Data Structures & Algorithms

Adil M. Khan
Professor of Computer Science
Innopolis University

Six ways to make people like you – Dale Carnegie

1. Become genuinely interested in other people.

Recap

- Algorithmic Strategies
  - Brute-Force
  - Divide-and-Conquer
  - Dynamic Programming
  - Greedy Algorithms

Objectives

- What is sorting?
- Why must one learn about sorting algorithms in this course?
- Properties of sorting algorithms
- Sorting Algorithms
    - ❖ Bubble Sort
    - ❖ Selection Sort
    - ❖ Insertion Sort

Sorting

- Arranging items of the same kind, class or nature, in some ordered sequence
- Sorting Algorithm: an algorithm that arranges elements of a collection in a certain order
- The sorting problem has attracted a great deal of research, perhaps due to the ***complexity of solving it efficiently*** despite its simple statement

Reasons to Study Sorting Algorithms

- Almost all of the ideas used in design of algorithms appear in the context of sorting
    - ❖ Time Analysis, Algorithmic Strategies, Data Structures
- Computers have spent and will keep spending more time sorting than doing anything else
- Most thoroughly studied problem in computer science

Applications

- Punch Line: $O(n \log n)$
- So many important algorithms can be reduced to sorting
  - Searching
  - Closest pair
  - Element uniqueness
  - Frequency distribution

Sorting lies at the heart of many algorithms. Sorting the data is one of the first things any algorithm designer should try in the quest for efficiency.

Try Yourself!

- Punch Line: $O(n \log n)$

❖ Give an algorithm to determine whether two sets (of size m and n, respectively) are ***disjoint***.

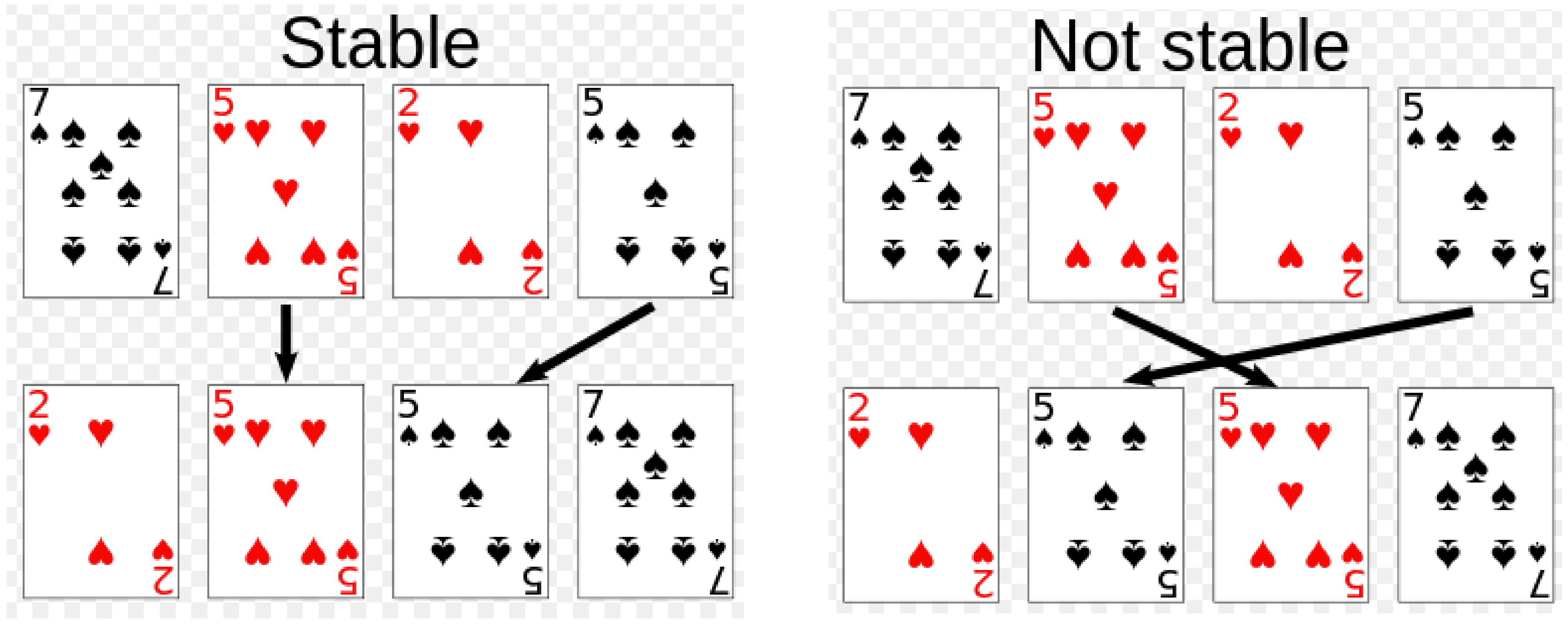
Sorting lies at the heart of many algorithms. Sorting the data is one of the first things any algorithm designer should try in the quest for efficiency.

Sorting Algorithms

- Many ways to classify sorting algorithms
  - ❖ Time vs. Space Complexity
  - ❖ Whether it works by comparing items or not
  - ❖ Stable vs. Unstable
  - ❖ In place sorting or not!

Stability

- Stable sort is one which preserves the original order of the input sent whenever it encounters items of the same rank it



http://en.wikipedia.org/wiki/Sorting_algorithm#/media/File:Sorting_stability_playing_cards.svg

Sorting Algorithms

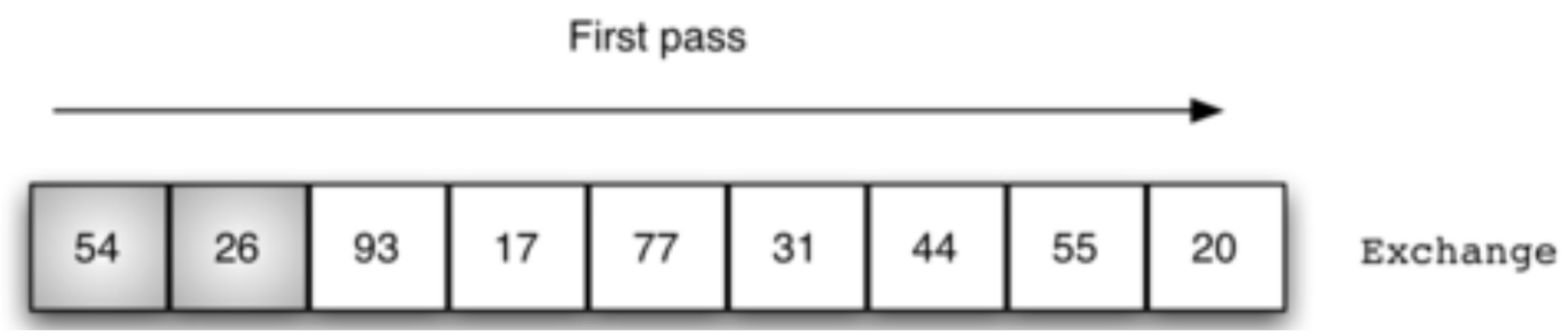
- **Bubble Sort**
- **Selection Sort**
- **Insertion Sort**
- Merge Sort
- Heap Sort
- Quick Sort
- …

Bubble Sort

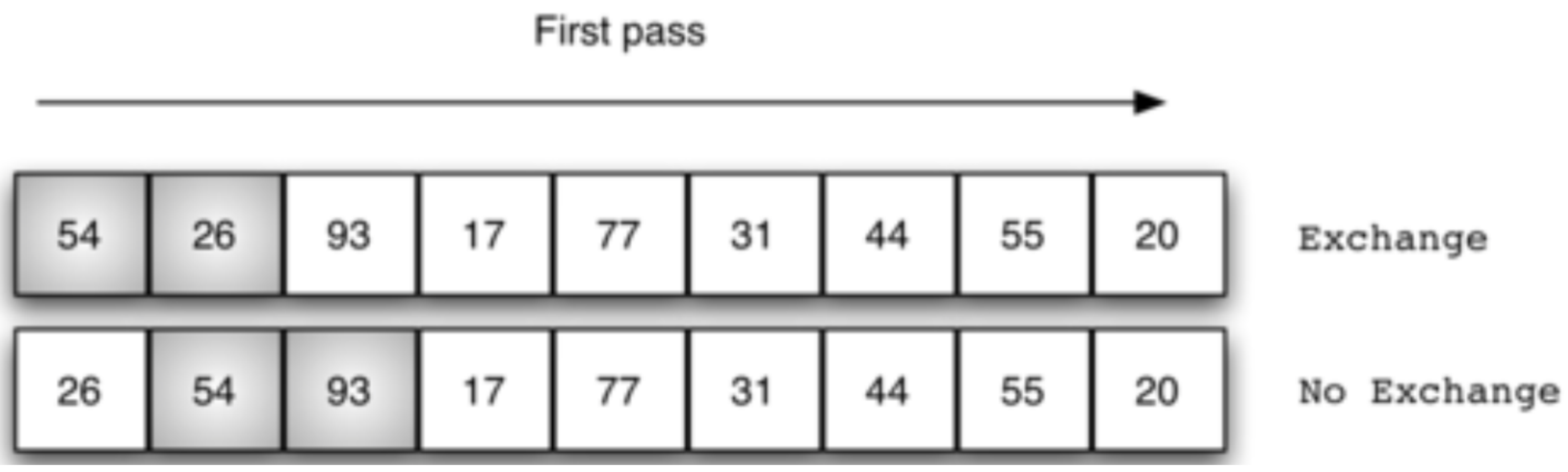
- Systematically examines all pairs of adjacent elements, swapping them when they are out of order.
- Simple rules
  1. Start from the left most position
  2. Compare two adjacent keys
  3. If one on the left is bigger, swap the keys
  4. Move on to the next key

As elements are sorted they gradually "bubble" (or rise) to their proper location in the array, like bubbles rising in a glass of soda

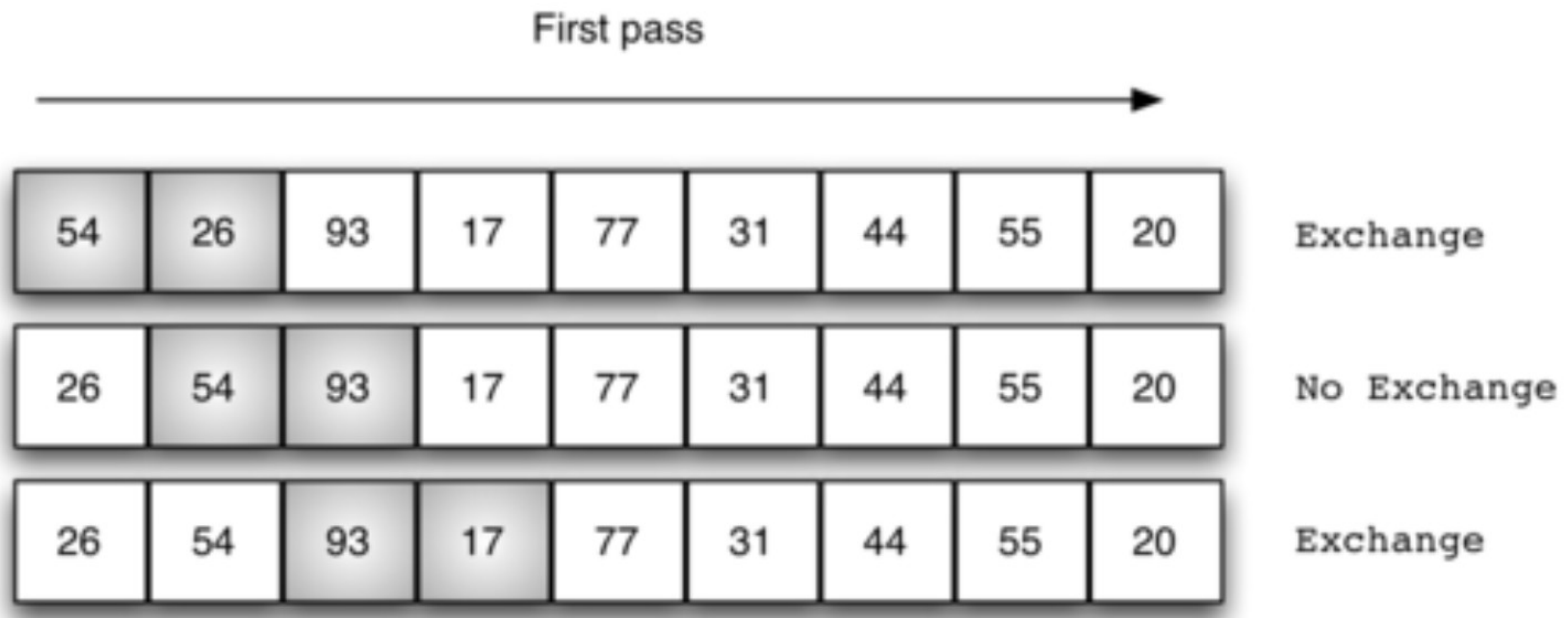
Bubble Sort



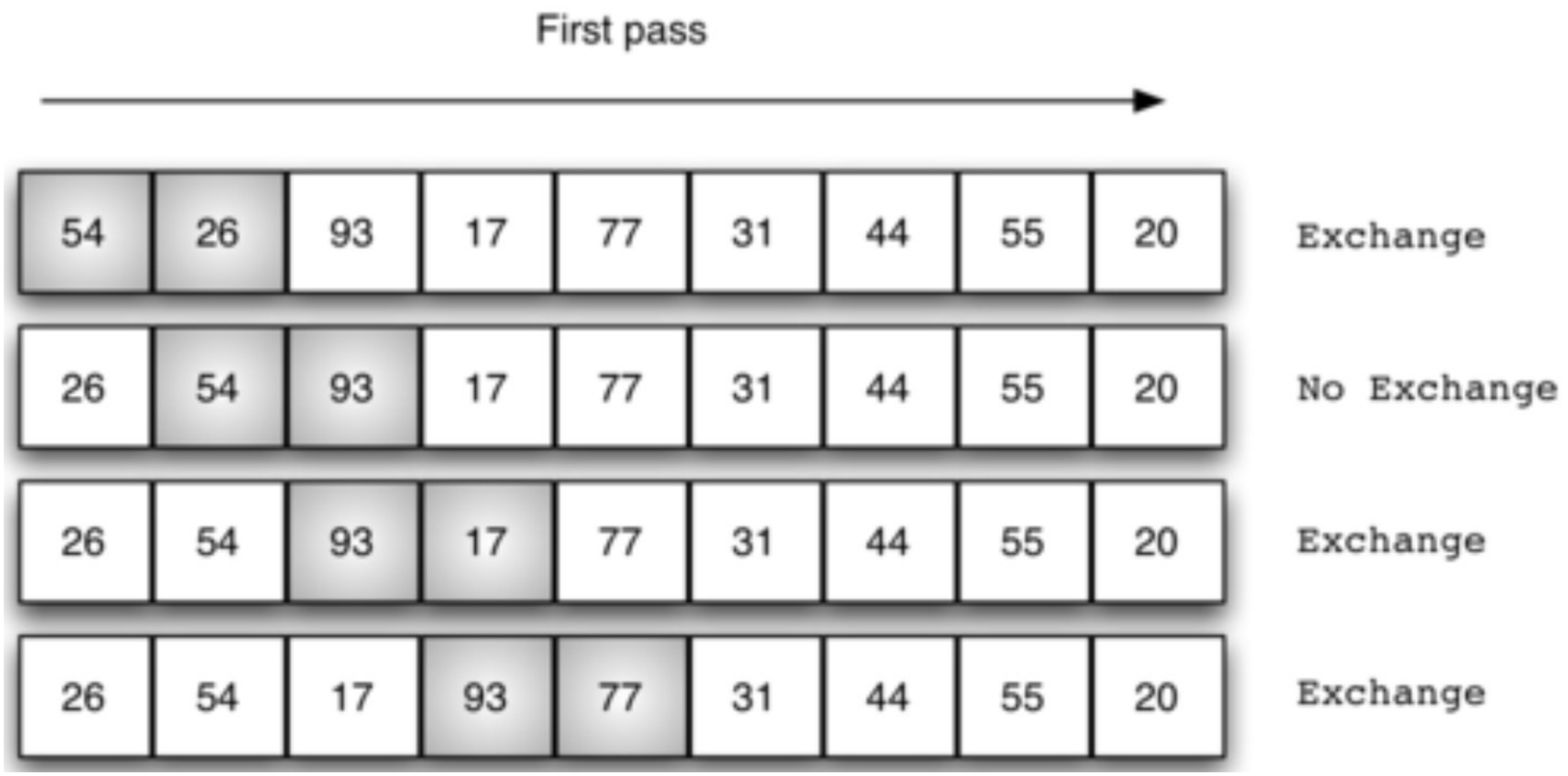
Bubble Sort


First pass
54
26
93
17
77
31
44
55
20
Exchange
26
54
93
17
77
31
44
55
20
No Exchange

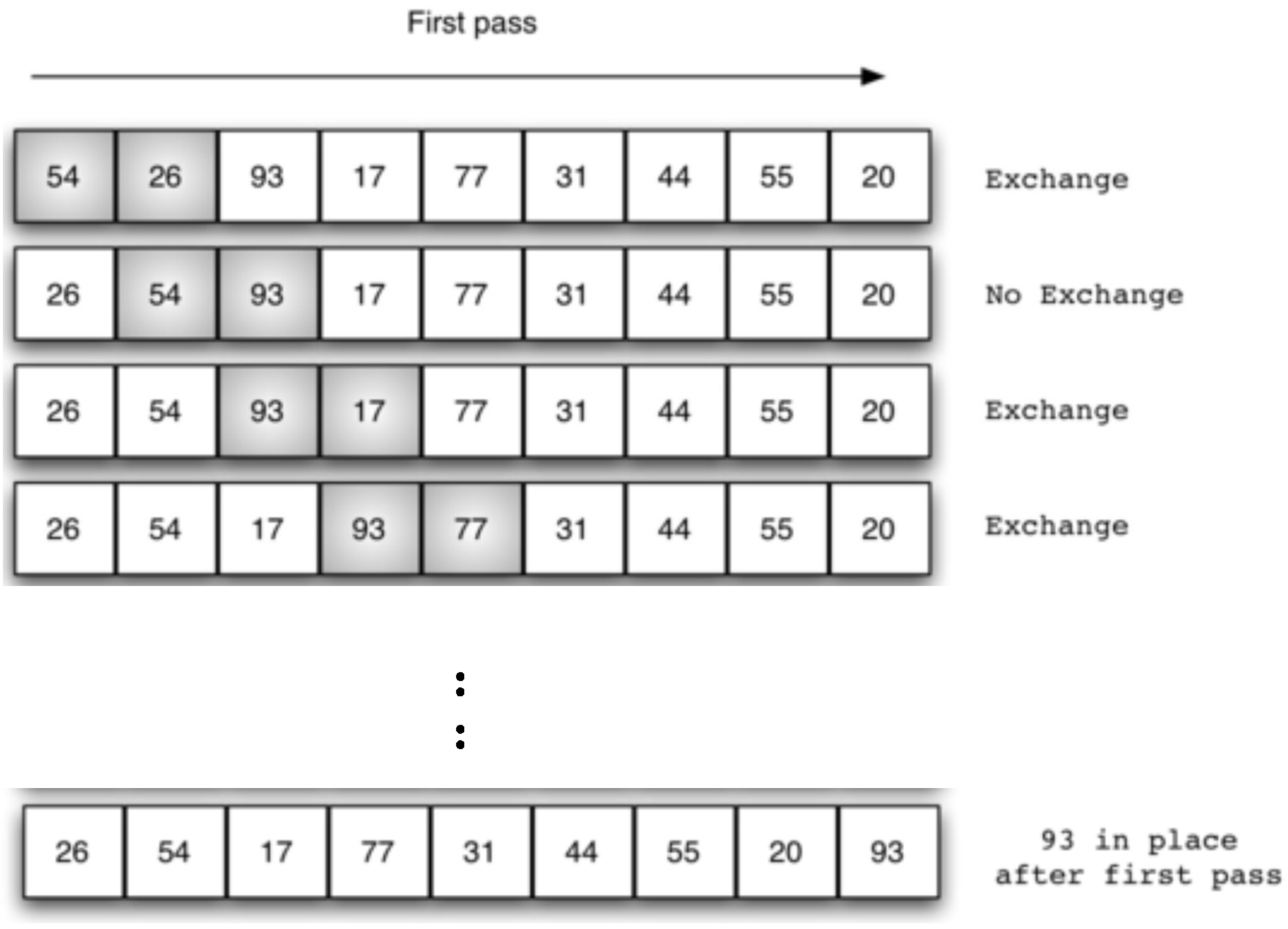
Bubble Sort


First pass
54 26 93 17 77 31 44 55 20 Exchange
26 54 93 17 77 31 44 55 20 No Exchange
26 54 93 17 77 31 44 55 20 Exchange

Bubble Sort


First pass
54 26 93 17 77 31 44 55 20 Exchange
26 54 93 17 77 31 44 55 20 No Exchange
26 54 93 17 77 31 44 55 20 Exchange
26 54 17 93 77 31 44 55 20 Exchange

Bubble Sort

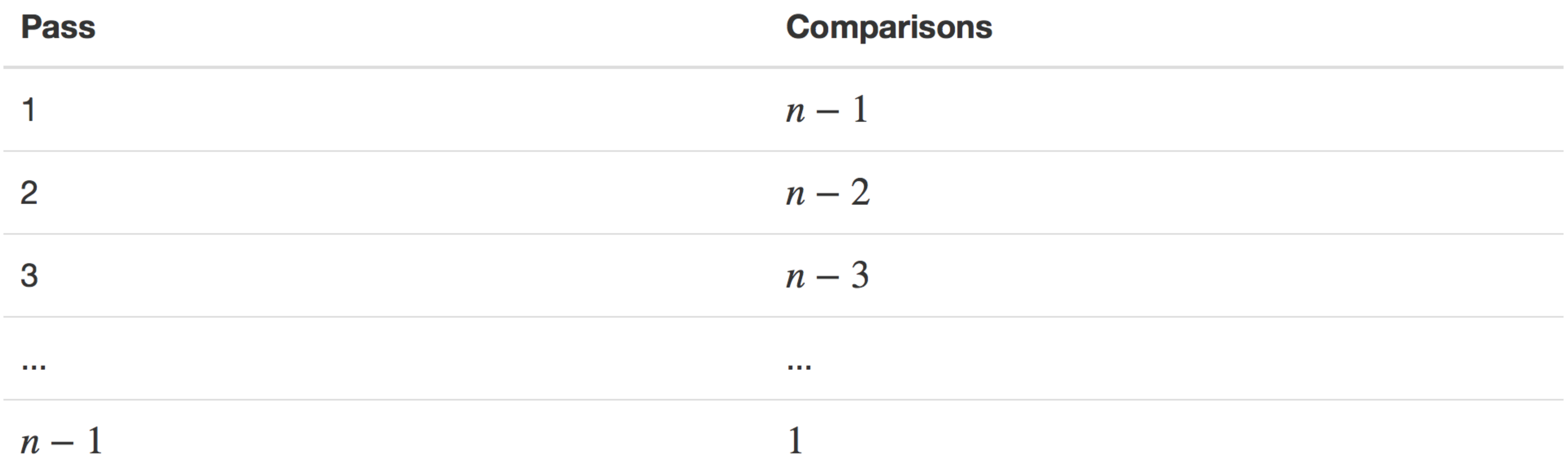

First pass
54 26 93 17 77 31 44 55 20
Exchange
26 54 93 17 77 31 44 55 20
No Exchange
26 54 93 17 77 31 44 55 20
Exchange
26 54 17 93 77 31 44 55 20
Exchange
26 54 17 77 31 44 55 20 93
93 in place
after first pass

Bubble Sort

- Time Complexity Analysis
- First, answer these questions:
  - For a sequence of n keys,
    - How many passes would we make?
    - How many comparisons would we make in the first pass?
    - How many comparisons would we make in the last pass?

Bubble Sort

- Time Complexity Analysis



Pass	Comparisons
1	$n-1$
2	$n-2$
3	$n-3$
...	...
$n-1$	1

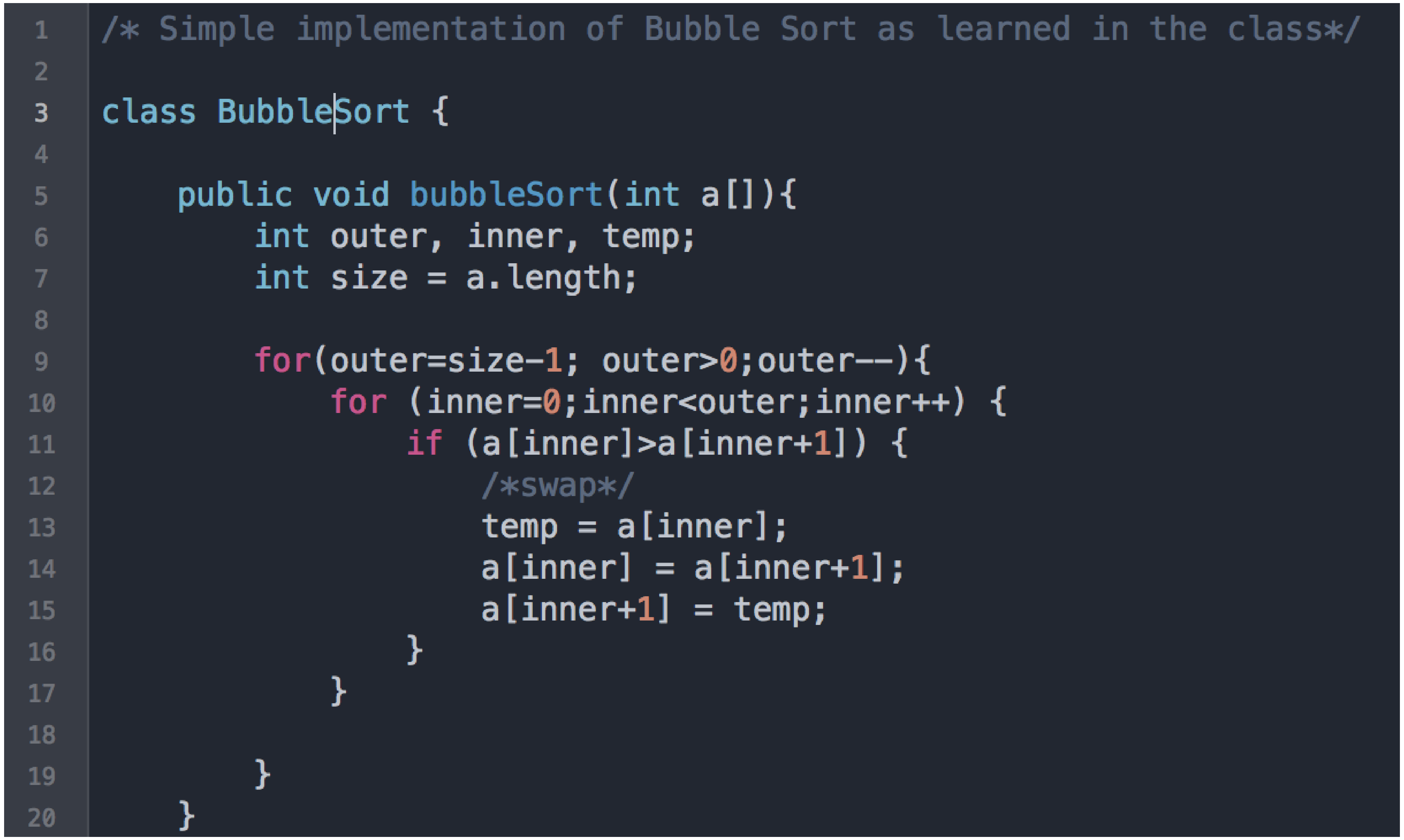
Bubble Sort

- Time Complexity Analysis

Pass	Comparisons
1	$n-1$
2	$n-2$
3	$n-3$
...	...
$n-1$	1

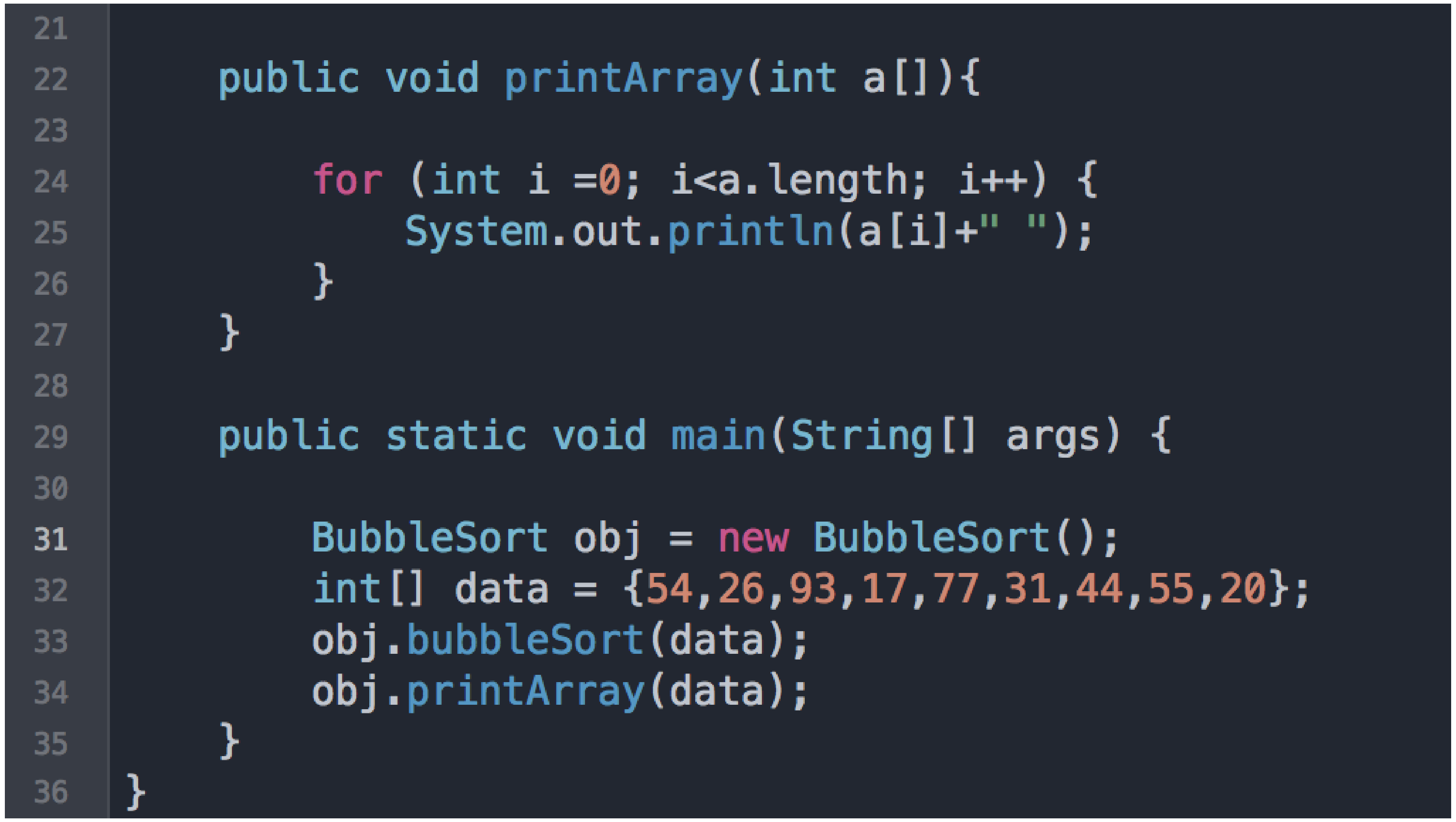
Total Comparisons $= \frac{1}{2}n^2 - \frac{1}{2}n$

Bubble Sort



```
/* Simple implementation of Bubble Sort as learned in the class*/

class BubbleSort {

    public void bubbleSort(int a[]){
        int outer, inner, temp;
        int size = a.length;

        for(outer=size-1; outer>0;outer--){
            for (inner=0;inner<outer;inner++) {
                if (a[inner]>a[inner+1]) {
                    /*swap*/
                    temp = a[inner];
                    a[inner] = a[inner+1];
                    a[inner+1] = temp;
                }
            }

        }
    }
```

Bubble Sort



```

    public void printArray(int a[]){

        for (int i =0; i<a.length; i++) {
            System.out.println(a[i]+" ");
        }
    }

    public static void main(String[] args) {

        BubbleSort obj = new BubbleSort();
        int[] data = {54,26,93,17,77,31,44,55,20};
        obj.bubbleSort(data);
        obj.printArray(data);
    }
}
```

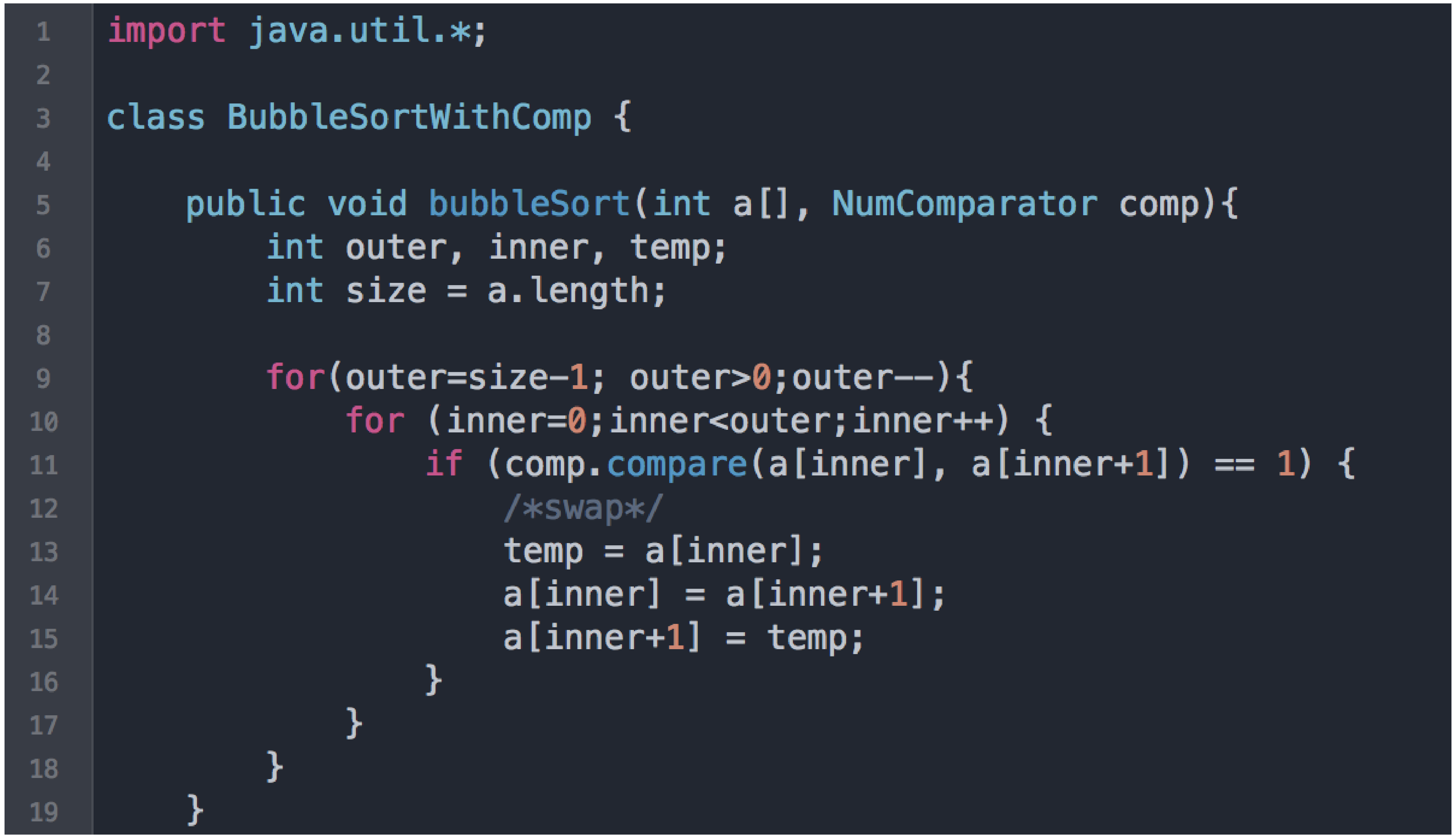
Bubble Sort

- A few observations
  - We don’t usually sort numbers; we usually sort records with keys
    - the key can be a number
    - or the key could be a string

Bubble Sort with Comparator

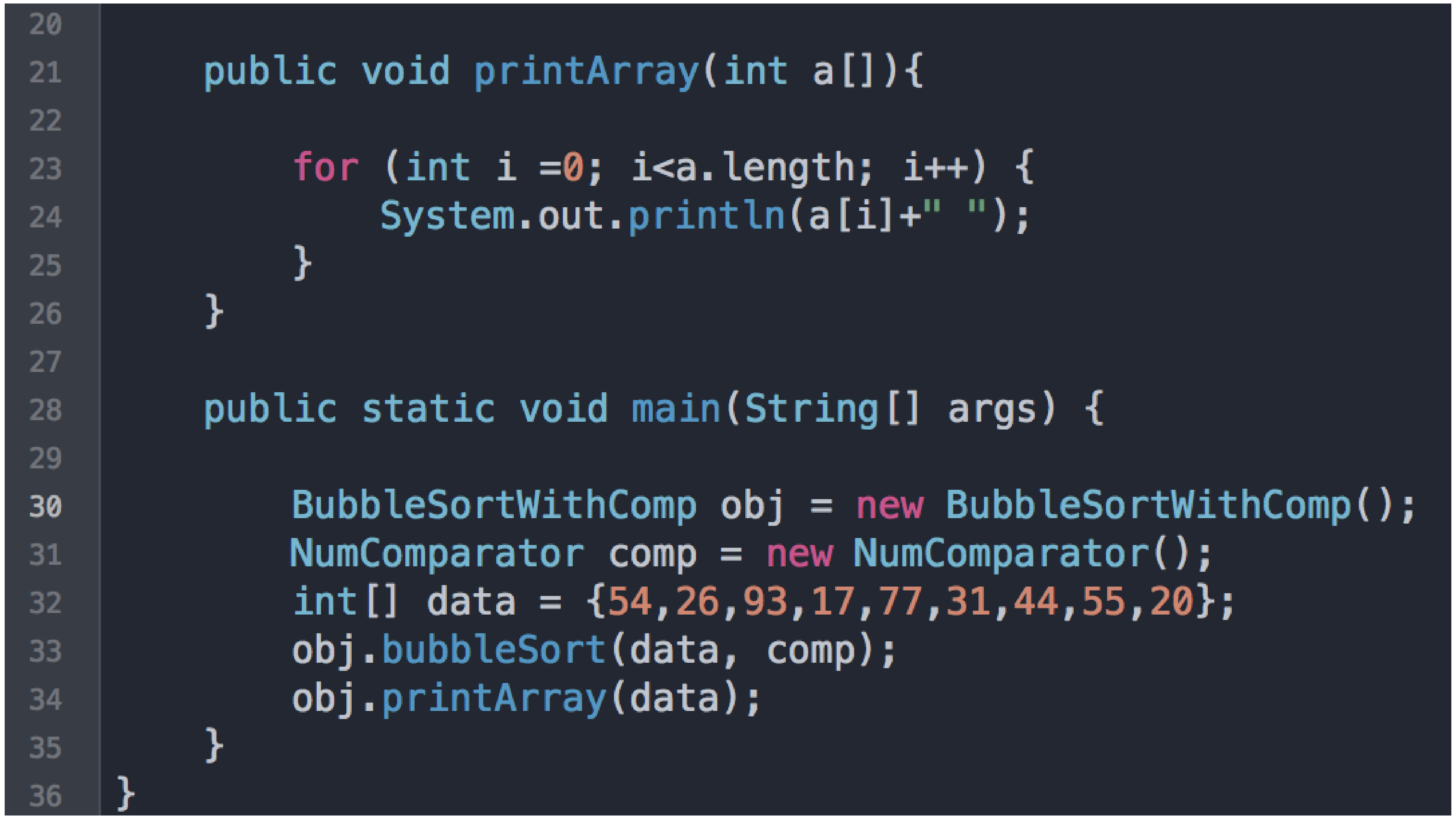
- Comparator<T>:
  - An interface in java
  - imposes a total ordering on some objects using the ***compare*** function
- Comparators can be passed to a sort method

Bubble Sort with Comparator



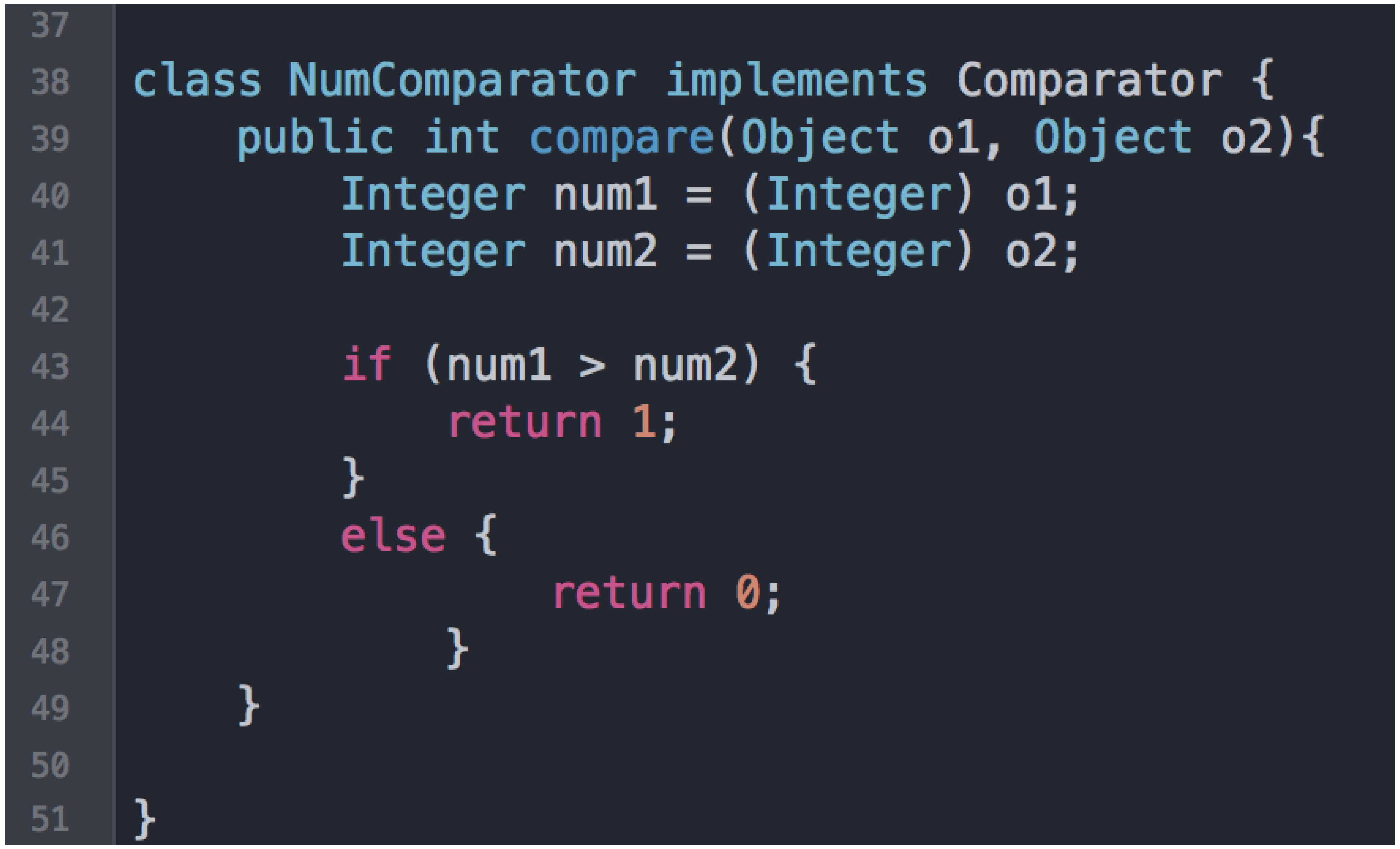
```java
import java.util.*;

class BubbleSortWithComp {

    public void bubbleSort(int a[], NumComparator comp){
        int outer, inner, temp;
        int size = a.length;

        for(outer=size-1; outer>0;outer--){
            for (inner=0;inner<outer;inner++) {
                if (comp.compare(a[inner], a[inner+1]) == 1) {
                    /*swap*/
                    temp = a[inner];
                    a[inner] = a[inner+1];
                    a[inner+1] = temp;
                }
            }
        }
    }
```

Bubble Sort with Comparator



```java

    public void printArray(int a[]){

        for (int i =0; i<a.length; i++) {
            System.out.println(a[i]+" ");
        }
    }

    public static void main(String[] args) {

        BubbleSortWithComp obj = new BubbleSortWithComp();
        NumComparator comp = new NumComparator();
        int[] data = {54,26,93,17,77,31,44,55,20};
        obj.bubbleSort(data, comp);
        obj.printArray(data);
    }
}
```

Bubble Sort with Comparator

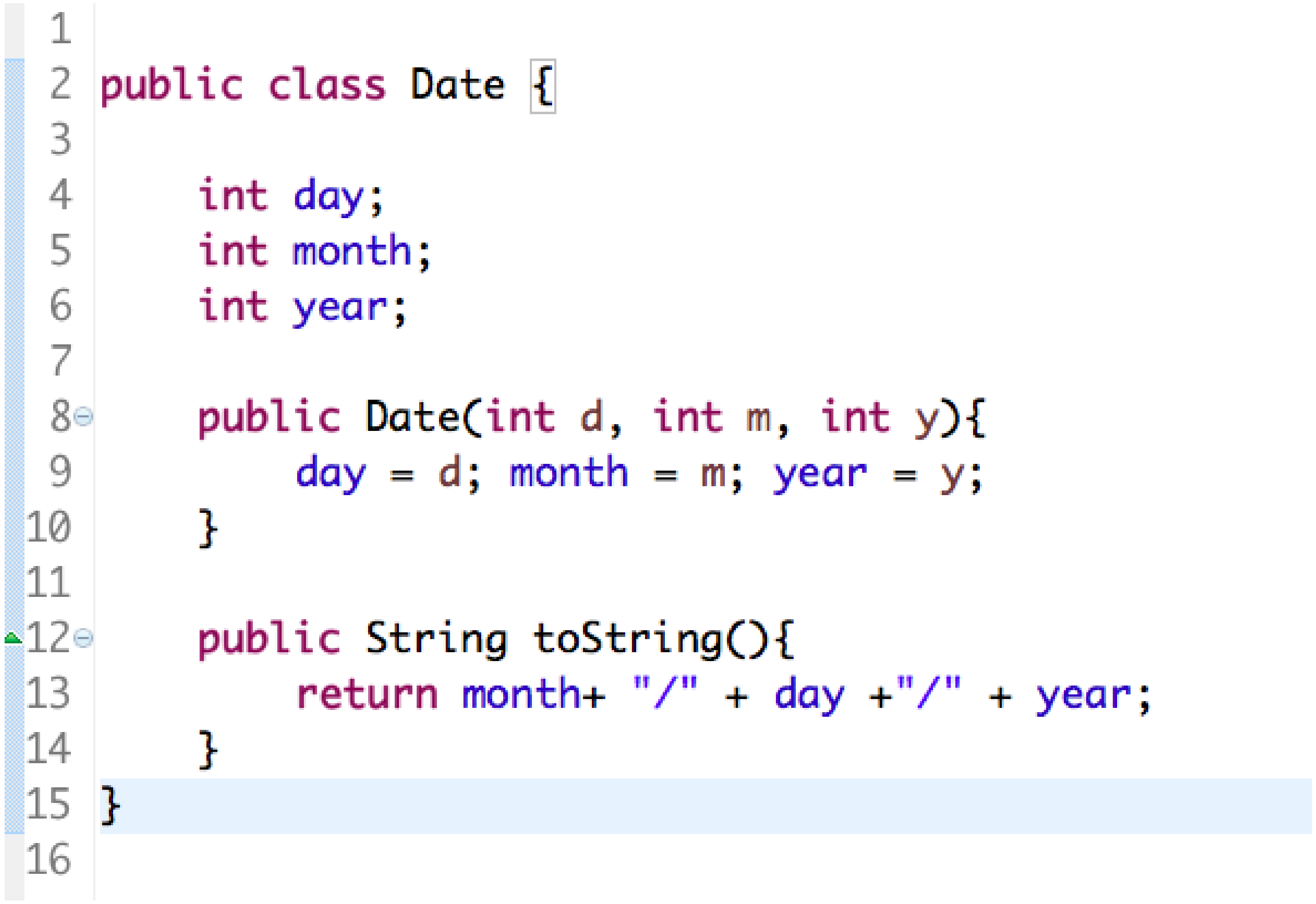


```java

class NumComparator implements Comparator {
    public int compare(Object o1, Object o2){
        Integer num1 = (Integer) o1;
        Integer num2 = (Integer) o2;

        if (num1 > num2) {
            return 1;
        }
        else {
                return 0;
            }
    }

}
```

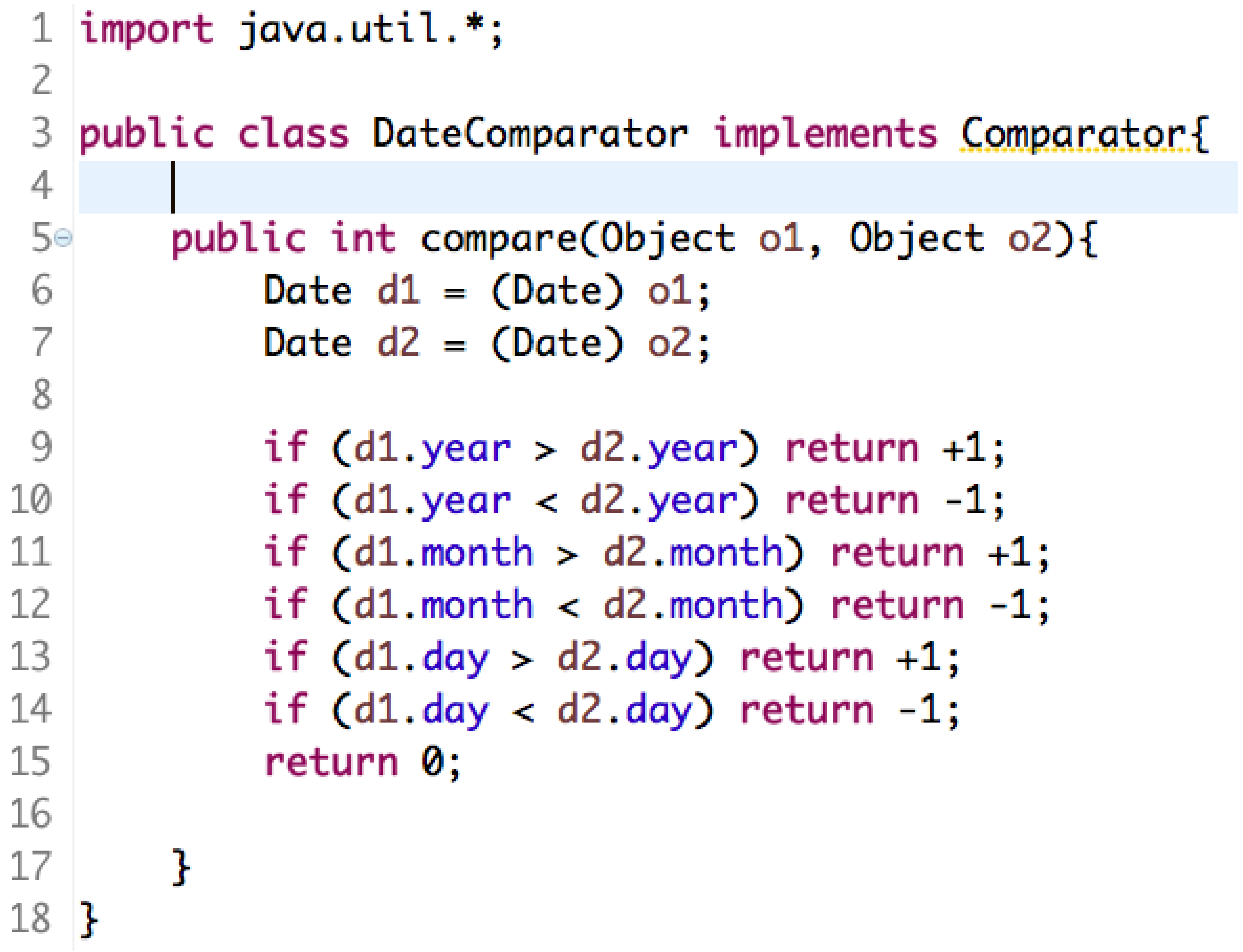
Bubble Sort with Comparator

- What if we are interested in sorting a list something other than integers, for example: dates

- What will the ***DateComparator*** be like?



```

public class Date {

    int day;
    int month;
    int year;

    public Date(int d, int m, int y){
        day = d; month = m; year = y;
    }

    public String toString(){
        return month+ "/" + day +"/" + year;
    }
}

```



```java
import java.util.*;

public class DateComparator implements Comparator{
	
	public int compare(Object o1, Object o2){
		Date d1 = (Date) o1;
		Date d2 = (Date) o2;

		if (d1.year > d2.year) return +1;
		if (d1.year < d2.year) return -1;
		if (d1.month > d2.month) return +1;
		if (d1.month < d2.month) return -1;
		if (d1.day > d2.day) return +1;
		if (d1.day < d2.day) return -1;
		return 0;

	}
}
```

Bubble Sort

- Exercise: Implement the bubble sort with the driver program
  - the original bubble sort
  - the bubble sort with the comparator for numbers
  - the bubble sort with the comparator for string
    - ascending order
    - descending order
  - Compute the time complexity of the bubble sort
  - Is Bubble sort stable or unstable? Why?

Selection Sort

- A combination of searching and sorting
- During each pass, the unsorted element with the smallest (or largest) value is moved to its proper position

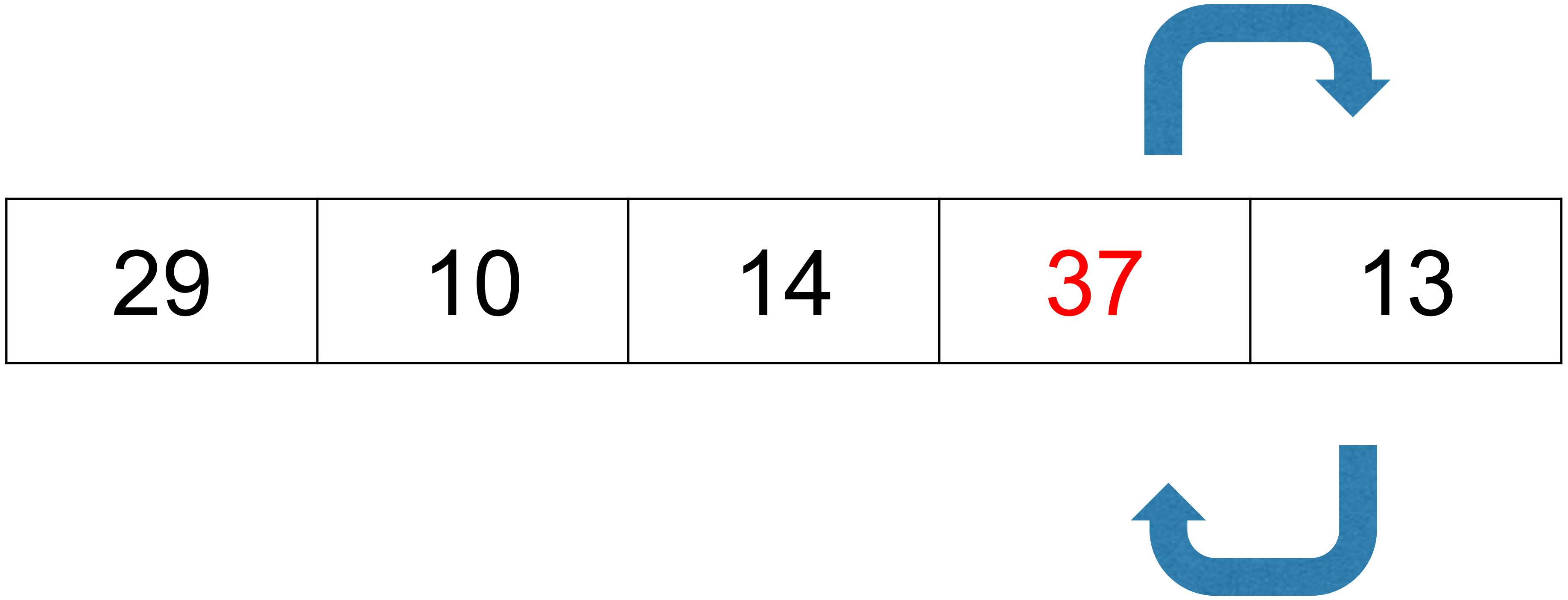
Selection Sort

29	10	14	37	13

Selection Sort

29	10	14	37	13

Selection Sort

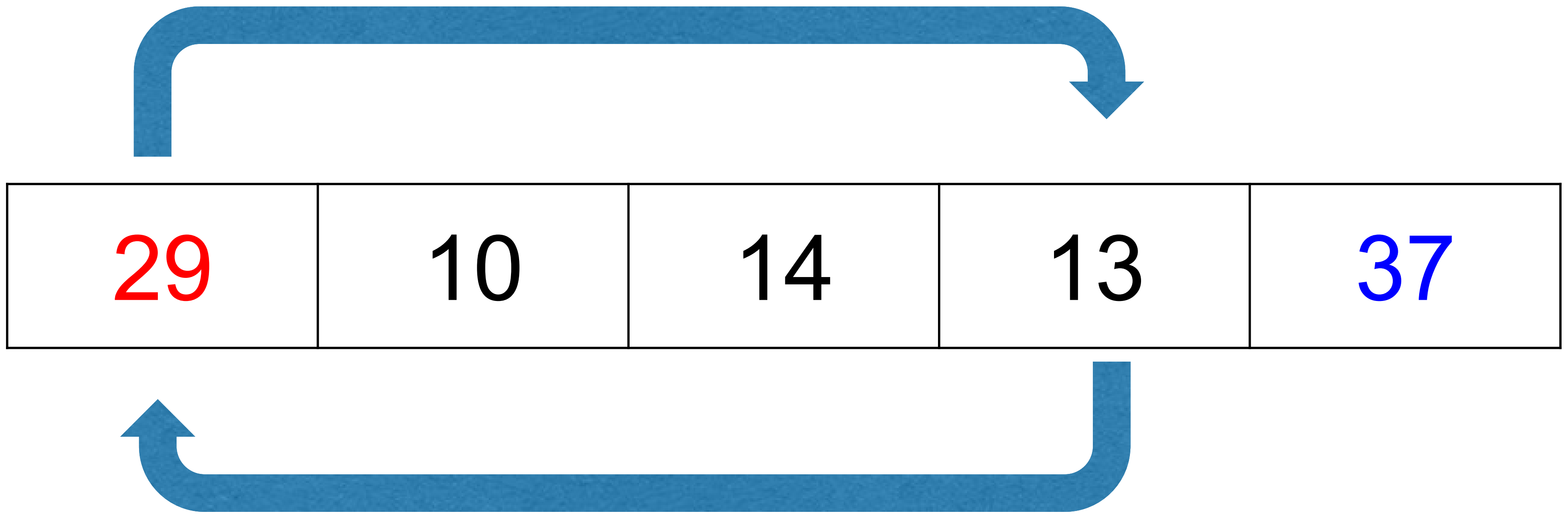


29	10	14	37	13

Selection Sort

29	10	14	13	37

Selection Sort



Selection Sort

13	10	14	29	37

Selection Sort

13	10	14	29	37

Selection Sort

13	10	14	29	37

Selection Sort

13	10	14	29	37

Selection Sort

10	13	14	29	37

Selection Sort

10	13	14	29	37

Selection Sort

10	13	14	29	37

Selection Sort

- Assume we are sorting a list represented by array A of n integers

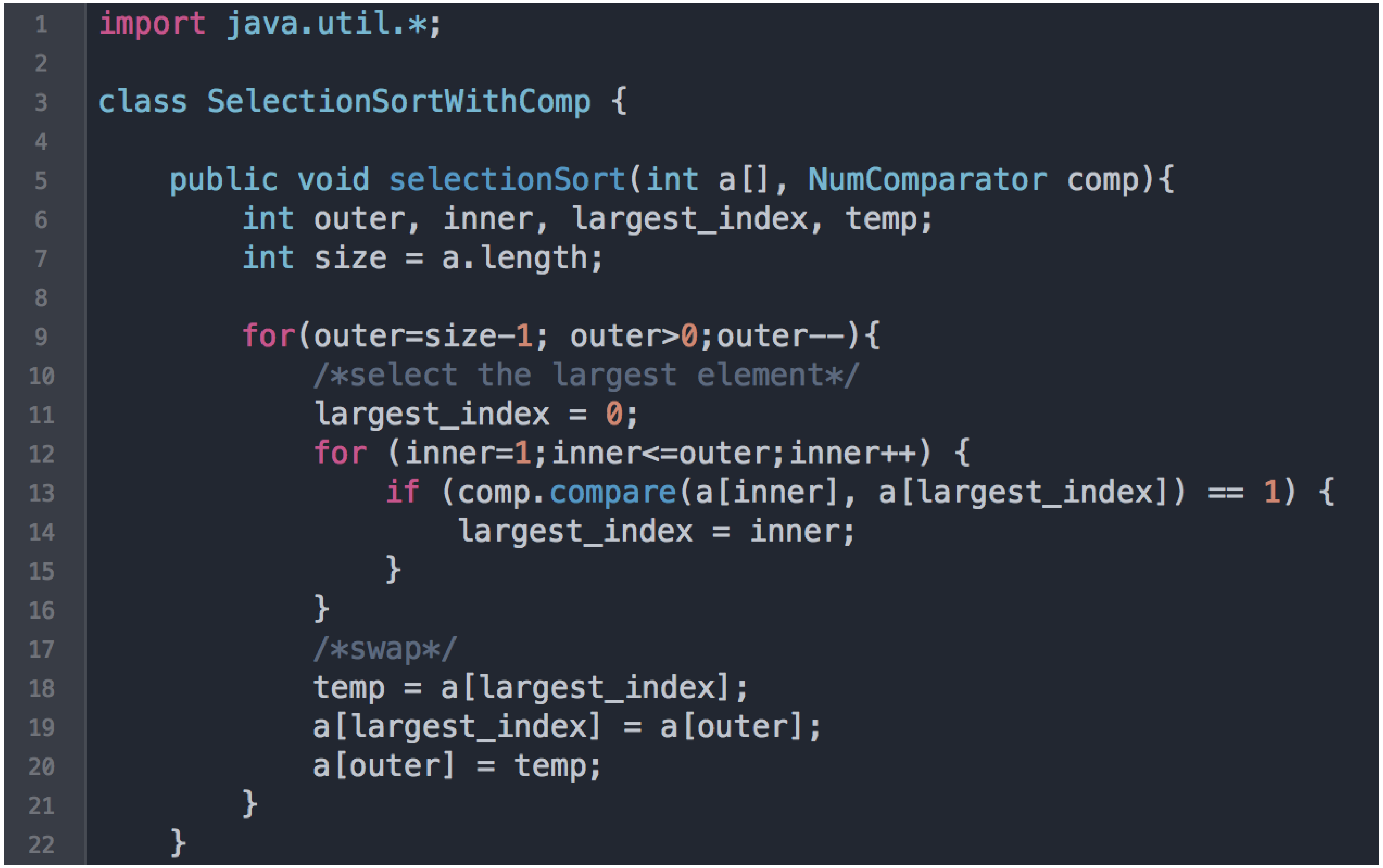
```
last = n-1
 Do
    Select largest element from a[0..last]
    Swap it with a[last]
    last = last-1
 While (last >= 1)
```

Selection Sort

- Time Complexity?

```
last = $n$-1
 Do
    Select largest element from a[0..last]
    Swap it with a[last]
    last = last–1
 While (last >= 1)
```

Selection Sort



```java
import java.util.*;

class SelectionSortWithComp {

    public void selectionSort(int a[], NumComparator comp){
        int outer, inner, largest_index, temp;
        int size = a.length;

        for(outer=size-1; outer>0;outer--){
            /*select the largest element*/
            largest_index = 0;
            for (inner=1;inner<=outer;inner++) {
                if (comp.compare(a[inner], a[largest_index]) == 1) {
                    largest_index = inner;
                }
            }
            /*swap*/
            temp = a[largest_index];
            a[largest_index] = a[outer];
            a[outer] = temp;
        }
    }
```

Selection Sort

- Exercise: Implement the selection sort with the driver program
  - the original selection sort without a comparator
  - the selection sort with the comparator for numbers
  - the selection sort with the comparator for string
    - ascending order
    - descending order
  - Compute the time complexity of the selection sort.
  - Is the selection sort stable or unstable? Why?

Insertion Sort

- Commonly compared to organizing a handful of playing cards
  - ❖ You pick up the random cards from the table one at a time
  - ❖ As you pick each card, you insert it into its correct position in your hand or organized (sorted) cards

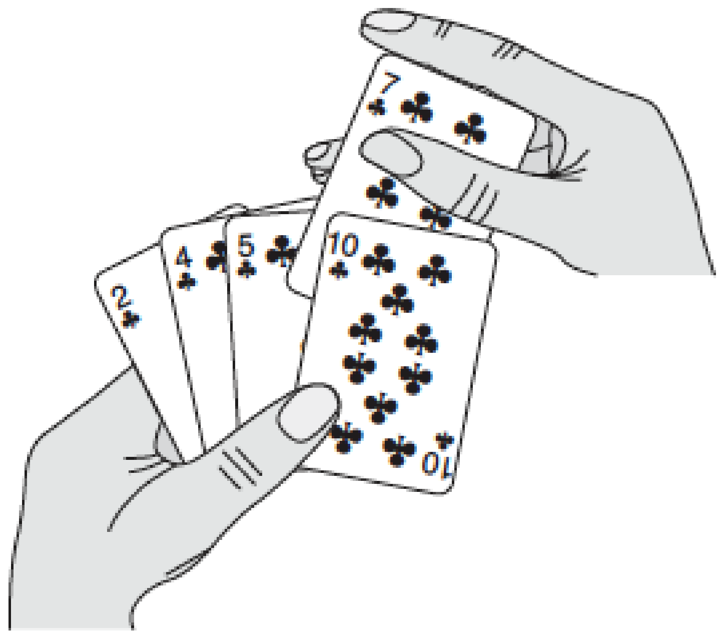


Figure 2.1 Sorting a hand of cards using insertion sort.

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

<u>20</u>	54	16	36	99	11	74	88

Insertion Sort

<u>20</u>	54	16	36	99	11	74	88

Insertion Sort

<u>20</u>	<u>54</u>	16	36	99	11	74	88

Insertion Sort

<u>20</u>	<u>54</u>	16	36	99	11	74	88

Insertion Sort

16	<u>20</u>	<u>54</u>	36	99	11	74	88

Insertion Sort

16	20	54	36	99	11	74	88

Insertion Sort

16	20	54	36	99	11	74	88

Insertion Sort

16	20	36	54	99	11	74	88

Insertion Sort

16	20	36	54	99	11	74	88

Insertion Sort

16	20	36	54	99	11	74	88

Insertion Sort

16	20	36	54	99	11	74	88

.

.

.

11	16	20	36	54	74	88	99

Insertion Sort

```
for i in 1 .. n-1
    current := a[i]
    j := i
    while (j > 0 and current < a[j-1])
        a[j] := a[j-1]
        j--
    a[j] = current
```

Insertion Sort

Time Complexity?

```
for i in 1 .. n-1
    current := a[i]
    j := i
    while (j > 0 and current < a[j-1])
        a[j] := a[j-1]
        j--
    a[j] = current
```

Insertion Sort

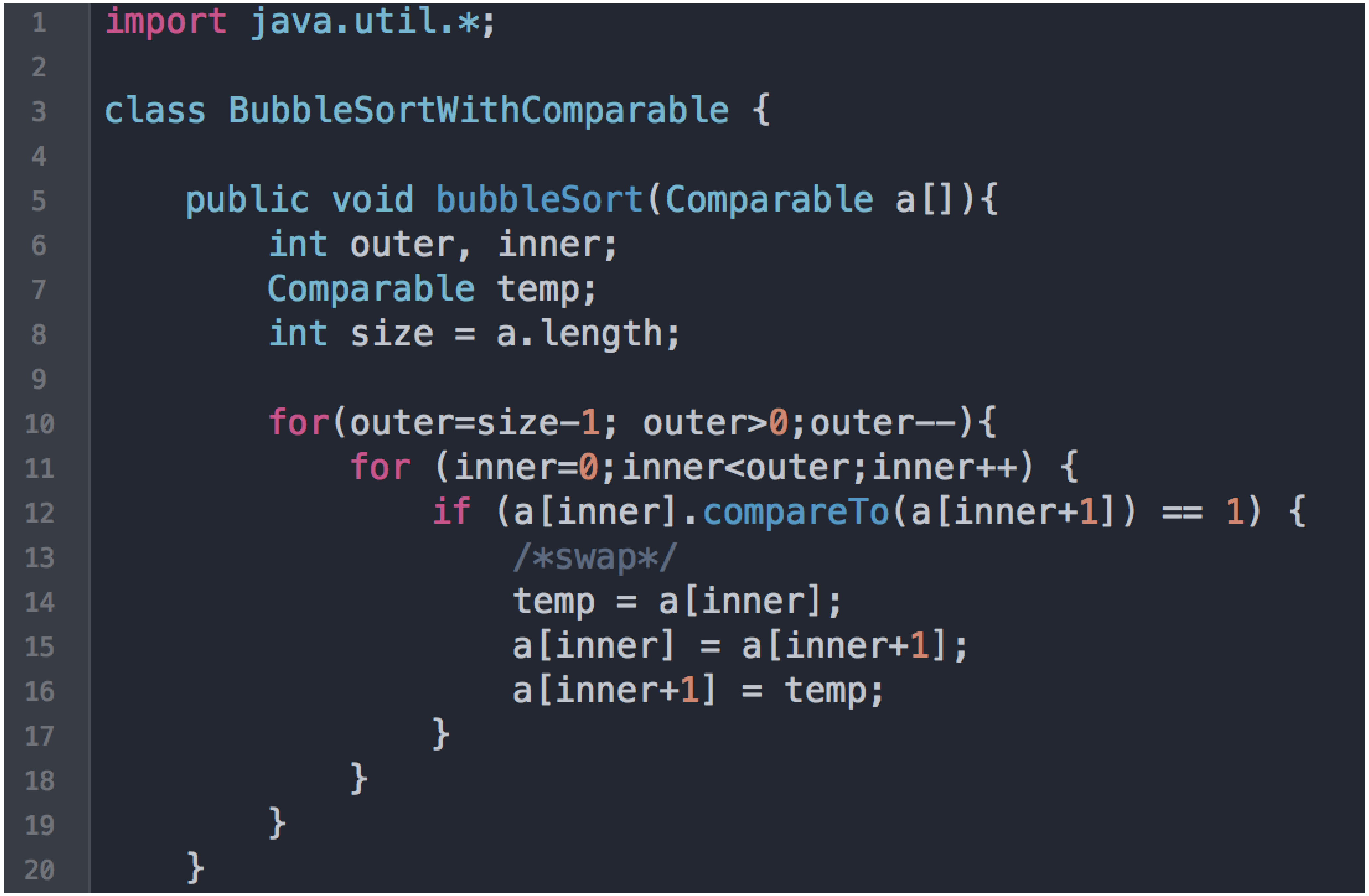
- Quadratic running time on average (linear if data is already sorted)
- After k pass, the first k+1 elements are in relative sorted order
- In practice, does far fewer compares than the selection sort
- But has moves (swaps) in the inner loop
- If data is sorted, this algorithm works best, worst if data is reversed
- Better than selection sort if moves are cheap, worse if expensive

Insertion Sort

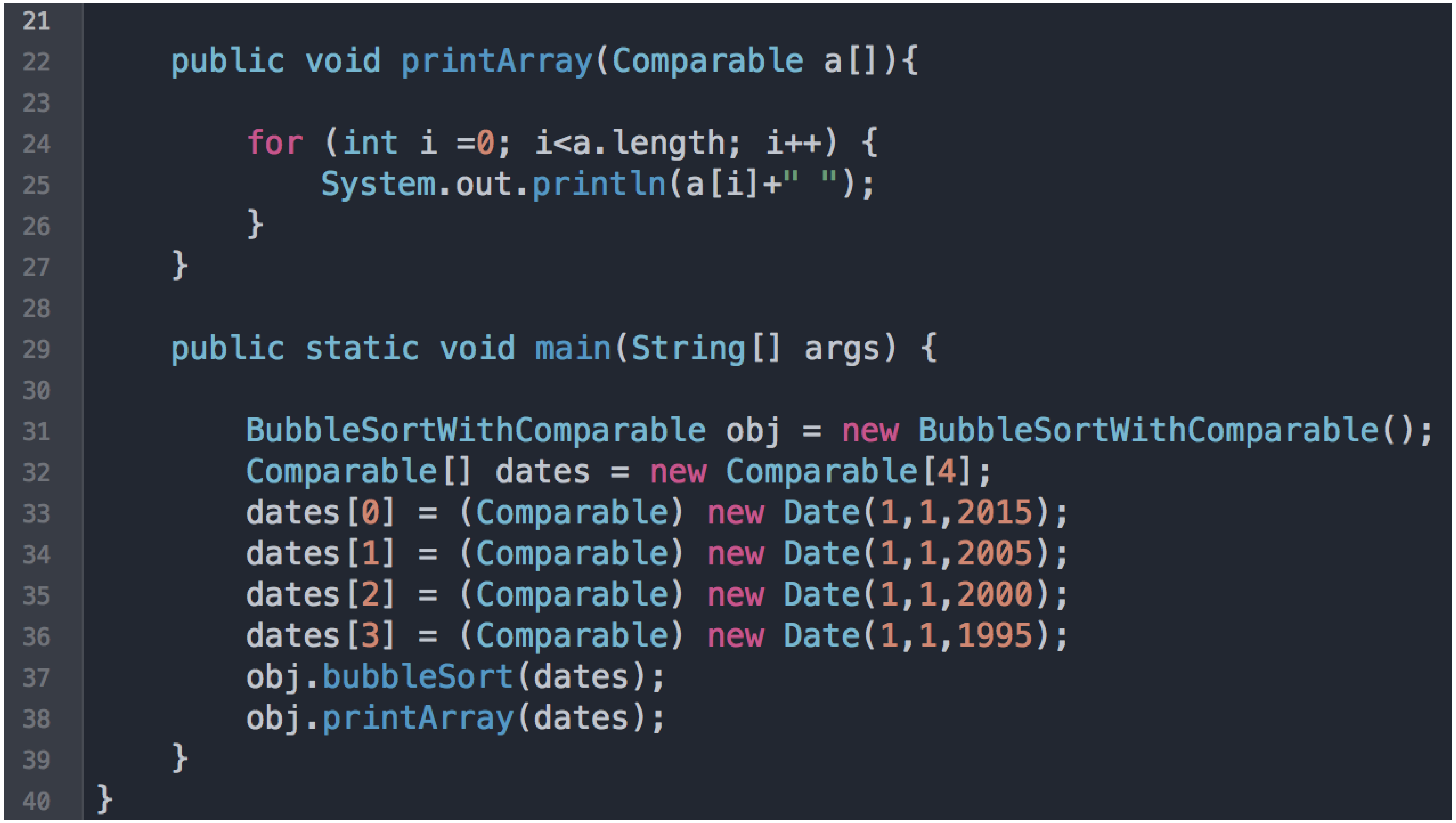
- Exercise: Implement the insertion sort with the driver program
  - the original insertion sort without a comparator
  - the insertion sort with the comparator for numbers
  - the insertion sort with the comparator for string
    - ascending order
    - descending order
  - Compute the time complexity of the insertion sort.
  - Is the insertion sort stable or unstable? Why?

Extra

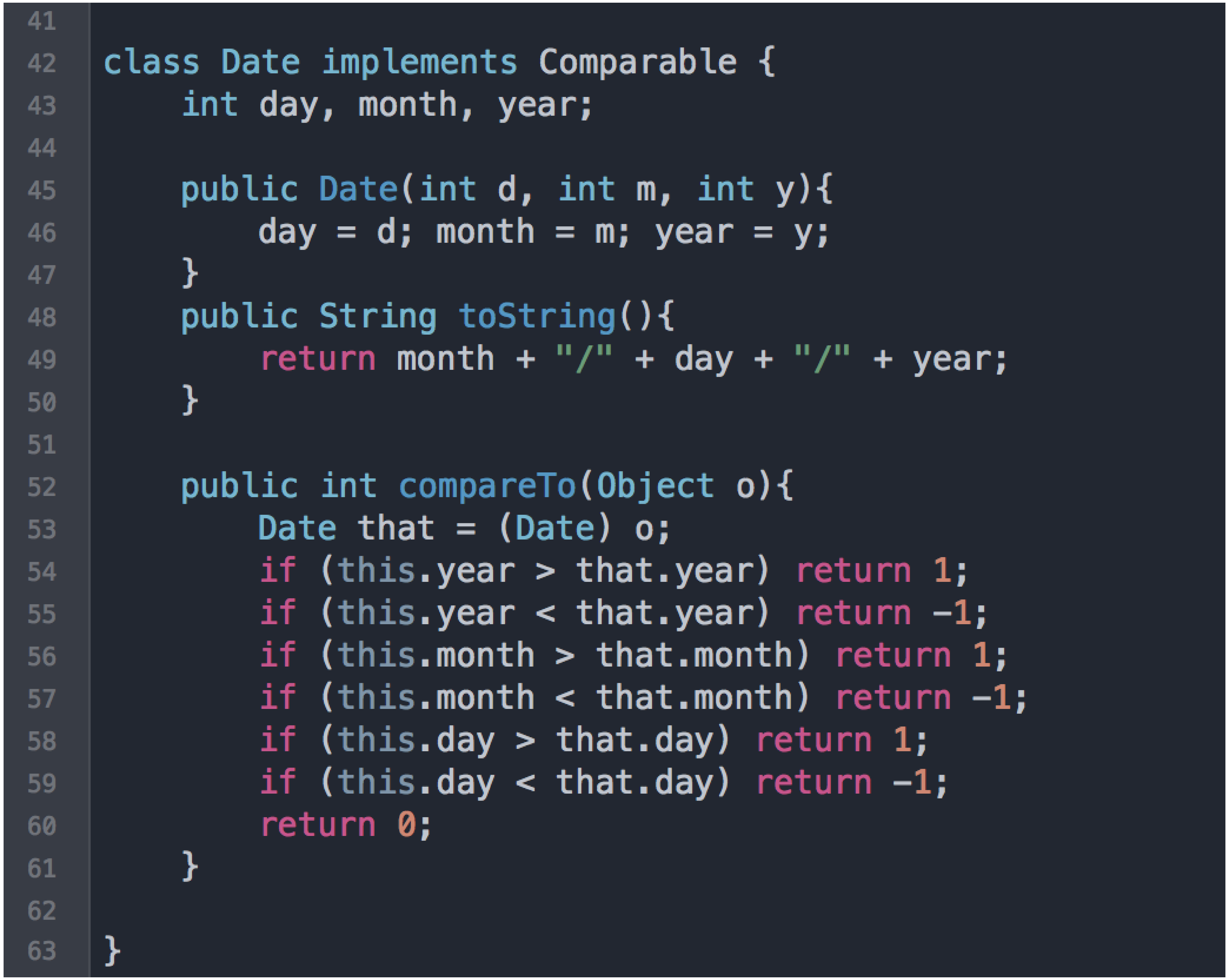
Using ***Comparable***



```java
import java.util.*;

class BubbleSortWithComparable {

    public void bubbleSort(Comparable a[]){
        int outer, inner;
        Comparable temp;
        int size = a.length;

        for(outer=size-1; outer>0;outer--){
            for (inner=0;inner<outer;inner++) {
                if (a[inner].compareTo(a[inner+1]) == 1) {
                    /*swap*/
                    temp = a[inner];
                    a[inner] = a[inner+1];
                    a[inner+1] = temp;
                }
            }
        }
    }
```



```

    public void printArray(Comparable a[]){

        for (int i =0; i<a.length; i++) {
            System.out.println(a[i]+" ");
        }
    }

    public static void main(String[] args) {

        BubbleSortWithComparable obj = new BubbleSortWithComparable();
        Comparable[] dates = new Comparable[4];
        dates[0] = (Comparable) new Date(1,1,2015);
        dates[1] = (Comparable) new Date(1,1,2005);
        dates[2] = (Comparable) new Date(1,1,2000);
        dates[3] = (Comparable) new Date(1,1,1995);
        obj.bubbleSort(dates);
        obj.printArray(dates);
    }
}
```



```java

class Date implements Comparable {
    int day, month, year;

    public Date(int d, int m, int y){
        day = d; month = m; year = y;
    }
    public String toString(){
        return month + "/" + day + "/" + year;
    }

    public int compareTo(Object o){
        Date that = (Date) o;
        if (this.year > that.year) return 1;
        if (this.year < that.year) return -1;
        if (this.month > that.month) return 1;
        if (this.month < that.month) return -1;
        if (this.day > that.day) return 1;
        if (this.day < that.day) return -1;
        return 0;
    }

}
```

Did we achieve today's objectives?

- What is sorting?
- Why must one learn about sorting algorithms in this course?
- Properties of sorting algorithms
- Sorting Algorithms
  - ❖ Bubble Sort
  - ❖ Selection Sort
  - ❖ Insertion Sort