WIA 1002 DATA STRUCTURE

SEM 2, SESSION 2024/205

NURUL JAPAR
nuruljapar@um.edu.my

HOO WAI LAM
wlhoo@um.edu.my

SEARCH & SORT

SEARCH

Searching

- Searching is the process of looking for a specific element in a group of items (such as in an array)
- Two common searching approaches: Linear and Binary Search

Linear Search

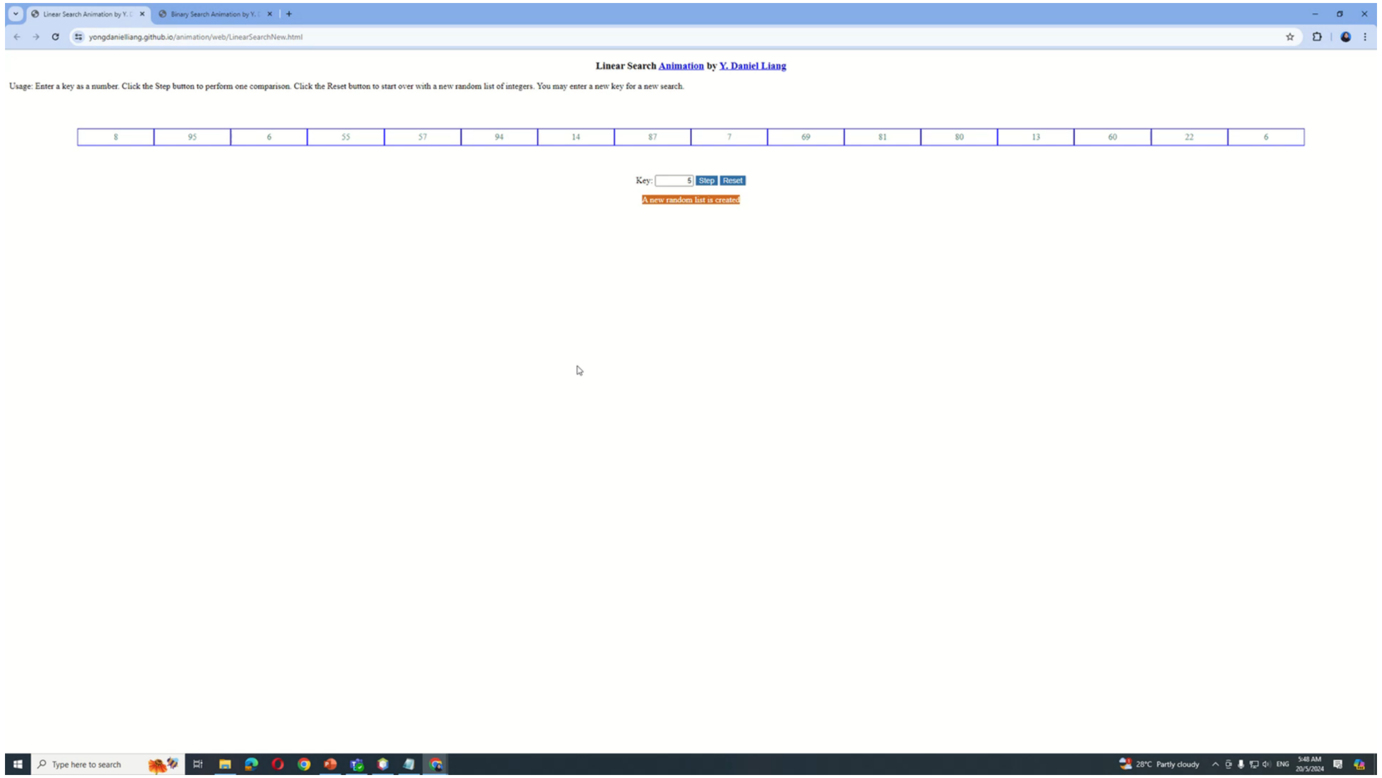
- compares the key element, key, **sequentially** with each element in the group (such as array list).

- continues to do so until the key matches an element in the list or the list is exhausted without a match being found.
- If found, returns the index of the element in the array that matches the key.
- If no match is found, the search returns -1.

Linear Search Animation

- https://yongdanielliang.github.io/animation/web/LinearSearchNew.html

Refer Recorded Lecture



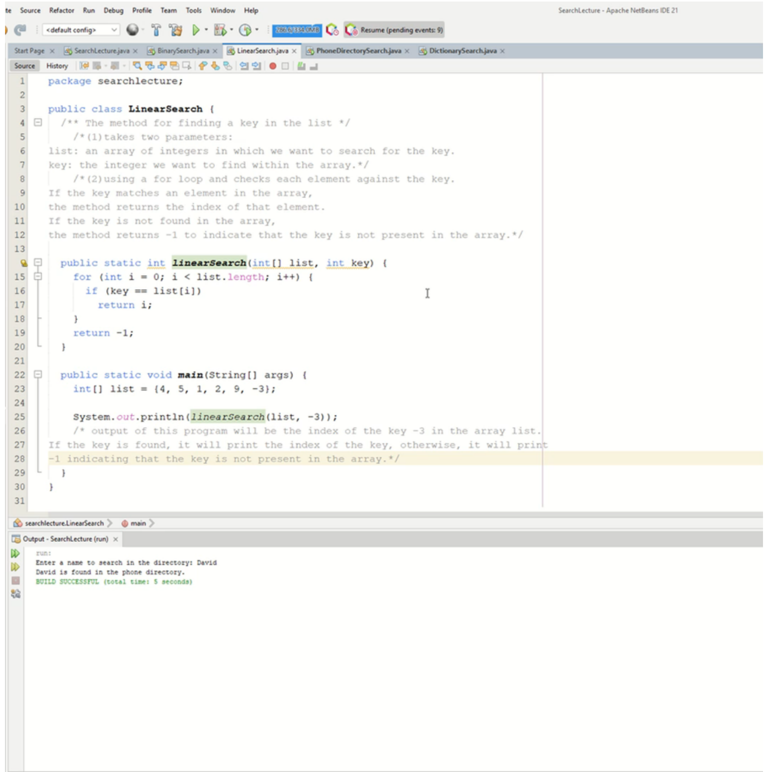
From Idea to Solution

```java
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
    if (key == list[i])
      return i;
  return -1;
}
```

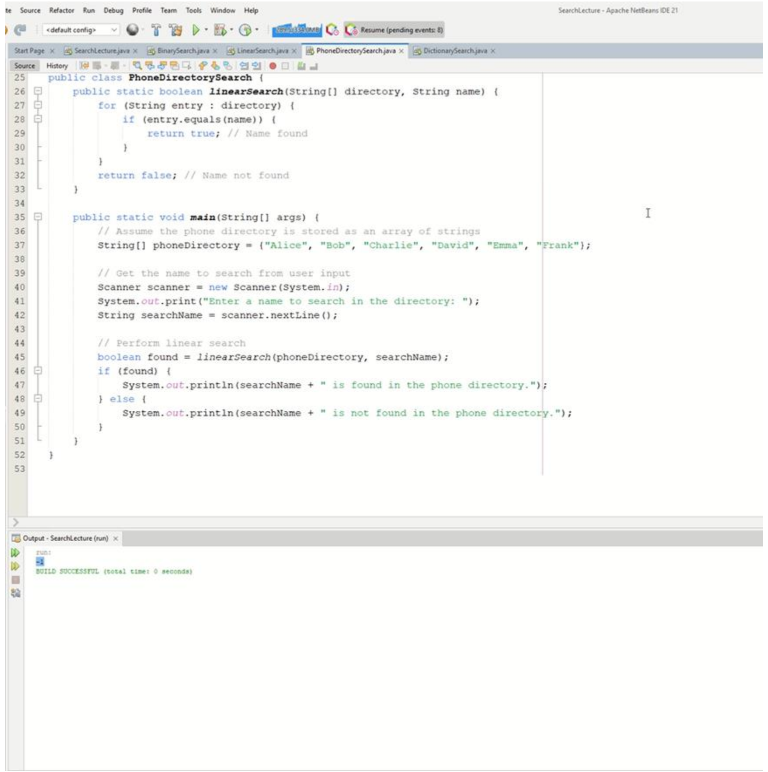
Trace the method

```java
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

Refer Recorded Lecture



Refer Recorded Lecture



Binary Search

- Pre-requisite: the elements in the group must already be ordered. E.g., 2 4 7 10 11 45 50 59 60 66 69 70 79
- First compares the key with the element in the middle of the group.

Binary Search, cont.

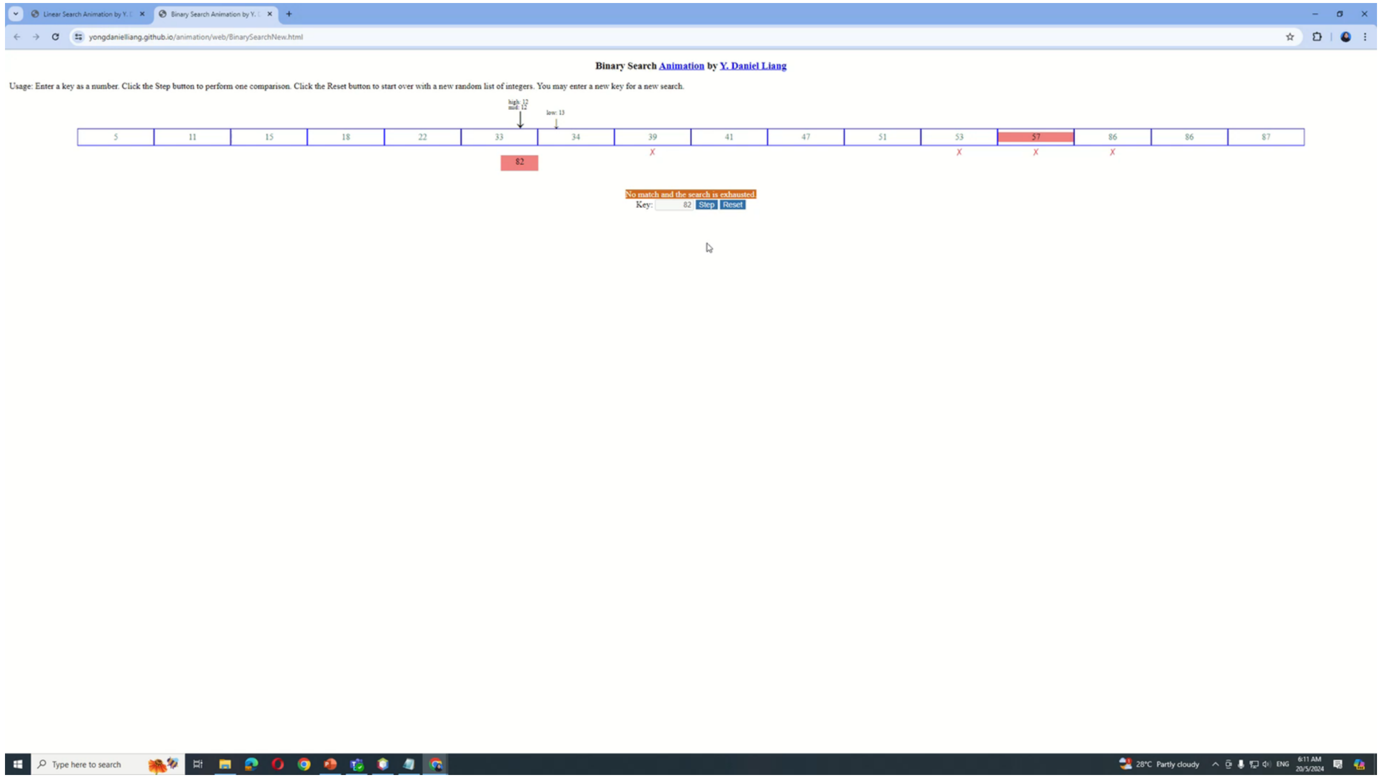
Consider the following three cases:

- If the key is less than the middle element, you only need to search the key in the first half of the group.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the group.

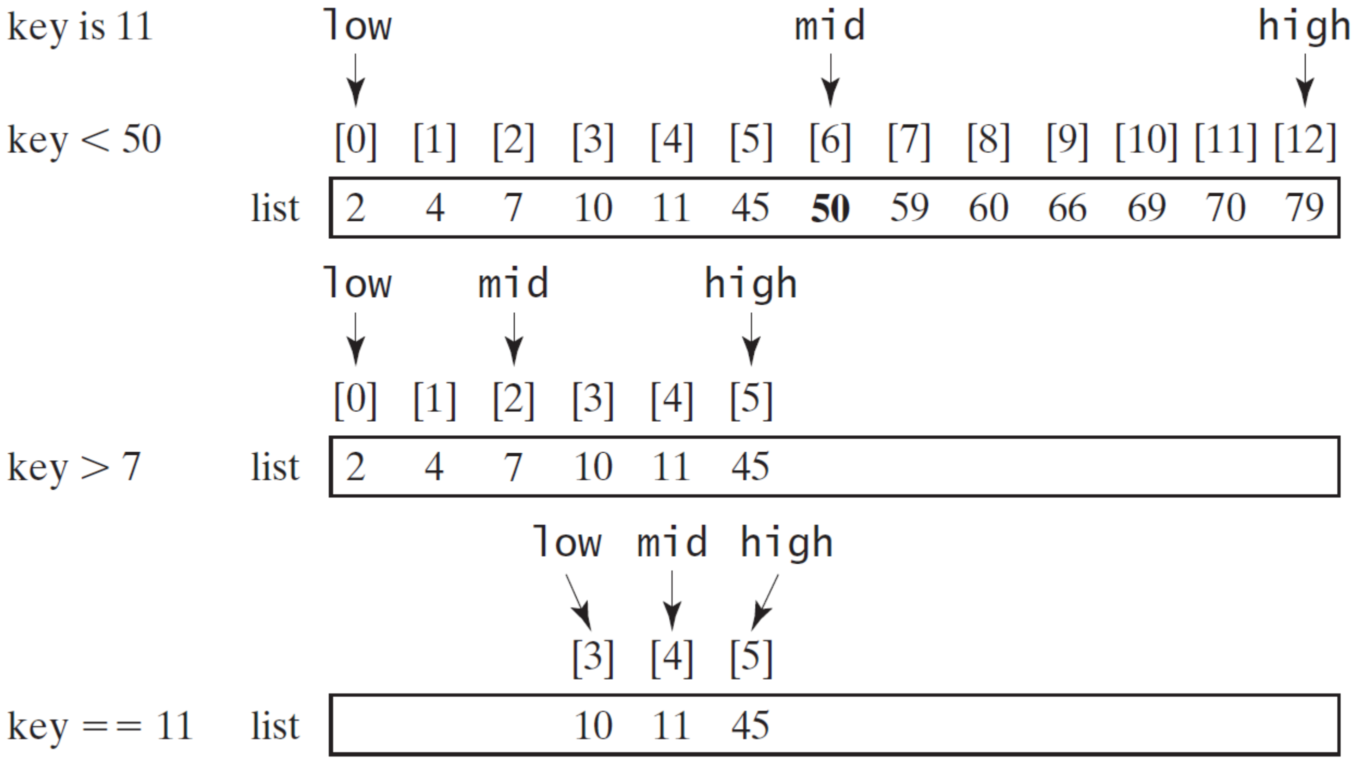
Binary Search

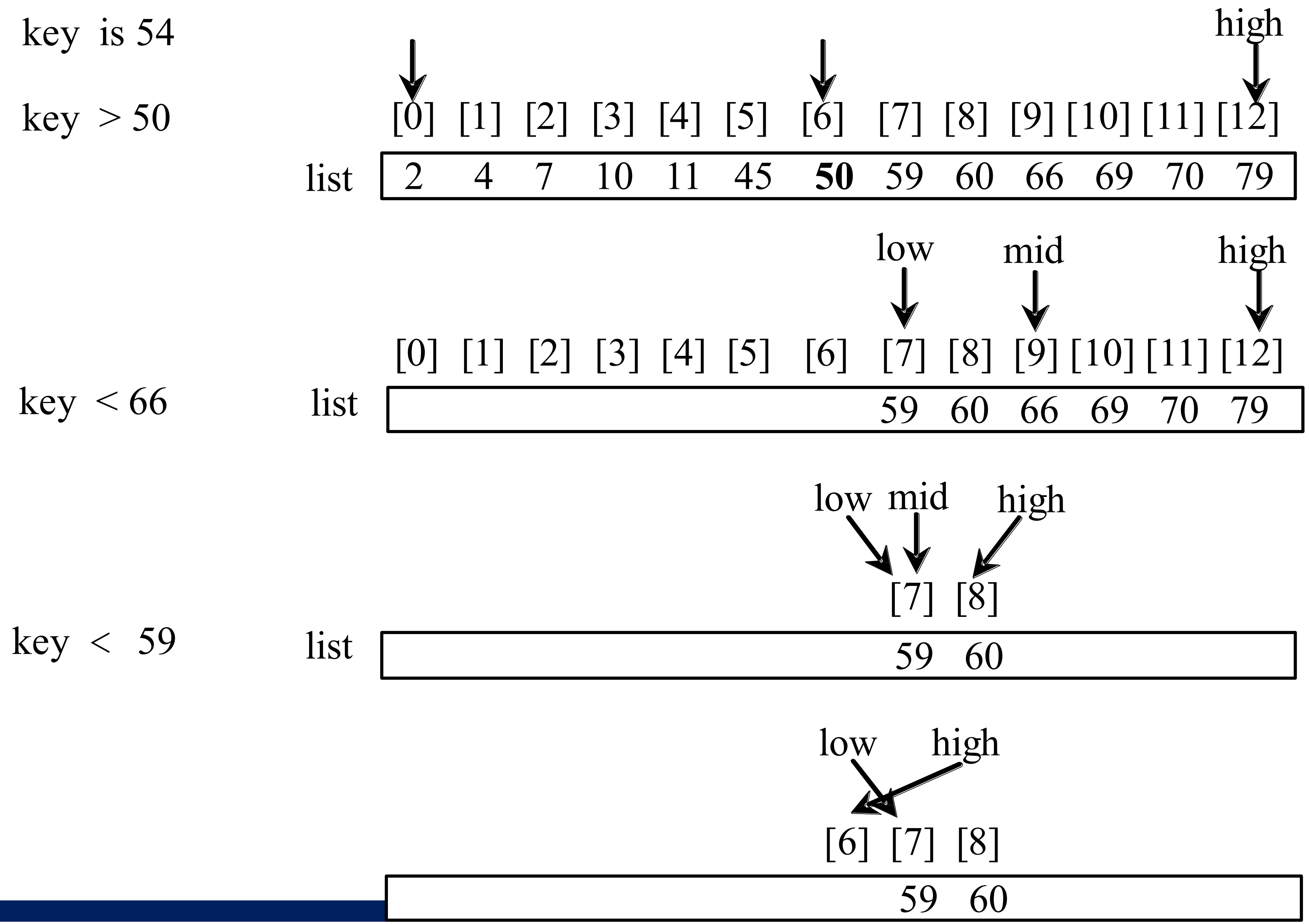
- https://yongdanielliang.github.io/animation/web/BinarySearchNew.html

Refer Recorded Lecture



Binary Search, cont.




key is 54
key > 50
high
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
list
2 4 7 10 11 45 **50** 59 60 66 69 70 79
low mid high
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
key < 66
list
59 60 66 69 70 79
low mid high
[7] [8]
key < 59
list
59 60
low high
[6] [7] [8]
59 60

Binary Search, cont.

- returns the index of the element in the list that matches the search key if it is contained in the list.
- Otherwise, it returns :
  - insertion point - 1.
  - The insertion point is the point at which the key would be inserted into the list.

From Idea to Solution

```java
/** Use binary search to find the key in the list */
public static int binarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;

  while (high >= low) {
    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;
  }

  return -1 - low;
}
```

SORT

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

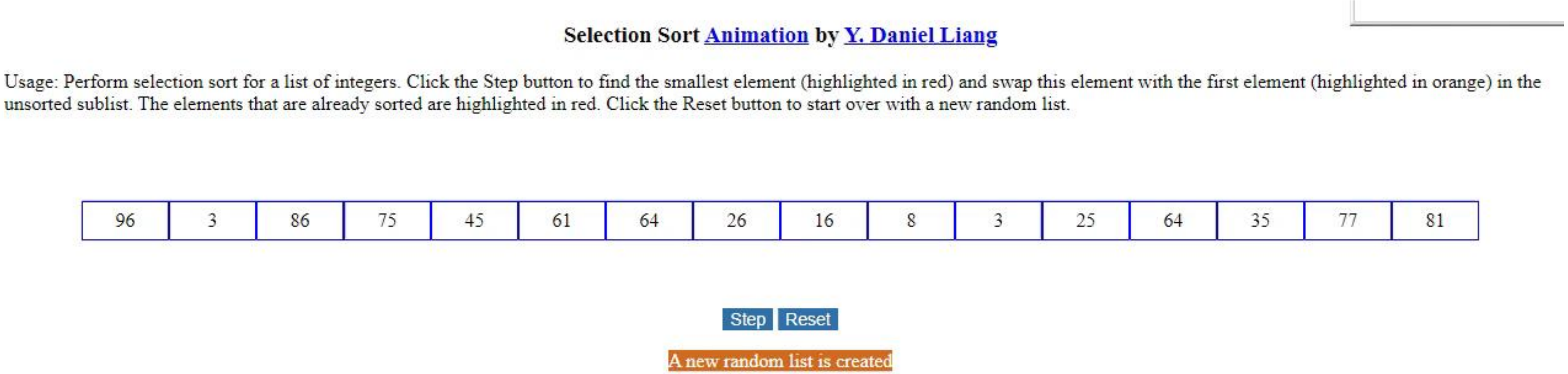
- Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

Selection Sort

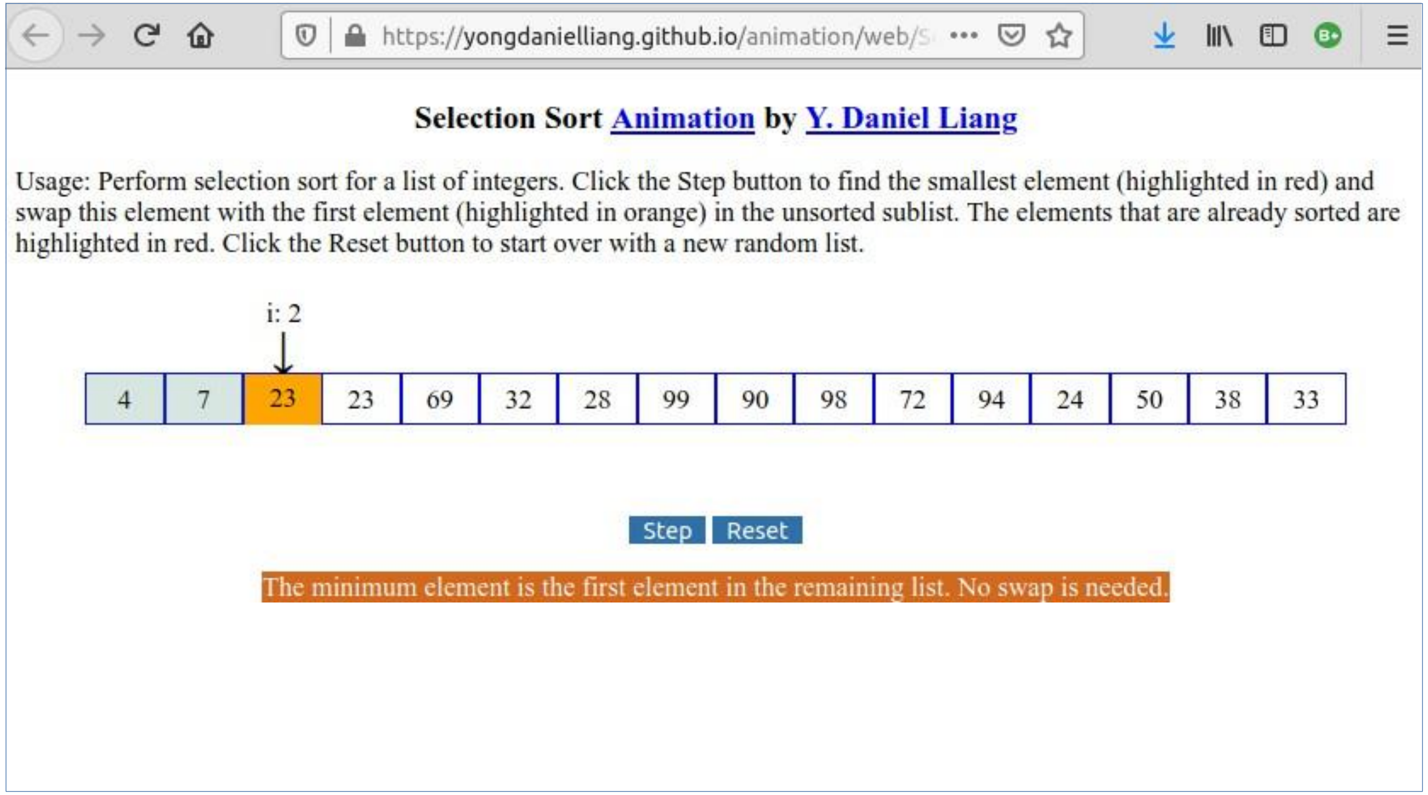
Select 1 (the smallest) and swap it with 2 (the first) in the list.	2	9	5	4	8	1	6	
The number 1 is now in the correct position and thus no longer needs to be considered.	1	9	5	4	8	2	6	Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.
The number 2 is now in the correct position and thus no longer needs to be considered.	1	2	5	4	8	9	6	Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.
The number 4 is now in the correct position and thus no longer needs to be considered.	1	2	4	5	8	9	6	5 is the smallest and in the right position. No swap is necessary.
The number 5 is now in the correct position and thus no longer needs to be considered.	1	2	4	5	8	9	6	Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.
The number 6 is now in the correct position and thus no longer needs to be considered.	1	2	4	5	6	9	8	Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.
The number 8 is now in the correct position and thus no longer needs to be considered.	1	2	4	5	6	8	9	Since there is only one element remaining in the list, the sort is completed.

Selection Sort Animation

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



Refer Recorded Lecture



```
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]
```

```java
/** 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 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] + " ");
  }
}
```

Refer Recorded Lecture

```java
package sortlecture;

public class SelectionSort {
  /** The method for sorting the numbers
     * @param list */
  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] + " ");
  }
}
```

Output - SortLecture (run)

```
run:
-2 1 2 2 3 3 5 6 12 14 BUILD SUCCESSFUL (total time: 0 seconds)
```

Insertion Sort

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

Insertion Sort

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.

2 9 5 4 8 1 6

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

2 9 5 4 8 1 6

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

2 5 9 4 8 1 6

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

2 4 5 9 8 1 6

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

2 4 5 8 9 1 6

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

1 2 4 5 8 9 6

Step 7: The entire list is now sorted.

1 2 4 5 6 8 9

Insertion Sort

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

Insertion Sort Animation by Y. Daniel Liang

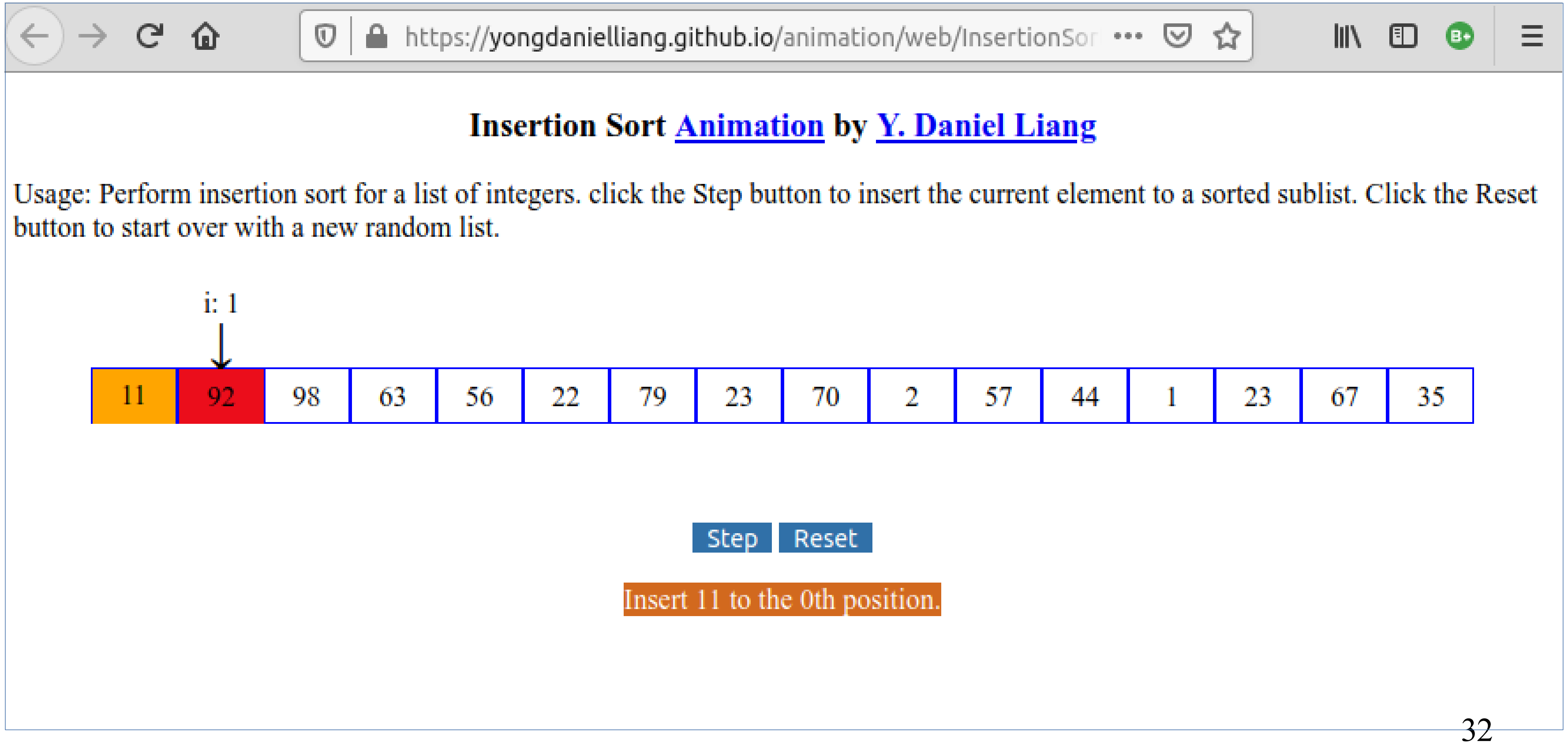
Usage: Perform insertion sort for a list of integers. click the Step button to insert the current element to a sorted sublist. Click the Reset button to start over with a new random list.

92	12	64	18	31	8	90	12	20	20	58	74	14	89	48	45

Step Reset

A new random list is created

Refer Recorded Lecture



Insertion Sort

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

2	9	5	4	8	1	6

2	9	5	4	8	1	6

2	5	9	4	8	1	6

2	4	5	9	8	1	6

2	4	5	8	9	1	6

1	2	4	5	8	9	6

1	2	4	5	6	8	9

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.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	
list	2	5	9	4				Step 1: Save 4 to a temporary variable currentElement
list	2	5		9				Step 2: Move list[2] to list[3]
list	2		5	9				Step 3: Move list[1] to list[2]
list	2	4	5	9				Step 4: Assign currentElement to list[1]

```
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] ...
```

```
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] + " ");
  }
}
```

Refer Recorded Lecture

```java
package sortlecture;

public class InsertionSort {
  /** The method for sorting the numbers
     * @param list */
  public static void insertionSort(int[] list) {
    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. */
      int 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;
    }
  }

  /** A test method
     * @param args */
  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] + " ");
  }
}
```

```
run:
-3.3 -2.0 1.0 2.0 4.5 5.0 BUILD SUCCESSFUL (total time: 0 seconds)
```

Bubble Sort

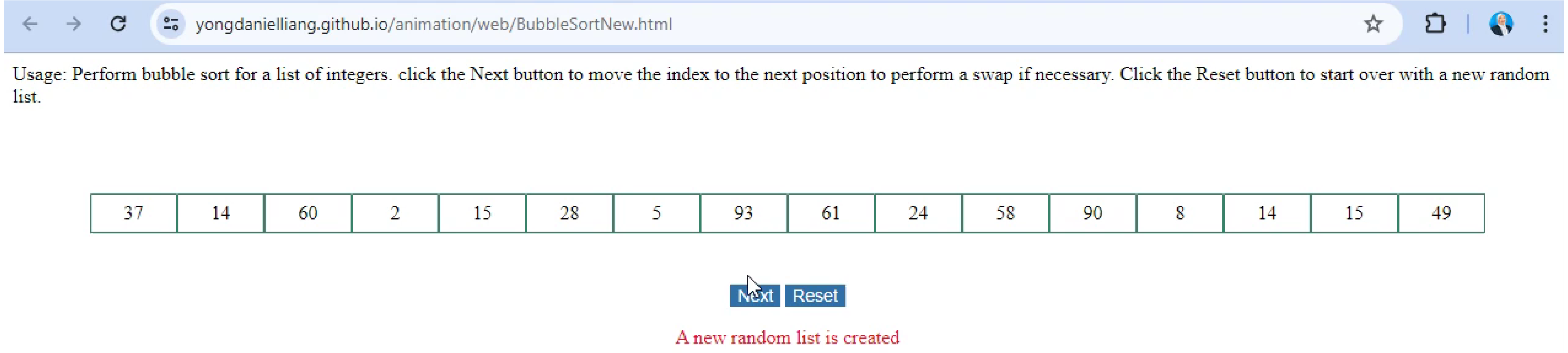
- The bubble sort algorithm makes several passes through the array.
- On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged.
- The smaller values gradually “bubble” their way to the top and the larger values “sink” to the bottom.

Bubble Sort

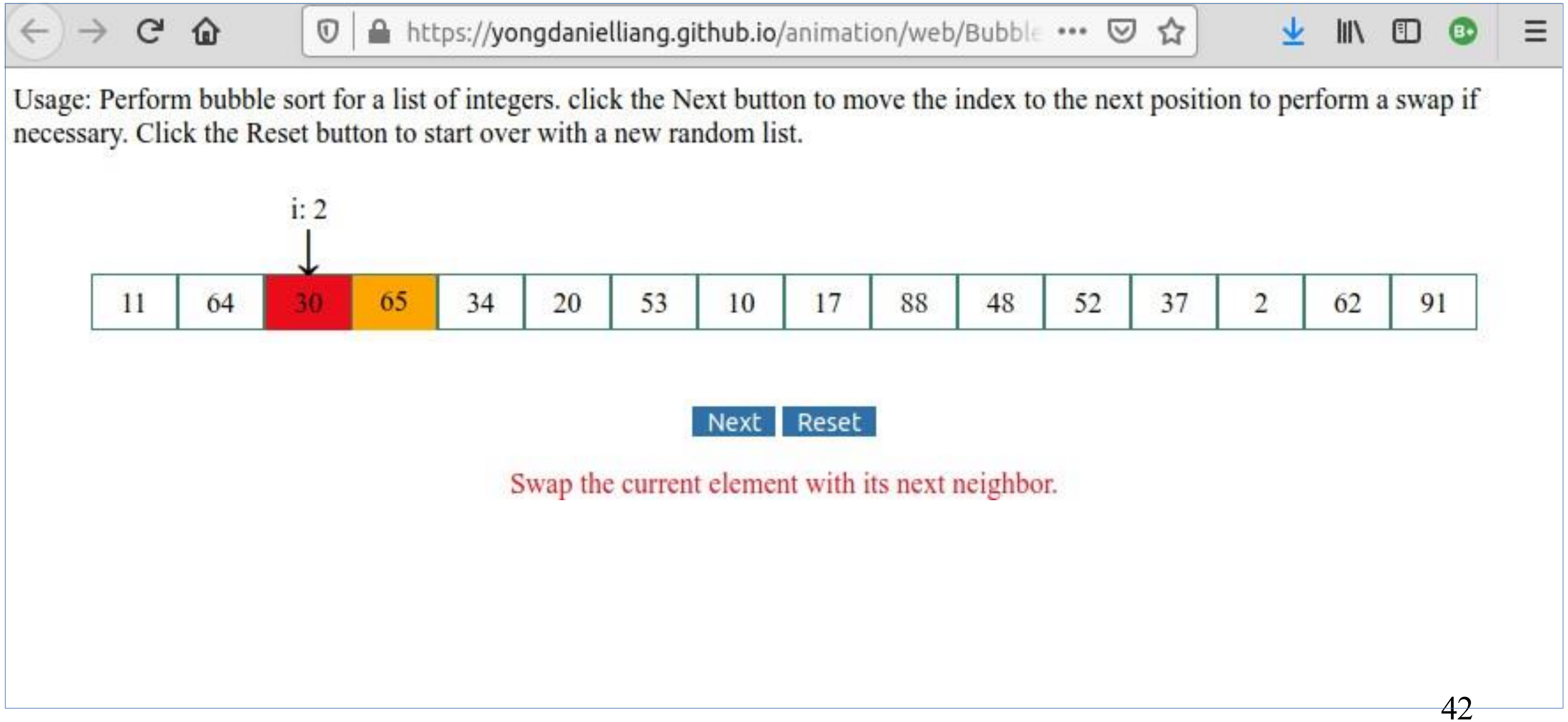
- After the first pass, the last element becomes the largest in the array.
- After the second pass, the second-to-last element becomes the second largest in the array.
- This process is continued until all elements are sorted.

Bubble Sort Animation

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



Refer Recorded Lecture



Bubble Sort

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

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.

```java
boolean needNextPass = true;
for (int k = 1; k < list.length && needNextPass; k++) {
  // Array may be sorted and next pass not needed
  needNextPass = false;
  // Perform the kth pass
  for (int i = 0; i < list.length - k; i++) {
    if (list[i] > list[i + 1]) {
      swap list[i] with list[i + 1];
      needNextPass = true; // Next pass still needed
    }
  }
}
```

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] + " ");
  }
}
```

Refer Recorded Lecture

...va | GenericSelectionSort.java | SortPhoneList.java | Contact.java | InsertionSort.java | BubbleSort.java | MergeSort.java

Source | History

```java

public class BubbleSort {
  /** Bubble sort method
    * @param list */
  public static void bubbleSort(int[] list) {
    boolean needNextPass = true;

    for (int k = 1; k < list.length && needNextPass; k++) {
      // Array may be sorted and next pass not needed
      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; // Next pass still needed
        }
      }
    }
  }

  /** A test method
    * @param args */
  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] + " ");
  }
}
```

sortlecture.BubbleSort > bubbleSort > for (int k = 1; k < list.length && needNextPass; k++) >

Output - SortLecture (run)

```
run:
-2 1 2 2 3 3 5 6 12 14 BUILD SUCCESSFUL (total time: 0 seconds)
```

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);
```

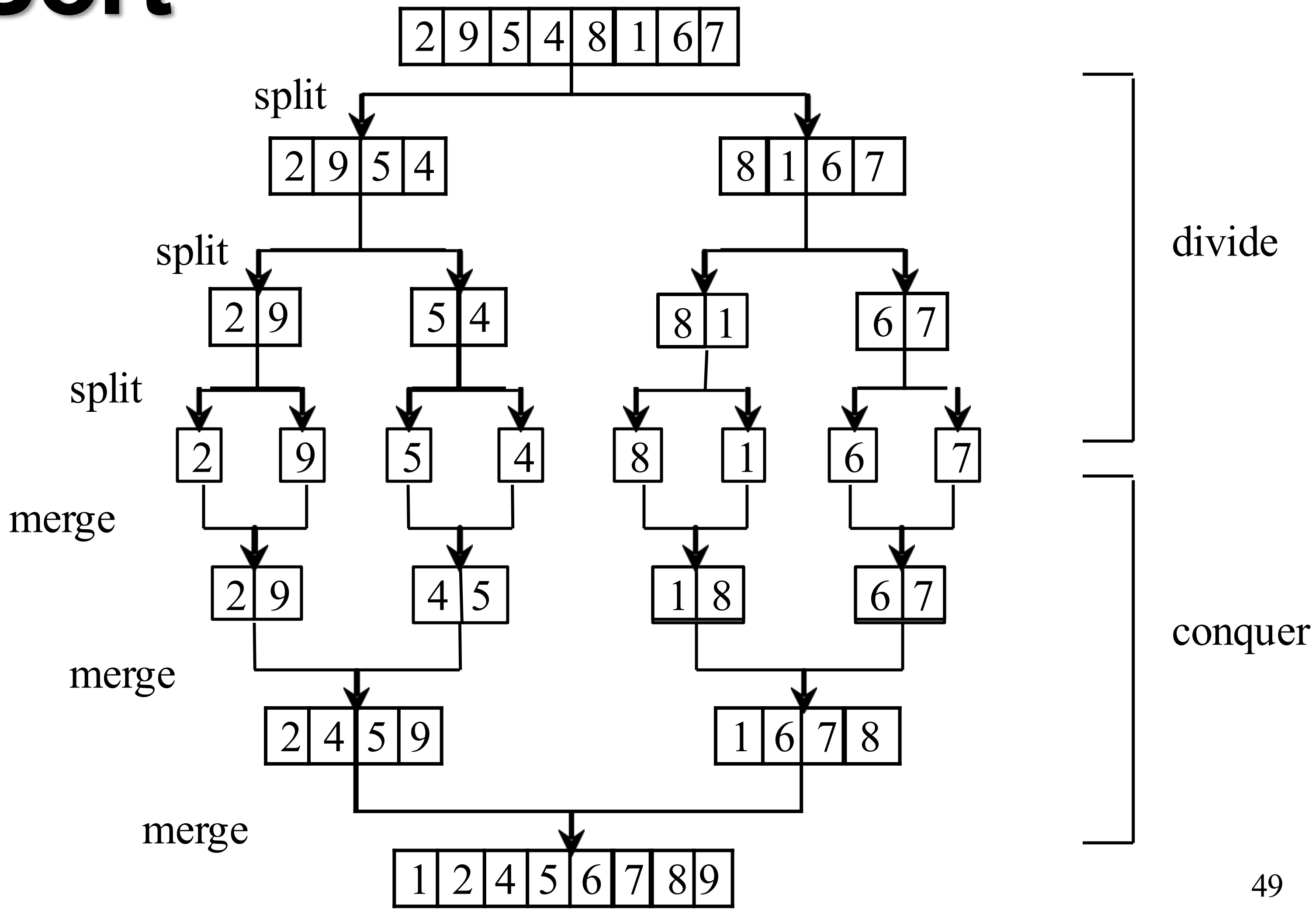
Merge Sort

LISTING 23.5 Merge Sort Algorithm

```
public static void mergeSort(int[] list) {
  if (list.length > 1) {
    mergeSort(list[0 ... list.length / 2]);
    mergeSort(list[list.length / 2 + 1 ... list.length]);
    merge list[0 ... list.length / 2] with
      list[list.length / 2 + 1 ... list.length];
  }
}
```

base condition
sort first half
sort second half
merge two halves

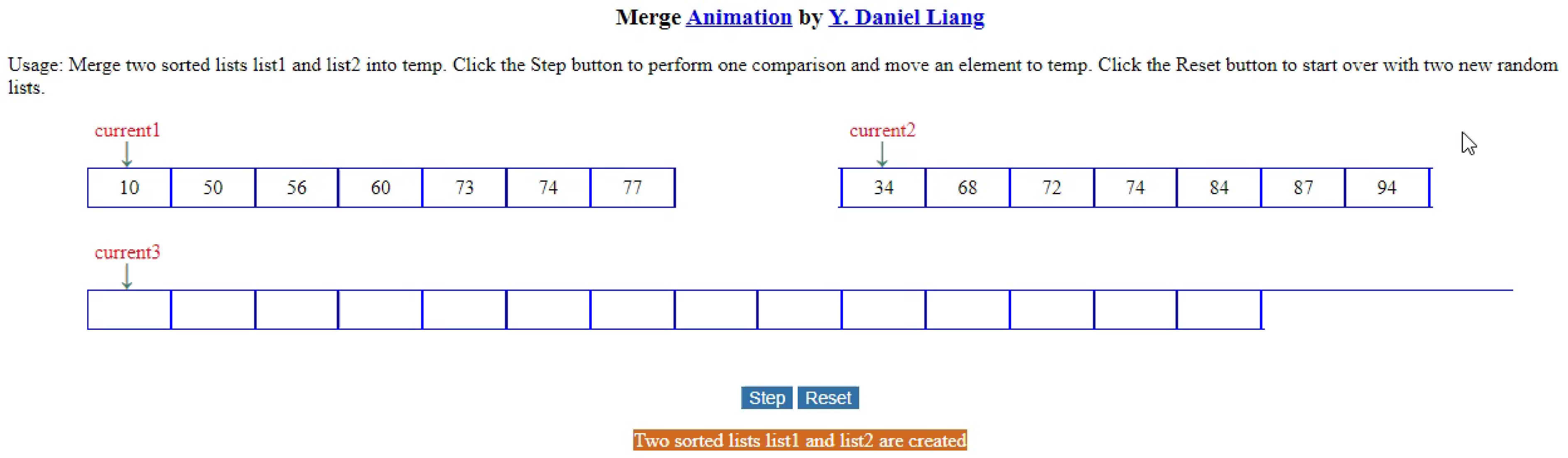
Merge Sort



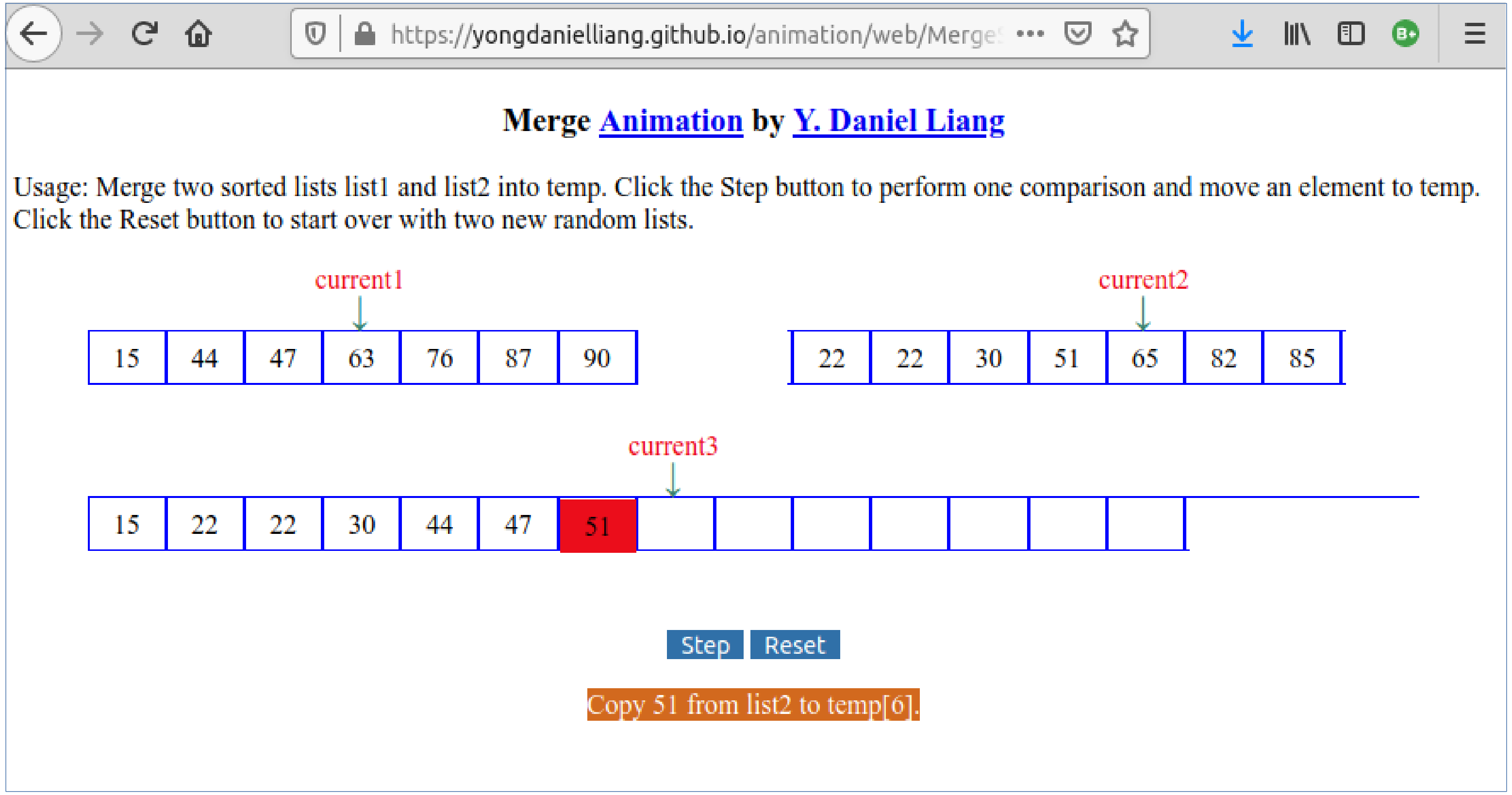
2 9 5 4 8 1 6 7
split
2 9 5 4
8 1 6 7
split
2 9
5 4
8 1
6 7
split
2
9
5
4
8
1
6
7
merge
2 9
4 5
1 8
6 7
merge
2 4 5 9
1 6 7 8
merge
1 2 4 5 6 7 8 9
divide
conquer

Merge Sort Animation

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



Refer Recorded Lecture



LISTING 23.6 MergeSort.java

```java
public class MergeSort {
  /** The method for sorting the numbers */
  public static void mergeSort(int[] list) {
    if (list.length > 1) {
      // Merge sort the first half
      int[] firstHalf = new int[list.length / 2];
      System.arraycopy(list, 0, firstHalf, 0, list.length / 2);
      mergeSort(firstHalf);

      // Merge sort the second half
      int secondHalfLength = list.length - list.length / 2;
      int[] secondHalf = new int[secondHalfLength];
      System.arraycopy(list, list.length / 2,
        secondHalf, 0, secondHalfLength);
      mergeSort(secondHalf);

      // Merge firstHalf with secondHalf into list
      merge(firstHalf, secondHalf, list);
    }
  }

```

base case

sort first half

sort second half

merge two halves

```java
    /** Merge two sorted lists */
    public static void merge(int[] list1, int[] list2, int[] temp) {
      int current1 = 0; // Current index in list1
      int current2 = 0; // Current index in list2
      int current3 = 0; // Current index in temp

      while (current1 < list1.length && current2 < list2.length) {
        if (list1[current1] < list2[current2])
          temp[current3++] = list1[current1++];
        else
          temp[current3++] = list2[current2++];
      }

      while (current1 < list1.length)
        temp[current3++] = list1[current1++];

      while (current2 < list2.length)
        temp[current3++] = list2[current2++];
    }

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

list1 to temp

list2 to temp

rest of list1 to temp

rest of list2 to temp

Refer Recorded Lecture

```java
package sortlecture;

public class MergeSort {
  /** The method for sorting the numbers
    * @param list */
  public static void mergeSort(int[] list) {
    if (list.length > 1) {
      // Merge sort the first half
      int[] firstHalf = new int[list.length / 2];
      System.arraycopy(list, 0, firstHalf, 0, list.length / 2);
      mergeSort(firstHalf);

      // Merge sort the second half
      int secondHalfLength = list.length - list.length / 2;
      int[] secondHalf = new int[secondHalfLength];
      System.arraycopy(list, list.length / 2,
        secondHalf, 0, secondHalfLength);
      mergeSort(secondHalf);

      // Merge firstHalf with secondHalf into list
      merge(firstHalf, secondHalf, list);
    }
  }

  /** Merge two sorted lists
    * @param list1
    * @param list2
    * @param temp */
  public static void merge(int[] list1, int[] list2, int[] temp) {
    int current1 = 0; // Current index in list1
    int current2 = 0; // Current index in list2
    int current3 = 0; // Current index in temp

    while (current1 < list1.length && current2 < list2.length) {
      if (list1[current1] < list2[current2])
        temp[current3++] = list1[current1++];
      else
        temp[current3++] = list2[current2++];
    }
```

```
run:
-2 1 2 2 3 3 5 6 12 14 BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
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
```

```java
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;
    }
}
```

Modify the following SelectionSort to be a generic method

```java
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;
    }
  }
}
```

```java
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

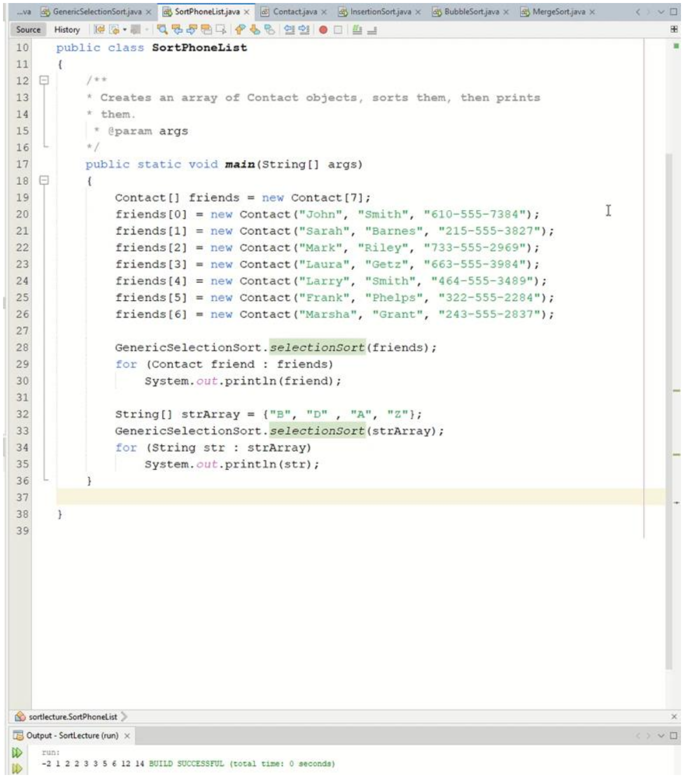
Comparable

```java
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;
   }
}
```

Refer Recorded Lecture



Sort Algorithms

1. **Selection Sort**: Selection Sort is a **simple** sorting algorithm that repeatedly selects the minimum (or maximum) element from the unsorted portion of the array and places it at the beginning (or end) of the sorted portion. It has a time complexity of **O(n^2)** on average and is not suitable for large datasets due to its inefficiency.
2. **Insertion Sort**: Insertion Sort is another **simple** sorting algorithm that builds the final sorted array one element at a time. It iterates through the array and, for each element, it finds the correct position to insert it into the already sorted part of the array. It has a time complexity of O(n^2) on average but performs well on small datasets and nearly sorted arrays.

Sort Algorithms

3. **Bubble Sort**: Bubble Sort is a **simple** sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. It has a time complexity of **O(n^2)** on average and is mostly used for educational purposes or for sorting small datasets due to its inefficiency.
4. **Merge Sort**: Merge Sort is a **divide-and-conquer** algorithm that divides the input array into two halves, sorts each half recursively, and then merges the sorted halves to produce a single sorted array. It has a time complexity of **O(n log n)** on average, making it more efficient than the previous sorting algorithms for large datasets. It's a stable sorting algorithm, meaning it preserves the relative order of equal elements.

THANK YOU