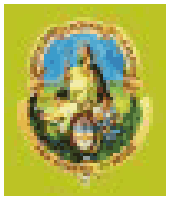


Algoritmos y Estructuras de Datos

Cursada 2014

Prof. Catalina Mostaccio
Prof. Alejandra Schiavoni

Facultad de Informática - UNLP

Objetivos de la materia

- Analizar algoritmos y evaluar su eficiencia
- Estudiar estructuras de datos dinámicas, tales como árboles y grafos: su implementación y aplicaciones

Agenda

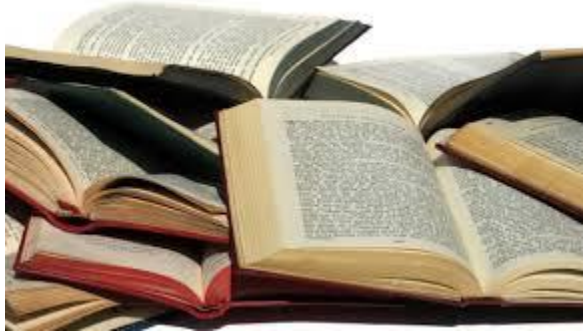
- Repaso de Listas
  - Especificación de operaciones
- Noción intuitiva del Análisis de algoritmos

Listas

Estructuras de Datos

Una estructura de datos es una forma de almacenar y organizar los datos con el fin de facilitar el acceso y las modificaciones.

Ejemplo:



Datos sin organización

Datos: libros

Datos organizados en una estructura

Estructura de datos: biblioteca

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)*

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.*

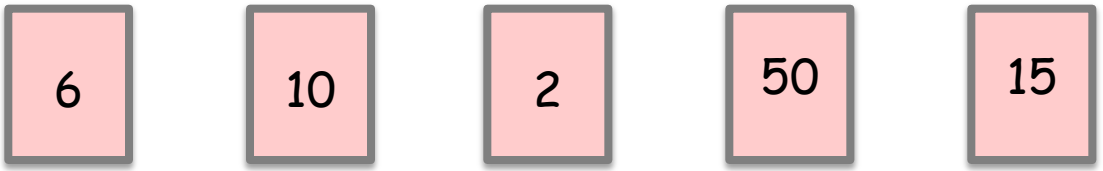
  Ej. 5 10 13 17 24 25 90

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

  Ej. 5 25 17 90 13 24 10

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 en la posición indicada

eliminarEn(int pos): Elimina el elemento de la posición indicada

eliminar(Integer elem): Elimina el elemento "elem" indicado

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

tamanio(): Retorna la longitud de la lista

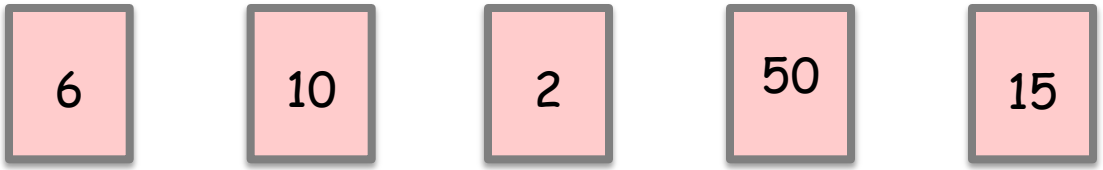
comenzar(): Se prepara para iterar los elementos de la lista

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

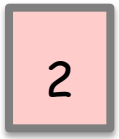
fin(): Determina si llegó o no al final de la lista

Listas: *algunas operaciones*

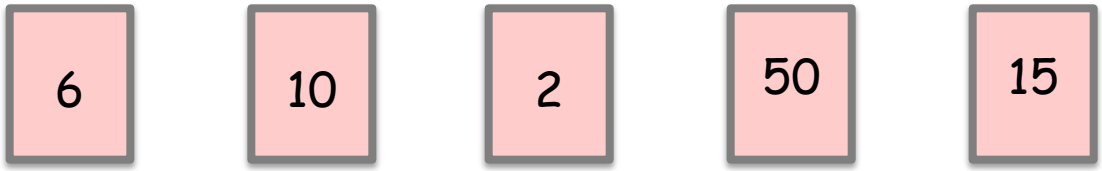
elemento (int pos): Retorna el elemento de la posición indicada



elemento (3) → resultado:



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



incluye (2) retornará “true”

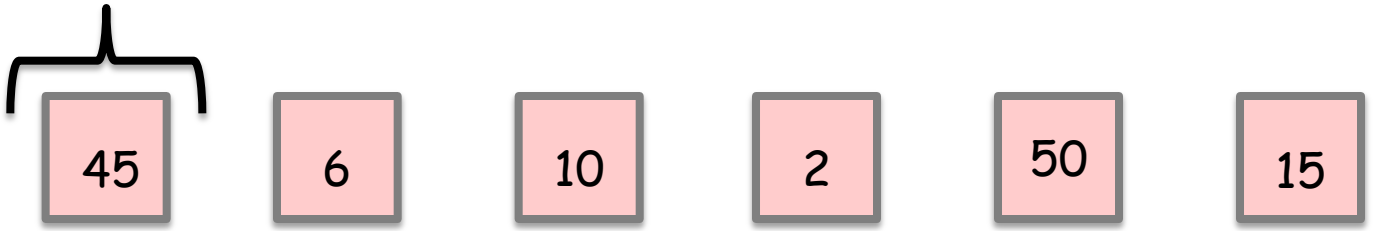
incluye (20) retornará “false”

Listas: *algunas operaciones*

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

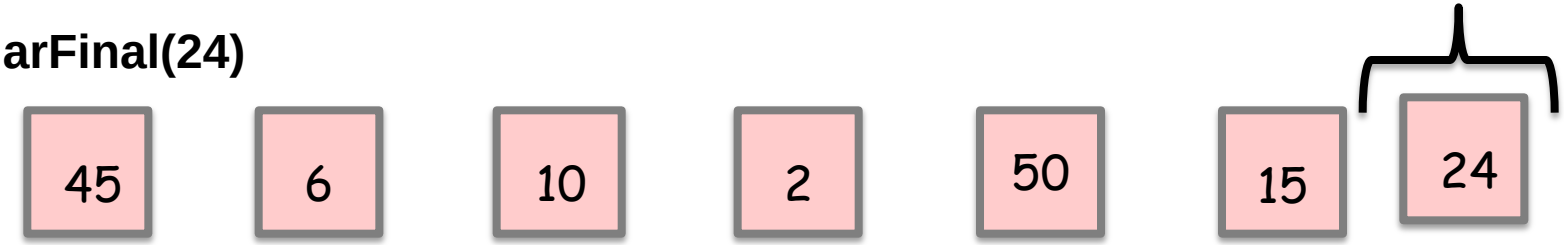
agregarInicio(45)

Resultado:



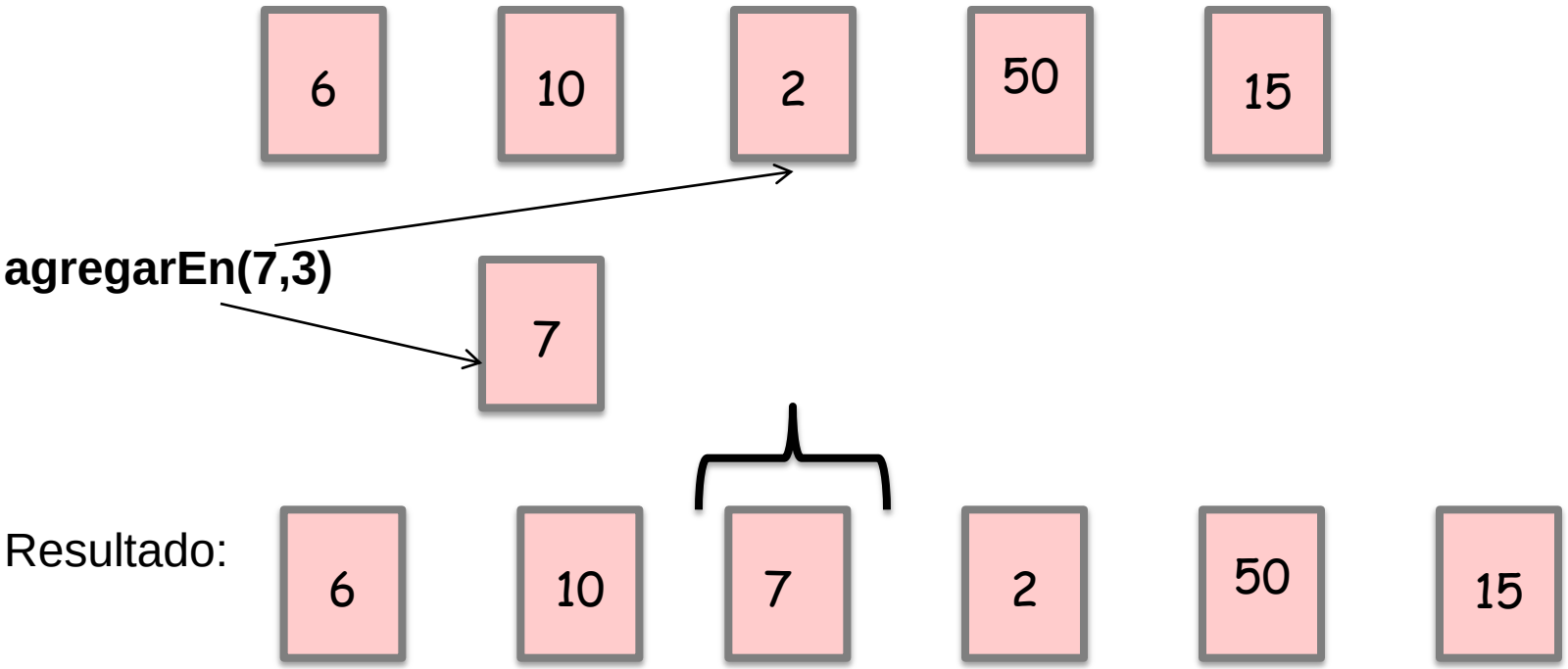
agregarFinal(Integer elem): Agrega al final de la lista

agregarFinal(24)



Listas: *algunas operaciones*

agregarEn(Integer elem, int pos): Agrega el elemento en la posición indicada



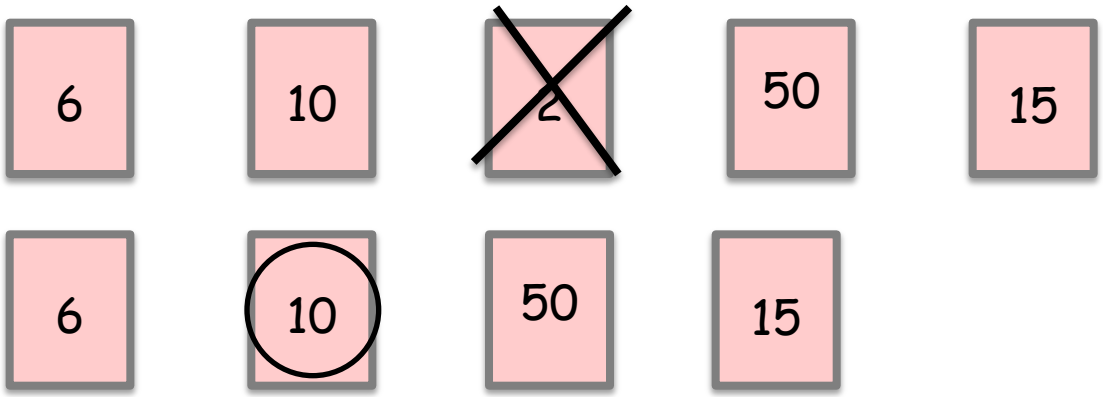
agregarInicio y ***agregarFinal*** *en términos de* ***agregar(Integer elem, int pos)*** *?*

agregarEn(elem, 1) ***agregarEn(elem, tamanio()+1)***

Listas: *algunas operaciones*

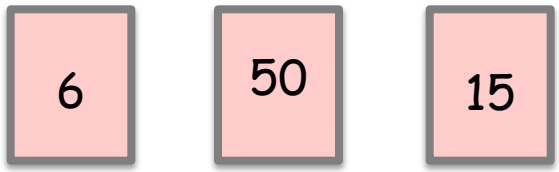
eliminarEn(int pos): Elimina el elemento de la posición indicada

eliminarEn(3)



eliminar(Integer elem):Elimina el elemento "elem" indicado

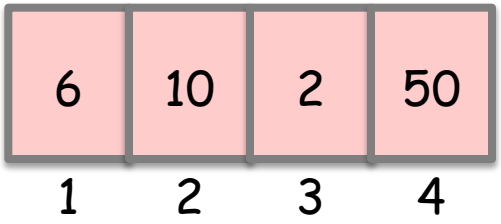
eliminar(10)



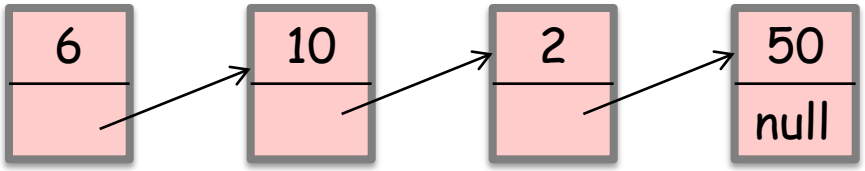
Listas: *implementaciones*

Una lista puede estar implementada a través de:

- *una estructura estática (arreglo)*



- *una estructura dinámica (usando punteros)*



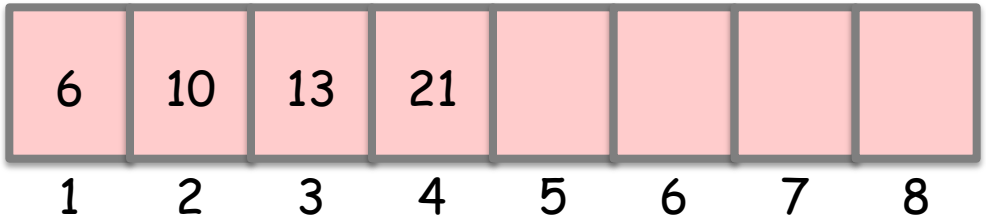
Listas: *con estructura estática*

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

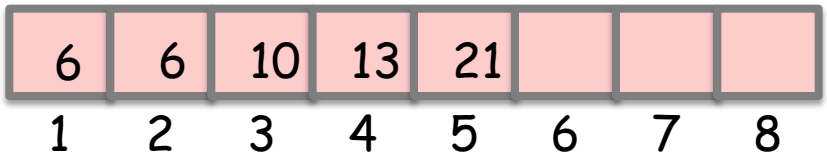
agregarInicio(11)

Estado Inicial

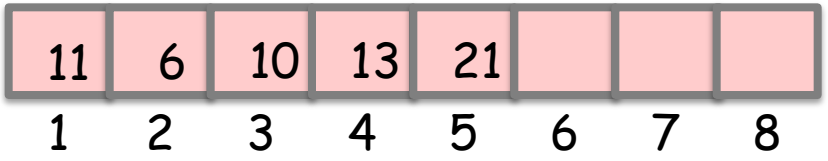
elem = 11
tamanio ← 4



(1) Corrimiento a derecha

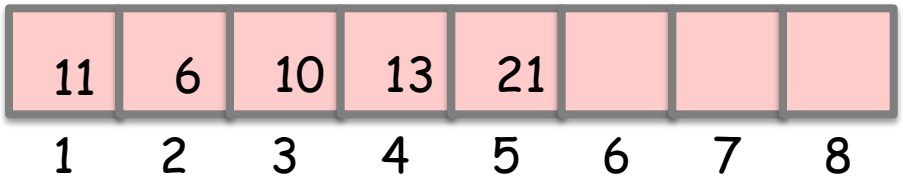


(2) inserción de “elem” en la posición 1
Incrementar “tamanio”



Estado Final

tamanio ← 5



Listas: *con estructura estática*

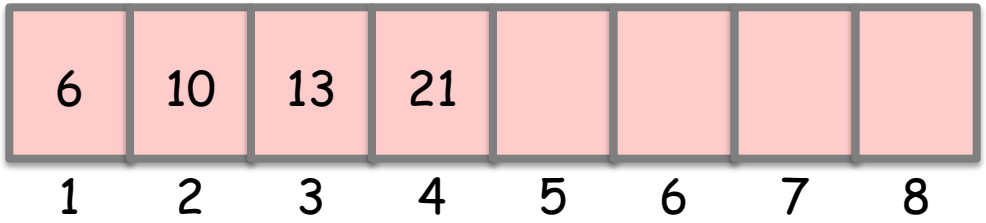
agregarFinal(Integer elem): Agrega al final de la lista

agregarFinal(11)

Estado Inicial

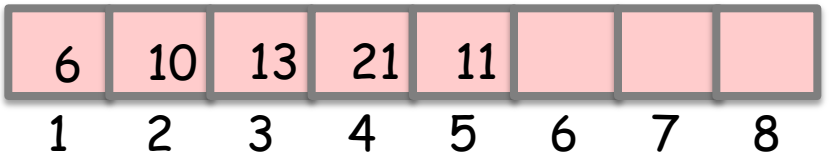
elem = 11

tamanio ← 4

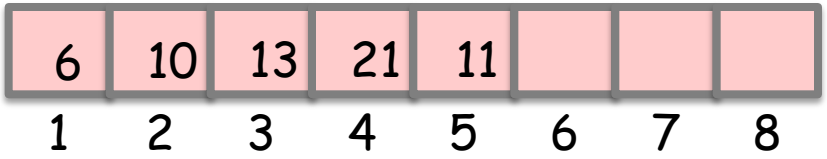


(1) Incrementar tamaño de la lista:
tamanio = tamanio+1

(2) agregar “elem” en la posición “tamanio”



Estado Final



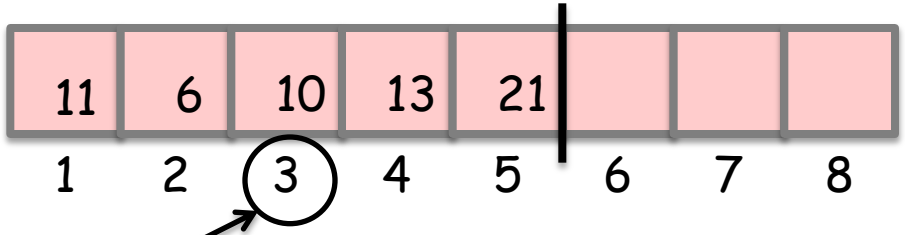
Listas: *con estructura estática*

eliminarEn(int pos): Elimina el elemento de la posición indicada

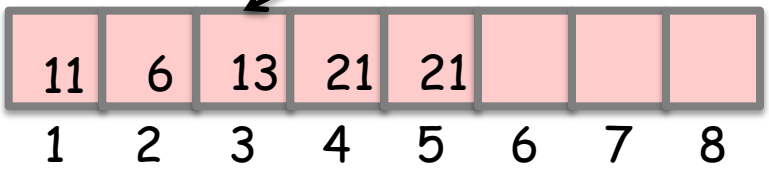
eliminarEn(3):

tamanio ← 5

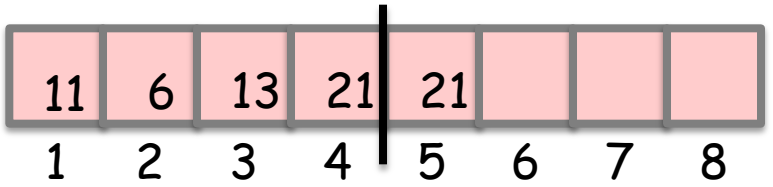
Estado Inicial



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

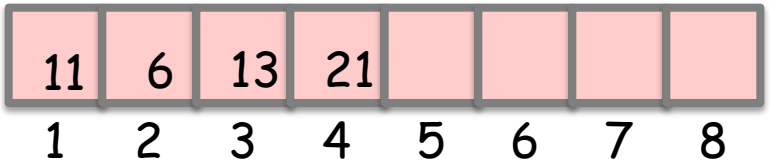


(2) Decrementar tamaño de la lista: tamanio = tamanio -1



Estado Final

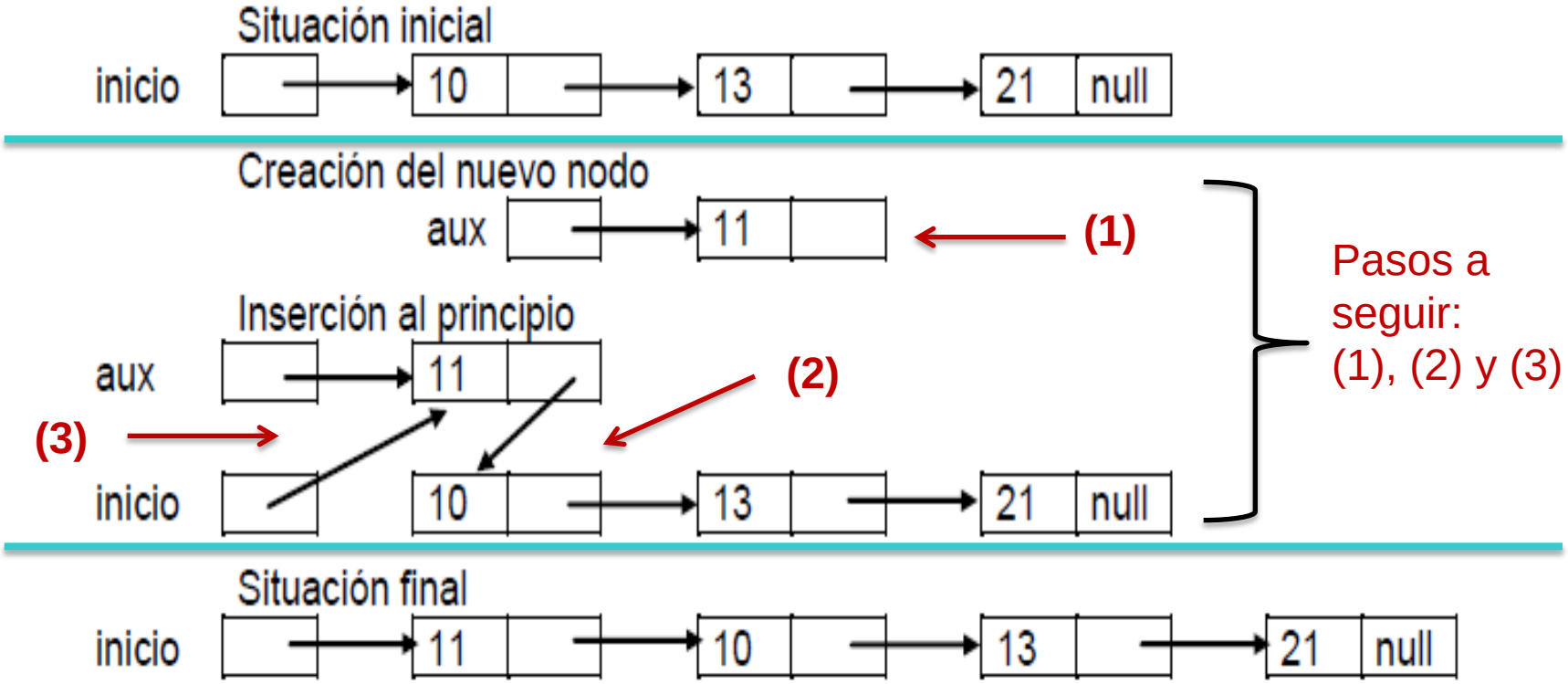
tamanio ← 4



Listas: *con estructura dinámica*

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

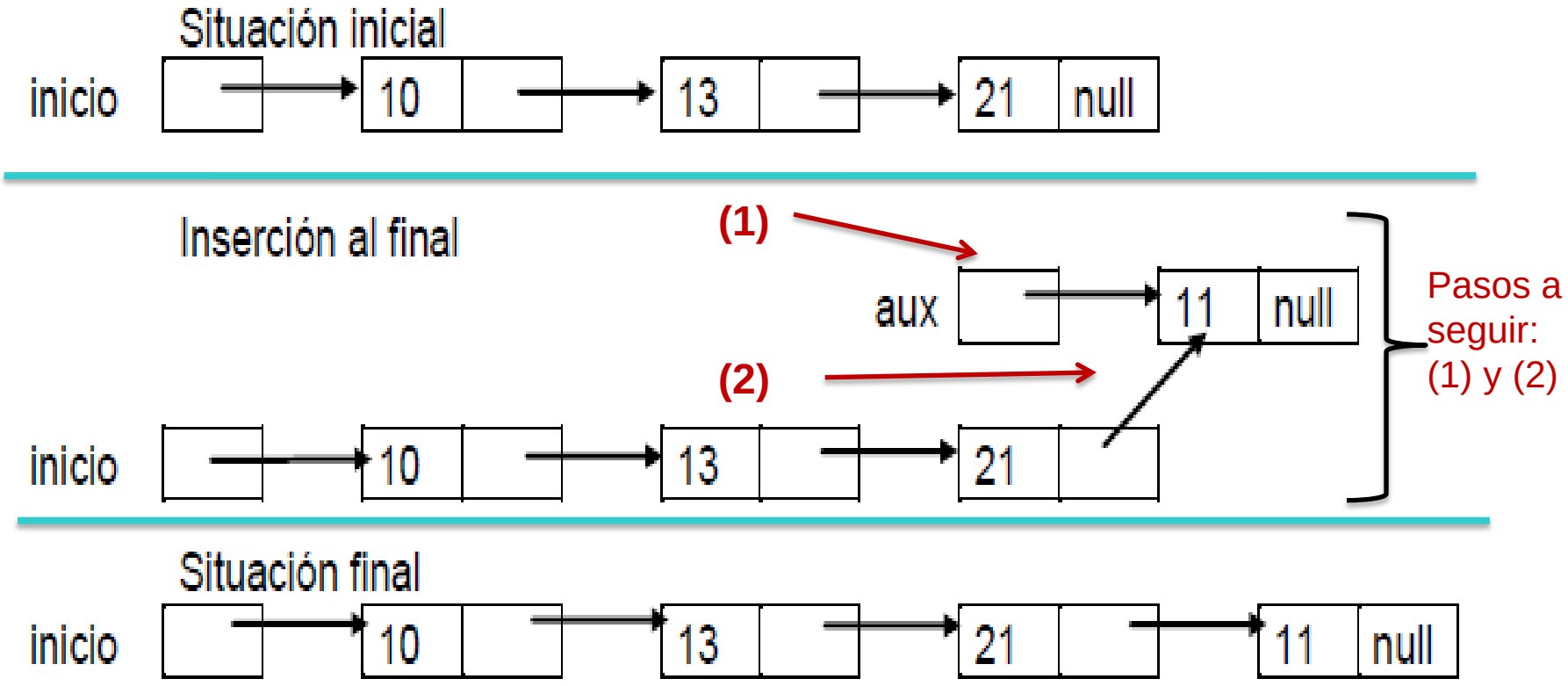
agregarInicio(11)



Listas: *con estructura dinámica*

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

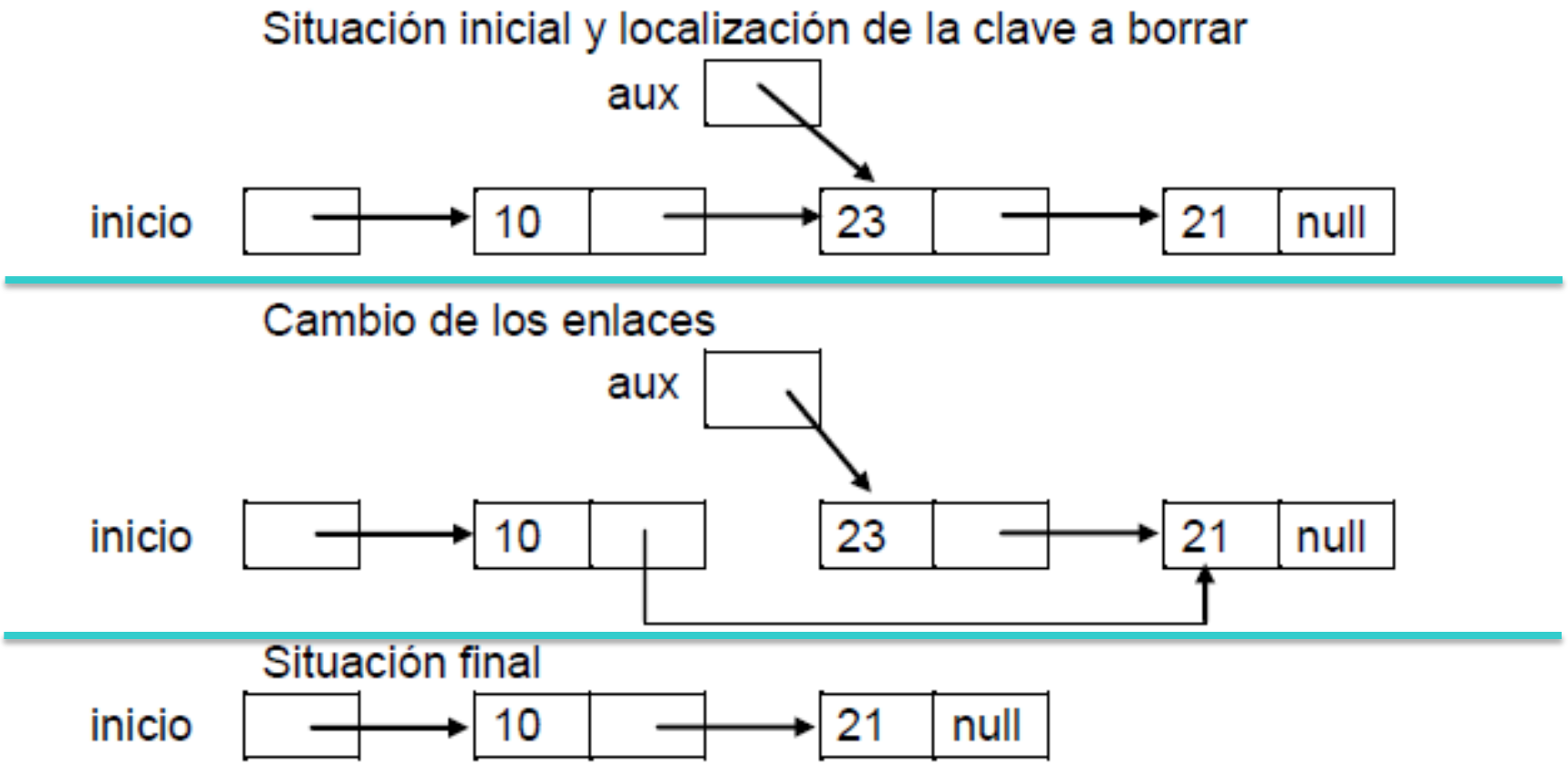
agregarFinal(11)



Listas: *con estructura dinámica*

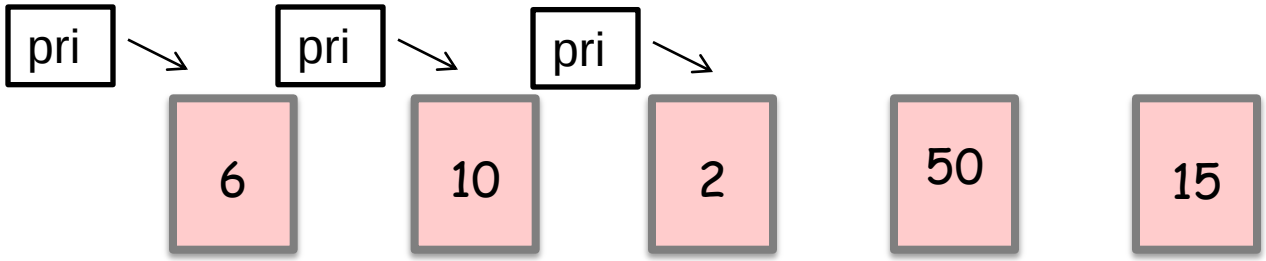
eliminar(Integer elem): Elimina el elemento "elem" indicado

eliminar(23)



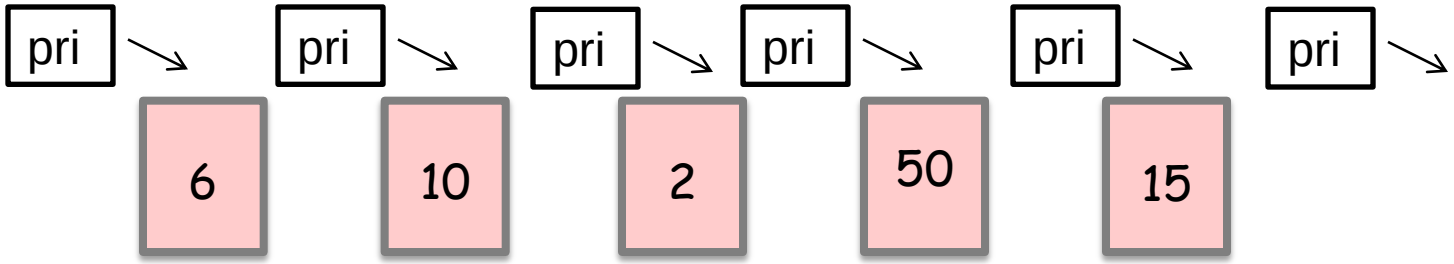
Listas: *operaciones* *comenzar(), proximo(), fin()*

Ejemplo : ***incluye (Integer elem)***



incluye (2) retornará “true”

Si no se guarda el inicio de la lista en otra variable se pierde.



incluye (20) retornará “false”

Listas:

- Ejemplo sin usar ***comenzar(), proximo()*** ni ***fin()*** -

```
function buscar ( pri: lista; x: integer): boolean;
Var
    encontre : boolean;
begin
        encontre := false;
        while ( not encontre ) and ( pri <> NIL ) do
        if x = pri^.datos then encontre := true
                        else pri := pri^.sig;
    buscar := encontre
end;
```

(1)

(2)

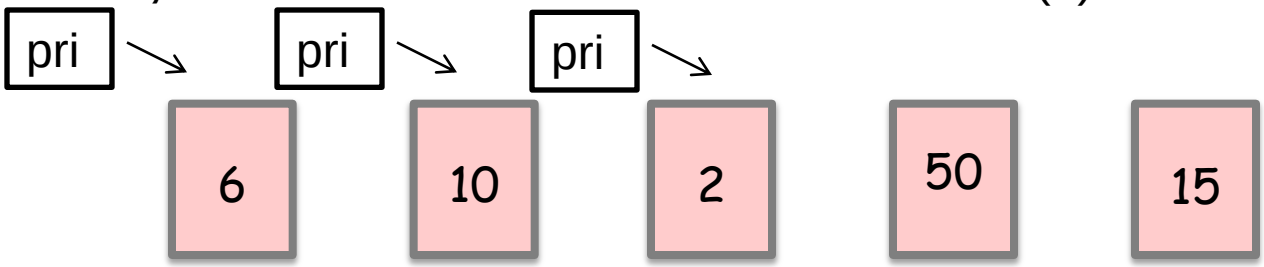
comenzar()

fin()

proximo()

estática

dinámica

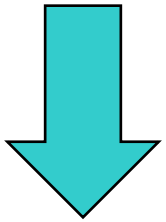


Si no se guarda el inicio de la lista en otra variable se pierde.

Análisis de algoritmos

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:

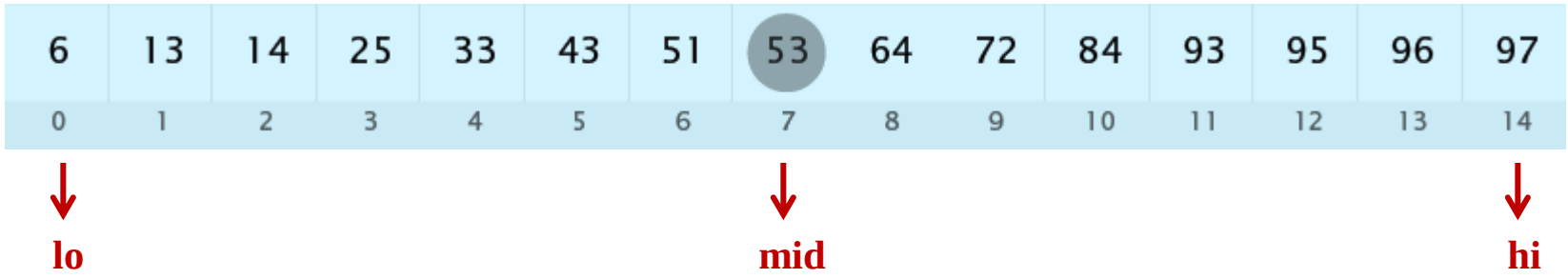
Buscar un elemento en un arreglo

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



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

`public class Stopwatch` (part of `stdlib.jar`)	
`Stopwatch()`	*create a new stopwatch*
`double elapsedTime()`	*time since creation (in seconds)*

Análisis empírico

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

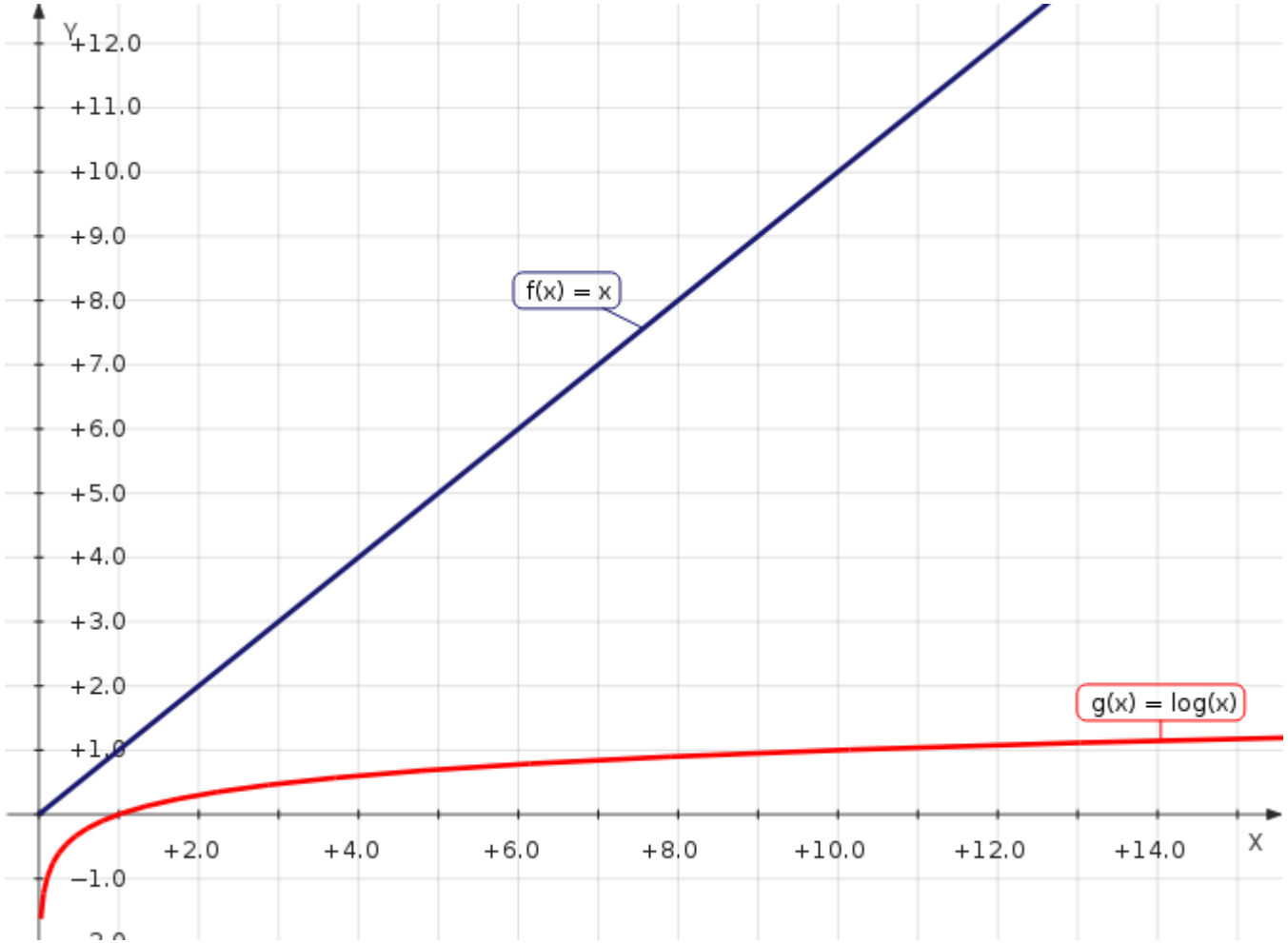
Búsqueda secuencial

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



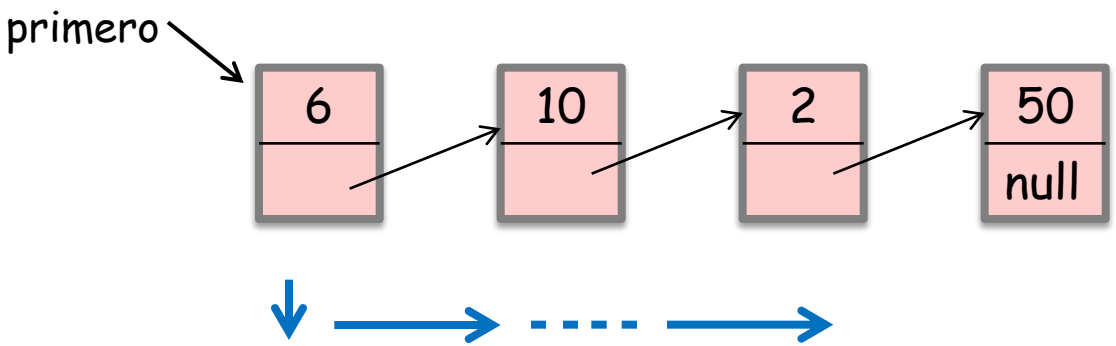
Caso:
Buscar un elemento en una lista dinámica

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

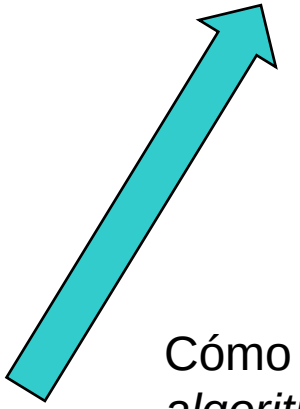
Análisis de algoritmos

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



Cómo se comporta el *algoritmo* dependiendo de la *entrada de datos*.

Factores independientes del sistema

- Algoritmo
- Entrada de datos

cantidad y distribución

Factores dependientes del sistema

- Hardware: CPU, memoria
- Software: lenguaje, compilador, ..
- Sistema: SO, red, otras aplicaciones

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: ~ log (N)

Fin clase 1