Northeastern University

Lecture 26: An Introduction to Sorting - 2

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Insertion Sort

Insertion Sort

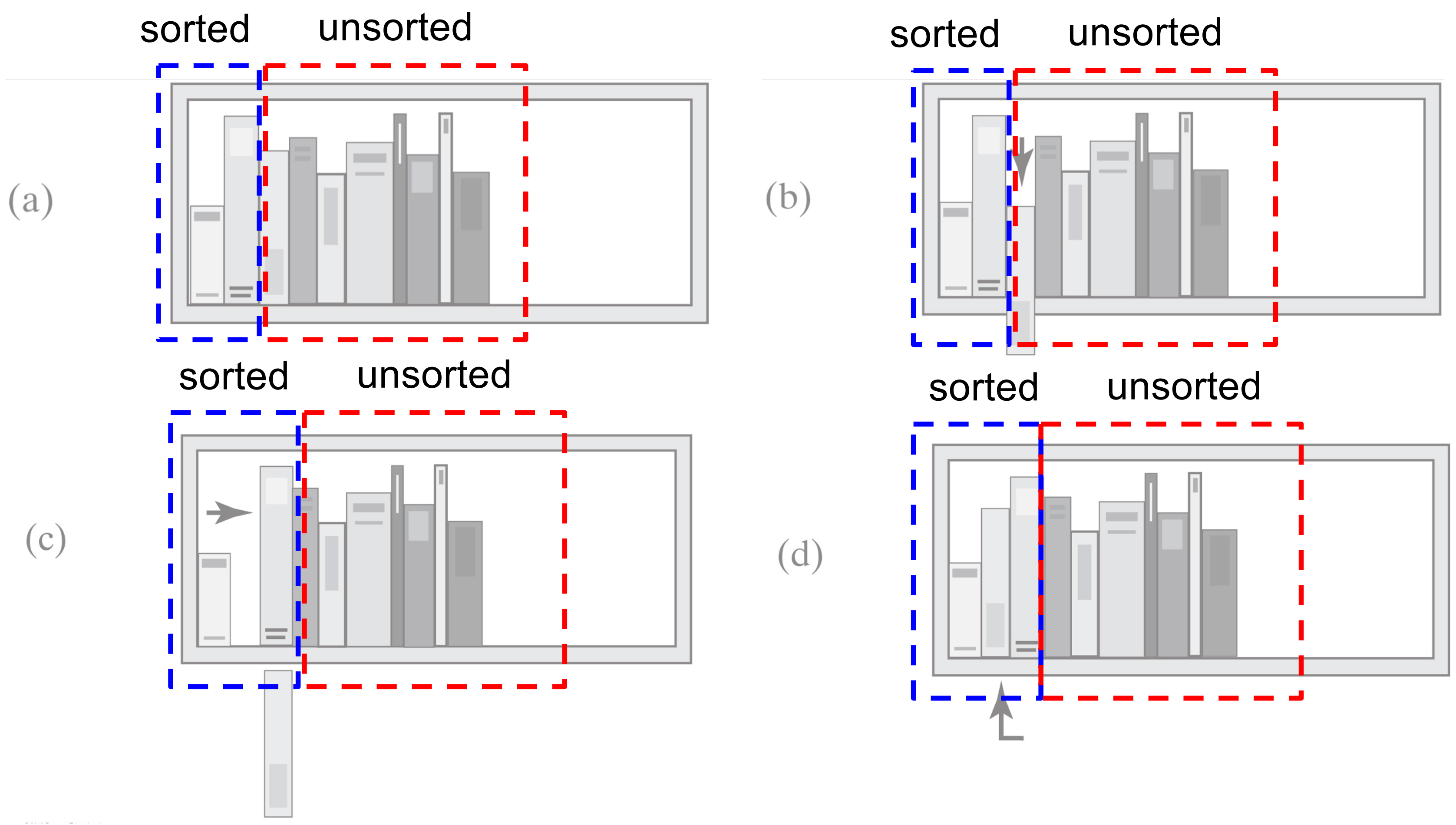
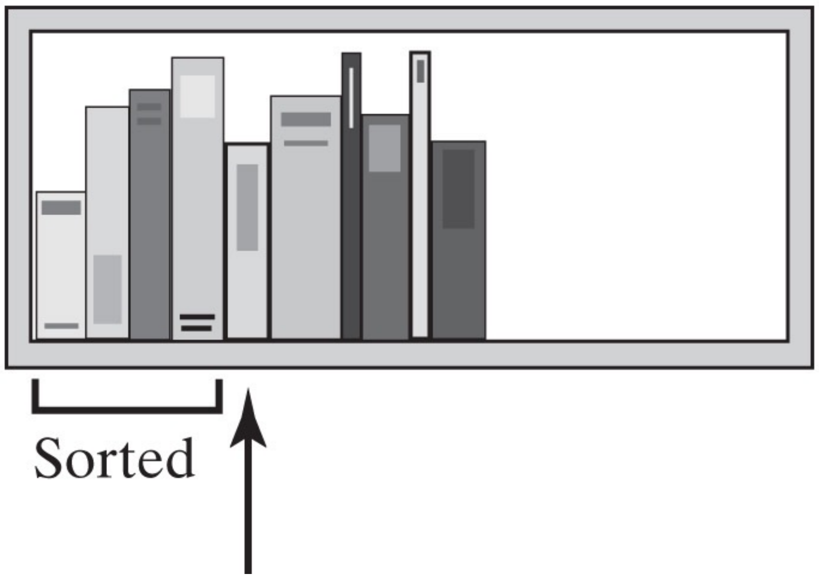


Figure 15-3: The placement of the third book during an insertion sort

Insertion Sort



1. Remove the next unsorted book.
2. Slide the sorted books to the right one by one until you find the right spot for the removed book.
3. Insert the book into its new position

Figure 15-4: An insertion sort of books

Iterative Insertion Sort

Iterative Insertion Sort

```
Algorithm insertionSort(a, first, last)

// Sorts the array entries a[first] through a[last] iteratively.
for (unsorted = first + 1 through last)
{
    nextToInsert = a[unsorted]
    insertInOrder(nextToInsert, a, first, unsorted – 1)
}
```

Iterative algorithm describes an insertion sort of the entries at indices `first` through `last` of the array `a`

Iterative Insertion Sort

```
Algorithm insertInOrder(anEntry, a, begin, end)
// Inserts anEntry into the sorted entries a[begin] through a[end]

index  =  end // Index of last entry in the sorted portion

// Make room, if needed, in sorted portion for another entry
while ( (index >= begin) and (anEntry < a[index]) )
{
     a[index + 1] = a[index] // Make room
     index--
}

// Assertion:  a[index  +  1] is available.
a[index + 1]  = anEntry  // Insert
```

Pseudocode of method, **insertInOrder**, to perform the insertions.

Iterative Insertion Sort

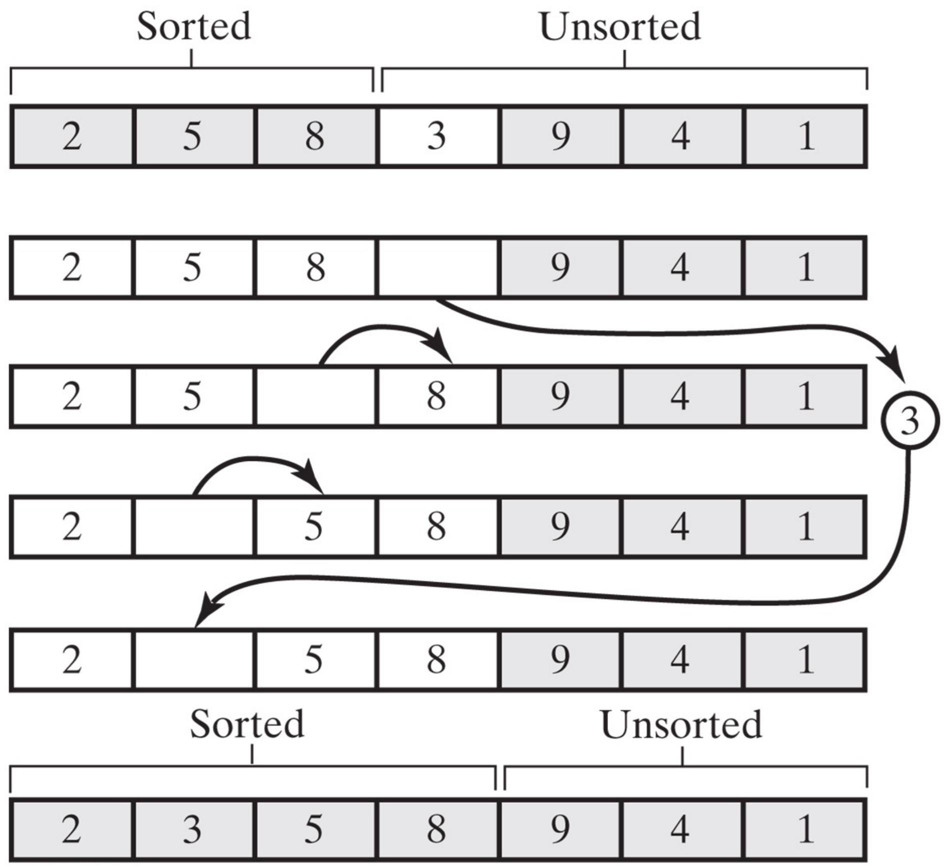


Figure 15-5: Inserting the next unsorted entry into its proper location within the sorted portion of an array during an insertion sort

Iterative Insertion Sort

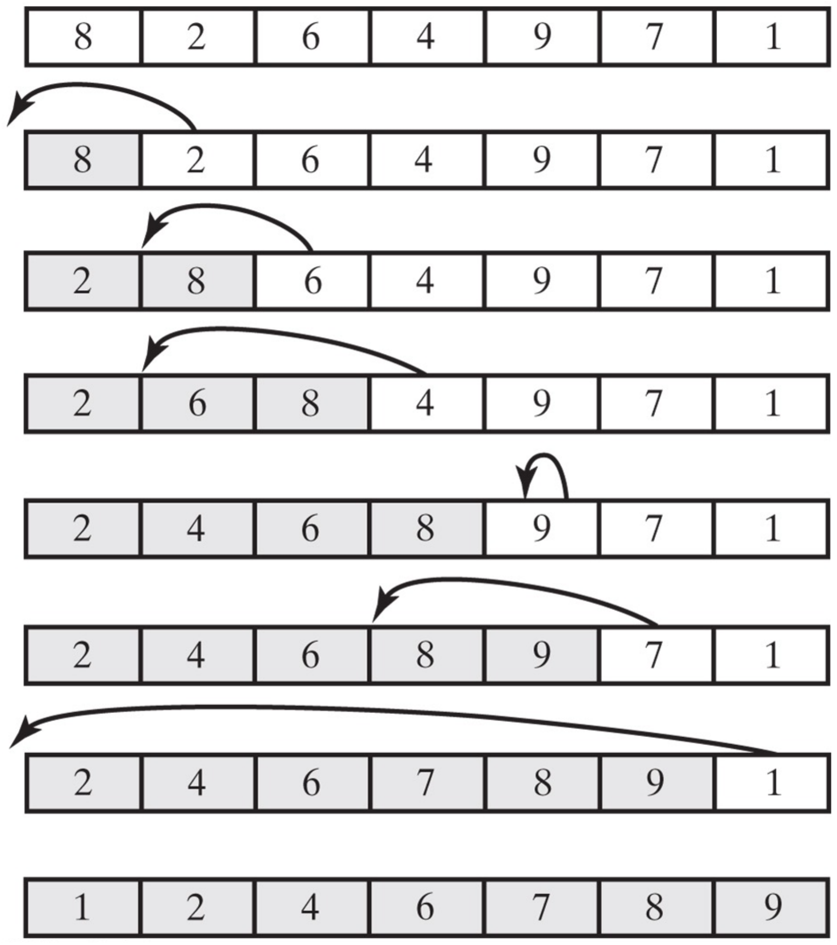


Figure 15-6: An insertion sort of an array of integers into ascending order

The Efficiency of Insertion Sort

- For an array of *n* entries, first is 0 and last is n-1.
- The `for` loop execute n-1 times, i.e., `insertInOrder` is called n-1 times. Within `insertInOrder`, begin is 0 and end ranges from 0 to n-2. The loop within `insertInOrder` executes at most end-begin+1 times. This loop executes at most a total of

$$1 + 2 + \cdots + (n - 1)$$

- This sum is $n(n-1)/2$.
- It means the insertion sort is $O(n^2)$.

Exercise (L26_E1)

- Please download “L26_E1” project from “In-class Exercises” on the Canvas.
- Please fill in the blank to finish the Insertion Sort

```
Input: 2, 4, 5, 3, 34, 11, 23

Output:
    2, 4, 5, 3, 34, 11, 23,
    2, 3, 4, 5, 34, 11, 23,
    2, 3, 4, 5, 11, 34, 23,
    2, 3, 4, 5, 11, 23, 34,
```

Exercise

```
public class L25_E1 {
  public static void main(String[] args){
    int[] inputData = {2, 4, 5, 3, 34, 11, 23};
    insertionSort(inputData);
  }

  public static void insertionSort(int array[]){
    int n = array.length;
    boolean swap = false;
    printNumber(array);
    for(int i = 1; i < n; i++){
      //ADD YOUR CODE HERE - INSERTION SORT
    }
  }

  private static void printNumber(int[] input){
    for(int i = 0; i< input.length;i++){
      System.out.print(input[i] + ", ");
    }
    System.out.print("\n");
  }
}
```

Answer

```
public class L25_E1 {

  public static void main(String[] args){
    int[] inputData = { 2, 4, 5, 3, 34, 11, 23};
    insertionSort(inputData);
  }

  public static void insertionSort(int array[]){
    int n = array.length;
    boolean swap = false;
    printNumber(array);
    for(int i = 1; i < n; i++){
      int key = array[i];
      int j = i - 1;
      while((j > -1)&&(array[j]>key)){
        array[j+1] = array[j];
        j--;
        swap = true;
      }

      array[j+1] = key;
      if(swap){
        printNumber(array);
        swap = false;
      }
    }
  }

  private static void printNumber(int[] input){
    for(int i = 0; i< input.length;i++){
      System.out.print(input[i] + ", ");
    }
    System.out.print("\n");
  }

}
```

Recursive Insertion Sort

Recursive Insertion Sort

```
Algorithm insertionSort(a, first, last)
// Sorts the array entries a[first] through a[last] recursively.

if (the array contains more than one entry)
{
    Sort the array entries a[first] through a[last – 1]

    Insert the last entry a[last] into its correct sorted position
    within the rest of the array
}
```

This pseudocode describes a recursive insertion sort.

Recursive Insertion Sort (cont.)

```
public static <T extends Comparable<? super T>>
       void insertionSort(T[] a, int first, int last)
{
   if (first < last)
   {

      // Sort all but the last entry
      insertionSort(a, first, last - 1);

      // Insert the last entry in sorted order
      insertInOrder(a[last], a, first, last - 1);

   } // end if
} // end insertionSort
```

Implementation of the recursive insertion sort.

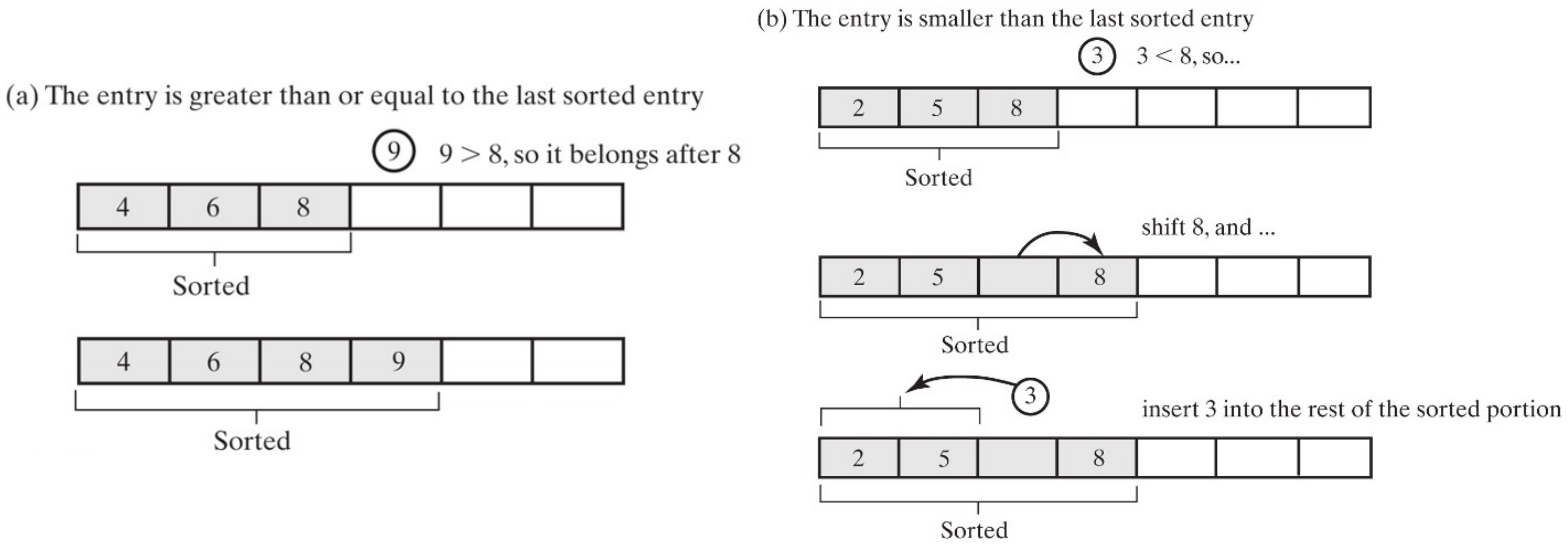
Recursive Insertion Sort (cont.)

```
Algorithm insertInOrder(anEntry, a, begin, end)
// Inserts anEntry into the sorted array entries a[begin] through a[end].

if (anEntry >= a[end])
    a[end + 1] = anEntry
else
{
    a[end + 1] = a[end]
    insertInOrder(anEntry, a, begin, end - 1)
}
```

First draft of `insertInOrder` algorithm

Recursive Insertion Sort (cont.)


(a) The entry is greater than or equal to the last sorted entry
9
9 > 8, so it belongs after 8
4
6
8
Sorted
4
6
8
9
Sorted
(b) The entry is smaller than the last sorted entry
3
3 < 8, so...
2
5
8
Sorted
shift 8, and ...
2
5
8
Sorted
3
insert 3 into the rest of the sorted portion
2
5
8
Sorted

Inserting the first unsorted entry into the sorted portion of the array

Recursive Insertion Sort (cont.)

```
Algorithm insertInOrder(anEntry, a, begin, end)
// Inserts anEntry into the sorted array entries a[begin] through a[end].

// Revised draft.
if (anEntry >= a[end])
    a[end + 1] = anEntry
else if (begin < end)
{
    a[end + 1] = a[end]
    insertInOrder(anEntry, a, begin, end – 1)
}
else // begin == end and anEntry < a[end]
{
    a[end + 1] = a[end]
    a[end] = anEntry
}
```

The algorithm `insertInOrder`: final draft.

Insertion Sort of a Chain of Linked Nodes

Insertion Sort of a Chain of Linked Nodes

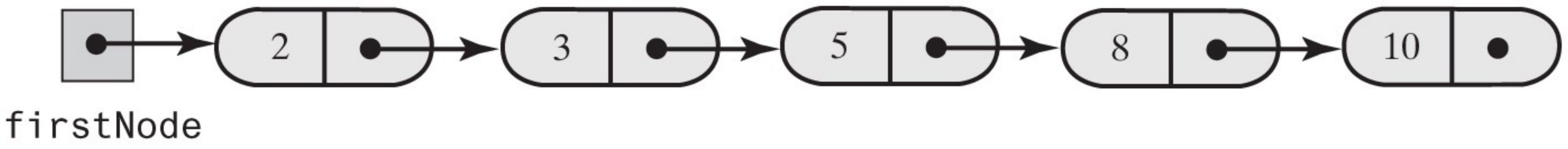


Figure 15-8: A chain of integers sorted into ascending order

Insertion Sort of a Chain of Linked Nodes

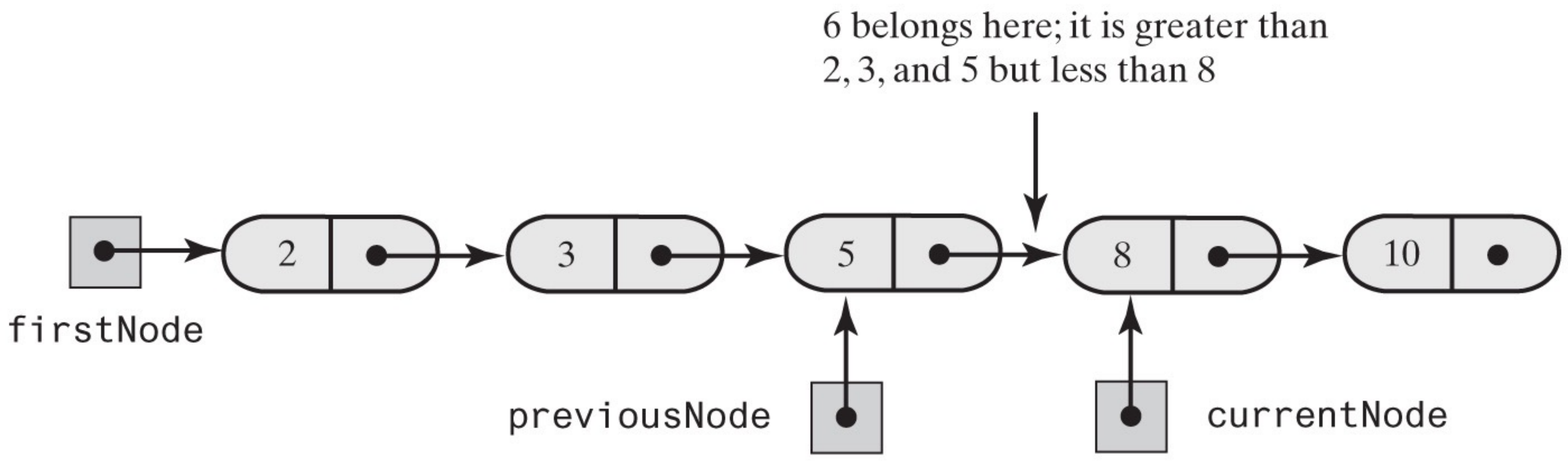

6 belongs here; it is greater than
2, 3, and 5 but less than 8
2
3
5
8
10
firstNode
previousNode
currentNode

Figure 15-9: During the traversal of a chain to locate the insertion point, save a reference to the node before the current one

Insertion Sort of a Chain of Linked Nodes

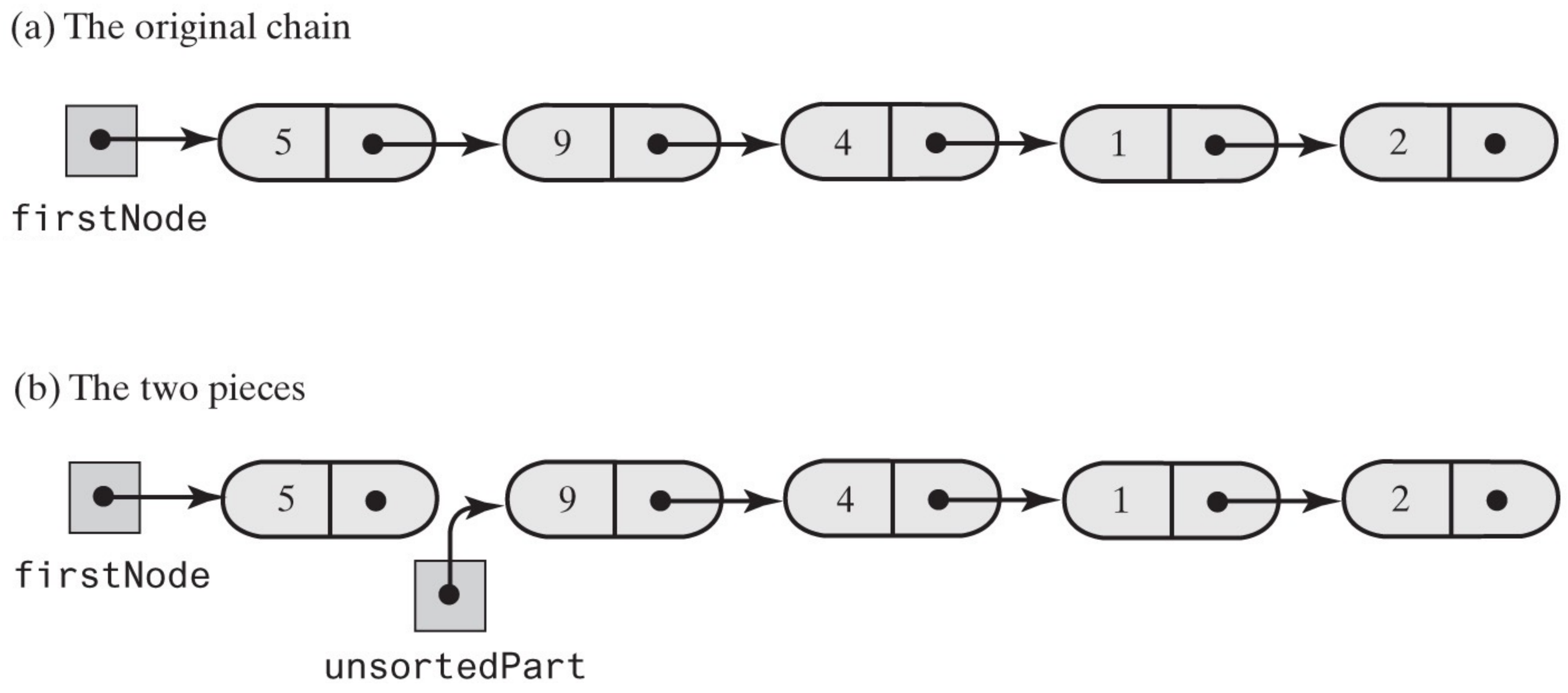


Figure 15-10: Breaking a chain of nodes into two pieces as the first step in an insertion sort: (a) the original chain; (b) the two pieces

Insertion Sort of a Chain of Linked Nodes

```
public class LinkedGroup<T extends Comparable<? super T>>
{
   private Node firstNode;
   int length; // Number of objects in the group

   //  . . .
   private class Node
   {
     //  private inner class Node is implemented here.
   }
```

Add a sort method to a class `LinkedGroup`
that uses a linked chain to represent a certain collection

Insertion Sort of a Chain of Linked Nodes

```
public void insertionSort()
{
   // If fewer than two items are in the list, there is nothing to do
   if (length > 1)
   {
      // Assertion: firstNode != null

      // Break chain into 2 pieces: sorted and unsorted
      Node unsortedPart = firstNode.getNextNode();

      // Assertion: unsortedPart != null
      firstNode.setNextNode(null);

      while (unsortedPart != null)
      {
         Node nodeToInsert = unsortedPart;
         unsortedPart = unsortedPart.getNextNode();
         insertInOrder(nodeToInsert);
      } // end while
   } // end if
} // end insertionSort
```

The method to perform the insertion sort.

```java
private void insertInOrder(Node nodeToInsert)
{
   T item = nodeToInsert.getData();
   Node currentNode = firstNode;
   Node previousNode = null;

   // Locate insertion point
   while ( (currentNode != null) &&
           (item.compareTo(currentNode.getData()) > 0) )
   {
      previousNode = currentNode;
      currentNode = currentNode.getNextNode();
   } // end while

   // Make the insertion
   if (previousNode != null)
   {  // Insert between previousNode and currentNode
      previousNode.setNextNode(nodeToInsert);
      nodeToInsert.setNextNode(currentNode);
   }
   else // Insert at beginning
   {
      nodeToInsert.setNextNode(firstNode);
      firstNode = nodeToInsert;
   } // end if
} // end insertInOrder
```

This class has an inner class `Node` that has set and get methods