CanvasTA Assignment Submit
CS1131 Fall 2020 - Combined Section
Program: Implementing Sort

Caleb Jacobs
cajacobs@mtu.edu

Submit Summary

- Compilation succeeded!
- You passed all the instructor tests.
- I found some style issues in your code.

Compilation Results

Compilation: Succeeded with 0 Errors and 0 Warnings

Instructor Unit Tests

Success! All tests passed.

- **Number of tests: 264**
- **Number of passed: 264**
- **Number of failed: 0**
- **Number of warnings: 0**

Style Critique

File:"Sort.java" has 3 style issues.

```
1   import java.util.ArrayList;
2
3   /**
4    * Program: Implementing Sort
5    * 1. Implement Insertion Sort.
6    * 2. Implement Shell Sort.
7    * 3. Develop your own efficient sorting algorithm.
8    * 4. Analyze the algorithmic efficiency of the above sorts.
9    *
10   * Course: CS1131
11   * Lab Section: L03
12   * @author Caleb Jacobs
13   */
14  public class Sort implements SortInterface {
15
16      /**
17       * Implementation of the Insertion Sort algorithm
18       * Sort data in the range lowindex to highindex, inclusive.
19       * Sort in ascending order unless reversed is true.
20       *
21       * @param list      - ArrayList containing data to be
sorted.
22       * @param lowindex  - lower index of elements to be sorted
23       * @param highindex - upper index of elements to be sorted
24       * @param reversed  - if true, sort in descending order,
otherwise sort in ascending order.
25       *                  <p>
26       *                  Growth Function: T(n) = n^2 - n + cn -
c = O(n^2)
27       *                  Big-Oh: O(n^2)
```

Style Critique
Status: **Warning**
Line: 27 **Critique:** Don't cram operators. Put a space on both sides.

```
28       */
29      @Override
```

```
30      public void insertionsort(ArrayList<Integer> list, int
    lowindex, int highindex, boolean reversed) {
31          for (int i = lowindex + 1; i <= highindex; i++) {
32              int tmp = list.get(i);
33              int j = i;
34              while (j > lowindex && list.get(j - 1) > tmp) {
35                  list.set(j, list.get(j - 1));
36                  j--;
37              }
38              list.set(j, tmp);
39          }
40
41          // Reverse list if required
42          if (reversed) {
43              reverse(list, lowindex, highindex);
44          }
45      }
46
47      /**
48       * Implementation of the Shell Sort algorithm
49       * Sort data in the range lowindex to highindex, inclusive.
50       * Sort in ascending order unless reversed is true.
51       *
52       * @param list      - ArrayList containing data to be
    sorted.
53       * @param lowindex  - lower index of elements to be sorted
54       * @param highindex - upper index of elements to be sorted
55       * @param reversed  - if true, sort in descending order,
    otherwise sort in ascending order.
56       *                  <p>
57       *                  Growth Function: T(n) = O(n^2)
58       *                  Big-Oh: O(n^2)
```

Style Critique
Status: **Warning**
Line: 58 **Critique:** Don't cram operators.

```
59       */
```

```
60      @Override
61      public void shellsort(ArrayList<Integer> list, int lowindex, int highindex, boolean reversed) {
62          int size = highindex - lowindex;
63
64          // Compute initial gap
65          int gap = 0;
66          while (gap < (size + 1)/2) {
67              gap = 2*(gap + 1) - 1;
68          }
69
70          while (gap > 0) {
71              for (int k = 0; k < gap; k++) {
72                  for (int i = lowindex + k + gap; i <= highindex; i += gap) {
73                      int tmp = list.get(i);
74                      int j = i;
75                      for (; j >= lowindex + k + gap; j -= gap) {
76                          if (list.get(j - gap) > tmp) {
77                              list.set(j, list.get(j - gap));
78                          } else {
79                              break;
80                          }
81                      }
82
83                      if (j != i) {
84                          list.set(j, tmp);
85                      }
86                  }
87              }
88
89              gap = (gap + 1)/2 - 1;
90          }
91
92          // Reverse list if required
93          if (reversed) {
```

```
94              reverse(list, lowindex, highindex);
95          }
96      }
97
98      /**
99       * Custom implementation of an original, efficient Sort algorithm
100      * Sort data in the range lowindex to highindex, inclusive.
101      * Sort in ascending order unless reversed is true.
102      *
103      * @param list      - ArrayList containing data to be sorted.
104      * @param lowindex  - lower index of elements to be sorted
105      * @param highindex - upper index of elements to be sorted
106      * @param reversed  - if true, sort in descending order, otherwise sort in ascending order.
107      *                  <p>
108      *                  Growth Function: T(n) = O(n log(n)) + O(n/10) = O(n log(n) + n/10) = O(n log(n))
```

Style Critique
Status: **Warning**
Line: 108 **Critique:** Don't cram operators.

```
109     *                  Big-Oh: O(n log(n))
110     */
111     @Override
112     public void mysort(ArrayList<Integer> list, int lowindex, int highindex, boolean reversed) {
113         modifiedQuicksort(list, lowindex, highindex);
114
115         // Reverse list if required
116         if (reversed) {
117             reverse(list, lowindex, highindex);
118         }
119     }
120
121     /**
```

```
122         * Hybrid quicksort/insertionsort method that uses quicksort when the unsorted array is larger
123         * than 10 elements and then switches to insertionsort to finish the sorting of each subarray.
124         * @param list - list to be sorted.
125         * @param low - low bound on array indices to be sorted.
126         * @param high - upper bound on array indices to be sorted.
127         */
128        public void modifiedQuicksort(ArrayList<Integer> list, int low, int high) {
129            // Use insertion sort if sorting space is less than 10
130            if (high - low < 10) {
131                insertionsort(list, low, high, false);
132            } else if (high > low) {
133                int pivotIdx = partition(list, low, high);
134                modifiedQuicksort(list, low, pivotIdx - 1);
135                modifiedQuicksort(list, pivotIdx + 1, high);
136            }
137        }
138
139        /**
140         * Array partitioning method for quicksort. Splits ArrayList into two partitions about a pivot.
141         * @param list - list to be partitioned/pivoted.
142         * @param first - lower bound on indices of array to be partitioned.
143         * @param last - upper bound on indices of array to be partitioned.
144         * @return - partition pivot index.
145         */
146        public int partition(ArrayList<Integer> list, int first, int last) {
147            int pivot = list.get(first);
148            int low = first + 1;
149            int high = last;
150
151            while (high > low) {
```

```
152                 while (low <= high && list.get(low) <= pivot) {
153                     low++;
154                 }
155 
156                 while (low <= high && list.get(high) > pivot) {
157                     high--;
158                 }
159 
160                 if (high > low) {
161                     int tmp = list.get(high);
162                     list.set(high, list.get(low));
163                     list.set(low, tmp);
164                 }
165             }
166 
167             while (high > first && list.get(high) >= pivot) {
168                 high--;
169             }
170 
171             if (pivot > list.get(high)) {
172                 list.set(first, list.get(high));
173                 list.set(high, pivot);
174                 return high;
175             } else {
176                 return first;
177             }
178         }
179 
180         /**
181          * Reverse the order of elements in a subarray.
182          *
183          * @param list - ArrayList to be reversed.
184          * @param low - lower index of subarray.
185          * @param high - upper index of subarray.
186          */
```

```
187       public void reverse(ArrayList<Integer> list, int low, int
    high) {
188           for (int i = 0; i <= (high - low) / 2; i++) {
189               int tmp = list.get(high - i);
190               list.set(high - i, list.get(low + i));
191               list.set(low + i, tmp);
192           }
193       }
194   }
```