Project 1 Summary
Michael Tan

Note: SortingFactory, SortingAlgorithm, and Practice05Test used in this project are modified versions of classes originally created by David Guy Brizan.

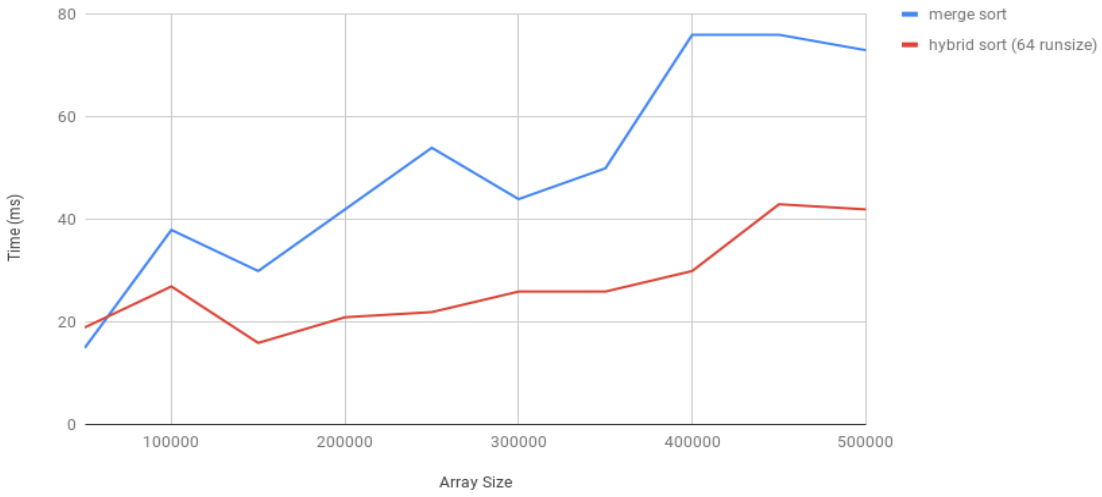
My algorithm is essentially optimized Merge Sort. It's a hybrid between Merge Sort and Insertion Sort. I tried many different approaches to this assignment but this remained the most efficient algorithm I could come up with (my ArrayList idea was much slower than Merge Sort.) I'd also like to point out that this algorithm doesn't exactly follow step 1 of the assignment but instead combines step 1 and 2 together for added efficiency. Instead of only utilizing natural runs that satisfy the run size length, the algorithm also utilizes smaller presorted sections by combining them together along with unsorted sections.

The algorithm operates by recursively splitting the array into sub-arrays until they are less than or equal to the run size (64 for best results.) Those sub-arrays are then sorted using Insertion Sort. I picked this in-place sorting algorithm because it operates faster than Selection Sort on small arrays with a best case of $O(n)$ on presorted/naturally occurring runs (while Selection Sort has $O(n^2)$), and its looping mechanism was quite similar to how I was detecting natural runs originally. Moreover, Insertion Sort beats out Bubble Sort due to early termination when no swaps are made.

There are multiple benefits of dividing the array into sub-array sections:

- It reduces the impact of Insertion Sort's worst case runtime
- It reduces the amount of merges required
- A good run size will create sections of the same length for faster merging (the merging of arrays of significantly different lengths is slow)

Merge Sort Running Time vs Hybrid Sort



[Google Sheets Link]

On average, the hybrid-sorting algorithm consistently performs better than merge sort alone. Furthermore, the benefit of being a hybrid/optimized merge sort algorithm is that in its worse case, it will behave like merge sort *O(nlogn).*

Picking a Good Run Size

An optimal run size requires a balance between being big enough that it reduces the merges used but not being too big that Insertion Sort's worst-case runtime becomes an issue. After a comparison of the running times for different run sizes, 64 seemed to be the best fit. A graph of the different running times is included below.

Example of Hybrid Sort Algorithm Operation

```
Array Length:50
Presorted Array:[75, 26, 84, 61, 52, 6, 63, 36, 97, 47, 31, 46,
 20, 42, 6, 68, 74, 67, 29, 19, 50, 77, 34, 79, 75, 86, 10, 70,
 81, 13, 81, 21, 95, 54, 26, 80, 25, 97, 96, 3, 54, 59, 89, 93,
 13, 80, 90, 22, 26, 24]
------------------------------------------------------------
Called InsertionSort on:[75, 26, 84, 61, 52, 6, 63, 36, 97, 47,
 31, 46, 20]
Called InsertionSort on:[42, 6, 68, 74, 67, 29, 19, 50, 77, 34,
 79, 75]
Merging:[6, 20, 26, 31, 36, 46, 47, 52, 61, 63, 75, 84, 97]
With:[6, 19, 29, 34, 42, 50, 67, 68, 74, 75, 77, 79]
------------------------------------------------------------
Called InsertionSort on:[86, 10, 70, 81, 13, 81, 21, 95, 54, 26
, 80, 25, 97]
Called InsertionSort on:[96, 3, 54, 59, 89, 93, 13, 80, 90, 22,
 26, 24]
Merging:[10, 13, 21, 25, 26, 54, 70, 80, 81, 81, 86, 95, 97]
With:[3, 13, 22, 24, 26, 54, 59, 80, 89, 90, 93, 96]
------------------------------------------------------------
Merging:[6, 6, 19, 20, 26, 29, 31, 34, 36, 42, 46, 47, 50, 52,
61, 63, 67, 68, 74, 75, 75, 77, 79, 84, 97]
With:[3, 10, 13, 13, 21, 22, 24, 25, 26, 26, 54, 54, 59, 70, 80
, 80, 81, 81, 86, 89, 90, 93, 95, 96, 97]
------------------------------------------------------------
MergeInsert Time:2 ms
Sorted Array:[3, 6, 6, 10, 13, 13, 19, 20, 21, 22, 24, 25, 26,
26, 26, 29, 31, 34, 36, 42, 46, 47, 50, 52, 54, 54, 59, 61, 63,
 67, 68, 70, 74, 75, 75, 77, 79, 80, 80, 81, 81, 84, 86, 89, 90
, 93, 95, 96, 97, 97]
```

Note that merging happens in the fashion of (run 1 + run 2) + (run3 + run 4)