Lecture 23:An Introduction to Sorting

CS 0445: Data Structures

Constantinos Costa

http://db.cs.pitt.edu/courses/cs0445/current.term/

Nov 05, 2019, 8:00-9:15
University of Pittsburgh, Pittsburgh, PA

Sorting

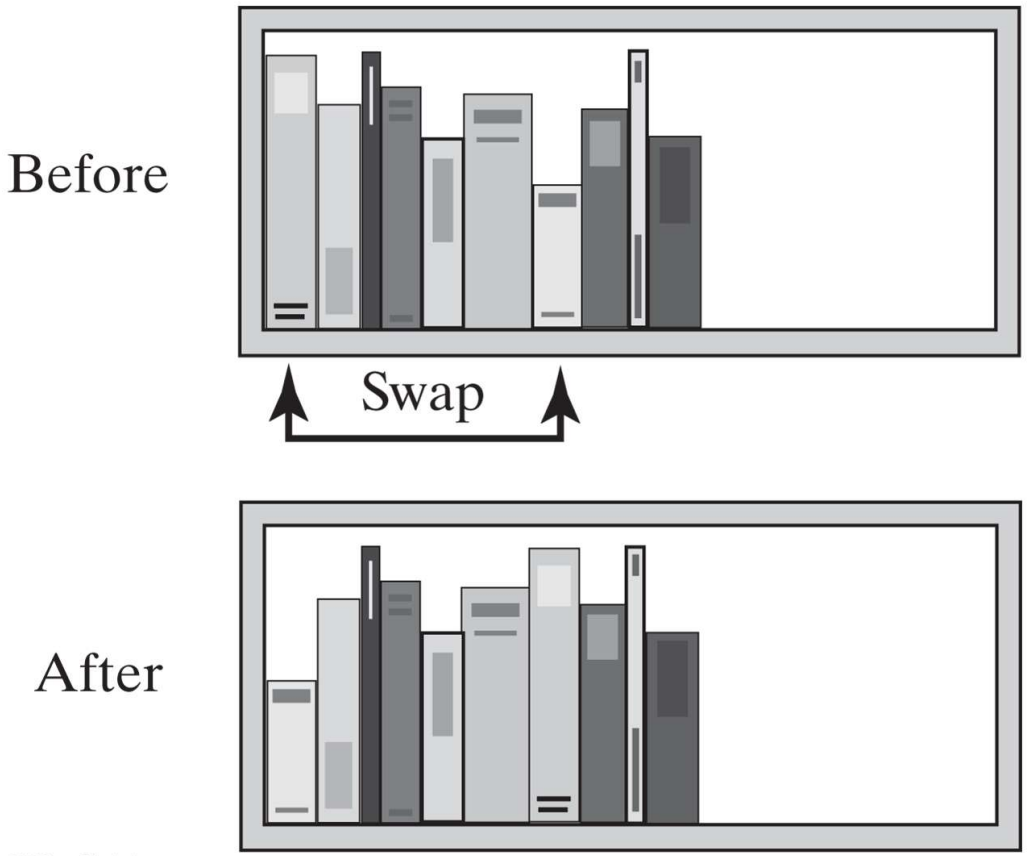
- We seek algorithms to arrange items, a_i such that:

$$\text{entry 1} \leq \text{entry 2} \leq \ldots \leq \text{entry n}$$

- Sorting an array is usually easier than sorting a chain of linked nodes
- Efficiency of a sorting algorithm is significant

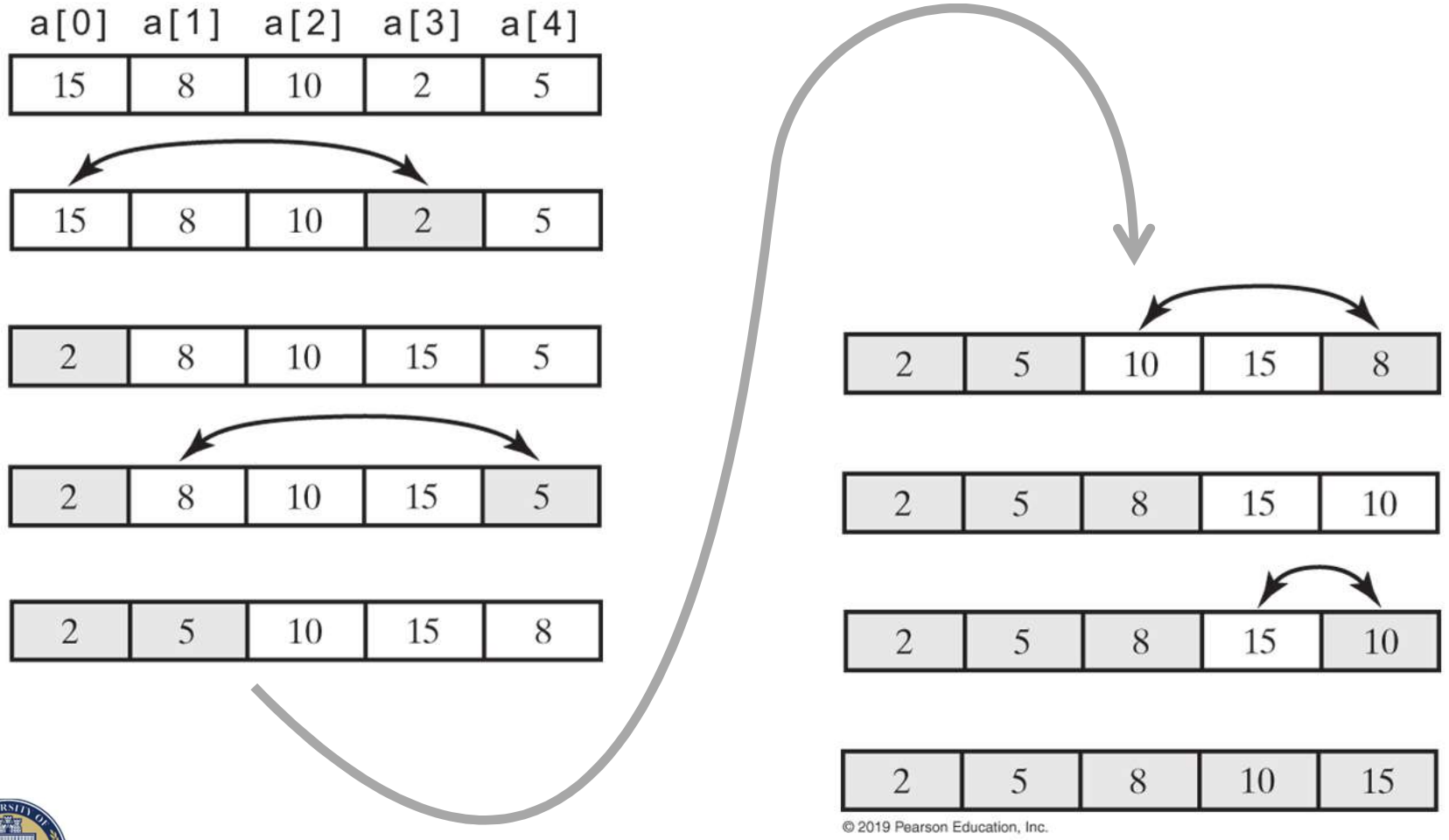
Selection Sort

- Before and after exchanging the shortest book and the first book



Selection Sort

- A selection sort of an array of integers into ascending order



Iterative Selection Sort

- This pseudocode describes an iterative algorithm for the selection sort

```
Algorithm selectionSort(a, n)
// Sorts the first n entries of an array a.
for (index = 0; index < n − 1; index++)
{
        indexOfNextSmallest = the index of the smallest value among
                                                a[index], a[index + 1], . . . , a[n − 1]
        Interchange the values of a[index] and a[indexOfNextSmallest]
        // Assertion: a[0] ≤ a[1] ≤ . . . ≤ a[index], and these are the smallest
        //of the original array entries. The remaining array entries begin at a[index + 1].
}
```

Iterative Selection Sort (Part 1)

- A class for sorting an array using selection sort

```
/**  A class of static, iterative methods for sorting an array of
  Comparable objects from smallest to largest.  */
public class SortArray
{
  /** Sorts the first n objects in an array into ascending order.
     @param a  An array of Comparable objects.
     @param n  An integer > 0. */
  public static <T extends Comparable<? super T>>
       void selectionSort(T[] a, int n)
  {
    for (int index = 0; index < n - 1; index++)
    {
      int indexOfNextSmallest = getIndexOfSmallest(a, index, n - 1);
      swap(a, index, indexOfNextSmallest);
    // Assertion: a[0] <= a[1] <= . . . <= a[index] <= all other a[i]
    } // end for
  } // end selectionSort
```

Iterative Selection Sort (Part 2)

- A class for sorting an array using selection sort

```
// Finds the index of the smallest value in a portion of an array a.
// Precondition: a.length > last >= first >= 0.
// Returns the index of the smallest value among
// a[first], a[first + 1], . . . , a[last].
private static <T extends Comparable<? super T>>
      int getIndexOfSmallest(T[] a, int first, int last)
{
  T min = a[first];
  int indexOfMin = first;
  for (int index = first + 1; index <= last; index++)
  {
    if (a[index].compareTo(min) < 0)
    {
      min = a[index];
      indexOfMin = index;
    } // end if
    // Assertion: min is the smallest of a[first] through a[index].
  } // end for

  return indexOfMin;
} // end getIndexOfSmallest
```

Iterative Selection Sort (Part 3)

- A class for sorting an array using selection sort

```
// Swaps the array entries a[i] and a[j].
 private static void swap(Object[] a, int i, int j)
 {
   Object temp = a[i];
   a[i] = a[j];
   a[j] = temp;
 } // end swap
} // end SortArray
```

Recursive Selection Sort

- Recursive selection sort algorithm

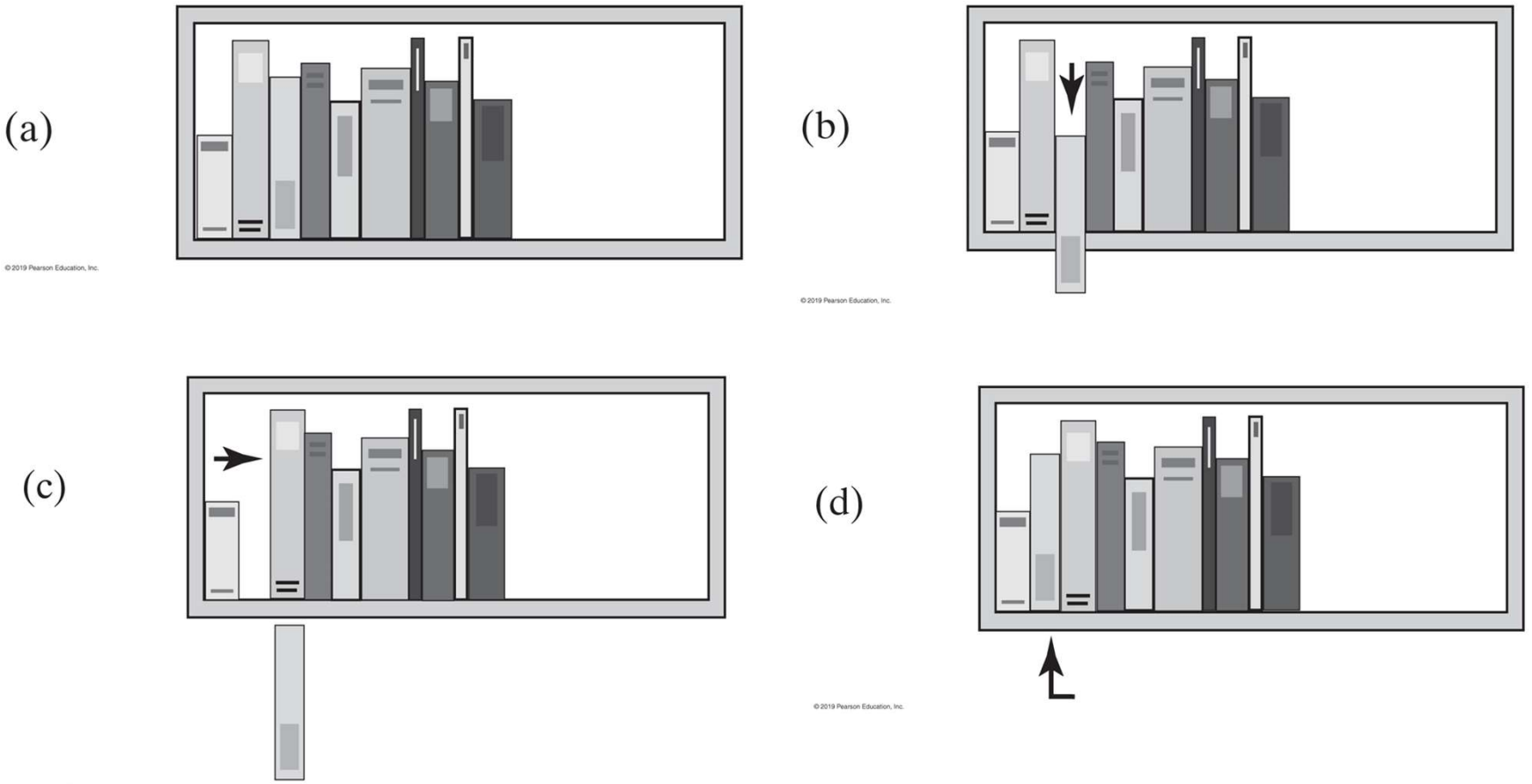
```
Algorithm selectionSort(a, first, last)
// Sorts the array entries a[first] through a[last] recursively.
if (first < last)
{
   indexOfNextSmallest = the index of the smallest value among
                                          a[first], a[first + 1], . . . , a[last]
   Interchange the values of a[first] and a[indexOfNextSmallest]
   // Assertion: a[0] ≤ a[1] ≤ . . . ≤ a[first] and these are the smallest
   // of the original array entries. The remaining array entries begin at a[first + 1].
   selectionSort(a, first + 1, last)
}
```

Efficiency of Selection Sort

- Selection sort is $O(n^2)$ regardless of the initial order of the entries.
  - Requires $O(n^2)$ comparisons
  - Does only $O(n)$ swaps

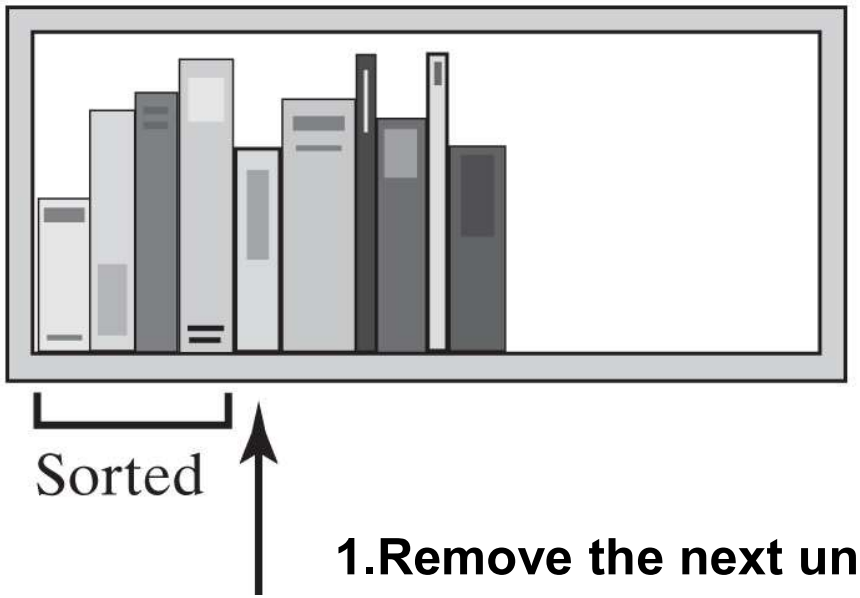
Insertion Sort

- The placement of the third book during an insertion sort



Insertion Sort

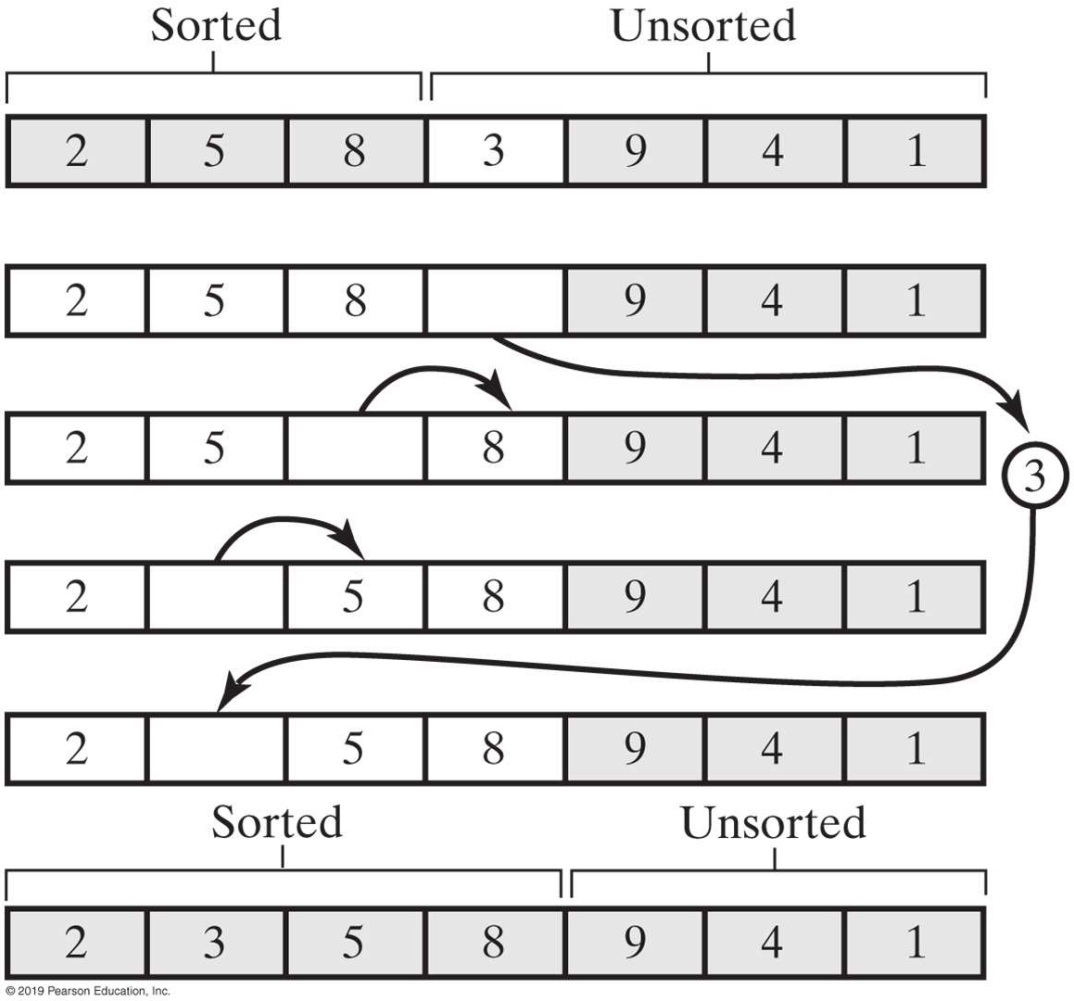
- An insertion sort of books



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

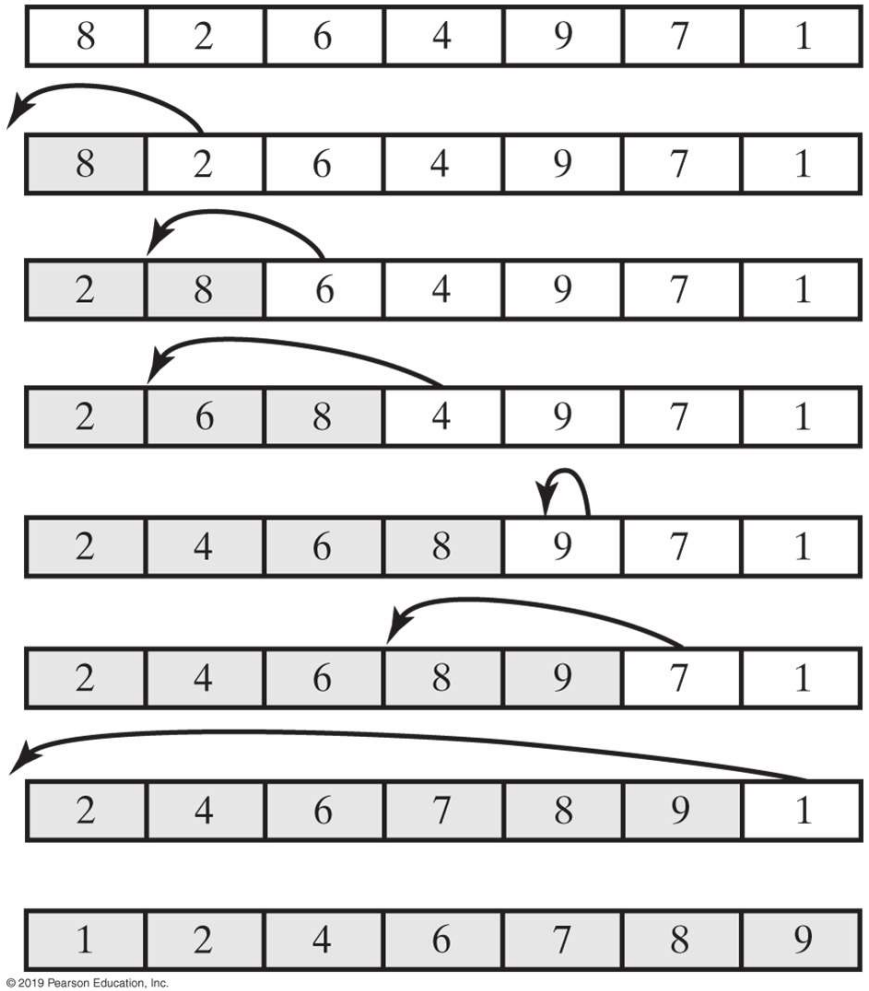
Insertion Sort

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



Insertion Sort

- An insertion sort of an array of integers into ascending order



Iterative Insertion Sort

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

```
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 Insertion Sort

- Pseudocode of method, `insertInOrder`, to perform the insertions.

```
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
```

Recursive Insertion Sort

- This pseudocode describes a 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
}
```

Recursive Insertion Sort

- Implementing the algorithm in Java

```
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
```

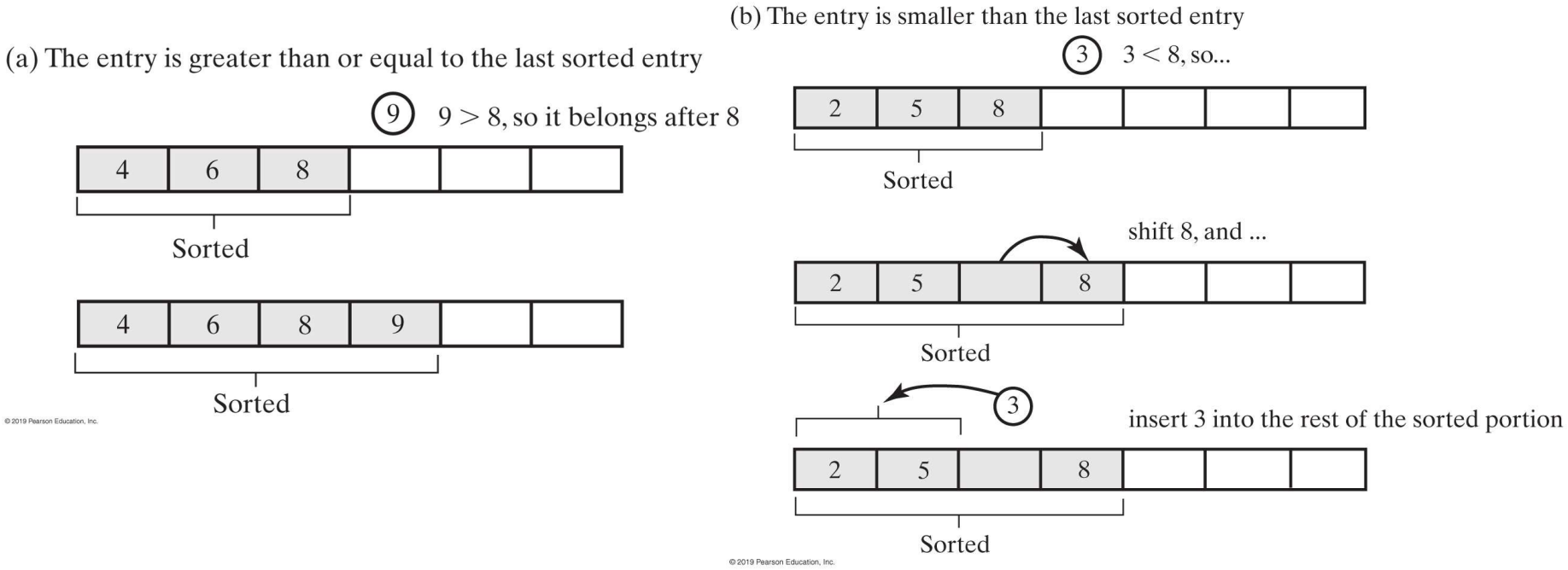
Recursive Insertion Sort

- First draft of `insertInOrder` algorithm.

```
Algorithm insertInOrder(anEntry, a, begin, end)
// Inserts anEntry into the sorted array entries a[begin] through a[end].
//First draft.
if (anEntry >= a[end])
                a[end + 1] =  anEntry
else
{
                a[end  +  1]  =  a[end]
                insertInOrder(anEntry, a, begin, end − 1)
}
```

Insertion Sort

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



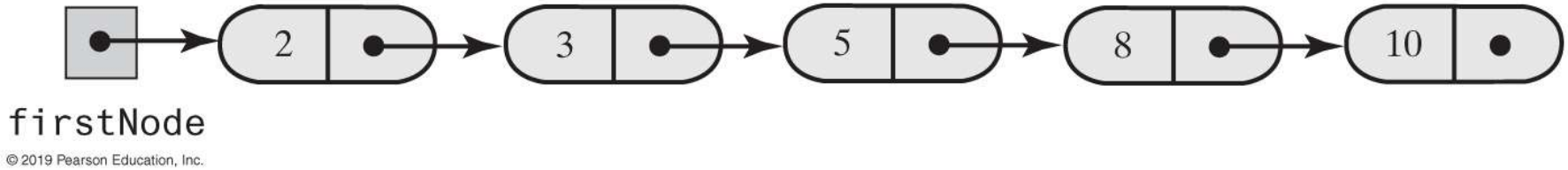
Recursive Insertion Sort

- The algorithm `insertInOrder`: final draft.
  Note: insertion sort efficiency (worst case) is $O(n^2)$

```
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
}
```

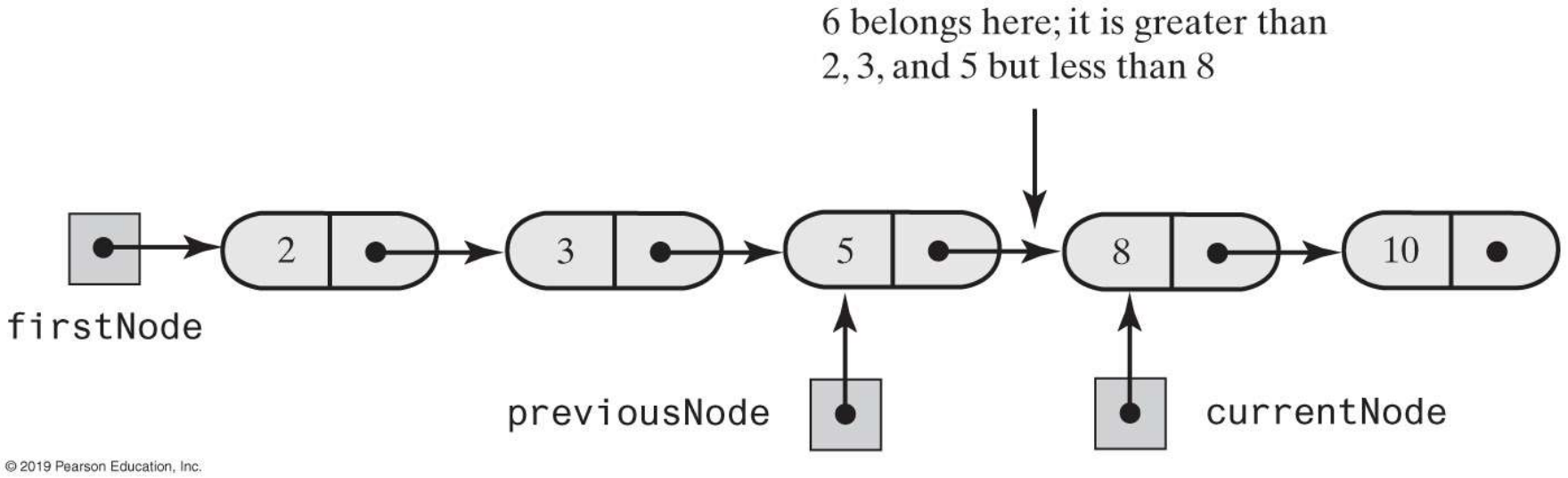
Insertion Sort with a Linked Chain

- A chain of integers sorted into ascending order



Insertion Sort with a Linked Chain

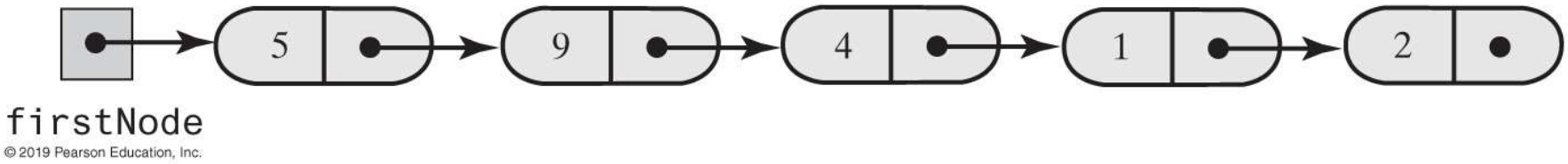
- During the traversal of a chain to locate the insertion point, save a reference to the node before the current one



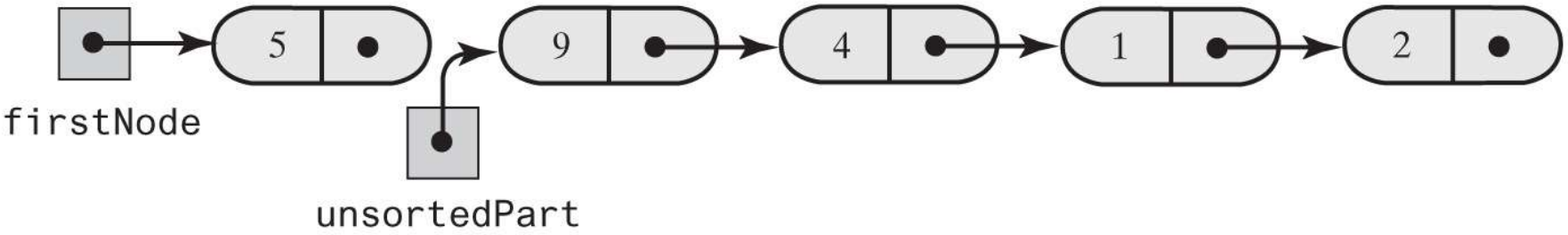
Insertion Sort with a Linked Chain

- 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 with a Linked Chain

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

```
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.
      }
```

Insertion Sort with a Linked Chain

- Class has an inner class `Node` with `set` and `get` methods

```
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
```

Insertion Sort with a Linked Chain

- Insertion sort method

```
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
```

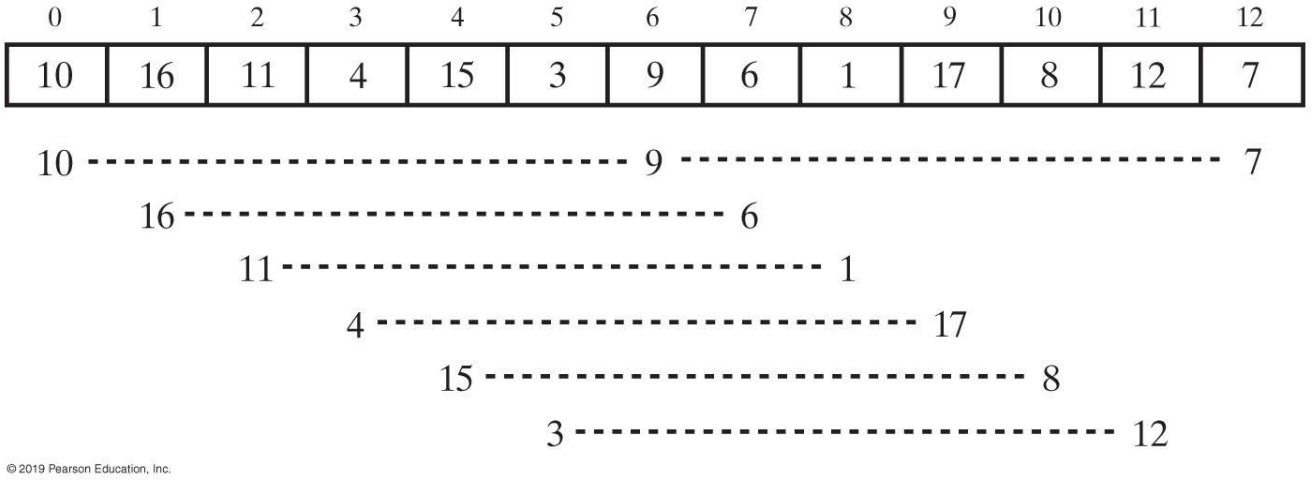
Shell Sort

- Algorithms so far are simple
  - but inefficient for large arrays at $O(n^2)$
- The more sorted an array is, the less work `insertInOrder` must do
- Improved insertion sort developed by Donald Shell

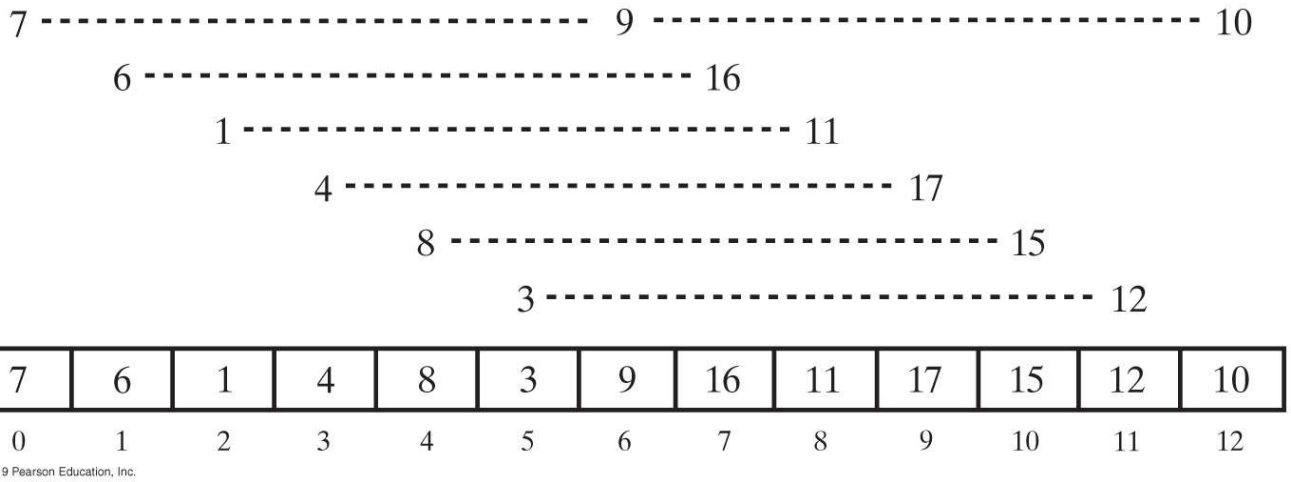
Shell Sort

- An array and the groups of entries whose indices are 6 apart before and after ordering groups

Before Ordering

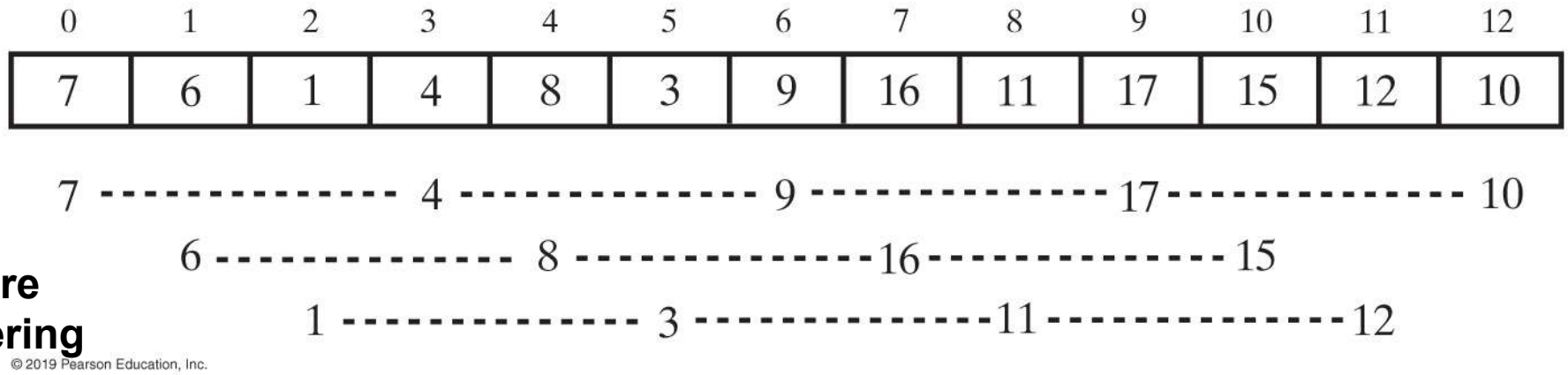


After Ordering

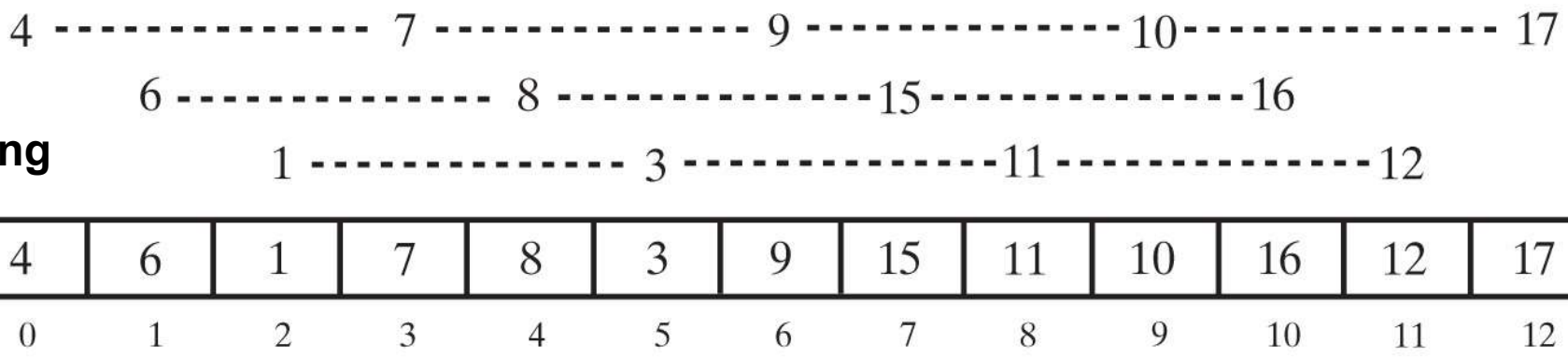


Shell Sort

- Grouped entries in the previous array whose indices are 3 apart before and after ordering groups



Before Ordering



After Ordering

Comparing Algorithms

- The time efficiencies of three sorting algorithms, expressed in Big Oh notation

	Best Case	Average Case	Worst Case
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Shell Sort	$O(n)$	$O(n^{1.5})$	$O(n^{1.5})$