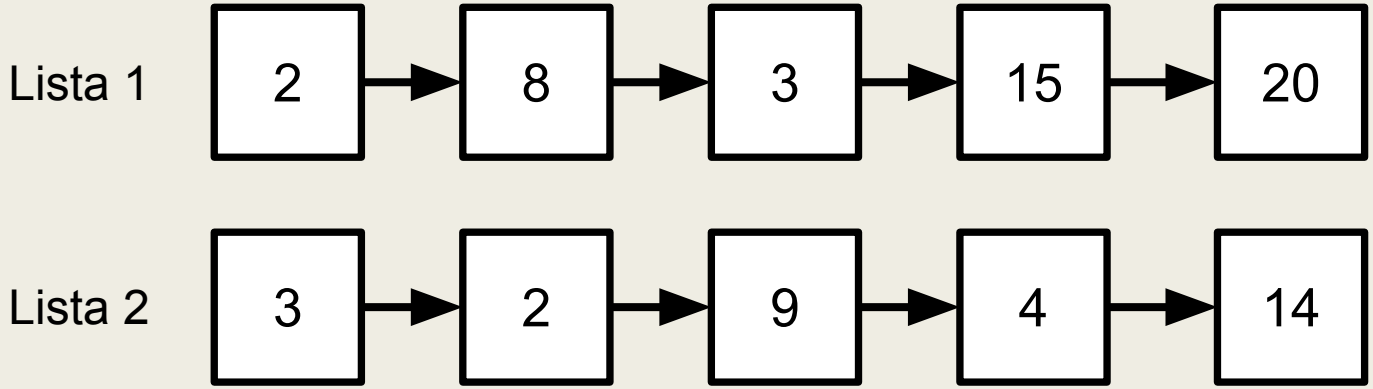
PRÁCTICO 1 - LISTAS

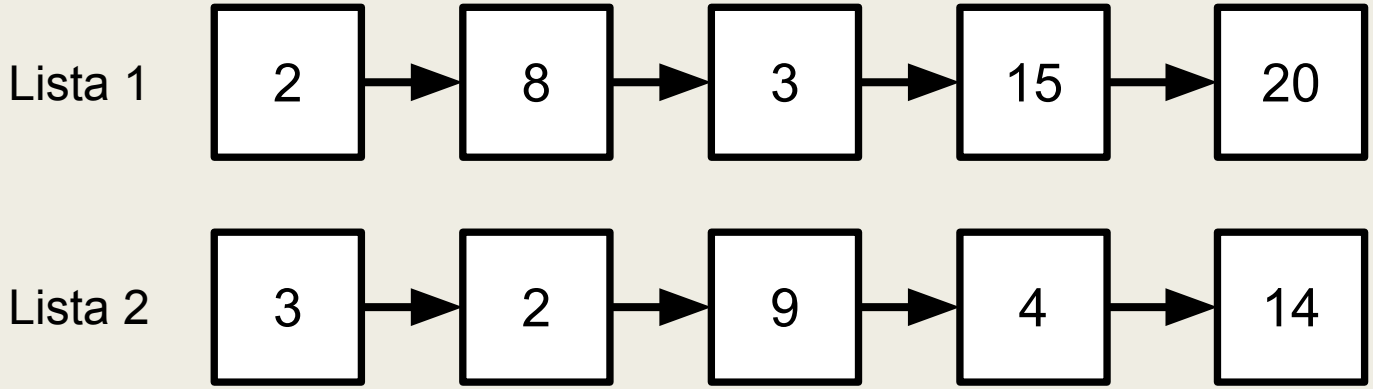
Ejercicio. Escriba una función que dadas dos listas construya otra con los elementos que están en la primera y también en la segunda.

Dos listas desordenadas




```
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2
```

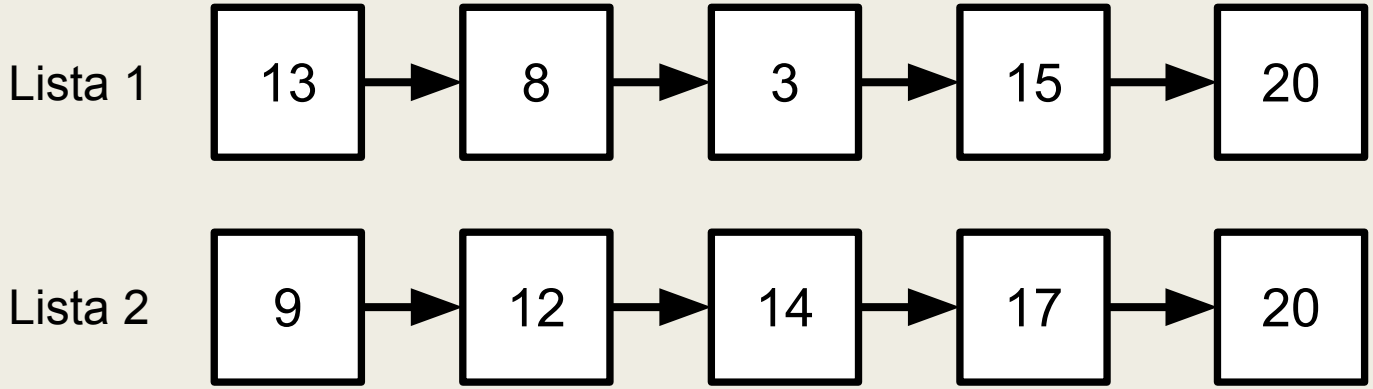
Dos listas desordenadas



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

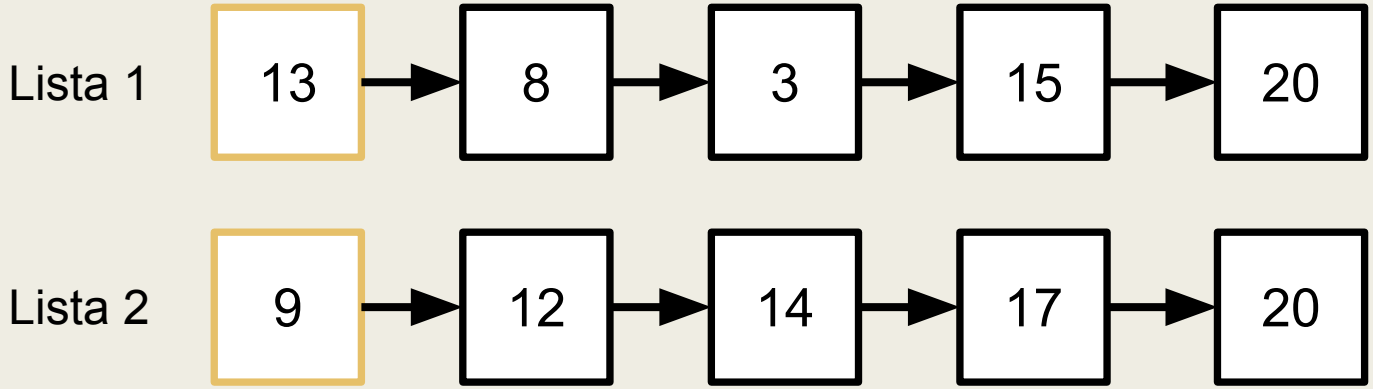
→ O(n*h) → se puede representar como O(n^2) si n >= h

Segunda lista ordenada



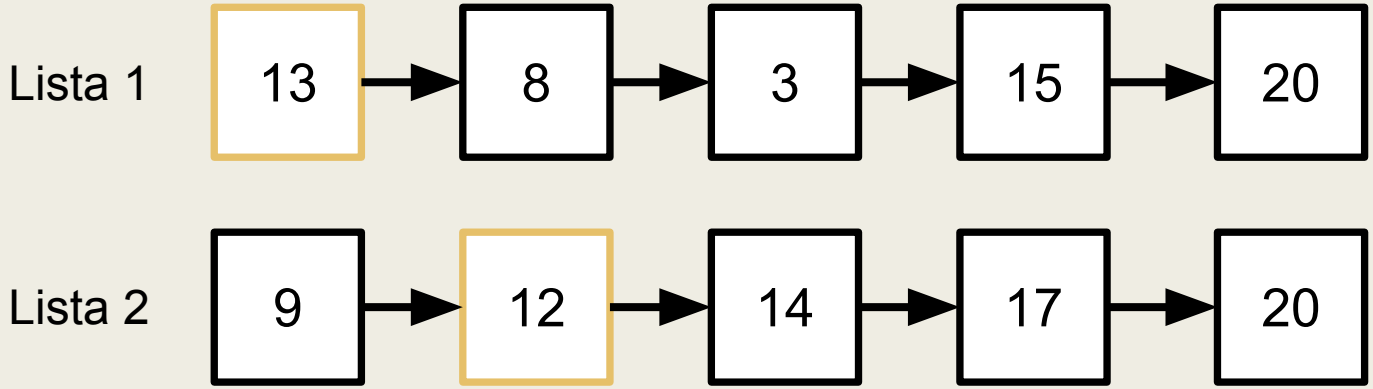

```
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2
```

Segunda lista ordenada



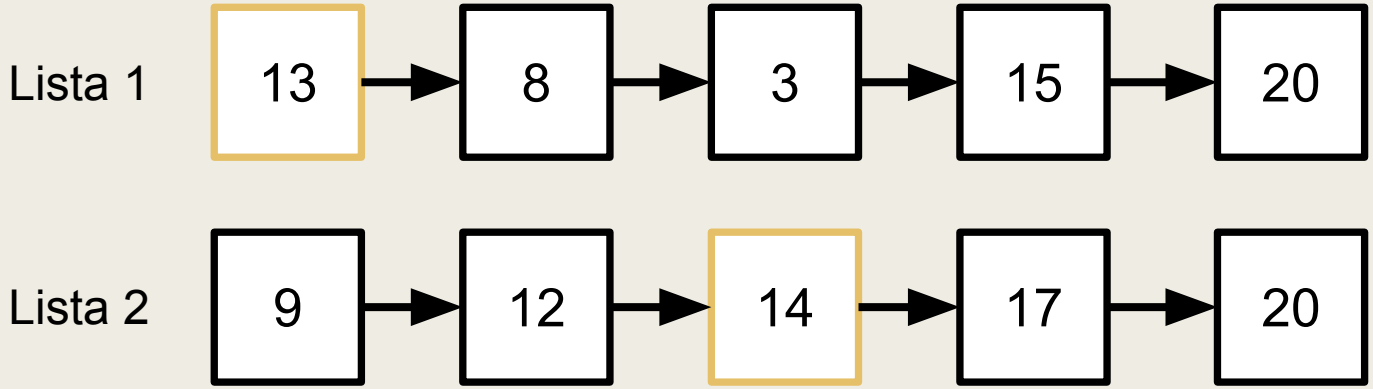
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

Segunda lista ordenada



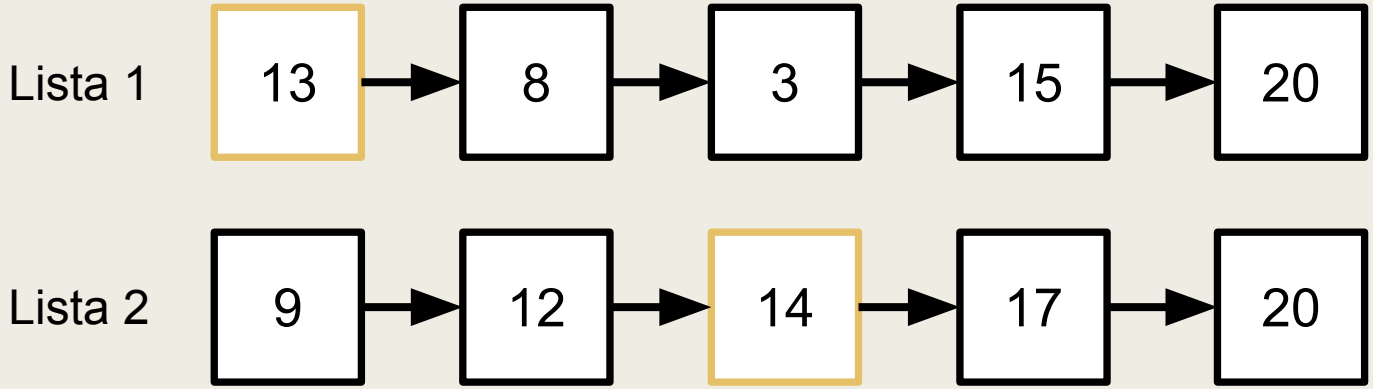
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

Segunda lista ordenada



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

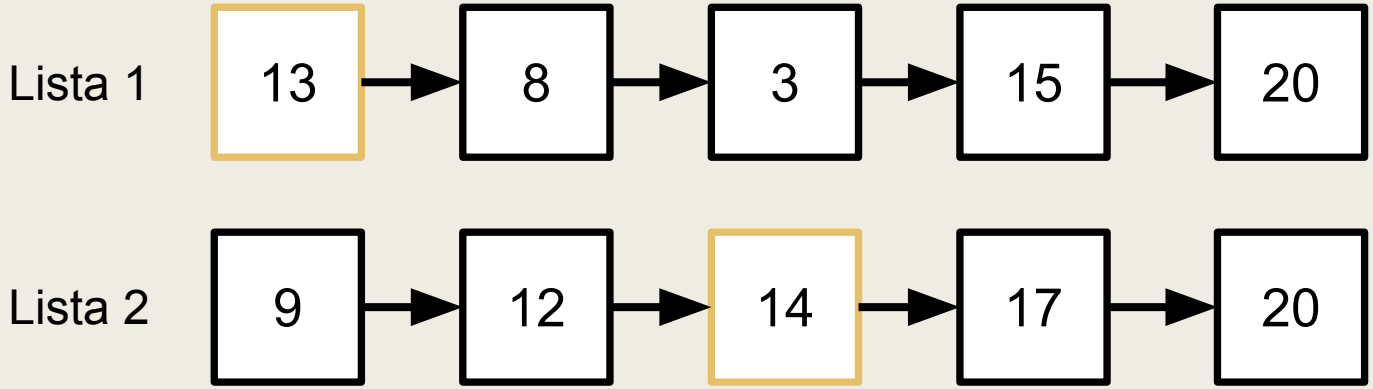
Segunda lista ordenada



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

¡Si en la busqueda encuentro un elemento mas grande, puedo cortar la busqueda!

Segunda lista ordenada

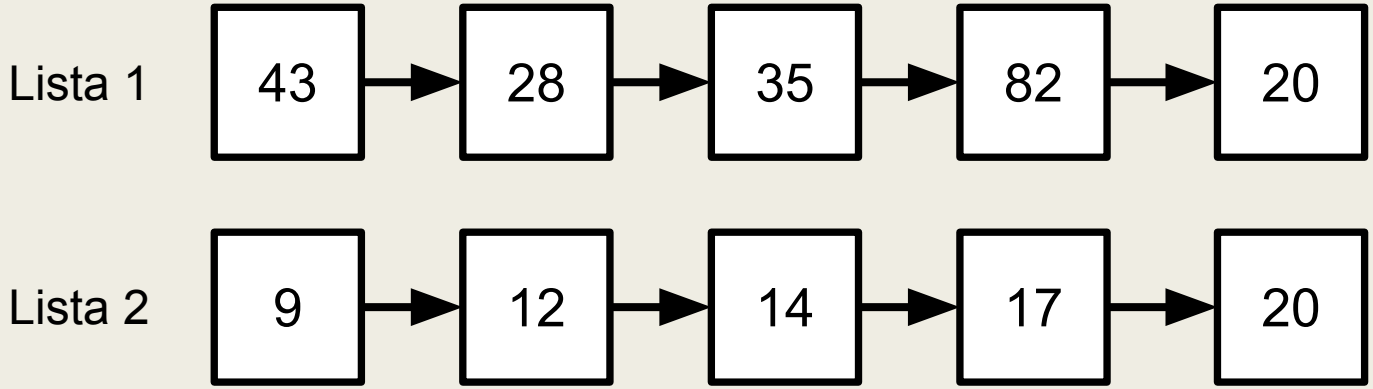


// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

¡Si en la busqueda encuentro un elemento mas grande, puedo cortar la busqueda!

¿Cuanto me mejora esto? ¿Cual es el peor caso ahora?

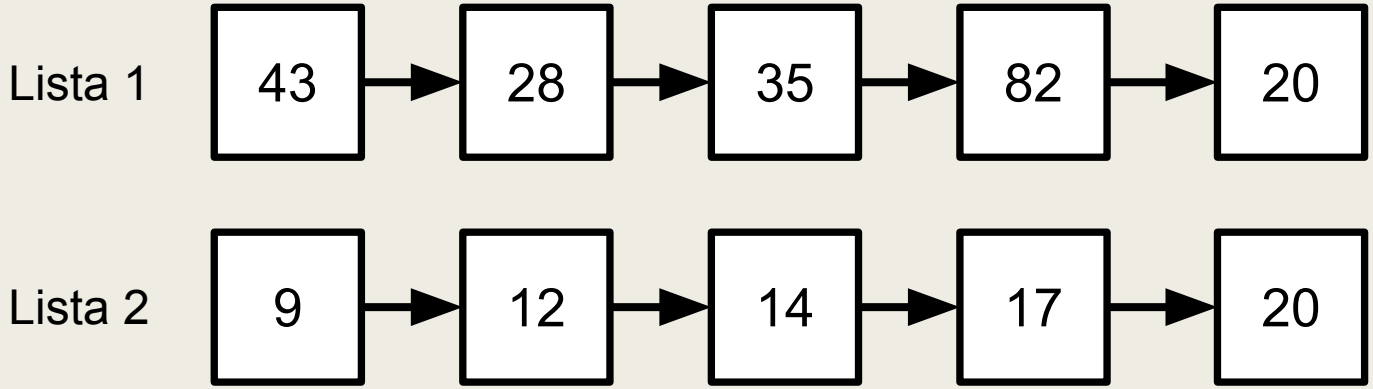
Segunda lista ordenada



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

El peor caso es que TODOS los elementos de la primera lista sean mas grandes que los elementos de la segunda lista.

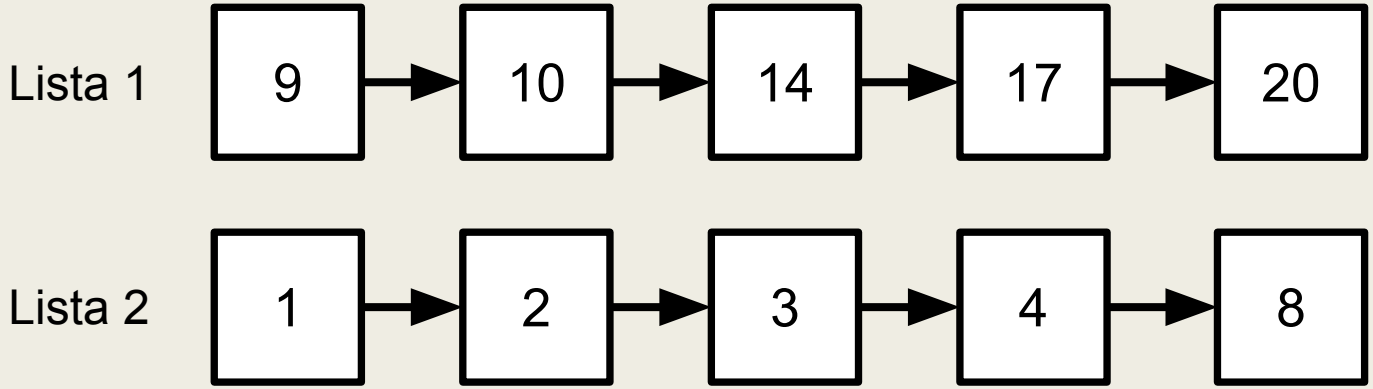
Segunda lista ordenada



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

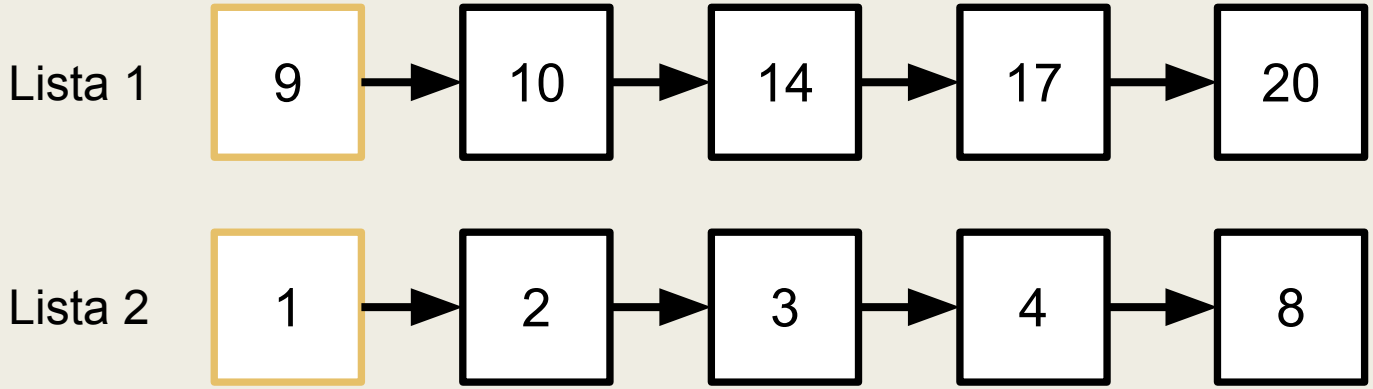
El peor caso es que TODOS los elementos de la primera lista sean mas grandes que los elementos de la segunda lista. → Siempre tengo que recorrer la segunda lista → hasta el final. → O(n^2)

Dos listas ordenadas



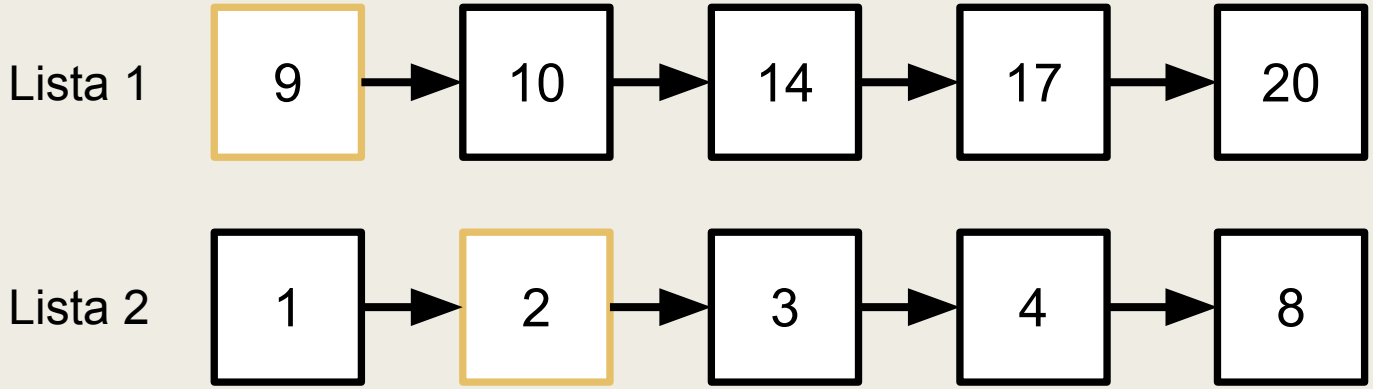
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

Dos listas ordenadas



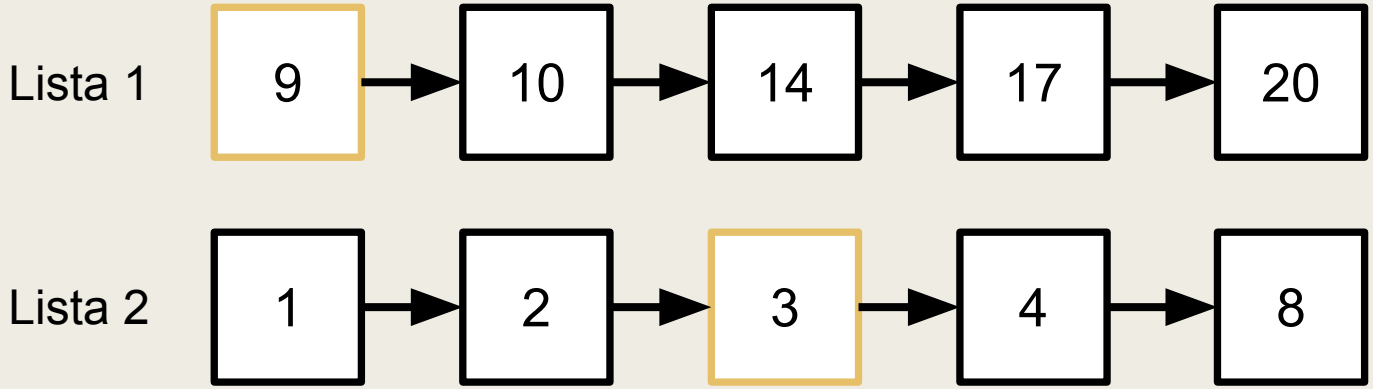

```
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2
```

Dos listas ordenadas



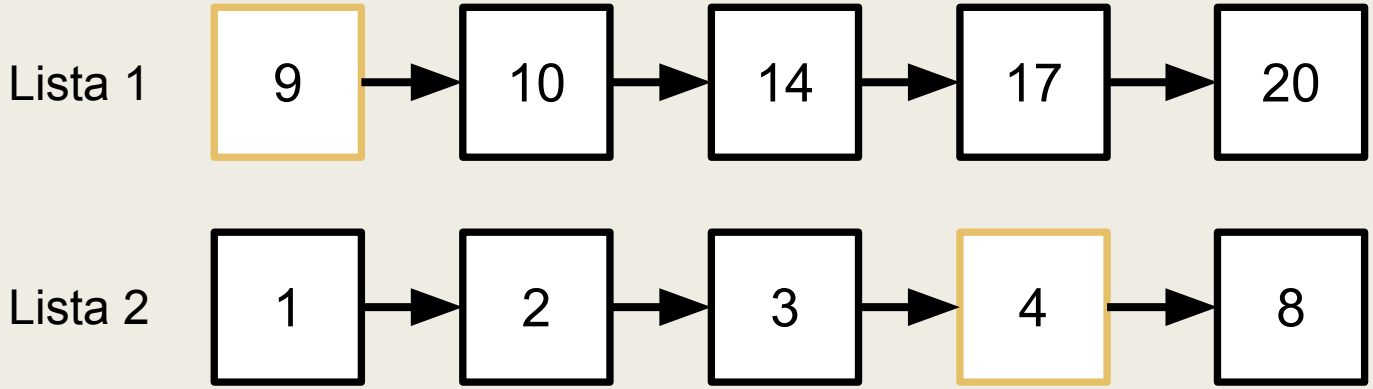
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

Dos listas ordenadas



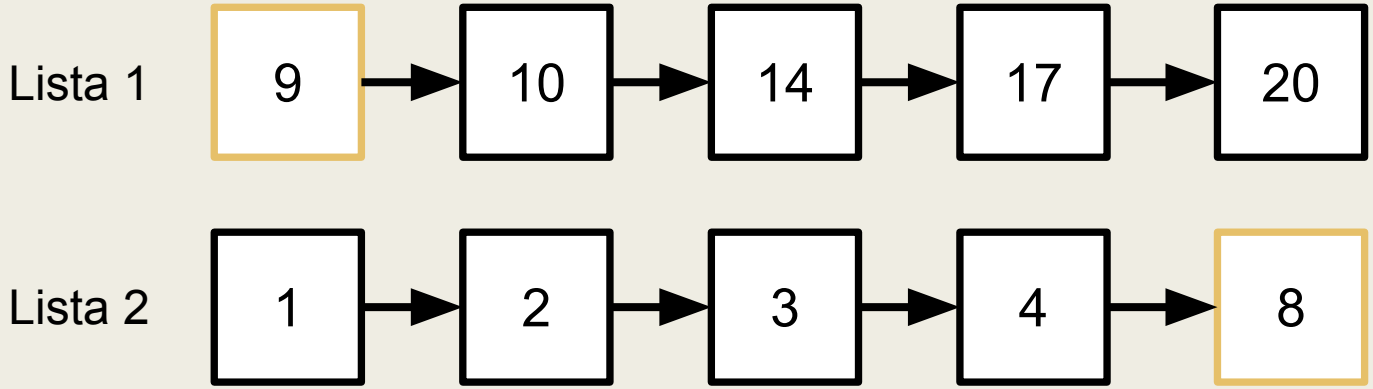

```
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2
```

Dos listas ordenadas



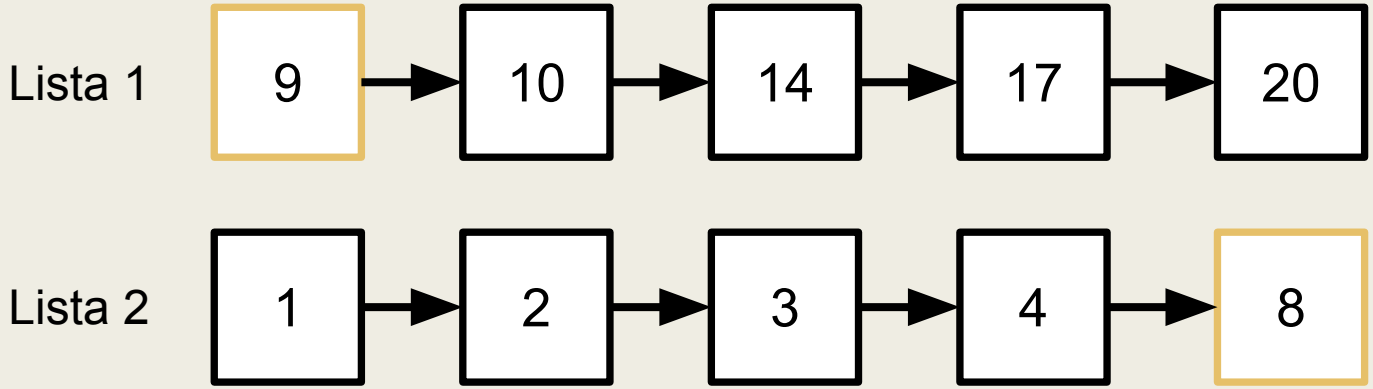
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

Dos listas ordenadas




```
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2
```

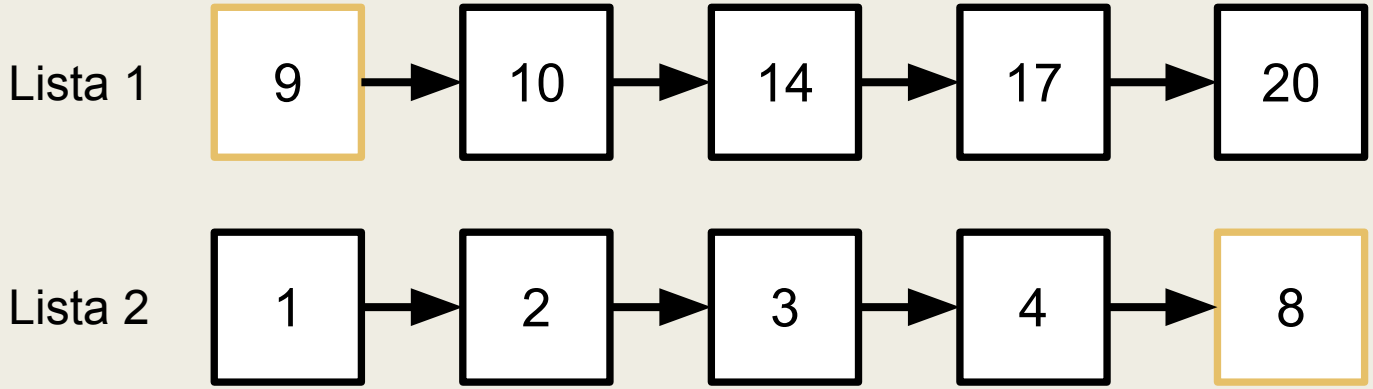
Dos listas ordenadas



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

¡Si llego al final de la segunda lista, puedo cortar el recorrido de la primera lista!

Dos listas ordenadas

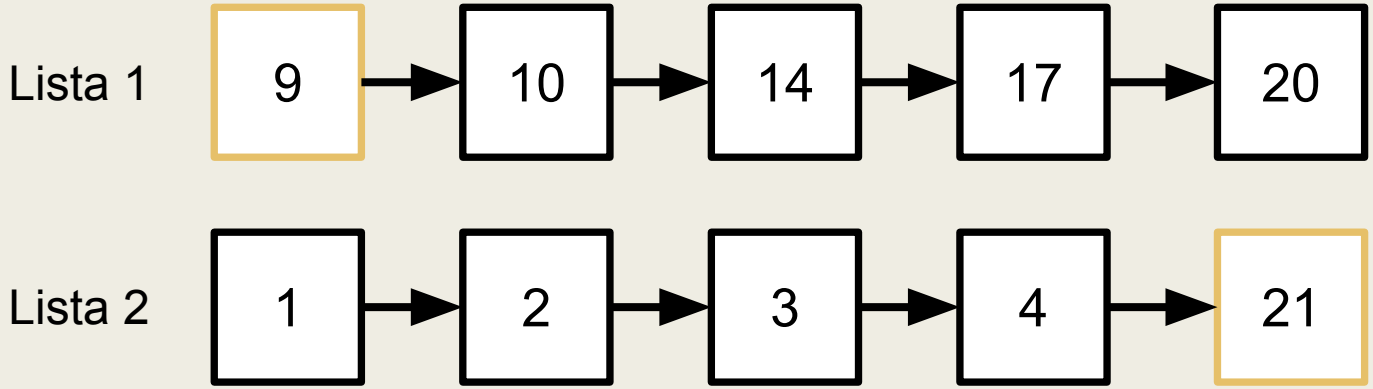


// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

¡Si llego al final de la segunda lista, puedo cortar el recorrido de la primera lista!

¿Cuanto me mejora esto? ¿Cual es el peor caso ahora?

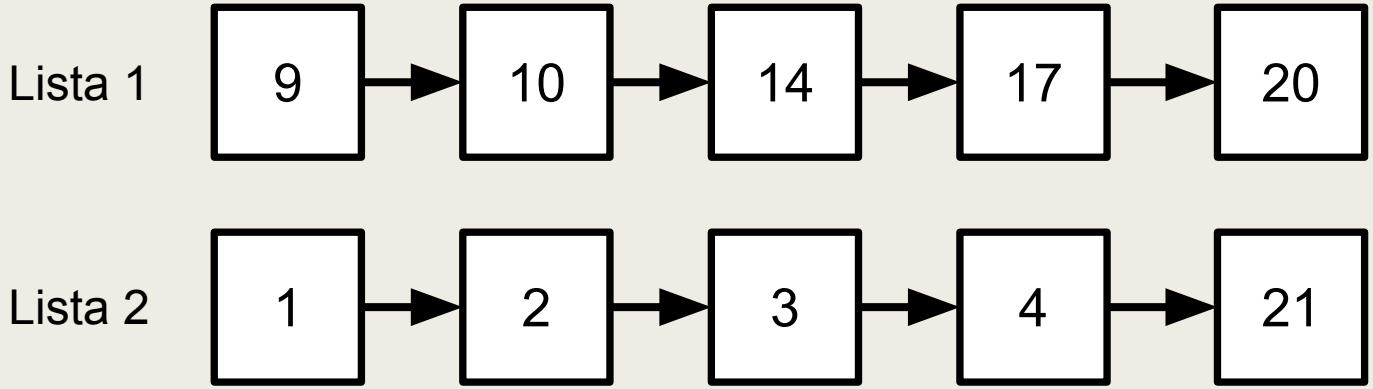
Dos listas ordenadas



// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2

Siempre puede haber una configuración de listas que obligue a mi algoritmo a recorrer todos los elementos de la lista → O(n^2)

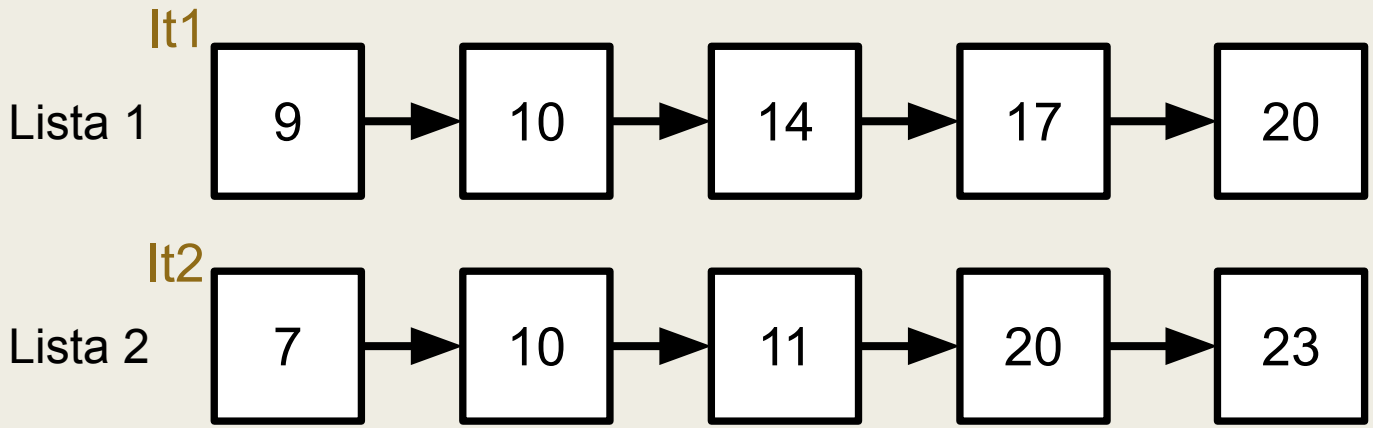
Dos listas ordenadas



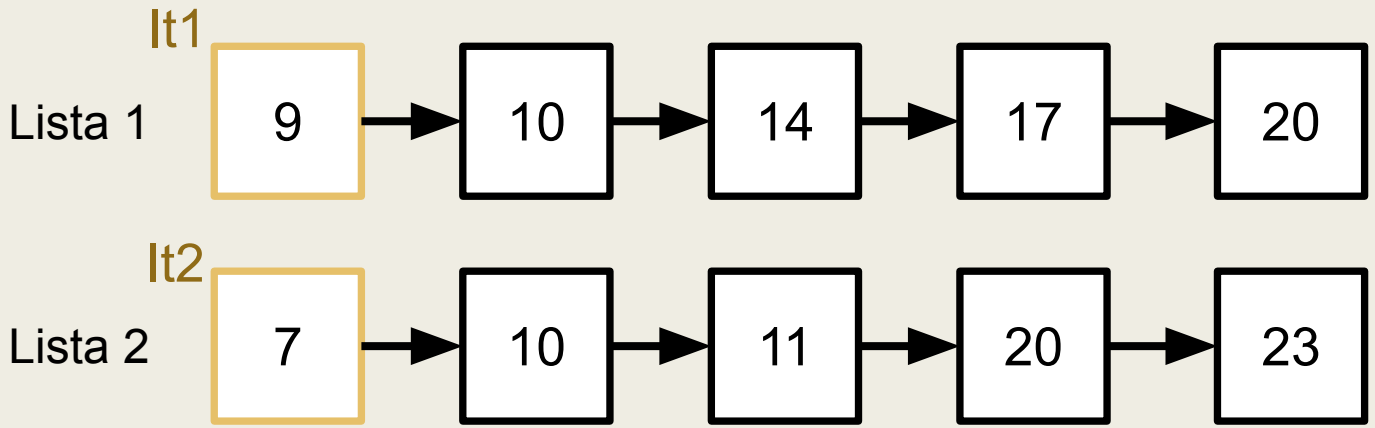

```
// Recorrido lineal sobre la primera lista.
for (Integer e: lista1) → O(n) donde n = cantidad de elementos de Lista 1
    // Por cada elemento, recorrido lineal sobre la segunda lista para ver si existe el elemento.
    buscar(e, lista2) → O(h) donde h = cantidad de elementos de Lista 2
```

Para mejorar realmente la complejidad en la resolución debo cambiar mi algoritmo y/o la estructura de mis datos

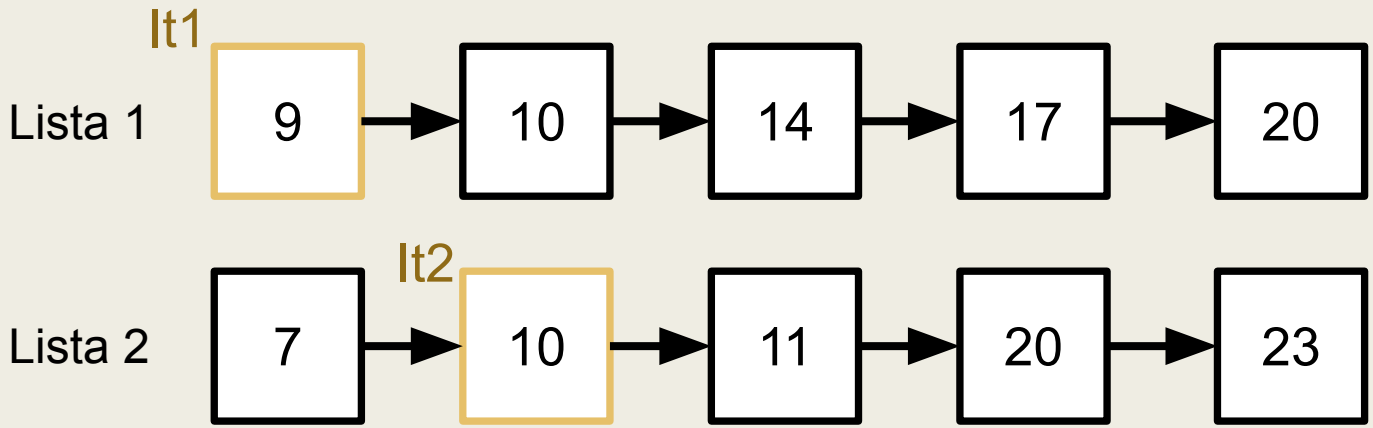
Dos listas ordenadas con iteradores



Dos listas ordenadas con iteradores

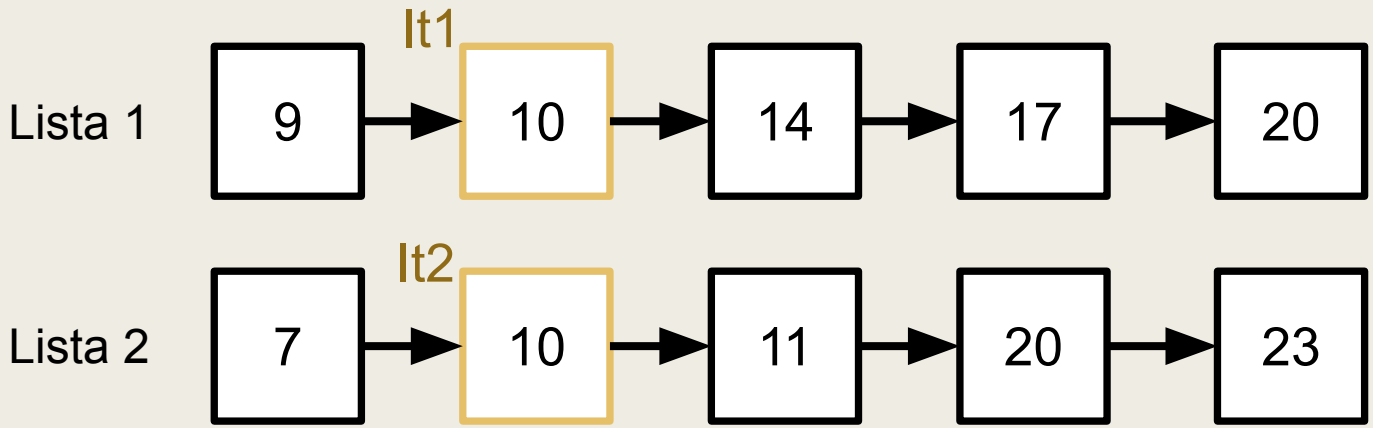


Dos listas ordenadas con iteradores



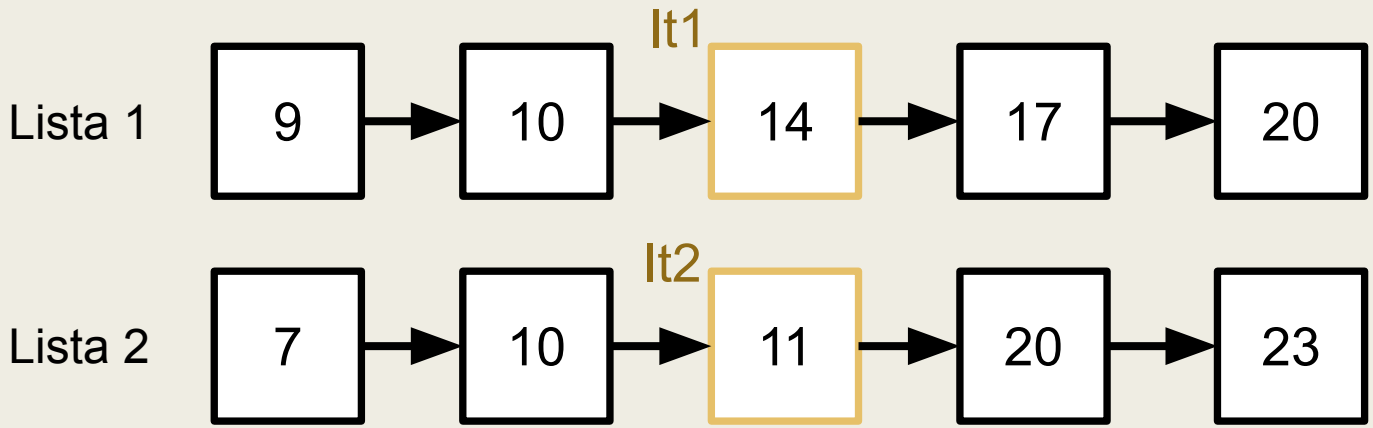

```
// Si el valor de iter2 es menor al de iter1, avanzo el iter2
if (iter2.valor() < iter1.valor())
    iter2.avanzar()
```

Dos listas ordenadas con iteradores




```
// Si el valor de iter2 es menor al de iter1, avanzo el iter2
if (iter2.valor() < iter1.valor())
    iter2.avanzar()
// Si el valor de iter1 es menor al de iter2, avanzo el iter1
if (iter1.valor() < iter2.valor())
    iter1.avanzar()
```

Dos listas ordenadas con iteradores

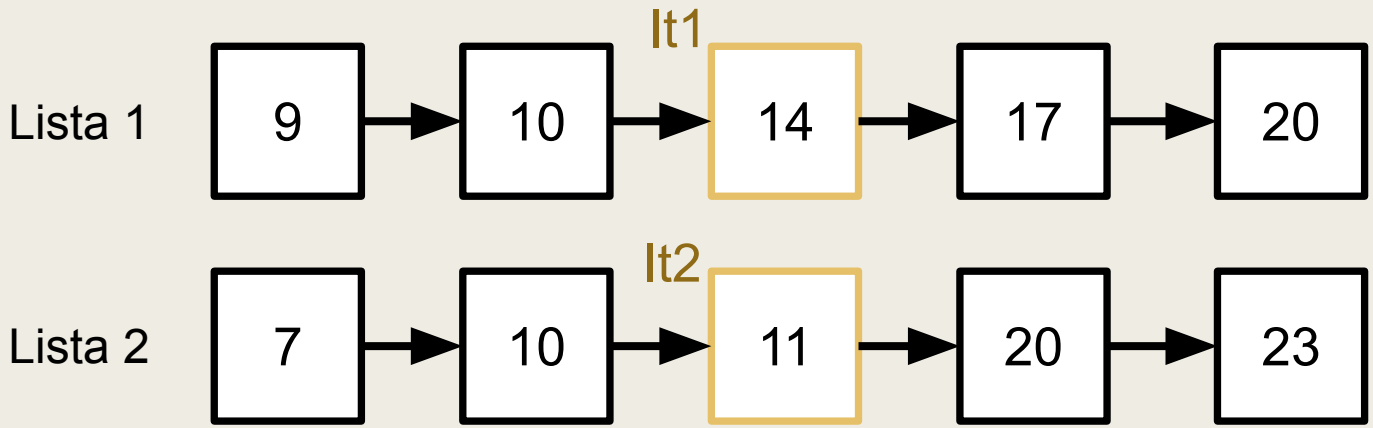



```
// Si el valor de iter2 es menor al de iter1, avanzo el iter2
if (iter2.valor() < iter1.valor())
    iter2.avanzar()
// Si el valor de iter1 es menor al de iter2, avanzo el iter1
if (iter1.valor() < iter2.valor())
    iter1.avanzar()
```

```
// Si los valores son iguales, avanzo ambos
if (iter1.valor() == iter2.valor())
    // Y agrego el valor a la solución
    solucion.agregar(iter1.valor())
    iter1.avanzar()
    iter2.avanzar()
```

Solución: 10

Dos listas ordenadas con iteradores




```
// Si el valor de iter2 es menor al de iter1, avanzo el iter2
if (iter2.valor() < iter1.valor())
    iter2.avanzar()
// Si el valor de iter1 es menor al de iter2, avanzo el iter1
if (iter1.valor() < iter2.valor())
    iter1.avanzar()
```

```
// Si los valores son iguales, avanzo ambos
if (iter1.valor() == iter2.valor())
    // Y agrego el valor a la solución
    solucion.agregar(iter1.valor())
    iter1.avanzar()
    iter2.avanzar()
```

En cada iteración avanzo al menos uno de los dos iteradores → O(n+h) → O(2*n) → O(n)

Solución: 10