WIA1002/ WIB1002
Data Structure

Sorting

Sorting

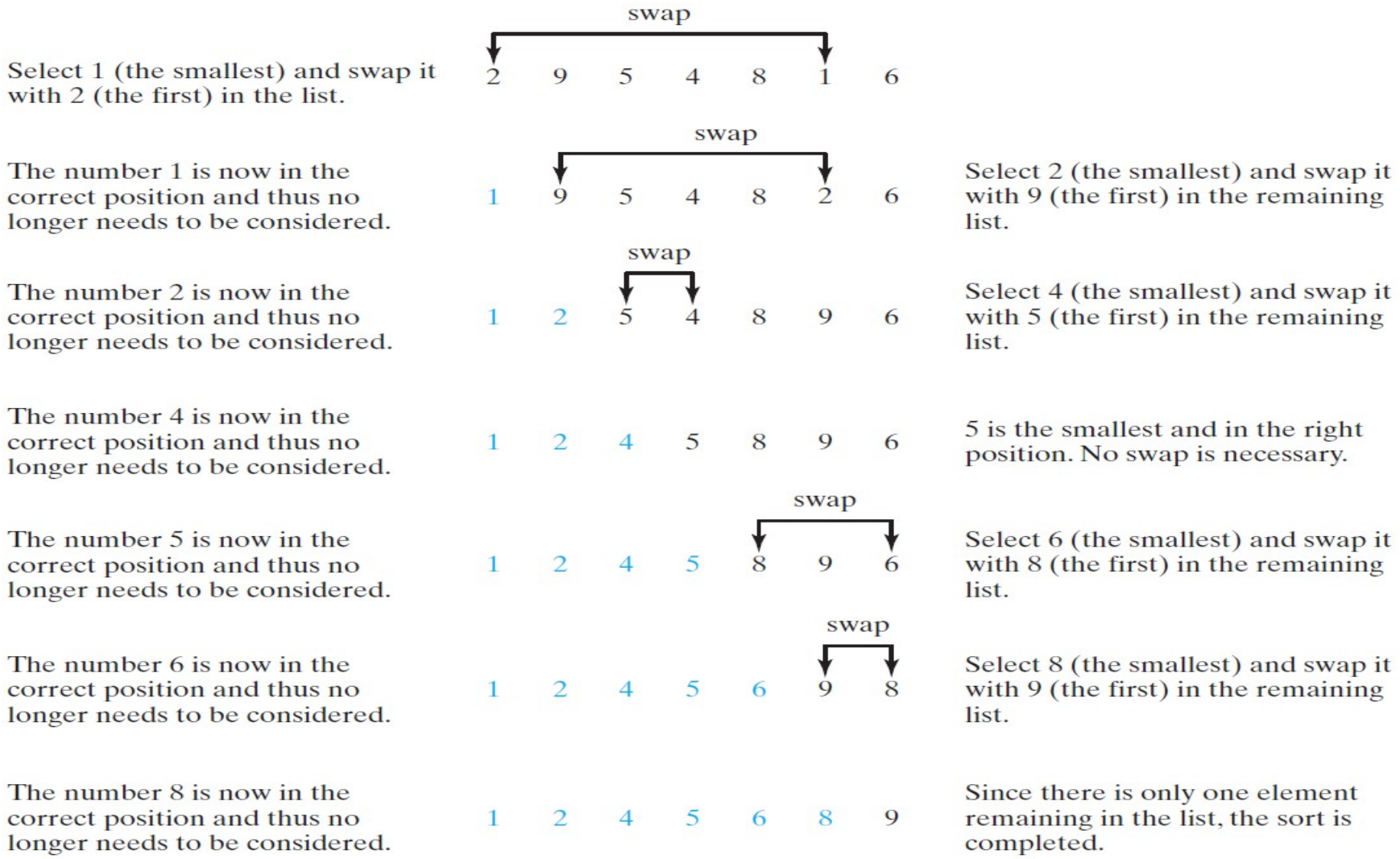
Sorting, like searching, is also a common task in computer programming.

Some sorting algorithms:

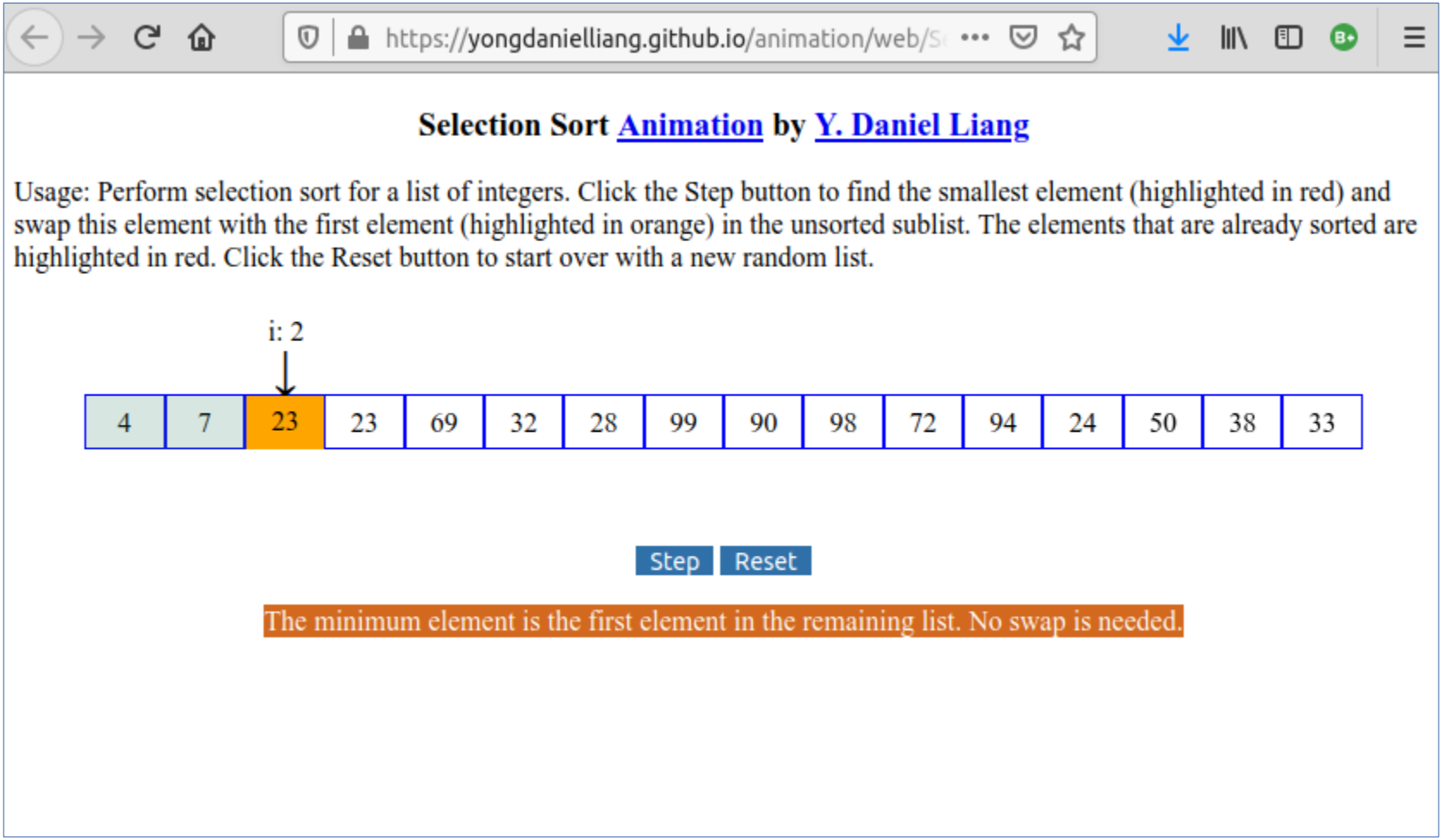
1. Selection sort
2. Insertion sort
3. Bubble sort
4. Merge Sort

Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.



Selection Sort Animation



https://yongdanielliang.github.io/animation/web/SelectionSortNew.html

From Idea to Solution

```
for (int i = 0; i < list.length - 1; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration applies on list[i+1..listSize-1]
}
```

```
list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

                                ...

list[0] list[1] list[2] list[3] ...                    list[10]
```

```
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length - 1; i++) {
    // Find the minimum in the list[i..list.length-1]
    double currentMin = list[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

```java
public class SelectionSort {
  /** The method for sorting the numbers */
  public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length - 1; i++) {
      // Find the minimum in the list[i..list.length-1]
      double currentMin = list[i];
      int currentMinIndex = i;

      for (int j = i + 1; j < list.length; j++) {
        if (currentMin > list[j]) {
          currentMin = list[j];
          currentMinIndex = j;
        }
      }

      // Swap list[i] with list[currentMinIndex] if necessary;
      if (currentMinIndex != i) {
        list[currentMinIndex] = list[i];
        list[i] = currentMin;
      }
    }
  }

  public static void main(String[] args) {
    double[] list = {-2, 4.5, 5, 1, 2, -3.3};
    selectionSort(list);
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }
}
```

Insertion Sort

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

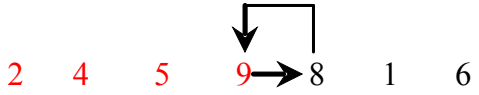
The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.

Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 into the sublist.

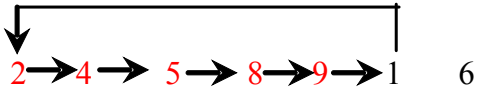
Step2: The sorted sublist is {2, 9}. Insert 5 into the sublist.

Step 3: The sorted sublist is {2, 5, 9}. Insert 4 into the sublist.

Step 4: The sorted sublist is {2, 4, 5, 9}. Insert 8 into the sublist.



Step 5: The sorted sublist is {2, 4, 5, 8, 9}. Insert 1 into the sublist.



Step 6: The sorted sublist is {1, 2, 4, 5, 8, 9}. Insert 6 into the sublist.

Step 7: The entire list is now sorted.

Insertion Sort Animation

https://yongdanielliang.github.io/animation/web/InsertionSortNew.html

animation

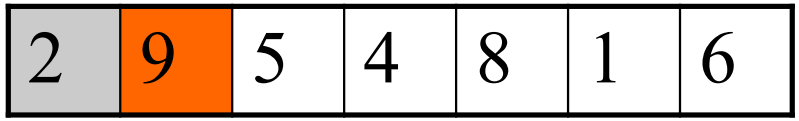
Insertion Sort

```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```

animation

Insertion Sort

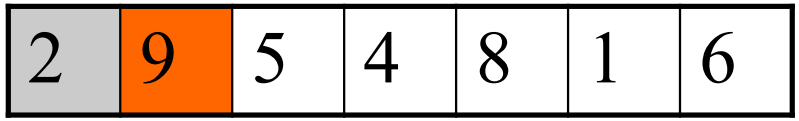
```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```

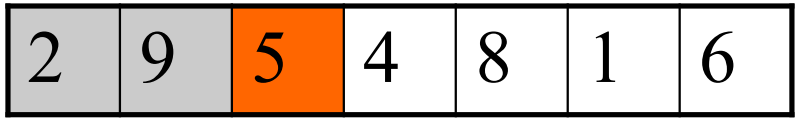


animation

Insertion Sort

```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```

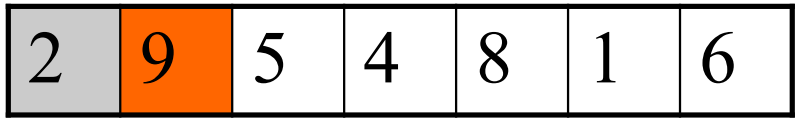


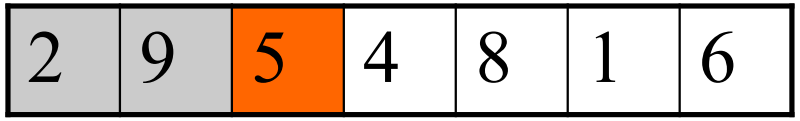


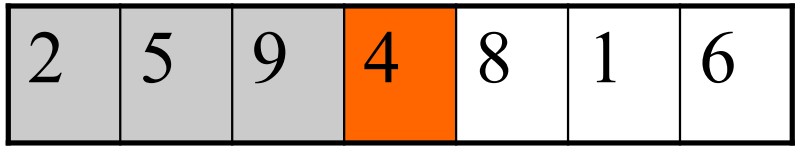
animation

Insertion Sort

```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```



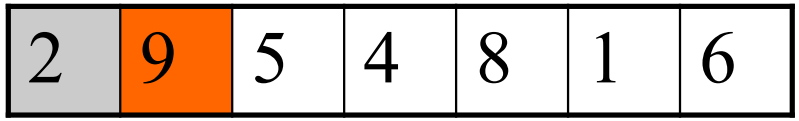


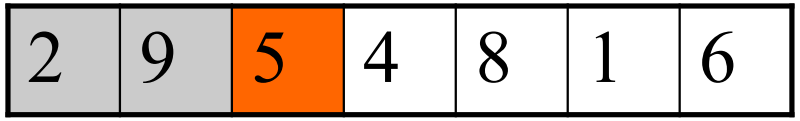


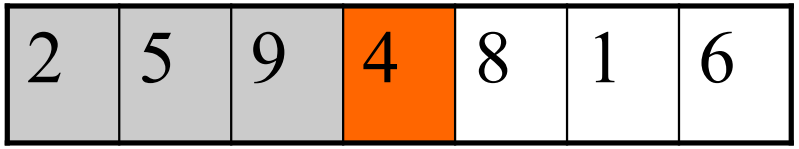
animation

Insertion Sort

```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```



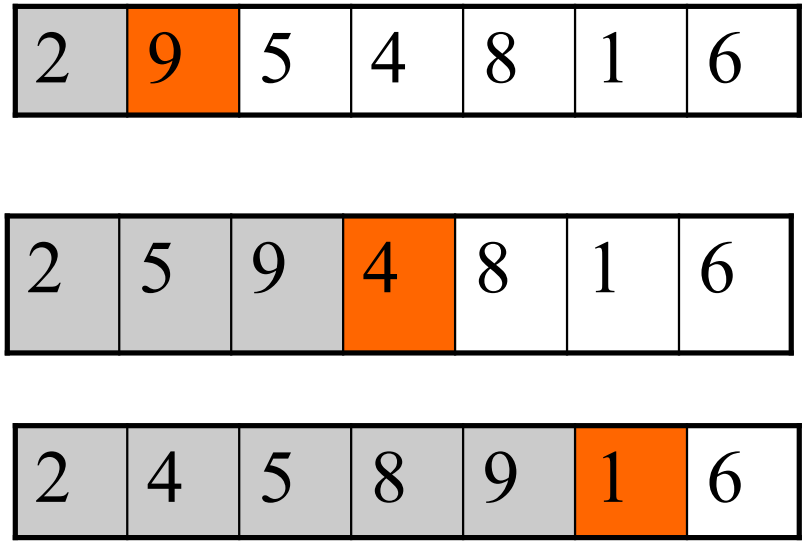




animation

Insertion Sort

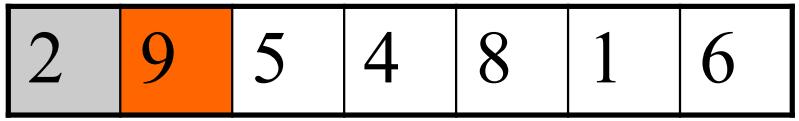
```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```

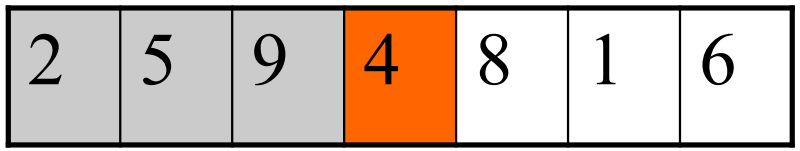


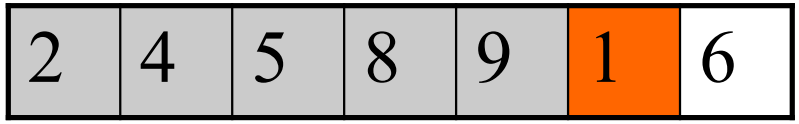
animation

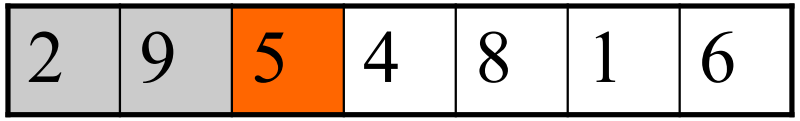
Insertion Sort

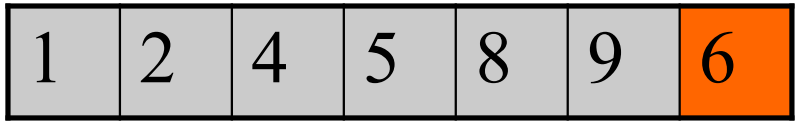
```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```







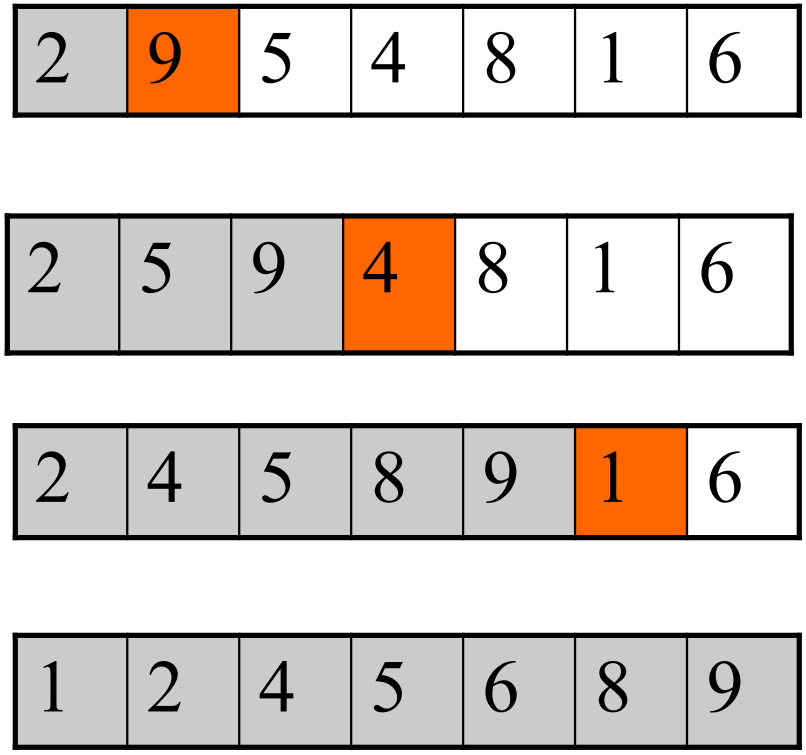


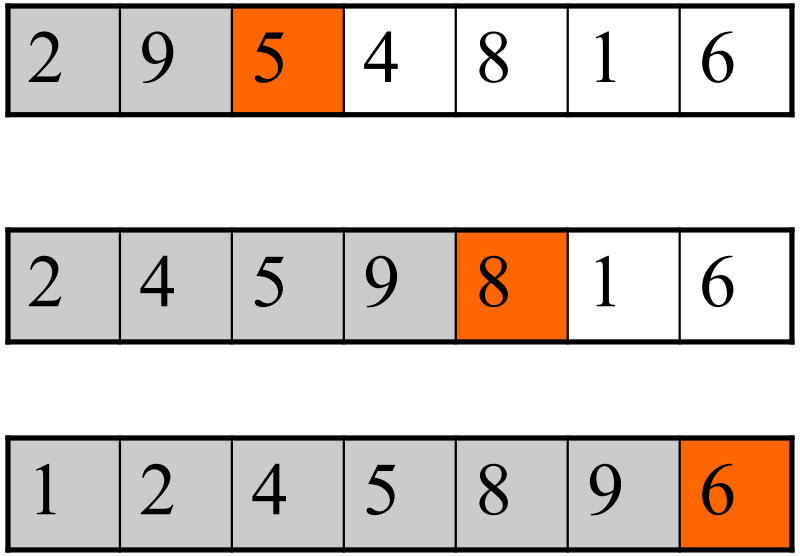


animation

Insertion Sort

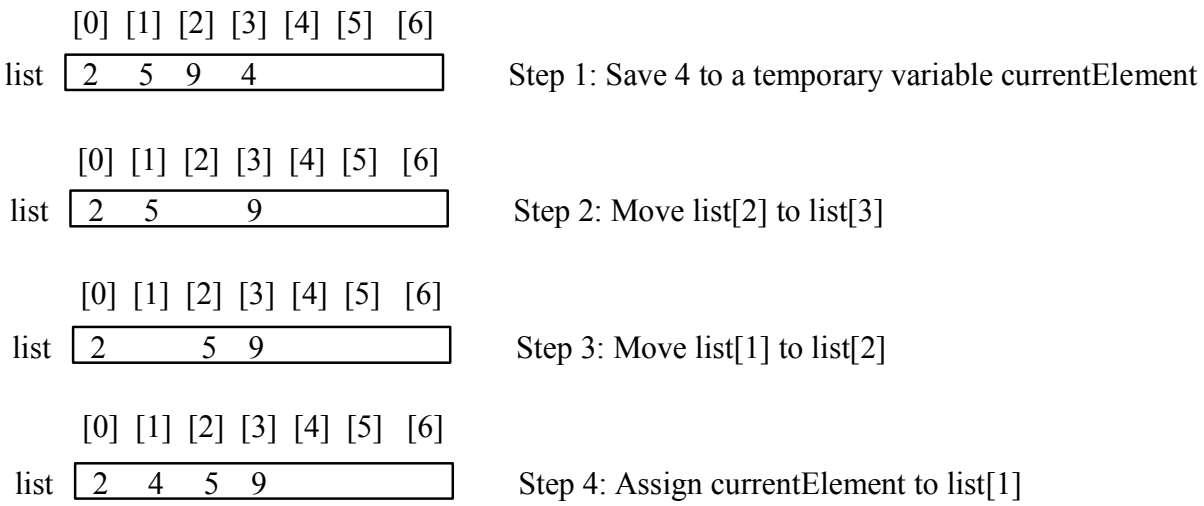
```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```





How to Insert?

The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.



From Idea to Solution

```
for (int i = 1; i < list.length; i++) {
  insert list[i] into a sorted sublist list[0..i-1] so that
  list[0..i] is sorted
}
```

```
list[0]

list[0] list[1]

list[0] list[1] list[2]

list[0] list[1] list[2] list[3]

list[0] list[1] list[2] list[3] ...
```

From Idea to Solution

```
for (int i = 1; i < list.length; i++) {
  insert list[i] into a sorted sublist list[0..i-1] so that
  list[0..i] is sorted
}
```

Expand

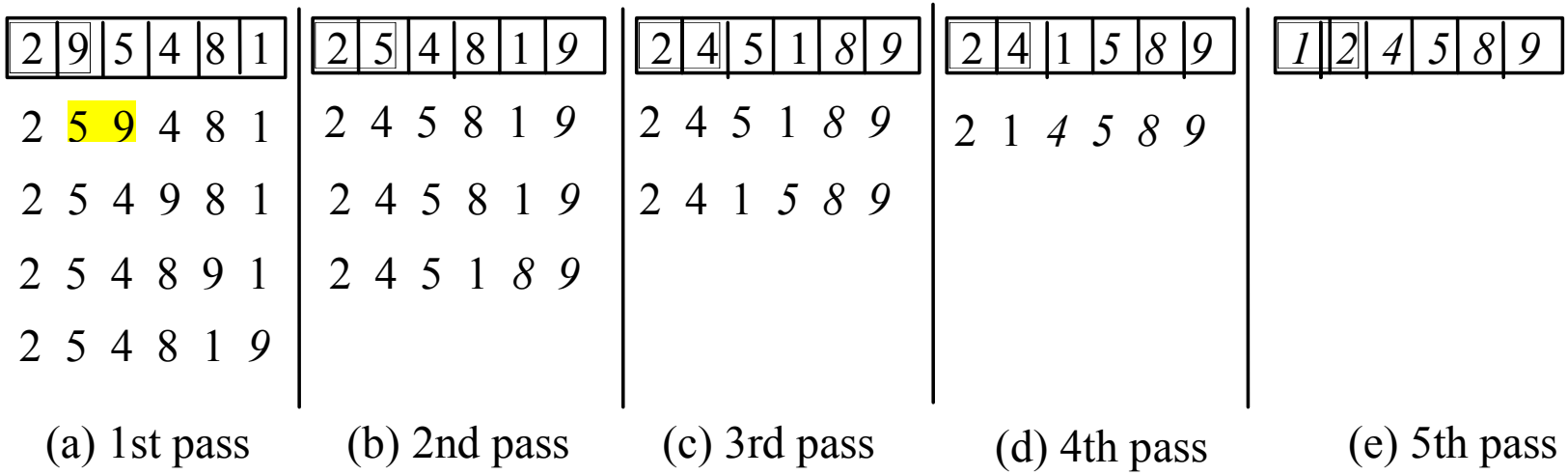
```
double currentElement = list[i];
int k;
for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
  list[k + 1] = list[k];
}
// Insert the current element into list[k + 1]
list[k + 1] = currentElement;
```

```java
public class InsertionSort {
  public static void insertionSort(int[] list) {
    for (int i = 1; i < list.length; i++) {
      int currentElement = list[i];
      int k;
      for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
        list[k + 1] = list[k];
      }

      list[k + 1] = currentElement;
    }
  }

  public static void main(String[] args) {
    int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
    insertionSort(list);
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }
}
```

Bubble Sort

- The bubble sort algorithm makes several passes through the array.
- On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged.
- The smaller values gradually "bubble" their way to the top and the larger values "sink" to the bottom.
- After the first pass, the last element becomes the largest in the array.
- After the second pass, the second-to-last element becomes the second largest in the array.
- This process is continued until all elements are sorted.

Bubble Sort



(a) 1st pass (b) 2nd pass (c) 3rd pass (d) 4th pass (e) 5th pass

Bubble Sort Animation

https://yongdanielliang.github.io/animation/web/BubbleSortNew.html

Algorithm: Bubble sort

```
1  for (int k = 1; k < list.length; k++) {
2    // Perform the kth pass
3    for (int i = 0; i < list.length - k; i++) {
4      if (list[i] > list[i + 1])
5        swap list[i] with list[i + 1];
6    }
7  }
```

Algorithm: Bubble sort

```
1  for (int k = 1; k < list.length; k++) {
2    // Perform the kth pass
3    for (int i = 0; i < list.length - k; i++) {
4      if (list[i] > list[i + 1])
5        swap list[i] with list[i + 1];
6    }
7  }
```

If no swap takes place in a pass, there is no need to perform the next pass, because all elements are sorted. So use, boolean operator to improve this algorithm.

Algorithm: Bubble sort

```
1  for (int k = 1; k < list.length; k++) {
2    // Perform the kth pass
3    for (int i = 0; i < list.length - k; i++) {
4      if (list[i] > list[i + 1])
5        swap list[i] with list[i + 1];
6    }
7  }
```

If no swap takes place in a pass, there is no need to perform the next pass, because all elements are sorted. So use, boolean operator to improve this algorithm.

```
1  boolean needNextPass = true;
2  for (int k = 1; k < list.length && needNextPass; k++) {
3    // Array may be sorted and next pass not needed
4    needNextPass = false;
5    // Perform the kth pass
6    for (int i = 0; i < list.length - k; i++) {
7      if (list[i] > list[i + 1]) {
8        swap list[i] with list[i + 1];
9        needNextPass = true; // Next pass still needed
10     }
11   }
12 }
```

Algorithm: Bubble sort

```
1  for (int k = 1; k < list.length; k++) {
2    // Perform the kth pass
3    for (int i = 0; i < list.length - k; i++) {
4      if (list[i] > list[i + 1])
5        swap list[i] with list[i + 1];
6    }
7  }
```

If no swap takes place in a pass, there is no need to perform the next pass, because all elements are sorted. So use, boolean operator to improve this algorithm.

```
1  boolean needNextPass = true;
2  for (int k = 1; k < list.length && needNextPass; k++) {
3    // Array may be sorted and next pass not needed
4    needNextPass = false;
5    // Perform the kth pass
6    for (int i = 0; i < list.length - k; i++) {
7      if (list[i] > list[i + 1]) {
8        swap list[i] with list[i + 1];
9        needNextPass = true; // Next pass still needed
10     }
11   }
12 }
```

By default each *iteration* will set *needNextPass* to false

If swap still occurs it means that the list is not yet sort out
So we set the flag to true so that it will perform the next pass

```java
public class BubbleSort {
  public static void bubbleSort(int[] list) {
    boolean needNextPass = true;
    for (int k = 1; k < list.length && needNextPass; k++) {
      needNextPass = false;
      for (int i = 0; i < list.length - k; i++) {
        if (list[i] > list[i + 1]) {
          // Swap list[i] with list[i + 1]
          int temp = list[i];
          list[i] = list[i + 1];
          list[i + 1] = temp;
           needNextPass = true;
        }
      }
    }
  }

  /** A test method */
  public static void main(String[] args) {
    int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
    bubbleSort(list);
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }
}
```

Merge Sort

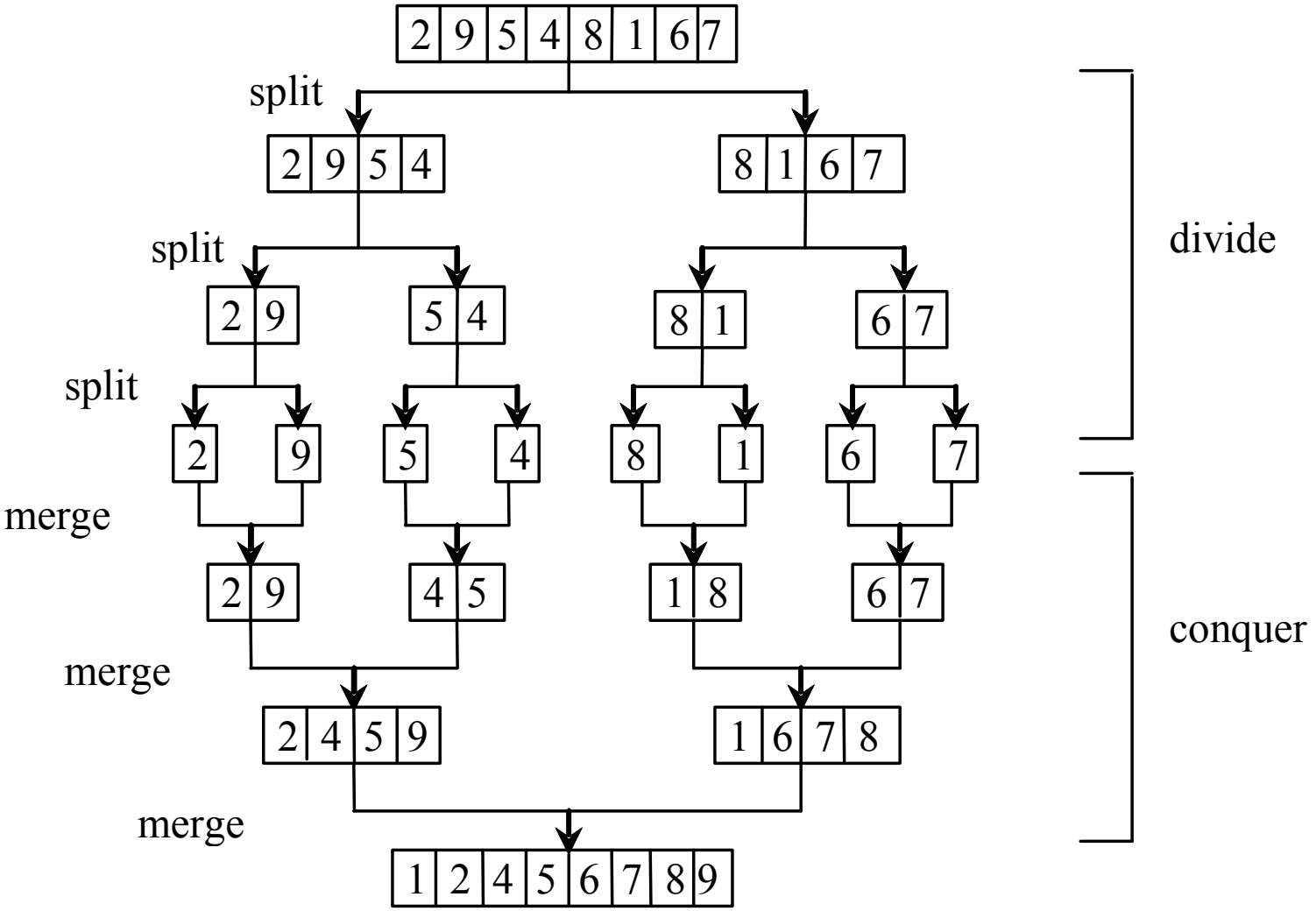
- can be described recursively as follows: divides the array into two halves and applies a merge sort on each half recursively. After the two halves are sorted, merge them.

```
mergeSort(list):
    firstHalf = mergeSort(firstHalf);
    secondHalf = mergeSort(secondHalf);
    list = merge(firstHalf, secondHalf);
```

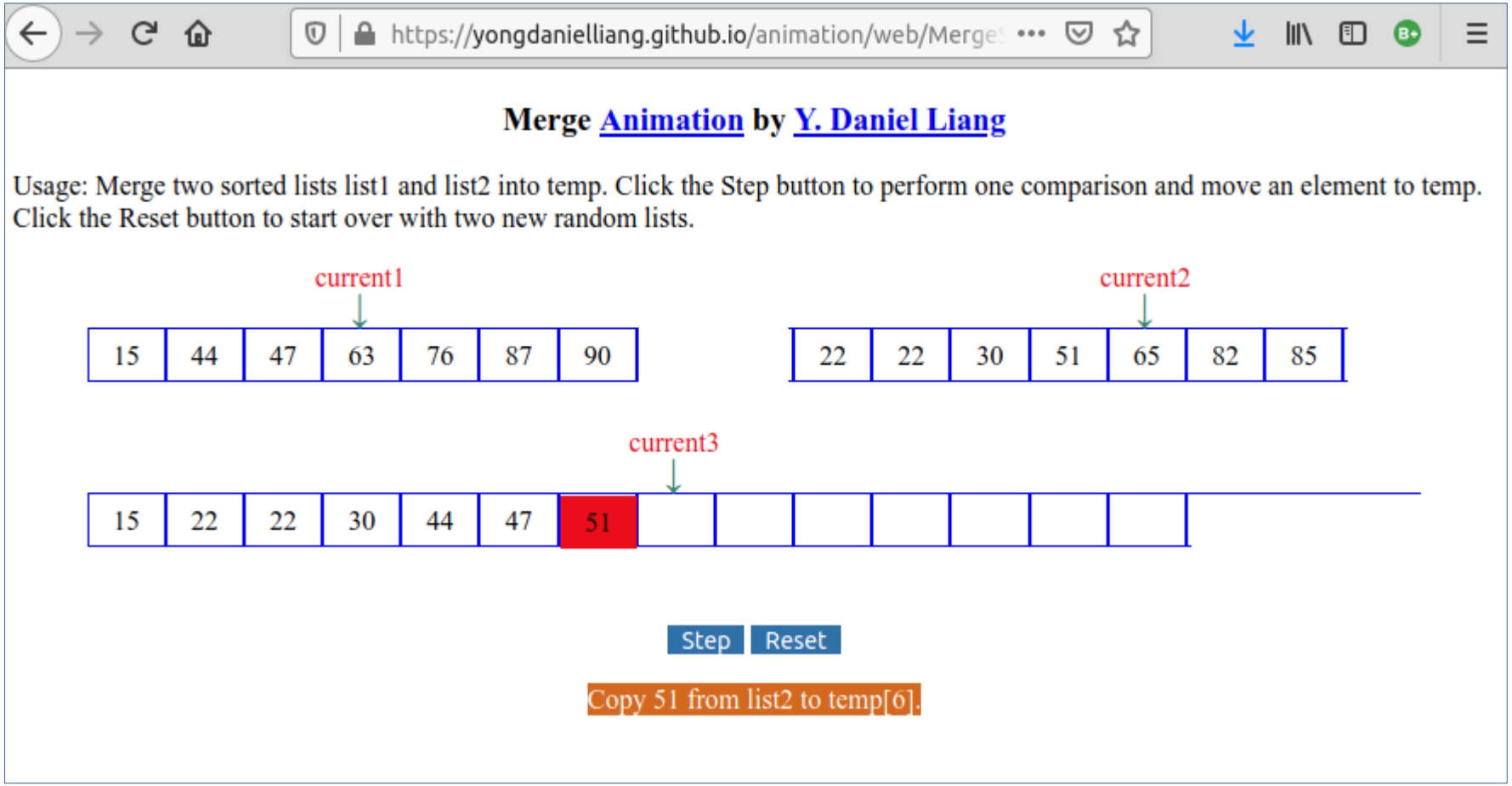
LISTING 23.5 Merge Sort Algorithm

```
1  public static void mergeSort(int[] list) {
2    if (list.length > 1) {                                  base condition
3      mergeSort(list[0 ... list.length / 2]);               sort first half
4      mergeSort(list[list.length / 2 + 1 ... list.length]); sort second half
5      merge list[0 ... list.length / 2] with                merge two halves
6        list[list.length / 2 + 1 ... list.length];
7    }
8  }
```

Merge Sort


2 9 5 4 8 1 6 7
split
2 9 5 4
8 1 6 7
split
2 9
5 4
8 1
6 7
split
2
9
5
4
8
1
6
7
merge
2 9
4 5
1 8
6 7
merge
2 4 5 9
1 6 7 8
merge
1 2 4 5 6 7 8 9
divide
conquer

Merging Two Sorted Lists



https://yongdanielliang.github.io/animation/web/MergeSortNew.html

LISTING 23.6 MergeSort.java

```
1  public class MergeSort {
2    /** The method for sorting the numbers */
3    public static void mergeSort(int[] list) {
4      if (list.length > 1) {                                    base case
5        // Merge sort the first half
6        int[] firstHalf = new int[list.length / 2];
7        System.arraycopy(list, 0, firstHalf, 0, list.length / 2);
8        mergeSort(firstHalf);                                   sort first half
9
10       // Merge sort the second half
11       int secondHalfLength = list.length - list.length / 2;
12       int[] secondHalf = new int[secondHalfLength];
13       System.arraycopy(list, list.length / 2,
14         secondHalf, 0, secondHalfLength);
15       mergeSort(secondHalf);                                  sort second half
16
17       // Merge firstHalf with secondHalf into list
18       merge(firstHalf, secondHalf, list);                     merge two halves
19     }
20   }
21
```

```
22   /** Merge two sorted lists */
23   public static void merge(int[] list1, int[] list2, int[] temp) {
24     int current1 = 0; // Current index in list1
25     int current2 = 0; // Current index in list2
26     int current3 = 0; // Current index in temp
27
28     while (current1 < list1.length && current2 < list2.length) {
29       if (list1[current1] < list2[current2])
30         temp[current3++] = list1[current1++];
31       else
32         temp[current3++] = list2[current2++];
33     }
34
35     while (current1 < list1.length)
36       temp[current3++] = list1[current1++];
37
38     while (current2 < list2.length)
39       temp[current3++] = list2[current2++];
40   }
41
```

list1 to temp

list2 to temp

rest of list1 to temp

rest of list2 to temp

```
41
42   /** A test method */
43   public static void main(String[] args) {
44     int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
45     mergeSort(list);
46     for (int i = 0; i < list.length; i++)
47       System.out.print(list[i] + " ");
48   }
49 }
```

Sort Phone List : generic sort

```
public class SortPhoneList {
   // Creates an array of Contact objects, sorts them, then prints them.
   public static void main(String[] args)     {
       Contact[] friends = new Contact[7];
       friends[0] = new Contact("John", "Smith", "610-555-7384");
       friends[1] = new Contact("Sarah", "Barnes", "215-555-3827");
       friends[2] = new Contact("Mark", "Riley", "733-555-2969");
       friends[3] = new Contact("Laura", "Getz", "663-555-3984");
       friends[4] = new Contact("Larry", "Smith", "464-555-3489");
       friends[5] = new Contact("Frank", "Phelps", "322-555-2284");
       friends[6] = new Contact("Marsha", "Grant", "243-555-2837");

       GenericSelectionSort.selectionSort(friends);
       for (Contact friend : friends)
           System.out.println(friend);

       String[] strArray = {"B", "D" , "A", "Z"};
       GenericSelectionSort.selectionSort(strArray);
       for (String str : strArray)
           System.out.println(str);
   }

}
```

Sort Phone List : generic sort

```
Barnes, Sarah   215-555-3827
Getz, Laura 663-555-3984
Grant, Marsha   243-555-2837
Phelps, Frank   322-555-2284
Riley, Mark 733-555-2969
Smith, John 610-555-7384
Smith, Larry    464-555-3489
A
B
D
Z
```

Sort Phone List : generic sort

```
public class Contact implements Comparable<Contact>
{
    private String firstName, lastName, phone;
    public Contact(String first, String last, String telephone)
    {
        firstName = first;
        lastName = last;
        phone = telephone;
    }

    public String toString()
    {
        return lastName + ", " + firstName + "\t" + phone;
    }

    public int compareTo(Contact other)
    {
        int result;
        if (lastName.equals(other.lastName))
            result = firstName.compareTo(other.firstName);
        else
            result = lastName.compareTo(other.lastName);
        return result;
    }
}
```

Sort Phone List : generic sort

Modify the following SelectionSort to be a generic method

```
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length - 1; i++) {
    double currentMin = list[i];
    int currentMinIndex = i;

    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

Sort Phone List : generic sort

Modify the following SelectionSort to be a generic method

```
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length - 1; i++) {
    double currentMin = list[i];
    int currentMinIndex = i;

    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

Generic type that extends comparable

Comparable

Sort Phone List : generic sort

```
public class GenericSelectionSort {
   public static <T extends Comparable<T>> void selectionSort(T[] list)
   {
      for (int i = 0; i < list.length-1; i++) {
         T currentMin = list[i];
         int currentMinIndex = i;

         for (int j = i + 1; j < list.length; j++) {
            if (currentMin.compareTo(list[j]) > 0) {
               currentMin = list[j];
               currentMinIndex = j;
            }
         }
         swap(list, currentMinIndex, i);
      }
   }

   private static <T extends Comparable<T>> void swap(T[] data, int index1, int index2)
   {
       T temp = data[index1];
       data[index1] = data[index2];
       data[index2] = temp;
   }
}
```

References

Chapters 7 and 23, Liang, Introduction to Java Programming 10th edition, Pearson 2015