Data Structures and Algorithms I

Jingsha He, Ph.D.
jhe@bjut.edu.cn
Faculty of Information Technology
Beijing University of Technology

北京工業大學 软件学院
School of Software Engineering, Beijing University of Technology

The List Abstract Data Type - Singly-Linked List

Learning Outcomes

- After this lecture and the related practical, students should...
  - understand the operations of the list and position abstract data types
  - be able to implement a singly-linked list
  - know the complexity of all of the operations of the singly-linked list implementation

Table of Contents

- The List Abstract Data Type
- Singly-Linked List Implementation
  - Algorithmic Complexity

Table of Contents

- List Abstract Data Type
- Singly-Linked List Implementation
  - Algorithmic Complexity

List Abstract Data Type

- Concept
  - The list abstract data type models a sequence of positions
  - Each position store one piece of data
  - There is a before/after relationship between the positions
    - This allows for efficient insertion into and deletion from anywhere in the list

Position Abstract Data Type

- Before we talk about the list abstract data type, let's first look at the position abstract data type
- Concept
  - Position abstract data type models the idea of one place in a data structure into which a single data item is stored
  - Position provides a simple view of many different ways of storing a single data item
    - An element in an array
    - A node in a linked list

Position Abstract Data Type

- Specification
  - Operation
    - element(): this returns the piece of data that is stored in a position
- Interface

```
public interface Position {
    public int element();
}
```

List Abstract Data Type: Specification

- Operations
  - first(): return the first position in the list
  - last(): return the last position in the list
  - before(p): return the position in the list before p
  - after(p): return the position in the list after p
  - insertBefore(p, d): insert the value d into the position in the list before p
  - insertAfter(p, d): insert the value d into the position in the list after p
  - insertFirst(d): insert the value d into the first position in the list
  - insertLast(d): insert the value d into the last position in the list
  - remove(p): remove the position p from the list
  - size(): return the number of elements stored in the list
  - isEmpty(): return true if the list is empty and false otherwise

List Abstract Data Type

- Interface

```
public interface List {
    public Position first();
    public Position last();
    public Position before(Position p);
    public Position after(Position p);
    public Position insertBefore(Position p, int d);
    public Position insertAfter(Position p, int d);
    public Position insertFirst(int d);
    public Position insertLast(int d);
    public int remove(Position p);
    public int size();
    public boolean isEmpty();
}
```

List Abstract Data Type

- Implementation strategies
  - Array based implementation
  - Link based implementation
    - Singly-linked list
      - Each Position object keeps a reference to the next Position in the sequence
    - Doubly-linked list
      - Each Position object keeps a reference to the next and the previous Positions in the sequence

Table of Contents

Singly-Linked List Implementation

- We create a Node class that implements the Position interface
- We add functionality to the class to store a Node in the sequence

```
public class Node implements Position {
    private int element;
    Node next;

    public Node(int e) {
        this.element = e;
    }

    public int element() {
        return element;
    }
}
```

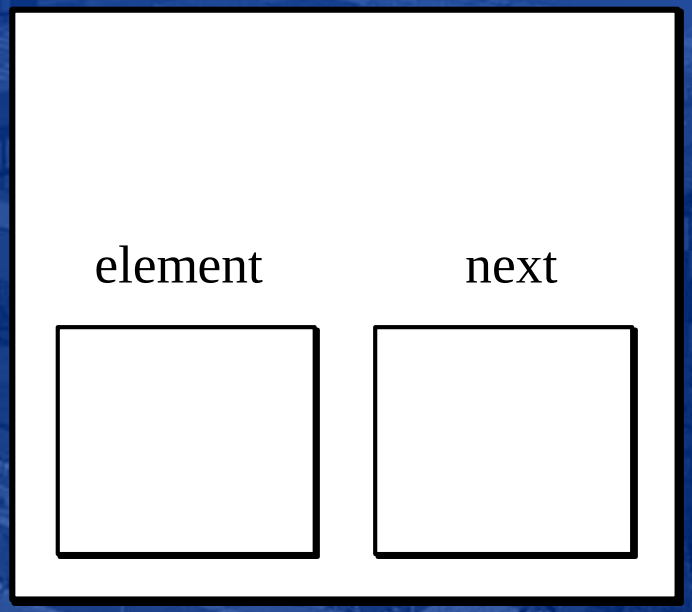
Singly-Linked List Implementation

- We keep a reference to the first position in the list
- We update the references when necessary
- We keep count of the number of positions in the list
- Variables
  - A reference to the first position in the list

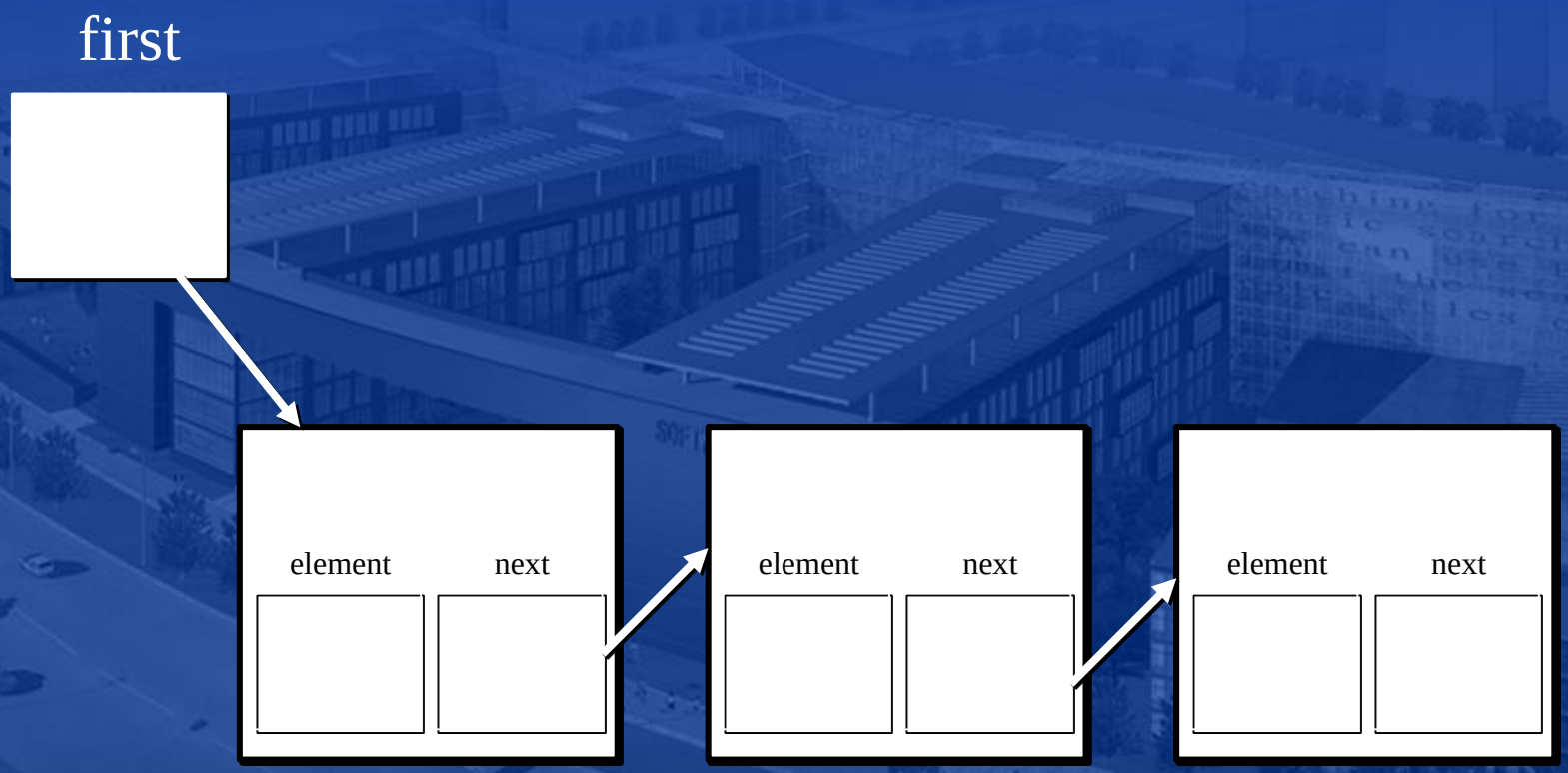
    private Node first;
  - A number to keep track of the number of nodes in the list

    private int size;

Representation of a Node Object



Representation of a List



Singly-Linked List Operations

- first()
  - Return the reference that is stored in the variable first
- size()
  - Return the value of the variable size
- isEmpty()
  - Return the result of the expression size==0

Singly-Linked List Operations

- after(p)
  - Return the next reference of p
- last()
  - Return the last position in the list
  - We only have a reference to the first position in the list
  - Each position only knows how to get to the position after it
  - We must follow the list from position to position until we reach the last position of the list
  - The last position is reached when there is no reference stored in the variable next of a position

Singly-Linked List Operations

- Pseudocode of last()

```
Algorithm last()
Output: the last position in the list

if isEmpty() then
    return null;
i ← first;
while (i.next !=null) do
    i ← i.next;
return i;
```

Singly-Linked List Operations

- Pseudocode of before(p)

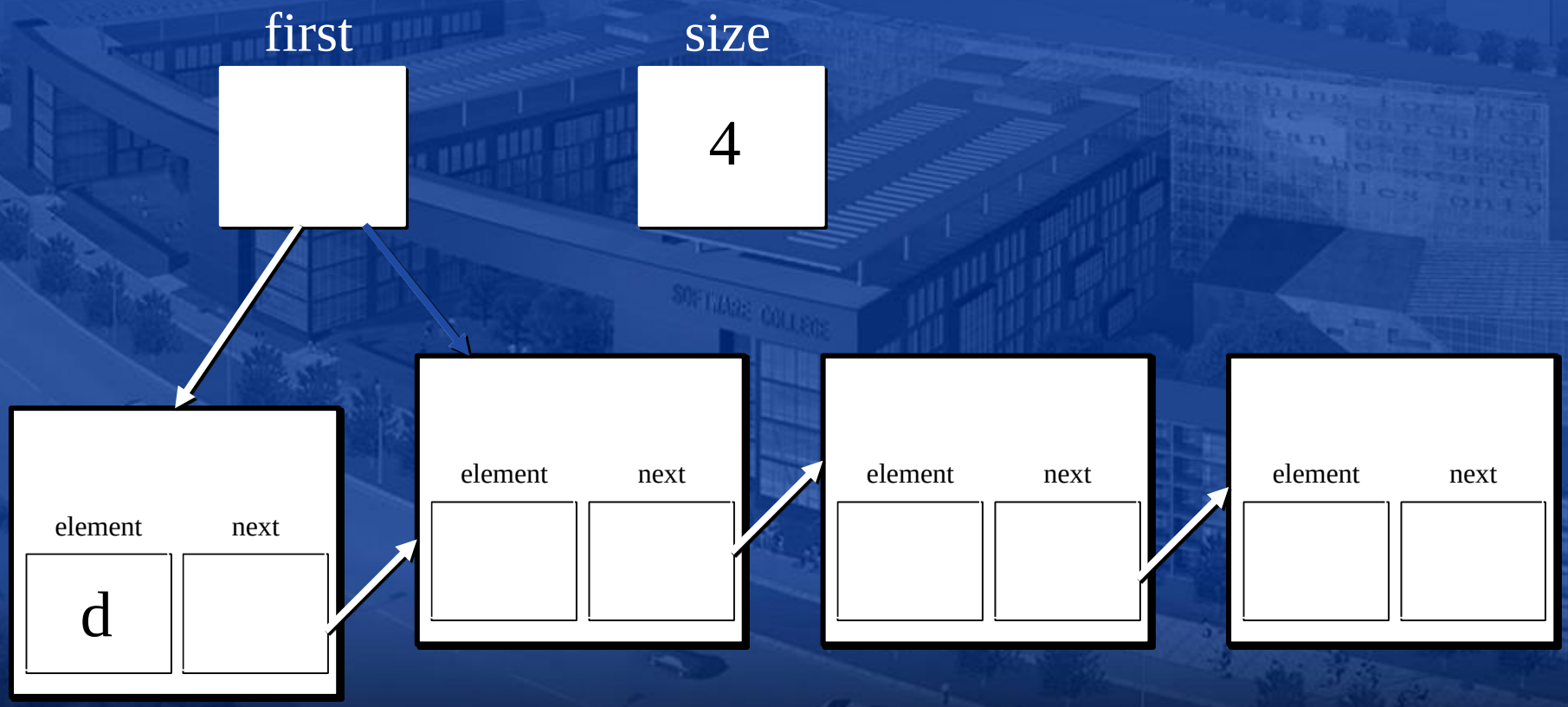
```
Algorithm before(p)
Input: the reference position
Output: the position before p

if isEmpty() OR p=first then
    return null;
i ← first;
while (i.next != null && p != i.next) do
    i ← i.next;
return i;
```

Singly-Linked List Operations

- insertFirst(d)
  - Add a Node with value d as the first position in the list
    - Construct a new Node object, called n, that contains value d
    - Change the next reference in n to make it point to the first node in the list
    - Change the first reference to make it point to n
    - Increment the size

Singly-Linked List Operations

- Example: insertFirst(d)



Singly-Linked List Operations

- Pseudocode of insertFirst(d)

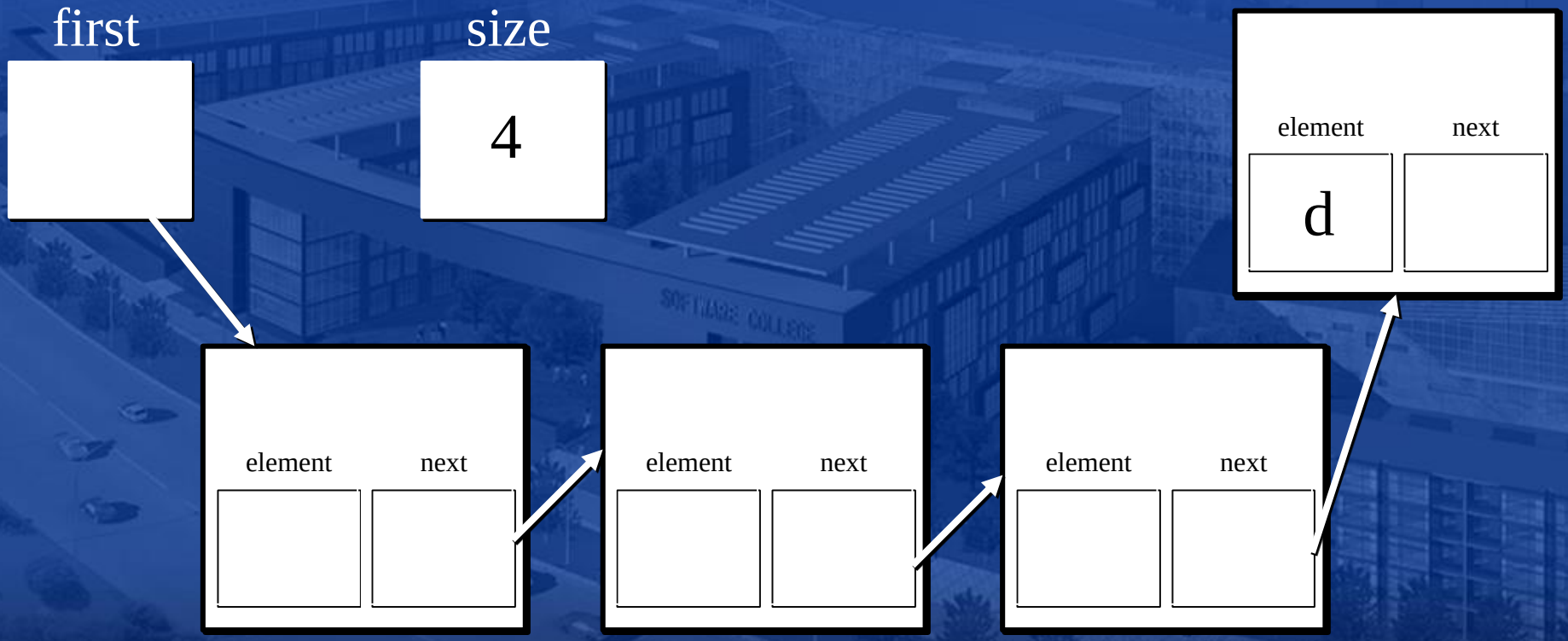
```
Algorithm insertFirst(d)
Input: the value d to be inserted into the list
Output: the position that contains d

Create node n containing d;
n.next ← first;
first ← n;
size ← size+1;
return n;
```

Singly-Linked List Operations

- insertLast(d)
  - Add a Node with value d as the last position in the list
    - Construct a new Node object, called n, that contains value d
    - Find the last position, called i, in the list using the last operation
    - Change the next reference of i to make it point to n
    - Increment the size
  - What would happen if the list is empty?

Singly-Linked List Operations

- Example: insertLast(d)



Singly-Linked List Operations

- Pseudocode of insertLast(d)

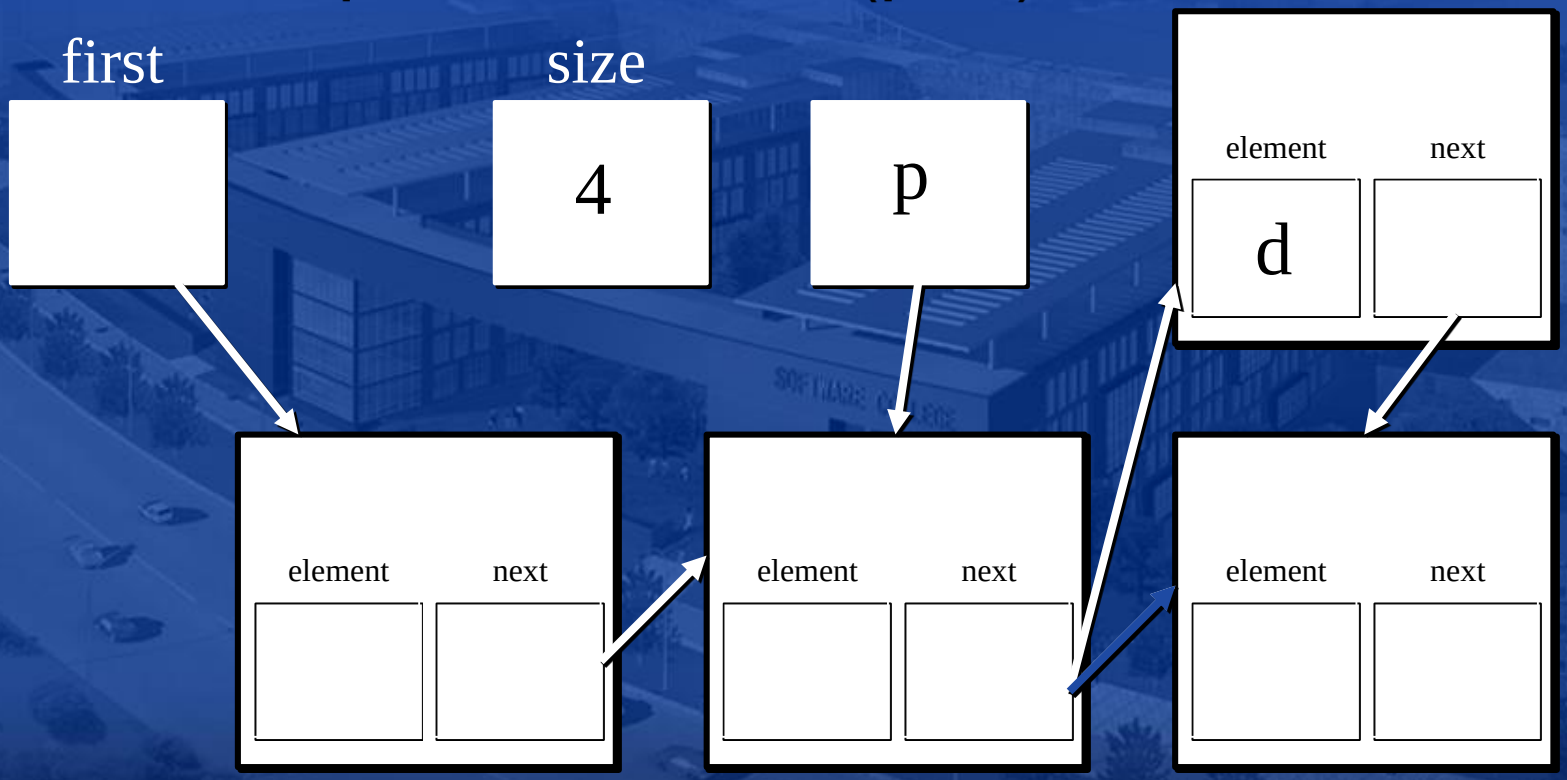
```
Algorithm insertLast(d)
Input : the value to be inserted
Output: the position that contains d

if isEmpty() then
    return insertFirst(d);
Create node n containing d;
i ← last();
i.next ← n;
size ← size + 1;
return n;
```

Singly-Linked List Operations

- insertAfter(p, d)
  - Insert a node containing d into the list at the position after p
    - Construct a new Node object, called n, that contains the value d
    - Change next of n to make it point to the next of p
    - Change the next of p to make it point to n
    - Increment the size

Singly-Linked List Operations

- Example: insertAfter(p, d)



Singly-Linked List Operations

- Pseudocode of insertAfter(p, d)

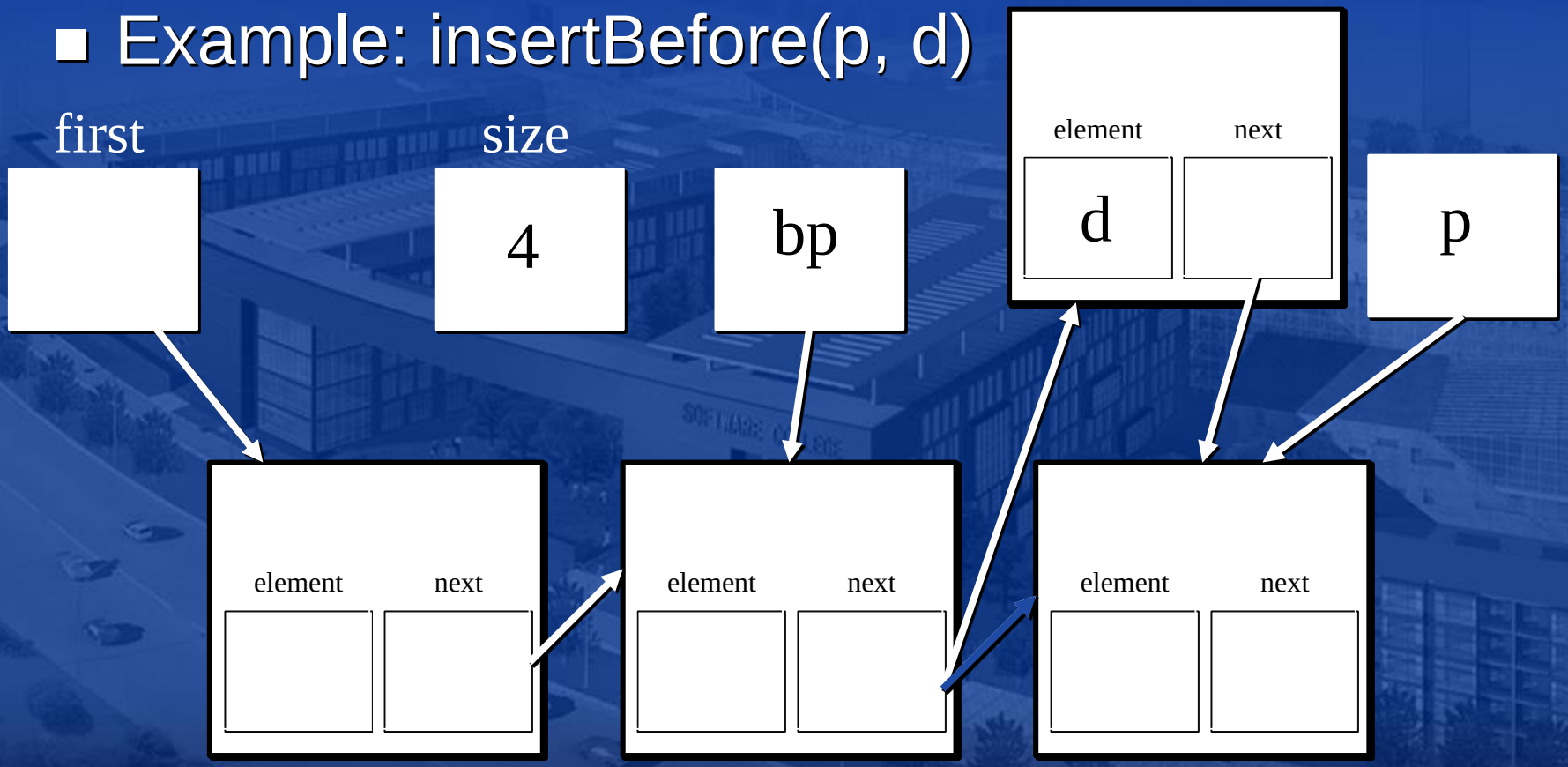
```
Algorithm insertAfter(p, d)
Input : the value to be inserted and the position it should
        be inserted after
Output: the position that contains d

Create node n containing d;
n.next ← p.next;
p.next ← n;
size ← size + 1;
return n;
```

Singly-Linked List Operations

- insertBefore(p, d)
  - Insert a node containing d into the list at the position before p
    - Construct a new Node object, called n, that contains the value d
    - Use the before operation to get to the position before p, call it bp
    - Change next of n to make it point to p
    - Change the next of bp to make it point to n
    - Increment the size
  - What happens if p is the first position?

Singly-Linked List Operations

- Example: insertBefore(p, d)



Singly-Linked List Operations

- Pseudocode of insertBefore(p, d)

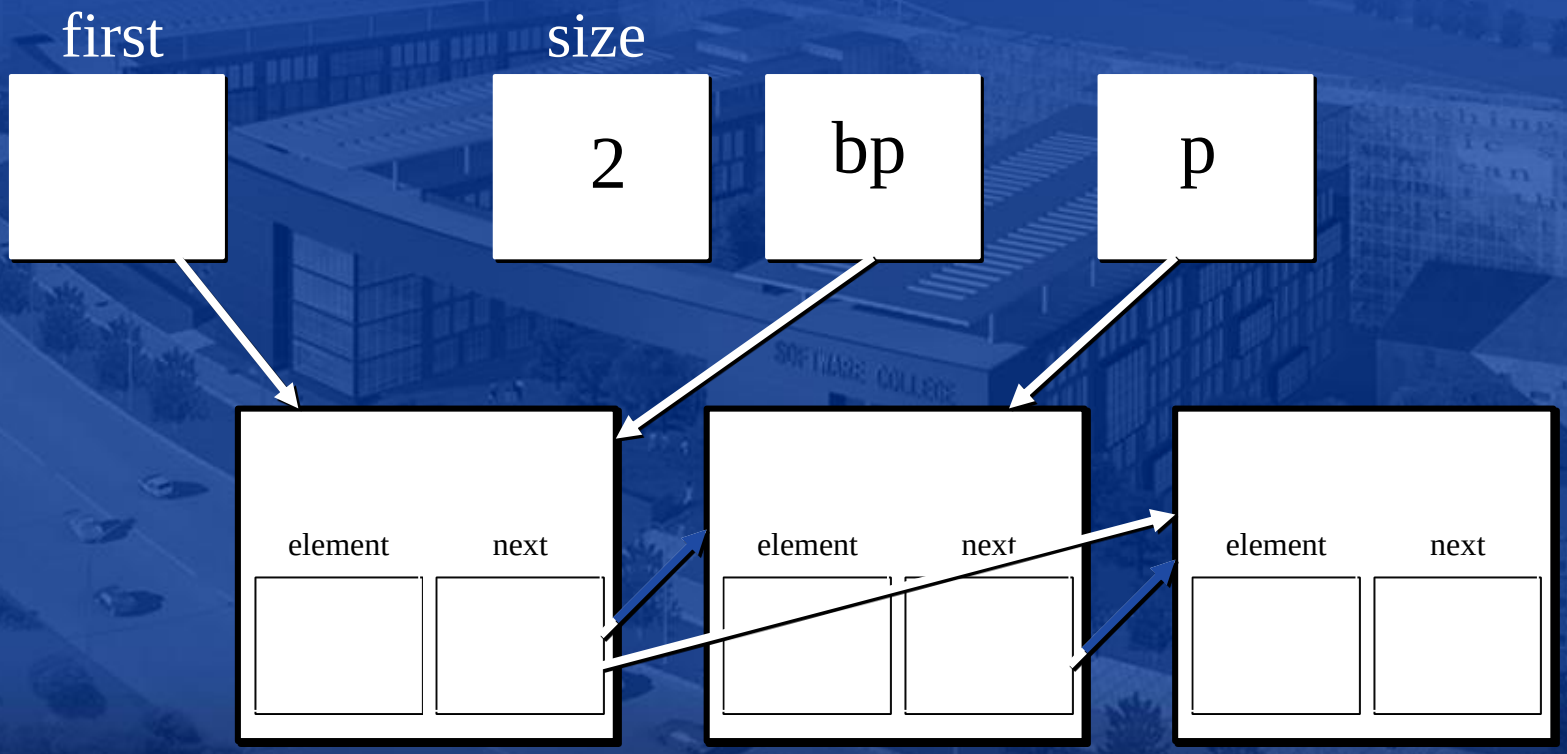
```
Algorithm insertBefore(p, d)
Input : the value to be inserted and the position it should be inserted before
Output: the position that contains d

if p=first then
    return insertFirst(d);
Create node n containing d;
bp ← before(p);
n.next ← p;
bp.next ← n;
size ← size + 1;
return n;
```

Singly-Linked List Operations

- remove(p)
  - remove node p from the list
    - Copy the data inside position p to variable d
    - Use the before operation to get the position before p, call it bp
    - Change next of bp to make it point to p.next
    - Decrement the size
    - return d
  - What happens if p is the first position?

Singly-Linked List Operations

- Example: remove(p)



Singly-Linked List Operations

- Pseudocode of remove(p)

```
Algorithm move(p)
Input : the position to be removed from the list
Output: the value that is removed

Copy data of p to variable d;
if p = first then
    first = p.next;
else
    bp ← before(p);
    bp.next ← p.next;
size ← size - 1;
return d;
```

Singly-Linked List Operations

- remove(d)
  - Remove the node that has value d from the list
    - Note: d is the value in the node, but not the node itself
  - We need to find the object(s) with value d, then remove it/them

Table of Contents

Algorithmic Complexity

Algorithm	Complexity
first()	O(1)
size()	O(1)
isEmpty()	O(1)
after(p)	O(1)
last()	O(n)
before(p)	O(n)
insertFirst(d)	O(1)
insertLast(d)	O(n)
insertAfter(p, d)	O(1)
insertBefore(p, d)	O(n)
remove(p)	O(n)
remove(d)	O(n)

Further Information and Review

- If you wish to review the materials covered in this lecture or to get further information, read the following sections in the text book Data Structures and Algorithms
  - 7 - List and Iterator ADTs

Q & A