

WIA 1002 DATA STRUCTURE

SEM 2, SESSION 2024/205

NURUL JAPAR

nuruljapar@um.edu.my

HOO WAI LAM

wlhoo@um.edu.my

Home of the Bright, Land of the Brave
Di Sini Bermulanya Pintar, Tanah Tumpahnya Berani

www.um.edu.my



SEARCH & SORT

Home of the Bright, Land of the Brave
Di Sini Bermulanya Pintar, Tanah Tumpahnya Berani



www.um.edu.my



[universityofmalaya](https://www.facebook.com/universityofmalaya)



[unimalaya](https://www.instagram.com/unimalaya)



[uniofmalaya](https://www.youtube.com/uniofmalaya)



UNIVERSITI
MALAYA

SEARCH

Home of the Bright, Land of the Brave
Di Sini Bermulanya Pintar, Tanah Tumpahnya Berani



www.um.edu.my



[universityofmalaya](https://www.facebook.com/universityofmalaya)



[unimalaya](https://www.instagram.com/unimalaya)



[uniofmalaya](https://www.youtube.com/uniofmalaya)



UNIVERSITI
MALAYA

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

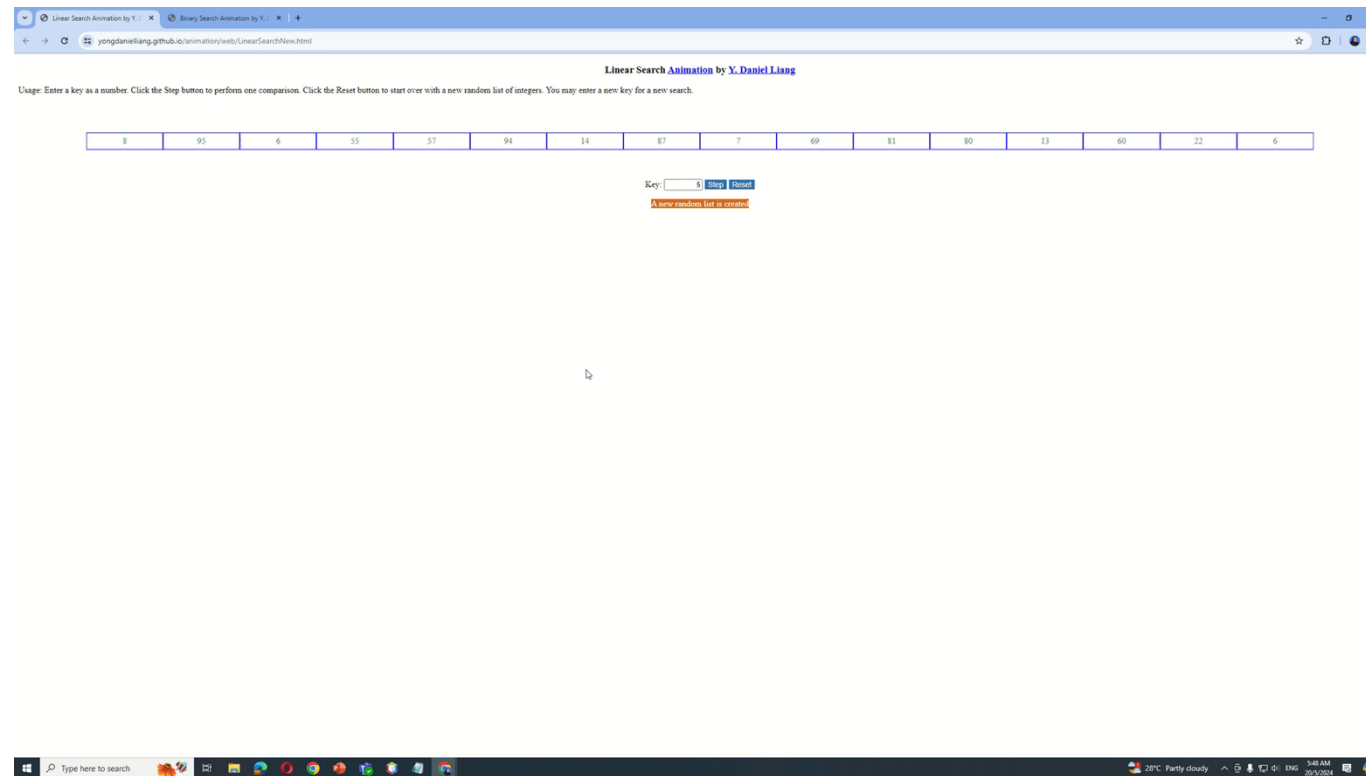
Linear Search

- 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

```
/** 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;  
}
```

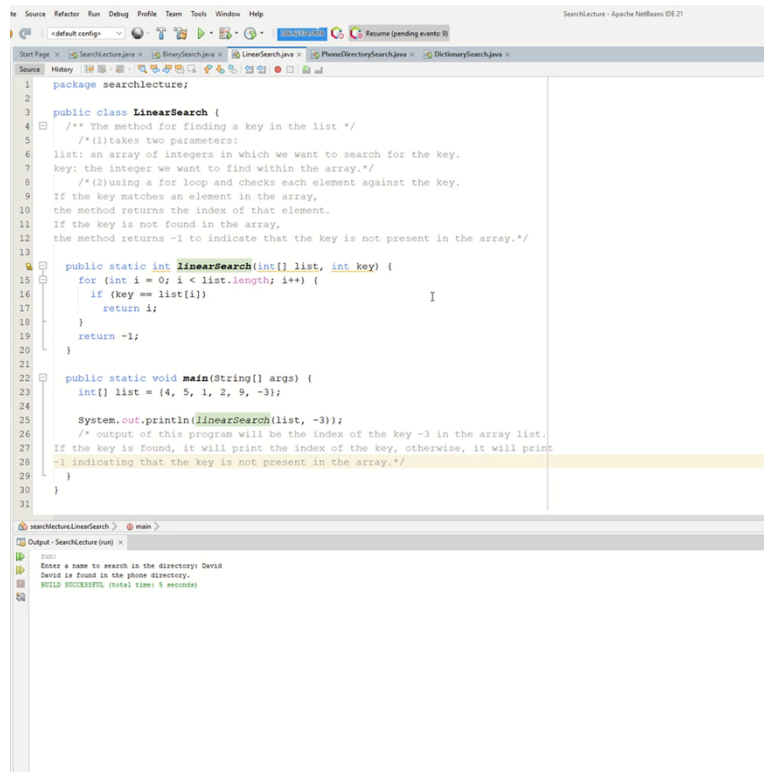
Trace the method

```
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
```

8



Refer Recorded Lecture



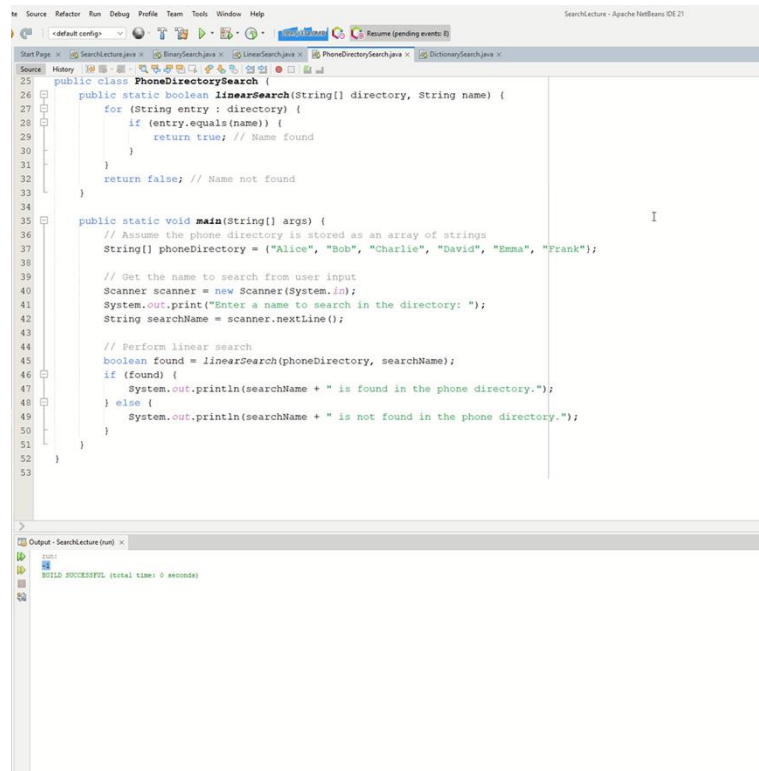
```
1 package searchlecture;
2
3 public class LinearSearch {
4     /** The method for finding a key in the list */
5     /** (1) takes two parameters:
6     list: an array of integers in which we want to search for the key.
7     key: the integer we want to find within the array.*/
8     /** (2) using a for loop and checks each element against the key.
9     If the key matches an element in the array,
10    the method returns the index of that element.
11    If the key is not found in the array,
12    the method returns -1 to indicate that the key is not present in the array.*/
13
14    public static int linearSearch(int[] list, int key) {
15        for (int i = 0; i < list.length; i++) {
16            if (key == list[i]) {
17                return i;
18            }
19        }
20        return -1;
21    }
22
23    public static void main(String[] args) {
24        int[] list = {4, 5, 1, 2, 9, -3};
25
26        System.out.println(linearSearch(list, -3));
27        /* output of this program will be the index of the key -3 in the array list.
28        If the key is found, it will print the index of the key, otherwise, it will print
29        -1 indicating that the key is not present in the array.*/
30    }
31 }
```

searchlecture.LinearSearch > main

Output: SearchLecture [out] x

Enter a name to search in the directory: David
David is found in the phone directory.
BUILD SUCCESSFUL (total time: 5 seconds)

Refer Recorded Lecture



The screenshot shows an IDE window titled "SearchLecture - Apache NetBeans IDE 21". The main editor displays the source code for a Java application. The code defines a `PhoneDirectorySearch` class with a `linearSearch` method and a `main` method. The `linearSearch` method iterates through a directory array to find a specific name. The `main` method initializes a phone directory, prompts the user for a search name, and uses the `linearSearch` method to check if the name is in the directory. The output window at the bottom shows the successful execution of the program.

```
25 public class PhoneDirectorySearch {
26     public static boolean linearSearch(String[] directory, String name) {
27         for (String entry : directory) {
28             if (entry.equals(name)) {
29                 return true; // Name found
30             }
31         }
32         return false; // Name not found
33     }
34
35     public static void main(String[] args) {
36         // Assume the phone directory is stored as an array of strings
37         String[] phoneDirectory = {"Alice", "Bob", "Charlie", "David", "Emma", "Frank"};
38
39         // Get the name to search from user input
40         Scanner scanner = new Scanner(System.in);
41         System.out.print("Enter a name to search in the directory: ");
42         String searchName = scanner.nextLine();
43
44         // Perform linear search
45         boolean found = linearSearch(phoneDirectory, searchName);
46         if (found) {
47             System.out.println(searchName + " is found in the phone directory.");
48         } else {
49             System.out.println(searchName + " is not found in the phone directory.");
50         }
51     }
52 }
53
```

Output: SearchLecture [out] x

```
Enter
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.

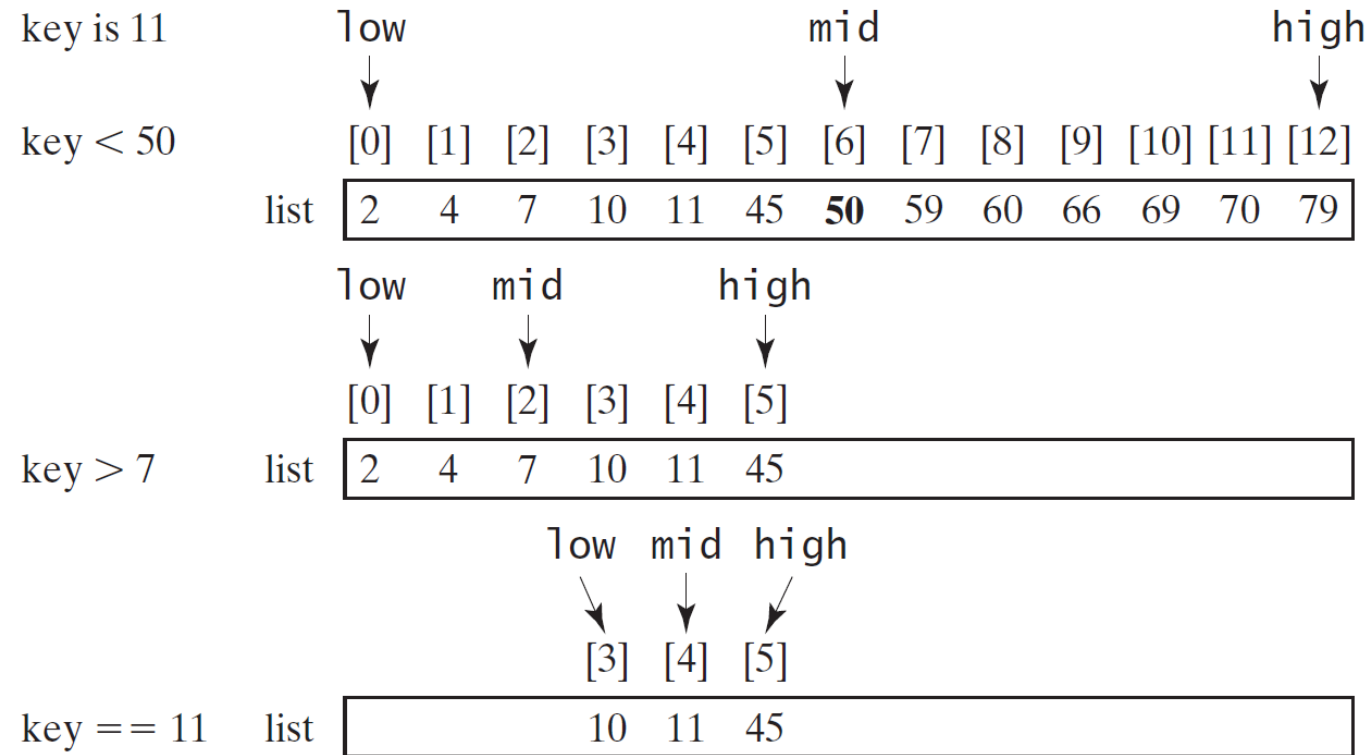
Consider the following three cases:

- 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

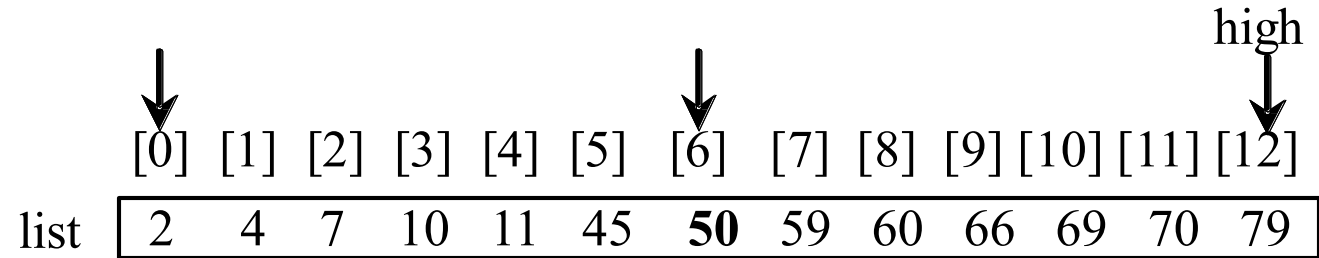
- <https://yongdanielliang.github.io/animation/web/BinarySearchNew.html>

Binary Search, cont.

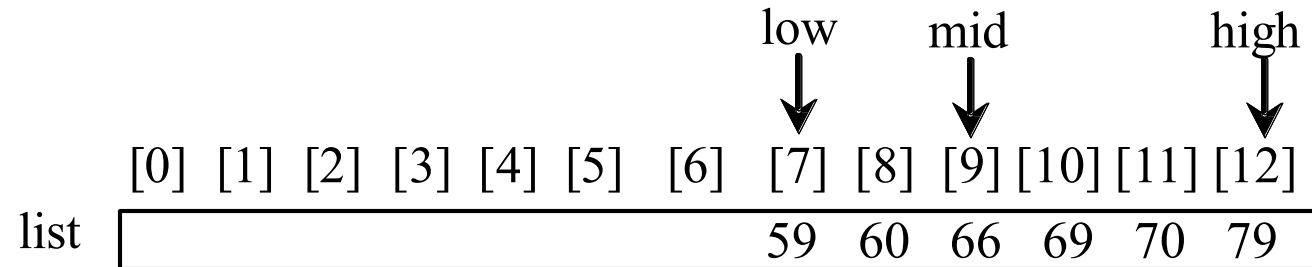


key is 54

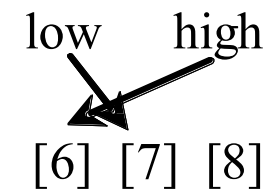
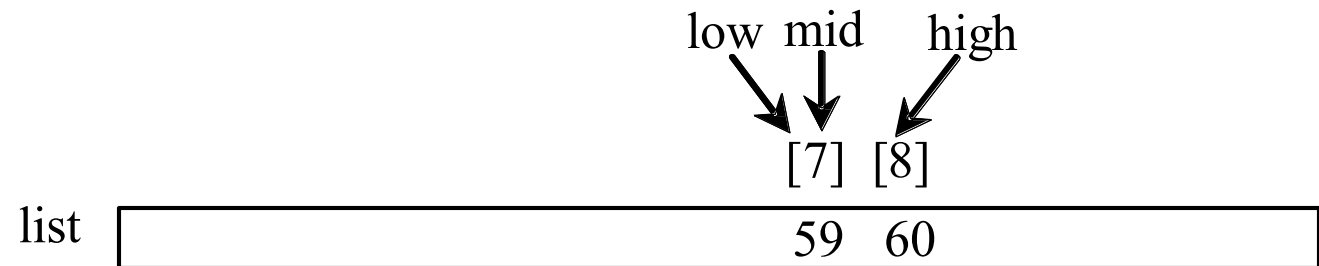
key > 50



key < 66



key < 59



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

```
/** 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

Home of the Bright, Land of the Brave
Di Sini Bermulanya Pintar, Tanah Tumpahnya Berani



www.um.edu.my



[universityofmalaya](https://www.facebook.com/universityofmalaya)



[unimalaya](https://www.instagram.com/unimalaya)



[uniofmalaya](https://www.youtube.com/uniofmalaya)



UNIVERSITI
MALAYA

Sorting

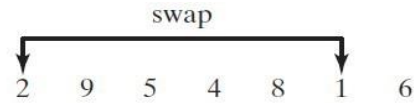
- Sorting, like searching, is also a common task in computer programming.
- Some sorting algorithms:
 1. Selection sort
 2. Insertion sort
 3. Bubble sort
 4. Merge Sort

Selection Sort

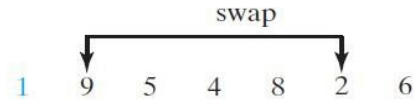
- Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

Selection Sort

Select 1 (the smallest) and swap it with 2 (the first) in the list.

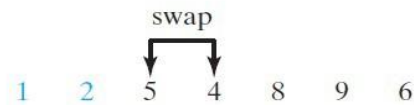


The number 1 is now in the correct position and thus no longer needs to be considered.



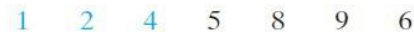
Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.

The number 2 is now in the correct position and thus no longer needs to be considered.



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.



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.



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.



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.



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>

Selection Sort [Animation](#) by [Y. Daniel Liang](#)

Usage: Perform selection sort for a list of integers. Click the Step button to find the smallest element (highlighted in red) and swap this element with the first element (highlighted in orange) in the unsorted sublist. The elements that are already sorted are highlighted in red. Click the Reset button to start over with a new random list.

96	3	86	75	45	61	64	26	16	8	3	25	64	35	77	81
----	---	----	----	----	----	----	----	----	---	---	----	----	----	----	----

Step Reset

A new random list is created

Refer Recorded Lecture

Selection Sort [Animation](#) by [Y. Daniel Liang](#)

Usage: Perform selection sort for a list of integers. Click the Step button to find the smallest element (highlighted in red) and swap this element with the first element (highlighted in orange) in the unsorted sublist. The elements that are already sorted are highlighted in red. Click the Reset button to start over with a new random list.

i: 2

4	7	23	23	69	32	28	99	90	98	72	94	24	50	38	33
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Step Reset

The minimum element is the first element in the remaining list. No swap is needed.


```
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;  
        }  
    }  
}
```

```

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;
                }
            }

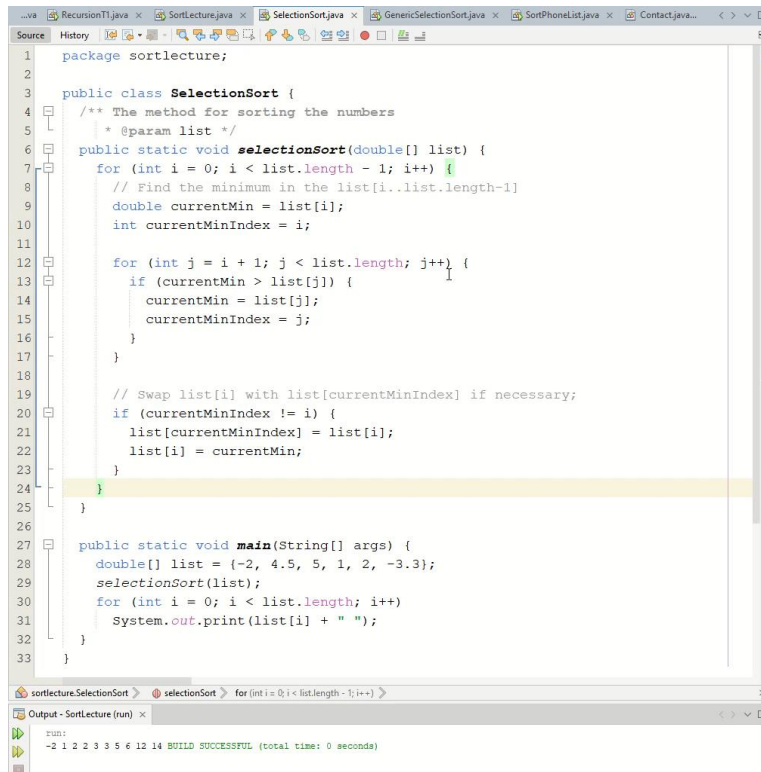
            // Swap list[i] with list[currentMinIndex] if necessary;
            if (currentMinIndex != i) {
                list[currentMinIndex] = list[i];
                list[i] = currentMin;
            }
        }
    }

    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] + " ");
    }
}

```

Liang, Introduction to Java Programming, Tenth Edition,
Global Edition. © Pearson Education Limited 2015

Refer Recorded Lecture



```
1 package sortlecture;
2
3 public class SelectionSort {
4     /** The method for sorting the numbers
5      * @param list */
6     public static void selectionSort(double[] list) {
7         for (int i = 0; i < list.length - 1; i++) {
8             // Find the minimum in the list[i..list.length-1]
9             double currentMin = list[i];
10            int currentMinIndex = i;
11
12            for (int j = i + 1; j < list.length; j++) {
13                if (currentMin > list[j]) {
14                    currentMin = list[j];
15                    currentMinIndex = j;
16                }
17            }
18
19            // Swap list[i] with list[currentMinIndex] if necessary;
20            if (currentMinIndex != i) {
21                list[currentMinIndex] = list[i];
22                list[i] = currentMin;
23            }
24        }
25    }
26
27    public static void main(String[] args) {
28        double[] list = {-2, 4.5, 5, 1, 2, -3.3};
29        selectionSort(list);
30        for (int i = 0; i < list.length; i++)
31            System.out.print(list[i] + " ");
32    }
33 }
```

sortlecture.SelectionSort > selectionSort > for (int i = 0; i < list.length - 1; i++) >

Output - SortLecture (run) x

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.

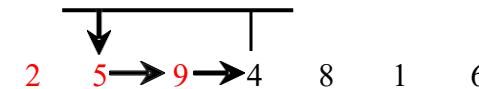
Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 into the sublist.



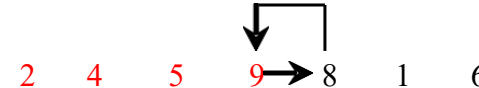
Step 2: The sorted sublist is {2, 9}. Insert 5 into the sublist.



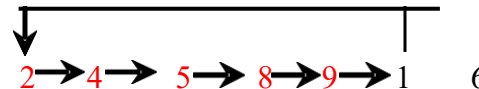
Step 3: The sorted sublist is {2, 5, 9}. Insert 4 into the sublist.



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

- <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 [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.

i: 1

↓

11	92	98	63	56	22	79	23	70	2	57	44	1	23	67	35
----	----	----	----	----	----	----	----	----	---	----	----	---	----	----	----

Step Reset

Insert 11 to the 0th position.

32

Insertion Sort

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

2	9	5	4	8	1	6
---	---	---	---	---	---	---

2	5	9	4	8	1	6
---	---	---	---	---	---	---

2	4	5	8	9	1	6
---	---	---	---	---	---	---

1	2	4	5	6	8	9
---	---	---	---	---	---	---

2	9	5	4	8	1	6
---	---	---	---	---	---	---

2	4	5	9	8	1	6
---	---	---	---	---	---	---

1	2	4	5	8	9	6
---	---	---	---	---	---	---

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.

list

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2	5	9	4			

Step 1: Save 4 to a temporary variable currentElement

list

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2	5		9			

Step 2: Move list[2] to list[3]

list

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2		5	9			

Step 3: Move list[1] to list[2]

list

[0]	[1]	[2]	[3]	[4]	[5]	[6]
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;
```

```

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

```
...va SelectionSort.java x GenericSelectionSort.java x SortPhoneList.java x Contact.java x InsertionSort.java x BubbleSort.java...
Source History
package sortlecture;

1
2
3 public class InsertionSort {
4     /** The method for sorting the numbers
5      * @param list */
6     public static void insertionSort(int[] list) {
7         for (int i = 1; i < list.length; i++) {
8             /** insert list[i] into a sorted sublist list[0..i-1] so that
9              * list[0..i] is sorted. */
10            int currentElement = list[i];
11            int k;
12            for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
13                list[k + 1] = list[k];
14            }
15
16            // Insert the current element into list[k+1]
17            list[k + 1] = currentElement;
18        }
19    }
20
21    /** A test method
22     * @param args */
23    public static void main(String[] args) {
24        int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
25        insertionSort(list);
26        for (int i = 0; i < list.length; i++)
27            System.out.print(list[i] + " ");
28    }
29 }
```

sortlecture.InsertionSort > main > list

Output - SortLecture (run) x

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

Bubble Sort

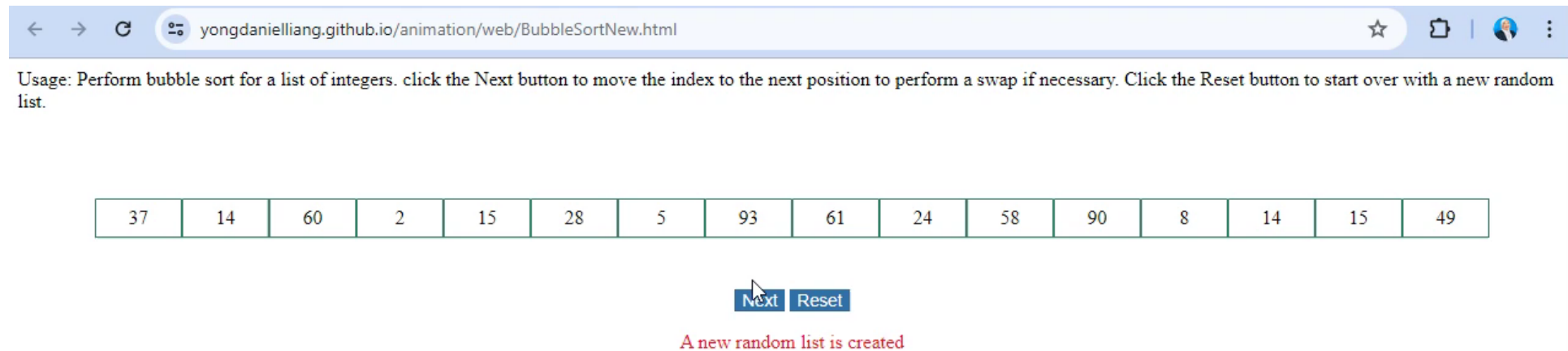
- The bubble sort algorithm makes several passes through the array.
- On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged.
- The smaller values gradually “bubble” their way to the top and the larger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

Usage: Perform bubble sort for a list of integers. click the Next button to move the index to the next position to perform a swap if necessary. Click the Reset button to start over with a new random list.

i: 2

11	64	30	65	34	20	53	10	17	88	48	52	37	2	62	91
----	----	----	----	----	----	----	----	----	----	----	----	----	---	----	----

Next Reset

Swap the current element with its next neighbor.

42



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; ← By default each iteration will set needNextPass to 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 }

```

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

```

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;
                }
            }
        }
    }

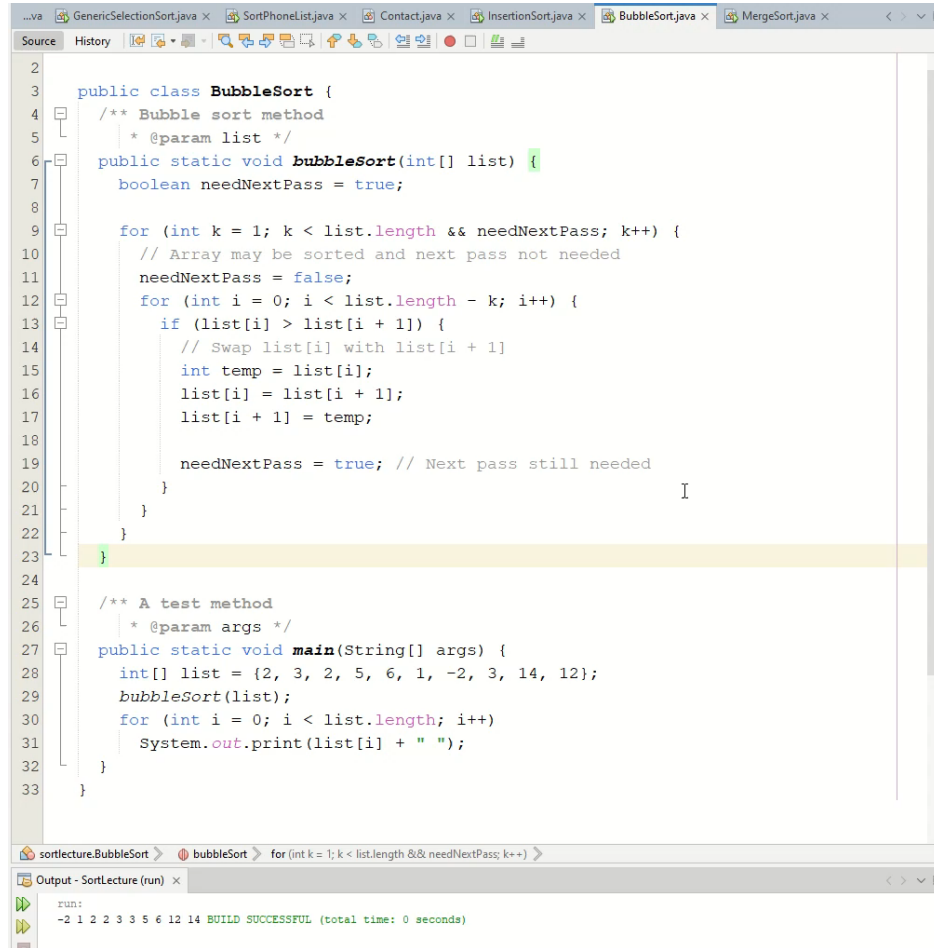
    /** A test method */
    public static void main(String[] args) {
        int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
        bubbleSort(list);
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
    }
}

```

Liang, Introduction to Java Programming, Tenth Edition,
 Global Edition. © Pearson Education Limited 2015



Refer Recorded Lecture



```
2
3 public class BubbleSort {
4     /** Bubble sort method
5      * @param list */
6     public static void bubbleSort(int[] list) {
7         boolean needNextPass = true;
8
9         for (int k = 1; k < list.length && needNextPass; k++) {
10             // Array may be sorted and next pass not needed
11             needNextPass = false;
12             for (int i = 0; i < list.length - k; i++) {
13                 if (list[i] > list[i + 1]) {
14                     // Swap list[i] with list[i + 1]
15                     int temp = list[i];
16                     list[i] = list[i + 1];
17                     list[i + 1] = temp;
18
19                     needNextPass = true; // Next pass still needed
20                 }
21             }
22         }
23     }
24
25     /** A test method
26     * @param args */
27     public static void main(String[] args) {
28         int[] list = {2, 3, 2, 5, 6, 1, -2, 3, 14, 12};
29         bubbleSort(list);
30         for (int i = 0; i < list.length; i++)
31             System.out.print(list[i] + " ");
32     }
33 }
```

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

Output - SortLecture (run) x

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);
```

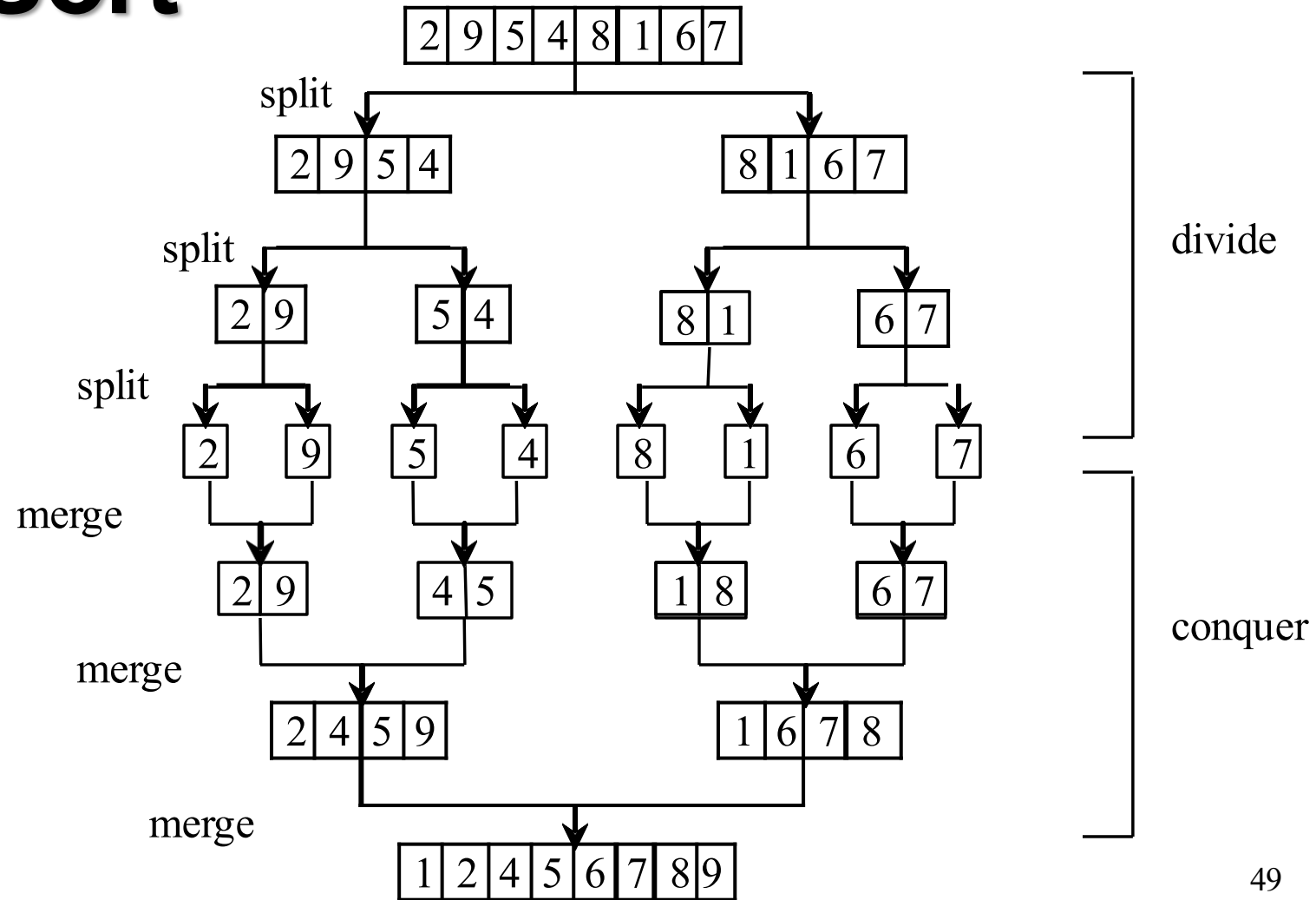

Merge Sort

LISTING 23.5 Merge Sort Algorithm

```
1 public static void mergeSort(int[] list) {  
2     if (list.length > 1) {  
3         mergeSort(list[0 ... list.length / 2]);  
4         mergeSort(list[list.length / 2 + 1 ... list.length]);  
5         merge list[0 ... list.length / 2] with  
6         list[list.length / 2 + 1 ... list.length];  
7     }  
8 }
```

base condition
sort first half
sort second half
merge two halves

Merge Sort

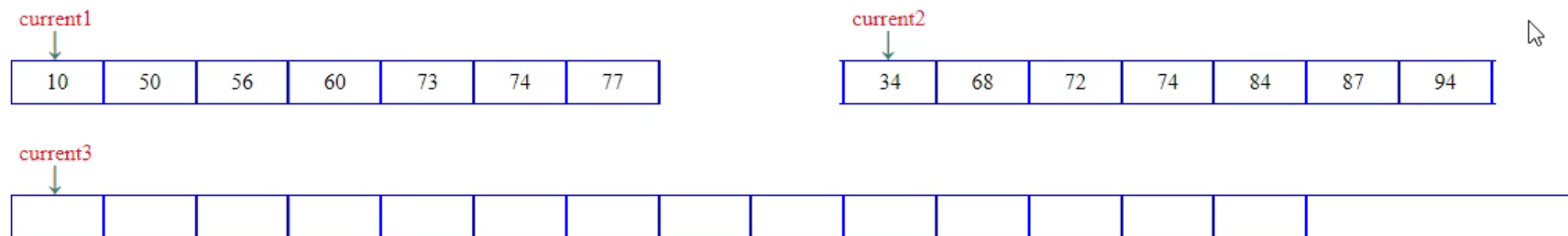


Merge Sort Animation

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

Merge Animation by Y. Daniel Liang

Usage: Merge two sorted lists list1 and list2 into temp. Click the Step button to perform one comparison and move an element to temp. Click the Reset button to start over with two new random lists.



Step Reset

Two sorted lists list1 and list2 are created

30

Refer Recorded Lecture

← → ↻ 🏠 🔒 https://yongdanielliang.github.io/animation/web/MergeSort ... 📄 ⚙️ ⌵

Merge [Animation](#) by [Y. Daniel Liang](#)

Usage: Merge two sorted lists list1 and list2 into temp. Click the Step button to perform one comparison and move an element to temp. Click the Reset button to start over with two new random lists.

current1

↓

15	44	47	63	76	87	90
----	----	----	----	----	----	----

current2

↓

22	22	30	51	65	82	85
----	----	----	----	----	----	----

current3

↓

15	22	22	30	44	47	51								
----	----	----	----	----	----	----	--	--	--	--	--	--	--	--

Step Reset

Copy 51 from list2 to temp[6].

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
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  }

```

list1 to temp

list2 to temp

rest of list1 to temp

rest of list2 to temp

Refer Recorded Lecture

```
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++];
        }
    }
}
```

sortlecture.MergeSort > mergeSort # (list.length > 1) >

Output - Sortlecture (run) x

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


```

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


```
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

```
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;  
        }  
    }  
}
```

```

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

```

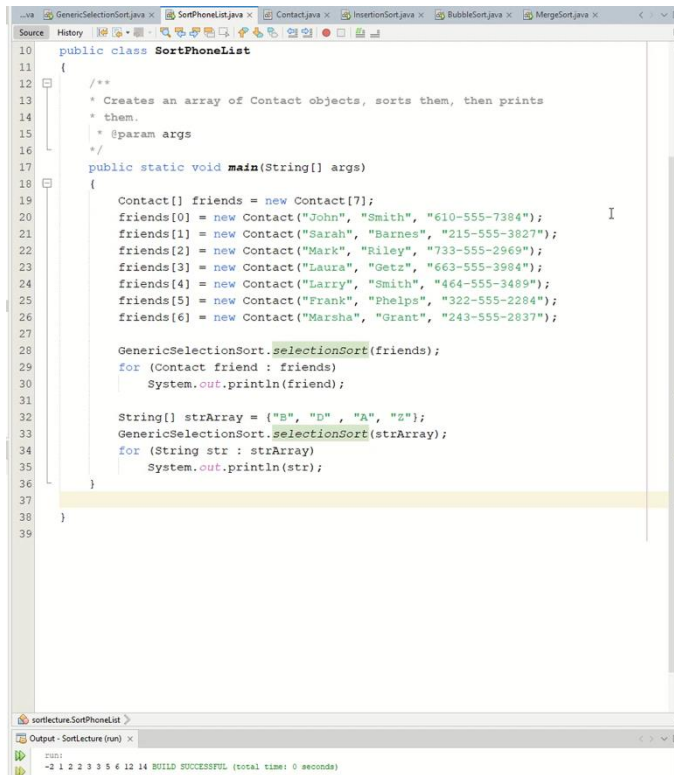
public class GenericSelectionSort {
    public static <T extends Comparable<T>> void selectionSort(T[] list)
    {
        for (int i = 0; i < list.length-1; i++) {
            T currentMin = list[i];
            int currentMinIndex = i;

            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



The screenshot shows an IDE with several tabs open: GenericSelectionSort.java, SortPhoneList.java, Contact.java, InsertionSort.java, BubbleSort.java, and MergeSort.java. The active tab is SortPhoneList.java, which contains the following Java code:

```
10 public class SortPhoneList
11 {
12     /**
13      * Creates an array of Contact objects, sorts them, then prints
14      * them.
15      * @param args
16      */
17     public static void main(String[] args)
18     {
19         Contact[] friends = new Contact[7];
20         friends[0] = new Contact("John", "Smith", "610-555-7384");
21         friends[1] = new Contact("Sarah", "Barnes", "215-555-3027");
22         friends[2] = new Contact("Mark", "Riley", "733-555-2969");
23         friends[3] = new Contact("Laura", "Getz", "663-555-3984");
24         friends[4] = new Contact("Larry", "Smith", "464-555-3489");
25         friends[5] = new Contact("Frank", "Phelps", "322-555-2284");
26         friends[6] = new Contact("Marsha", "Grant", "243-555-2837");
27
28         GenericSelectionSort.SelectionSort(friends);
29         for (Contact friend : friends)
30             System.out.println(friend);
31
32         String[] strArray = {"B", "D", "A", "Z"};
33         GenericSelectionSort.SelectionSort(strArray);
34         for (String str : strArray)
35             System.out.println(str);
36     }
37 }
38
39
```

The output window at the bottom shows the following output:

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

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

Home of the Bright, Land of the Brave
Di Sini Bermulanya Pintar, Tanah Tumpahnya Berani



www.um.edu.my



[universityofmalaya](https://www.facebook.com/universityofmalaya)



[unimalaya](https://www.instagram.com/unimalaya)



[uniofmalaya](https://www.youtube.com/uniofmalaya)



UNIVERSITI
MALAYA