PROGRAMACION 3

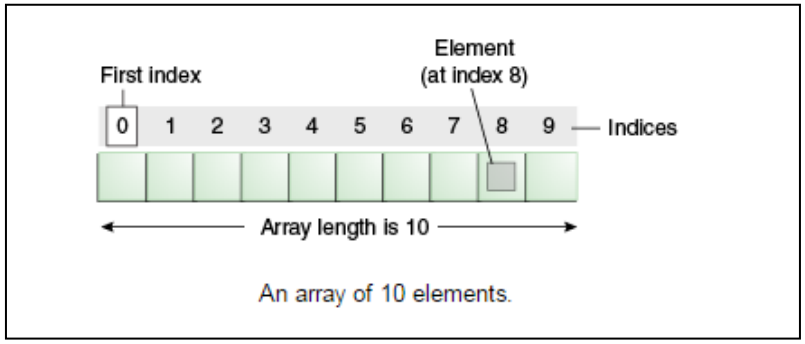
TUDAI

CURSADA 2024

Federico Casanova
Facultad de Ciencias Exactas - UNICEN

Iniciar Grabación

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:

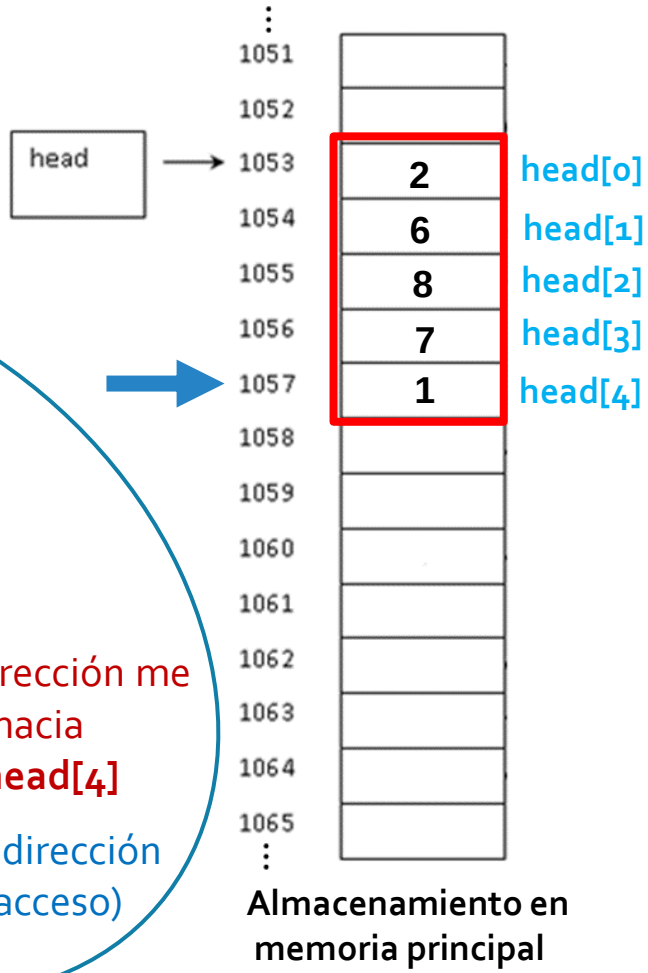
int[] head = new int[5];
head[0] = 2;
head[1] = 6;
head[2] = 8;
....

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 ahi estará **head[4]**

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



Almacenamiento en memoria principal

Complejidad Computacional (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). Nos va a ayudar a hacer buenos diseños.

- Llamaremos **n** como concepto general al "tamaño de la entrada".
- **Pensemos en ese n como lo que hace que la operación o algoritmo tarde más o menos tiempo en finalizar.**

- 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 supone el peor caso (pesimista).**
- Nos indica cómo se comporta la operación o el algoritmo en el peor caso respecto al tamaño de la entrada **n**. ¿Cómo se escribe ? Diremos que tal operación es **O(de algo en función de n)**.

notación Big-O

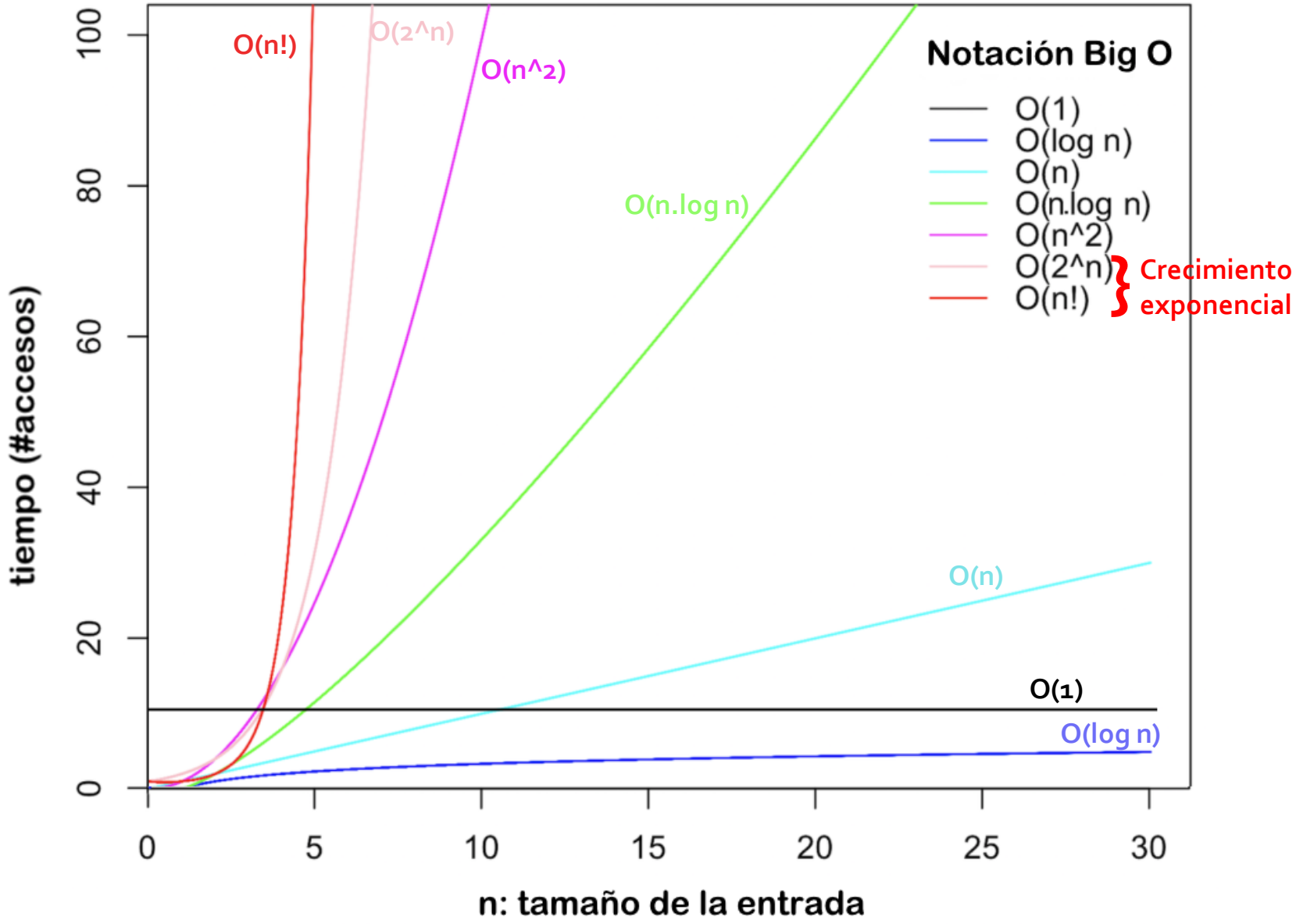
- Por ejemplo

- O(n) nos indica que el tiempo de ejecución que insume en el peor caso (eepc) es proporcional n.
- O(n^2) nos indica que el tiempo de ejecución que insume (eepc) es proporcional a n^2
- O(log n) nos indica que el tiempo de ejecución que insume (eepc) es proporcional a log n.
- **O(1)** es algo especial ya que indica que el tiempo no depende del tamaño de la entrada **n**, o sea la operación o algoritmo siempre tarda lo mismo, un tiempo constante independiente de **n**.
- ....
- En esta notación no se tienen en cuenta los términos o factores constantes dentro de O() asi es que por ejemplo

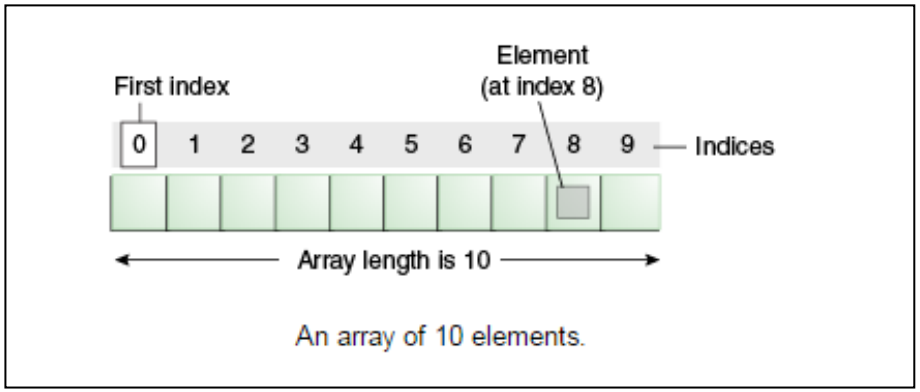
  O(n/2) , O(2.n) , O(100.n) , O(n+2) , O(n-25) , O(n-1) son todas O(n)

- 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*

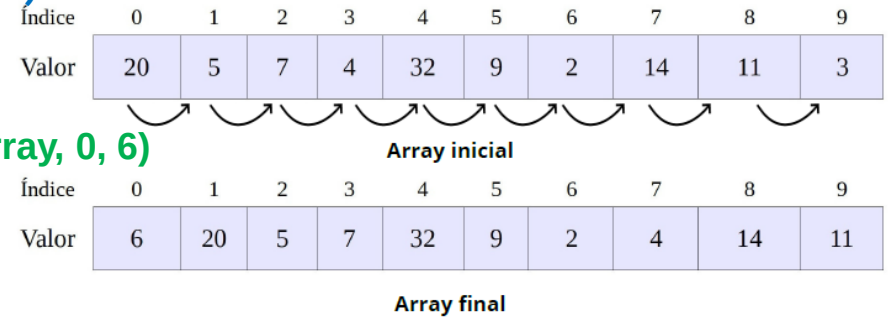
// valor por defecto 0

unArray = new int[10];

Notación Big O (peor caso)

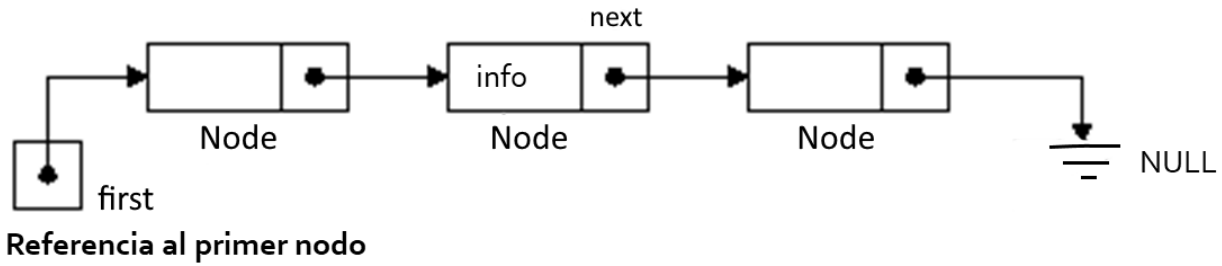
- Es una secuencia en un **bloque de memoria contiguo**.
- acceso directo mediante el operador [ ], por ej: ***int k = unArray[10];*** **O(1)** insume tiempo constante (siempre lo mismo, 1 acceso).
  (en un sólo acceso a memoria obtengo elemento en el índice 10)
- ¿tamaño ? Debe declararse al momento 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 ?
  O(n) Generar nuevo array, copiar el anterior.

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* al siguiente que sirve para encadenar dicho nodo con el siguiente de la lista**.




```
public class MySimpleLinkedList<T> {

        private Node<T> first;

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

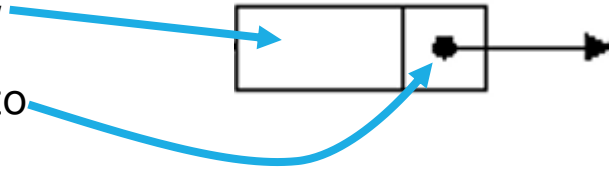
```
public class Node<T> {

        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);
        }
....
```

Lista vinculada

Cada nodo tiene al menos dos campos:

- un campo de información o valor,

- la referencia al siguiente elemento

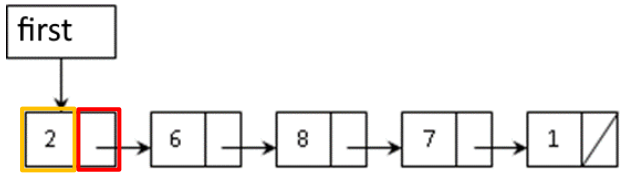


No es necesario que los nodos de la lista sean almacenados en posiciones físicas adyacentes en la memoria, la referencia (puntero) al siguiente nodo indica dónde se encuentra dicho nodo en la memoria.

Una ventaja de esta estructura es que va creciendo o achicándose a medida que necesitamos (agregando o borrando nodos).

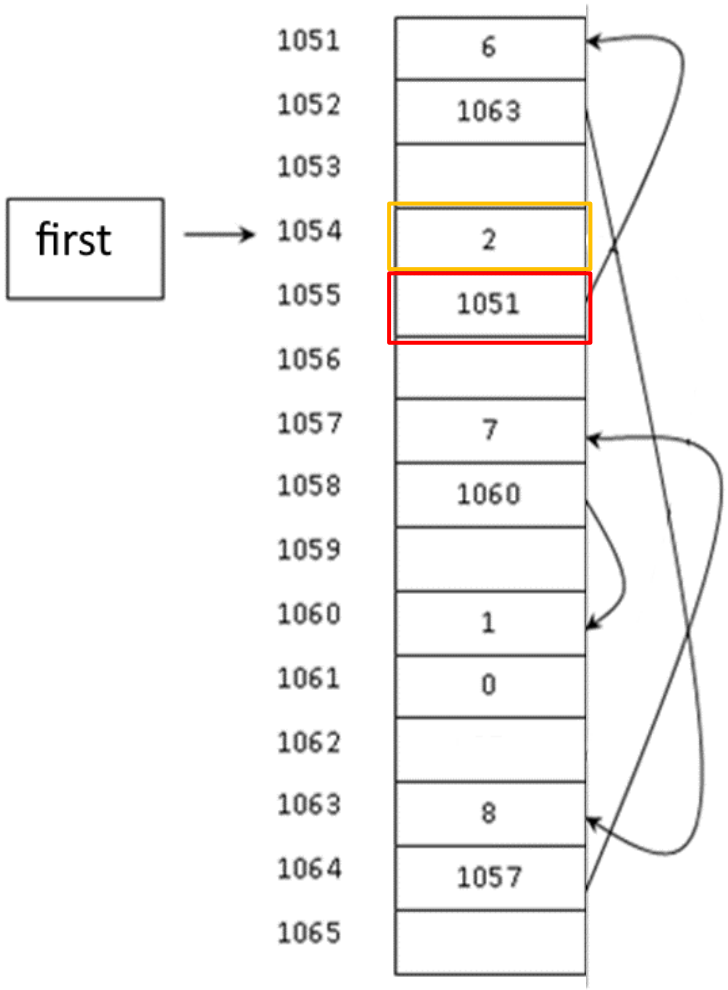
Una lista vinculada sin ningún nodo se llama lista vacía, en este caso la variable que apunta al primer nodo tiene el valor nulo (null).

Lista vinculada



Representación de una lista vinculada de nros. enteros

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



Almacenamiento en memoria principal

Cantidad de accesos para obtener el cuarto elemento?

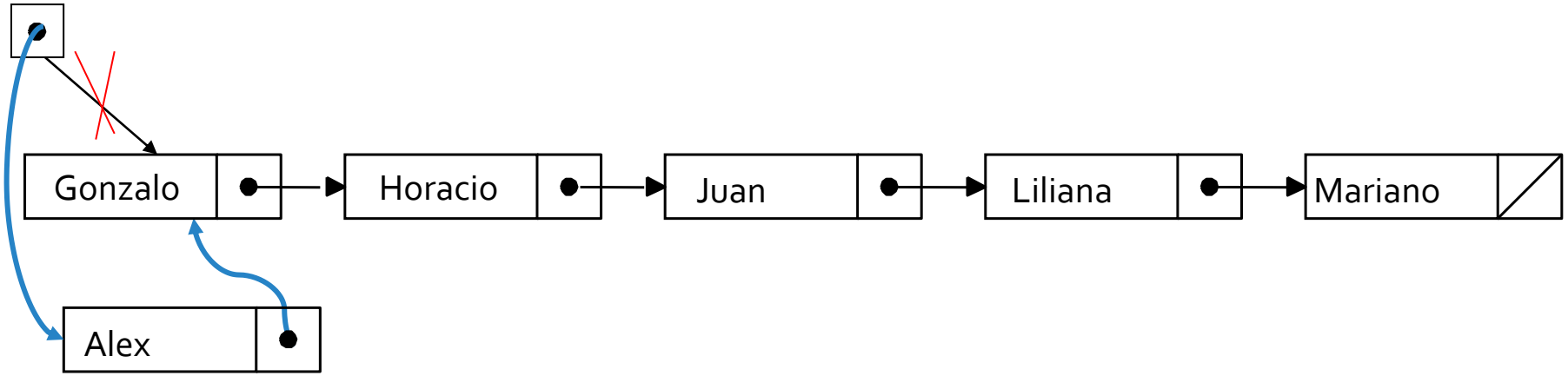
Y para obtener el último?
Y si tengo n elementos?

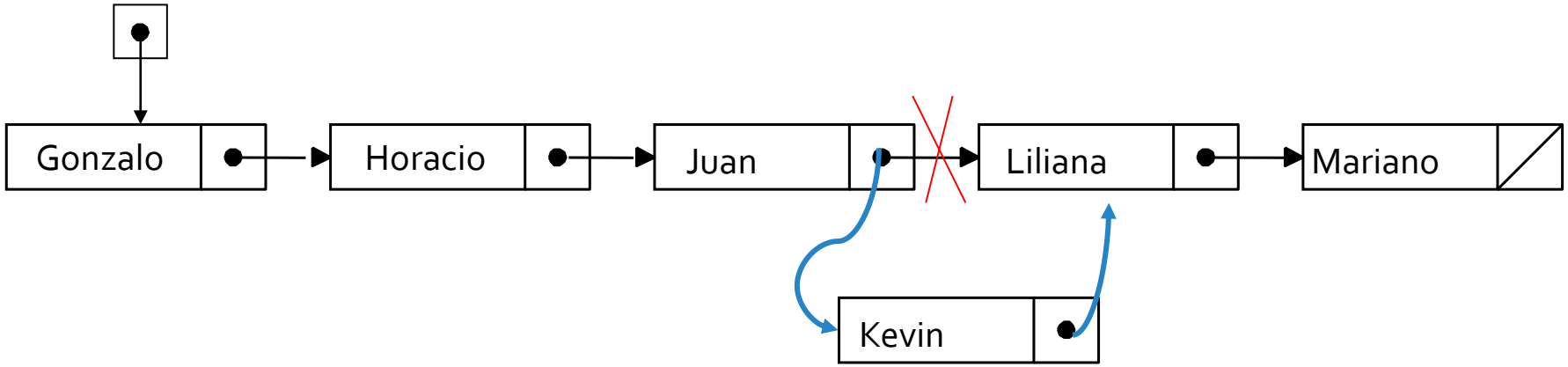
Obtener el valor en una posición dada es O(n)

Lista vinculada - Operaciones

Inserción de elementos

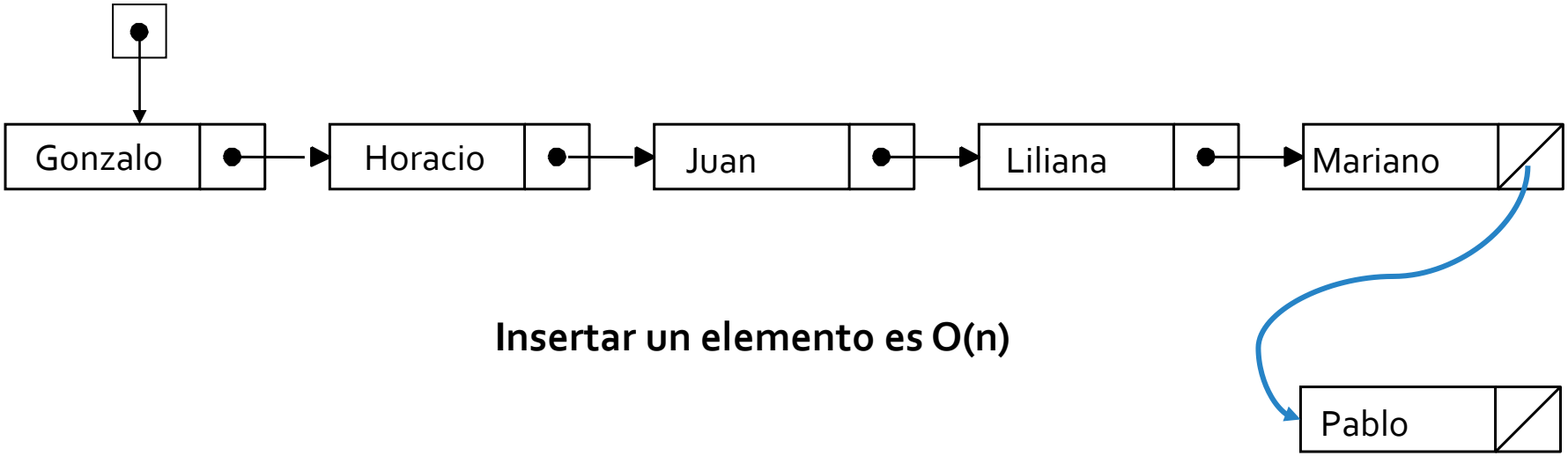
Supongamos que queremos mantener la lista de nombres ordenada.





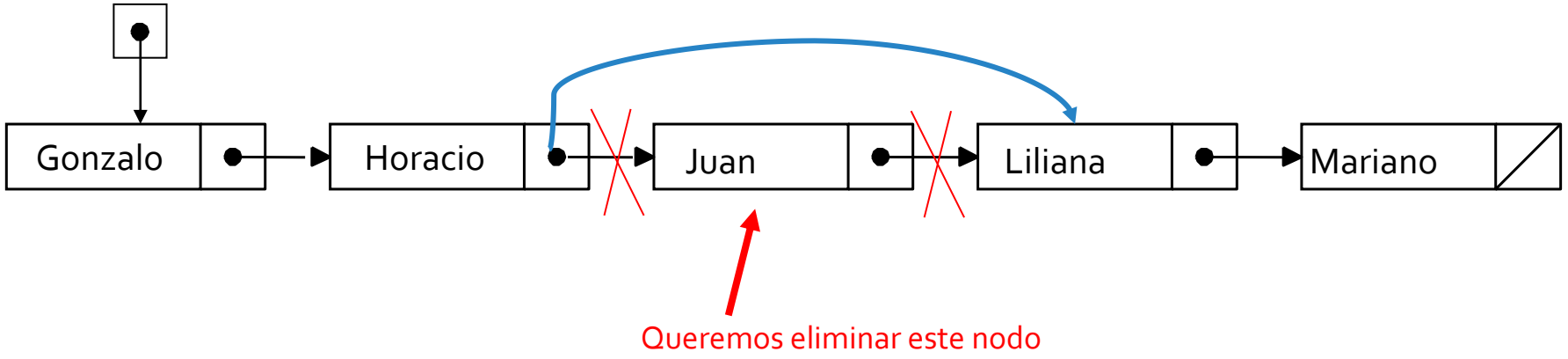
Lista vinculada - Operaciones

Inserción de elementos

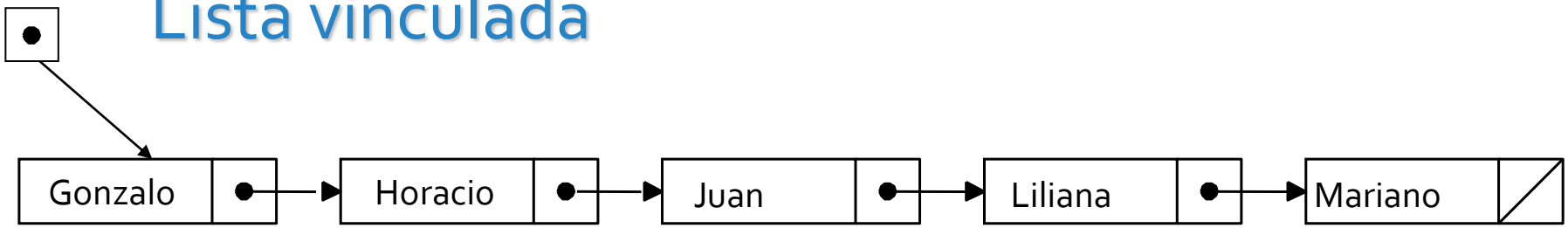


Insertar un elemento es O(n)

Borrado de elementos



Lista vinculada



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

```
public class MySimpleLinkedList<T> {
  private Node<T> first;
  private int size;

 public MySimpleLinkedList() {
    this.first = null;
    this.size = 0;
   }

 public void insertFront(T info) {
     Node<T> tmp = new Node<T>(info, null);
     tmp.setNext(this.first);
     this.first = tmp;
     this.size = this.size +1;
    }

  public int size () {
    return this.size;
   }
```

O(1) ESTO ES UNA EXCELENTE DECISION DE DISEÑO !!

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

Definiendo una lista vinculada en JAVA

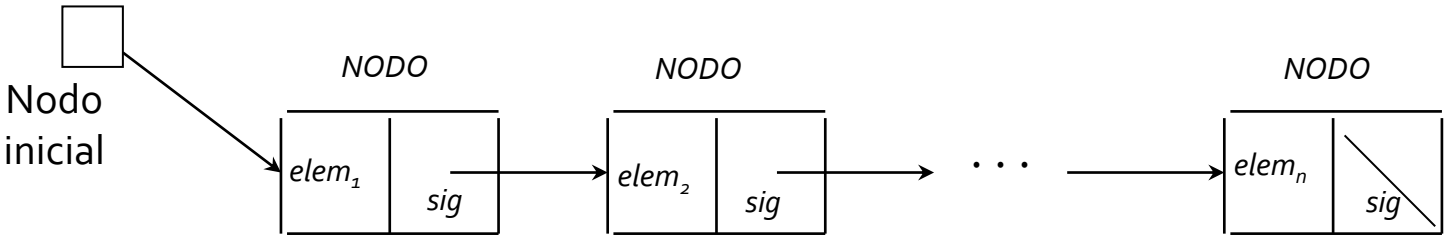
```java
public class Node<T> {

	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;
	}

}
```

- Otros métodos ?

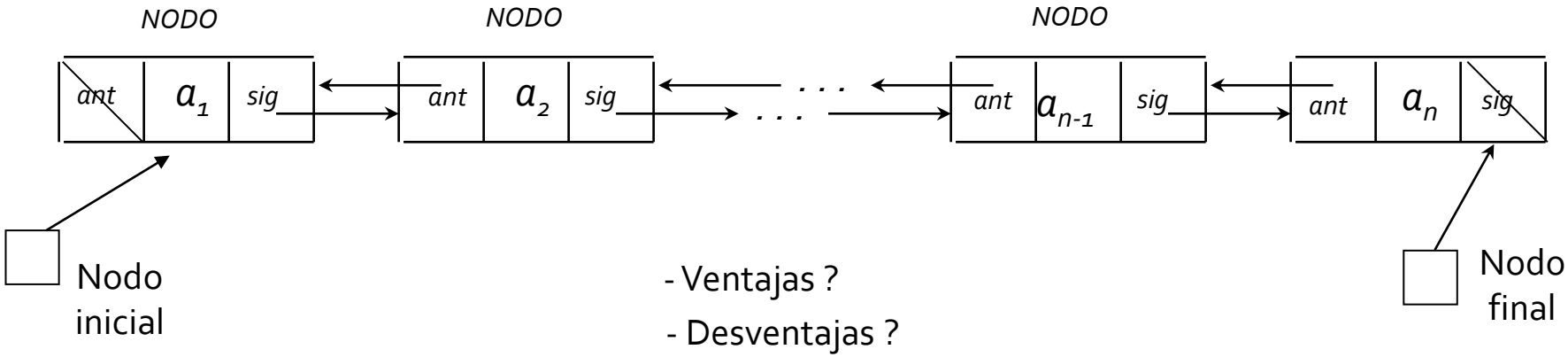
```java
public class MySimpleLinkedList<T> {

	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() {
		// TODO
		return null;
	}

	public boolean isEmpty() {
		// TODO
		return false;
	}

	public T get(int index) {
		// TODO
		return null;
	}

	public int size() {
		// TODO
		return 0;
	}

	@Override
	public String toString() {
		// TODO
		return "";
	}

}
```

Lista vinculada

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



Doblemente vinculadas:



- Ventajas ?
- Desventajas ?

Operación	Array	Lista vinculada	
		Simple	Doble
último	*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)*
vacia	*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 (size), la cual debemos mantener actualizada. O una variable manteniendo una referencia al último nodo de la lista para la operación último.