# Estructuras de datos

## Estructuras de datos

Estructuras de datos

TAD

Arreglos

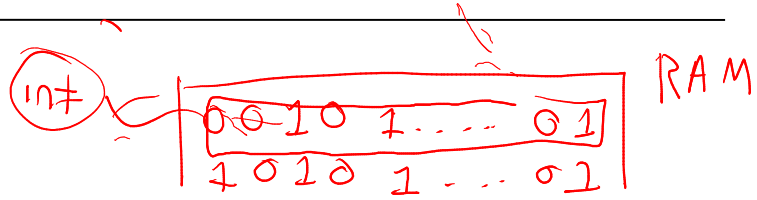
Algoritmos sobre arreglos

Pilas

Colas

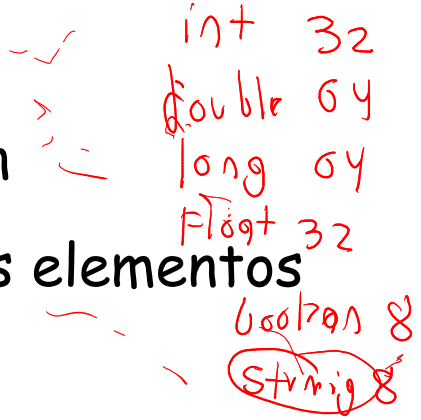
Listas enlazadas

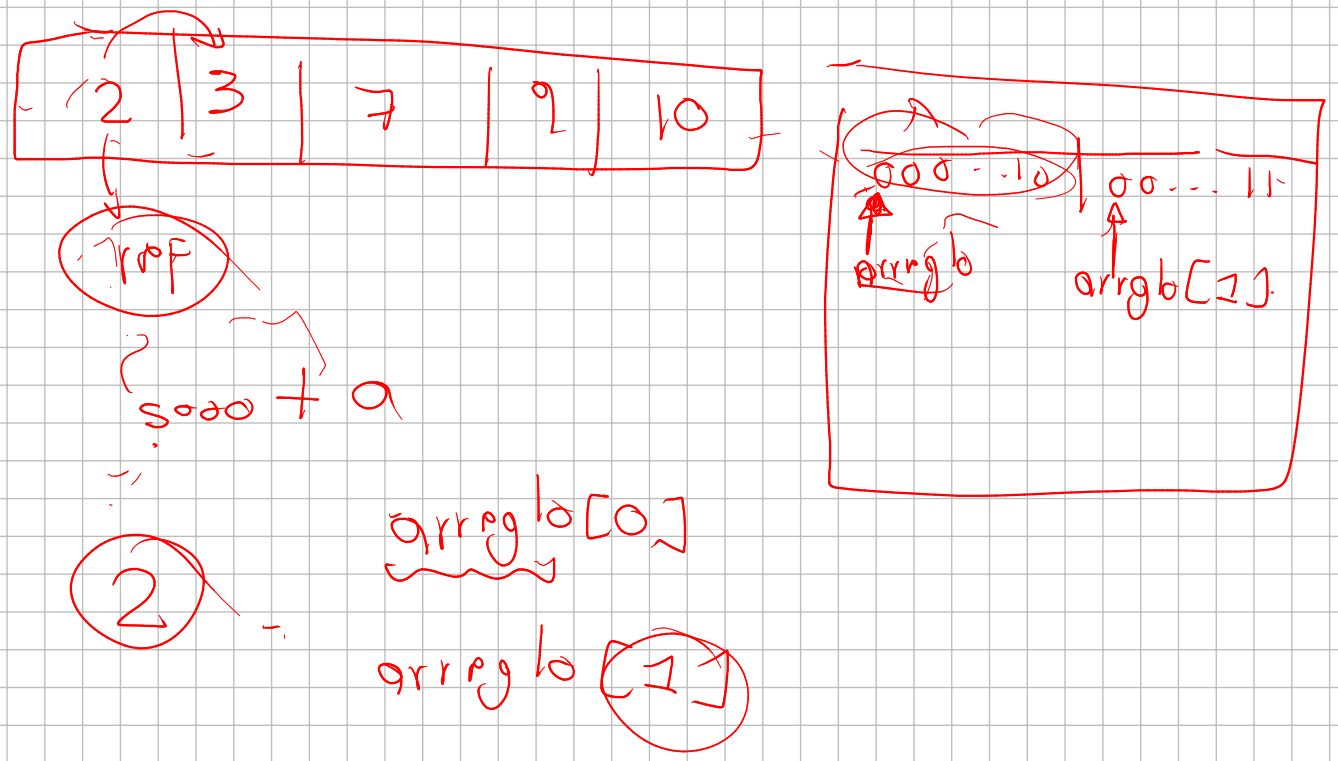
# Estructuras de datos




## ¿Qué es una estructura de datos?

- Es una representación de datos junto a sus operaciones
- Permite la reutilización de componentes
- Debemos separar la interfaz de la implementación
- Todas las estructuras almacenan referencias a los elementos insertados y no realizan copias internas
- El costo de las operaciones depende del tipo de estructura que se implemente
- Se pueden construir como paquetes o módulos




2
3
7
9
10
ptr
5000 + a
2
arrglo[0]
arrglo[1]
000...10
00...11
arrglo
arrglo[1]

# Tipos de Datos Abstractos (TDAs)

## ¿Qué es un TDA?

- Es una representación matemática de los datos que especifica los tipos de datos almacenados y las operaciones definidas
- Un TDA separa la implementación (interna) de la interfaz (operaciones)
- Un TDA define que cada operación debe hacer, más no como lo debe de hacer.
- Un TDA es materializado por una estructura de datos concreta.

S  enteros sin signo

$0 \in S$

inc

add1(n) = n + 1

sub1(n) = n − 1

zero?(n) = #T n=0

#F n≠0

add1(0) = 1

add1(1) = 2

...

Suma(n, m)

Suma(S(S(3)))

add1(add1(add1(S(S(3)))))

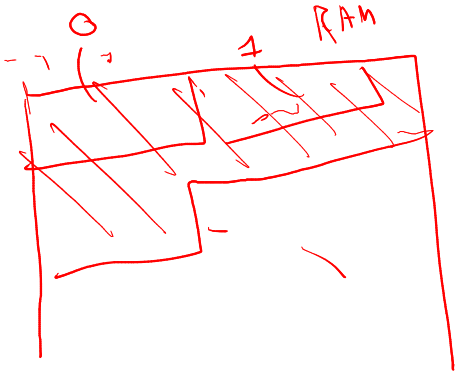
= 8

# Arreglos

## ¿Qué es un arreglo?

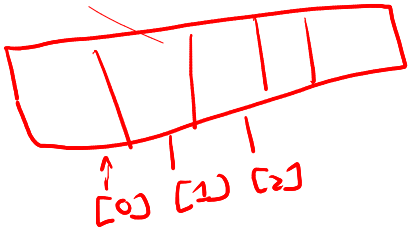
- Conjunto de datos homogéneos que se encuentran ubicados en forma consecutiva en la memoria RAM
- Los arreglos son una colección finita de elementos del mismo tipo
- Un arreglo es representado por la dirección de memoria que apunta al su primer elemento
- El direccionamiento de los arreglos es la dirección de memoria más la posición deseada, esta posición considera el número de bits que ocupa el tipo de dato.

# Arreglos



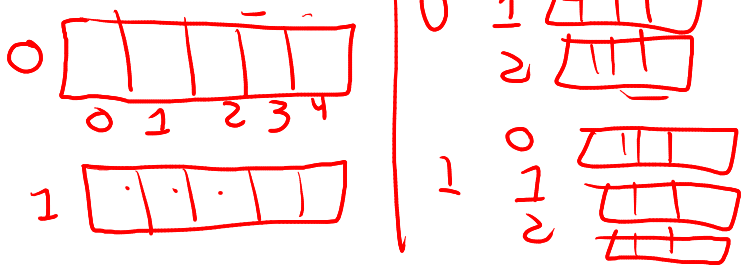

## Tipos de arreglos

- Unidimensionales: Requieren un índice
- Multidimensionales: Requiere n índices
- Un arreglo es representado por la dirección de memoria que apunta al su primer elemento



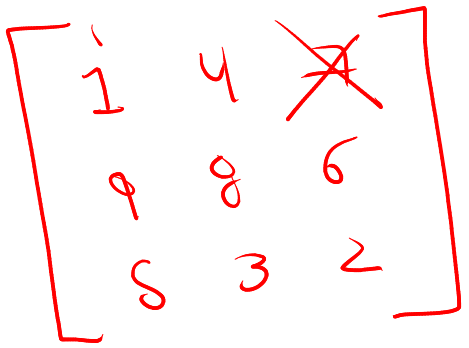

## Operaciones

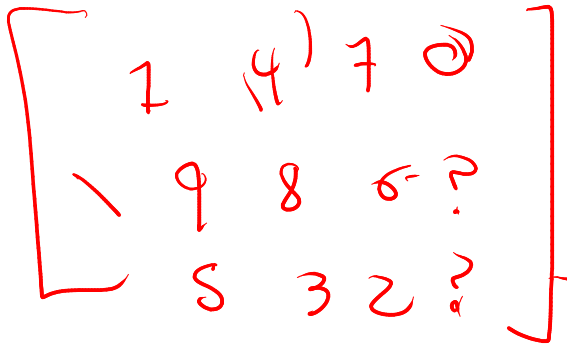
- Lectura / Búsqueda
- Escritura
- Asignación
- Insertar
- Modificación / Borrado *



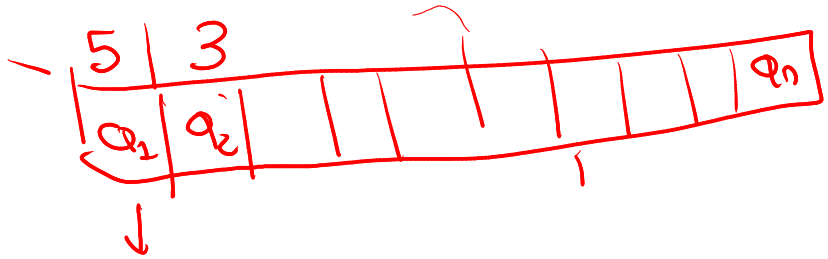

* No es optimo

Instanciar = { Definido reservando p/ espaço de memória }





Atribuir / definir




maxi = -∞

maxi = 5

If arr[0] ≥ maxi :

maxi = 5

If arr[1] ≥ maxi : ✗

m

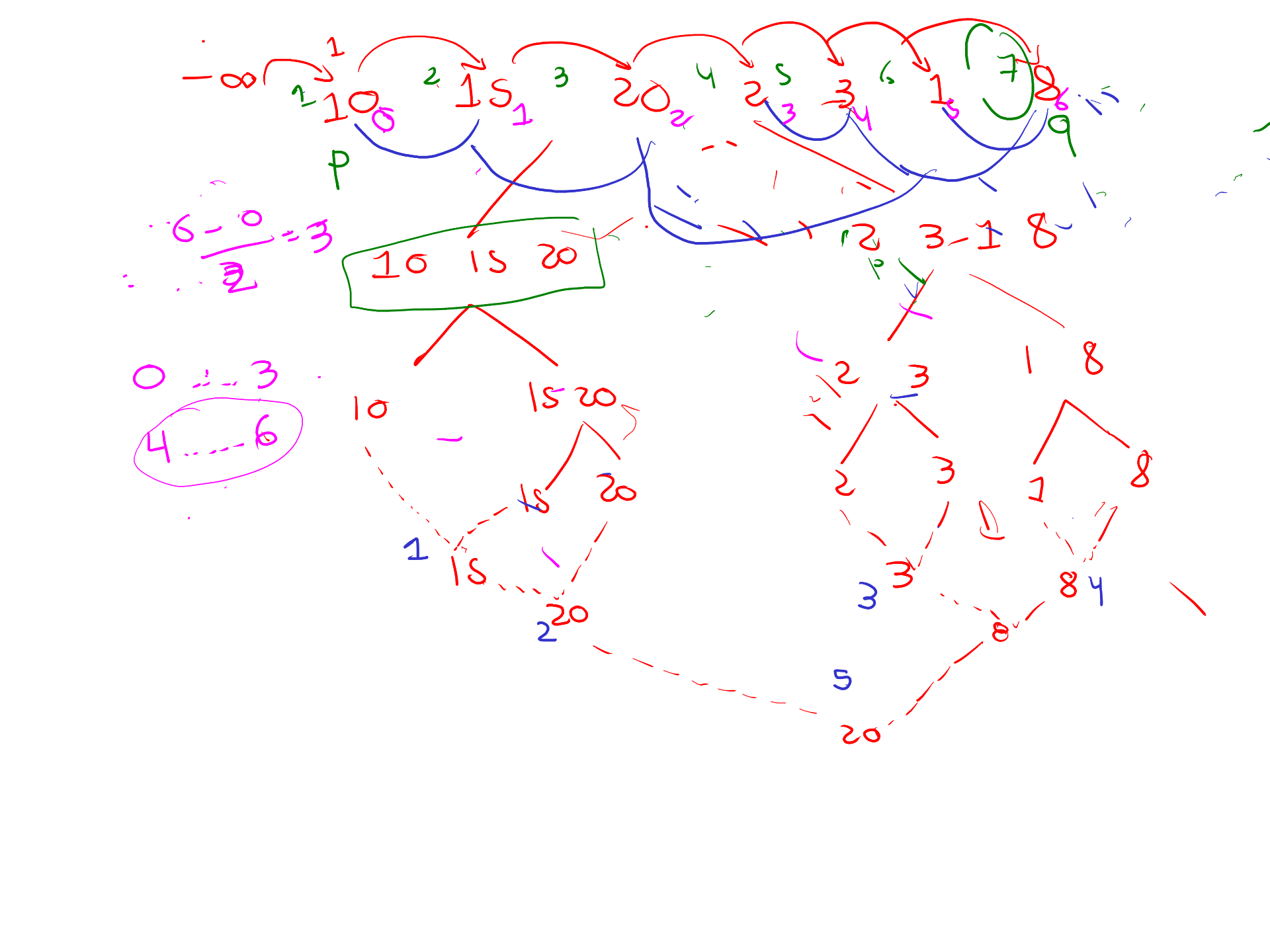
# Algoritmos sobre arreglos

Buscar el máximo de una lista

Dividir divida la lista a la mitad sucesivamente,

Conquistar Llegar al caso trivial de tener un elemento. Este será el mayor de la lista.

Combinar Combinar sucesivamente las listas, dejando como primer elemento el mayor. Así, al llegar al la lista completa el primer elemento será el mayor


-∞
10
15
20
P
6 - 0 / 2 = 3
10 15 20
0 ... 3
4 ... 6
12 3 1 8
P
10
15 20
1
15
20
2
20
5
20
12
3
1
8
2
3
1
8
3
3
8 4
8
9

# Algoritmos sobre arreglos

Busqueda binaria

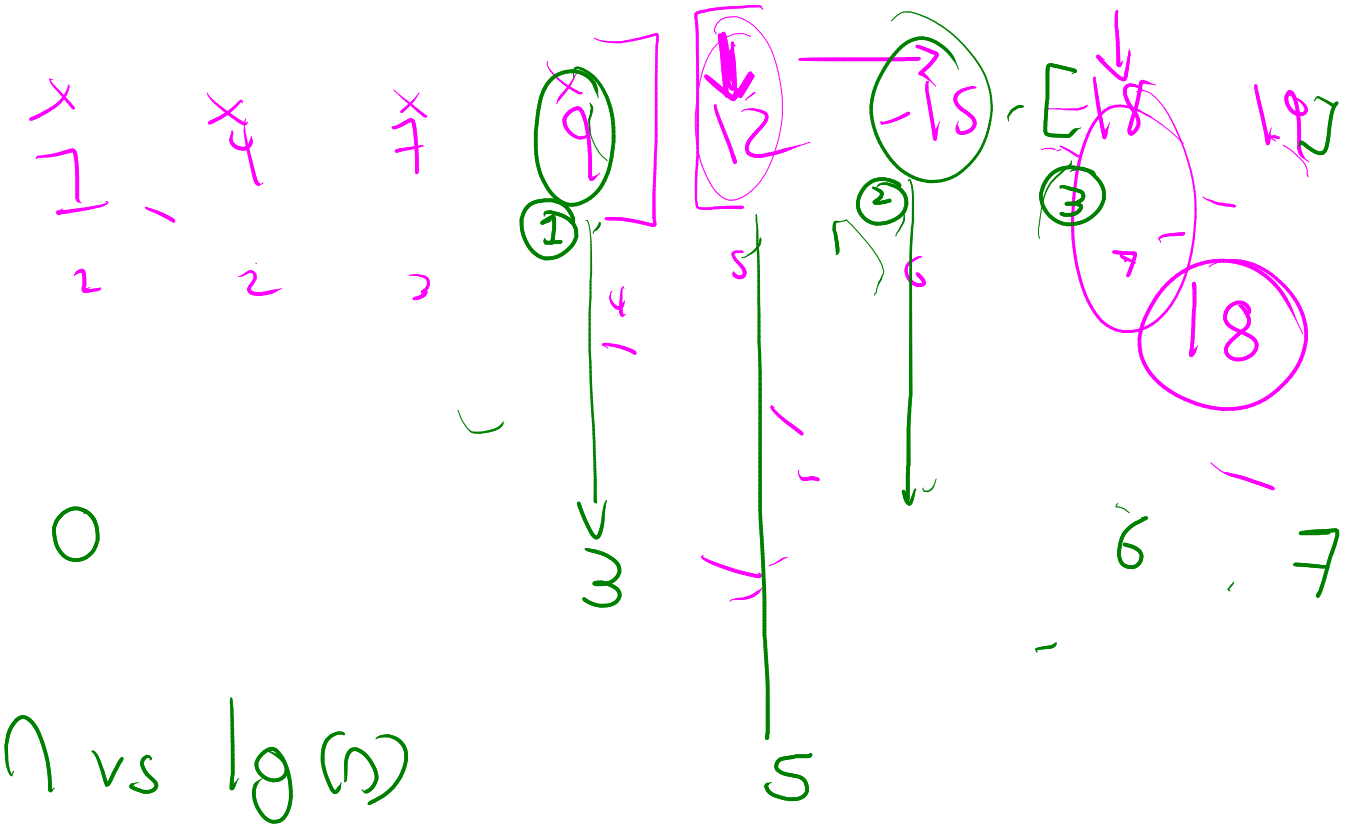
Suponga que la lista está ordenada y que busca un elemento x

Dividir divida la lista a la mitad,

Conquistar Examine el ultimo elemento de la primera lista y el primero de la segunda. Si el primero es menor o igual que x, repita dividir sobre la primera lista. En caso contrario hagalo sobre la segunda.

Combinar El espacio de búsqueda irá reduciendose hasta encontrar el elemento.

¿Cual es el costo computacional?. ¿Si la lista está desordenada, vale la pena ordenar y aplicar este algoritmo?

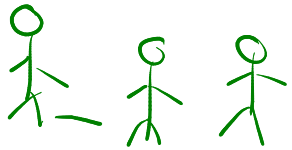



n vs log(n)

# Estructuras de datos

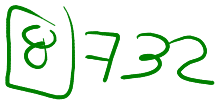
## Pila

Una pila es una estructura de datos tipo LIFO (Last In First Out), por lo que el último elemento insertado será el primero en ser borrado



Operaciones básicas:

STACK-EMPTY(S)



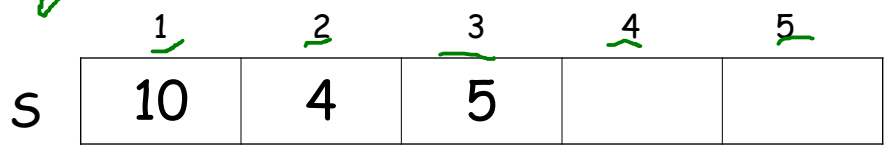
PUSH(S,x)

POP(S)

# Estructuras de datos

## Pila

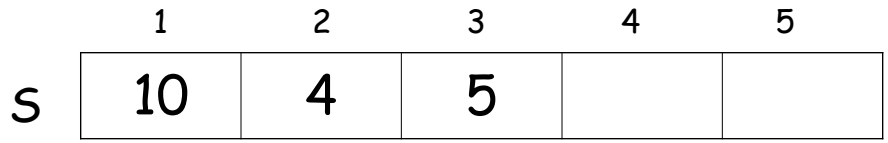
Una forma de implementar la pila es por medio de un arreglo unidimensional




Esto supone varios aspectos:

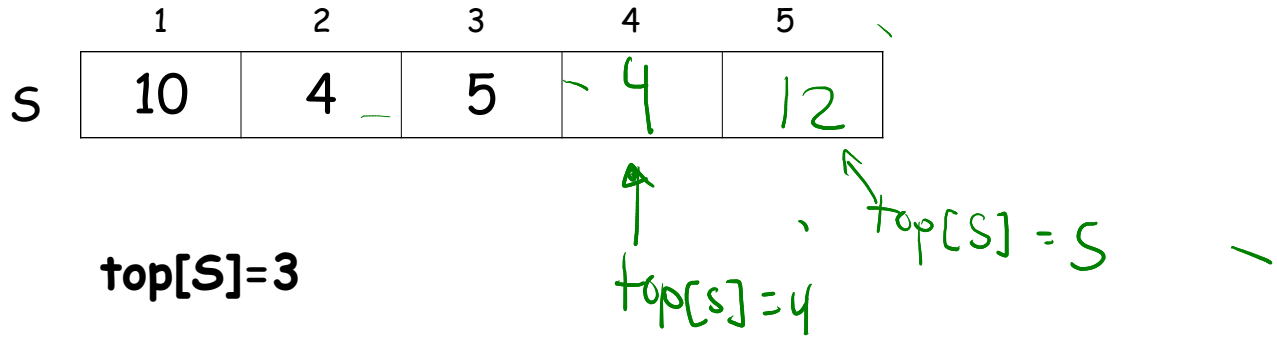
- La pila tiene una capacidad limitada
- Se cuenta con un atributo adicional, llamado top[S], que almacena el índice en el arreglo que guarda el último valor, esto es, el tope de la pila
- Cuando la pila esté vacía, top[S]=0

# Estructuras de datos




**top[S]=3**

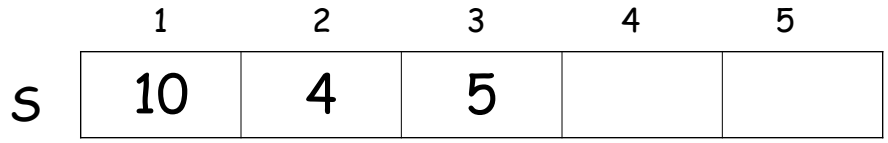
# Estructuras de datos




Indique lo que sucede después de cada instrucción, siendo S la pila que se muestre arriba:

STACK-EMPTY(S)
PUSH(S,4)
PUSH(S,12)
PUSH(S,7)

# Estructuras de datos

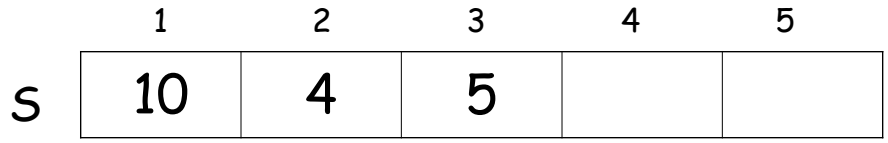



**top[S]=3**

Indique lo que sucede después de cada instrucción, siendo S la pila que se muestre arriba:

STACK-EMPTY(S)
PUSH(S,4)
PUSH(S,12)
PUSH(S,7) ← Overflow - desbordamiento en su capacidad máxima

# Estructuras de datos




**top[S]=3**

Indique lo que sucede después de cada instrucción, siendo S la pila que se muestre arriba:

STACK-EMPTY(S)
POP(S)
POP(S)
POP(S)
STACK-EMPTY(S)
POP(S) ← Underflow

# Estructuras de datos

Indique un algoritmo para cada operación básica, acompañado de su respectiva complejidad usando la notación O

- STACK-EMPTY(S), retorna true o false
- PUSH(S,x), adiciona x al tope, no devuelve ningún valor
- POP(S), borra el elemento que esté en el tipa y devuelve ese valor

# Estructuras de datos

```
STACK-EMPTY(S)
1  if (top[S]==0)
2   then return true
3   else return false
```

# Estructuras de datos

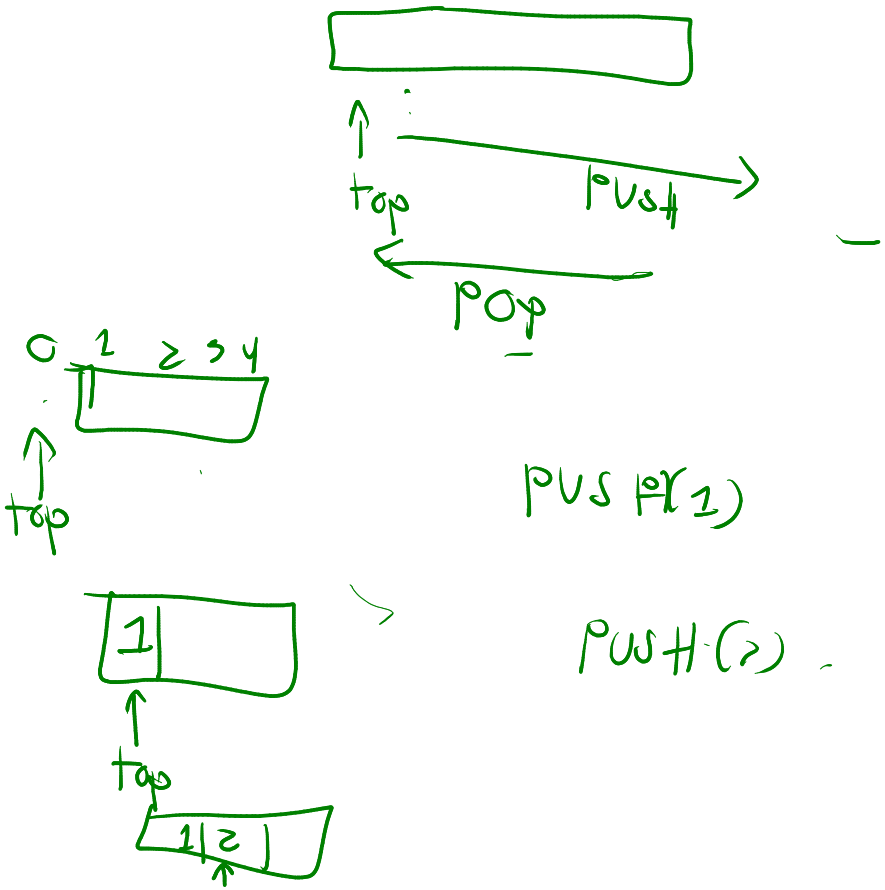
**STACK-EMPTY**(S)

1 if (top[S]==0)

2 then return true

3 else return false

T(n)=O(1), tiempo constante



# Estructuras de datos

## Cola

Una cola es una estructura de datos tipo FIFO (First In First Out), por lo que el primer elemento que es insertado, es el primero en ser borrado

Operaciones básicas:

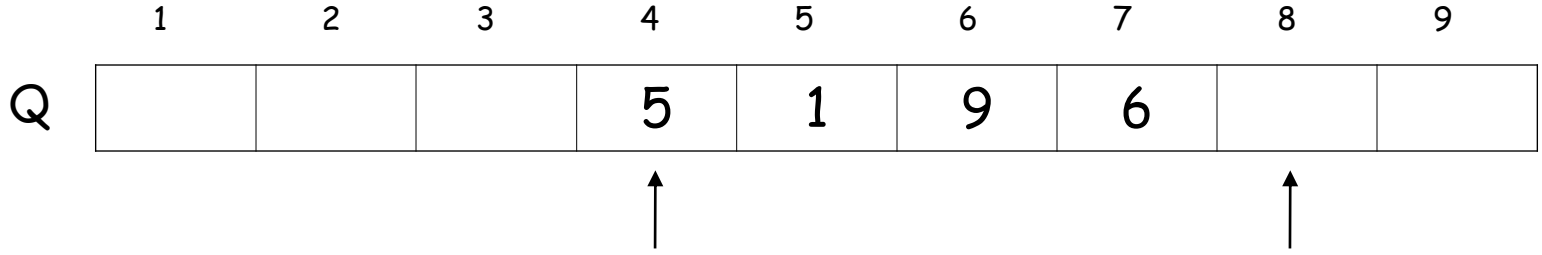
ENQUEUE(Q,x)

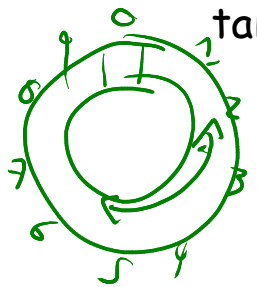
DEQUEUE(Q)

# Estructuras de datos

## Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional



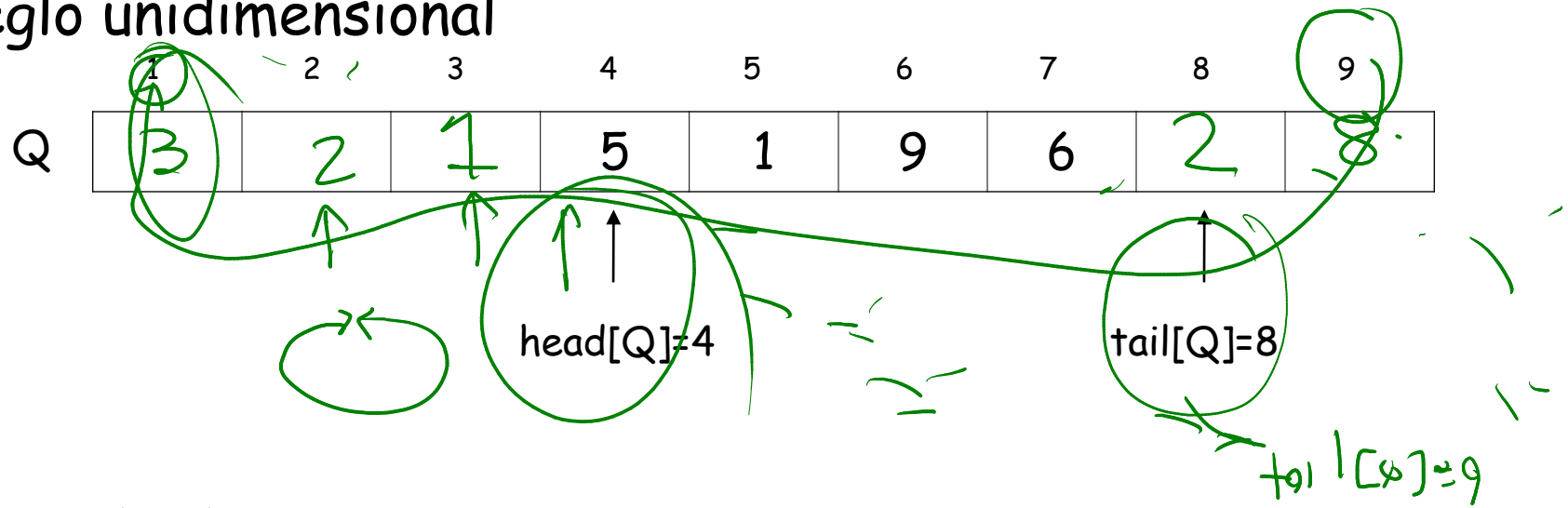
Esto supone varios aspectos:

- La cola tiene una capacidad limitada
- Se cuenta con dos atributos adicionales, head[Q] que guarda el indice de la cabeza y tail[Q] que apunta al siguiente lugar en el cual será insertado un elemento

# Estructuras de datos

## Cola

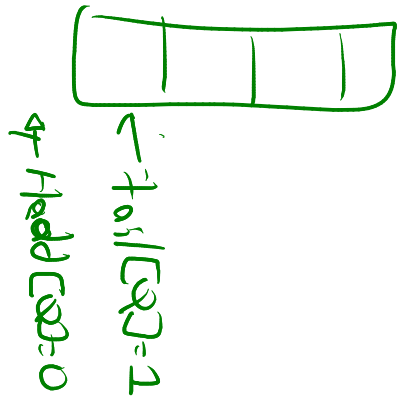
Una forma de implementar la cola es por medio de un arreglo unidimensional




ENQUEUE(Q,2)

ENQUEUE(Q,8) ← tail[Q]=1

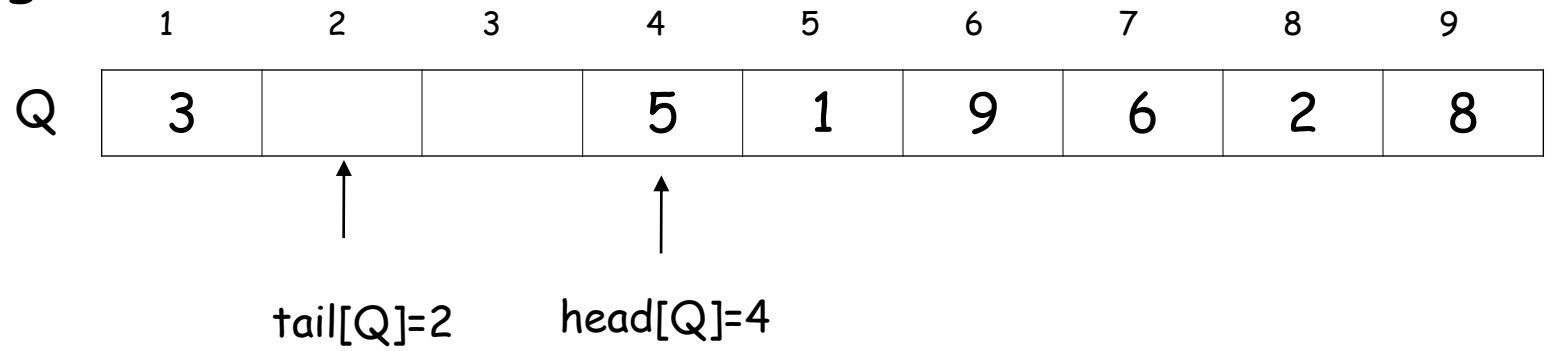
ENQUEUE(Q,3)



# Estructuras de datos

## Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional




ENQUEUE(Q,2)

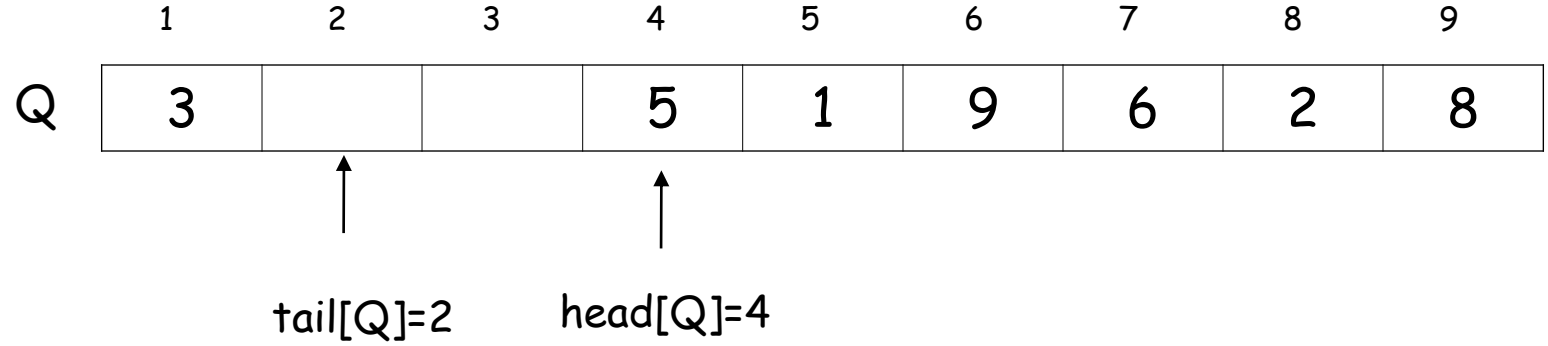
ENQUEUE(Q,8) ← Si se llega al final del arreglo, se intenta insertar en la posición 1

ENQUEUE(Q,3)

# Estructuras de datos

## Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional

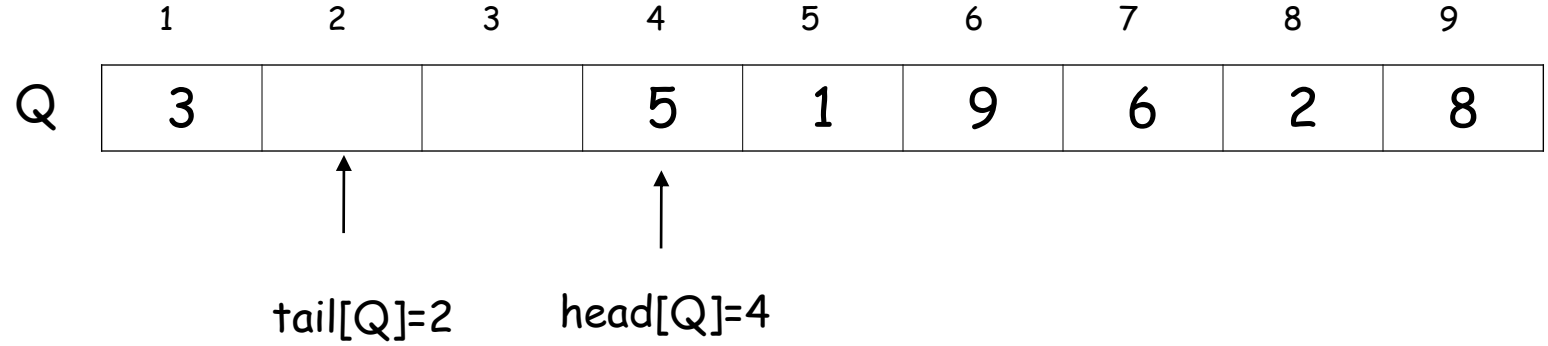



Cómo sabe que la cola está llena?

# Estructuras de datos

## Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional



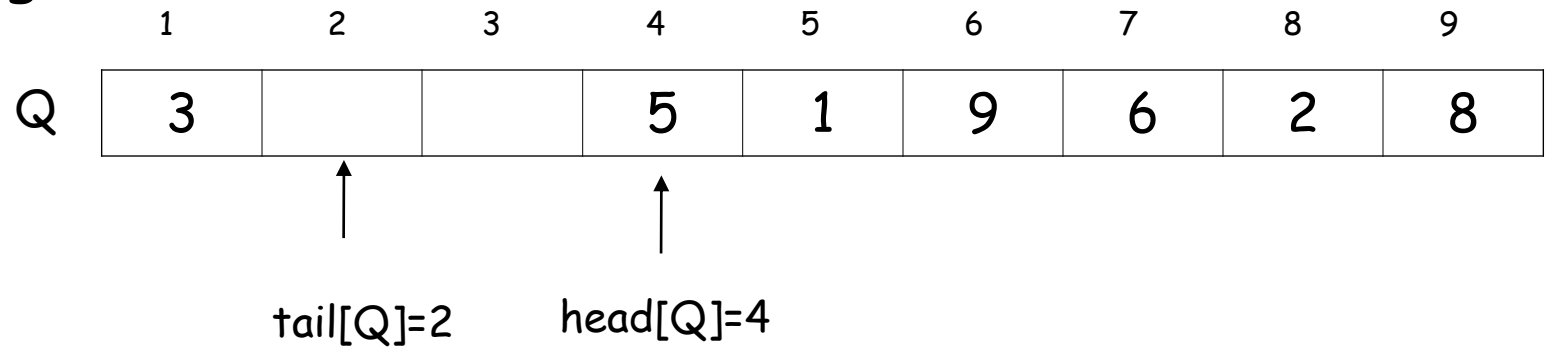

Cómo sabe que la cola está llena?

tail[Q]=head[Q]

# Estructuras de datos

## Cola

Una forma de implementar la cola es por medio de un arreglo unidimensional




Inicialmente tail[Q]=head[Q]=1

# Estructuras de datos

Indique un algoritmo para cada operación básica, acompañado de su respectiva complejidad usando la notación *O*

- ENQUEUE(Q,x)
- DEQUEUE(Q)

Arreglos: Reserva de espacio en memoria RAM
La idea es que tengan un tamaño fijo
Para acceder a los elementos usamos indices

Pos elemento = pos del arreglo + indice

Indice (considera el tiempo del arreglo, por ejemplo si el arreglo es tipo int 32 bits, long 64 bits ....)

Listas: Son colecciones de elementos que estan conectados, las listas manejan internamente las posiciones de memoria de los elementos, que no necesariamente son contiguos, estan hechas para insertar elementos rapidamente. Linkedlist ArrayList (libreria util), Python: list

Colas: Estructuras tipo FIFO, operaciones sobre el primer elemento de la estructura.

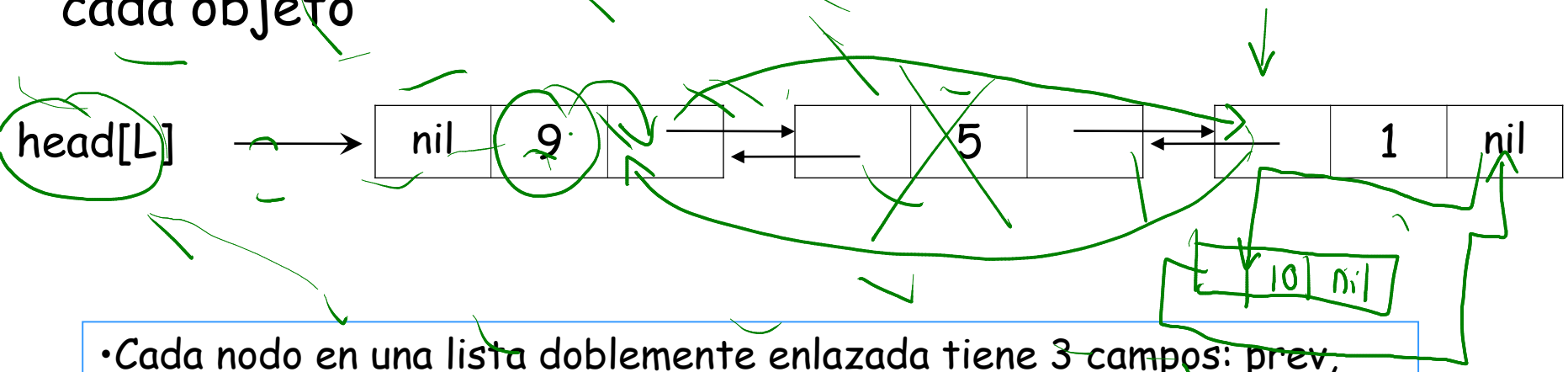
Pilas: Estructura tipo LIFO, operaciones sobre último elemento insertado

Pilas y Colas solo trabaja con un ELEMENTO a la vez

# Estructuras de datos

## Listas doblemente enlazadas

Es una estructura de datos en la cual los objetos son organizados en un orden lineal. A diferencia de los arreglos, el orden en las listas está dado por un puntero a cada objeto

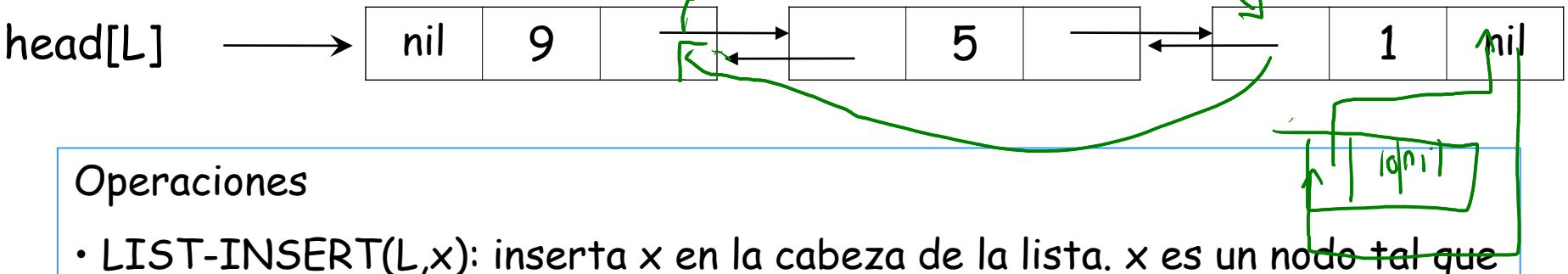



- Cada nodo en una lista doblemente enlazada tiene 3 campos: prev, key y next
- Se tiene además un puntero al primer nodo

# Estructuras de datos

## Listas doblemente enlazadas

Es una estructura de datos en la cual los objetos son organizados en un orden lineal. A diferencia de los arreglos, el orden en las listas está dado por un puntero a cada objeto




Operaciones

- LIST-INSERT(L,x): inserta x en la cabeza de la lista. x es un nodo tal que key[x]=k, y prev=next=nil
- LIST-DELETE(L,x): donde x es el nodo que se desea borrar
- LIST-SEARCH(L,k): busca el primer nodo que tiene llave k y retorna un puntero a ese nodo

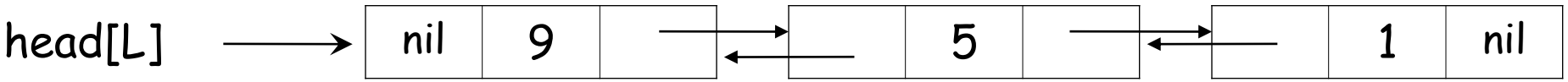
# Estructuras de datos

*LIST-SEARCH busca el primer nodo que tiene llave k y retorna un puntero a ese nodo*

```
LIST-SEARCH(L,k)
1. x←head[L]
2. while x≠nil and key[x]≠k
3.         x←next[x]
4.  return x
```

lineal O(n)

¿Cuál es la complejidad en el peor caso?

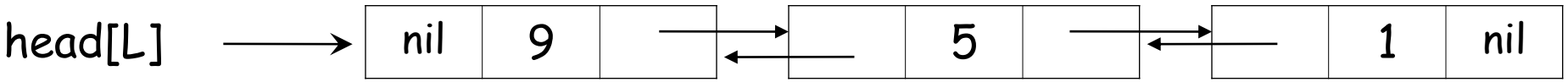


# Estructuras de datos

*LIST-SEARCH busca el primer nodo que tiene llave k y retorna un puntero a ese nodo*

```
LIST-SEARCH(L,k)
1. x←head[L]
2. while x≠nil and key[x]≠k
3.         x←next[x]
4.  return x
```

En el peor caso será O(n)



# Estructuras de datos

Indique el resultado de realizar las siguientes operaciones:

prev[z]=nil

next[z]=nil

key[z]=10

LIST-INSERT(L,z)

prev[w]=nil

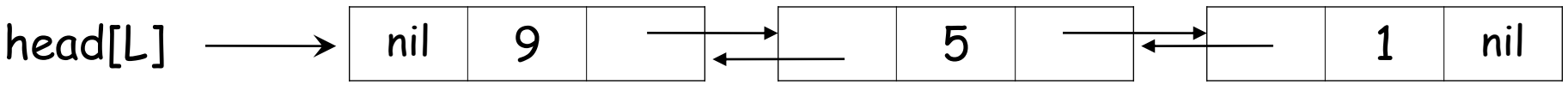
next[w]=nil

key[w]=8

LIST-INSERT(L,w)

x=LIST-SEARCH(L,10)

LIST-DELETE(L,x)



# Estructuras de datos

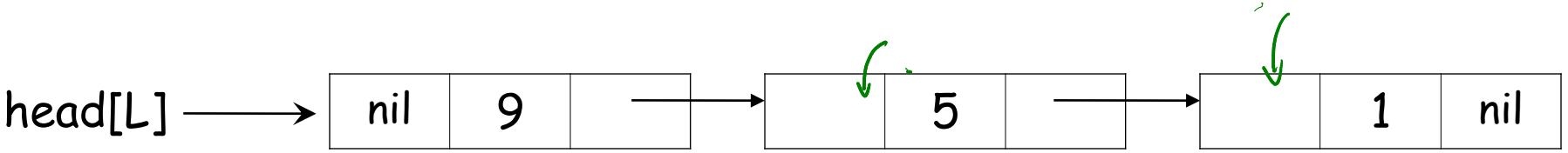
Indique el algoritmo para las siguiente operaciones y muestre su complejidad en el peor caso:

- LIST-INSERT(L,x): inserta x en la cabeza de la lista. x es un nodo tal que key[x]=k, y prev=next=nil

- LIST-DELETE(L,x): donde x es el nodo que se desea borrar

# Estructuras de datos

## Listas simplemente enlazada




Operaciones

- LIST-INSERT(L,x): inserta x al final de la lista. x es un nodo tal que key[x]=k, y prev=next=nil
- LIST-DELETE(L): donde x es el nodo al final de la lista
- LIST-SEARCH(L,k): busca el primer nodo que tiene llave k y retorna un puntero a ese nodo

# Referencias

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press. Chapter 10

# Gracias

Próximo tema:

Estructuras de datos: Conjuntos, tuplas y diccionarios