# ADS LAB 3 REPORT

Elsayed, Ahmed

MEMBER B  TEAM 15

# Problem 6

An Invariant is a property of the sorting algorithm that has to be maintained in order to achieve the sorted array or in order to achieve the sorted array, the algorithm must follow these invariants (Properties)

The Selection Sort has 2 invariants which are:

1. All elements to the left of the element on which we have the current index must be sorted and won't undergo anymore operations through out the rest of the sorting process.
2. All elements to the right of the elements on which we have the current index must be greater than all the elements to the left of that index.

```java
public class Selection extends Sort
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++) // i: destination site
        {
            int min = i; // index of minimum candidate
            for (int j = i+1; j < N; j++){
                if (less(a[j], a[min])){
                    min = j; // better minimum candidate found
                }
            }
            // now min is the index of the minimum
            exch(a, i, min);
        }
    }
}
```

This is the condition in the invariant loop of the selection sort algorithm. (Increment in order not to include the last element in the sorting operation anymore and exchange of the smaller and the greater element in order to obtain sorted left part compared to the current index)

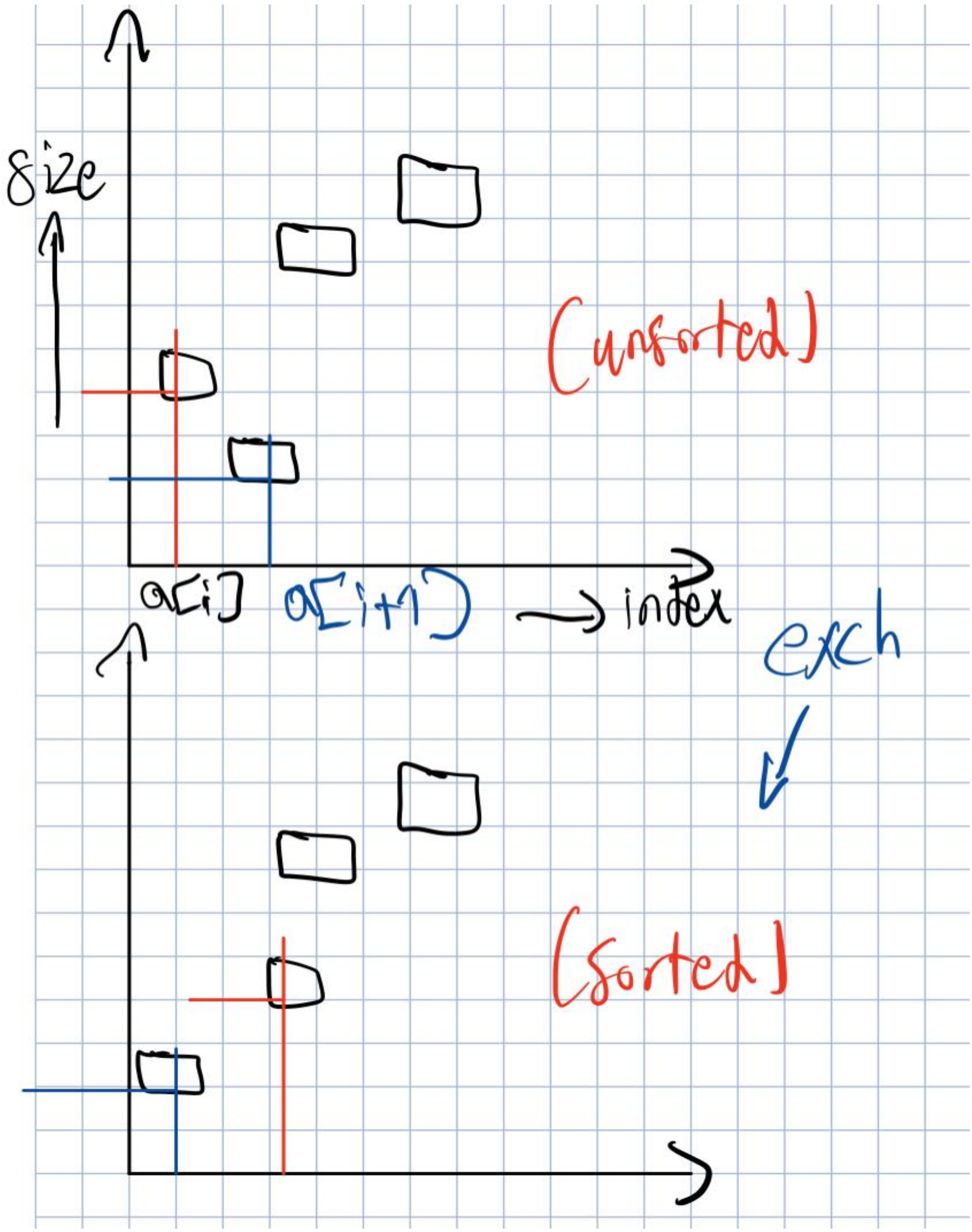
On the other hand, the Insertion Sort has only 1 invariant which is:

1. Elements of the array to the left of the current index are already sorted

```java
public class Insertion extends Sort
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 1; i < N; i++){
            for (int j = i; j > 0; j--){
                if (less(a[j], a[j-1])){
                    exch(a, j, j-1);
                }
                else{
                    break; // input dependent
                }
            }
        }
    }
}
```

This is the condition in the invariant loop of the insertion sort algorithm (exchange of the smaller and greater element to get the sorted left part compared to the current index)

# Visual Representation of Selection Algorithm



# Recurrence Relation of MergeSort

Relation of MergeSort in any case is $T(n) = 2 * T(N/2) + O(N)$ where n: number of elements in the current array, N: number of all elements in the main array

Best case: $T(n) = (½)N \log N$

Worst Case: $T(n) = N \log_2 N - (N - 1)$

Best case in the number of comparisons would be that the algorithm compares only N times.

Worst case in the number of comparisons would be that the algorithm compares only $(2N - 1)$ times.

After doing the mathematical induction: $C(N)_{best} = N \log_2 N + N$

# Problem 8:

1. For more information on the assert statement, please check the attached source code
2. Results of the terminal for the required tests are clear in the following picture:

```
Problem 8 QuickSort test
Original Array: 20 19 68 45
Sorted Array: 19 20 45 68
isSorted is true
```

   For more information on the assert statement, please check the source code attached to this document
3. QuickSort Algorithm test:

4.

```
Problem 8 QuickSort test 1
Original Array: 20 19 68 45
Sorted Array: 19 20 45 68
isSorted is true
Number of Comparisons is 21
Number of Copies is 9


Problem 8 QuickSort test 2
Original Array: 20 19 68 45 2 1 0 4
Sorted Array: 0 1 2 4 19 20 45 68
isSorted is true
Number of Comparisons is 33
Number of Copies is 24


Problem 8 QuickSort test 3
Original Array: 20 19 68 45 2 1 0 4 24 18 76 67 55 43 32 23
Sorted Array: 0 1 2 4 18 19 20 23 24 32 43 45 55 67 68 76
isSorted is true
Number of Comparisons is 95
Number of Copies is 45


Problem 8 QuickSort test 4
Original Array: 20 19 68 45 2 1 0 4 24 18 76 67 55 43 32 23 14 7 3 90 27 13 12 11 5 8 6 70 71 62 36 34
Sorted Array: 0 1 2 3 4 5 6 7 8 11 12 13 14 18 19 20 23 24 27 32 34 36 43 45 55 62 67 68 70 71 76 90
isSorted is true
Number of Comparisons is 185
Number of Copies is 123
```

InsertionSort Algorithm test:

```
Problem 8 InsertionSort test 1
Original Array: 20 19 68 45
Sorted Array: 19 20 45 68
isSorted is true
Number of Comparisons is 7
Number of Copies is 6


Problem 8 InsertionSort test 2
Original Array: 20 19 68 45 2 1 0 4
Sorted Array: 0 1 2 4 19 20 45 68
isSorted is true
Number of Comparisons is 31
Number of Copies is 63


Problem 8 InsertionSort test 3
Original Array: 20 19 68 45 2 1 0 4 24 18 76 67 55 43 32 23
Sorted Array: 0 1 2 4 18 19 20 23 24 32 43 45 55 67 68 76
isSorted is true
Number of Comparisons is 78
Number of Copies is 156


Problem 8 InsertionSort test 4
Original Array: 20 19 68 45 2 1 0 4 24 18 76 67 55 43 32 23 14 7 3 90 27 13 12 11 5 8 6 70 71 62 36 34
Sorted Array: 0 1 2 3 4 5 6 7 8 11 12 13 14 18 19 20 23 24 27 32 34 36 43 45 55 62 67 68 70 71 76 90
isSorted is true
Number of Comparisons is 294
Number of Copies is 708
```

Both algorithms’ runs match the theoretical results.

# LAB

1. For information concerning the implementation of the SortCases Classes, please check out the attached source code.
2. The following tables show the number of copies and comparisons count for each case and for both QuickSort and InsertionSort.

QuickSort Count for each case:

Average case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 2 | 3 |
| 4 | 7 | 9 |
| 8 | 34 | 21 |
| 16 | 68 | 48 |
| 32 | 194 | 123 |
| 64 | 441 | 279 |
| 128 | 1024 | 654 |
| 256 | 2397 | 1500 |
| 512 | 6075 | 3225 |

| | | |
|---|---|---|
| 1024 | 12716 | 7263 |
| 2048 | 28658 | 15969 |
| 4096 | 60870 | 35232 |
| 8192 | 140045 | 75468 |
| 16384 | 278280 | 164346 |
| 32768 | 649688 | 346932 |

Presorted case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 3 | 3 |
| 4 | 9 | 6 |
| 8 | 26 | 24 |
| 16 | 72 | 48 |
| 32 | 203 | 129 |
| 64 | 414 | 297 |
| 128 | 974 | 648 |
| 256 | 2486 | 1482 |
| 512 | 5562 | 3351 |
| 1024 | 11587 | 7488 |
| 2048 | 26071 | 16308 |
| 4096 | 62716 | 34827 |
| 8192 | 143681 | 74523 |
| 16384 | 276329 | 164826 |
| 32768 | 653733 | 347871 |

Reverse case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 2 | 3 |
| 4 | 12 | 9 |
| 8 | 26 | 21 |
| 16 | 67 | 51 |
| 32 | 172 | 129 |
| 64 | 490 | 270 |
| 128 | 1080 | 669 |
| 256 | 2543 | 1473 |
| 512 | 5364 | 3339 |
| 1024 | 12984 | 7386 |
| 2048 | 27754 | 16077 |
| 4096 | 59482 | 35178 |
| 8192 | 133543 | 75159 |
| 16384 | 289715 | 163182 |
| 32768 | 604533 | 351354 |

Random example case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 2 | 3 |
| 4 | 7 | 9 |

| | | |
|---|---|---|
| 8 | 33 | 21 |
| 16 | 60 | 48 |
| 32 | 160 | 138 |
| 64 | 477 | 297 |
| 128 | 1138 | 663 |
| 256 | 2487 | 1455 |
| 512 | 5551 | 3336 |
| 1024 | 12757 | 7266 |
| 2048 | 27728 | 16059 |
| 4096 | 66437 | 35061 |
| 8192 | 140918 | 75171 |
| 16384 | 294823 | 163788 |
| 32768 | 616265 | 350100 |

## InsertionSort count for each case:

Average case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 1 | 3 |
| 4 | 3 | 3 |
| 8 | 15 | 27 |
| 16 | 84 | 222 |
| 32 | 321 | 885 |
| 64 | 1072 | 3036 |
| 128 | 4259 | 12417 |
| 256 | 16811 | 49689 |
| 512 | 65921 | 196254 |
| 1024 | 264854 | 791514 |
| 2048 | 1055109 | 3159213 |
| 4096 | 4202615 | 12595587 |
| 8192 | 16729990 | 50165415 |
| 16384 | 66864026 | 200542947 |
| 32768 | 267912514 | 803639268 |

Presorted case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 1 | 0 |
| 4 | 3 | 0 |
| 8 | 7 | 0 |
| 16 | 15 | 0 |
| 32 | 31 | 0 |
| 64 | 63 | 0 |
| 128 | 127 | 0 |
| 256 | 255 | 0 |
| 512 | 511 | 0 |
| 1024 | 1023 | 0 |

| | | |
|---|---|---|
| 2048 | 2047 | 0 |
| 4096 | 4095 | 0 |
| 8192 | 8191 | 0 |
| 16384 | 16383 | 0 |
| 32768 | 32767 | 0 |

Reverse case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 1 | 3 |
| 4 | 6 | 18 |
| 8 | 28 | 84 |
| 16 | 120 | 360 |
| 32 | 496 | 1488 |
| 64 | 2016 | 6048 |
| 128 | 8128 | 24384 |
| 256 | 32640 | 97920 |
| 512 | 130816 | 392448 |
| 1024 | 523776 | 1571328 |
| 2048 | 2096128 | 6288384 |
| 4096 | 8386560 | 25159680 |
| 8192 | 33550336 | 100651008 |
| 16384 | 134209536 | 402628608 |
| 32768 | 536854528 | 1610563584 |

Random example case:

| N | #(cmp) | #(copies) |
|---|---|---|
| 2 | 1 | 0 |
| 4 | 6 | 9 |
| 8 | 20 | 42 |
| 16 | 73 | 186 |
| 32 | 306 | 834 |
| 64 | 1162 | 3309 |
| 128 | 4019 | 11694 |
| 256 | 17446 | 51585 |
| 512 | 65237 | 194184 |
| 1024 | 261884 | 782595 |
| 2048 | 1040608 | 3115698 |
| 4096 | 4142400 | 12414933 |
| 8192 | 16801140 | 50378880 |
| 16384 | 66628439 | 199836186 |
| 32768 | 269239963 | 807621606 |

3. For information on the best and worst cases for each algorithm and the charts for the previous tables, please check out the excel file attached.
4. Concerning the last 2 requirements which are:

4.1. In your report, explain how the superviser is able to **reproduce your data** with the seed value (not system time!) used.

I really can't understand what is it that is required eventually. I mean I read the question more than 5 times and I still don't get it. Who is the supervisor? And what seed value is being spoken of?

I can only assume that the supervisor would be the abstract class UseCase that we can use to generate the data each time and the seed value would be an array that follows one of the four InputCases that were mentioned in the Lab Sheet. If that is really the case, then we have a value which is the seed value to enter to the random case initialization and starting this seed value, we have an array that is randomly generated, then sorting would take place and the number of copies and comparisons are consequently printed.

4.2. Include also the data of Problem 5, best cases, into your plots and interpret your findings.

In problem 5, there was no sorting let alone any findings to be interpreted.

I can only assume that a typo was made and what is really meant in this situation is problem 6 not 5. Please find in the attached excel sheet the results to the best case of the MergeSort.

## References:


1. Sorting algorithms implementation, author: Wolfgang Renz
2. Problem 6 D, Author: Mohamed ElGhamrawy
3. UseCase Class Implementation, author: Wolfgang Renz