Yang Hu

Professor Russell Lewis

CSC345: Analysis of Discrete Structures

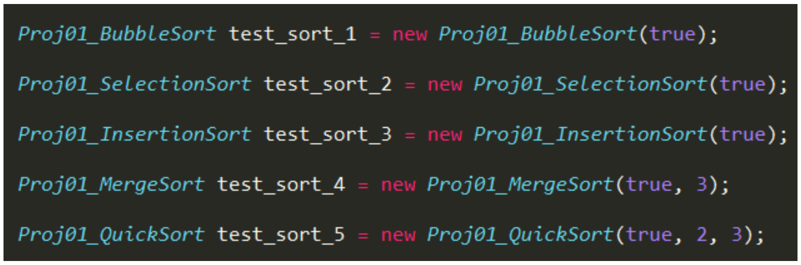
5 September 2018

Assignment 1 Debug Strategy

To demonstrate my debugger, I will use identical data to debug all my sorting classes, which is:

Integer[] arr = {2, 1, 5, 4, 7, 6, 3, 10, 9, 8};

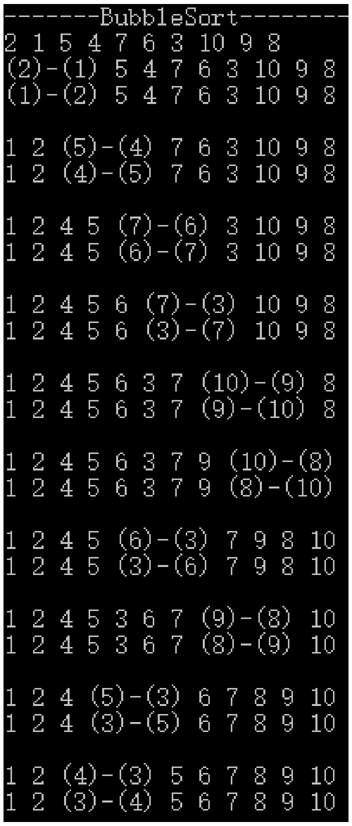
The code I used to initialize sorting classes is:



```java
Proj01_BubbleSort test_sort_1 = new Proj01_BubbleSort(true);
Proj01_SelectionSort test_sort_2 = new Proj01_SelectionSort(true);
Proj01_InsertionSort test_sort_3 = new Proj01_InsertionSort(true);
Proj01_MergeSort test_sort_4 = new Proj01_MergeSort(true, 3);
Proj01_QuickSort test_sort_5 = new Proj01_QuickSort(true, 2, 3);
```

Bubble Sort

My Bubble Sort debugger will print out all of the swaps, decorated by parentheses. All of the two-line output has similar lines with former/later output, which clearly show which pair is swapped, and which pair is going to be swapped.

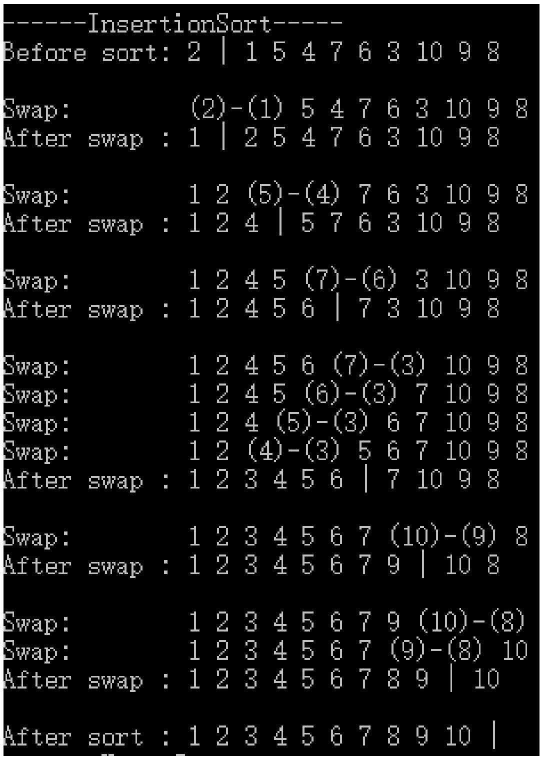


```
-------BubbleSort--------
2 1 5 4 7 6 3 10 9 8
(2)-(1) 5 4 7 6 3 10 9 8
(1)-(2) 5 4 7 6 3 10 9 8

1 2 (5)-(4) 7 6 3 10 9 8
1 2 (4)-(5) 7 6 3 10 9 8

1 2 4 5 (7)-(6) 3 10 9 8
1 2 4 5 (6)-(7) 3 10 9 8

1 2 4 5 6 (7)-(3) 10 9 8
1 2 4 5 6 (3)-(7) 10 9 8

1 2 4 5 6 3 7 (10)-(9) 8
1 2 4 5 6 3 7 (9)-(10) 8

1 2 4 5 6 3 7 9 (10)-(8)
1 2 4 5 6 3 7 9 (8)-(10)

1 2 4 5 (6)-(3) 7 9 8 10
1 2 4 5 (3)-(6) 7 9 8 10

1 2 4 5 3 6 7 (9)-(8) 10
1 2 4 5 3 6 7 (8)-(9) 10

1 2 4 (5)-(3) 6 7 8 9 10
1 2 4 (3)-(5) 6 7 8 9 10

1 2 (4)-(3) 5 6 7 8 9 10
1 2 (3)-(4) 5 6 7 8 9 10
```

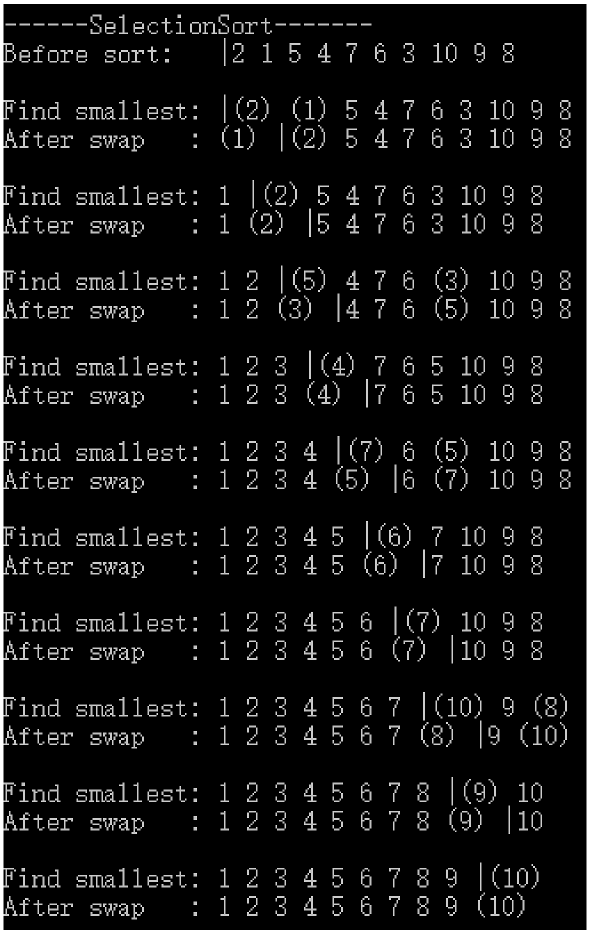
Selection Sort

Similar to Bubble Sort, the Selection Sort debugger will show all the swaps by parentheses.

Meanwhile, it will print out the boundary between sorted and unsorted parts.



```
------InsertionSort-----
Before sort: 2 | 1 5 4 7 6 3 10 9 8

Swap:        (2)-(1) 5 4 7 6 3 10 9 8
After swap : 1 | 2 5 4 7 6 3 10 9 8

Swap:        1 2 (5)-(4) 7 6 3 10 9 8
After swap : 1 2 4 | 5 7 6 3 10 9 8

Swap:        1 2 4 5 (7)-(6) 3 10 9 8
After swap : 1 2 4 5 6 | 7 3 10 9 8

Swap:        1 2 4 5 6 (7)-(3) 10 9 8
Swap:        1 2 4 5 (6)-(3) 7 10 9 8
Swap:        1 2 4 (5)-(3) 6 7 10 9 8
Swap:        1 2 (4)-(3) 5 6 7 10 9 8
After swap : 1 2 3 4 5 6 | 7 10 9 8

Swap:        1 2 3 4 5 6 7 (10)-(9) 8
After swap : 1 2 3 4 5 6 7 9 | 10 8

Swap:        1 2 3 4 5 6 7 9 (10)-(8)
Swap:        1 2 3 4 5 6 7 (9)-(8) 10
After swap : 1 2 3 4 5 6 7 8 9 | 10

After sort : 1 2 3 4 5 6 7 8 9 10 |
```



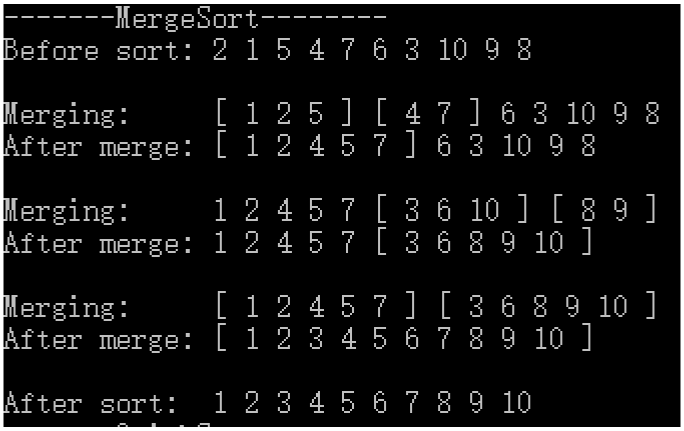
```
------SelectionSort-------
Before sort:   |2 1 5 4 7 6 3 10 9 8

Find smallest: |(2) (1) 5 4 7 6 3 10 9 8
After swap   : (1) |(2) 5 4 7 6 3 10 9 8

Find smallest: 1 |(2) 5 4 7 6 3 10 9 8
After swap   : 1 (2) |5 4 7 6 3 10 9 8

Find smallest: 1 2 |(5) 4 7 6 (3) 10 9 8
After swap   : 1 2 (3) |4 7 6 (5) 10 9 8

Find smallest: 1 2 3 |(4) 7 6 5 10 9 8
After swap   : 1 2 3 (4) |7 6 5 10 9 8

Find smallest: 1 2 3 4 |(7) 6 (5) 10 9 8
After swap   : 1 2 3 4 (5) |6 (7) 10 9 8

Find smallest: 1 2 3 4 5 |(6) 7 10 9 8
After swap   : 1 2 3 4 5 (6) |7 10 9 8

Find smallest: 1 2 3 4 5 6 |(7) 10 9 8
After swap   : 1 2 3 4 5 6 (7) |10 9 8

Find smallest: 1 2 3 4 5 6 7 |(10) 9 (8)
After swap   : 1 2 3 4 5 6 7 (8) |9 (10)

Find smallest: 1 2 3 4 5 6 7 8 |(9) 10
After swap   : 1 2 3 4 5 6 7 8 (9) |10

Find smallest: 1 2 3 4 5 6 7 8 9 |(10)
After swap   : 1 2 3 4 5 6 7 8 9 (10)
```

Insertion Sort

This debugger is almost same with Selection sort. Each group of output (Swap and after swap) shows one insertion which insert the smallest unsorted number into the sorted part.

Merge Sort

The Merge Sort debugger will show the merging process of the corresponding subarrays.

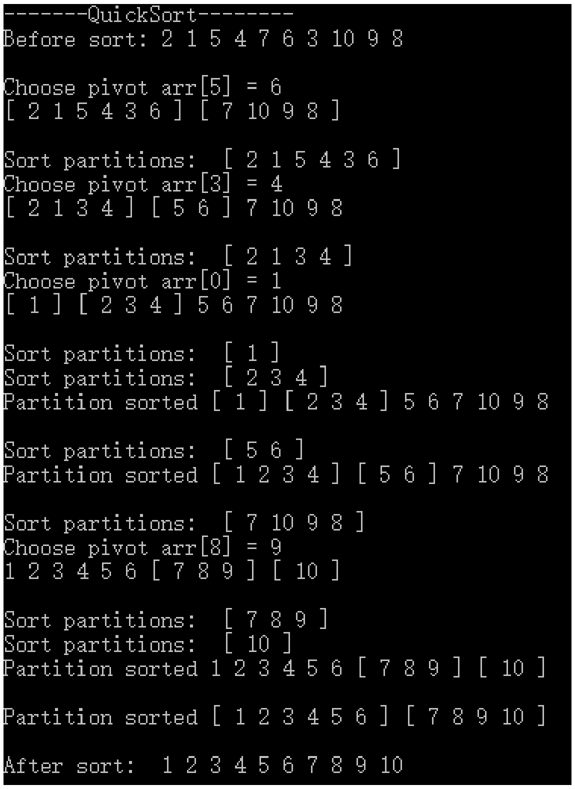


```
-------MergeSort--------
Before sort: 2 1 5 4 7 6 3 10 9 8

Merging:     [ 1 2 5 ] [ 4 7 ] 6 3 10 9 8
After merge: [ 1 2 4 5 7 ] 6 3 10 9 8

Merging:     1 2 4 5 7 [ 3 6 10 ] [ 8 9 ]
After merge: 1 2 4 5 7 [ 3 6 8 9 10 ]

Merging:     [ 1 2 4 5 7 ] [ 3 6 8 9 10 ]
After merge: [ 1 2 3 4 5 6 7 8 9 10 ]

After sort:  1 2 3 4 5 6 7 8 9 10
```

Quick Sort

Quick Sort debugger is similar to the Merge Sort debugger. It highlights the undergoing parts (pivot, low, high) by square brackets, and prints the pivot before going into partitions.

Besides, debugger will display the O(n^2) sort on small partitions to make process clearer.



```
-------QuickSort--------
Before sort: 2 1 5 4 7 6 3 10 9 8

Choose pivot arr[5] = 6
[ 2 1 5 4 3 6 ] [ 7 10 9 8 ]

Sort partitions:  [ 2 1 5 4 3 6 ]
Choose pivot arr[3] = 4
[ 2 1 3 4 ] [ 5 6 ] 7 10 9 8

Sort partitions:  [ 2 1 3 4 ]
Choose pivot arr[0] = 1
[ 1 ] [ 2 3 4 ] 5 6 7 10 9 8

Sort partitions:  [ 1 ]
Sort partitions:  [ 2 3 4 ]
Partition sorted [ 1 ] [ 2 3 4 ] 5 6 7 10 9 8

Sort partitions:  [ 5 6 ]
Partition sorted [ 1 2 3 4 ] [ 5 6 ] 7 10 9 8

Sort partitions:  [ 7 10 9 8 ]
Choose pivot arr[8] = 9
1 2 3 4 5 6 [ 7 8 9 ] [ 10 ]

Sort partitions:  [ 7 8 9 ]
Sort partitions:  [ 10 ]
Partition sorted 1 2 3 4 5 6 [ 7 8 9 ] [ 10 ]

Partition sorted [ 1 2 3 4 5 6 ] [ 7 8 9 10 ]

After sort:  1 2 3 4 5 6 7 8 9 10
```