



Algoritmos y Estructuras de Datos

**Cursada 2017
Redictado**

Prof. Alejandra Schiavoni (ales@info.unlp.edu.ar)

Prof. Catalina Mostaccio (catty@lifa.info.unlp.edu.ar)

Prof. Pablo Iuliano (piuliano@info.unlp.edu.ar)

Agenda

- ❖ Temas de la materia
- ❖ Objetivos de la materia
- ❖ Introducción al Análisis de Algoritmos
- ❖ Repaso de Listas

Temas del curso

- Árboles
- Cola de Prioridades
- Análisis de Algoritmos
- Grafos

Objetivos de la materia

- Analizar Algoritmos y evaluar su eficiencia
- Estudiar estructuras de datos avanzadas: su implementación y aplicaciones

¿De qué se trata el curso?

Estudiar formas **inteligentes** de organizar la información, de forma tal de obtener Algoritmos **eficientes**.

Listas, Pilas, Colas

Árboles Binarios

Árboles AVL

Árboles Generales

Heaps

Grafos

Estructuras de Datos

Insertar

Borrar

Buscar

Caminos mínimos

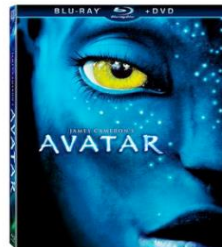
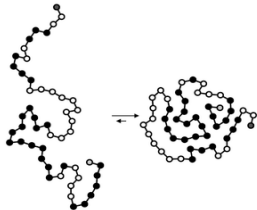
Ordenación

Algoritmos

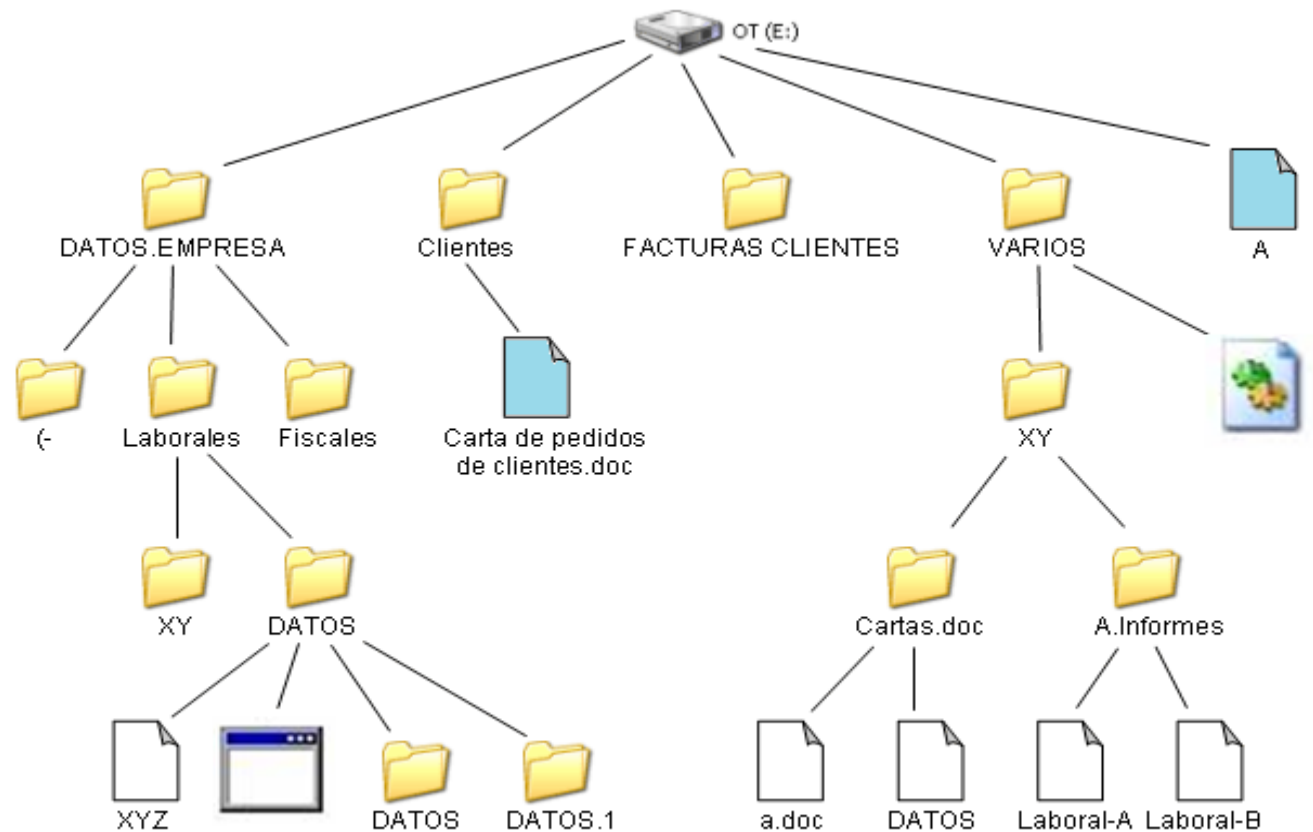
Las estructuras de datos y sus Algoritmos son....



Google
YAHOO!
bing



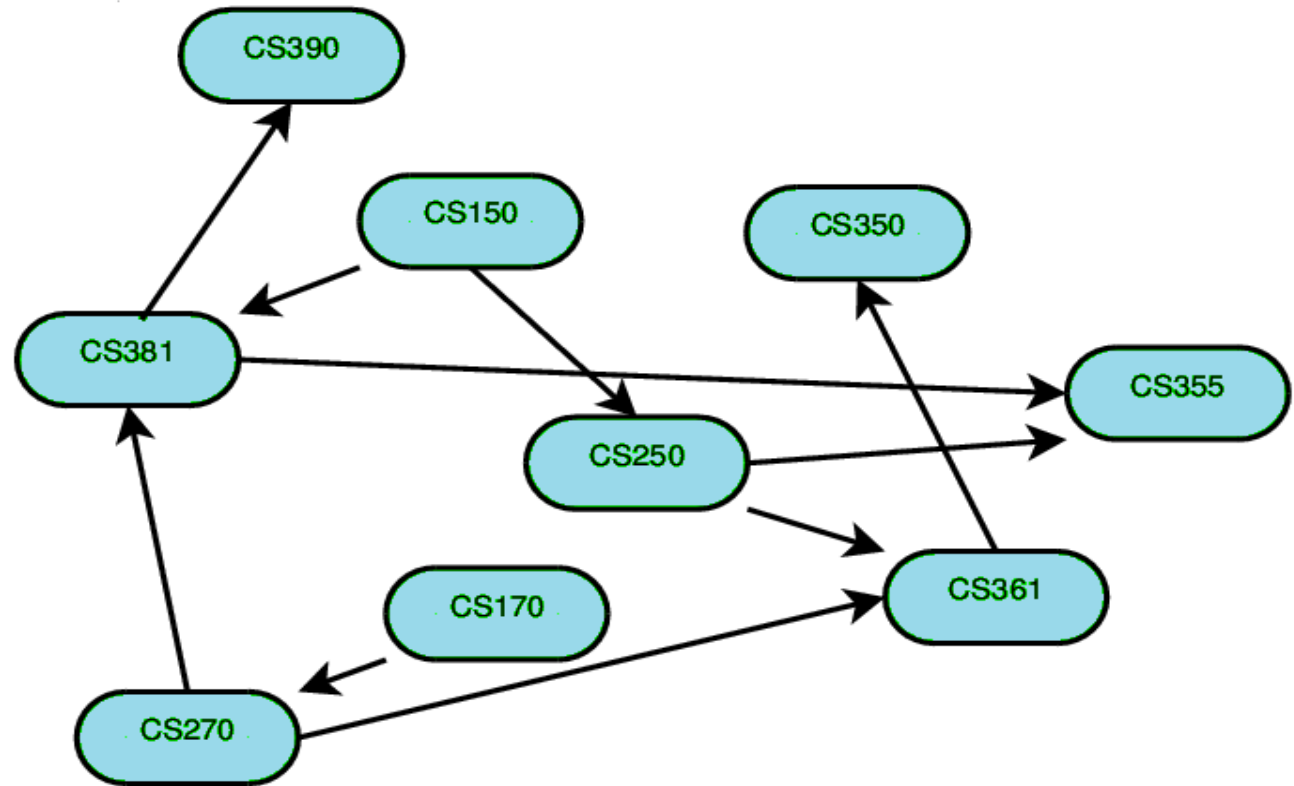
Ejemplo 1: Árbol de carpetas y archivos



Nodos: Carpetas/Archivos

Aristas: representan la relación “contiene”

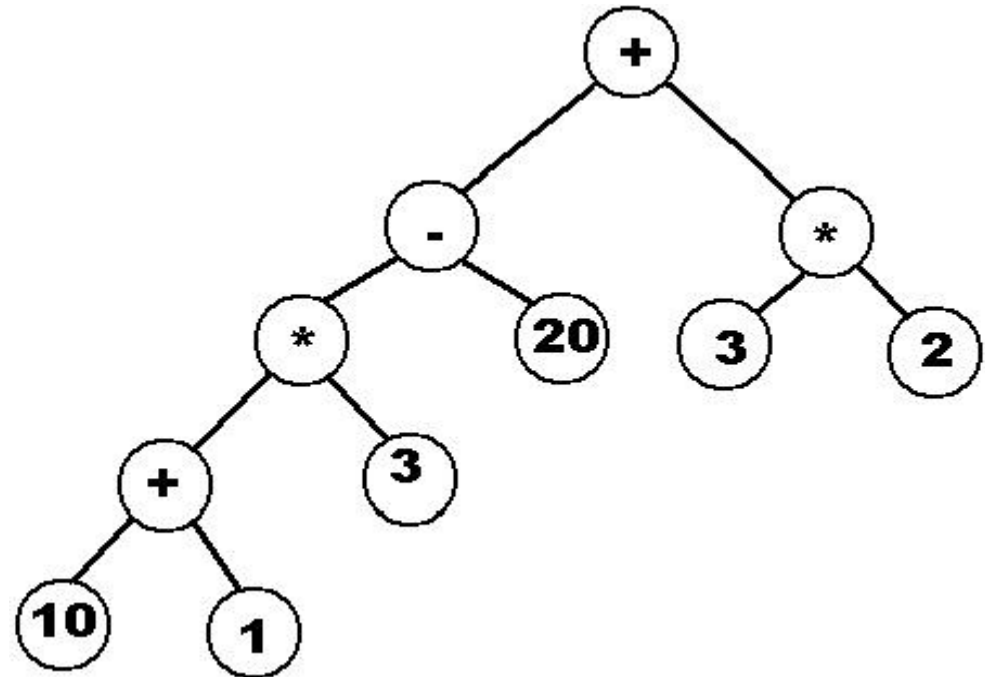
Ejemplo 2: Prerrequisitos de un curso



Nodos: Cursos

Aristas: relación de "prerrequisito"

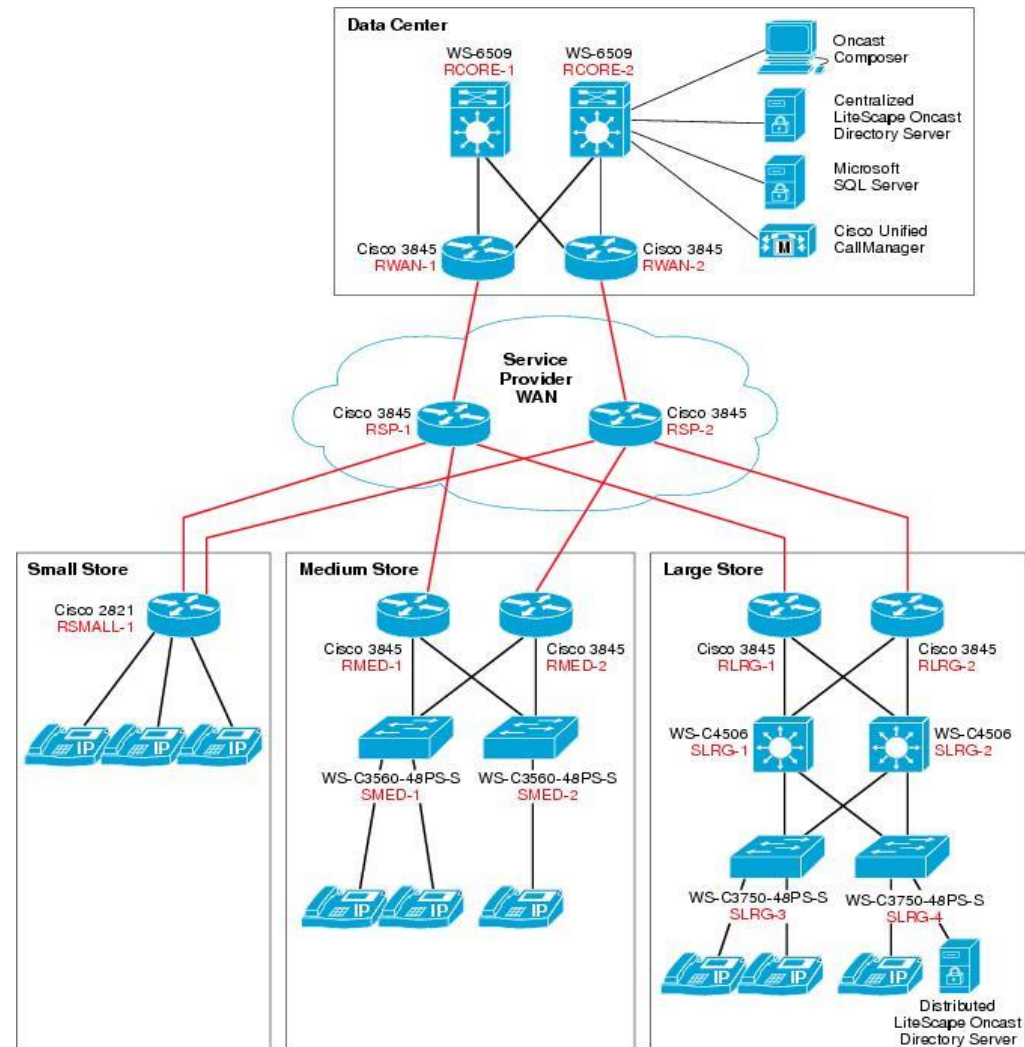
Ejemplo 3: Representación de una expresión en un compilador



Nodos: Operandos/Operadores

Aristas: representan las relaciones entre las operaciones

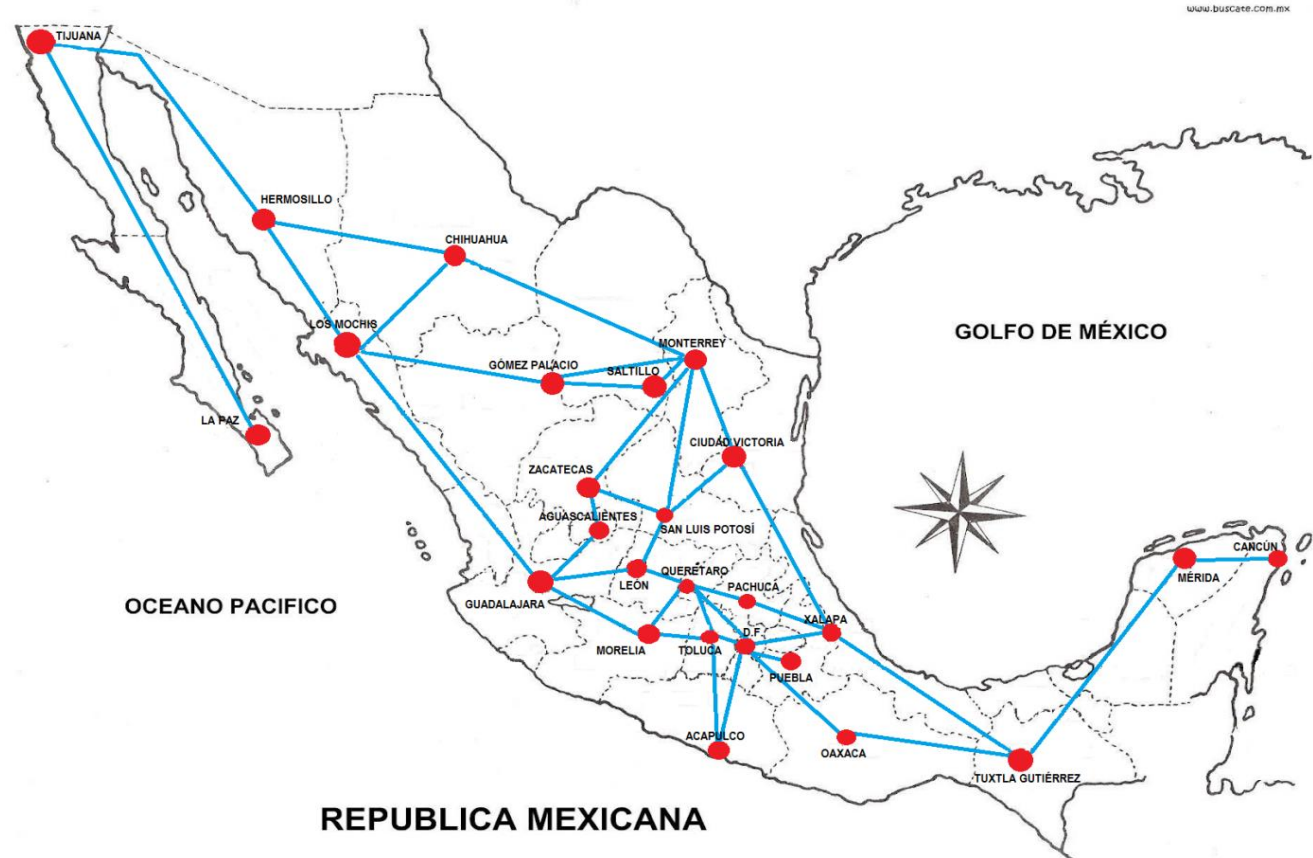
Ejemplo 4: Esquema de una red informática



Nodos: Equipos

Aristas: representan las conexiones

Ejemplo 5: Mapa de ciudades

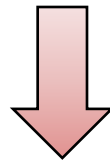


Nodos: Ciudades

Aristas: Rutas

Estructuras de Datos: Qué, Cómo y Por qué?

- Los programas reciben, procesan y devuelven datos
- Necesidad de organizar los datos de acuerdo al problema que vamos a resolver



Las estructuras de datos son formas de organización de los datos

Estructuras de Datos: Qué, Cómo y Por qué?

- Un programa depende fundamentalmente de la organización de los datos
- cómo se organizan los datos está relacionado con:
 - ➡ Implementación de algunas operaciones: pueden resultar más fácil o más difícil
 - ➡ La velocidad del programa: puede aumentar o disminuir
 - ➡ La memoria usada: puede aumentar o disminuir

Objetivos del curso respecto de las Estructuras de Datos

- Aprender a implementar las estructuras de datos usando abstracción
- Estudiar diferentes representaciones e implementaciones para las estructuras de datos
- Aprender a elegir la “mejor” estructura de datos para cada problema

Algoritmos y su Análisis

- ¿Qué es un Algoritmo?
 - ➡ Es una secuencia de pasos que resuelven un problema
 - ➡ Es independiente del lenguaje de programación
- Existen varios Algoritmos que resuelven correctamente un problema
- La elección de un Algoritmo particular tiene un enorme impacto en el tiempo y la memoria que utiliza

La elección de un Algoritmo y de la estructura de datos para resolver un problema son interdependientes

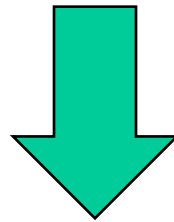
Objetivos del curso respecto del Análisis de los Algoritmos

- Entender los fundamentos matemáticos necesarios para analizar Algoritmos
- Aprender a comparar la eficiencia de diferentes Algoritmos en términos del tiempo de ejecución
- Estudiar algunos Algoritmos estándares para el manejo de las estructuras de datos y aprender a usarlos para resolver nuevos problemas

Problemas y Algoritmos

➤ Problemas:

- Buscar un elemento en un arreglo
- Ordenar una lista de elementos
- Encontrar el camino mínimo entre dos puntos



Encontrar **el Algoritmo** que lo resuelve

Caso: Buscar un elemento en un arreglo

El arreglo puede estar:

- desordenado
- ordenado

Si el arreglo está desordenado  **Búsqueda secuencial**

64	13	93	97	33	6	43	14	51	84	25	53	95
0	1	2	3	4	5	6	7	8	9	10	11	12



Algoritmo: Búsqueda secuencial

```
public static int seqSearch(int[] a, int
    key)
{
    int index = -1;
    for (int i = 0; i < N; i++)
        if (key == a[i])
            index = i;
    return index;
}
```

¿Cuántas comparaciones hace?

Caso: Buscar un elemento en un arreglo

El arreglo puede estar:



- desordenado
- ordenado

Si el arreglo está ordenado 

Búsqueda binaria: Comparo la clave con la entrada del centro

- Si es menor, voy hacia la izquierda
- Si es mayor, voy hacia la derecha
- Si es igual, la encontré

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

 lo  mid  hi

Algoritmo: Búsqueda binaria

```
public static int binarySearch(int[] a, int key)
{
    int lo = 0, hi = a.length-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

¿Cuántas comparaciones hace?

¿Cuántas operaciones hace cada Algoritmo?

**Búsqueda
secuencial**

N	Cantidad de operaciones
1000	1000
2000	2000
4000	4000
8000	8000
16000	16000



Hace N operaciones

**Búsqueda
binaria**

N	Cantidad de operaciones
1000	~10
2000	~11
4000	~12
8000	~13
16000	~14



Hace $\log(N)$ operaciones

¿Cómo medir el tiempo?



✓ Manual

Tomando el tiempo que tarda

✓ Automática

Usando alguna instrucción del lenguaje para medir tiempo

Análisis empírico

Correr el programa para varios tamaños de la entrada y medir el tiempo. Suponemos que cada comparación tarda 1 seg.

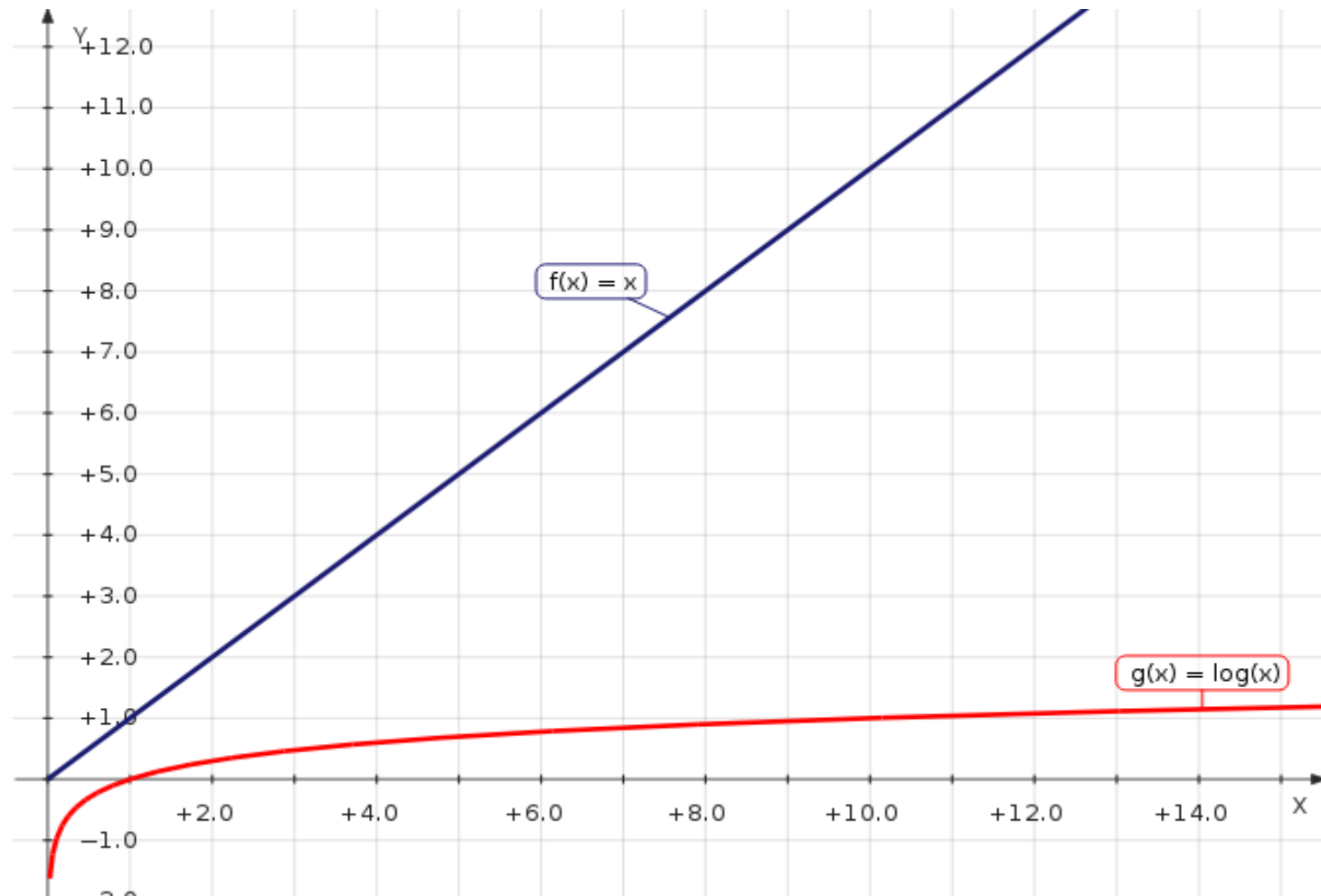
**Búsqueda
secuencial**

N	Tiempo (seg)
1000	1000
2000	2000
4000	4000 ~ 1 hs.
8000	8000 ~ 2 hs
16000	16000 ~ 4 hs.

**Búsqueda
binaria**

N	Tiempo (seg)
1000	~10
2000	~11
4000	~12
8000	~13
16000	~14

Análisis de Algoritmos



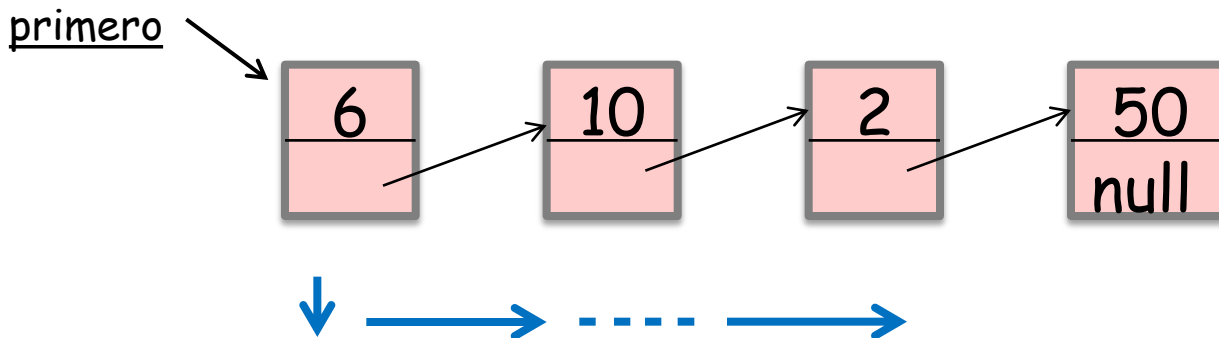
Caso: Buscar un elemento en una lista dinámica

Si los elementos están almacenados en una lista dinámica

La lista puede estar:

- desordenada
- ordenada

¿Cómo sería el Algoritmo de búsqueda?



¿Cuántas
comparaciones
hace?



Hace N comparaciones

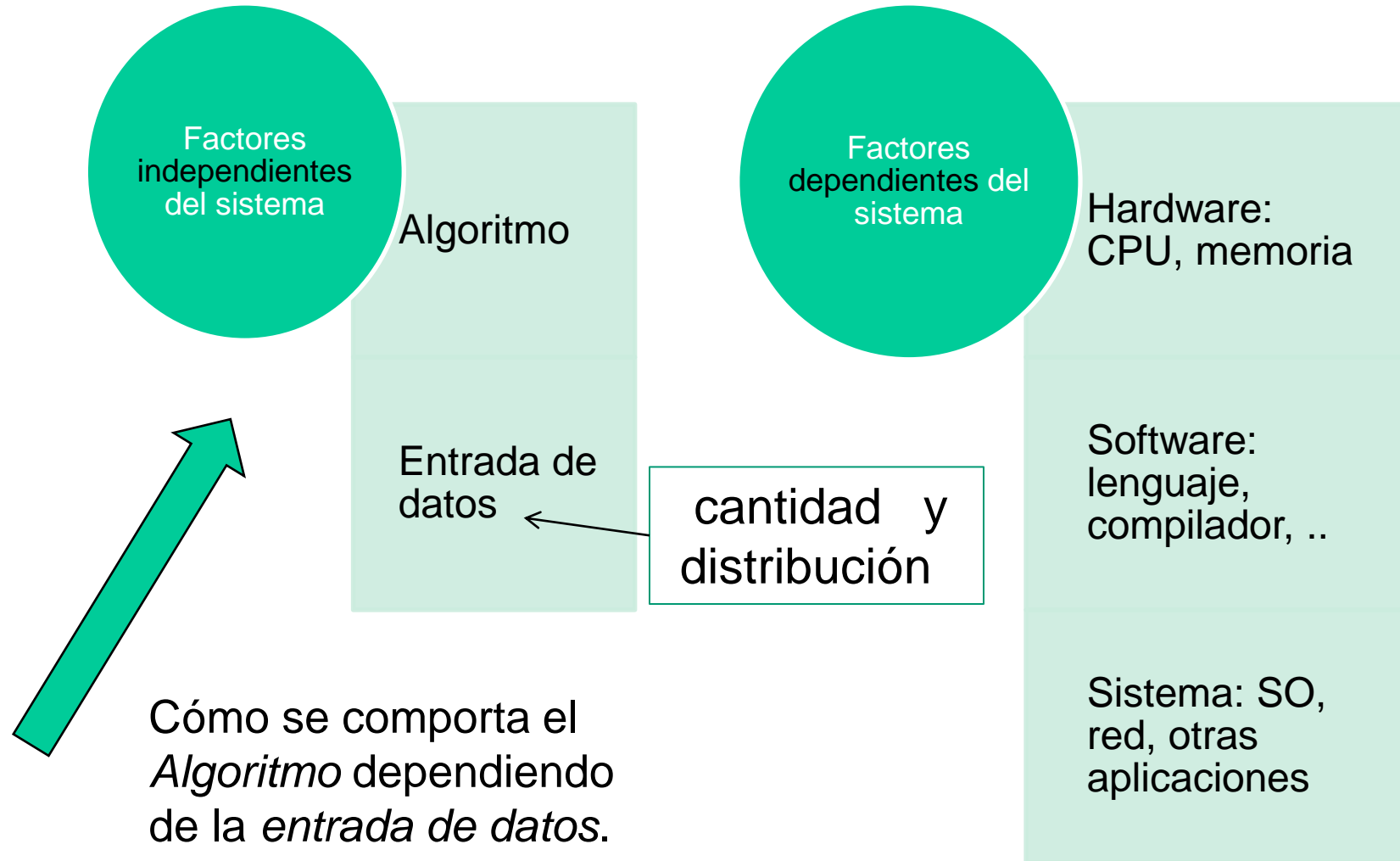
Análisis de Algoritmos

***Marco para predecir la performance y
comparar Algoritmos***

Desafío:

Escribir programas que puedan resolver en forma eficiente problemas con una gran entrada de datos

Análisis de Algoritmos



Análisis de Algoritmos

Existe un modelo matemático para medir el tiempo

Tiempo total:

*Suma del **costo** x **frecuencia** de todas las operaciones*

- Es necesario analizar el Algoritmo para determinar el conjunto de operaciones
 - **Costo** depende de la máquina, del compilador, del lenguaje
 - **Frecuencia** depende del Algoritmo y de la entrada

Análisis de Algoritmos

Mejor Caso

—————→ Cota inferior para el costo

- ✓ Determinado por la entrada “óptima”

Peor Caso

—————→ Cota superior para el costo

- ✓ Determinado por la “peor” entrada
- ✓ Provee una cota para todas las entradas

Ejemplo: Búsqueda binaria, número de operaciones

- Mejor caso: ~ 1
- Peor caso: $\sim \log(N)$

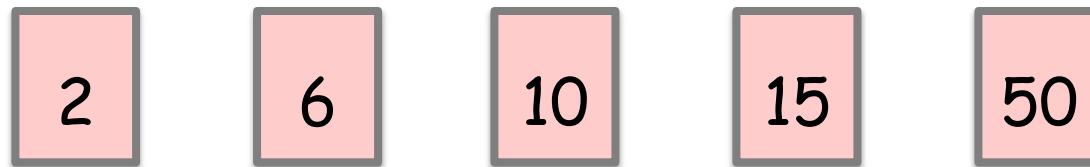
Repaso de Listas

- Una lista es una estructura de datos en donde los objetos están ubicados en forma secuencial.
- Se diferencian de la Pila y la Cola, en que en la **Lista** se puede “agregar” y “eliminar” en **cualquier** posición.
- Puede estar implementada a través de:
 - una estructura estática (arreglo)
 - una estructura dinámica (usando punteros)

Repaso de Listas

➤ *La lista puede estar ordenada o no*

➤ *Si está ordenada, los elementos se ubican siguiendo el orden de las claves almacenadas en la lista.*

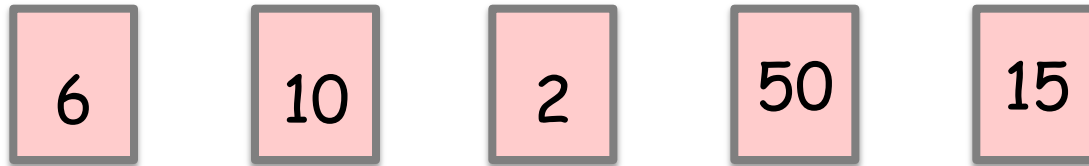


➤ *Si está desordenada, los elementos pueden aparecer en cualquier orden.*



Repaso de Listas

- Definiremos una especificación para Listas desordenadas



Listas: algunas operaciones

elemento(int pos): retorna el elemento de la posición indicada

incluye(Integer elem): retorna true si ***elem*** está en la lista, false en caso contrario

agregarInicio(Integer elem): agrega al inicio de la lista

agregarFinal(Integer elem): agrega al final de la lista

agregarEn(Integer elem, int pos): agrega el elemento ***elem*** en la posición ***pos***

eliminarEn(int pos): elimina el elemento de la posición ***pos***

eliminar(Integer elem): elimina, si existe, el elemento ***elem***

esVacia(): retorna true si la lista está vacía, false en caso contrario

tamano(): retorna la cantidad de elementos de la lista

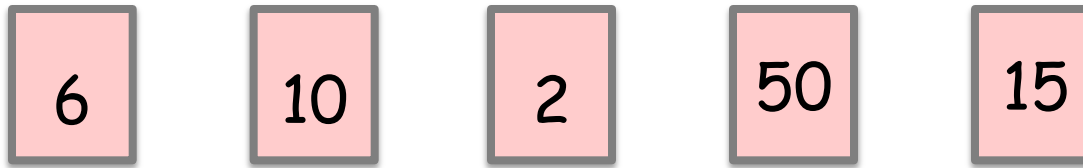
comenzar(): se prepara para iterar los elementos de la lista

proximo(): retorna el elemento y avanza al próximo elemento de la lista.

fin(): determina si llegó o no al final de la lista

Listas: algunas operaciones

elemento (int pos): retorna el elemento de la posición indicada por **pos**

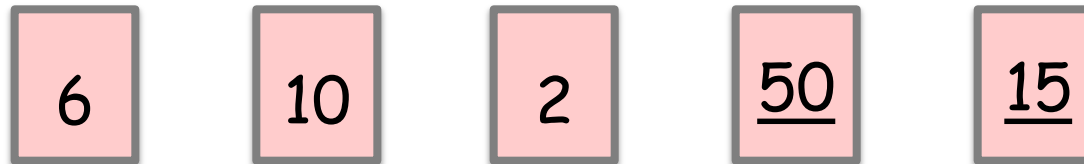


elemento (3) →

resultado:



incluye (Integer elem): retorna true si **elem** está contenido en la lista, false en caso contrario

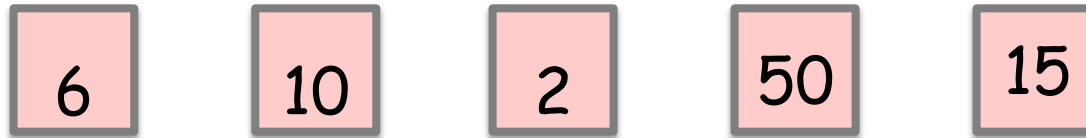


incluye (2) retornará “true”

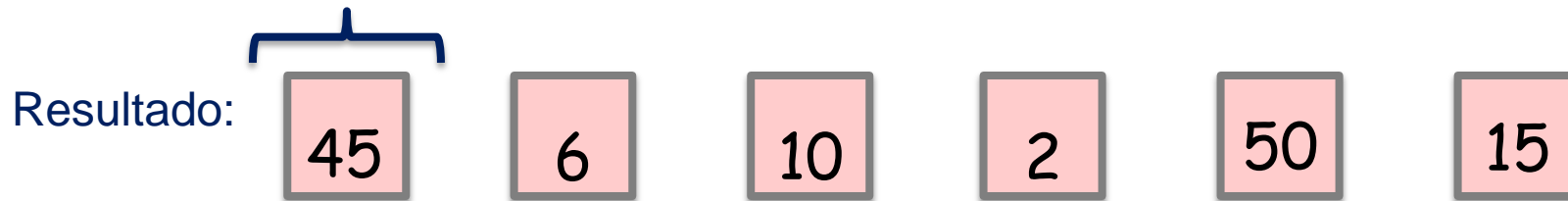
incluye (20) retornará “false”

Listas: algunas operaciones

agregarInicio(Integer elem): agrega el elemento **elem** al inicio de la lista

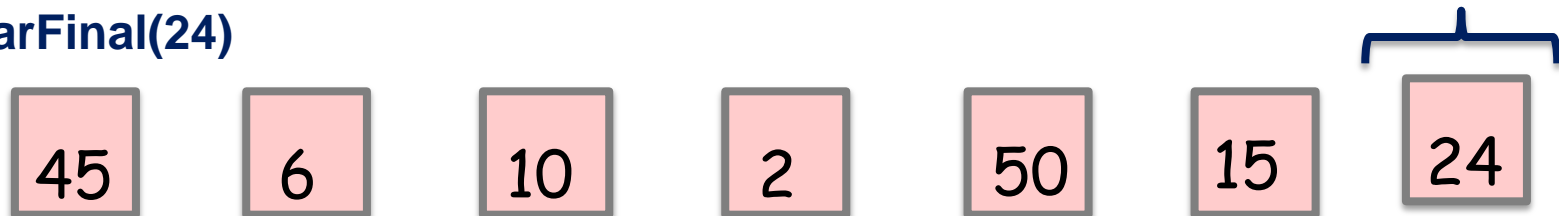


agregarInicio(45)



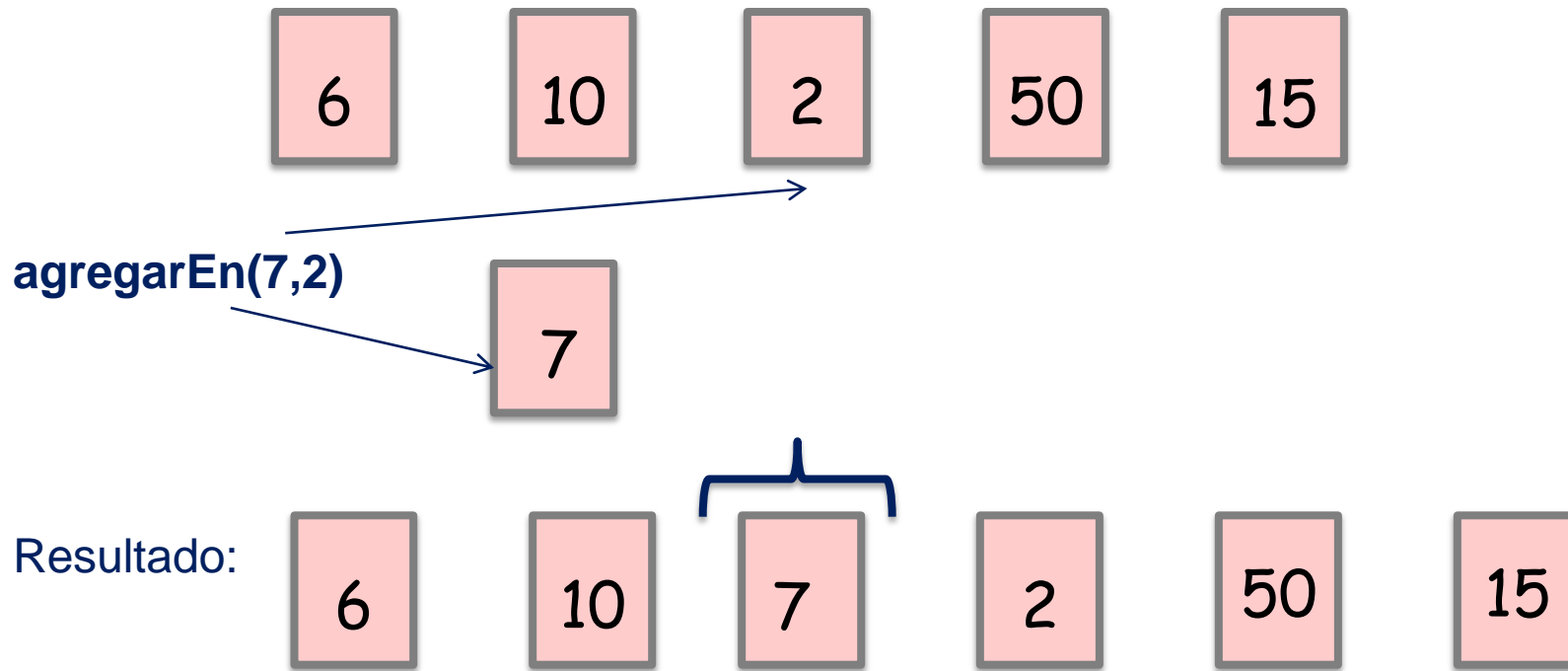
agregarFinal(Integer elem): agrega el elemento **elem** al final de la lista

agregarFinal(24)



Listas: algunas operaciones

agregarEn(Integer elem, int pos): agrega el elemento **elem** en la posición **pos**



agregarInicio y **agregarFinal** en términos de **agregarEn(Integer elem, int pos) ?**

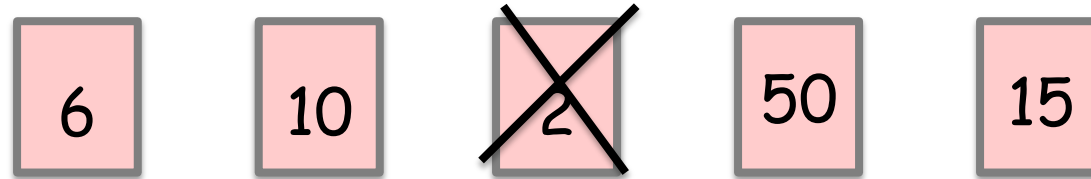
agregarEn(elem, 0)

agregarEn(elem, tamaño())

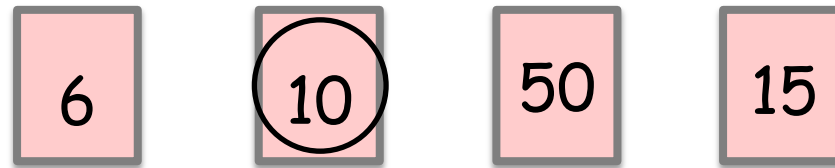
Listas: algunas operaciones

eliminarEn(int pos): elimina el elemento de la posición ***pos***

eliminarEn(2)



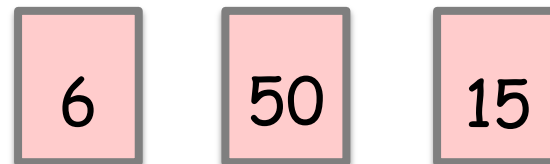
Resultado:



eliminar(Integer elem): elimina el elemento ***elem***

eliminar(10)

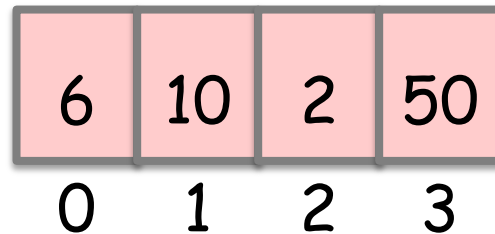
Resultado:



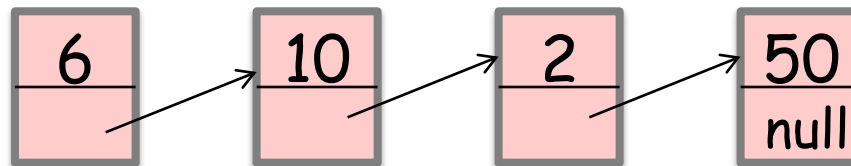
Listas: implementaciones

Una lista puede estar implementada a través de:

➤ *una estructura estática (arreglo)*



➤ *una estructura dinámica (nodos enlazados)*



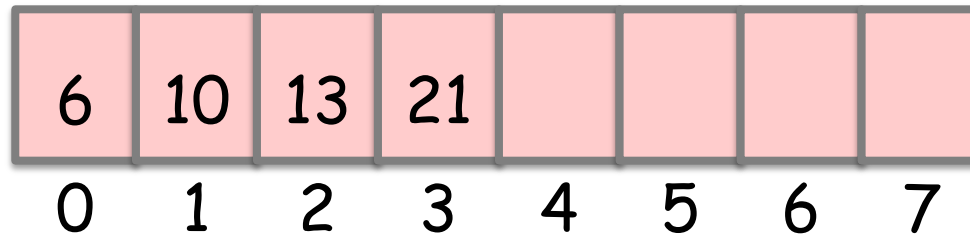
Listas: con estructura estática

agregarInicio(Integer elem): agrega el elemento **elem** al inicio de la lista

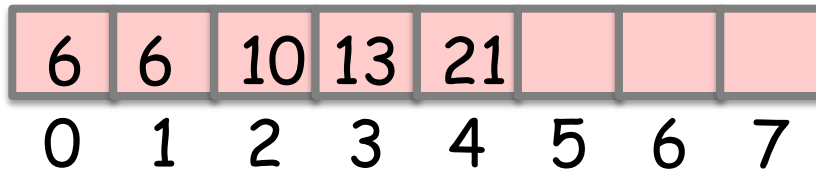
agregarInicio(11)

elem = 11
tamaño \leftarrow 4

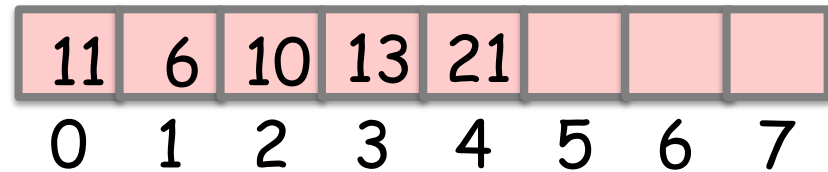
Estado Inicial



(1) Corrimiento a derecha

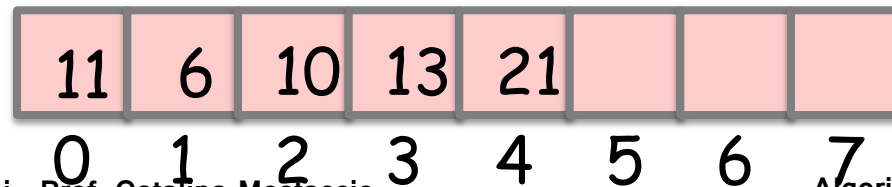


(2) inserción de **elem** en la posición 0
Incrementar **tamaño**



Estado Final

tamaño \leftarrow 5



Listas: *con estructura estática*

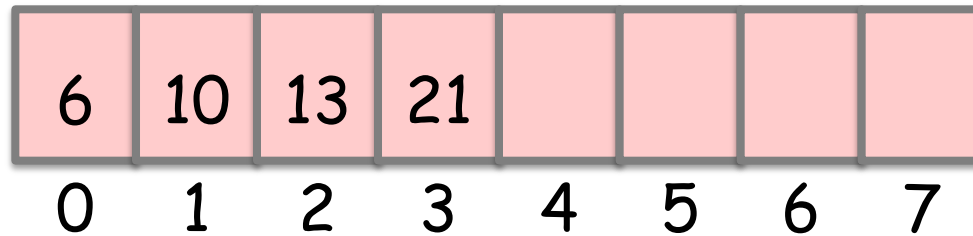
agregarFinal(Integer elem): agrega el elemento **elem** al final de la lista

agregarFinal(11)

Estado Inicial

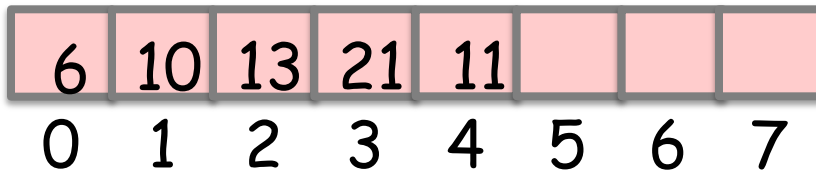
elem = 11

tamaño \leftarrow 4



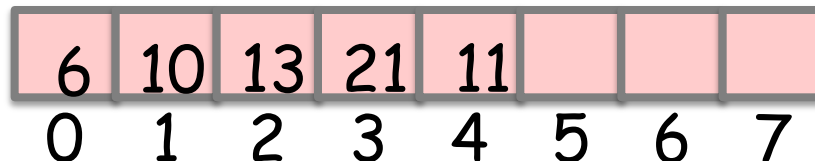
(1) agregar **elem** en la posición **tamaño**

(2) Incrementar tamaño de la lista:
tamaño = tamaño+1



Estado Final

tamaño \leftarrow 5



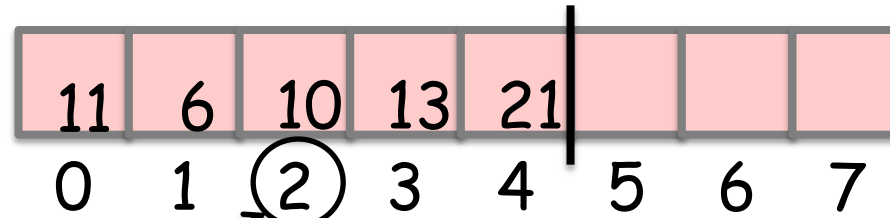
Listas: *con estructura estática*

eliminarEn(int pos): elimina el elemento de la posición **pos**

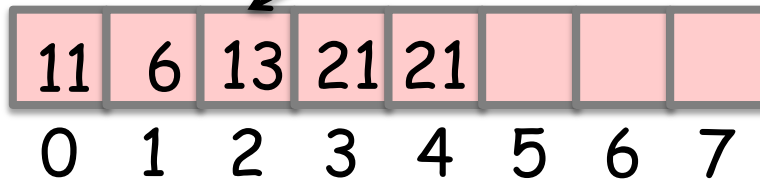
eliminarEn(2):

tamaño \leftarrow 5

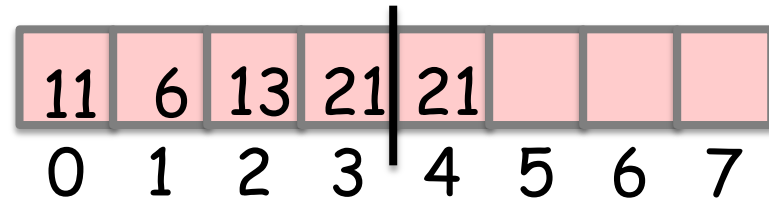
Estado Inicial



(1) Corrimiento a “izquierda” sobre la posición “3”

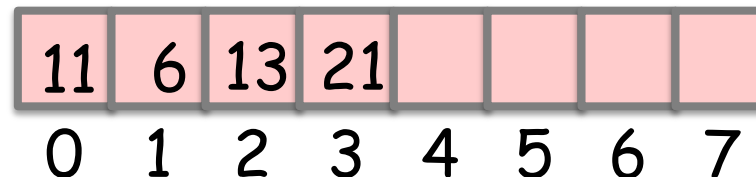


(2) Decrementar tamaño de la lista:
tamaño = tamaño - 1



Estado Final

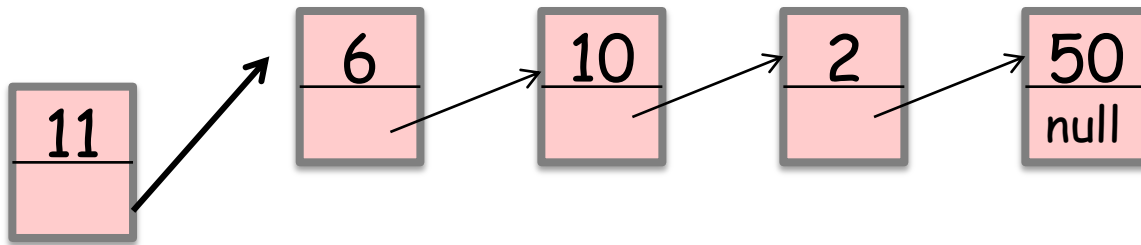
tamaño \leftarrow 4



Listas: *con estructura dinámica*

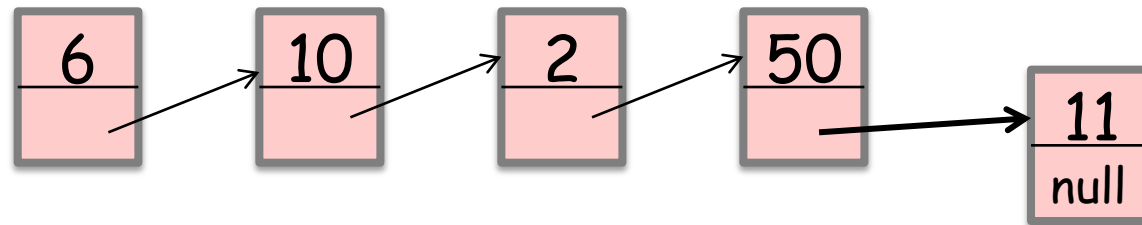
agregarInicio(Integer elem): agrega el elemento **elem** al inicio de la lista

agregarInicio(11)



agregarFinal(Integer elem): agrega el elemento **elem** al final de la lista

agregarFinal(11)



Listas: *con estructura dinámica*

eliminar(Integer elem): elimina el elemento ***elem*** indicado

eliminar(10)

