```
::::::::::::::
Lab9Status.txt
::::::::::::::
Problem 1: compiles, runs correctly on all provided input
Extra Credit: compiles, runs correctly on all provided input.
::::::::::::::
Lab9Conclusions.txt
::::::::::::::
I found the lab to be very insightful on a variety of common sorting algorithms, a
s well as how we as programmers can expand on said algorithms and improve upon the
m. I can definitely see myself utilizing these techniques in the field. ::::::::::
::::
Sorting.java
::::::::::::::
/*
 * Purpose: Data Structure and Algorithms Lab 9
 * Status: Complete and thoroughly tested
 * Last update: 04/06/23
 * Submitted:  04/06/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.04.06
 */
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Sorting
{
    static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.
in));
    public static void main (String[] args) throws IOException
    {
        boolean exit = false;
        int[] array;
        while (!exit)
        {
            System.out.println("Select from the following menu: \n"
                               + "0. Exit the program \n"
                               + "1. Bubblesort an array \n"
                               + "2. Improved Bubblesort an array \n"
                               + "3. Selection sort an array \n"
                               + "4. Improved selection sort an array \n"
                               + "5. Insertion sort an array \n"
                               + "6. Improved Insertion Sort an array");

            System.out.print("Make your menu selection now: " );
            int input = Integer.parseInt(stdin.readLine().trim());
            System.out.println(input);
            // possible cases for initial input
            switch (input)
            {
            case 0:
                System.out.println("Exiting program... good bye");
                exit = true;
                break;

            case 1:
                Bubblesort(insert());
                break;

            case 2:
                ImprovedBubblesort(insert());
                break;

            case 3:
                SelectionSort(insert());
                break;

            case 4:
                ImprovedSelectionSort(insert());
                break;

            case 5:
                InsertionSort(insert());
                break;

            case 6:
                ImprovedInsertionSort(insert());

            default:
                System.out.println("Invalid choice.");
                break;
            }
        }
    }

    /**
     * Insert items into the array to be sorted
     * @return array            Array of values
     * @throws IOException
     */
    private static int[] insert() throws IOException
    {
        System.out.println("How many items would you like to insert in the array?
");
        int amount = Integer.parseInt(stdin.readLine().trim());
        int[] array = new int[amount]; // array size = amount given
        for(int index = 0; index < amount; index++)
        {
            // populate array
            System.out.println("Enter integer " + index + " : ");
            array[index] = Integer.parseInt(stdin.readLine().trim());
            System.out.println(array[index]);
        }
        return array;
    }

    /**
     * Implements the bubble sort algorithm to sort an integer array in ascending
order.
     *
     * @param array     Array of values
     * @return array
     */
    private static void Bubblesort(int[] array)
    {
        int comps = 0; // # of comparisons
        int swaps = 0; // # of swaps
        int length = array.length;
        for(int index = 0; index < length; index++)
        {
            for(int j = 1; j < length - index; j++)
            {
                for (int k = 1; k < length - index; k++)
                {
                    comps++;
                    // If the current element is greater than the next element...
                    // swap them and increment the swap counter.
                    if(array[j - 1] > array[j])
                    {
                        int temp = array[j];
                        array[j] = array[j - 1];
                        array[j - 1] = temp;
                        swaps++;
                    }
                }
            }
        }
        toString(array, comps, swaps);
    }

    /**
     * Implements an improved bubble sort algorithm to sort an integer array in as
cending order.
     *
     * @param array     Array of values
     * @return array
     */
    private static void ImprovedBubblesort(int[] array)
    {
        int comps = 0; // # of comparisons
        int swaps = 0; // # of swaps
        int temp = 0;
        int length = array.length;
        boolean swapped;
        for(int index = 0; index < length - 1; index++)
        {
            swapped = false;
            for(int j = 0; j < length - 1 - index; j++)
            {
                comps++;
                if(array[j] > array[j + 1])
                {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swaps++;
                    swapped = true;
                }
            }

            if(!swapped)
            {
                break;
            }
        }
        toString(array, comps, swaps);
    }

    /**
     * Implements the selection sort algorithm to sort an integer array in ascendi
ng order.
     *
     * @param array     Array of values
     * @return array
```

```
     */
    private static void SelectionSort(int[] array)
    {
        int comps = 0; // # of comparisons
        int swaps = 0; // # of swaps
        int length = array.length;
        for(int index = 0; index < length - 1; index++)
        {
            int midIndex = index;
            for(int j = index + 1; j < length; j++)
            {   comps++;
                if(array[j] < array[midIndex])
                {
                    midIndex = j;
                }
            }

            if(midIndex != index)
            {
                int temp = array[index];
                array[index] = array[midIndex];
                array[midIndex] = temp;
                swaps++;
            }
        }
        toString(array, comps, swaps);
    }

    /**
     * Implements an improved selection sort algorithm to sort an integer array in
ascending order.
     *
     * @param array     Array of values
     * @return array
     */
    private static void ImprovedSelectionSort(int[] array)
    {
        int comps = 0;
        int swaps = 0;
        int length = array.length;
        for (int index = 0; index < length / 2; index++)
        {
            int minIndex = index, maxIndex = index;
            for (int j = index + 1; j < length - index; j++)
            {
                // compare both elements
                comps += 2;
                if (array[j] < array[minIndex])
                {
                    minIndex = j;
                }

                if (array[j] > array[maxIndex])
                {
                    maxIndex = j;
                }
            }

            if (minIndex != index)
            {
                int temp = array[index];
                array[index] = array[minIndex];
                array[minIndex] = temp;
                swaps++;
            }

            /*
            * If max val is in the first unsorted position...
            * it’s already swapped with the min val, so update the maxindex
            * to the index of the second highest value
            */
            if (maxIndex == index)
            {
                maxIndex = minIndex;
            }

            if (maxIndex != length - index - 1)
            {
                int temp = array[length - index - 1];
                array[length - index - 1] = array[maxIndex];
                array[maxIndex] = temp;
                swaps++;
            }
        }
        toString(array, comps, swaps);
    }

    /**
     * Implements an insertion selection sort algorithm to sort an integer array i
n ascending order.
     *
     * @param array     Array of values
     */
    private static void InsertionSort(int[] array)
    {
        int comps = 0;
        int swaps = 0;
        int length = array.length;
        for(int index = 1; index < length; index ++)
        {
            int key = array[index]; // bgein @ 1
            int pos = (index - 1); // begin @ 0
            for( ; pos >= 0 && array[pos] > key; pos--)
            {
                comps++;
                swaps++;
                array[pos + 1] = array[pos];
            }
            array[pos + 1] = key;
            swaps++;
        }
        toString(array, comps, swaps);
    }

    /**
     * EXTRA CREDIT W/ BINARY SEARCH
     * Implements an improved insertion sort algorithm to sort an integer array in
 ascending order.
     *
     * @param array     Array of values
     */
    private static void ImprovedInsertionSort(int[] array)
    {
        int comps = 0;
```

```
        int swaps = 0;
        int length = array.length;
        for (int index = 1; index < length; index++)
        {
            int key = array[index]; // begins at 1
            int pos = binarySearch(index, key, array);
            for(int j = (index- 1); j >= pos; j--)
            {
                swaps++;
                array[j + 1] = array[j];
            }
            array[pos] = key;
            swaps++; // swaps++
        }
        toString(array, comps, swaps);
    }

    /**
    * Binary Search II
    * Searches for an item in the array
    *
    * @param key       the item to search for
    * @return          the index of the item if found OR -1 for duplicates/nonexisti
ng items
    */
    private static int binarySearch(int index, int key, int[] array)
    {
        {
            int low = 0;
            int length = array.length;
            int high = length - 1;
            while (low <= high)
            {
                int midIndex = (low + high) / 2;
                if (key == midIndex)
                {
                    // key found
                    return midIndex;
                }

                else if (key < midIndex)
                {
                    // key smaller, search left half
                    high = midIndex - 1;
                }

                else
                {
                    // key larger, search right half
                    low = midIndex + 1;
                }
            }
            // return index where key should be inserted
            return -(low + 1);
        }
    }

    /**
     * Returns string representation of array
     *
     * @param array     Array of values
     */
    private static void toString(int[] array, int comps, int swaps)
    {
        int length = array.length;
        System.out.println("Sorted data: ");
        for(int index = 0; index < length; index++)
        {
            System.out.print(array[index] + " ");
        }
        System.out.println();
        System.out.println("\n Comparisons: " + comps + "\n Swaps: " + swaps + "\n
");
    }
}
::::::::::::::
Lab9Sampleruns.txt
::::::::::::::
Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 1
How many items would you like to insert in the array?
Sorted data:


 Comparisons: 0
 Swaps: 0

Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 1
How many items would you like to insert in the array?
Enter integer 0 :
-10
Enter integer 1 :
-100
Enter integer 2 :
3
Enter integer 3 :
4
Enter integer 4 :
100
Sorted data:
-100 -10 3 4 100

 Comparisons: 30
 Swaps: 1

Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 2
How many items would you like to insert in the array?
Enter integer 0 :
101
Enter integer 1 :
102
Enter integer 2 :
-100
Enter integer 3 :
-300
Enter integer 4 :
600
Sorted data:
-300 -100 101 102 600

 Comparisons: 10
 Swaps: 5

Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 3
How many items would you like to insert in the array?
Enter integer 0 :
1000
Enter integer 1 :
10
Enter integer 2 :
11
Enter integer 3 :
-3
Enter integer 4 :
15
Sorted data:
-3 10 11 15 1000

 Comparisons: 10
 Swaps: 2

Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 4
How many items would you like to insert in the array?
Enter integer 0 :
945
Enter integer 1 :
864
```

```
Enter integer 2 :
955
Enter integer 3 :
-955
Enter integer 4 :
10
Sorted data:
-955 10 864 945 955

 Comparisons: 12
 Swaps: 3

Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 5
How many items would you like to insert in the array?
Enter integer 0 :
10
Enter integer 1 :
11
Enter integer 2 :
-14
Enter integer 3 :
-35
Enter integer 4 :
58
Sorted data:
-35 -14 10 11 58

 Comparisons: 5
 Swaps: 9

Select from the following menu:
0. Exit the program
1. Bubblesort an array
2. Improved Bubblesort an array
3. Selection sort an array
4. Improved selection sort an array
5. Insertion sort an array
6. Improved Insertion Sort an array
Make your menu selection now: 0
Exiting program... good bye
::::::::::::::
P2Writeup.txt
::::::::::::::
```

Time Complexity:
Insertion Sort is a sorting algorithm that works by taking each element from an unsorted list and placing it in its correct position in a sorted list. Improved Insertion Sort is a sorting algorithm that arranges each element of an unsorted list into its appropriate position in a sorted list. Unlike the original algorithm, Improved Insertion Sort uses binary search to locate the correct position of each element, rather than comparing it with all previous elements.

To analyze the time complexity of Improved Insertion Sort, one can consider the number of comparisons and element movements required to sort the input list. In the worst-**case** scenario, when the input list is in reverse order, each element would need to be compared with all previous elements, leading to a time complexity of O(n^2).

However, by using binary search, the number of comparisons required **for** each element decreases, resulting in a more efficient algorithm. The number of comparisons needed **for** each element is proportional to the logarithm of the size of the sorted sub-list, leading to a best-**case** time complexity of O(n log n) when the input list is already sorted.

On average, Improved Insertion Sort has a time complexity of O(n^2), which is slower than other efficient sorting algorithms such as Merge Sort and Quick Sort. However, Improved Insertion Sort has the advantage of being an in-place sorting algorithm, which means that it does not require extra memory space to perform the sorting operation.

Therefore, the time complexity of Improved Insertion Sort can be expressed as O(n^2) in the worst **case**, O(n log n) in the best **case**, and O(n^2) in the average **case**.