantener la lista de nombres ordenada.



Lista vinculada - Operaciones

Inserción de elementos

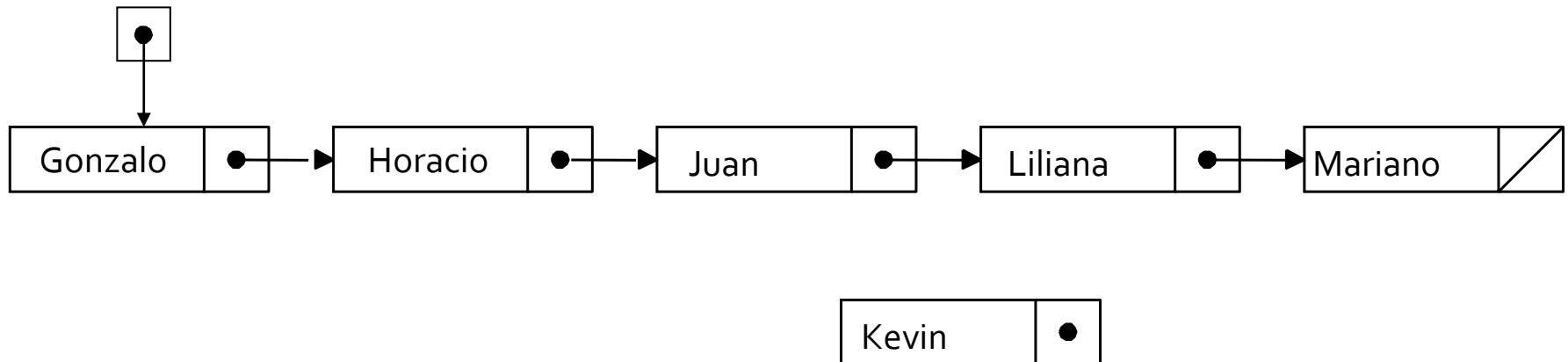
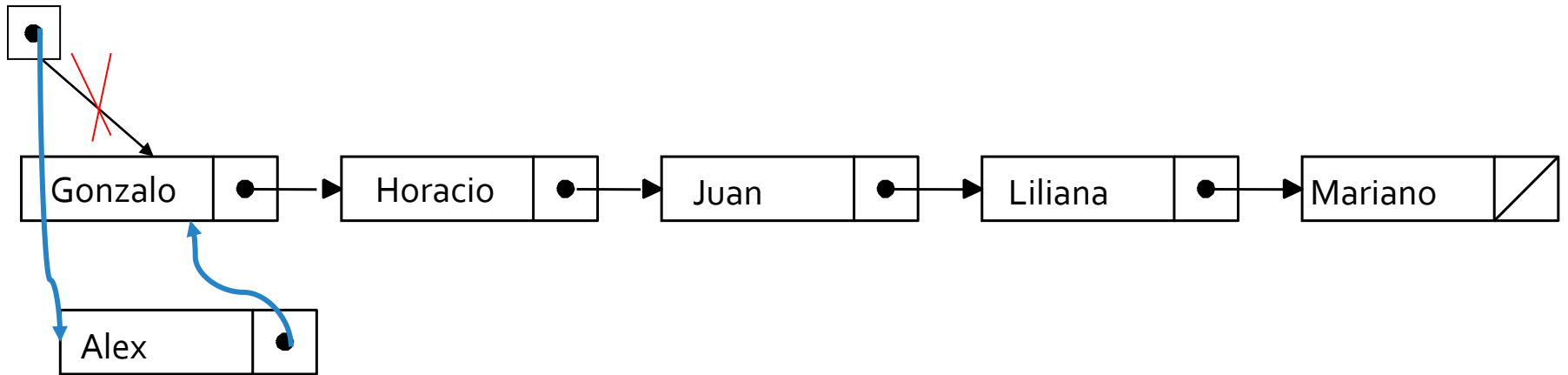
Supongamos que queremos mantener la lista de nombres ordenada.



Lista vinculada - Operaciones

Inserción de elementos

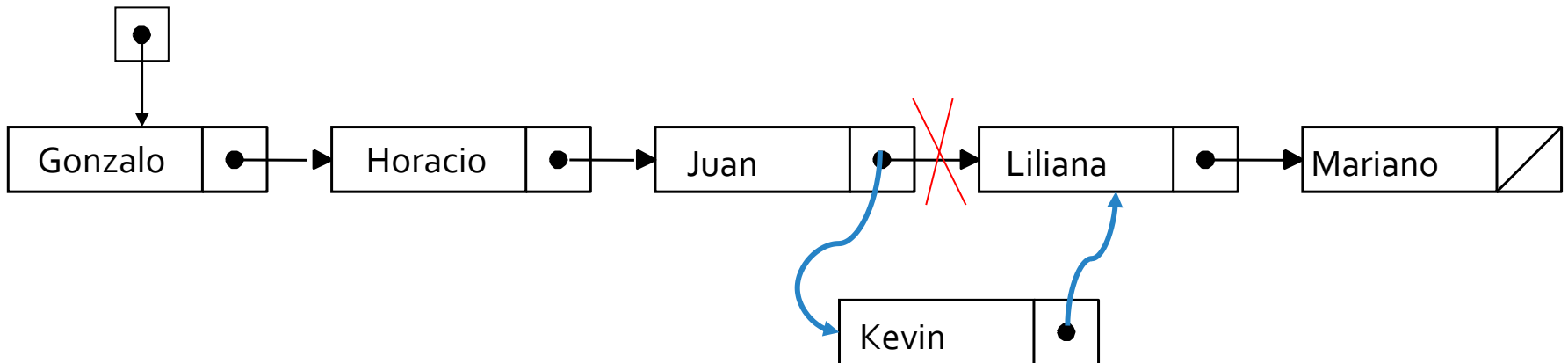
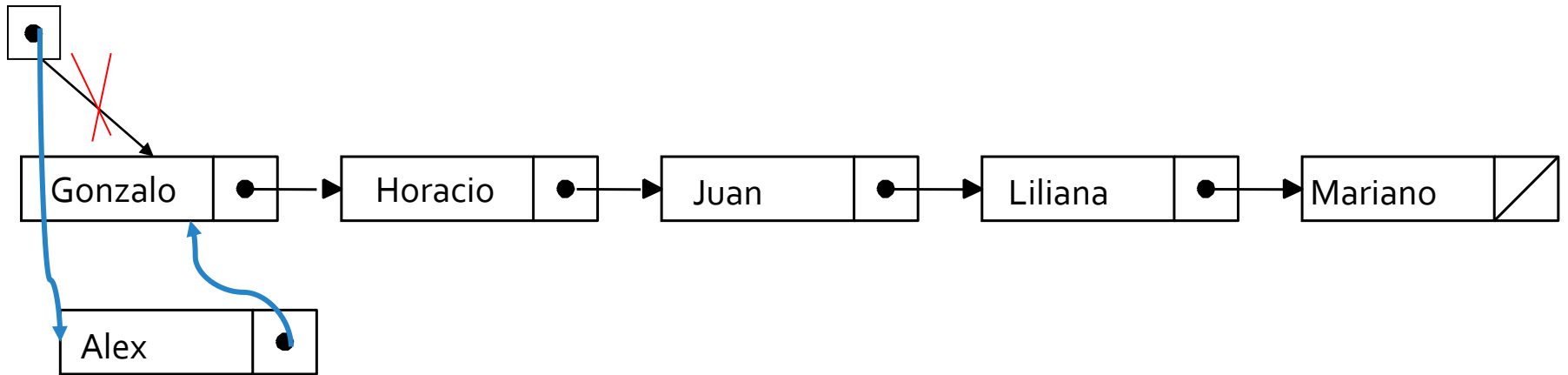
Supongamos que queremos mantener la lista de nombres ordenada.



Lista vinculada - Operaciones

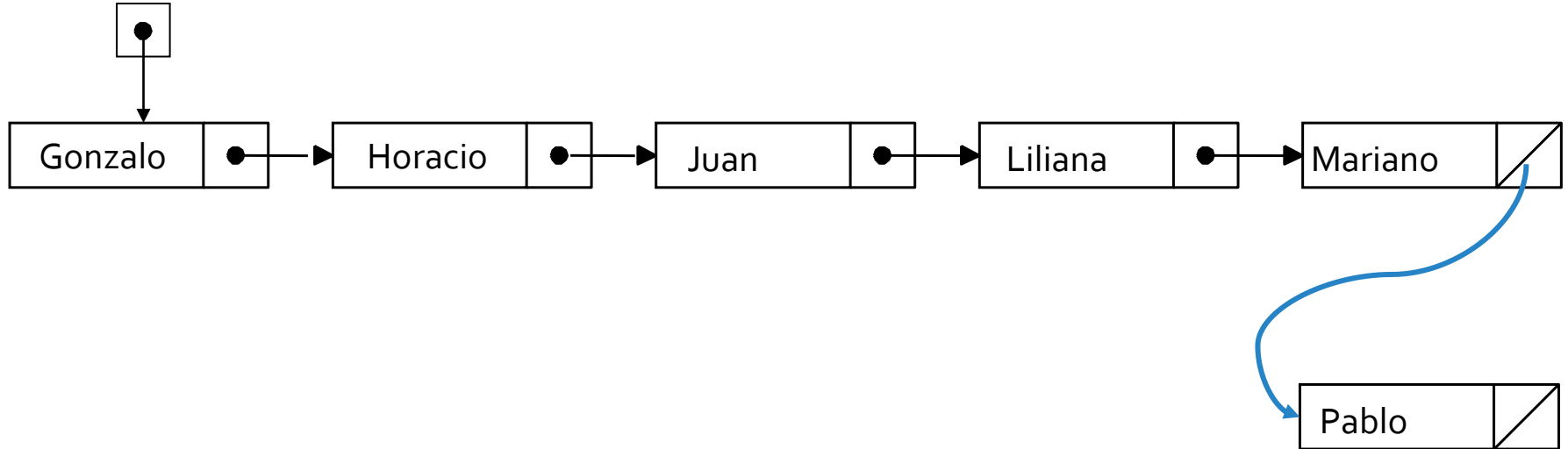
Inserción de elementos

Supongamos que queremos mantener la lista de nombres ordenada.



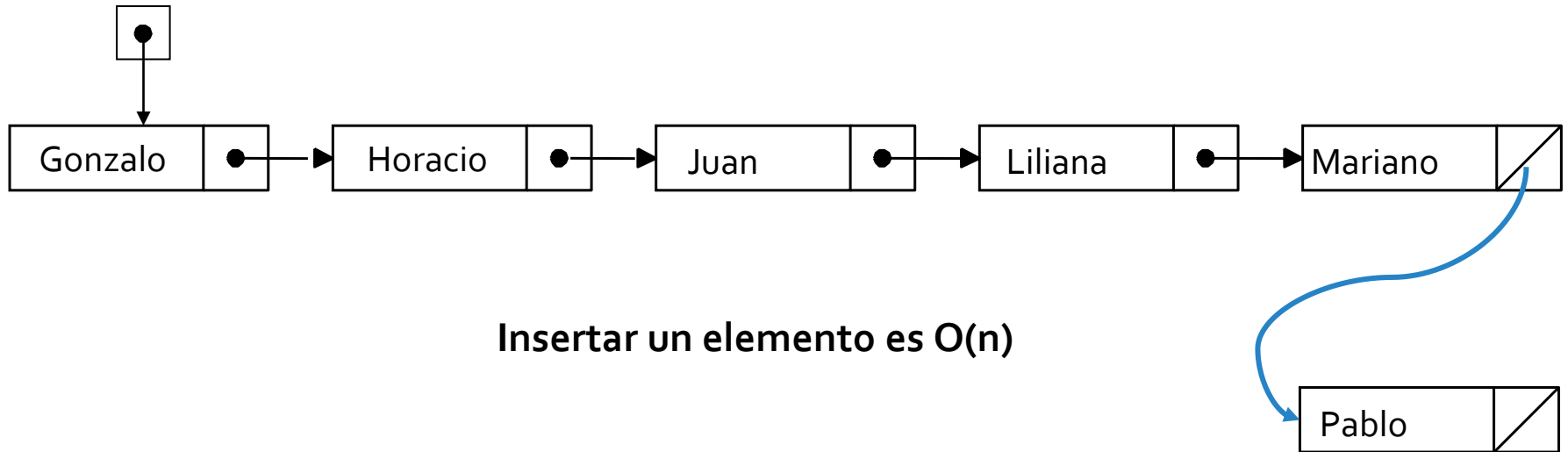
Lista vinculada - Operaciones

Inserción de elementos



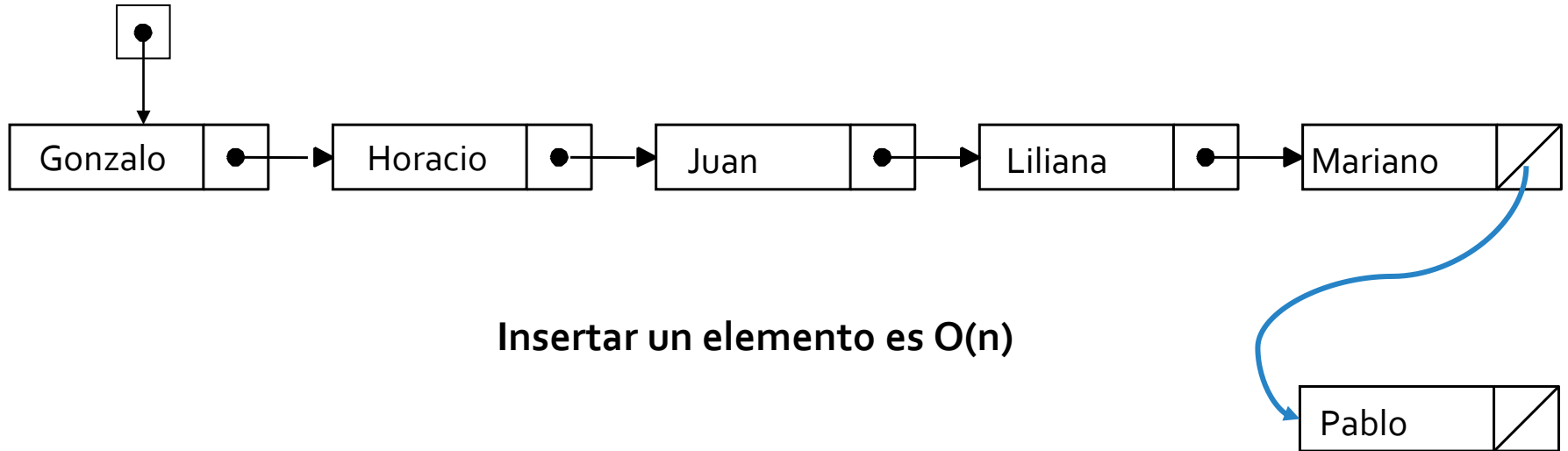
Lista vinculada - Operaciones

Insertión de elementos

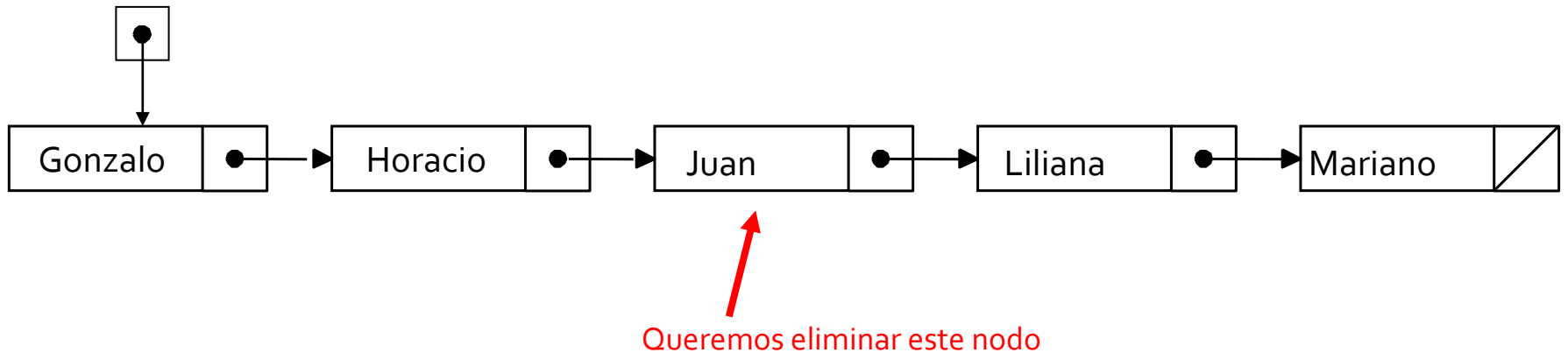


Lista vinculada - Operaciones

Insertión de elementos

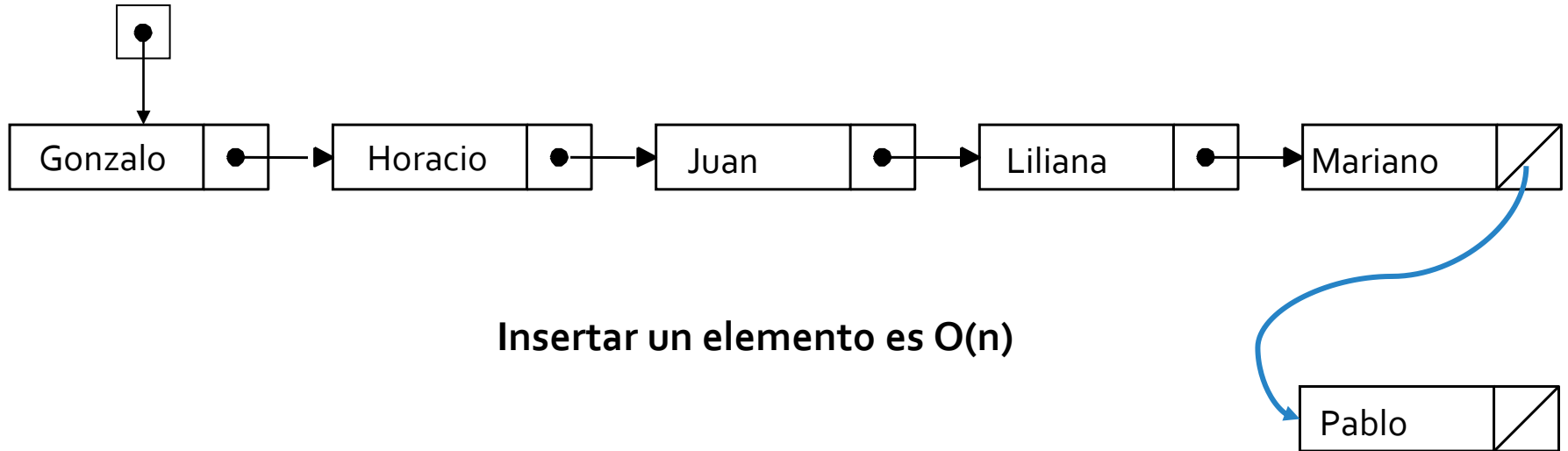


Borrado de elementos

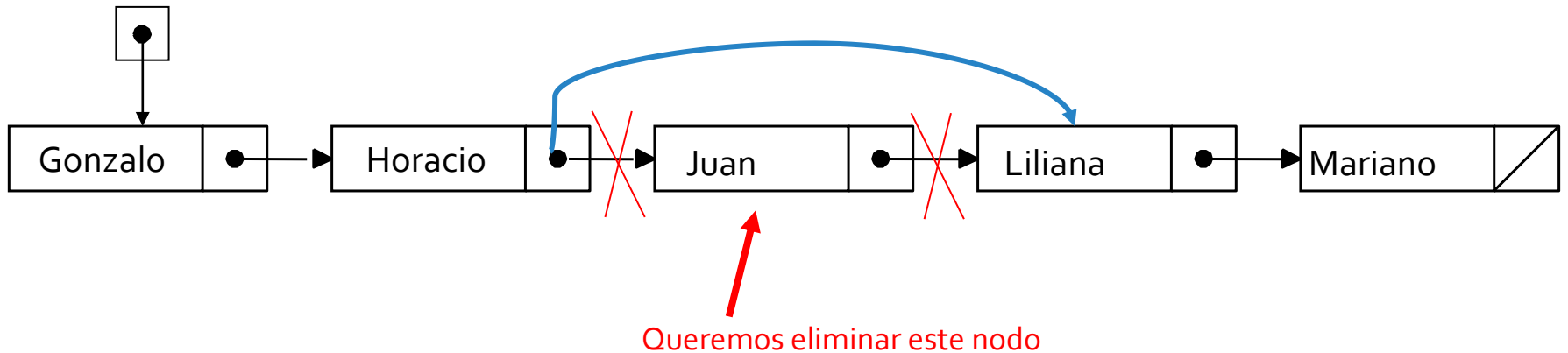


Lista vinculada - Operaciones

Insertión de elementos



Borrado de elementos



Creando una lista vinculada en JAVA

```
public class Node<T> {

    private T info;
    private Node<T> next;

    public Node() {
        this.info = null;
        this.next = null;
    }

    public Node(T info, Node<T> next) {
        this.setInfo(info);
        this.setNext(next);
    }

    public Node<T> getNext() {
        return next;
    }

    public void setNext(Node<T> next) {
        this.next = next;
    }

    public T getInfo() {
        return info;
    }

    public void setInfo(T info) {
        this.info = info;
    }

}
```

Creando una lista vinculada en JAVA

```
public class Node<T> {

    private T info;
    private Node<T> next;

    public Node() {
        this.info = null;
        this.next = null;
    }

    public Node(T info, Node<T> next) {
        this.setInfo(info);
        this.setNext(next);
    }

    public Node<T> getNext() {
        return next;
    }

    public void setNext(Node<T> next) {
        this.next = next;
    }

    public T getInfo() {
        return info;
    }

    public void setInfo(T info) {
        this.info = info;
    }

}
```

```
public class MySimpleLinkedList<T> {

    private Node<T> first;

    public MySimpleLinkedList() {
        this.first = null;
    }

    public void insertFront(T info) {
        Node<T> tmp = new Node<T>(info, null);
        tmp.setNext(this.first);
        this.first = tmp;
    }

    public T extractFront() { }

    public boolean isEmpty() { }

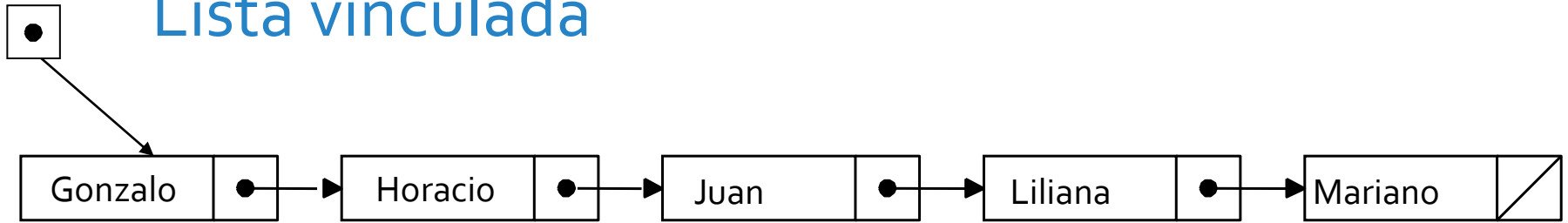
    public T get(int index) { }

    public int size() { }

    @Override
    public String toString() { }

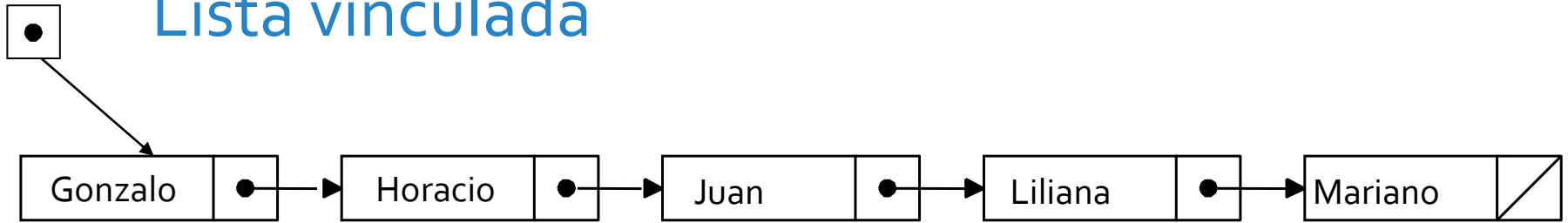
}
```


Lista vinculada



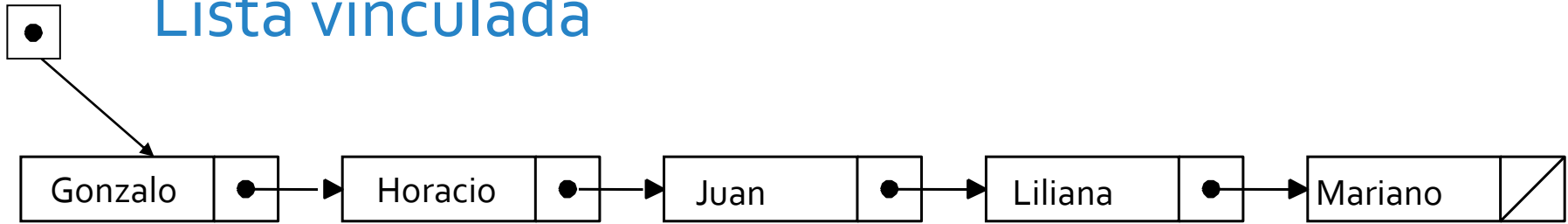
- Qué pasa si quiero imprimir el tamaño de la lista (cantidad de elementos) ?

Lista vinculada



- Qué pasa si quiero imprimir el tamaño de la lista (cantidad de elementos)? $O(n)$

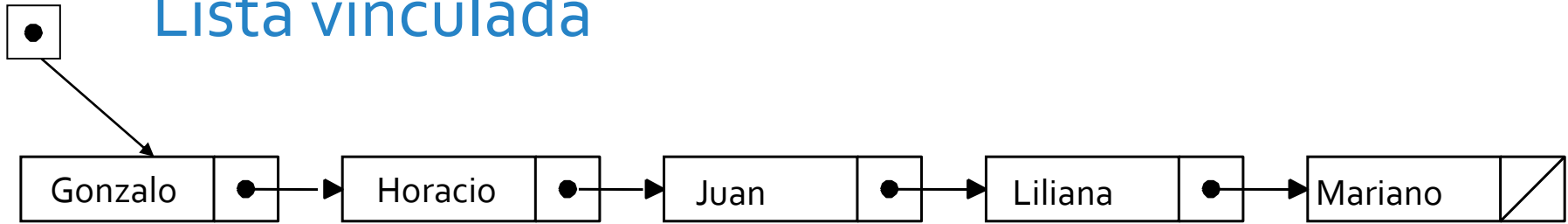
Lista vinculada



- Qué pasa si quiero imprimir el tamaño de la lista (cantidad de elementos)? $O(n)$
- Mejor estrategia: guardar el tamaño en una variable de instancia.

```
public class MySimpleLinkedList {  
    protected Node first;  
    protected int _size;  
  
    public MySimpleLinkedList() {  
        first = null;  
        _size = 0;  
    }  
  
    public void insertFront(Integer i) {  
        Node tmp = new Node(i, null);  
        tmp.setNext(first);  
        first = tmp;  
        _size = _size + 1;  
    }  
  
    public int size () {  
        return _size;  
    }  
}
```

Lista vinculada



- Qué pasa si quiero imprimir el tamaño de la lista (cantidad de elementos)? $O(n)$
- Mejor estrategia: guardar el tamaño en una variable de instancia.

```
public class MySimpleLinkedList {
    protected Node first;
    protected int _size;

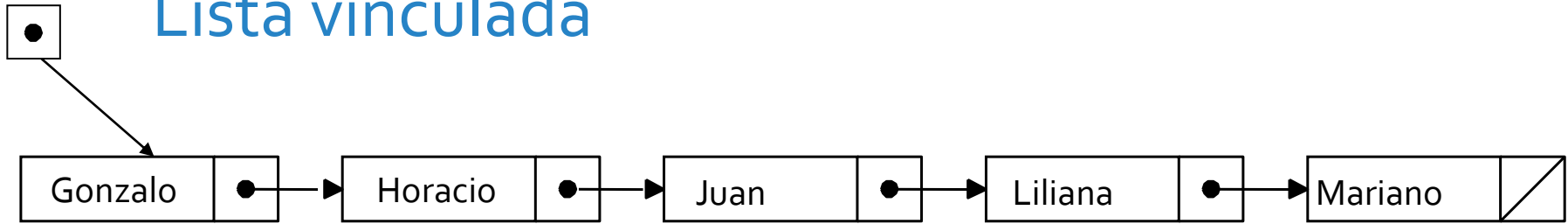
    public MySimpleLinkedList() {
        first = null;
        _size = 0;
    }

    public void insertFront(Integer i) {
        Node tmp = new Node(i, null);
        tmp.setNext(first);
        first = tmp;
        _size = _size + 1;
    }

    public int size () {
        return _size;
    }
}
```

$O(1)$

Lista vinculada



- Qué pasa si quiero imprimir el tamaño de la lista (cantidad de elementos)? $O(n)$
- Mejor estrategia: guardar el tamaño en una variable de instancia.

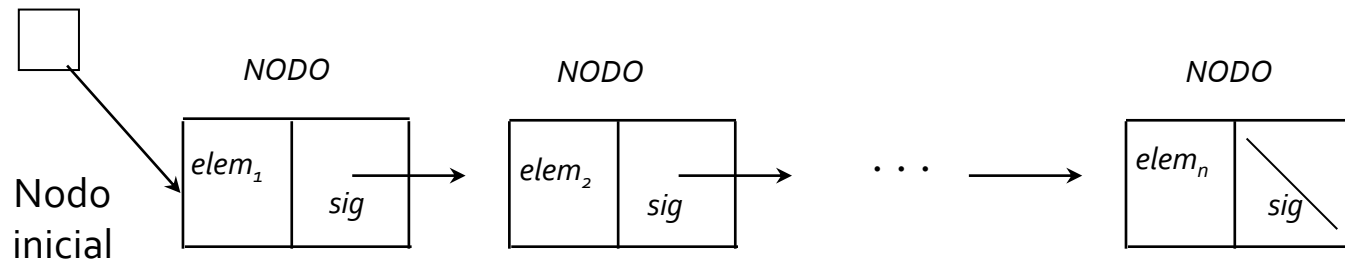
```
public class MySimpleLinkedList {  
    protected Node first;  
    protected int _size;  
  
    public MySimpleLinkedList() {  
        first = null;  
        _size = 0;  
    }  
  
    public void insertFront(Integer i) {  
        Node tmp = new Node(i, null);  
        tmp.setNext(first);  
        first = tmp;  
        _size = _size + 1;  
    }  
  
    public int size () {  
        return _size;  
    }  
}
```

$O(1)$

¿ Y si estoy en el elemento i , y quiero ir al anterior ?

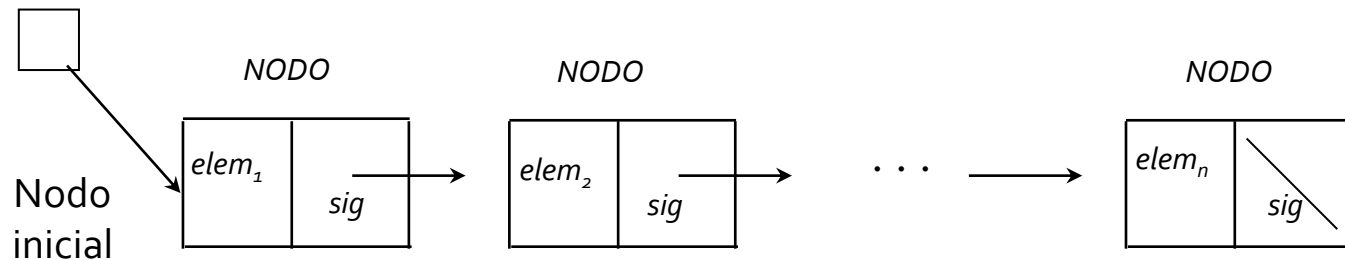
Lista vinculada

Simplemente vinculadas (son las que vimos hasta el momento):

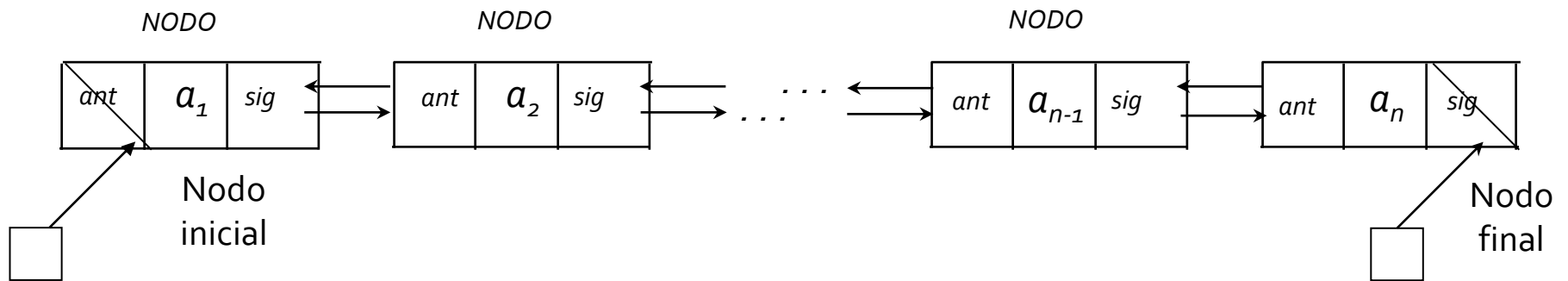


Lista vinculada

Simplemente vinculadas (son las que vimos hasta el momento):

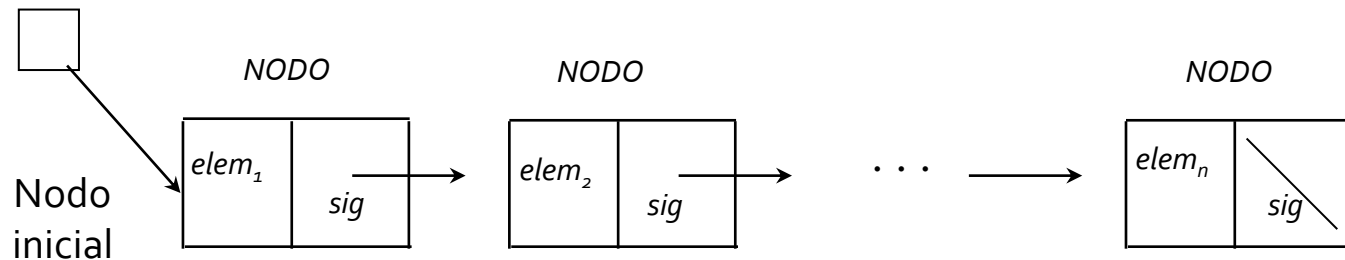


Doblemente vinculadas:

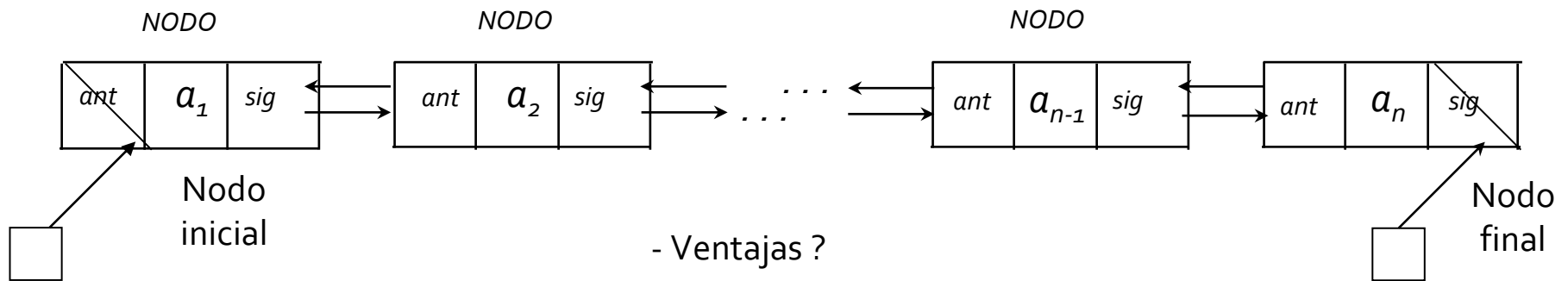


Lista vinculada

Simplemente vinculadas (son las que vimos hasta el momento):



Doblemente vinculadas:



- Ventajas ?
- Desventajas ?
- Cómo se puede implementar en una clase DoublyLinkedList ?

Operación	Array	Lista vinculada	
		Simple	Doble
fin	$O(1)$	$O(1)$	$O(1)$
primero	$O(1)$	$O(1)$	$O(1)$
sgte	$O(1)$	$O(1)$	$O(1)$
anterior	$O(1)$	$O(n)$	$O(1)$
vacía	$O(1)$	$O(1)$	$O(1)$
longitud	$O(1)$	$O(1)$	$O(1)$
insertar ppio	$O(n)$	$O(1)$	$O(1)$

Las operaciones en rojo logramos que sean $O(1)$ si utilizamos variables o punteros auxiliares. Por ejemplo, obtener la longitud de una lista vinculada será $O(1)$ si llevamos en la clase Lista una variable que actúe de contador, la cual debemos mantener actualizada. O manteniendo un puntero al último nodo de la lista.