Data Structures & Algorithms

Adil M. Khan
Professor of Computer Science
Innopolis University

Sorting Algorithms-1

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

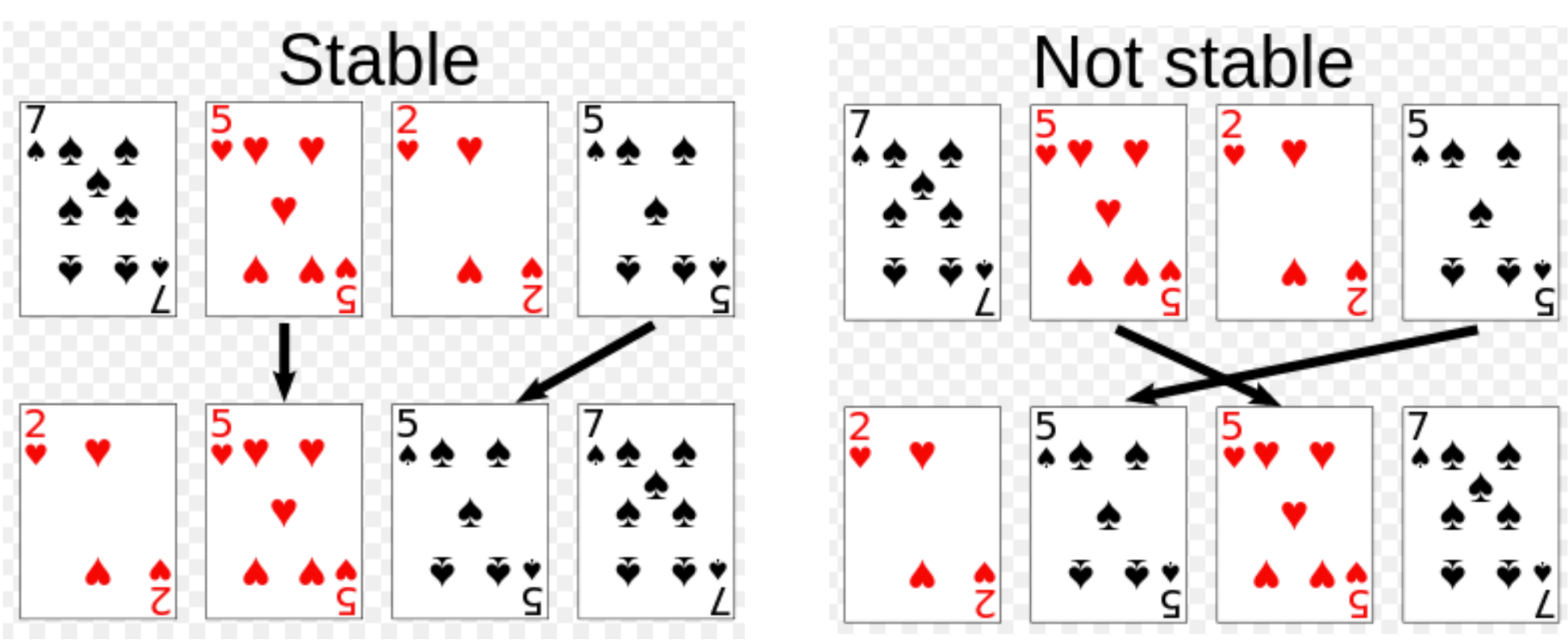
- An exercise of comparing algorithm performance
- Similar techniques are effective in addressing other problems
- Sorting plays a major role in commercial data processing, combinatorial optimization, molecular dynamics, genomics and many other fields

Sorting Algorithms

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

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.

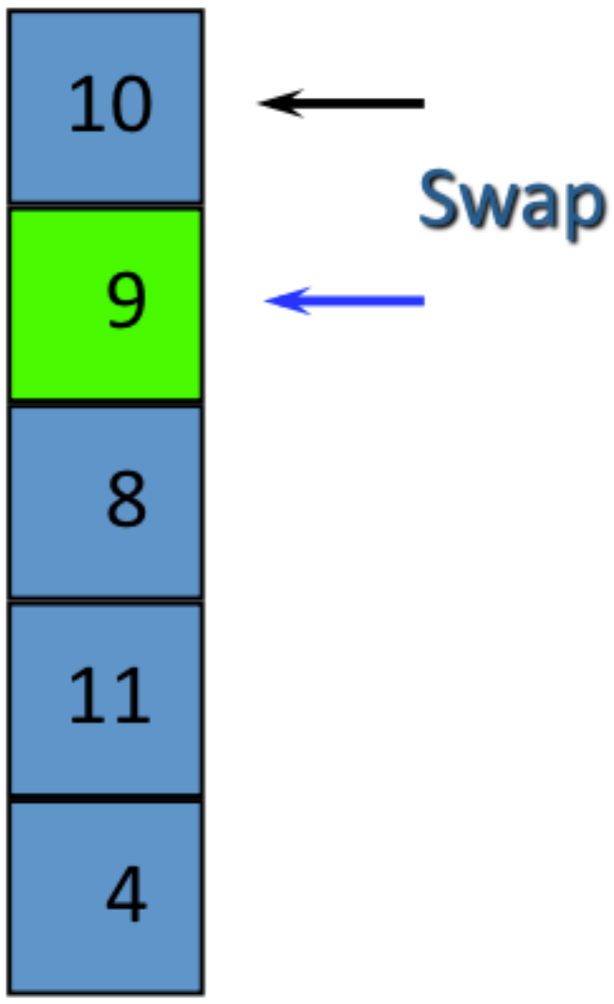
```
FOR every element in the list,
    proceeding for the first to the last

   WHILE list element > previous list element
      bubble element back (up) the list
      by successive swapping with
      the element just above/prior it
```

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

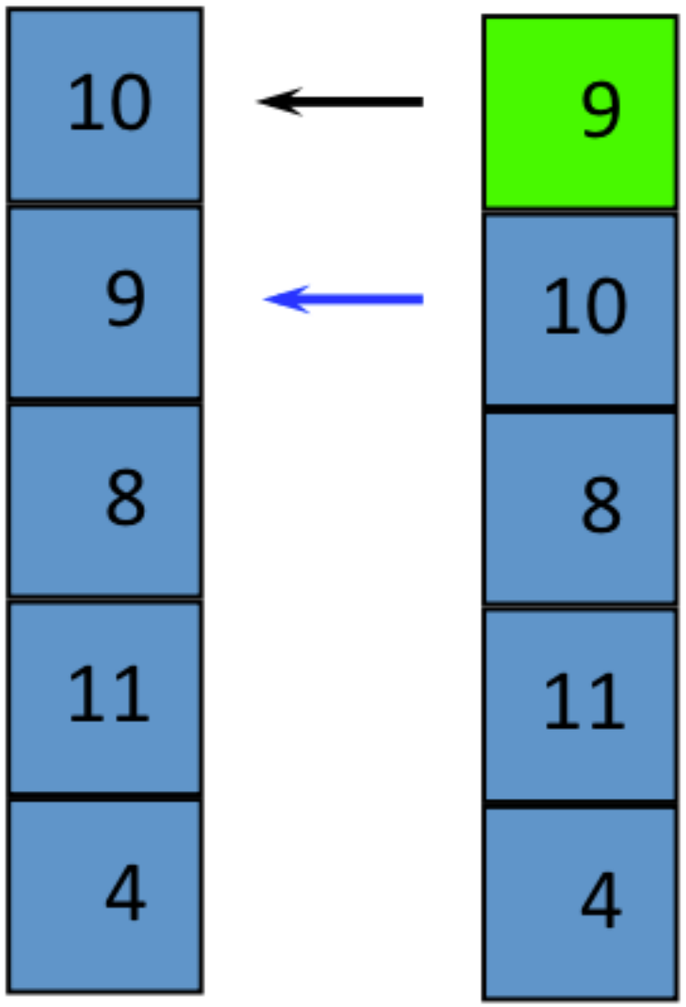
Bubble Sort

10	9	8	11	4

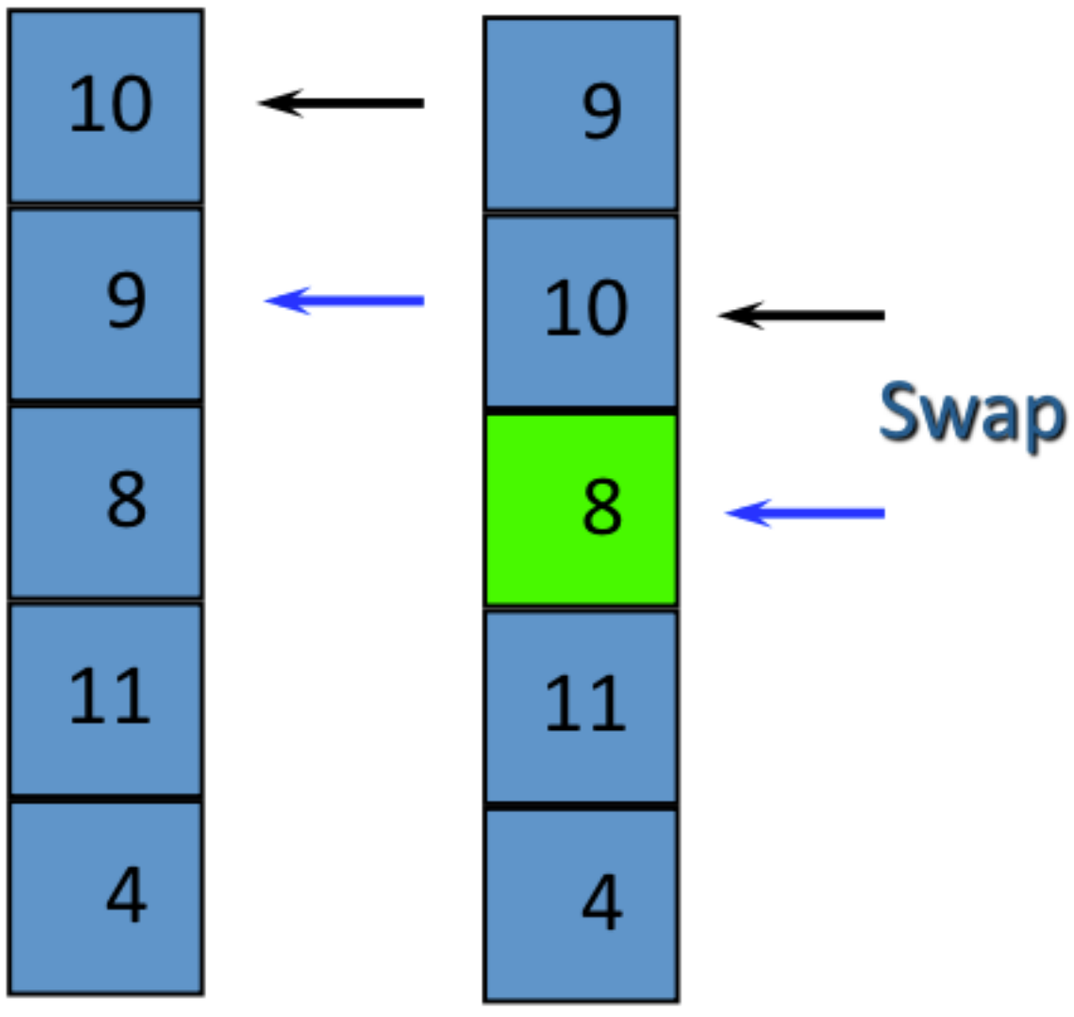
Bubble Sort



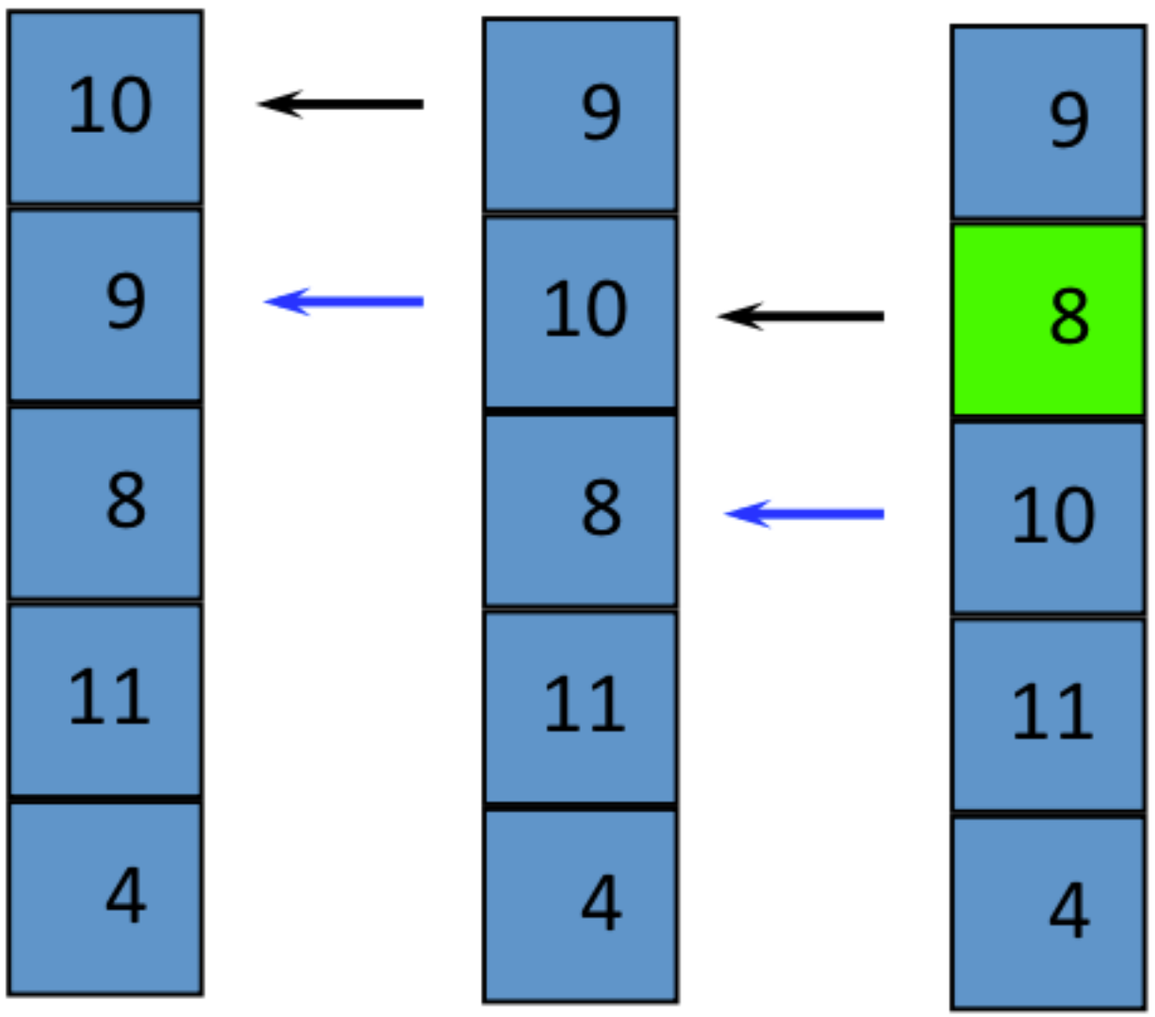
Bubble Sort



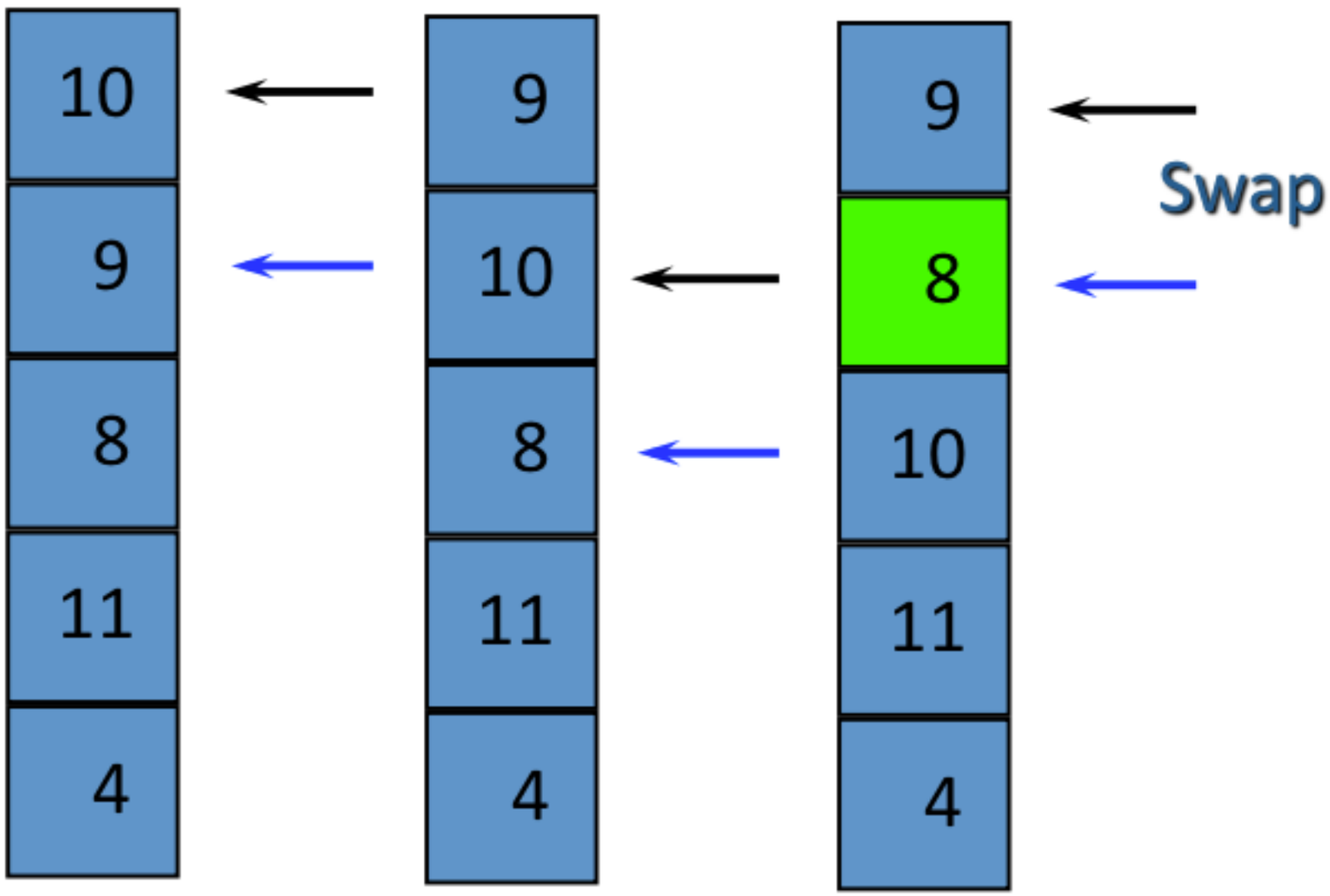
Bubble Sort



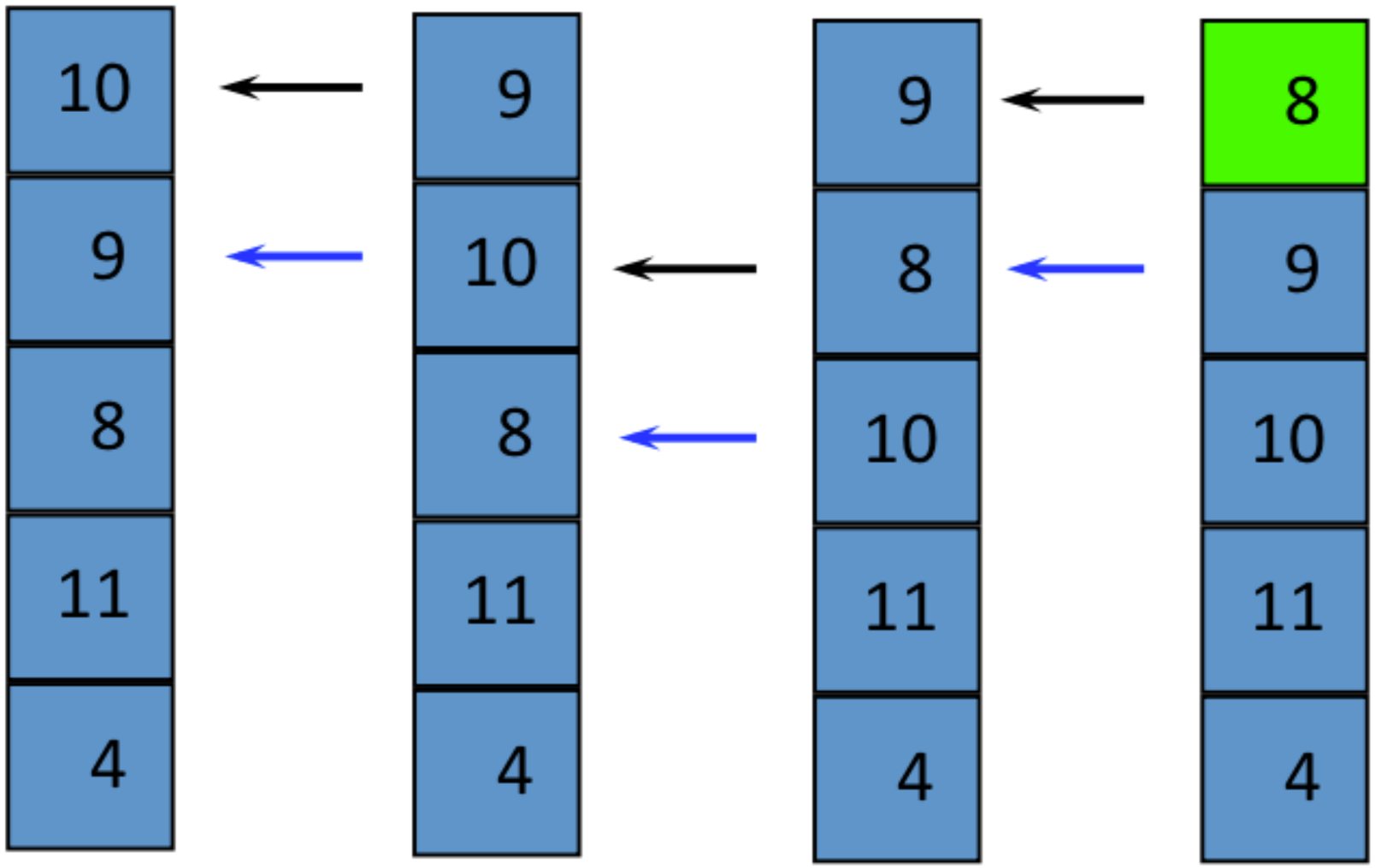
Bubble Sort



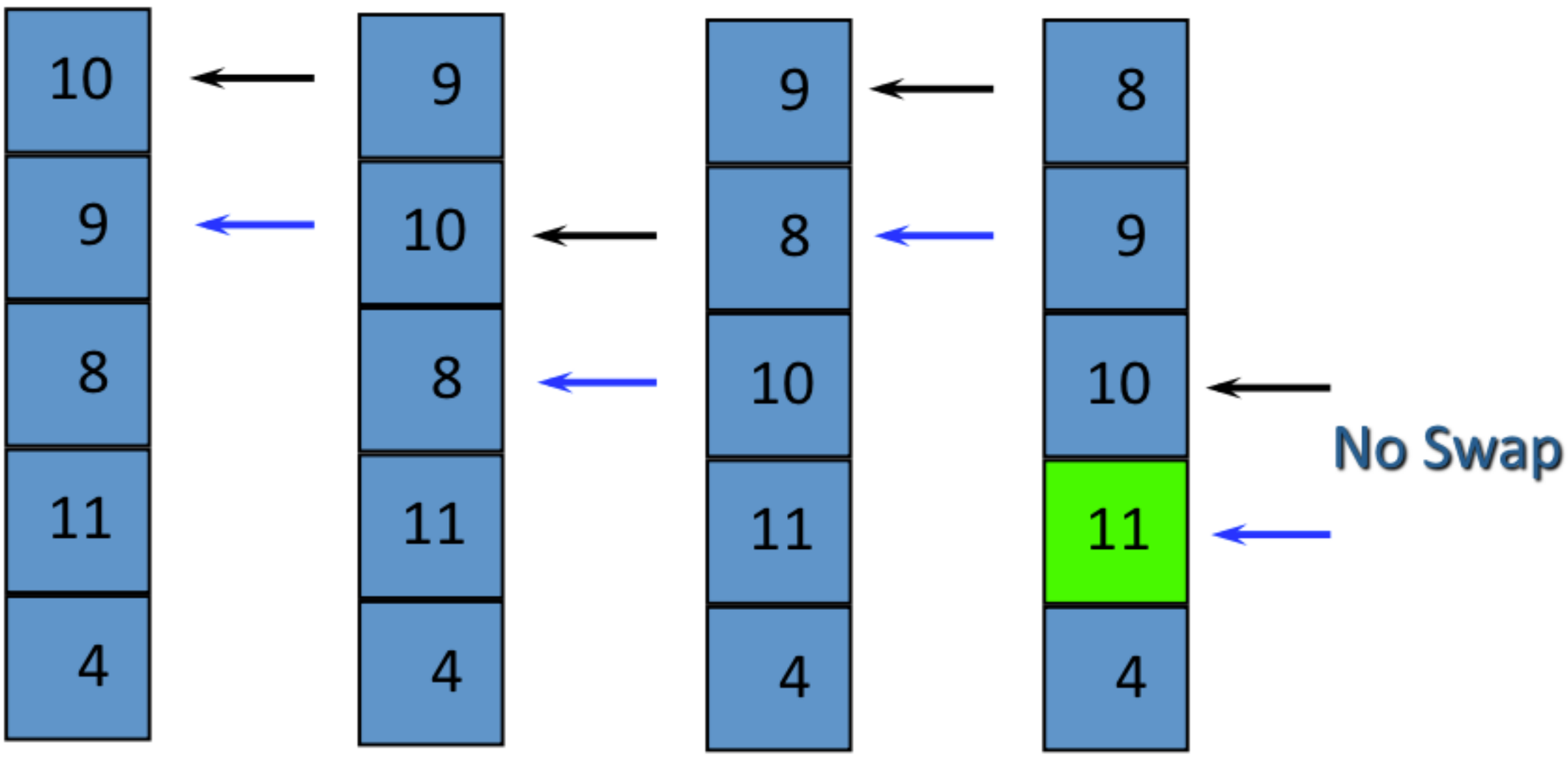
Bubble Sort



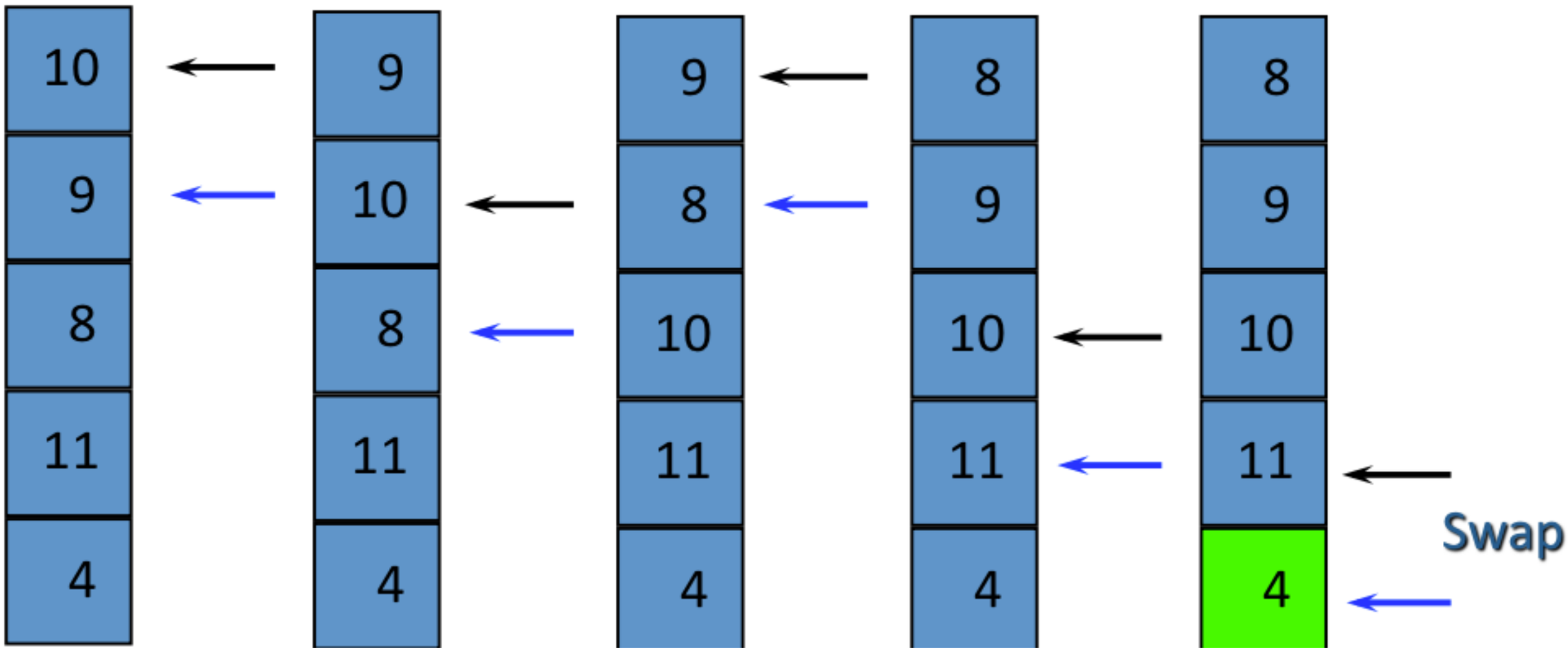
Bubble Sort



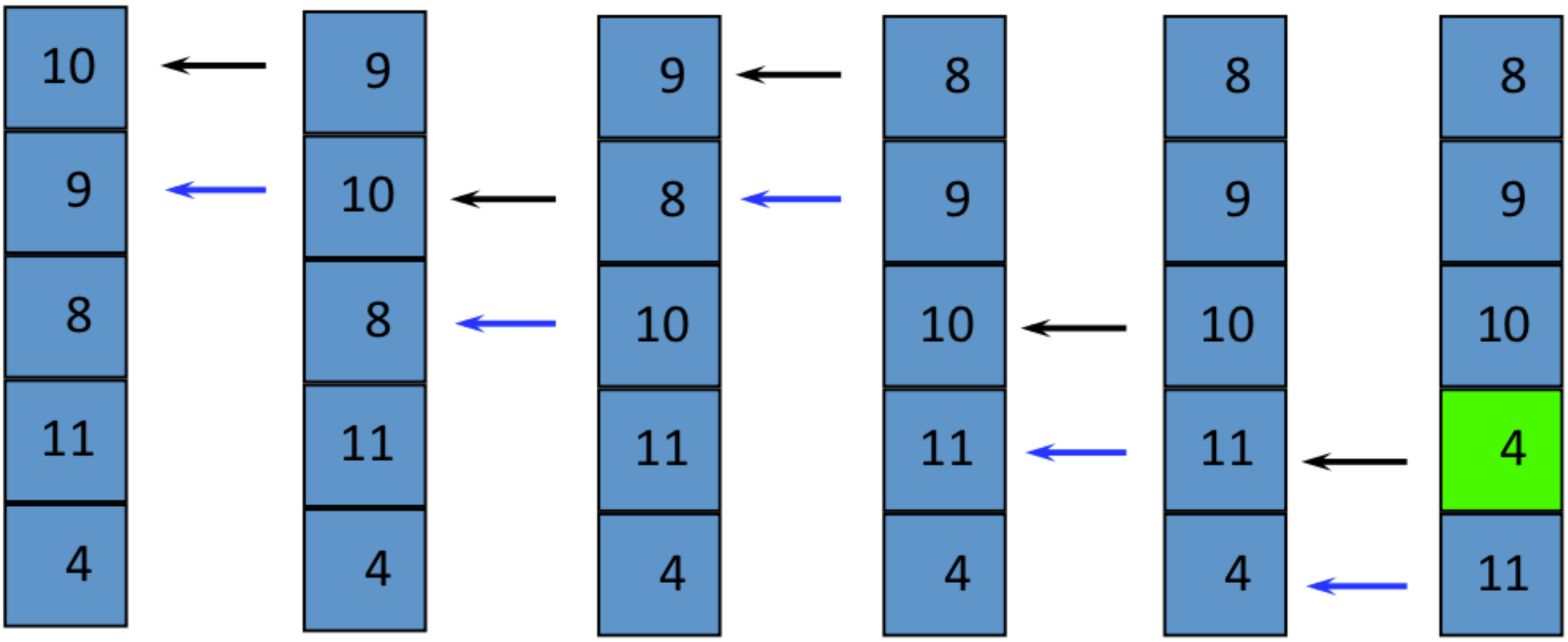
Bubble Sort



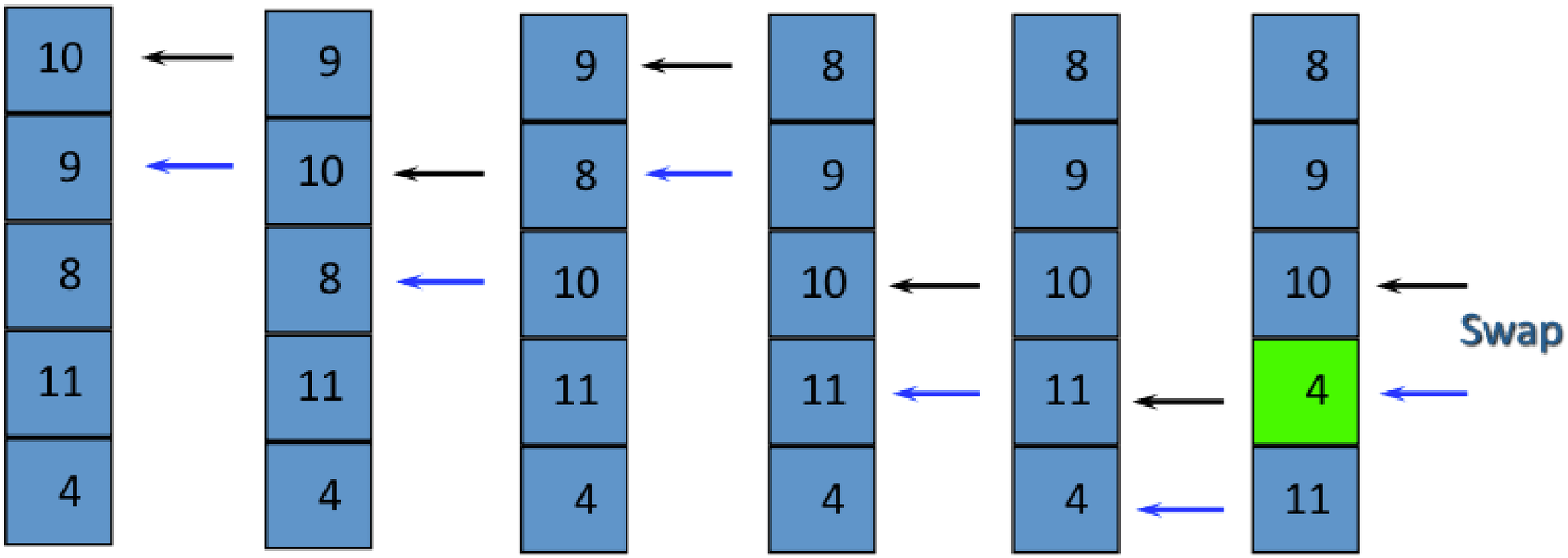
Bubble Sort



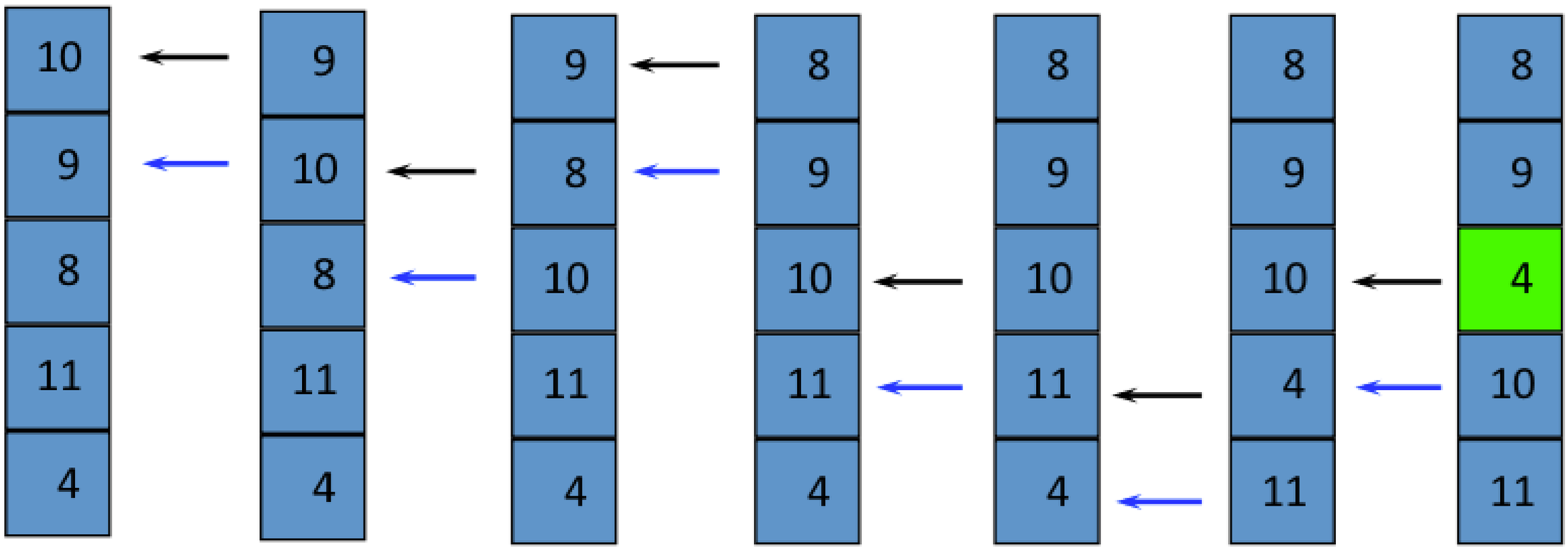
Bubble Sort



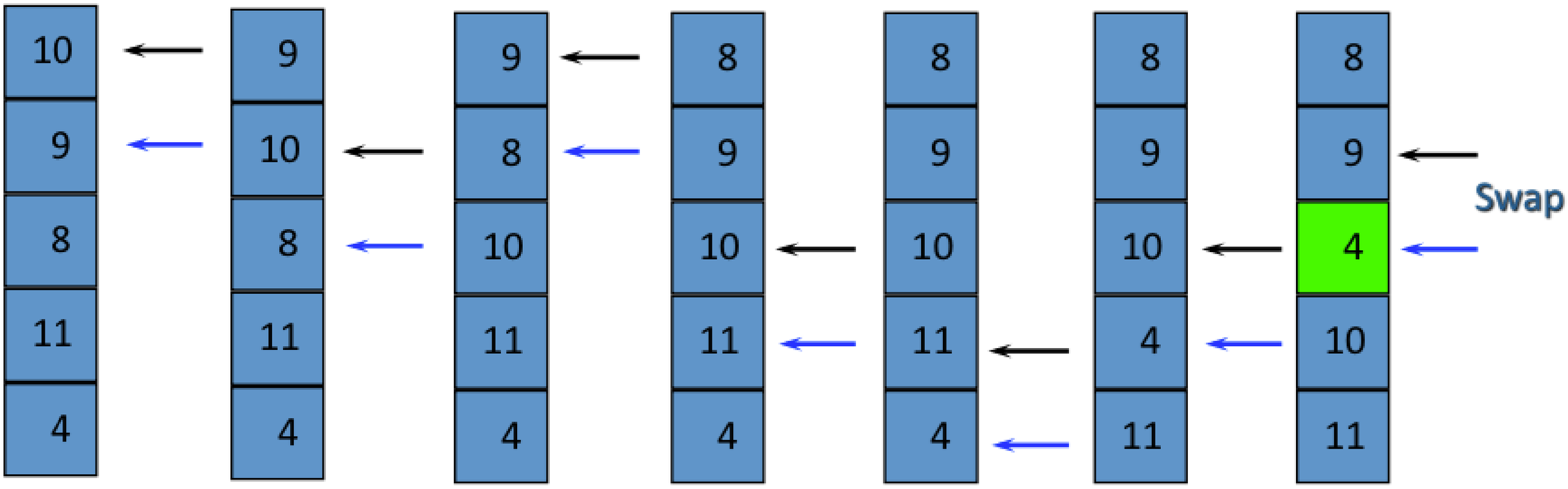
Bubble Sort



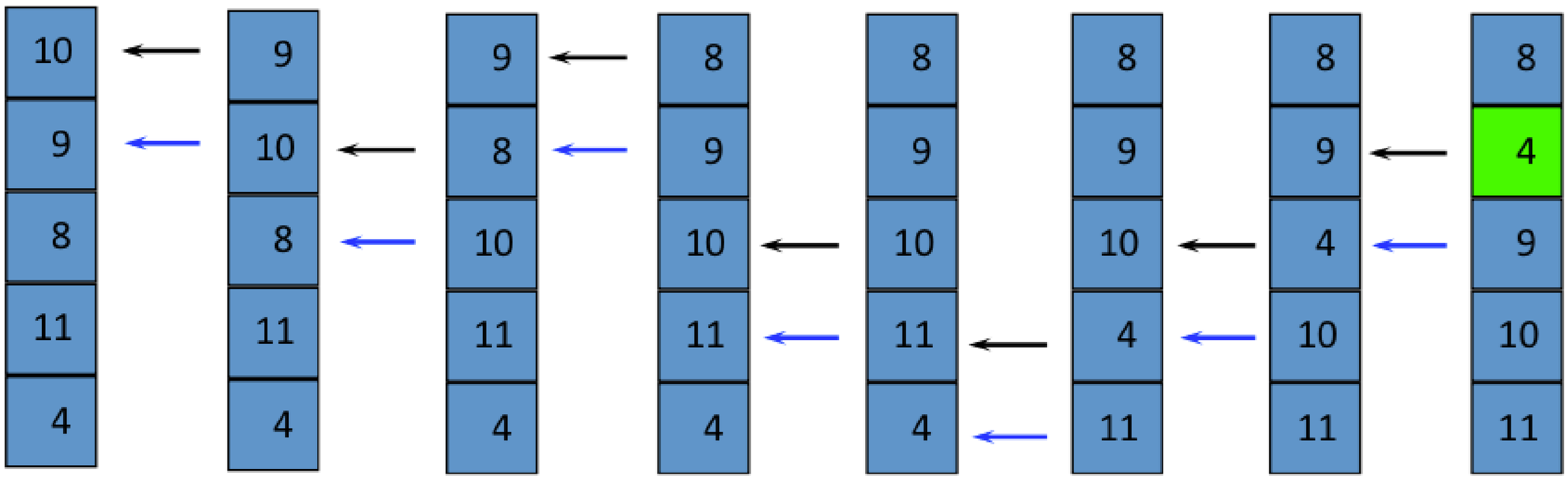
Bubble Sort



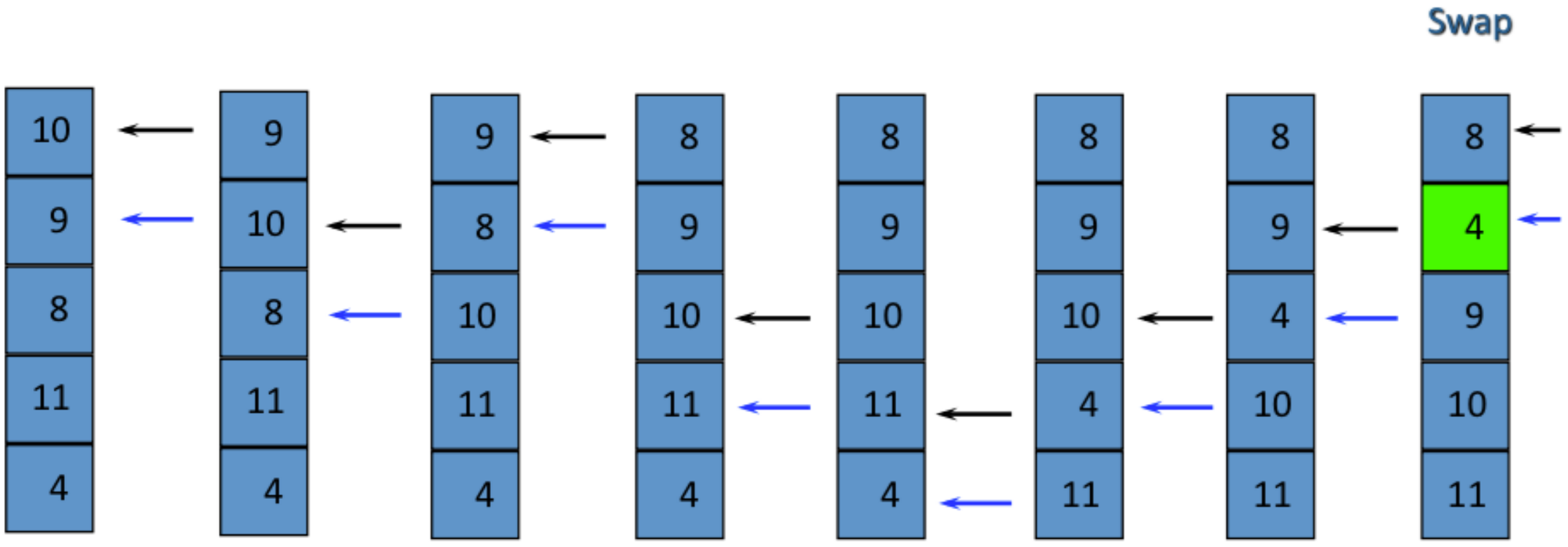
Bubble Sort



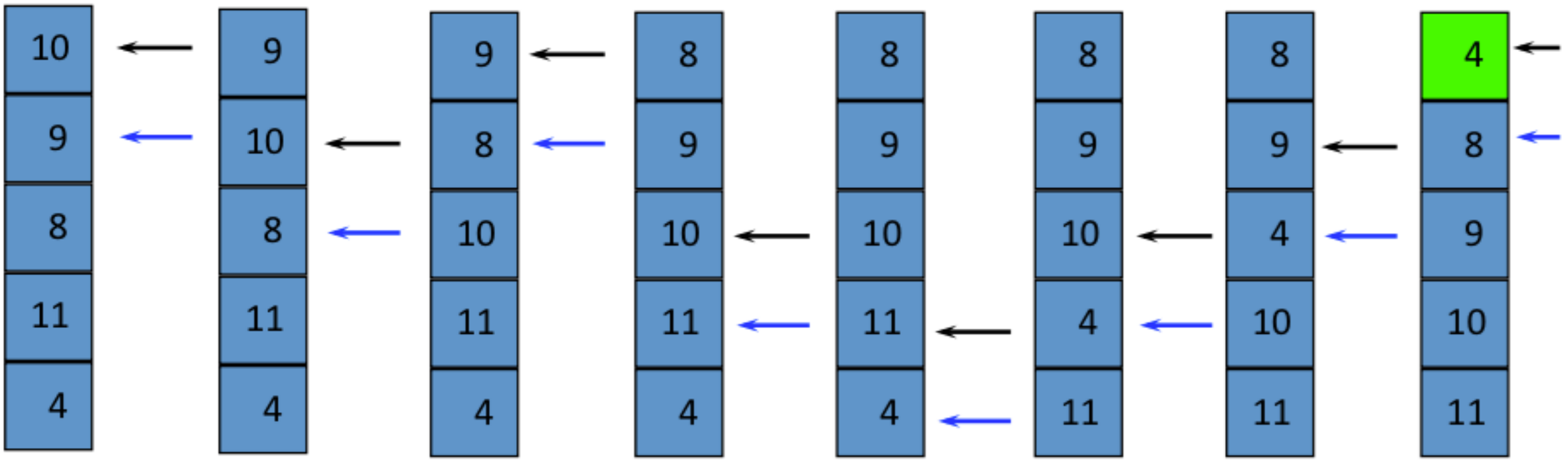
Bubble Sort



Bubble Sort


Swap
10
9
8
11
4
9
10
8
11
4
9
8
10
11
4
8
9
10
11
4
8
9
10
4
11
8
9
4
10
11
8
4
9
10
11

Bubble Sort


10
9
8
11
4
9
10
8
11
4
9
8
10
11
4
9
8
9
10
11
4
8
9
10
11
4
8
9
4
10
11
4
8
9
10
11

Bubble Sort

- Time Complexity?

```
FOR every element in the list,
    proceeding for the first to the last

   WHILE list element > previous list element
      bubble element back (up) the list
      by successive swapping with
      the element just above/prior it
```

Bubble Sort

```
public class MySort {

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

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

Bubble Sort

```
public void printArray(int a[]) {
   for(int i=0; i< a.length; i++) {
      System.out.print(a[i] + " ");
   }
}
```

```
   public static void main (String args[]) {
      MySort obj = new MySort();
      int[] num = {5,4,3,2,1};
      obj.bubbleSort(num);
      obj.printArray(num);
   }
}
```

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.*;

public class MySortComp {

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

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

```

Bubble Sort with Comparator

```
20
21      public void printArray(int a[]) {
22          for(int i=0; i< a.length; i++) {
23              System.out.print(a[i] + " ");
24          }
25      }
26
27
28      public static void main (String args[]) {
29          MySortComp obj = new MySortComp();
30          int[] num = {5,4,3,2,1};
31          NumComparator comp = new NumComparator();
32          obj.bubbleSort(num, comp);
33          obj.printArray(num);
34      }
35  }
```

Bubble Sort with Comparator

```
36
37 class NumComparator implements Comparator {
38
39     public int compare(Object o1, Object o2) {
40
41         Integer num1 = (Integer) o1;
42         Integer num2 = (Integer) o2;
43
44         if (num1 > num2)
45             return 1;
46         else
47             return 0;
48     }
49 }
50
```

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?

```java

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

- 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)
```

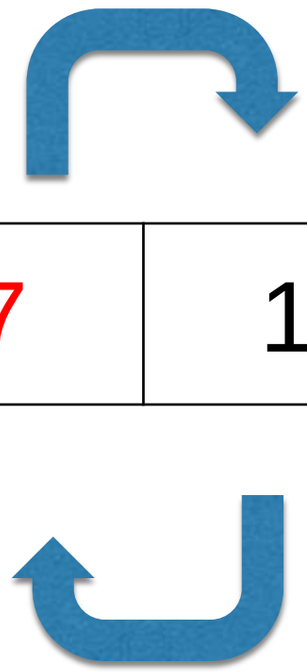
Selection Sort

29	10	14	37	13

Selection Sort

29	10	14	37	13

Selection Sort

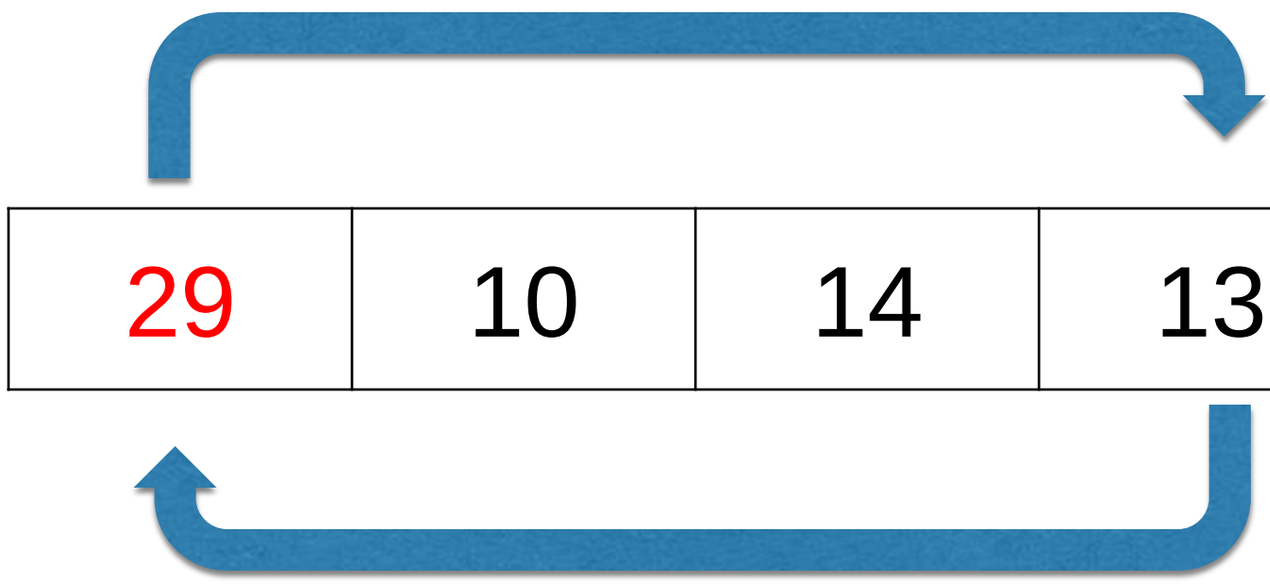


29	10	14	37	13

Selection Sort

29	10	14	13	37

Selection Sort



29	10	14	13	37

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

- 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

```
1  import java.util.*;
2
3  public class MySortComp2 {
4
5     public void selectionSort(int a[], NumComparator comp) {
6           int i,j, index_Largest, temp;
7        int size = a.length;
8
9           for (i= size-1; i >=1; i--) {
10             //select the largest item
11             index_Largest = 0;
12
13             for (j=1; j <= i; j++) {
14                   if (comp.compare(a[j], a[index_Largest]) == 1) {
15                      index_Largest = j;
16             }
17          }
18
19          /* swap the largest item with the last element */
20          temp = a[index_Largest];
21          a[index_Largest] = a[i];
22          a[i] = temp;
23       }
24    }
```

Selection Sort

```
25
26    public void printArray(int a[]) {
27       for(int i=0; i< a.length; i++) {
28          System.out.print(a[i] + " ");
29       }
30    }
31
32
33    public static void main (String args[]) {
34       MySortComp2 obj = new MySortComp2();
35       int[] num = {5,4,3,2,1};
36       NumComparator comp = new NumComparator();
37       obj.selectionSort(num, comp);
38       obj.printArray(num);
39    }
40 }
```

Selection Sort

```
41
42 class NumComparator implements Comparator {
43
44    public int compare(Object o1, Object o2) {
45
46       Integer num1 = (Integer) o1;
47       Integer num2 = (Integer) o2;
48
49       if (num1 > num2)
50          return 1;
51       else
52          return 0;
53    }
54 }
```

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 one at a time
    - As you pick each card, you insert it into its correct position in your hand or organized (sorted) cards

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

20	54	16	36	99	11	74	88

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

20	54	16	36	99	11	74	88

Insertion Sort

20	16	54	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	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

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.*;

public class MySortComp {
    public void bubbleSort(Comparable a[]){
            int i,j;
            Comparable temp;
            int size = a.length;
            for(i=0; i<size-1;i++){
                for(j=i; j>=0; j--){
                    if(a[j].compareTo(a[j+1]) == 1){
                        /* swap */
                        temp = a[j+1];
                        a[j+1] = a[j];
                        a[j] = temp;
                    }
                }
            }
    }

```

```
19
20      public void printArray(Comparable a[]){
21          for(int i=0; i<a.length; i++){
22              System.out.print(a[i] + " ");
23          }
24      }
25
26      public static void main(String args[]){
27          MySortComp obj = new MySortComp();
28          Comparable[] num = {5,4,3,2,1};
29          obj.bubbleSort(num);
30          obj.printArray(num);
31
32      }
33  }
```

```java
import java.util.*;

public class Date implements Comparable{

    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;
    }

```

```
17      public int compareTo(Object o){
18
19          Date that = (Date) o;
20
21          if (this.year > that.year) return +1;
22          if (this.year < that.year) return -1;
23          if (this.month > that.month) return +1;
24          if (this.month < that.month) return -1;
25          if (this.day > that.day) return +1;
26          if (this.day < that.day) return -1;
27          return 0;
28      }
29  }
```

Now, the same code can sort dates, too!

```
public static void main(String args[]){
    MySortComp obj = new MySortComp();
    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(1,1,1990);

    obj.bubbleSort(dates);
    obj.printArray(dates);
}
```