CSc345 Fall 2018
Project 01 – Sorting Algorithms in Java
Matt Burns

Debug Mode Explanation

1. Bubble Sort

The debug output for bubble sort prints a visualization of the array before each swap is made, highlighting the two elements about to be swapped in parentheses.

```
154 (214) (95) 148 100 179 10 214 123 8 122 232 126 237 92 195 174 210 220 103 129 4 1 160
```

This output is printed only when elements are about to be swapped, with the goal of limiting the debug printing to a manageable amount. A benefit of this method is that as the array is printed multiple times, the progression of the items in parentheses progresses along a diagonal, with the diagonal getting shorter and shorter as higher values "bubble up" to the top of the array.

```
95 (100) (10) 148 123 8 122 154 126 179 92 195 174 210 214 103 129 4 1 160 214 220 232 237
95 10 100 (148) (123) 8 122 154 126 179 92 195 174 210 214 103 129 4 1 160 214 220 232 237
95 10 100 123 (148) (8) 122 154 126 179 92 195 174 210 214 103 129 4 1 160 214 220 232 237
95 10 100 123 8 (148) (122) 154 126 179 92 195 174 210 214 103 129 4 1 160 214 220 232 237
95 10 100 123 8 122 148 (154) (126) 179 92 195 174 210 214 103 129 4 1 160 214 220 232 237
95 10 100 123 8 122 148 126 154 (179) (92) 195 174 210 214 103 129 4 1 160 214 220 232 237
95 10 100 123 8 122 148 126 154 92 179 (195) (174) 210 214 103 129 4 1 160 214 220 232 237
95 10 100 123 8 122 148 126 154 92 179 174 195 210 (214) (103) 129 4 1 160 214 220 232 237
95 10 100 123 8 122 148 126 154 92 179 174 195 210 103 (214) (129) 4 1 160 214 220 232 237
95 10 100 123 8 122 148 126 154 92 179 174 195 210 103 129 (214) (4) 1 160 214 220 232 237
95 10 100 123 8 122 148 126 154 92 179 174 195 210 103 129 4 (214) (1) 160 214 220 232 237
95 10 100 123 8 122 148 126 154 92 179 174 195 210 103 129 4 1 (214) (160) 214 220 232 237
(95) (10) 100 123 8 122 148 126 154 92 179 174 195 210 103 129 4 1 160 214 214 220 232 237
```

This also means that while skimming through the debug output, you can see the sorted elements grow, as they always appear the same to the right of the parentheisized numbers.

```
8 10 92 95 100 122 123 103 126 4 (129) (1) 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 122 (123) (103) 126 4 1 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 122 103 123 (126) (4) 1 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 122 103 123 4 (126) (1) 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 (122) (103) 123 4 1 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 103 122 (123) (4) 1 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 103 122 4 (123) (1) 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 103 (122) (4) 1 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 103 4 (122) (1) 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 (103) (4) 1 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 100 4 (103) (1) 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 (100) (4) 1 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 95 4 (100) (1) 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 (95) (4) 1 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 92 4 (95) (1) 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 (92) (4) 1 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 10 4 (92) (1) 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 (10) (4) 1 92 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
8 4 (10) (1) 92 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
(8) (4) 1 10 92 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
4 (8) (1) 10 92 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
(4) (1) 8 10 92 95 100 103 122 123 126 129 148 154 160 174 179 195 210 214 214 220 232 237
```

CSc345 Fall 2018
Project 01 – Sorting Algorithms in Java
Matt Burns

2. Selection Sort

The selection sort debug output prints a single line per iteration through the array, with the sorted portion contained in brackets, and the minimum value in the unsorted portion highlighted in parentheses.

```
[1 4 8 10] 179 148 214 123 95 122 232 126 237 (92) 195 174 210 220 103 129 214 154 160
```

This allows you to check that the correct element is being identified for each pass through the array, and also makes it possible to see the sorted portion grow by one element every line, on a nice diagonal.

```
[] 214 95 148 100 179 10 214 123 8 122 232 126 237 92 195 174 210 220 103 129 4 (1) 160
[1] 95 148 100 179 10 214 123 8 122 232 126 237 92 195 174 210 220 103 129 (4) 154 160
[1 4] 148 100 179 10 214 123 (8) 122 232 126 237 92 195 174 210 220 103 129 214 154 160
[1 4 8] 100 179 (10) 214 123 95 122 232 126 237 92 195 174 210 220 103 129 214 154 160
[1 4 8 10] 179 148 214 123 95 122 232 126 237 (92) 195 174 210 220 103 129 214 154 160
[1 4 8 10 92] 148 214 123 (95) 122 232 126 237 100 195 174 210 220 103 129 214 154 160
[1 4 8 10 92 95] 214 123 179 122 232 126 237 (100) 195 174 210 220 103 129 214 154 160
[1 4 8 10 92 95 100] 123 179 122 232 126 237 148 195 174 210 220 (103) 129 214 154 160
[1 4 8 10 92 95 100 103] 179 (122) 232 126 237 148 195 174 210 220 214 129 214 154 160
[1 4 8 10 92 95 100 103 122] (123) 232 126 237 148 195 174 210 220 214 129 214 154 160
```

3. Insertion Sort

The insertion sort debug output prints a heading stating which element is being examined. Then, if any swaps are made with that element, it prints a line containing the array each time, with the elements about to be swapped highlighted in parentheses.

```
Examining 195
8 10 92 95 100 122 123 126 148 154 179 214 214 232 (237) (195) 174 210 220 103 129 4 1 160
8 10 92 95 100 122 123 126 148 154 179 214 214 (232) (195) 237 174 210 220 103 129 4 1 160
8 10 92 95 100 122 123 126 148 154 179 214 (214) (195) 232 237 174 210 220 103 129 4 1 160
8 10 92 95 100 122 123 126 148 154 179 (214) (195) 214 232 237 174 210 220 103 129 4 1 160
```

This means you can see as a specific element descends through the array, swapping places with elements that are lower than it. The heading breaks indicating which element is being examined mean that you can quickly identify which elements traveled the furthest through the array, or which ones made no swaps themselves (though still may end up at a different spot than they started due to being swapped with other elements).

4. Merge Sort

Merge sort's debug output begins printing once the array has been split into blocks of length baseLen or less. This is because the matter of splitting the array into blocks was relatively straightforward and didn't need any debug assistance. At the top of the output, baseLen is printed as a reference. After that, the first two blocks shorter than baseLen are printed, as well as their start and end indices. The blocks are initially printed before being sorted with insertion sort to make it easier to compare to the original array.

```
baseLen: 11

Sorting current block (index 0 to 8): [133, 265, 292, 33, 66, 284, 252, 192, 35]
Sorting current block (index 9 to 16): [99, 180, 295, 298, 195, 250, 164, 17]
Merging these two blocks...
[33, 35, 66, 133, 192, 252, 265, 284, 292]
[17, 99, 164, 180, 195, 250, 295, 298]
...into:
[17, 33, 35, 66, 99, 133, 164, 180, 192, 195, 250, 252, 265, 284, 292, 295, 298]
```

Then the two blocks are printed after having been sorted, followed by the block resulting from merging them both. This makes it easy to determine whether the merge function is working properly. This output repeats, with each merge operation being separated into its own "chunk" of output to make it easier to keep track of which step in the operation the program is on. Eventually a final merge results in the whole array being in sorted order.

```
Sorting current block (index 17 to 24): [74, 222, 172, 35, 91, 140, 247, 102]
Sorting current block (index 25 to 32): [129, 256, 134, 158, 225, 243, 210, 129]
Merging these two blocks...
[35, 74, 91, 102, 140, 172, 222, 247]
[129, 129, 134, 158, 210, 225, 243, 256]
...into:
[35, 74, 91, 102, 129, 129, 134, 140, 158, 172, 210, 222, 225, 243, 247, 256]

Merging these two blocks...
[17, 33, 35, 66, 99, 133, 164, 180, 192, 195, 250, 252, 265, 284, 292, 295, 298]
[35, 74, 91, 102, 129, 129, 134, 140, 158, 172, 210, 222, 225, 243, 247, 256]
...into:
[17, 33, 35, 35, 66, 74, 91, 99, 102, 129, 129, 133, 134, 140, 158, 164, 172, 180, 192, 195,
210, 222, 225, 243, 247, 250, 252, 256, 265, 284, 292, 295, 298]
```

5. Quick Sort

The Quick sort debug output is similarly separated into "chunks," each chunk representing the operations of a single recursive call. Unlike the Merge sort output, this output starts at the top and works down. baseLen is again printed at the top for reference. After that, the portion of the array currently being partitioned is printed. For the first "chunk," this is the entire array.

```
baseLen: 11

[133, 265, 292, 33, 66, 284, 252, 192, 35, 99, 180, 295, 298, 195, 250, 164, 17, 74, 222, 172,
35, 91, 140, 247, 102, 129, 256, 134, 158, 225, 243, 210, 129]
pivot (mode 0): 133
Swapping elements at 0 and 32
Swapping elements at 1 and 25
Swapping elements at 2 and 24
Swapping elements at 5 and 21
Swapping elements at 6 and 20
Swapping elements at 7 and 17
Swapping elements at 10 and 16
Partition complete
LOW (index 0 to 10): [129, 129, 102, 33, 66, 91, 35, 74, 35, 99, 17]
HIGH (index 11 to 32): [295, 298, 195, 250, 164, 180, 192, 222, 172, 252, 284, 140, 247, 292,
265, 256, 134, 158, 225, 243, 210, 133]
```

The pivot is then printed, as well as the mode flag which determines how the pivot is chosen. Then the output lists all the swaps that are made while partitioning the block. Then both partitions are printed, along with their start and end indices. This makes it easy to compare their values with the pivot to ensure the block is being partitioned correctly. Any recursive call on a block of length baseLen or less results in only a single line of output saying which block is being sorted. This output pattern is repeated until all blocks have been partitioned and sorted, resulting in a fully sorted array.

```
[210, 195, 250, 164, 180, 192, 222, 172, 252, 284, 140, 247, 292, 265, 256, 134, 158, 225,
243]
pivot (mode 0): 210
Swapping elements at 12 and 28
Swapping elements at 14 and 27
Swapping elements at 18 and 22
Partition complete
LOW (index 12 to 19): [158, 195, 134, 164, 180, 192, 140, 172]
HIGH (index 20 to 30): [252, 284, 222, 247, 292, 265, 256, 250, 210, 225, 243]

Sorting block from 12 to 19

Sorting block from 20 to 30

Sorting block from 31 to 32
```