Dictionary Implementations

Chapter 20

Data Structures and Abstractions with Java, 4e, Global Edition
Frank Carrano

Array-Based Implementations

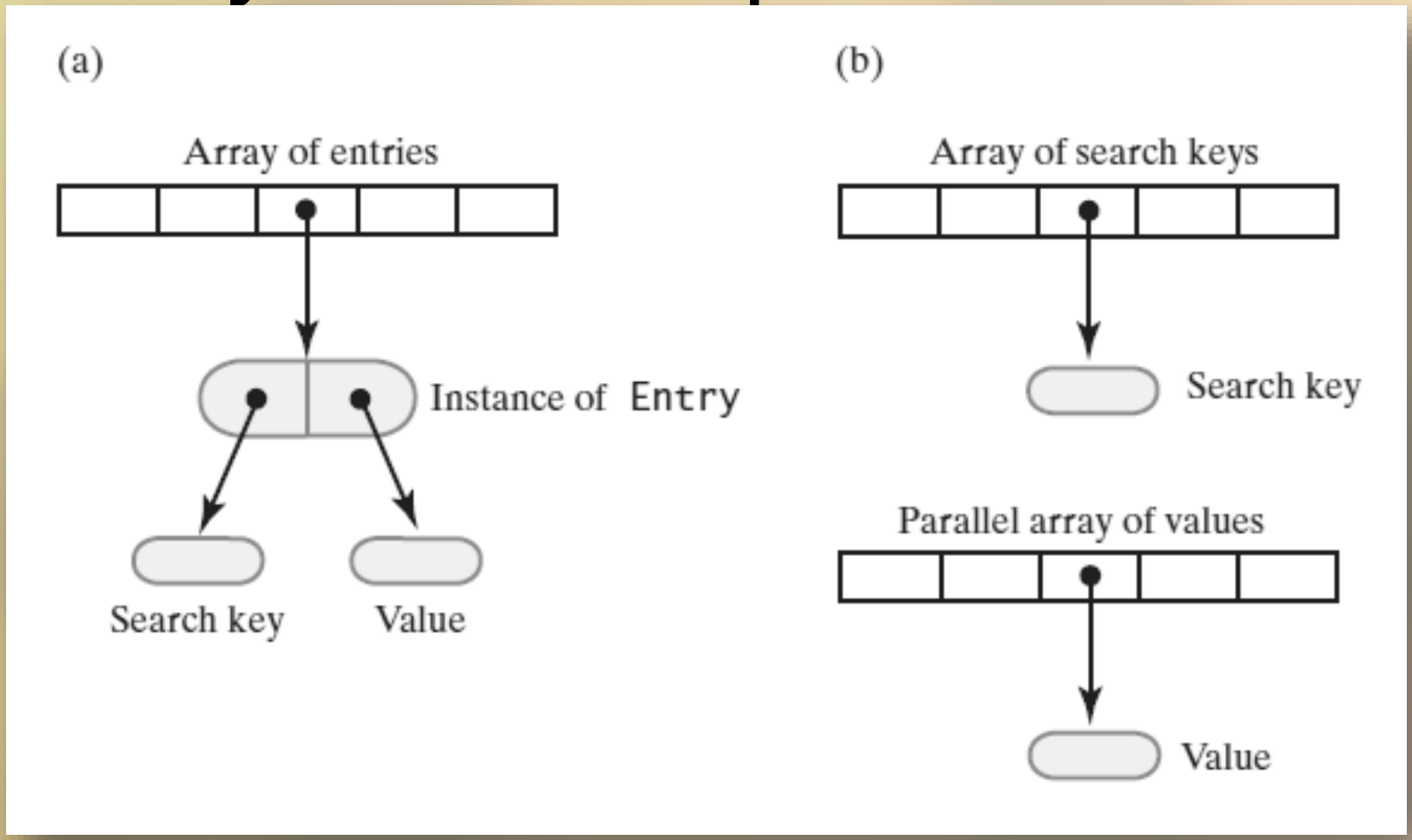

FIGURE 20-1 Two possible ways to use arrays to represent the entries in a dictionary: (a) an array of objects that encapsulate each search key and corresponding value; (b) parallel arrays of search keys and values

Array-Based Implementations

```
1  import java.util.Arrays;
2  import java.util.Iterator;
3  import java.util.NoSuchElementException;
4  /**
5      A class that implements a dictionary by using a resizable array.
6      The dictionary is unsorted and has distinct search keys.
7      @author Frank M. Carrano
8  */
9  public class ArrayDictionary<K, V> implements DictionaryInterface<K, V>
10 {
11    private Entry<K, V>[] dictionary; // Array of unsorted entries
12    private int numberOfEntries;
13    private boolean initialized = false;
14    private final static int DEFAULT_CAPACITY = 25;
15    private static final int MAX_CAPACITY = 10000;
16
17    public ArrayDictionary()
18    {
19       this(DEFAULT_CAPACITY);        // Call next constructor
20    } // end default constructor
21
22    public ArrayDictionary(int initialCapacity)
```

LISTING 20-1 The class **`ArrayDictionary`** and its private inner class **`Entry`**

Array-Based Implementations

```
22    public ArrayDictionary(int initialCapacity)
23    {
24       checkCapacity(initialCapacity);
25       // The cast is safe because the new array contains null entries
26       @SuppressWarnings("unchecked")
27       Entry<K, V>[] tempDictionary = (Entry<K, V>[])new Entry[initialCapacity];
28       dictionary = tempDictionary;
29       numberOfEntries = 0;
30       initialized = true;
31    } // end constructor
32
33    <Implementations of methods in DictionaryInterface. >
34    . . .
35
36    private class Entry<S, T>
37    {
38       private S key;
39       private T value;
40
41       private Entry(S searchKey, T dataValue)
42       {
43          key = searchKey;
44          value = dataValue;
45       } // end constructor
```

Array-Based Implementations

```
44            value = dataValue;
45        } // end constructor
46
47        private S getKey()
48        {
49            return key;
50        } // end getKey
51
52        private T getValue()
53        {
54            return value;
55        } // end getValue
56
57        private void setValue(T newValue)
58        {
59            value = newValue;
60        } // end setValue
61    } // end Entry
62 } // end ArrayDictionary
```

LISTING 20-1 The class **ArrayDictionary** and its private inner class **Entry**

Array-Based Implementations

```
Algorithm add(key, value)
// Adds a new key-value entry to the dictionary and returns null. If key already exists
// in the dictionary, returns the corresponding value and replaces it with value.
    result = null
    Search the array for an entry containing key
    if (an entry containing key is found in the array)
    {
        result = value currently associated with key
        Replace key's associated value with value
    }
    else // Insert new entry
    {
        if (array is full)
            Double size of array
        Insert a new entry containing key and value after the last entry in the array
        Increment the size of the dictionary
    }
    return result
```


Algorithm to add an entry.

Array-Based Implementations

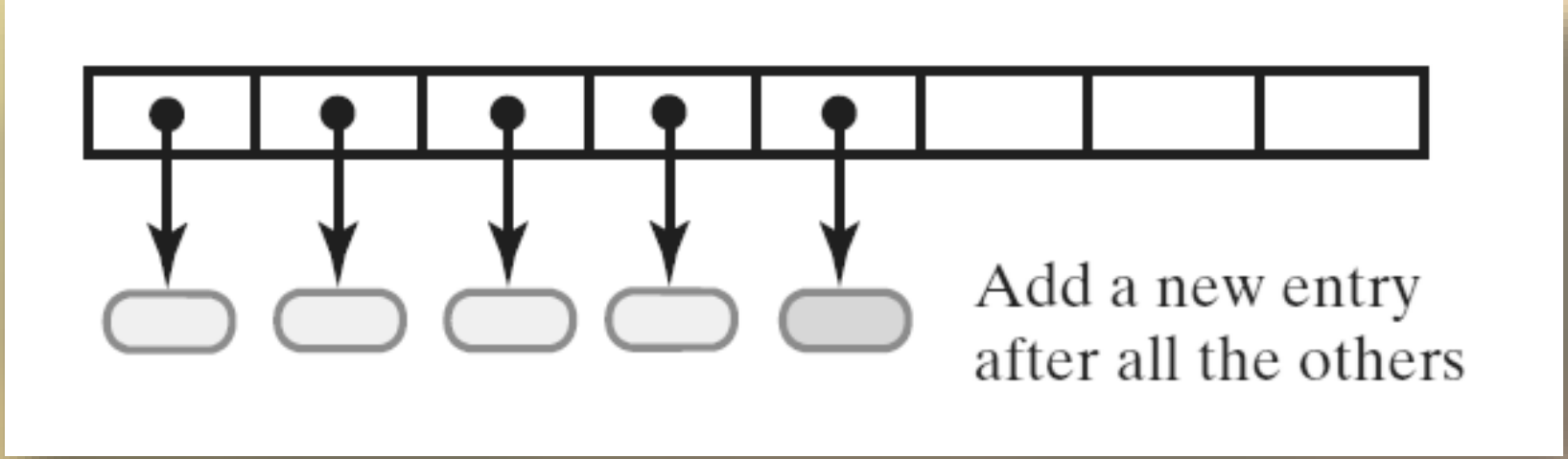

FIGURE 20-2 Adding a new entry to an unsorted array-based dictionary

Array-Based Implementations

```
public V add(K key, V value)
{
   checkInitialization();
   if ((key == null) || (value == null))
      throw new IllegalArgumentException();
   else
   {
      V result = null;

      int keyIndex = locateIndex(key);

      if (keyIndex < numberOfEntries)
      {
         // Key found; return and replace entry's value
         result = dictionary[keyIndex].getValue(); // Get old value
         dictionary[keyIndex].setValue(value);     // Replace value
      }
      else // Key not found; add new entry to dictionary
      {
         // Add at end of array
         dictionary[numberOfEntries] = new Entry<>(key, value);
         numberOfEntries++;
         ensureCapacity(); // Ensure enough room for next add
      } // end if
      return result;
   } // end if
} // end add
```

Array-Based Implementations

```
private int locateIndex(K key)
{
    int index = 0;
    while ( (index < numberOfEntries) &&
            !key.equals(dictionary[index].getKey()) )
        index++;

    return index;
} // end locateIndex
```

Method **`locate`** used by **`add`**

Array-Based Implementations

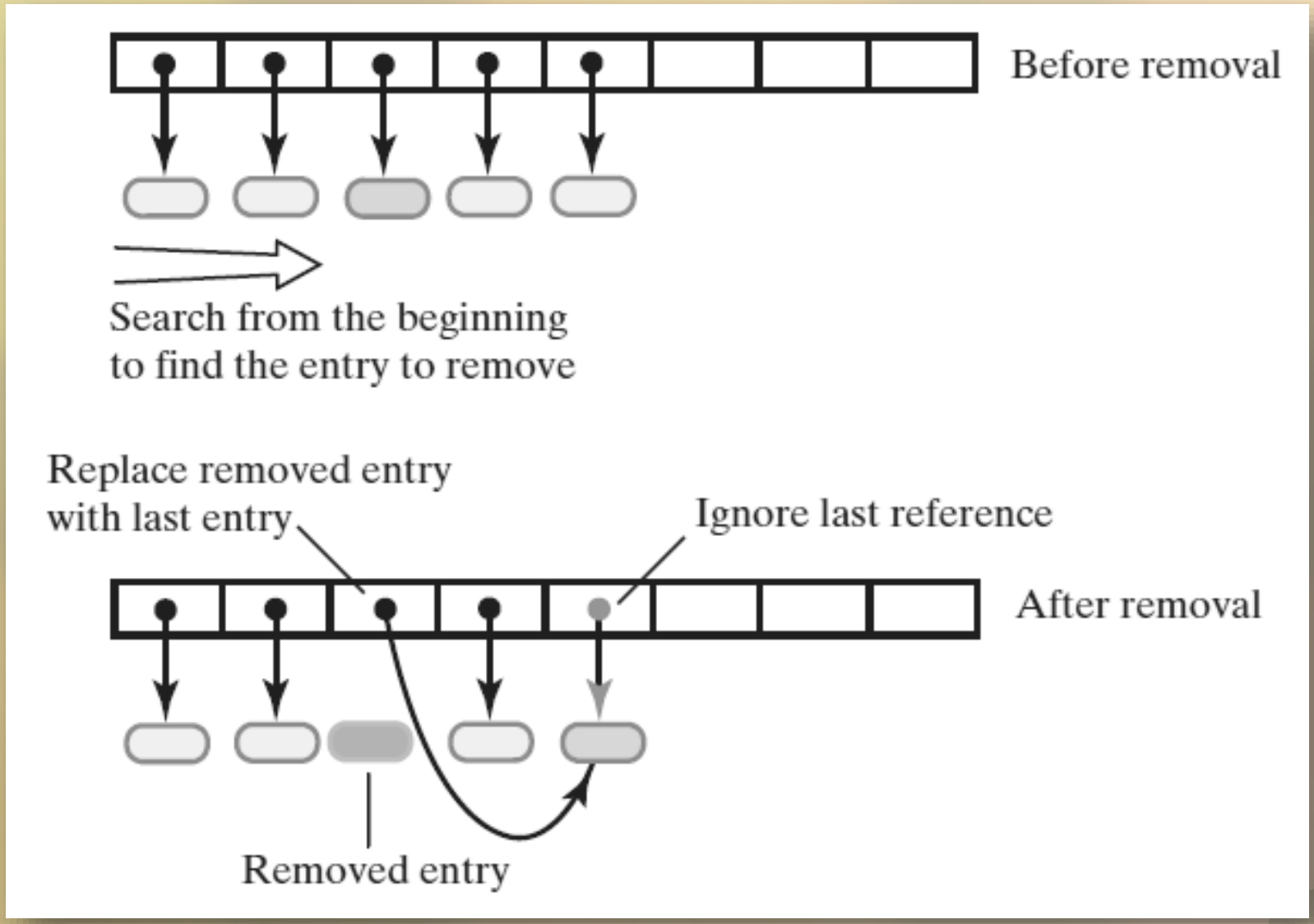

FIGURE 20-3 Removing an entry from an unsorted array-based dictionary

Array-Based Implementations

```
Algorithm remove(key)
// Removes an entry from the dictionary, given its search key, and returns its value.
// If no such entry exists in the dictionary, returns null.

result = null
Search the array for an entry containing key

if (an entry containing key is found in the array)
{
   result = value currently associated with key
   Replace the entry with the last entry in the array
   Decrement the size of the dictionary
}
// Else result is null

return result
```


Algorithm to describe the remove operation.

Array-Based Implementations

```
public V remove(K key)
{
   checkInitialization();
   V result = null;
   int keyIndex = locateIndex(key);
   if (keyIndex < numberOfEntries)
   {
      // Key found; remove entry and return its value
      result = dictionary[keyIndex].getValue();
      dictionary[keyIndex] = dictionary[numberOfEntries - 1];
      dictionary[numberOfEntries - 1] = null;
      numberOfEntries--;
   } // end if
   // Else result is null
   return result;
} // end remove
```


Implementation of the method `remove`.

Sorted Array-Based Dictionary

```
import java.util.Arrays;
import java.util.Iterator;
import java.util.NoSuchElementException;
/**
    A class that implements a dictionary by using a resizable sorted array.
    @author Frank M. Carrano
*/
public class SortedArrayDictionary<K extends Comparable<? super K>, V>
             implements DictionaryInterface<K, V>

   < Data fields as shown in Listing 20-1 of Segment 20.2. >
   . . .
   < Constructors analogous to those in Listing 20-1 >
```

LISTING 20-2 An outline of the class

SortedArrayDictionary

Sorted Array-Based Dictionary

```
14    . . .
15
16    public V add(K key, V value)
17    {
18       . . . < See Segment 20.11. >
19    } // end add
20
21    < Implementations of other methods in DictionaryInterface. >
22    . . .
23
24    < The private class Entry, as shown in Listing 20-1. >
25 } // end SortedArrayDictionary
```

LISTING 20-2 An outline of the class **SortedArrayDictionary**

Sorted Array-Based Dictionary

```
Algorithm add(key, value)
// Adds a new key-value entry to the dictionary and returns null. If key already exists
// in the dictionary, returns the corresponding value and replaces it with value.

result = null
Search the array until you either find an entry containing key or locate the point where it
    should be
if (an entry containing key is found in the array)
{
    result = value currently associated with key
    Replace key's associated value with value
}
else // Insert new entry
{

    Make room in the array for a new entry at the index determined by the previous search
    Insert a new entry containing key and value into the vacated location of the array
    Increment the size of the dictionary
    if (array is full)
    Double size of array
}
return result
```


Algorithm for adding an entry

Sorted Array-Based Dictionary

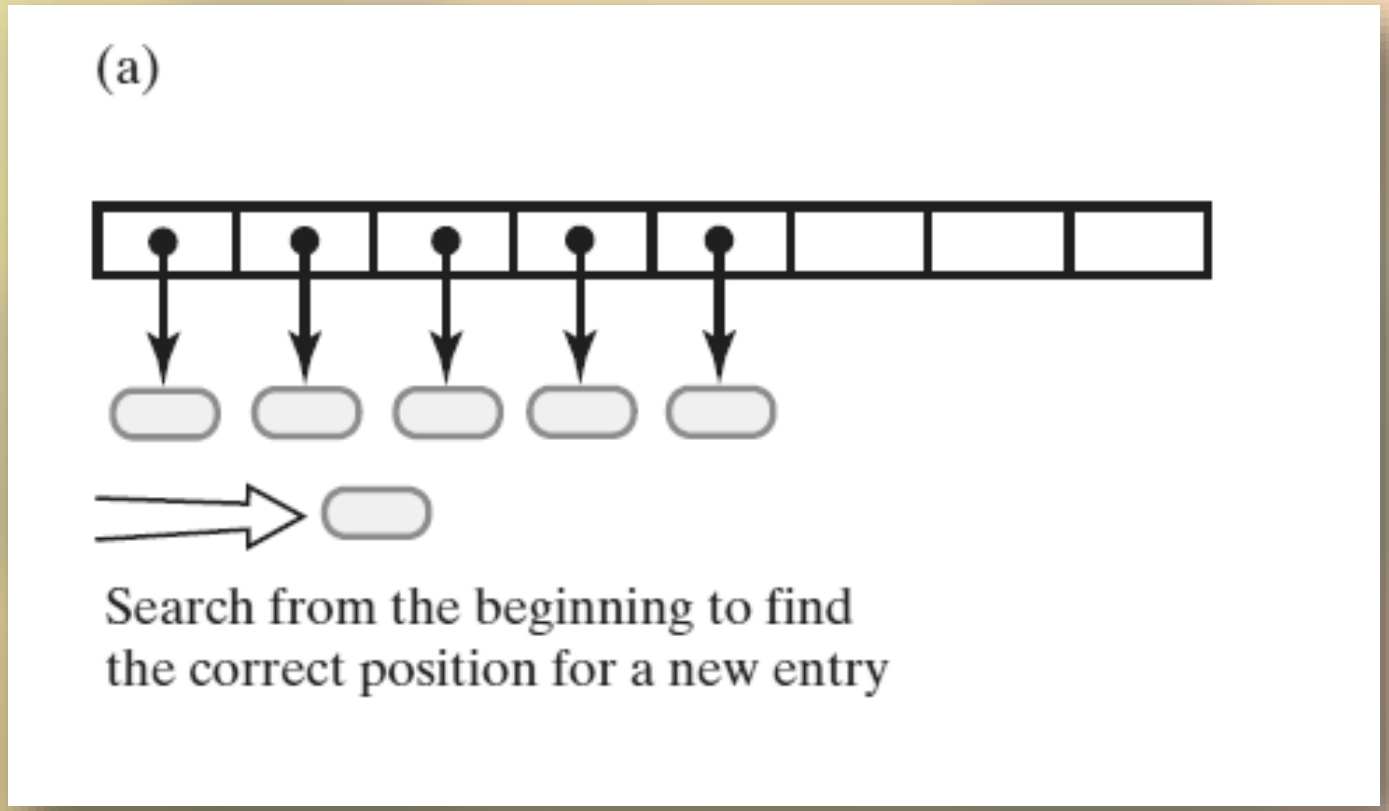

FIGURE 20-4 Adding an entry to a sorted array-based dictionary: (a) search

Sorted Array-Based Dictionary

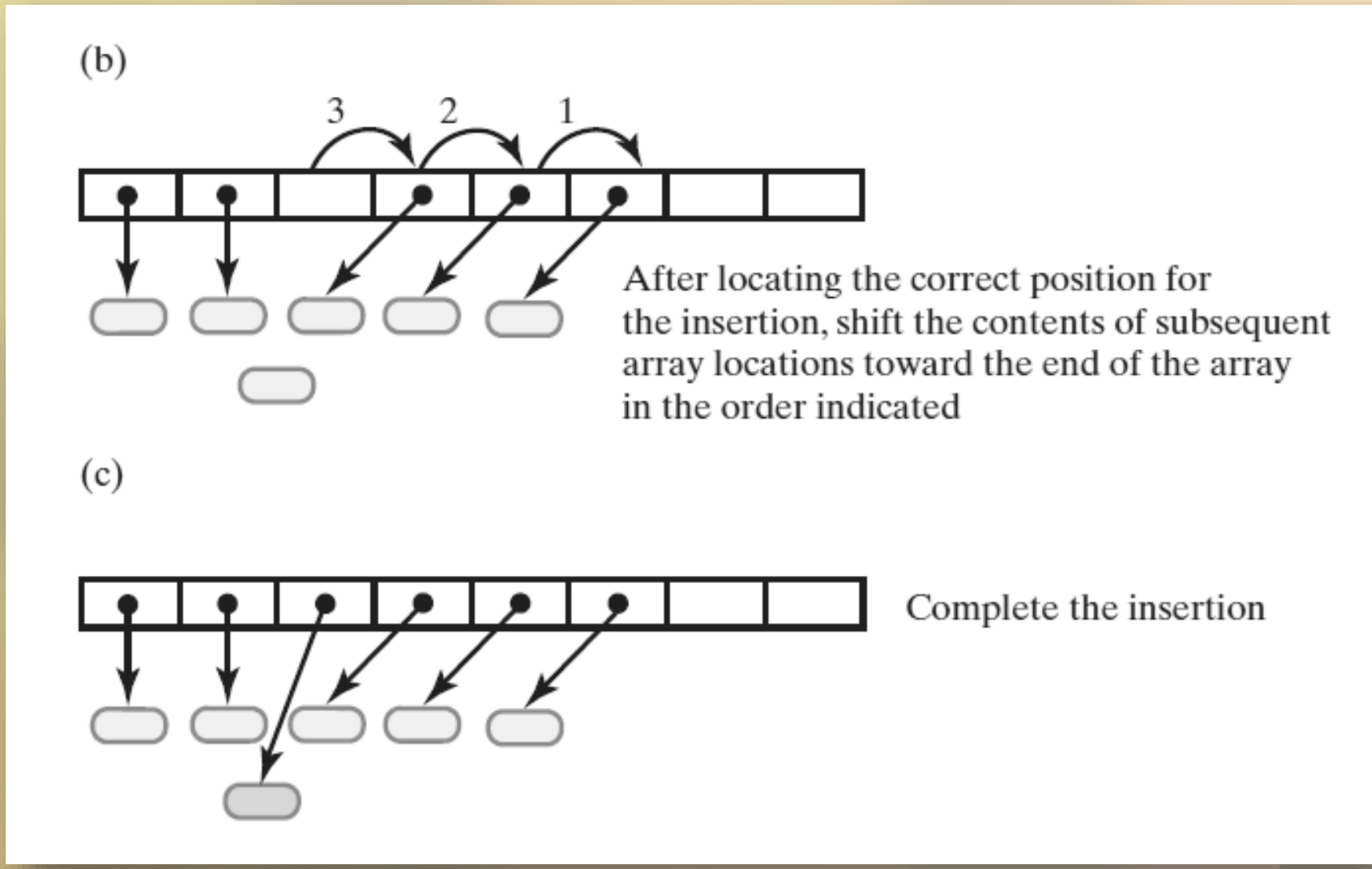

FIGURE 20-4 Adding an entry to a sorted array-based dictionary: (b) make room; (c) insert

Sorted Array-Based Dictionary

```
public V add(K key, V value)
{
   checkInitialization();
   if ((key == null) || (value == null))
      throw new IllegalArgumentException();
   else
   {
      V result = null;
      int keyIndex = locateIndex(key);
      if ( (keyIndex < numberOfEntries) &&
           key.equals(dictionary[keyIndex].getKey()) )
      {
         // Key found; return and replace entry's value
         result = dictionary[keyIndex].getValue(); // Get old value
         dictionary[keyIndex].setValue(value);     // Replace value
```


Implementation of method **add**.

Sorted Array-Based Dictionary

```
            result = dictionary[keyIndex].getValue(); // Get old value
            dictionary[keyIndex].setValue(value);     // Replace value
         }
         else // Key not found; add new entry to dictionary
         {
            makeRoom(keyIndex);
            dictionary[keyIndex] = new Entry<>(key, value);
            numberOfEntries++;
            ensureCapacity(); // Ensure enough room for next add
         } // end if
         return result;
      } // end if
   } // end add
```


Implementation of method **add**.

Sorted Array-Based Dictionary

```
private int locateIndex(K key)
{
   // Search until you either find an entry containing key or
   // pass the point where it should be
   int index = 0;
   while ( (index < numberOfEntries) &&
           key.compareTo(dictionary[index].getKey()) > 0 )
     index++;

   return index;
} // end locateIndex
```

Definition of the private method **`locateIndex`**

Sorted Array-Based Dictionary

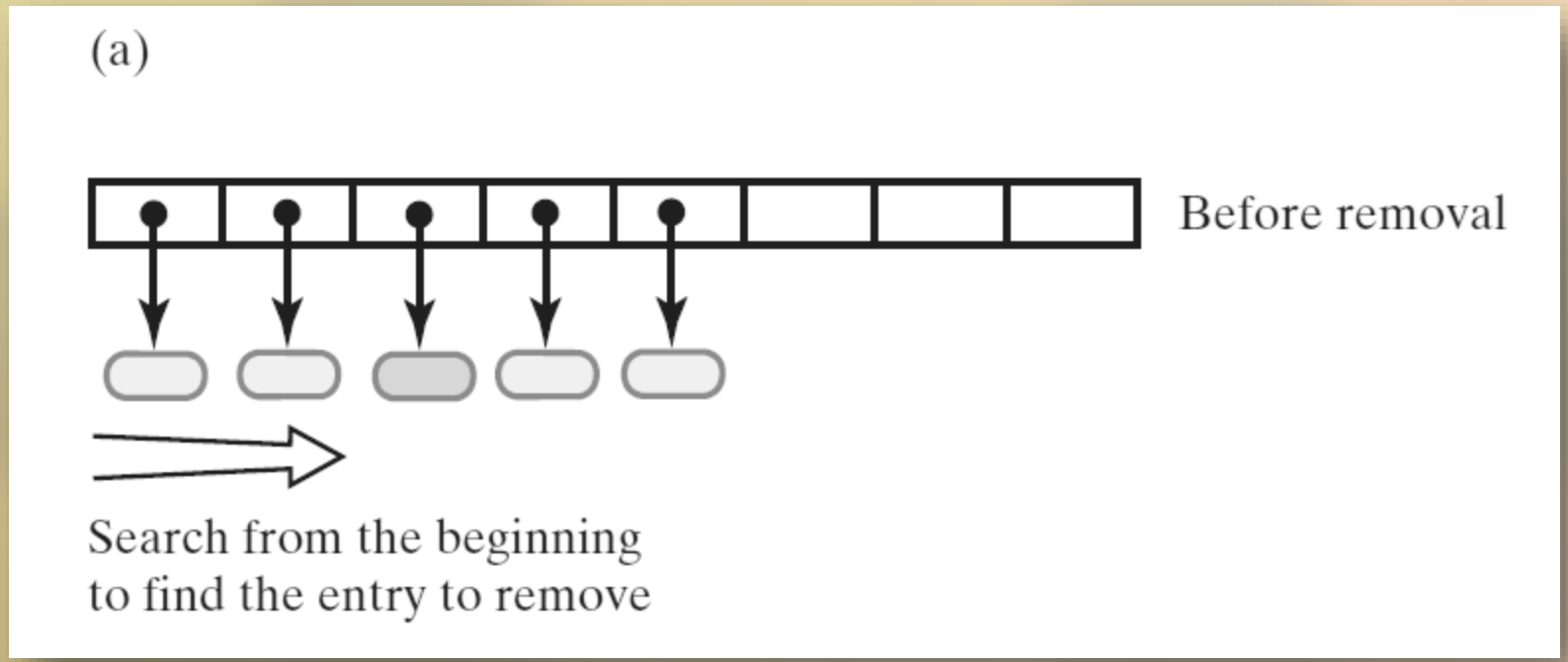

FIGURE 20-5 Removing an entry from a sorted array-based dictionary: (a) search;

Sorted Array-Based Dictionary

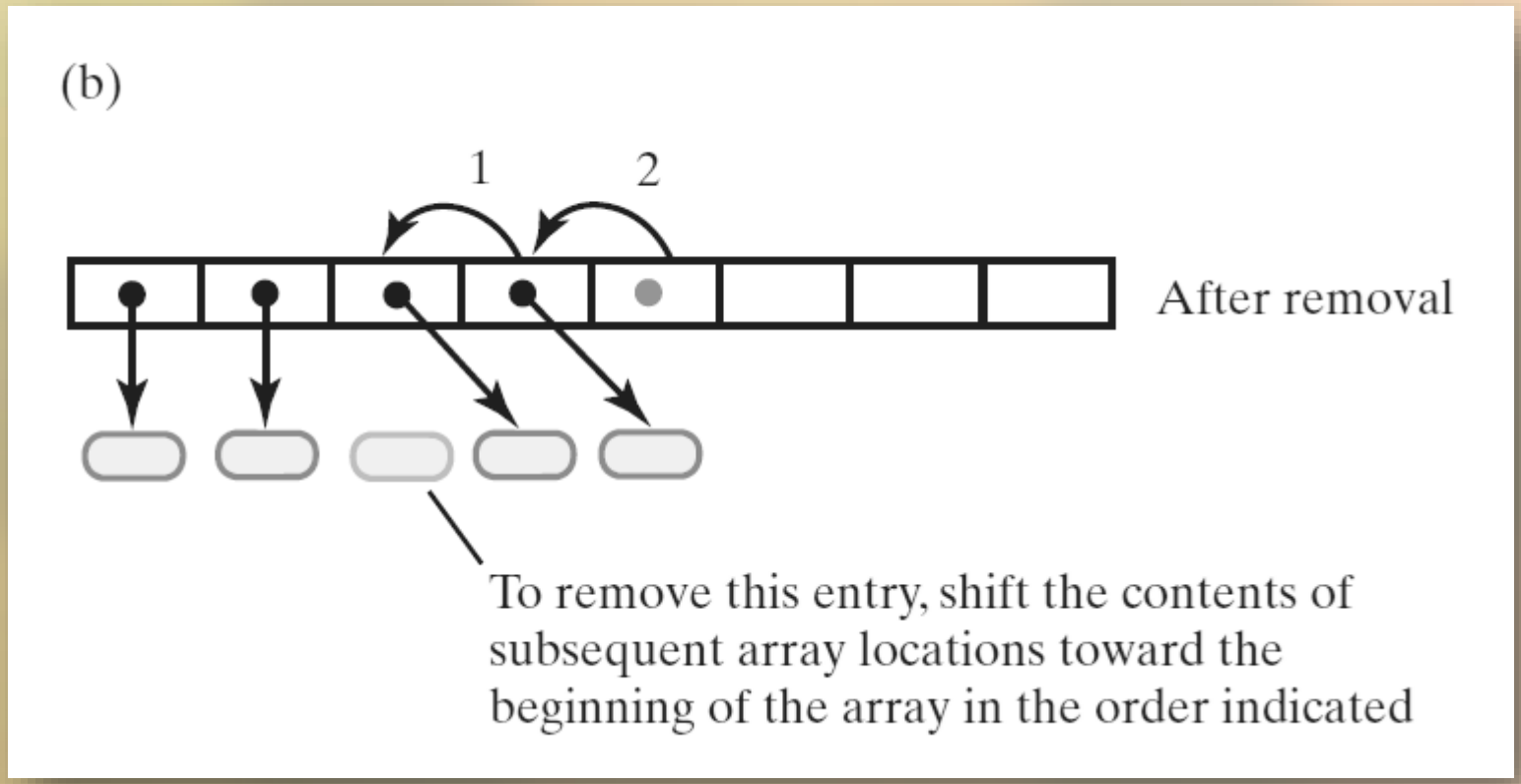

FIGURE 20-5 Removing an entry from a sorted array-based dictionary: (b) shift entries

Sorted Array-Based Dictionary

```
Algorithm remove(key)
// Removes an entry from the dictionary, given its search key, and returns its value.
// If no such entry exists in the dictionary, returns null.

result = null
Search the array for an entry containing key

if (an entry containing key is found in the array)
{
   result = value currently associated with key
   Shift any entries that are after the located one to the next lower position in the array
   Decrement the size of the dictionary
}
return result
```

Algorithm that describes the remove operation

Linked Implementations

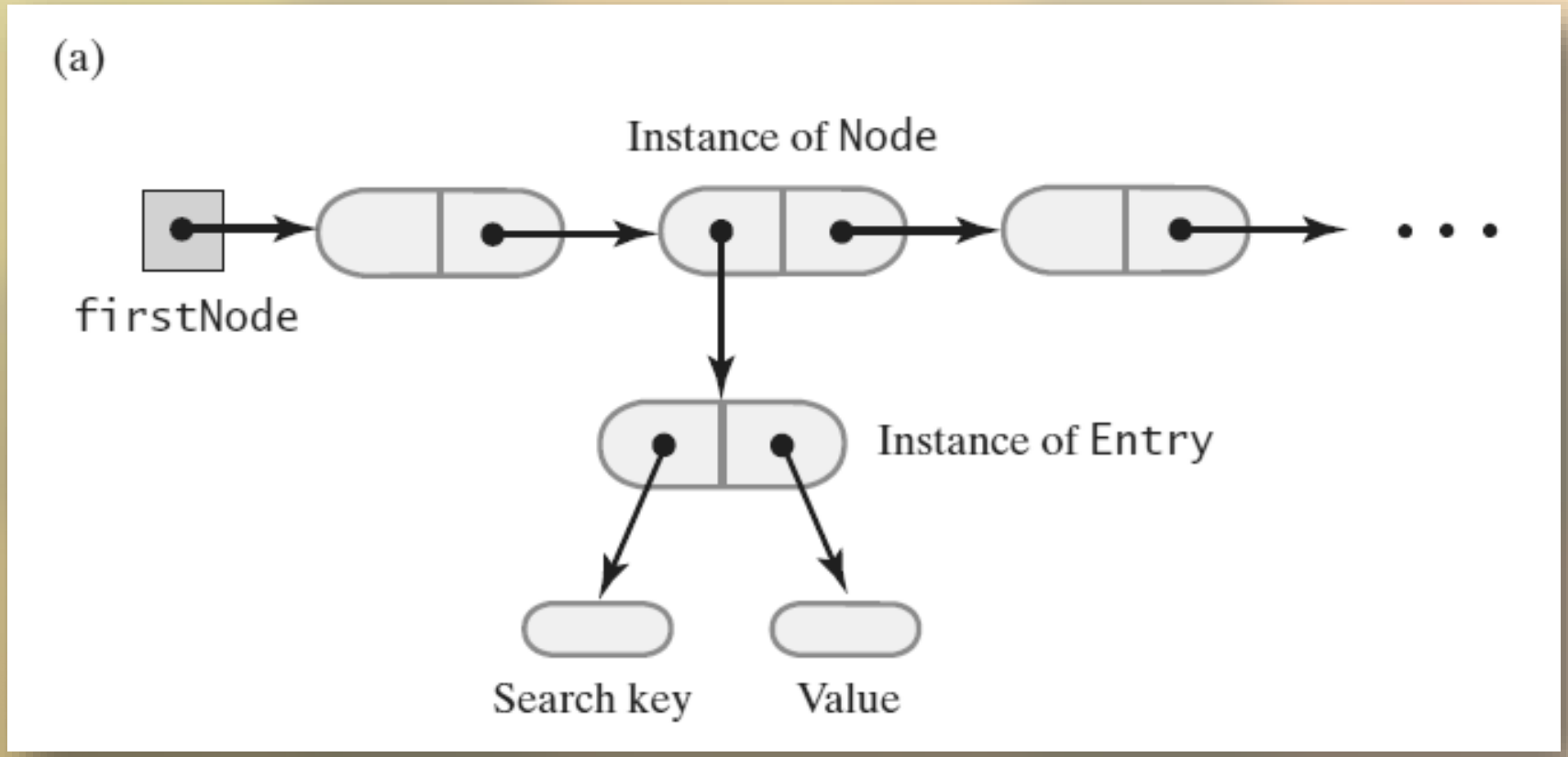

FIGURE 20-6 Three possible ways to use linked nodes to represent the entries in a dictionary:
(a) a chain of nodes that each reference an entry object

Linked Implementations

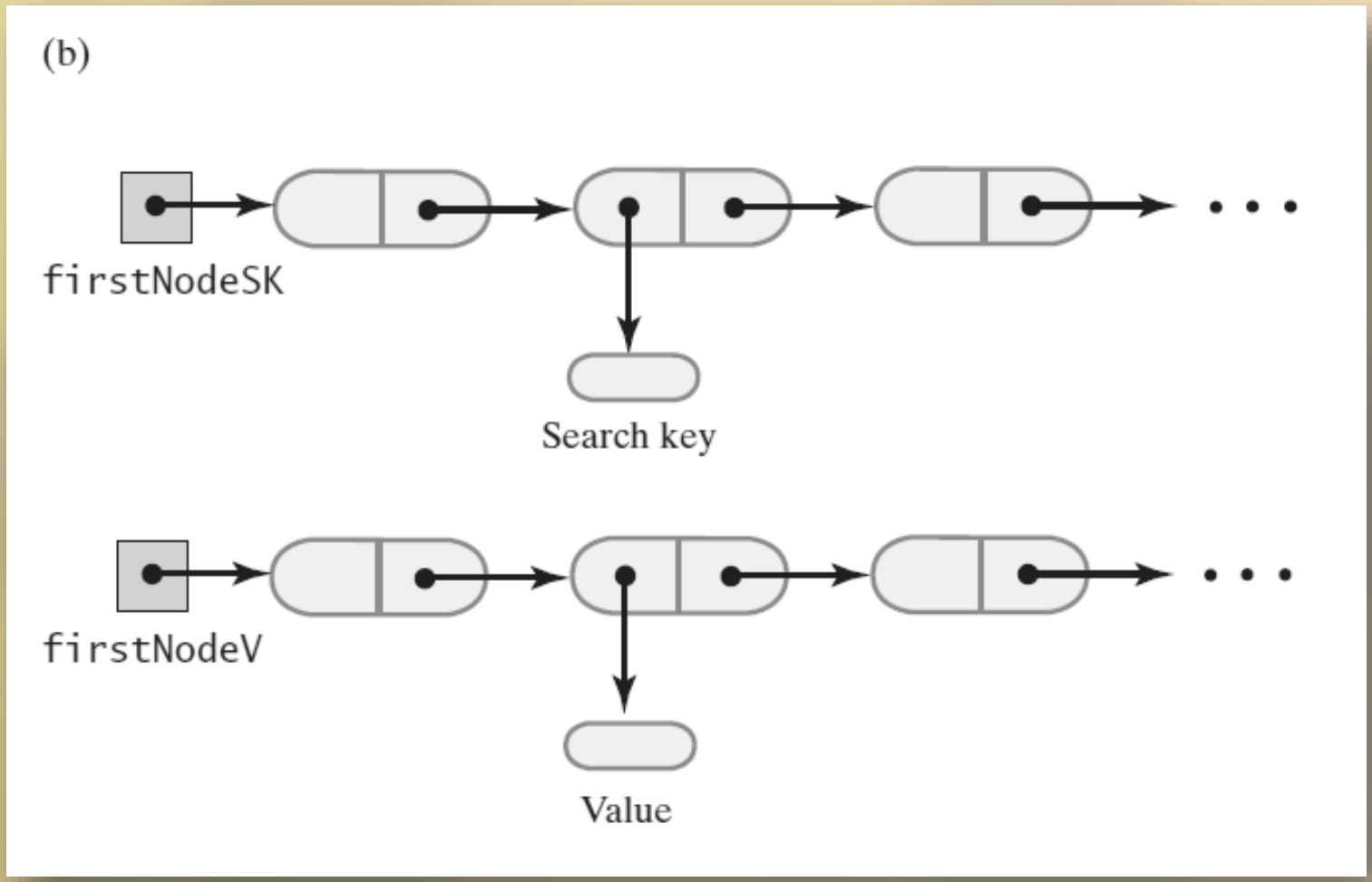

FIGURE 20-6 Three possible ways to use linked nodes to represent the entries in a dictionary:
(b) parallel chains of search keys and values;

Linked Implementations

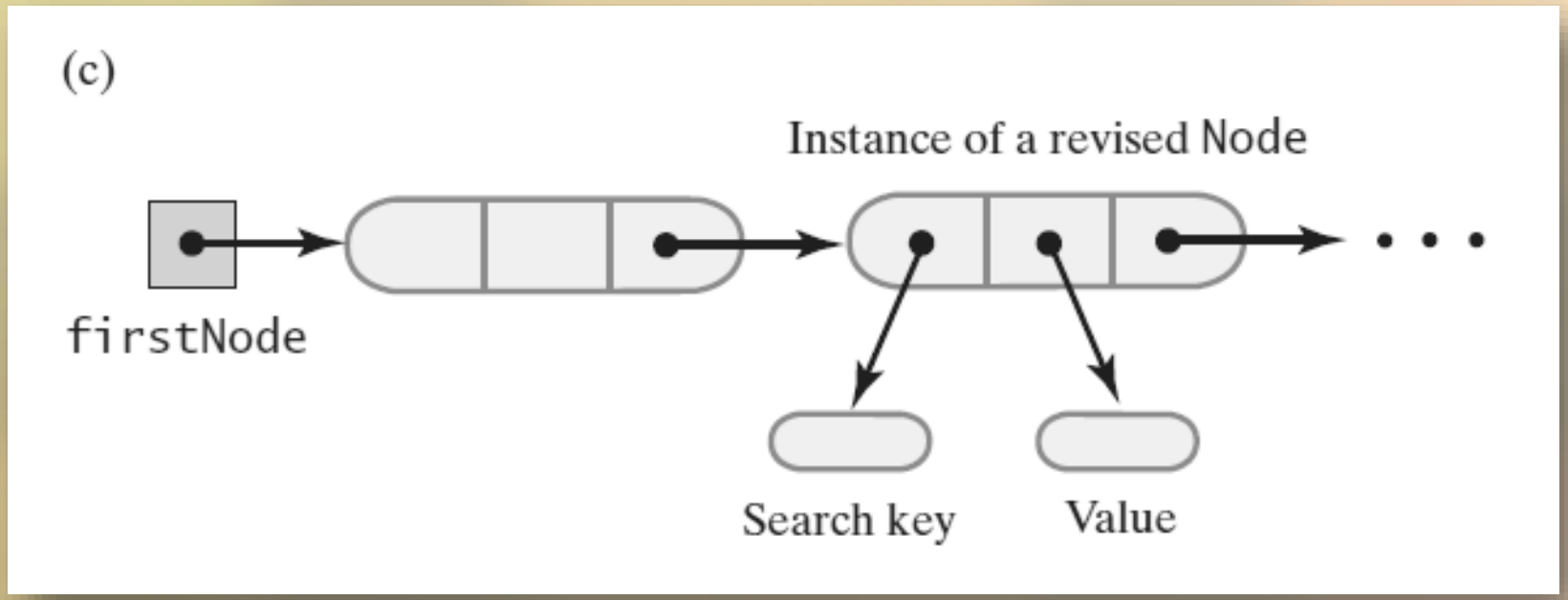

FIGURE 20-6 Three possible ways to use linked nodes to represent the entries in a dictionary: (c) a chain of nodes that each reference a search key and a value

An Unsorted Linked Dictionary

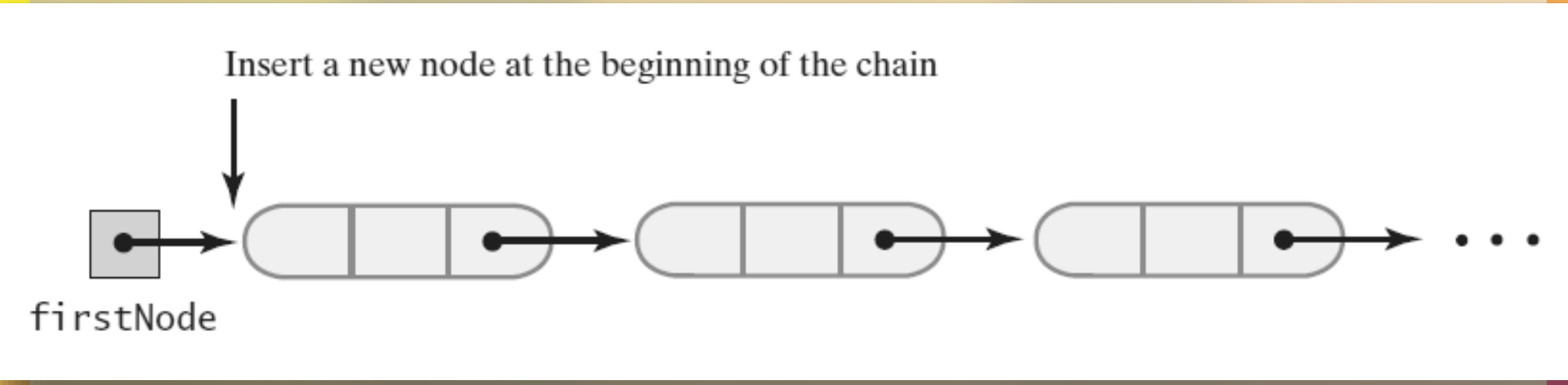

FIGURE 20-7 Adding to an unsorted linked dictionary

Sorted Linked Dictionary

```
Algorithm add(key, value)
// Adds a new key-value entry to the dictionary and returns null. If key already exists
// in the dictionary, returns the corresponding value and replaces that value with value.

result = null
Search the chain until either you find a node containing key or you pass the point where
    it should be
if (a node containing key is found in the chain)
{
    result = value currently associated with key
    Replace key's associated value with value
}
else
{
    Allocate a new node containing key and value
    if (the chain is empty or the new entry belongs at the beginning of the chain)
    Add the new node to the beginning of the chain
    else
        Insert the new node before the last node that was examined during the search
    Increment the size of the dictionary
}
return result
```


Algorithm for adding new entry to sorted linked dictionary

Sorted Linked Dictionary

```
1  import java.util.Iterator;
2  import java.util.NoSuchElementException;
3  /**
4      A class that implements a dictionary by using a sorted linked chain.
5      The dictionary has distinct search keys.
6      @author Frank M. Carrano
7  */
8  public class SortedLinkedDictionary<K extends Comparable<? super K>, V>
9                implements DictionaryInterface<K, V>
10 {
11    private Node firstNode;  // Reference to first node of chain
12    private int  numberOfEntries;
13
14    public SortedLinkedDictionary()
15    {
16       initializeDataFields();
17    } // end default constructor
18
19    public V add(K key, V value)
20    {
21       V result = null;
22       // Search chain until you either find a node containing key
```


LISTING 20-5 The class `SortedLinkedDictionary`

Sorted Linked Dictionary

```
19    public V add(K key, V value)
20    {
21       V result = null;
22       // Search chain until you either find a node containing key
23       // or locate where it should be
24       Node currentNode = firstNode;
25       Node nodeBefore = null;
26       while ((currentNode != null) && key.compareTo(currentNode.getKey()) > 0)
27       {
28          nodeBefore = currentNode;
29          currentNode = currentNode.getNextNode();
30       } // end while
31
32       if ( (currentNode != null) && key.equals(currentNode.getKey()) )
33       {
34          result = currentNode.getValue();     // Get old value
35          currentNode.setValue(value);         // Replace value
36       }
37       else
38       {
```


LISTING 20-5 The class `SortedLinkedDictionary`

Sorted Linked Dictionary

```
else
{
   Node newNode = new Node(key, value); // Create new node
   if (nodeBefore == null)
   {  // Add at beginning (includes empty chain)
      newNode.setNextNode(firstNode);
      firstNode = newNode;
   }
   else // Add elsewhere in non-empty chain
   {
      newNode.setNextNode(currentNode); // currentNode is after new node

      nodeBefore.setNextNode(newNode);  // nodeBefore is before new node

   } // end if
   numberOfEntries++; // Increase length for both cases
} // end if
```


LISTING 20-5 The class **SortedLinkedDictionary**

Sorted Linked Dictionary

```
55          return result;
56       } // end add
57
58       < Implementations of the other methods in DictionaryInterface. >
59       . . .
60
61       < Private classes KeyIterator and ValueIterator (see Segment 20.20). >
62       . . .
63
64       < The private class Node. >
65       . . .
66
67    } // end SortedLinkedDictionary
```

LISTING 20-5 The class **SortedLinkedDictionary**

Sorted Linked Dictionary

```
private class KeyIterator implements Iterator<K>
{
   private Node nextNode; // Node containing next entry in iteration

   private KeyIterator()
   {
      nextNode = firstNode;
   } // end default constructor

   public boolean hasNext()
   {
      return nextNode != null;
   } // end hasNext

   public K next()
   {
```

LISTING 20-6 **SortedLinkedDictionary**'s private inner class **KeyIterator**

Sorted Linked Dictionary

```
15    public K next()
16    {
17       K result;
18       if (hasNext())
19       {
20          result = nextNode.getKey();
21          nextNode = nextNode.getNextNode();
22       }
23       else
24          throw new NoSuchElementException();
25       return result;
26    } // end next
27
28    public void remove()
29    {
30       throw new UnsupportedOperationException();
31    } // end remove
32 } // end KeyIterator
```

LISTING 20-6 **SortedLinkedDictionary**'s private inner class **KeyIterator**

Sorted Linked Dictionary

- Efficiencies of the dictionary operations for a sorted linked implementation
  - Addition O(*n*)
  - Removal O(*n*)
  - Retrieval O(*n*)
  - Traversal O(*n*)

Implementation of the ADT Dictionary

```
import java.util.Iterator;
public interface DictionaryInterface<K, V>
{
   public V add(K key, V value);
   public V remove(K key);
   public V getValue(K key);
   public boolean contains(K key);
   public Iterator<K> getKeyIterator();
   public Iterator<V> getValueIterator();
   public boolean isEmpty();
   public int getSize();
   public void clear();
} // end DictionaryInterface
```


Interface for a dictionary

BST Implementation of ADT Dictionary

```
1  import TreePackage.SearchTreeInterface;
2  import TreePackage.BinarySearchTree;
3  import java.util.Iterator;
4  public class BstDictionary<K extends Comparable<? super K>, V>
5          implements DictionaryInterface<K, V>
6  {
7     private SearchTreeInterface<Entry<K, V>> bst;
8
9     public BstDictionary()
10    {
11       bst = new BinarySearchTree<>();
12    } // end default constructor
13
14    < Methods that implement dictionary operations are here. >
15    . . .
16
17    private class Entry<S extends Comparable<? super S>, T>
18            implements Comparable<Entry<S, T>>
19    {
20       private S key;
```


ADT dictionary that uses a binary search tree

BST Implementation of ADT Dictionary

```
19    {
20       private S key;
21       private T value;
22
23       private Entry(S searchKey, T dataValue)
24       {
25          key = searchKey;
26          value = dataValue;
27       } // end constructor
28
29       public int compareTo(Entry<S, T> other)
30       {
31          return key.compareTo(other.key);
32       } // end compareTo
33
34    < The class Entry also defines the methods equals, toString, getKey, getValue,
35      and setValue; no setKey method is provided. >
36    . . .
37    } // end Entry
38 } // end BstDictionary
```

LISTING 25-3 An outline of an implementation of the ADT dictionary that uses a binary search tree

BST Implementation of the ADT Dictionary

```
public V add(K key, V value)
{
   Entry<K, V> newEntry = new Entry<>(key, value);
   Entry<K, V> returnedEntry = bst.add(newEntry);

   V result = null;
   if (returnedEntry != null)
      result = returnedEntry.getValue();

   return result;
} // end add
```

Dictionary's **add** method

BST Implementation of the ADT Dictionary

```
public V remove(K key)
{
    Entry<K, V> findEntry = new Entry<>(key, null);
    Entry<K, V> returnedEntry = bst.remove(findEntry);
```

Beginning of Dictionary's **`remove`** method

Implementation of the ADT Dictionary

```
public Iterator<K> getKeyIterator()
{
   return new KeyIterator();
} // end getKeyIterator
```

BST Implementation of the ADT Dictionary

```
private class KeyIterator implements Iterator<K>
{
   Iterator<Entry<K, V>> localIterator;

   public KeyIterator()
   {
      localIterator = bst.getInorderIterator();
   } // end default constructor

  public boolean hasNext()
  {
     return localIterator.hasNext();
  } // end hasNext

  public K next()
```

BST Implementation of the ADT Dictionary

```
    } // end hasNext

    public K next()
    {
       Entry<K, V> nextEntry = localIterator.next();
       return nextEntry.getKey();
    } // end next

    public void remove()
    {
       throw new UnsupportedOperationException();
    } // end remove
} // end KeyIterator
```

Iterators

End

Chapter 20 & Chapter 25