CS 250 - Fall 2021 - Lab 10: Sorting

Available: Nov 10, 2021.
Due date: **Nov 15, 2021, 11:59 PM**

ONE OF THE MOST COMMON OPERATIONS performed by computers is that of sorting a list of items. An example of this would be sorting a list of names into alphabetical order. This lab will help you to practice some fundamentals sorting algorithms such as bubble sort, selection sort, insertion sort, and merge sort.

Objectives:

The objectives of this lab are:

- To understand different sorting algorithms.
- To practice coding Selection Sort, Bubble Sort, Insertion Sort, and Merge Sort.

Prerequisite and Resource:

- Lecture 10: Array Sorting
- Sorting Tutorial on Javatpoint

Part 1: Implement the Bubble Sort Algorithm

The basic idea of Bubble Sort is to compare two neighboring items and, if they are in the wrong order, swap them. The computer moves through the list comparing and swapping. At the end of one pass, the largest item will have "bubbled up" to the last position in the list. On the second pass, the next-to-largest item moves into next-to-last position, and so forth.

1. Create a new project named Lab10 and create a class named SortingApp

2. Copy the following skeleton to replace the generated code in Eclipse

```
public class SortingApp {
        //Print all elements of the array in one line and separated by space
        private static void printArray(int[] arrInt) {
                //TODO add code below
        }

        //Swap two elements at indexes i and j in array arrInt
        private static void swap(int[] arrInt, int i, int j) {
                //TODO add code below
        }

        //sort the array using the bubble sort algorithm
        private static void bubbleSort(int[] arrInt) {
                //TODO add code below
        }

        private static void insertionSort(int[] arrInt) {
                //TODO add code below
        }

        private static int[] mergeSort(int[] arrInt) {
                //TODO add code below
        }

        public static void main(String[] args) {
                //TODO add code below
        }
}
```

3. **Complete the method printArray(int[] arrInt)**: Use a for loop that iterates through each element in the array arrInt to print such an element with a space at the end using System.out.print statement. Finally, print a new line character.

4. **Complete the method swap(int[] arrInt, int i, int j)**: to two elements at the indexes i and j in the array arrInt. Recall the lecture about this swap operation. Hint: use a temporary integer to swap these two elements.

5. **Complete the method bubbleSort(int[] arrInt)**: to sort the array arrInt using the bubble sort algorithm. We will use two for loops to compare two adjacent elements and swap them if they are not in order. Use the following hints:

- for index i from 0 to less than the length of arrInt - 1
- for index j from 0 to less than the length of arrInt - 1
  - If arrInt[j] > arrInt[j+1], perform swapping two of them using the **swap**() method. Print the two indexes and two elements that are involved with this swapping action.

6. Add code to the **main**() method to test **bubbleSort**() method:

- Create an integer array with the following elements: 22, 18, 12, -4, 27, 30, 36, 50, 7, 68, 91, 56, 2, 85, 42, 98
- Call the method **bubbleSort**() on the above array

- Print the array before and after sorting using the **printArray**() method

Part 2: Implement the Selection Sort Algorithm

Selection sort is most commonly used in alphabetizing lists -- you have most likely alphabetize a list at some point by scanning through it repeatedly, finding the item that goes next in order on each scan. Selection sort on a computer is no different. To sort a list into increasing order, go through the list and find the smallest item. Move that item to the beginning of the list by exchanging it with the item originally at the start position. Go through the list again and find the next smallest item except for the first one already found. Place it in the next slot by exchanging it with the item currently in that slot. Keep repeating this procedure, once for each item in the list, and you will end up with a sorted list. You will note that the principle behind bubble sort and selection sort is the same: repeatedly find the next element in sorted order, starting with the largest, and put it where it belongs. The main difference is that bubble sort does this by repeatedly exchanging neighboring items while moving down the list and selection sort does it by scanning down the list to find the next smallest item and then putting that item where it belongs. Complete the following tasks of this part.

Implement this program using the following steps:

1. Copy the following code to the SortingApp.java file before the main() method:

```
/**
 * Sort the integer array using selection sort algorithm
 * @param arrInt: the integer array to be sorted
 * @return: none (this method will internally modify the order of elements of the array)
 */
private static void selectionSort(int[] arrInt) {
        //TODO add code below
}
```

2. **Complete the selectionSort method**. to sort the array arrInt using the selection sort algorithm. We will use one for loop for each element in the list, find the smallest element from the rest and swap with the current element. Use the following hints:

- for index i from 0 to length of arrInt - 1
    - Use a for loop with index j from i to length of arrInt to **find the smallest element index min**
    - Perform swapping arrInt[i] with arrInt[min] using the **swap**() method. Print the two indexes and two elements that are involved with this swapping action.

3. Add code to the **main()** method to test the **selectionSort** method with the following modification:

- Create a Scanner object
- Prompt a text asking the user to select the sorting algorithm to execute
- If the users select "Bubble Sort", execute the bubbleSort() method on part 1
- Else if the users select "Selection Sort", execute the selectionSort() method
- Finally, print the sorted array arrInt using printArray() method

Part 3: Implement the Insertion Sort Algorithm

In the Insertion Sort algorithm, the basic idea is to take a sorted list and to insert a new item into its proper position in the list. The length of the sorted list grows by one every time a new item is inserted. You can start with a list containing just one item. Then you can insert the remaining items, one at a time, into the list. At the end of this process, all the items have been sorted. The problem is to determine where in the list the new item should be inserted.

The following steps guide you through implementing the code for this part:

1. Copy the following code to the SortingApp.java file before the main() method:

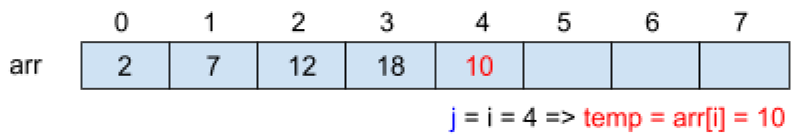
```
/**
 * Sort the integer array using insertion sort algorithm
 * @param arrInt: the integer array to be sorted
 * @return: none (this method will internally modify the order of elements of the array)
 */
private static void insertionSort(int[] arrInt) {
        //TODO add code below
}
```

2. **Complete the method insertionSort()**: to sort the array arrInt using the insertion sort algorithm. We will use one for loop from the second element to the last element in the list, trying to put each of these elements in the proper place before it. Use the following hints:

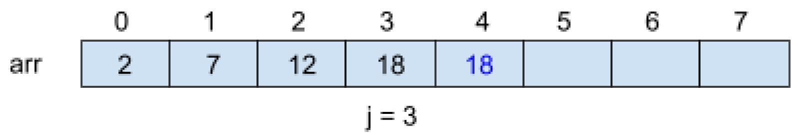
- for index i from 1 to length of arrInt - 1 (inclusive)
    - Here, we try to insert the element arrInt[i] in the correct place between [0, i-1]. So, first store the element arrInt[i] as a temporary integer (temp).
    - Then, use a while loop with index j from i-1 to 0 to shift any elements arrInt[j] in the index range of [0, i-1] that are greater than temp to the right one slot.
    - Finally, put temp (which originally store arrInt[i]) at the index j+1

- Print the two indexes and two elements that are involved with this shifting action.

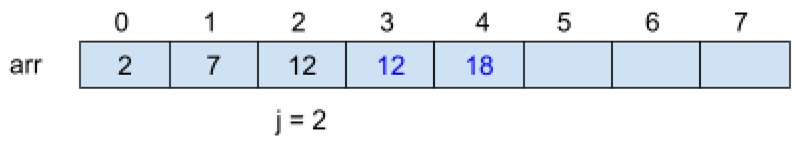
The following illustration example helps you to understand the step above



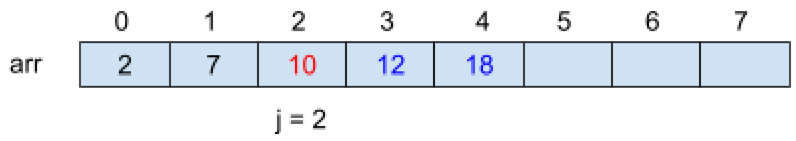
Start with j = 4, compare arr[j - 1] = 18 with temp: 18 > 10 => shift 18 to the right one place, decrease j



Next with j = 3, compare arr[j - 1] = 12 with temp: 12 > 10 => shift 12 to the right one place, decrease j



Next with j = 2, compare arr[j - 1] = 7 with temp: 7 < 10 => found the place to insert 10 here



3. **Complete the main() method:** to test the **insertionSort** method with the following modification:

- Add another else if statement to check if the user selects "Insertion Sort", then call the method **insertionSort**() on the arrInt.

Part 4: Implement the Merge Sort Algorithm

Merge Sort uses the idea of "merging" two sorted lists into one longer sorted list. To start, think of single items as being sorted lists of length one. In the first phase of the sort, these lists of length one are paired off and are merged to form sorted lists of length two. In the next phase of the sort, the lists of length two are merged to form lists of length four. Then lists of length four are merged into lists of length eight. This continues until all the items have been merged into one long sorted list.

The following steps guide you through implementing the code for this part:

1. Copy the following code to replace the generated code of that class:

```
/**
 * Merge two sorted arrays left and right into the result array
 * @param left: the left sorted array
 * @param right: the right sorted array
 * @param result: the merge sorted array from left and right
 * @return: none (this method will internally merge left and right into result)
 */
public static void merge(int[] left, int[] right, int[] result) {
        //TODO add code below
}

/**
 * Sort the integer array using merge sort algorithm
 * @param arrInt: the integer array to be sorted
 * @return: none (this method will internally modify the order of elements of the array)
 */
private static void mergeSort(int[] arrInt) {
        //TODO add code below
}
```

2. **Complete the method merge(int[] left, int[] right, int[] result)**: This method will merge the two already merged arrays left and right into the result array. The result array has its length equal to the total length of the left array and right array. Use the following hints:

- Initialize index1 and index2 as 0
- For an index i from 0 to less than the length of the result array:
    - If index2 is greater than or equal to the length of the right array or (index1 < length of the left array and left[index1] <= right[index2]):
        - Assign left[index1] to result[i]
        - Increase 1 to index1
    - Otherwise,
        - Assign right[index2] to result[i]
        - Increase 1 to index2

3. **Complete the method mergeSort(int[] arrInt)**: This method will sort the arrInt using the merge sort algorithm. The idea here is to split the array into two equal half arrays, recursively sort these two halves, and merge them into the final array. The base case is the case the array has only one element as we do not need to do anything. We use the recursive approach to implement this method. Use the following hints:

- The base case: if the length of arrInt < 2, do nothing (we do not need to sort an empty array or one-element array)
- Otherwise, do the following steps:

- Copy the left half with size of arrInt.length/2 of the arrInt into a left array using System.arraycopy() method
- Copy the right half of the arrInt (remaining arrInt.length - arrInt.length/2 elements) into a right array using System.arraycopy() method
- Make a recursive call of mergeSort on the left array
- Make a recursive call of mergeSort on the right array
- Finally, merge the sorted left and sorted right by using the merge() method

4. **Add code to test the mergeSort() method in the main() method:**

- Add another else if statement to check if the user selects “Merge Sort”, then call the method **mergeSort**() on the arrInt.

Lab Assignment

Complete the following assignment of this lab:

1. Modify the **bubbleSort**(), **selectionSort**(), and **insertionSort**() methods to display the number of comparisons and the number of swapping.

2. Implement the generic **insertionSort** method to sort a generic array. Add code to the following empty method:

```
public static <E extends Comparable<E>> void insertionSort(E[] arr) {
        //TODO Add code below
}
```

Test this method on a String array: String[] arrStr = {"b", "d", "c", "a", "f", "g", "e", "h"};
After that, print the sorted array to verify.

Lab 10 Result Submission:

You need to submit the results of the following sections to the D2L assignment item of Lab09:

- Complete the required Lab Assignment

- Compress the whole Eclipse source code of your Lab10 project in zip format using the Export function