University of Pittsburgh

Algorithms and Data Structures 1
CS 0445

Fall 2022

Sherif Khattab

ksm73@pitt.edu

Pitt SCI

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS 0445 slides.)

Announcements

- Upcoming Deadlines:
  - **Assignment 2: Late Deadline 11/9 @ 11:59 pm**
  - Lab 8: next Monday 11/14 @ 11:59 pm
  - Midterm reattempts: Thursday 11/10 @ 11:59 pm
- Live Support Session for Assignment 2
  - Video and slides available on Canvas
- QA Session on Piazza every Friday 4:30-5:30 pm

Today …

- Sorting Algorithms

Muddiest Points

- **Q: I have not seen stringBuilder before. Could you explain that?**
- StringBuilder is mutable, whereas String is immutable
- Internally, a resizable array is used
- Appending to StringBuilder is O(1), whereas appending to String is O(n)
    - StringBuilder sb = new StringBuilder(“Hello”);
    - sb.append(“!”); //O(1)
    - String s = new String(“Hello”);
    - s = s + “!”; //O(n)

Muddiest Points

- **Q: Could you show us how to input the file under debug mode? I can not follow to cwd and json part.**
- You need to edit launch.json to add an “args” field to the run configuration

Sorting

- We have seen a few container data structures
  - Bag, Stack, List
- Sorting Problem: arrange items in a List such that *entry 1* ≤ *entry 2* ≤ *. . .* ≤ *entry n*
- Efficiency of a sorting algorithm is significant
- Sorting an array is usually easier than sorting a chain of linked nodes

Sorting Algorithms

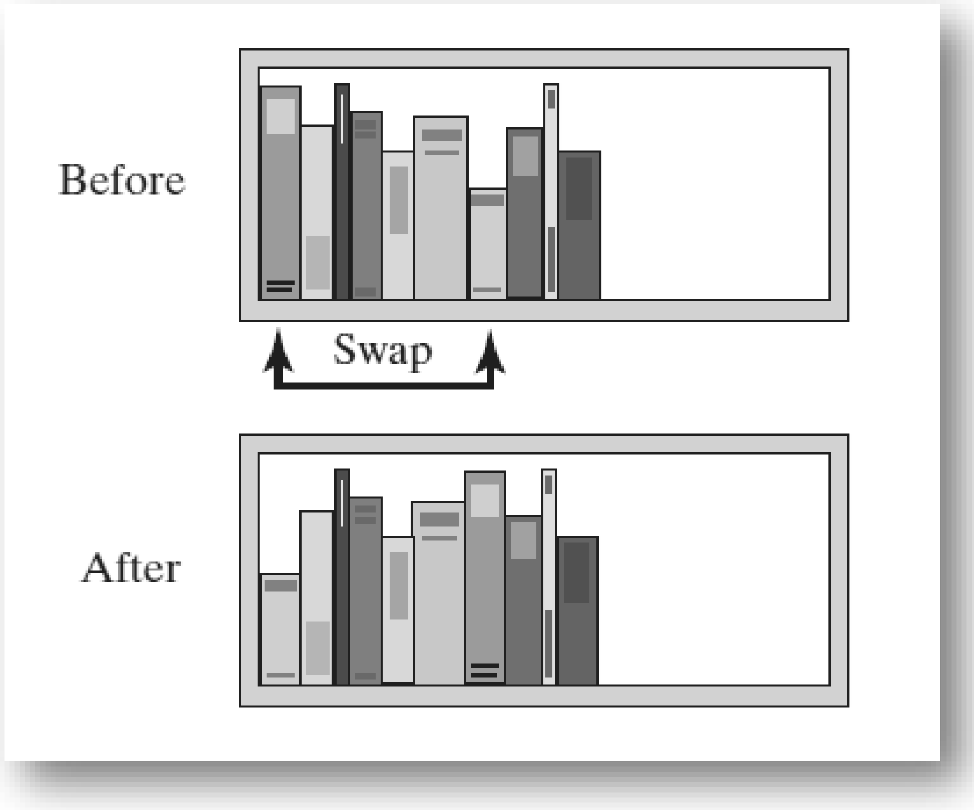
- $O(n^2)$
  - Selection Sort
  - Insertion Sort
  - Shell Sort
- $O(n \log n)$
  - Merge Sort
  - Quick Sort
- $O(n)$ Sorting
  - Radix Sort

Sorting Algorithms

- For each algorithm
  - understand the main concept using an example
  - implement the algorithm
    - on an Array
      - iterative
      - recursive
    - on a linked list
      - iterative
      - recursive

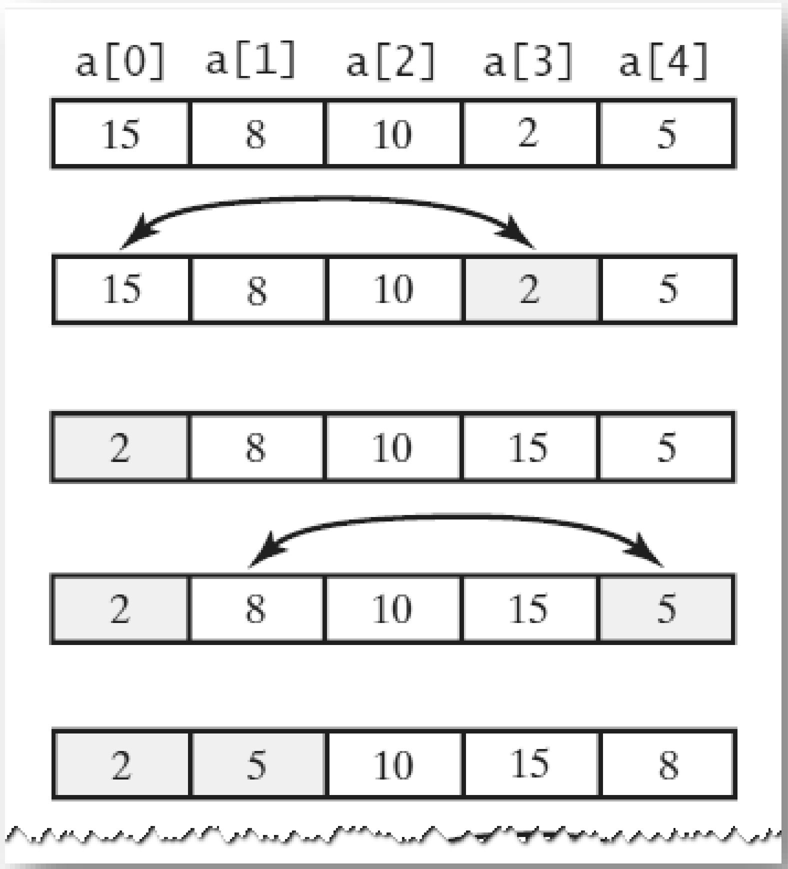
Selection Sort

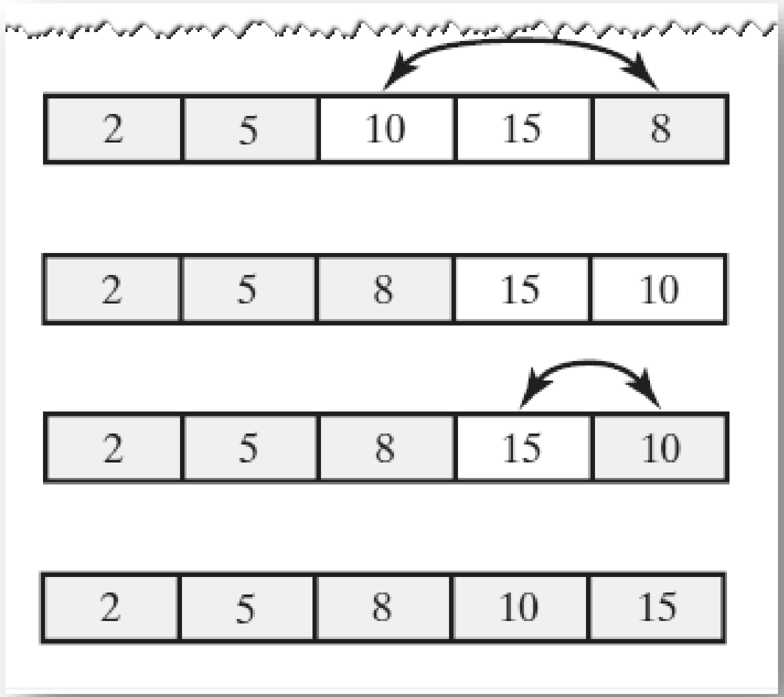
- Before and after exchanging the shortest book and the first book



Selection Sort

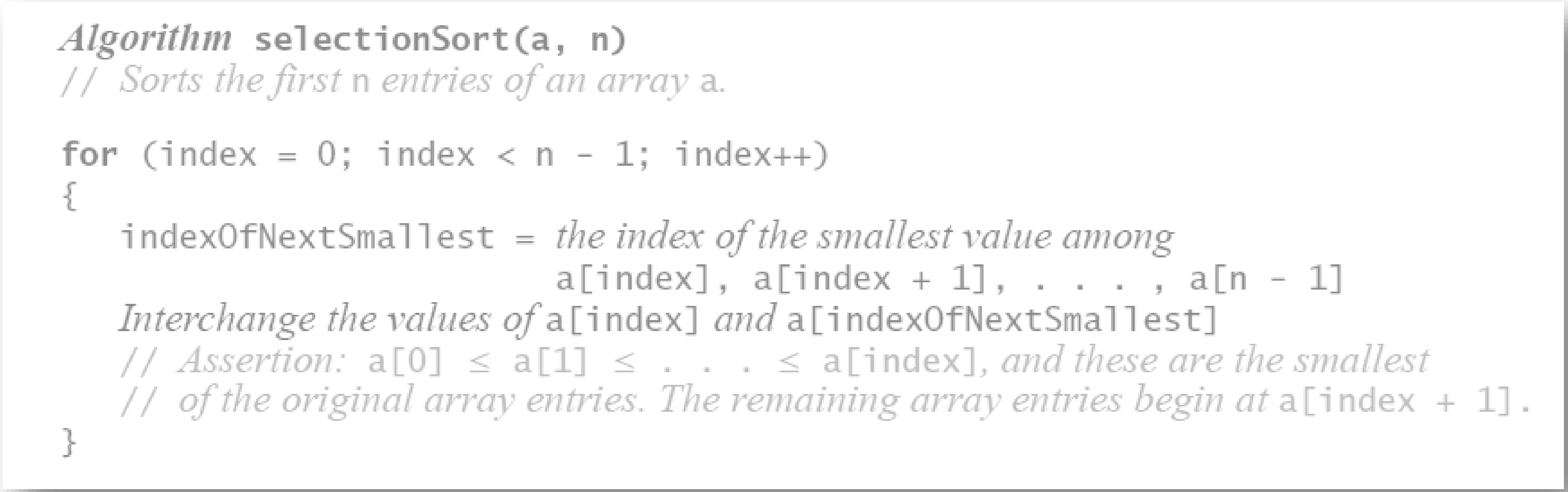
- A selection sort of an array of integers into ascending order





Iterative Selection Sort

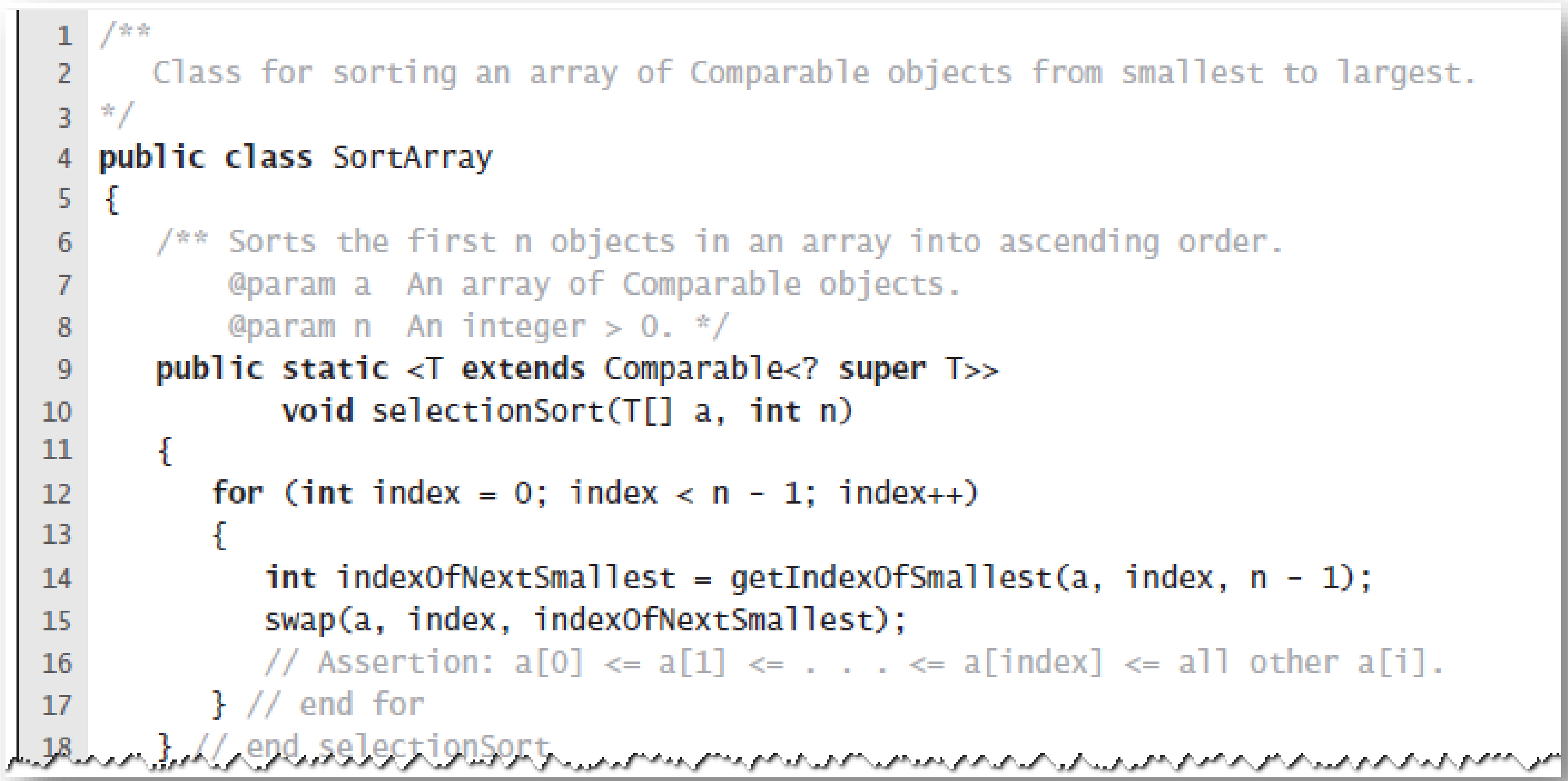
- This pseudocode describes an iterative algorithm for the selection sort



```
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

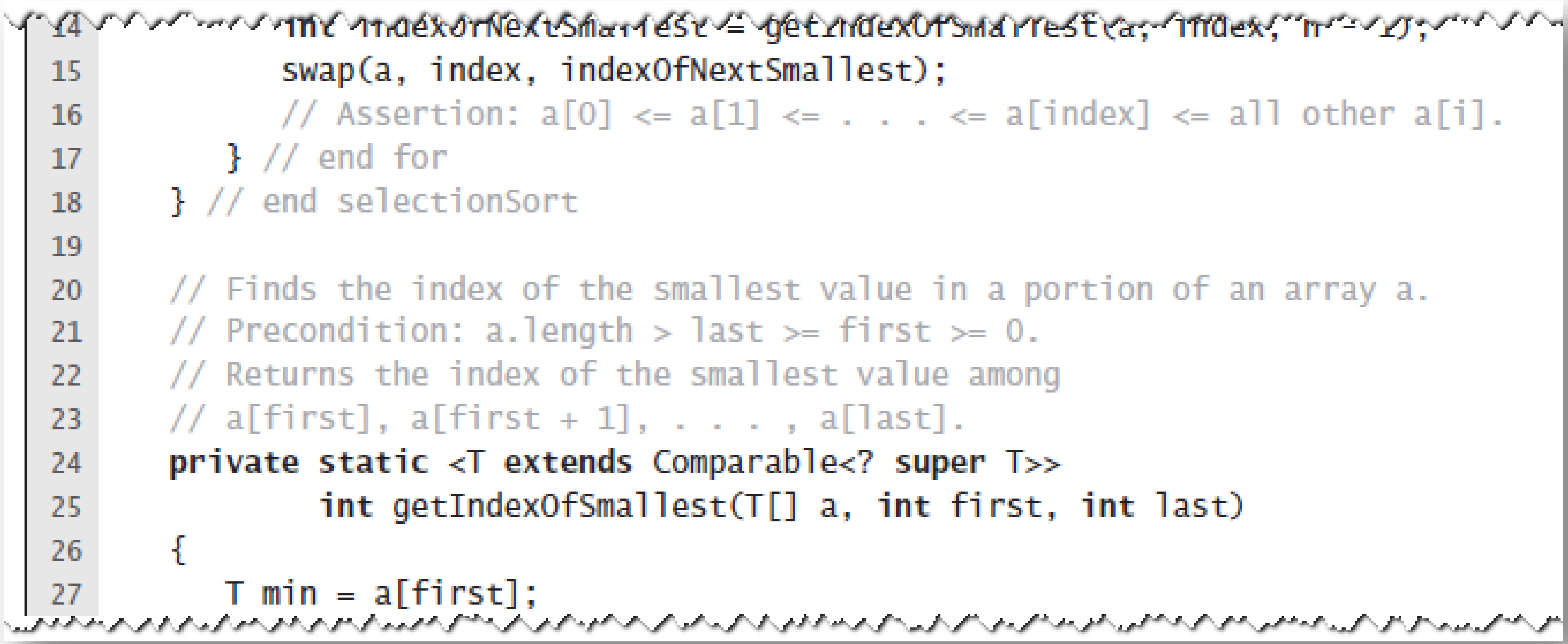
- A class for sorting an array using selection sort



```
/**
   Class 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

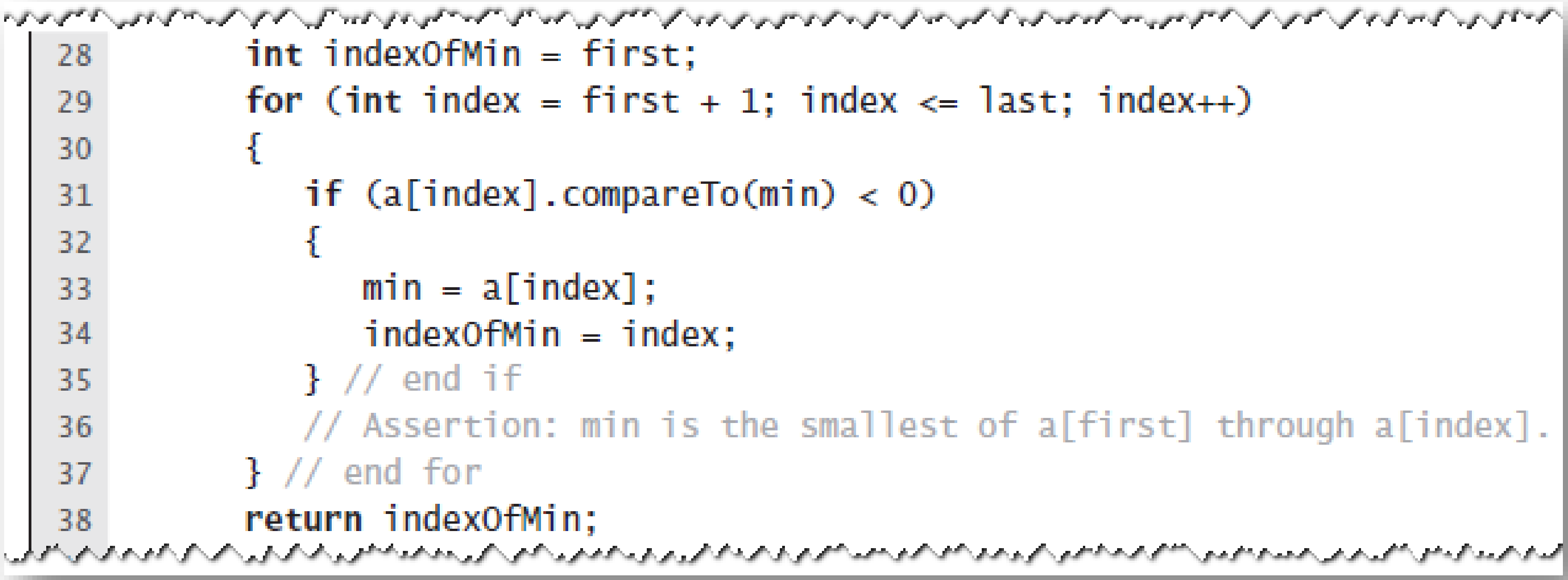
- A class for sorting an array using selection sort



```
          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

    // 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];
```

Iterative Selection Sort

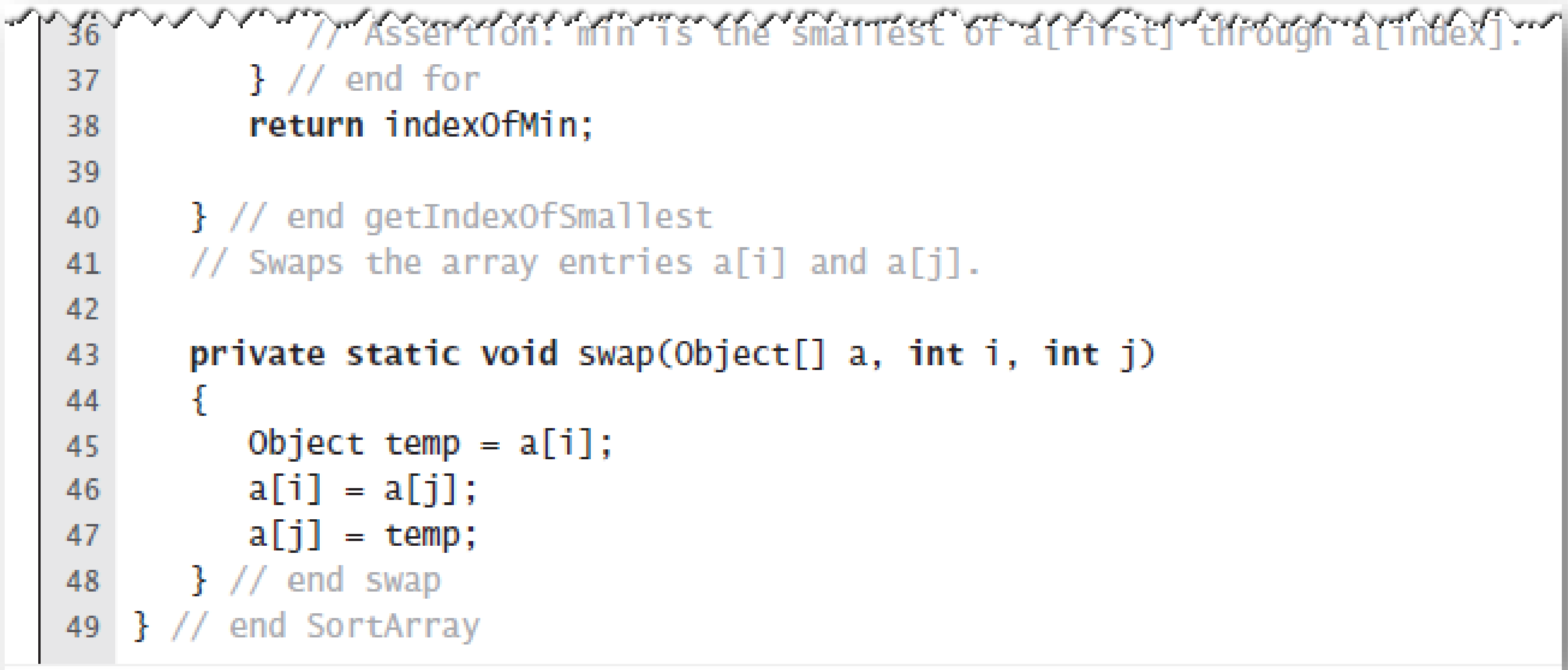
- A class for sorting an array using selection sort



```
      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;
```

Iterative Selection Sort

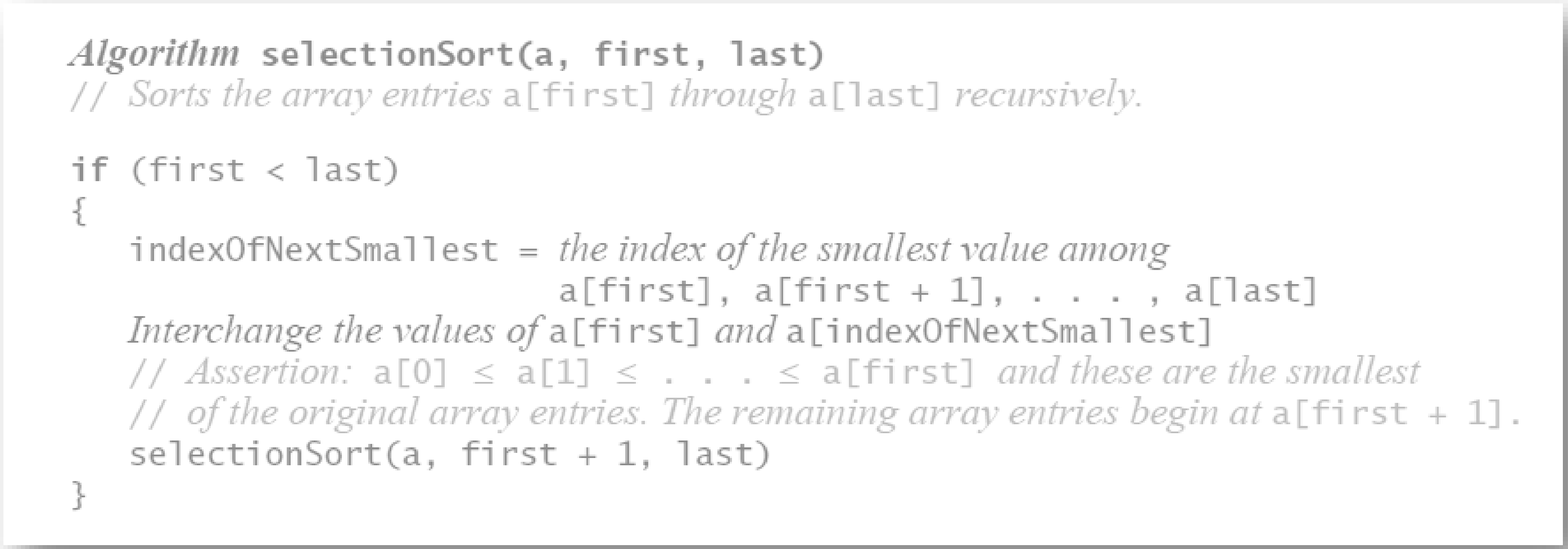
- A class for sorting an array using selection sort



```
            // Assertion: min is the smallest of a[first] through a[index].
        } // end for
        return indexOfMin;

    } // end getIndexOfSmallest
    // 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



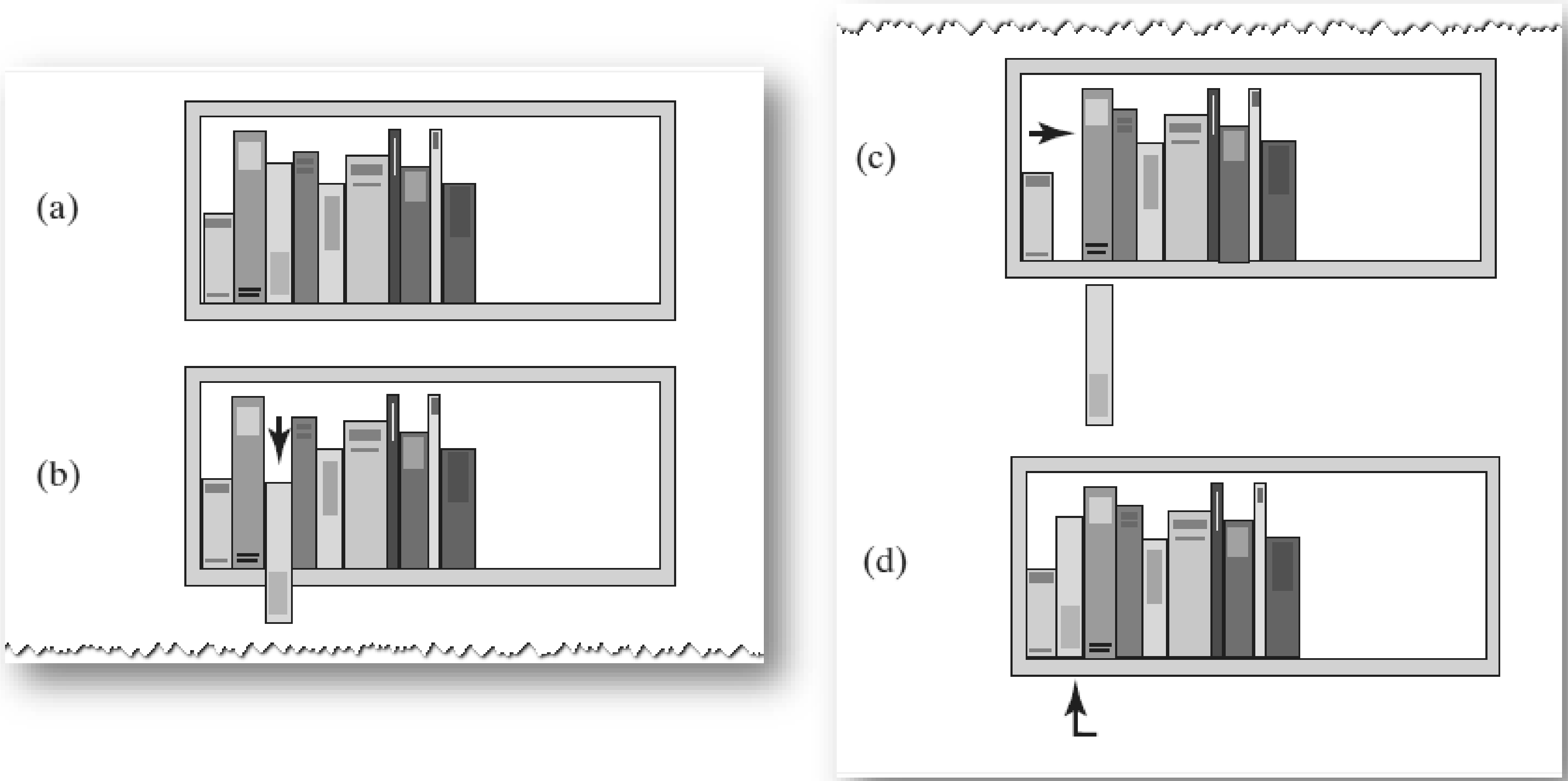
```
Algorithm selectionSort(a, first, last)
// Sorts the array entries a[first] through a[last] recursively.

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)
}
```

Selection Sort on a Chain

- Check code handouts and lecture recording

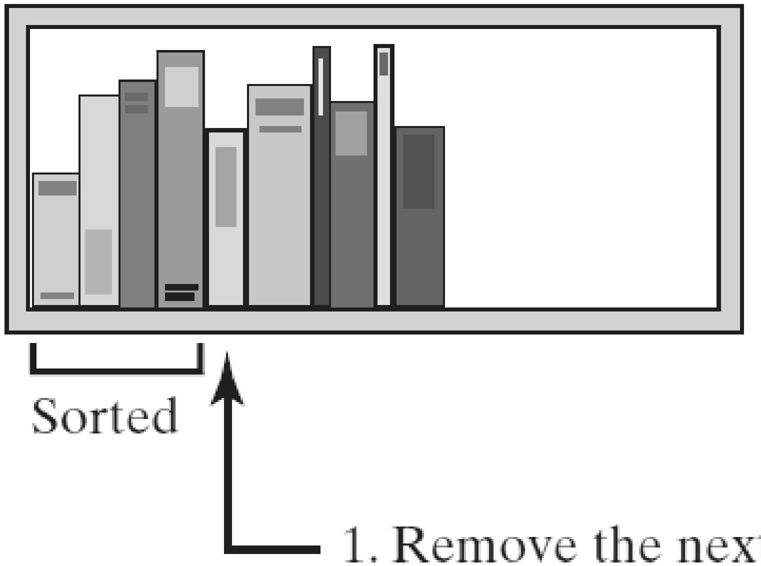
Insertion Sort

- The placement of the third book during an insertion sort



Insertion Sort

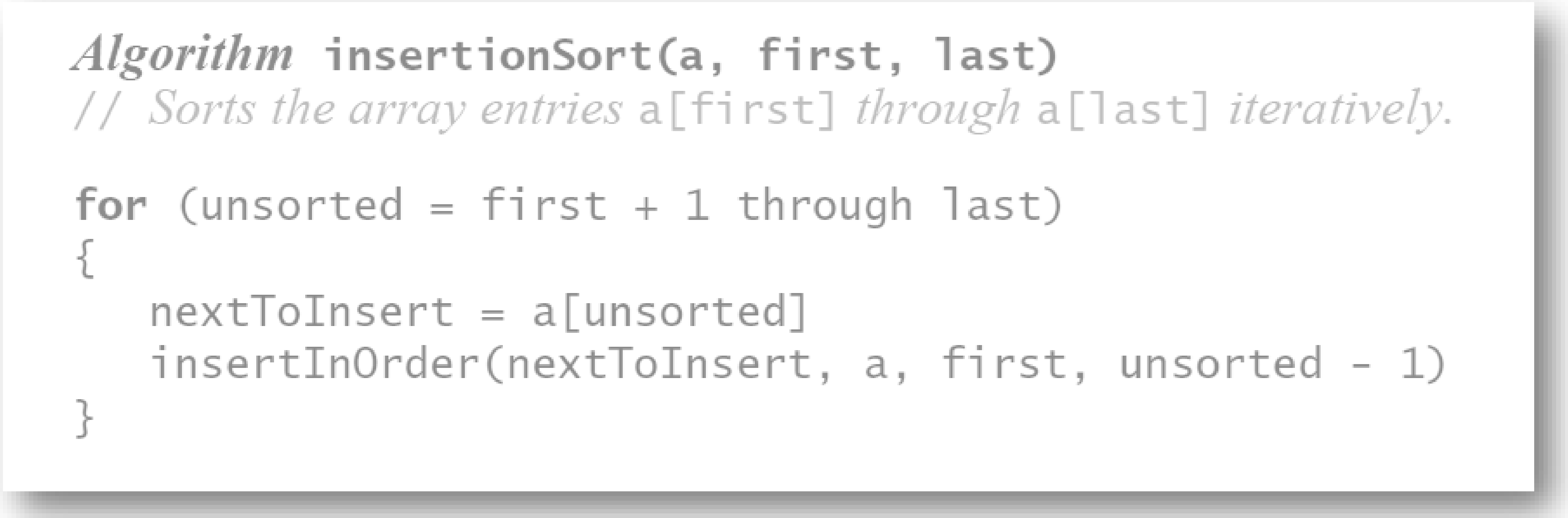
- An insertion sort of books



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.

Iterative Insertion Sort

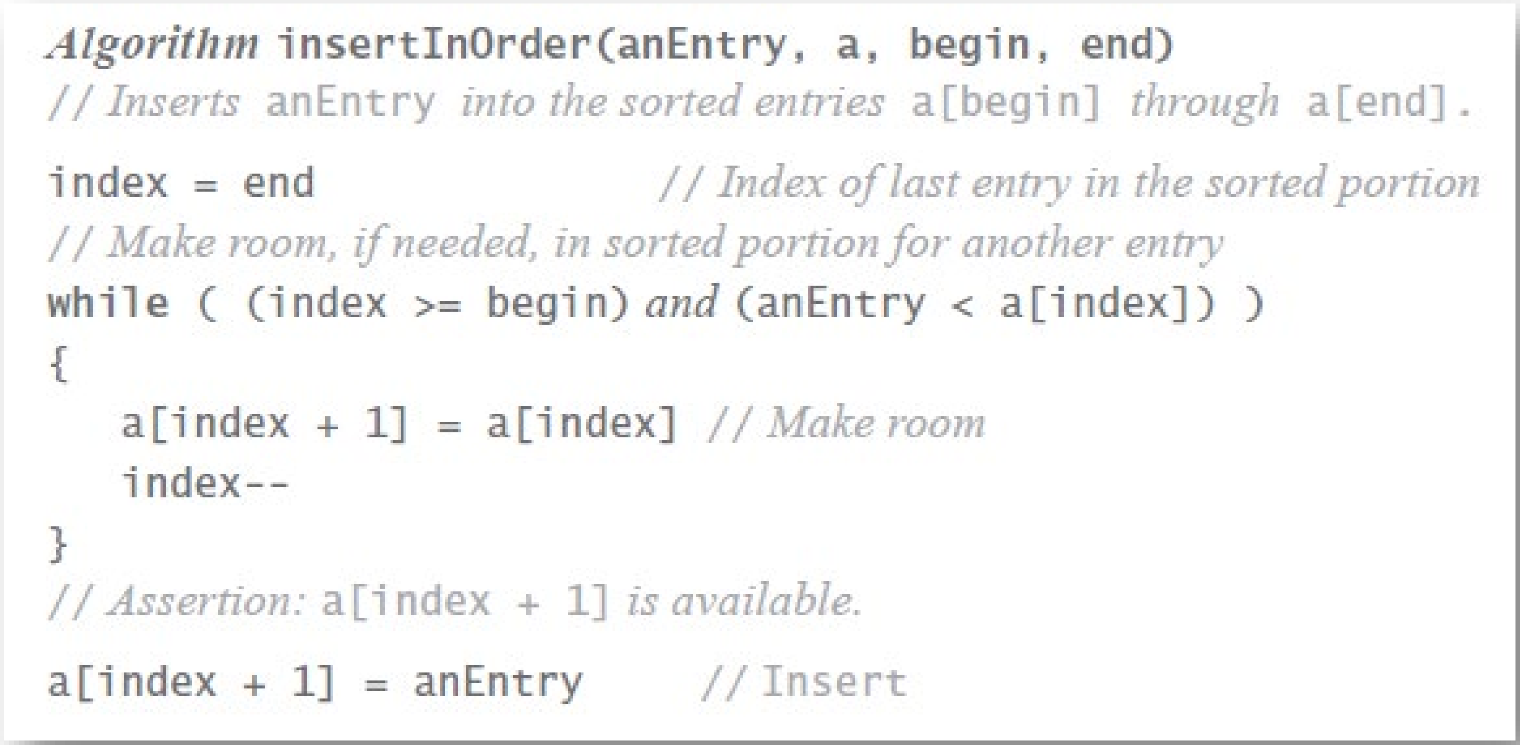
- Iterative algorithm describes an insertion sort of the entries at indices `first` through `last` of the array `a`



```
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

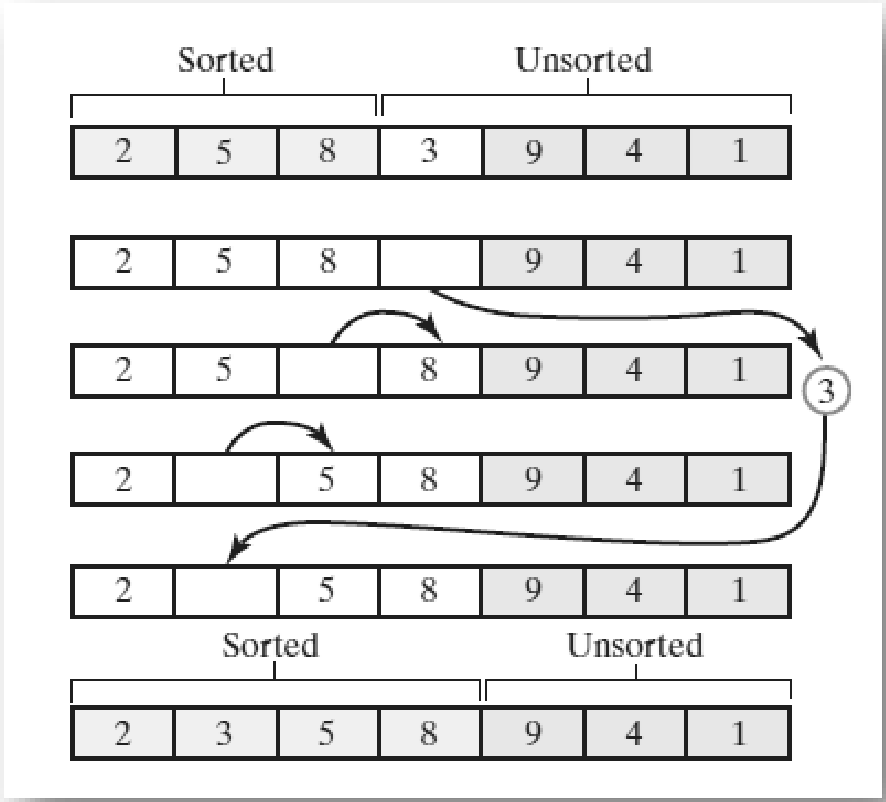
- Pseudocode of method, **`insertInOrder`**, to perform the insertions.



```
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
```

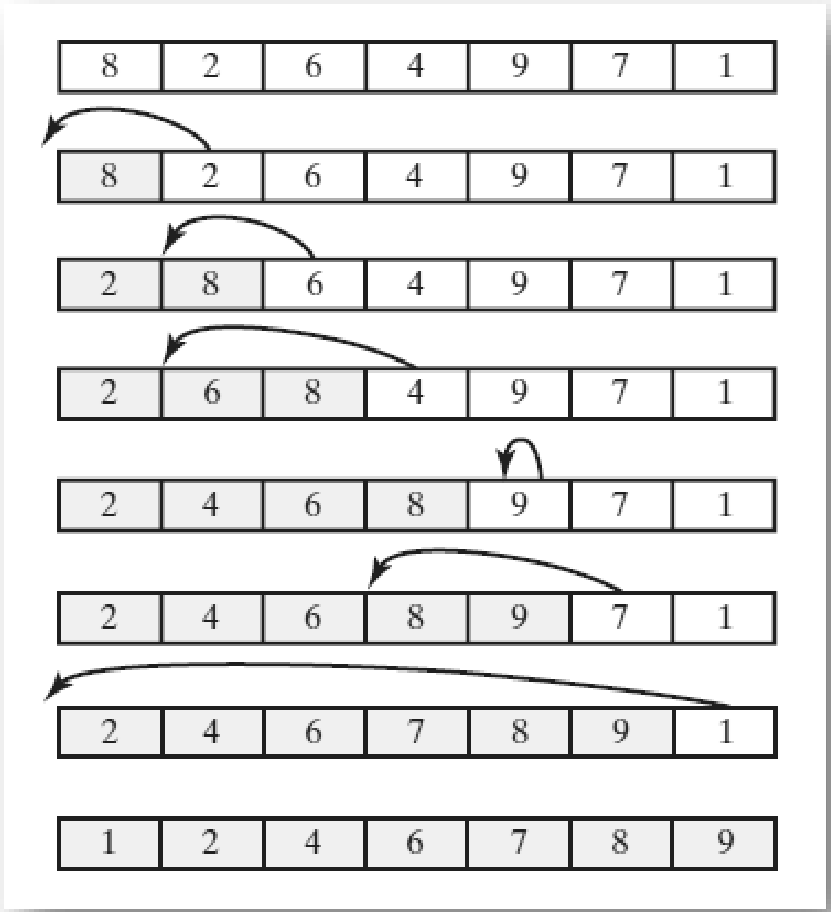
Iterative Insertion Sort

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



Iterative Insertion Sort

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



8 2 6 4 9 7 1
8 2 6 4 9 7 1
2 8 6 4 9 7 1
2 6 8 4 9 7 1
2 4 6 8 9 7 1
2 4 6 8 9 7 1
2 4 6 7 8 9 1
1 2 4 6 7 8 9