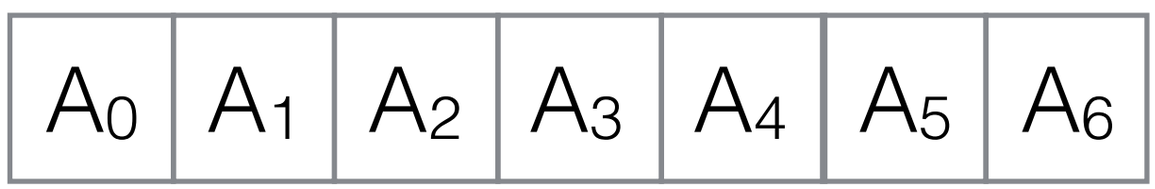
Data Structures in Java

Lecture 2: Array and Linked Lists.

9/9/2015

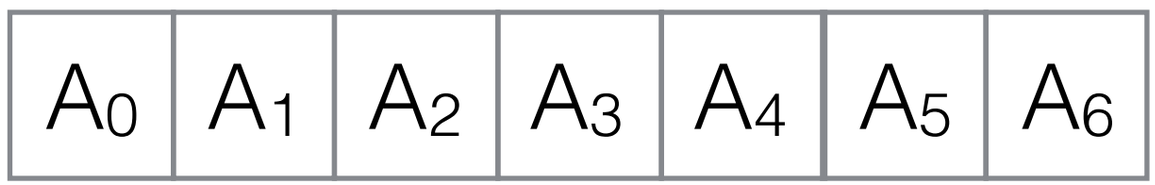
Daniel Bauer

The List ADT



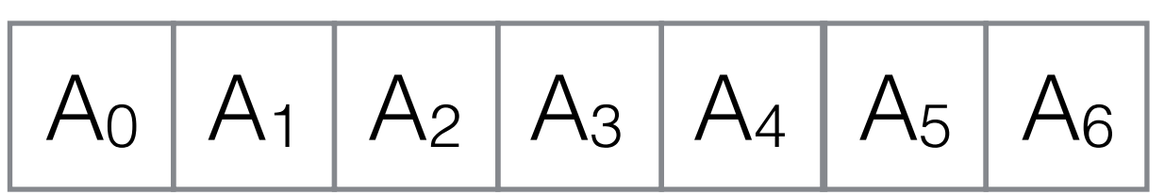
The List ADT

- A list *L* is a sequence of *N* objects A_0, A_1, A_2, ..., A_{N-1}



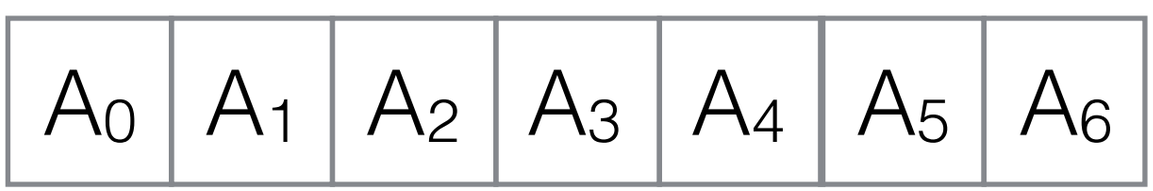
The List ADT

- A list *L* is a sequence of *N* objects A_0, A_1, A_2, ..., A_{N-1}
- *N* is the length/size of the list. List with length N=0 is called the *empty list.*



The List ADT

- A list *L* is a sequence of *N* objects A_0, A_1, A_2, ..., A_{N-1}
- *N* is the length/size of the list. List with length N=0 is called the *empty list.*
- A_i *follows/succeeds* A_{i-1} for i > 0.

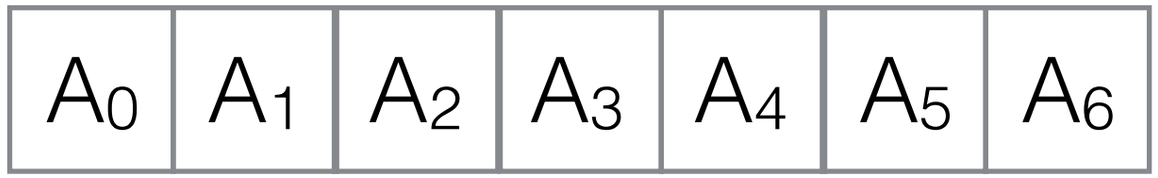


The List ADT

- A list *L* is a sequence of *N* objects A_0, A_1, A_2, ..., A_{N-1}
- *N* is the length/size of the list. List with length N=0 is called the *empty list.*
- A_i *follows/succeeds* A_{i-1} for i > 0.
- A_i *precedes* A_{i+1} for i < N.

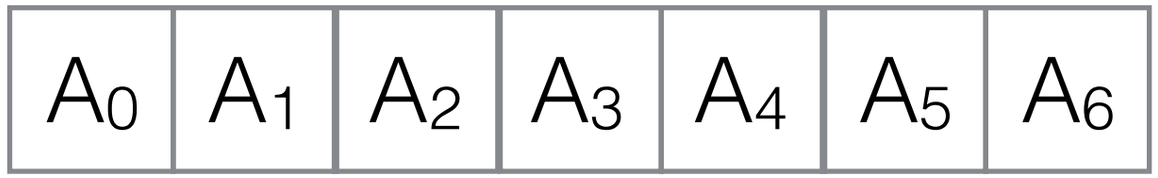
A_0	A_1	A_2	A_3	A_4	A_5	A_6

Typical List Operations



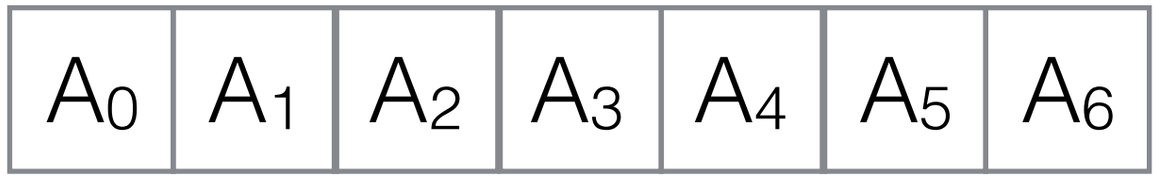
Typical List Operations

- `void printList()`



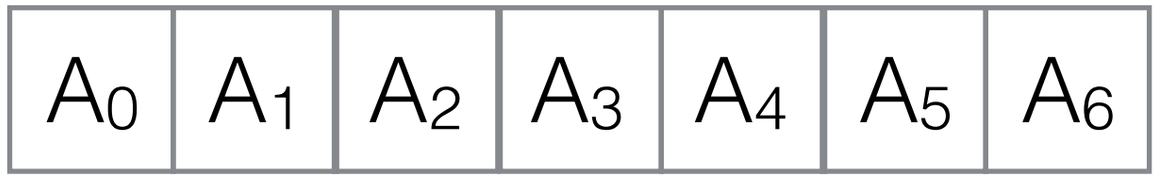
Typical List Operations

- `void printList()`
- `void makeEmpty()`



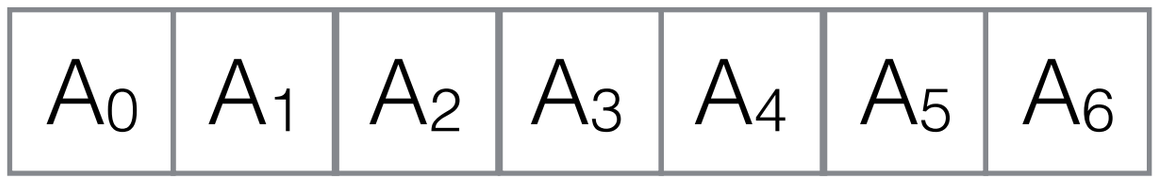
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`



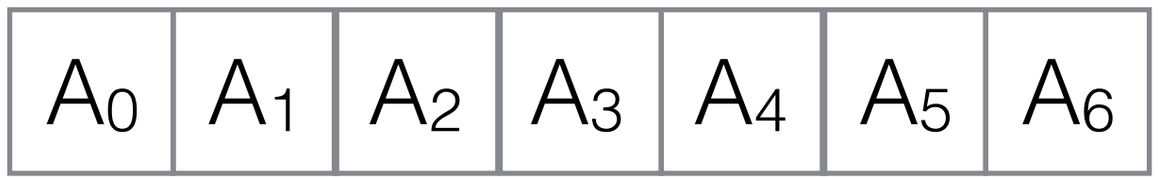
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`



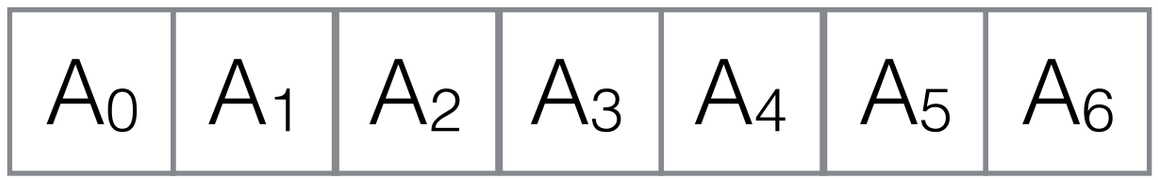
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`
- `boolean insert(x, k), append(x)`



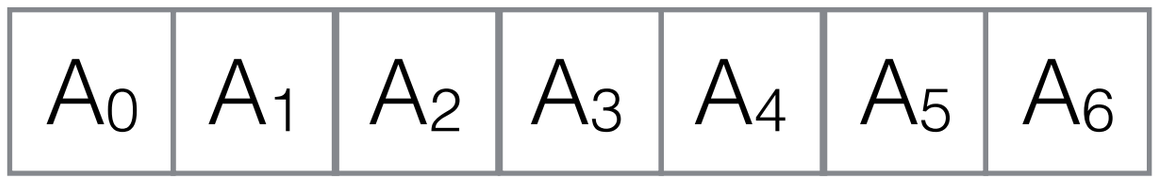
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`
- `boolean insert(x, k), append(x)`
- `boolean remove(k)`



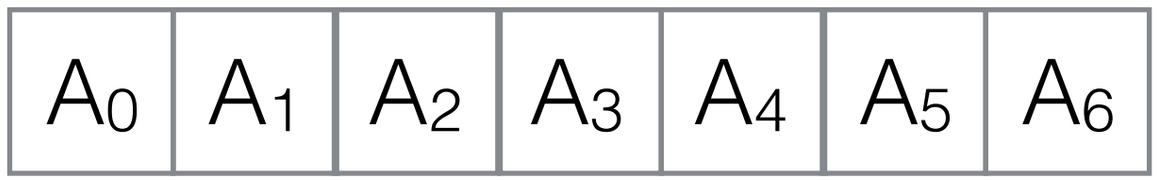
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`
- `boolean insert(x, k), append(x)`
- `boolean remove(k)`
- `int find(x) / indexOf(x)`



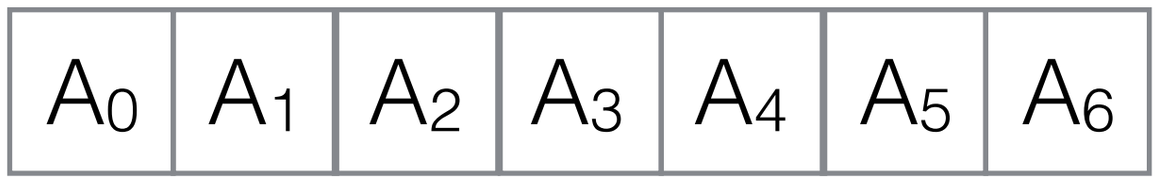
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`
- `boolean insert(x, k), append(x)`
- `boolean remove(k)`
- `int find(x) / indexOf(x)`
- `Object next()`



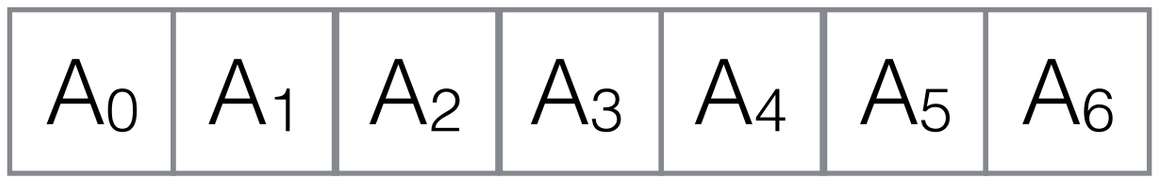
Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`
- `boolean insert(x, k), append(x)`
- `boolean remove(k)`
- `int find(x) / indexOf(x)`
- `Object next()`
- `Object previous()`



Typical List Operations

- `void printList()`
- `void makeEmpty()`
- `int size()`
- `Object findKth(k) / get(k)`
- `boolean insert(x, k), append(x)`
- `boolean remove(k)`
- `int find(x) / indexOf(x)`
- `Object next()`
- `Object previous()`
- `void removeCurrent()`



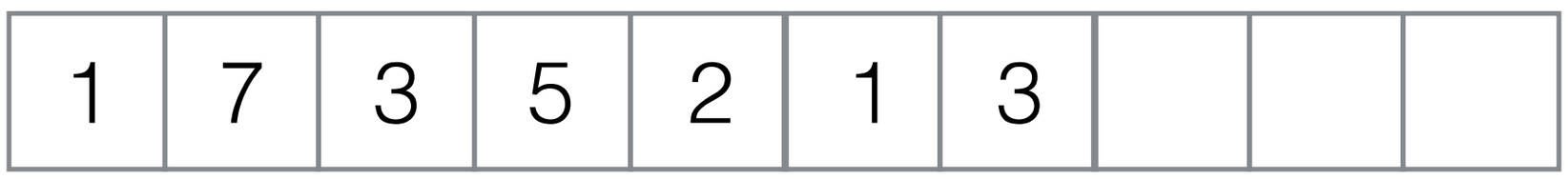
Array Lists

- Just a thin layer wrapping an array.

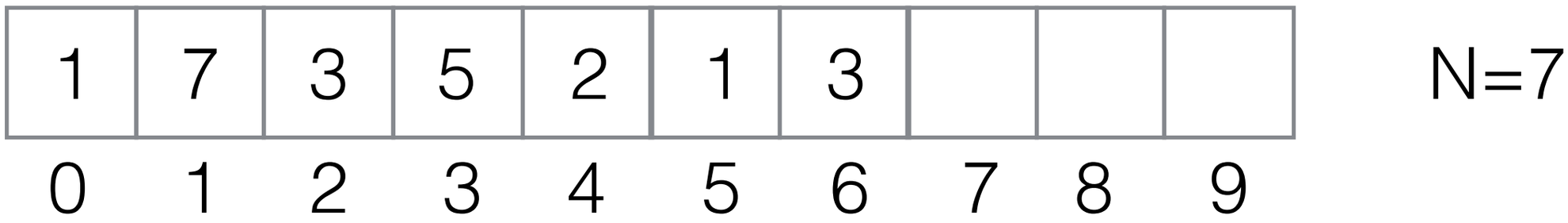
```
public class SimpleArrayList implements List{

    public static final int DEFAULT_CAPACITY = 10;
    private int theSize;
    private Integer[] theItems;

    public SimpleArrayList() {
        theItems = new Integer[DEFAULT_CAPACITY];
    }
}
```

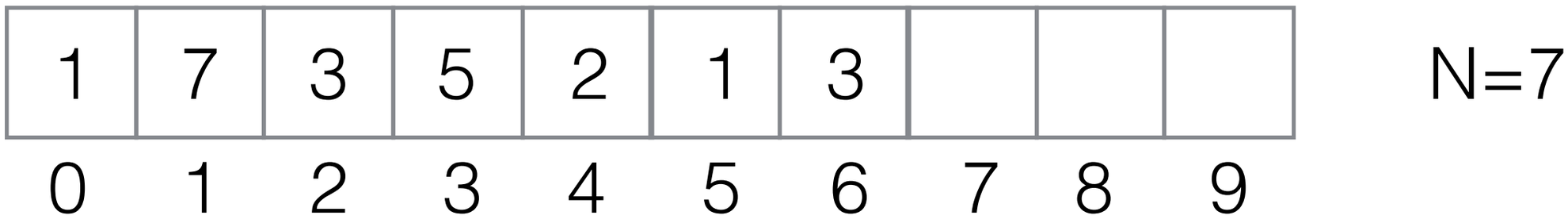


Running Time for Array List Operations



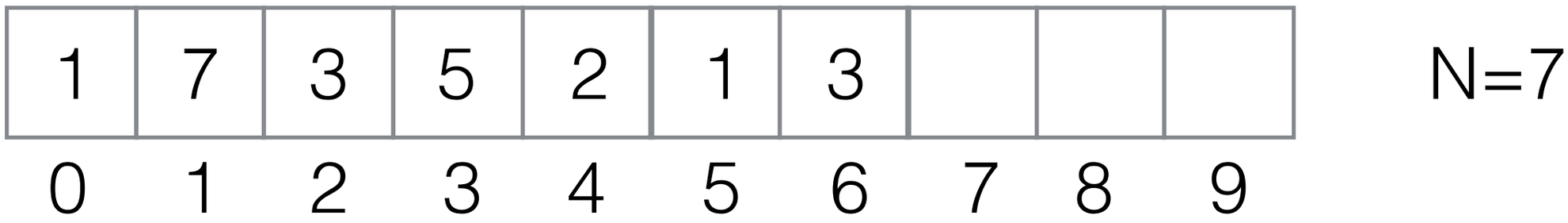
Operation	Number of Steps
printList	
find(x)	
findKth(k)	
insert(x,k)	
remove(x)	

Running Time for Array List Operations



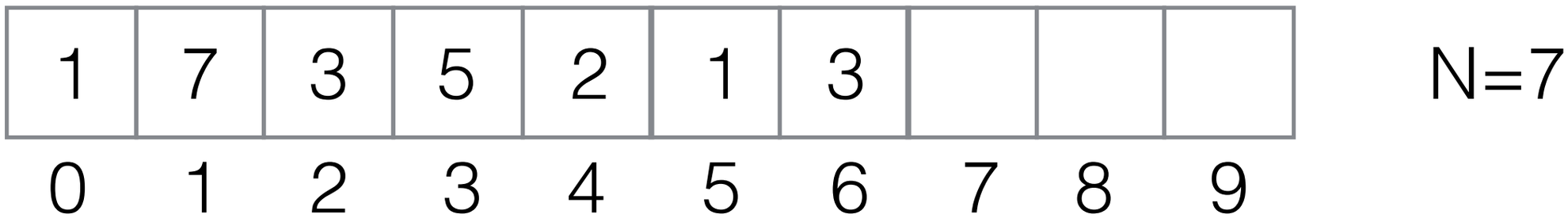
Operation	Number of Steps
printList	N
find(x)	N
findKth(k)	
insert(x,k)	
remove(x)	

Running Time for Array List Operations



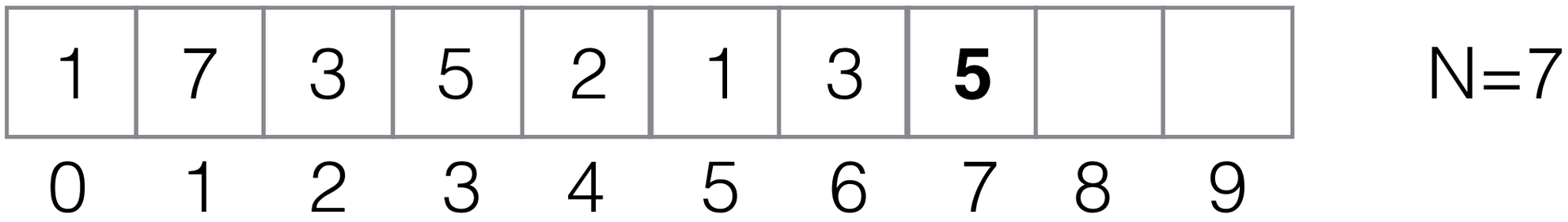
Operation	Number of Steps
printList	N
find(x)	N
findKth(k)	1
insert(x,k)	
remove(x)	

Array List: Insert/Remove



insert(x,k)	
remove(x)	

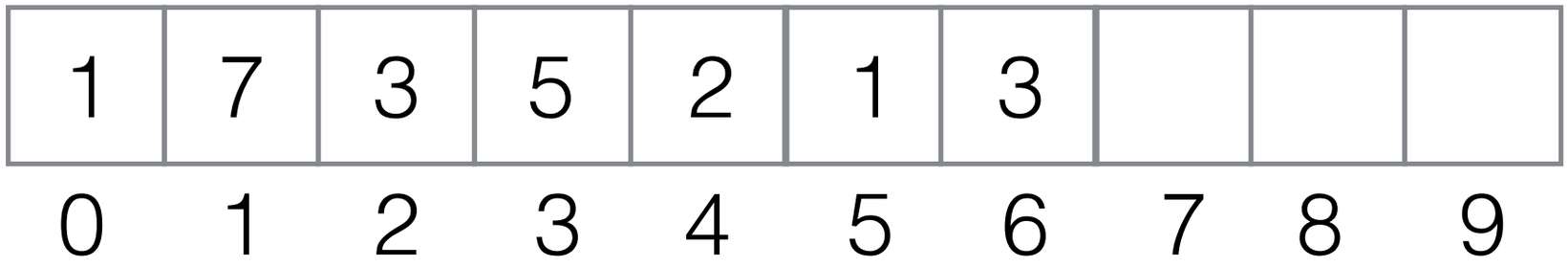
Array List: Insert/Remove



insert(5,7): 1 step

insert(x,k)	
remove(x)	

Array List: Insert/Remove



N=7

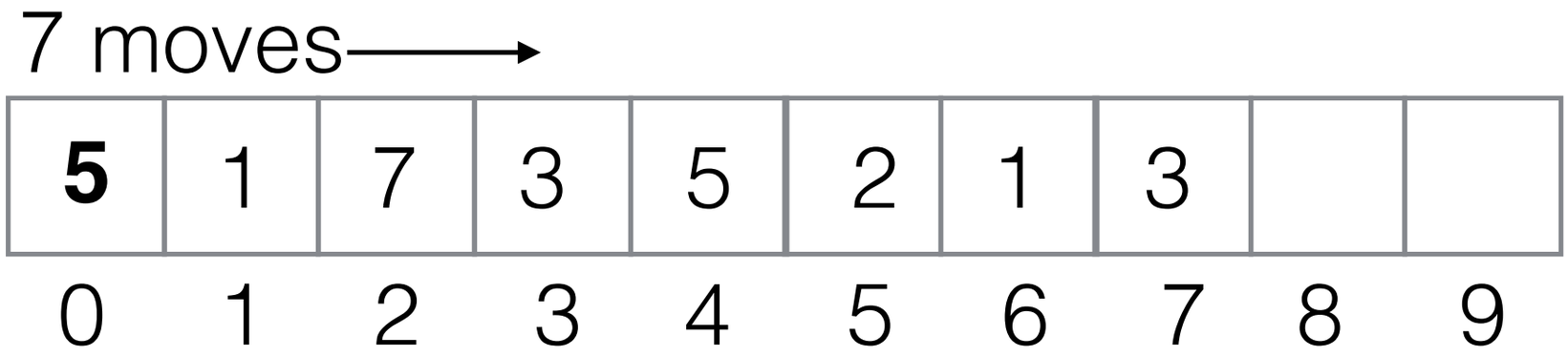
insert(5,7): 1 step

remove(7): 1 step

best case

insert(x,k)	
remove(x)	

Array List: Insert/Remove



insert(5,7): 1 step

remove(7): 1 step

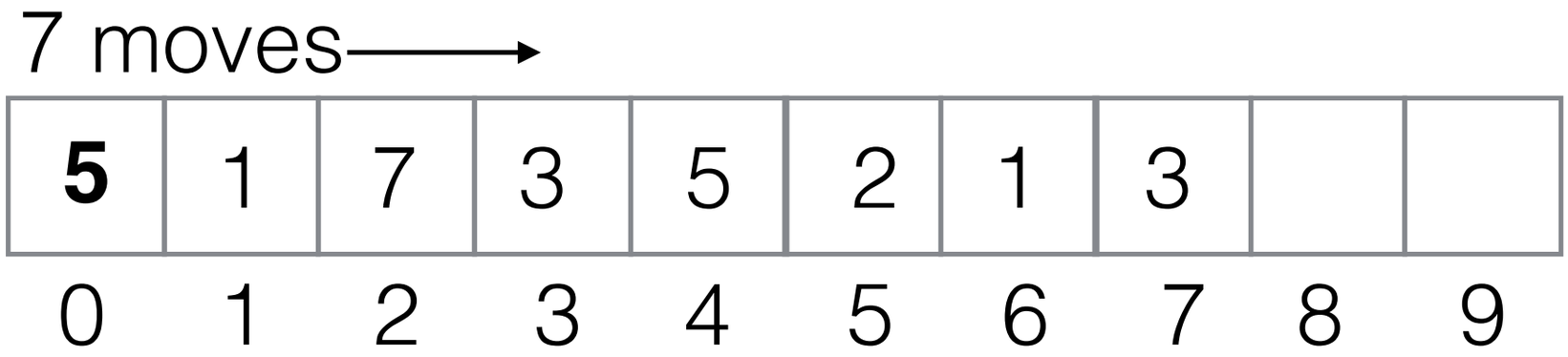
best case

insert(5,0): 7 steps

worst case

insert(x,k)	
remove(x)	

Array List: Insert/Remove



insert(5,7): 1 step

remove(7): 1 step

best case

insert(5,0): 7 steps

remove(0): O(N)

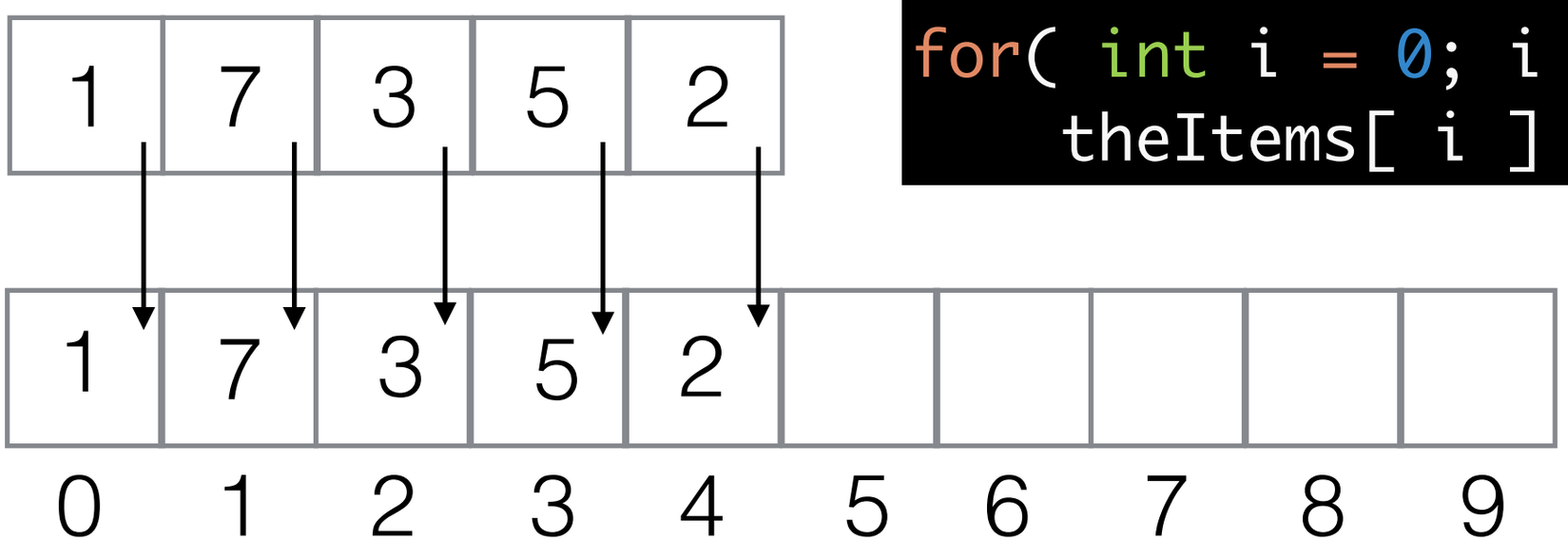
worst case

insert(x,k)	N
remove(x)	N

Expanding Array Lists

- What if we are running out of space during `append/insert`
- first copy all elements into a new array of sufficient size

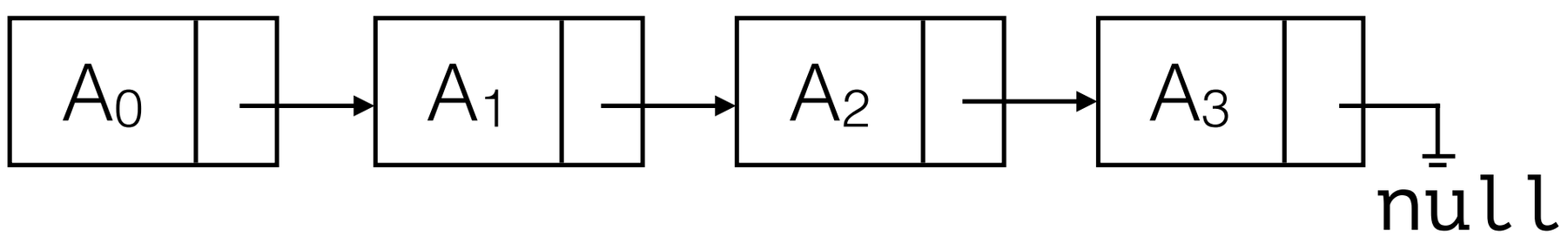
```
newCapacity = arr.length * 2;
Integer[ ] old = theItems;
theItems = new Integer[newCapacity];
for( int i = 0; i < size( ); i++ )
    theItems[ i ] = old[ i ];
```



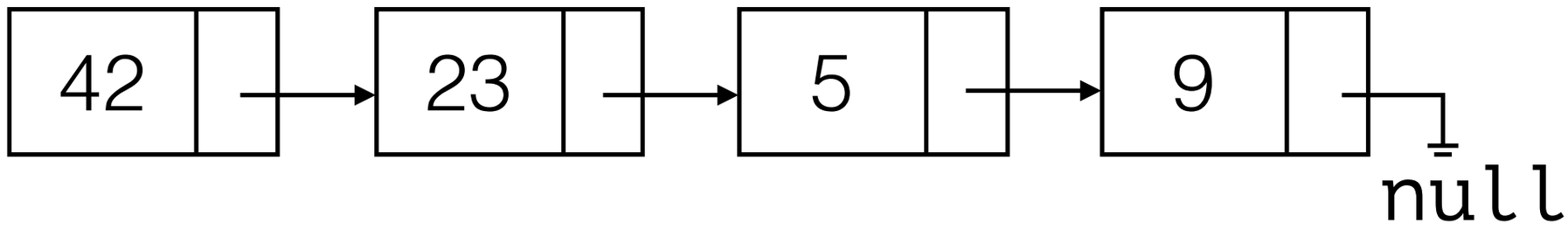
Simple Linked Lists

- Series of *Nodes*. Each *Node* contains:
  - A reference to the data object it contains.
  - A reference to the next node in the List.

```
public class Node {
    public Integer data;
    public Node next;
    public Node(Integer d, Node n) {
        data = d;
        next = n;
    }
}
```

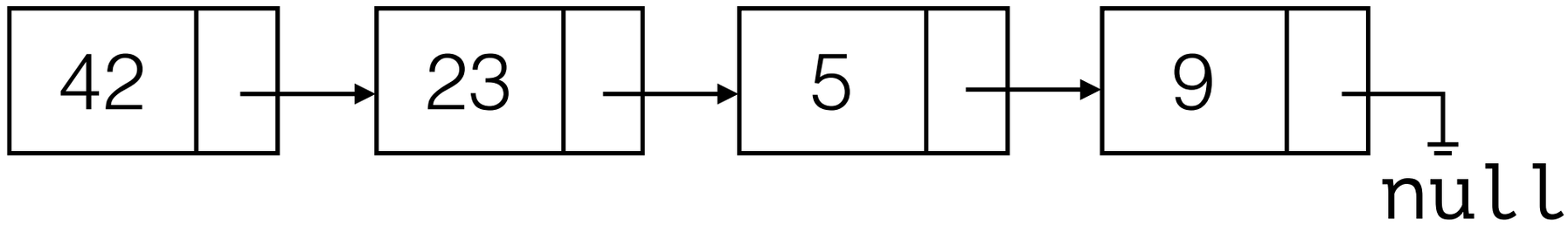


Running Time for Simple Linked List Operations



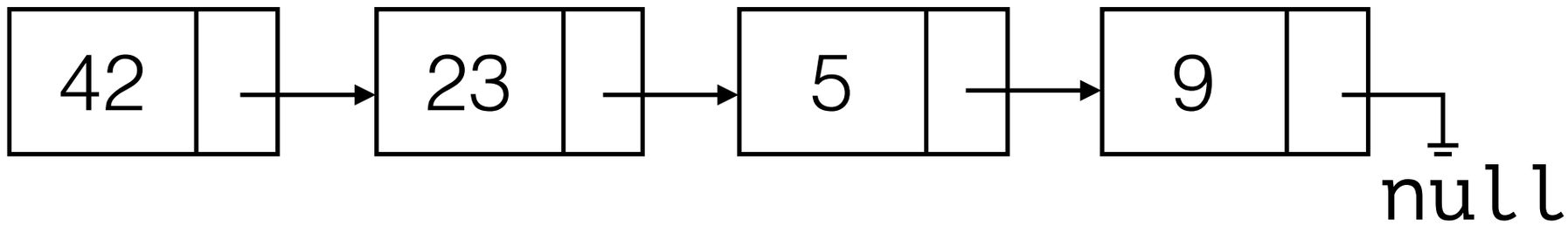
printList	
find(x)	
findKth(k)	
next()	

Running Time for Simple Linked List Operations



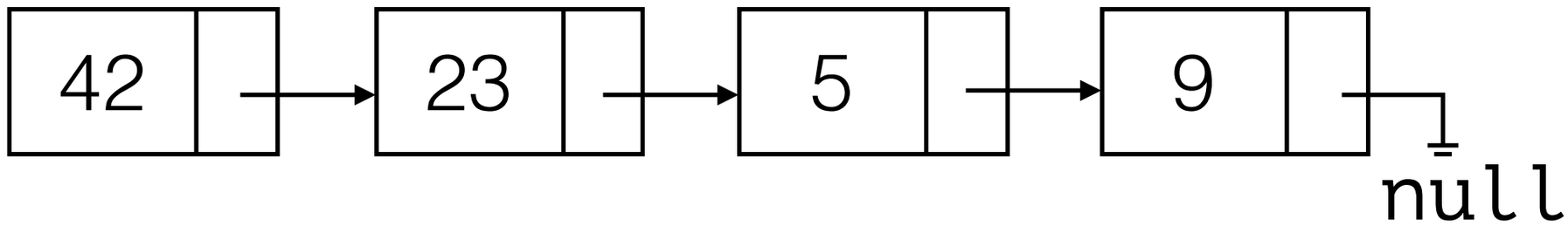
printList	N
find(x)	
findKth(k)	
next()	

Running Time for Simple Linked List Operations



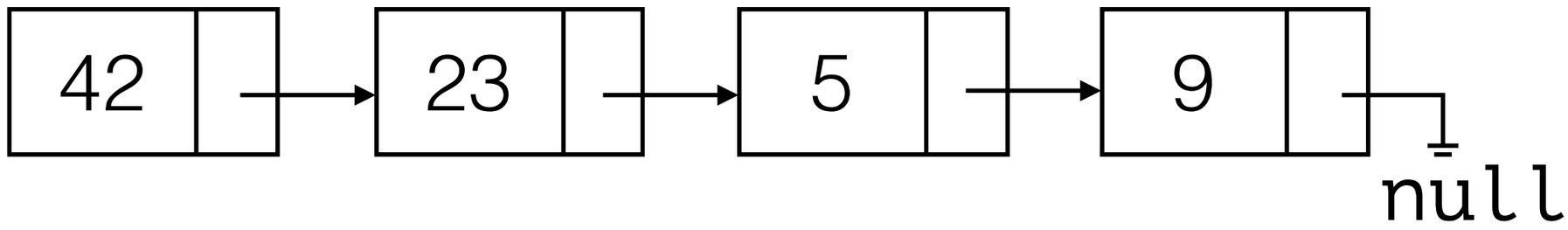
printList	N
find(x)	N
findKth(k)	
next()	

Running Time for Simple Linked List Operations



printList	N
find(x)	N
findKth(k)	k
next()	

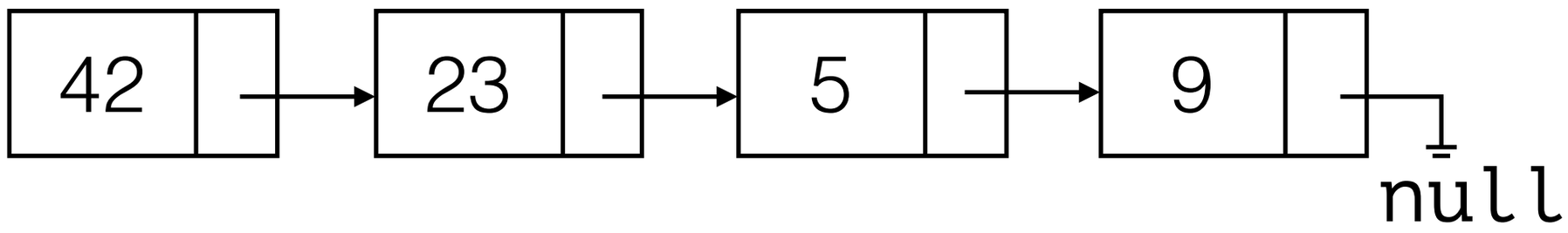
Running Time for Simple Linked List Operations



printList	N
find(x)	N
findKth(k)	k
next()	1

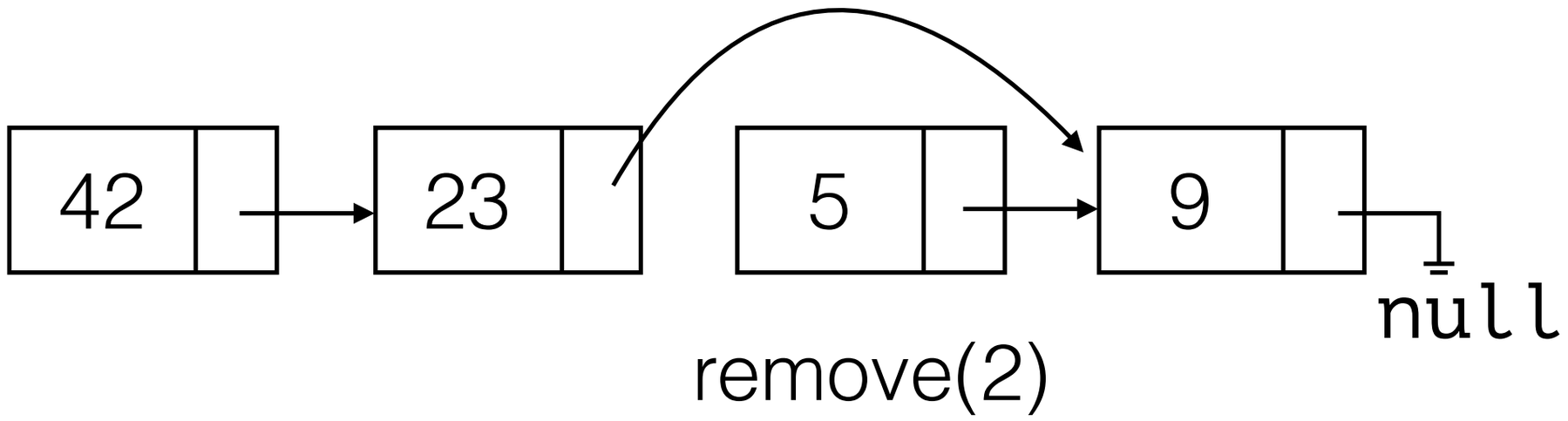
In many applications we can use next() instead of findKth(k).
(for every element in the list do… / filter the list …)

Simple Linked List Removal



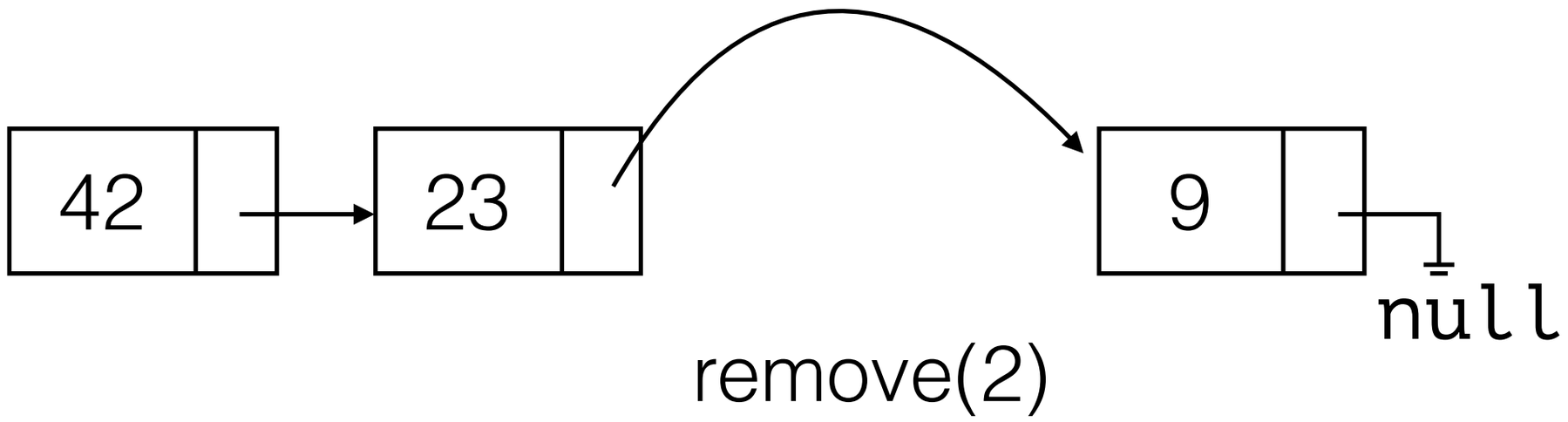
findKth(k)	k
next()	1
insert(x,k)	
remove(k)	

Simple Linked List Removal



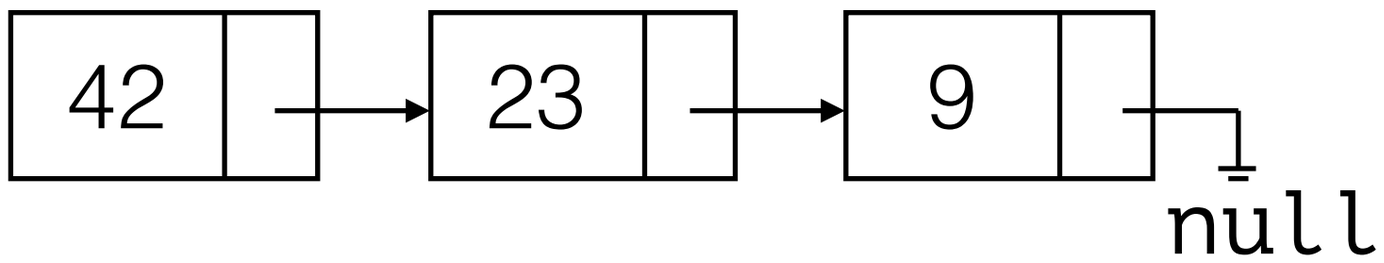
findKth(k)	k
next()	1
insert(x,k)	
remove(k)	search time + 1

Simple Linked List Removal



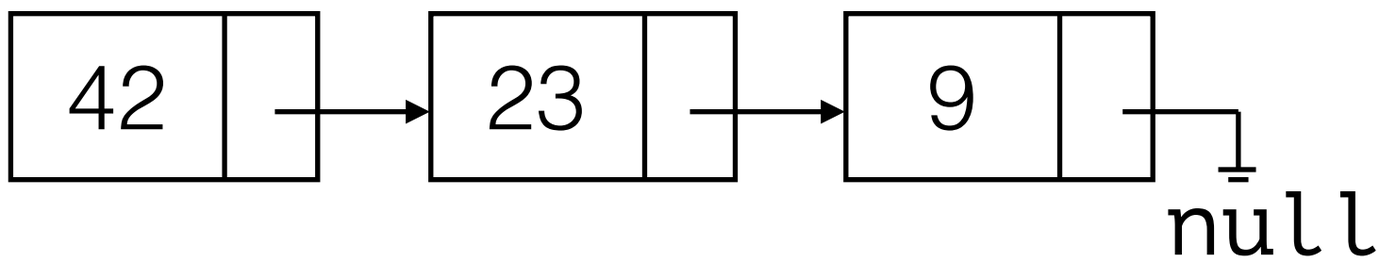
findKth(k)	k
next()	1
insert(x,k)	
remove(k)	search time + 1

Simple Linked List Insertion



insert(x,k)	

Simple Linked List Insertion



insert(5,2)

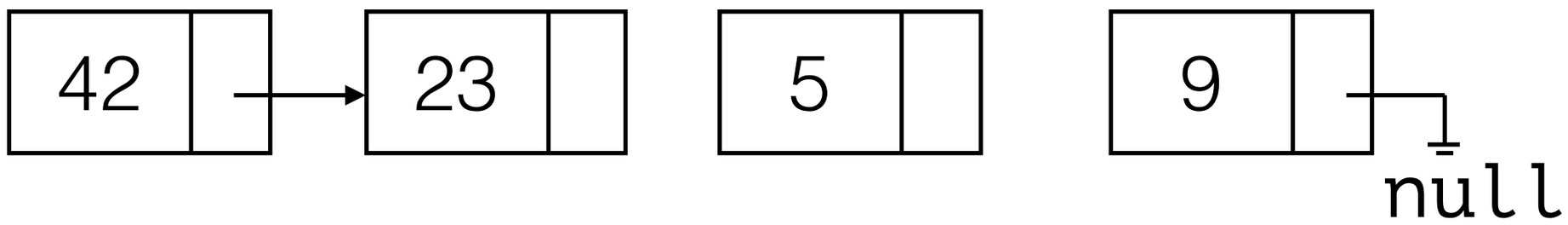
insert(x,k)	

Simple Linked List Insertion

insert(5,2)

insert(x,k)	

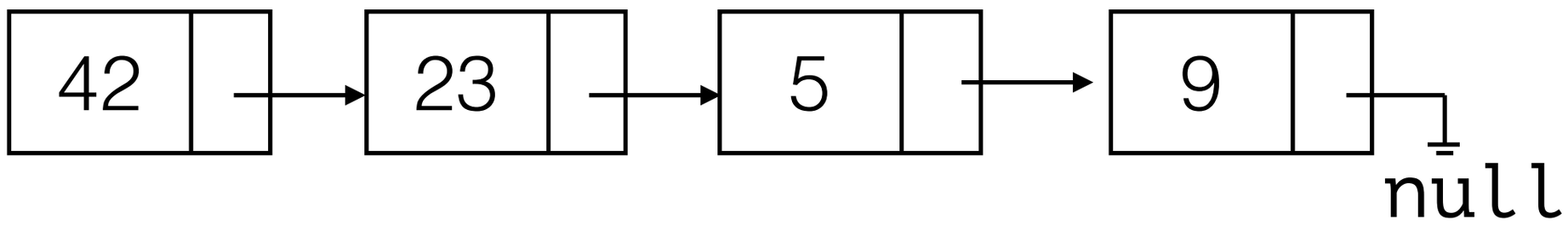
Simple Linked List Insertion



insert(5,2)

insert(x,k)	

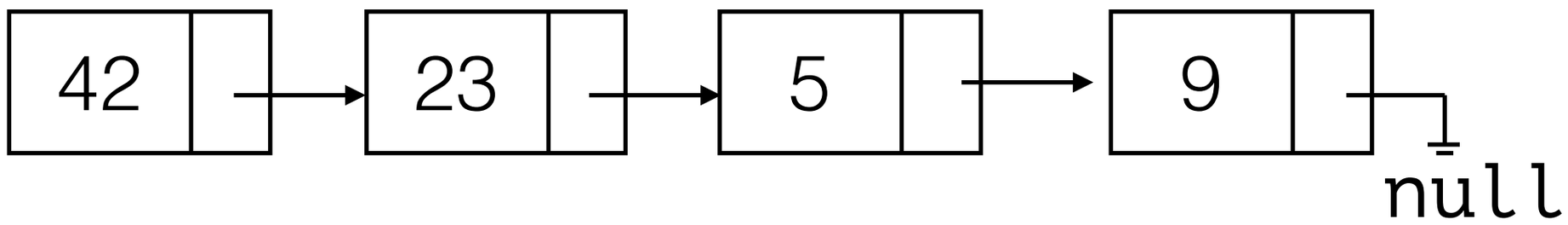
Simple Linked List Insertion



insert(5,2)

insert(x,k)	

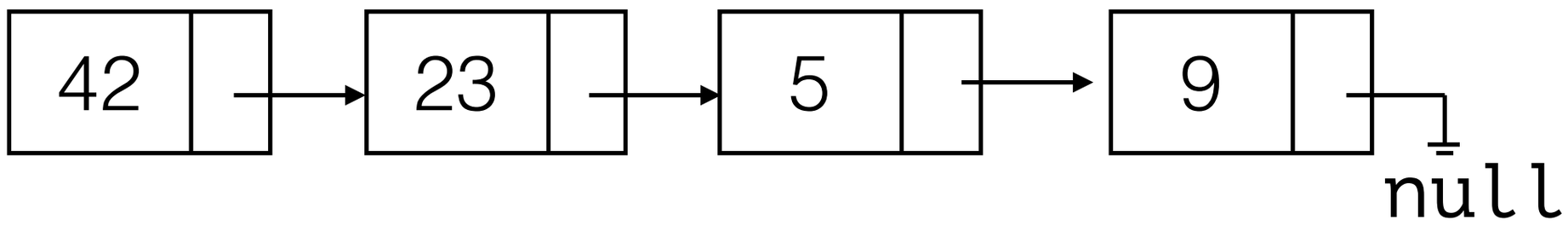
Simple Linked List Insertion



insert(5,2)

insert(x,k)	search time +1

Simple Linked List Insertion

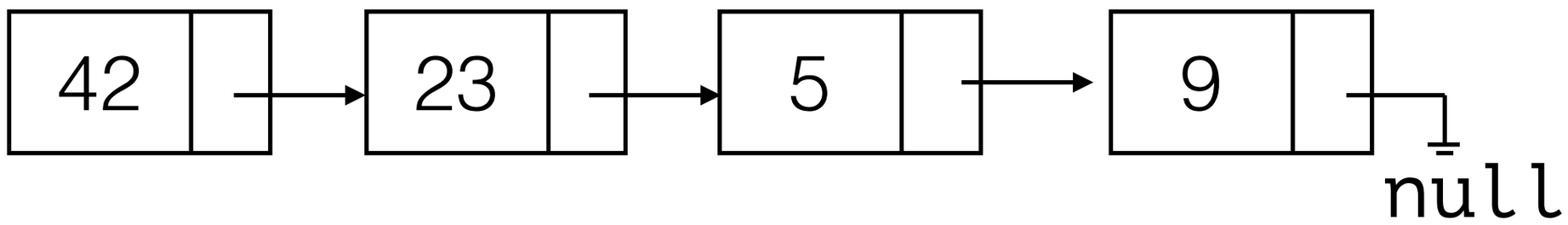


insert(5,2)

insert(x,k)	search time +1

Inserting in position 0?
Inserting in position N-1?

Simple Linked List Insertion



insert(5,2)

insert(x,k)	search time +1

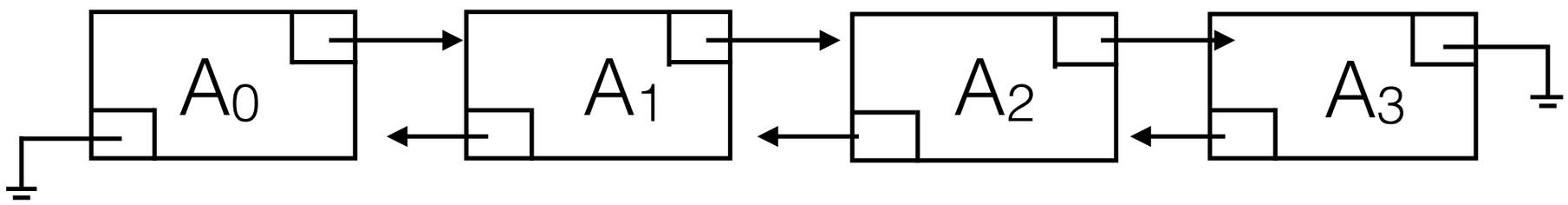
Inserting in position 0?
Inserting in position N-1?

Linked list should remember the first and last object.

Doubly Linked Lists

- Also maintain reference to previous node in the list.
- Speeds up append at end of list.

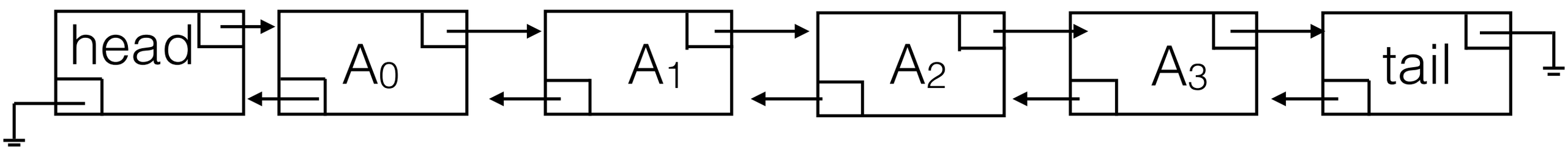
```
private class Node {
    public Integer data;
    public Node next;
    public Node prev;
    public Node(Integer d, Node n, Node p) {
        data = d; next = n; prev = n;
    }
}
```



Doubly Linked List with Sentinel Nodes

- header node, tail node
- make implementation of `next` / `previous` easier.
- Remove other special cases

(e.g. removing first node/last node)



Empty Doubly Linked List with Sentinel Nodes